

UNIVERSITÉ DE LILLE

Doctoral School **ED MADIS-631**

University Department **CRISAL**

Thesis defended by **Thomas FIRMIN**

Defended on **January 21, 2025**

In order to become Doctor from Université de Lille

Academic Field **Computer science**

Speciality **Machine Learning**

Parallel hyperparameter optimization of spiking neural networks

Thesis supervised by El-Ghazali TALBI Supervisor
Pierre BOULET Co-Supervisor

Committee members

<i>Referees</i>	Timothée MASQUELIER	Senior Researcher at Université de Toulouse	
	Amir NAKIB	Professor at Université Paris Est Créteil	
<i>Examiners</i>	Philippe PREUX	Professor at Université de Lille	Committee President
	Marina REYBOZ	HDR Junior Researcher at CEA	
<i>Supervisors</i>	El-Ghazali TALBI	Professor at Université de Lille	
	Pierre BOULET	Professor at Université de Lille	

COLOPHON

Doctoral dissertation entitled Parallel hyperparameter optimization of spiking neural networks, written by **Thomas FIRMIN**, completed on February 26, 2025, typeset with the document preparation system **L^AT_EX** and the **yathesis** class dedicated to theses prepared in France.

UNIVERSITÉ DE LILLE

Doctoral School **ED MADIS-631**

University Department **CRISAL**

Thesis defended by **Thomas FIRMIN**

Defended on **January 21, 2025**

In order to become Doctor from Université de Lille

Academic Field **Computer science**

Speciality **Machine Learning**

Parallel hyperparameter optimization of spiking neural networks

Thesis supervised by El-Ghazali TALBI Supervisor
Pierre BOULET Co-Supervisor

Committee members

<i>Referees</i>	Timothée MASQUELIER	Senior Researcher at Université de Toulouse	
	Amir NAKIB	Professor at Université Paris Est Créteil	
<i>Examiners</i>	Philippe PREUX	Professor at Université de Lille	Committee President
	Marina REYBOZ	HDR Junior Researcher at CEA	
<i>Supervisors</i>	El-Ghazali TALBI	Professor at Université de Lille	
	Pierre BOULET	Professor at Université de Lille	

UNIVERSITÉ DE LILLE

École doctorale **ED MADIS-631**

Unité de recherche **CRISAL**

Thèse présentée par **Thomas FIRMIN**

Soutenue le **21 janvier 2025**

En vue de l'obtention du grade de docteur de l'Université de Lille

Discipline **Informatique**

Spécialité **Apprentissage machine**

Optimisation parallèle des hyperparamètres des réseaux impulsionnels

Thèse dirigée par El-Ghazali TALBI directeur
Pierre BOULET co-directeur

Composition du jury

<i>Rapporteurs</i>	Timothée MASQUELIER	directeur de recherche à l'Université de Toulouse	
	Amir NAKIB	professeur à l'Université Paris Est Créteil	
<i>Examineurs</i>	Philippe PREUX	professeur à l'Université de Lille	président du jury
	Marina REYBOZ	chargé de recherche HDR au CEA	
<i>Directeurs de thèse</i>	El-Ghazali TALBI	professeur à l'Université de Lille	
	Pierre BOULET	professeur à l'Université de Lille	

The Université de Lille neither endorse nor censure authors' opinions expressed in the theses: these opinions must be considered to be those of their authors.

Keywords: spiking neural networks, hyperparameter optimization, global optimization, parallel computing, decomposition-based optimization, bayesian optimization

Mots clés : réseaux de neurones à impulsions, optimisation des hyperparamètres, optimisation globale, calculs parallèles, optimisation par décomposition, optimisation bayésienne

This thesis has been prepared at the following research units.

CRIS_tAL

UMR 9189
Université de Lille - Campus scientifique
Bâtiment ESPRIT
Avenue Henri Poincaré
59655 Villeneuve d'Ascq
France

Web Site <https://www.cristal.univ-lille.fr/>



IRCICA

USR 3380
CAMPUS Haute-Borne CNRS IRCICA-IRI-RMN
Parc Scientifique de la Haute Borne
50 Avenue Halley
BP 70478
59658 Villeneuve d'Ascq
France

Web Site <https://ircica.univ-lille.fr/>



*À mes grand-mères.
À ma famille.*

PARALLEL HYPERPARAMETER OPTIMIZATION OF SPIKING NEURAL NETWORKS**Abstract**

Artificial Neural Networks (ANNs) are a machine learning technique that has become indispensable. By learning from data, ANNs make it possible to solve certain complex cognitive tasks. Over the last three decades, ANNs have seen numerous major advances. These advances have enabled the development of image recognition, large language models, or text-to-image conversion. Undeniably, ANNs have become an invaluable tool for many applications, and this growing interest led in 2020 to the boom of generative models. However, several new barriers could put the brakes on the interest in these models. The first brake is the end of Moore's Law, due to the physical limits reached by transistors. But also, while research has long focused on the predictive performances of ANNs, other aspects have been neglected. These include energy efficiency, robustness, security, interpretability, transparency and so on^a. This is why we need to go beyond von Neumann architectures for reducing the energy footprint, and the neuromorphic approach is a serious breakthrough candidate through biomimicry of the human brain via Spiking Neural Networks (SNNs).

Unfortunately, SNNs are currently struggling to outperform conventional methods. As they are more recent and therefore less studied, a better approach to their design could make it possible to combine performance and low-energy cost. That is why the automatic design of SNNs is studied within this thesis, with a focus on HyperParameter Optimization (HPO). The aim is to improve the HPO algorithms and to better understand the behavior of SNNs regarding their hyperparameters.

Keywords: spiking neural networks, hyperparameter optimization, global optimization, parallel computing, decomposition-based optimization, bayesian optimization

OPTIMISATION PARALLÈLE DES HYPERPARAMÈTRES DES RÉSEAUX IMPULSIONNELS**Résumé**

Les Réseaux de Neurones Artificiels (RNAs) sont une technique d'apprentissage machine devenue aujourd'hui incontournable, permettant de résoudre certaines tâches cognitives complexes par un apprentissage automatique. Depuis ces trois dernières décennies, les RNAs ont connu de nombreuses avancées majeures. Ces avancées ont permis le développement de la reconnaissance d'images, des modèles de langage géants ou de la conversion texte-image. Indéniablement, les RNAs sont devenus un outil précieux ayant mené, depuis 2020, au boom des modèles générationnels. Cependant, certaines barrières pourraient freiner l'intérêt pour ces modèles. Notamment, la fin de la loi de Moore, due aux limites physiques atteintes par les transistors. Mais, tandis que la recherche s'est longtemps concentrée sur les performances prédictives des RNAs, d'autres aspects ont été négligés. C'est le cas de l'efficacité énergétique, mais également de la robustesse, de la sécurité, de l'interprétabilité, de la transparence, etc^a. Il faut donc aller au-delà des architectures de von Neumann afin de réduire l'empreinte énergétique, et l'approche neuromorphique est un candidat de rupture sérieux utilisant le biomimétisme du cerveau via des Réseaux de Neurones à Impulsions (RNIs).

Toutefois, les RNIs peinent à surpasser les performances des RNAs. Les RNIs étant plus récents, et donc moins étudiés, une meilleure approche de leur conception pourrait permettre d'allier performances et faible coût énergétique. C'est pourquoi la conception automatique des RNIs est étudiée dans cette thèse. L'intérêt est notamment porté sur l'Optimisation des HyperParamètres (OHP). Ainsi, nous étudions l'impact de l'OHP sur les RNIs, et l'impact des RNIs sur l'OHP. Le but étant d'améliorer les algorithmes utilisés et de mieux comprendre le comportement des RNIs au regard de leurs hyperparamètres.

^a<https://futureoflife.org/open-letter/pause-giant-ai-experiments/>

Mots clés : réseaux de neurones à impulsions, optimisation des hyperparamètres, optimisation globale, calculs parallèles, optimisation par décomposition, optimisation bayésienne

CRIStAL

UMR 9189 – Université de Lille - Campus scientifique – Bâtiment ESPRIT – Avenue Henri Poincaré – 59655 Villeneuve d'Ascq – France

Remerciements

La réussite d'une thèse tient moins à l'aspect scientifique qu'à l'entourage et à l'accompagnement du doctorant. Ces quelques lignes expriment humblement ma gratitude envers les personnes qui, à leur manière, ont pu rendre ce travail possible.

Je tiens à saluer la contribution de mon directeur, le Pr. El-Ghazali Talbi, pour m'avoir fait confiance dès le départ. Je reconnais votre accompagnement et votre investissement. Je vous suis gré de m'avoir offert cette opportunité unique de m'épanouir au sein de cet étonnant environnement aussi stimulant qu'exigeant. Cette façon unique de poser des questions avec franchise et d'encourager la remise en question permanente a été une source constante de stimulation. Votre regard aiguisé sur la recherche et son approche méthodologique si singulière ont su nourrir ma réflexion et me confronter à des perspectives inattendues. En somme, ces trois années ont été une aventure intellectuelle et humaine d'une richesse insoupçonnée. *"Trop de rigueur"*.

Je remercie sincèrement mon codirecteur, le Pr. Pierre Boulet, pour votre bienveillance, votre écoute compatissante, et pour la grande liberté que vous m'avez laissée. Merci pour votre confiance en m'accordant d'importantes responsabilités, pour m'avoir petit-à-petit associé dans de grands projets et pour m'avoir intégré dans la communauté neuromorphique française.

Je suis reconnaissant envers Timothée Masquelier et le Pr. Amir Nakib pour avoir accepté de faire partie de mon jury et pour le temps qu'ils ont accordé à la relecture de ce manuscrit. Je les remercie pour leurs remarques pertinentes, ainsi que leurs commentaires élogieux.

Je remercie mes collègues qui m'ont accompagnés, conseillés et écoutés durant ces trois années. Merci Aurélie, Soukaina, Mazdak, et Hammouda pour avoir partagé mon bureau, pour les discussions tant scientifiques que futiles ayant allégé et soulagé ces longues journées de travail. Merci Philippe d'être venu m'accompagner durant ma soutenance. Merci à Julie Keisler, avec qui j'ai pu partager nos difficultés communes durant ces 3 ans. J'ai pu découvrir avec toi le voyage en train de nuit jusqu'en Sicile et l'Etna ! Merci à mes collègues de l'équipe BONUS, qui ont effectué un bout de chemin avec moi durant cette thèse. Je remercie notamment Nouredine, et les restaurant les vendredi midi après de longues semaines. Merci Jan, Guillaume, Jérôme, et à l'ensemble des stagiaires et visiteurs avec qui j'ai eu l'occasion quelques semaines de travail.

Enfin, lorsque les problèmes se montraient tenaces, que les efforts décevaient mes espoirs et que le doute m'envahissait, j'ai toujours pu compter sur le soutien indéfectible de mes proches.

Je suis donc infiniment reconnaissant envers ma famille, mes parents Frédérique et Christian, ma sur Adèle, ma marraine Valérie et mon parrain Philippe. Merci de m'avoir soutenu durant ces trois années, dans les moments les plus durs comme dans les plus joyeux. Merci pour votre écoute, votre empathie, votre réconfort précieux, et vos petits gestes qui à mes yeux ont une valeur inestimable. Vous avez rendu ce manuscrit possible.

À mes amis, qu'ils se reconnaissent ici, je tiens à exprimer toute ma gratitude. Vous avez été là dans les moments de doute, pour célébrer les petites victoires, ou simplement pour m'écouter. Merci à chacun de vous pour ce soutien unique et irremplaçable. Merci à ceux qui n'ont pas hésité à venir de loin pour me soutenir et m'encourager durant ma soutenance, Antoine, Hannah et Hugo, ce geste me va droit au cœur. Merci à Théo et Émile qui ont su, de façon indéfectible, alléger les moments difficiles et offrir des instants de répit au cœur de cette période exigeante. Merci à tous mes amis qui sont venus m'encourager durant cette soutenance éprouvante ! Merci Vivien, Thomas, Matthieu, Cougan, et Hyppolite ! Merci à tous mes amis qui m'ont encouragé pendant et après la thèse.

À toutes les personnes qui lors de mon parcours m'ont aidé, merci.

Extended thesis abstract

Artificial Neural Networks (ANNs) are a machine learning technique that has become indispensable. By learning from data, ANNs make it possible to solve certain complex cognitive tasks. Over the last three decades, ANNs have seen numerous major advances. These include convolution networks and attention mechanisms. These advances have enabled the development of image recognition, large language models, and text-to-image conversion. Undeniably, ANNs have become an invaluable tool for many applications, such as chemistry with AlphaFold, translation with DeepL, archaeology, healthcare, and recently, in February 2024, video generation with Sora.

In 1943, McCulloch and Pitt's work on the formal neuron allowed Rosenblatt to give birth to the first ANNs known as perceptrons in 1958. Machine learning then went through periods of disinterest, due to theoretical obstacles such as the NP-completeness of the problems tackled, technological issues such as limited computing power, and budgetary constraints. Since the 1990s, we've seen an exponential revival of interest¹ in ANNs thanks to the democratization of graphics processing units (GPUs). This growing interest led to the first artificial intelligence spring in the 2010s, and since 2020 to the boom of generative models. However, several new barriers could put the brakes on the interest in these models. The first is the end of Moore's Law, due to the physical limits reached by transistors. The second is energy consumption. Indeed, while research has long focused largely on the predictive performances of ANNs, other aspects have been neglected. These include energy efficiency, robustness, security, interpretability, transparency, and so on².

This is why we need to go beyond von Neumann architectures, which currently slow down calculation throughput because of the separation of processing and memory units. The neuromorphic approach is a serious breakthrough candidate for reducing the energy footprint of machine learning. Indeed, this mode of calculation is based on Spiking Neural Networks (SNNs), which are closer to the biological brain. The human brain consumes only 20 watts to perform numerous complex cognitive tasks simultaneously. So, the challenge of neuromorphic computing is to considerably reduce the energy consumption of current models through biomimicry. Neuromorphic computing could also enable advances in other cross-disciplinary research fields, such as neuroscience³.

Today, SNNs are struggling to outperform conventional methods. As they are more recent and therefore less studied, a better approach to their design could make it possible to combine performance and low-energy cost. That is why the automatic design of SNNs is studied within this thesis, with a focus on HyperParameter Optimization (HPO). A hyperparameter is a parameter controlling various aspects of the training phase of a SNN, but whose value cannot be determined by training. Thus, we study the impact of HPO on SNNs and the impact of SNNs on HPO. The aim is to improve the HPO algorithms and to better understand the behavior of SNNs regarding their hyperparameters.

In the literature, the HPO is treated in the same way, whether for ANNs or SNNs. However, the "No Free Lunch theorem" specifies that there is no universal algorithm that is significantly efficient for all optimization problems. A consequence of this theorem is that, without prior knowledge of the problem, it is impossible to optimize efficiently. That is, we need a clear definition of the problem before selecting an HPO algorithm. So, while ANNs and SNNs share common properties, SNNs are known for their unique behaviors. In particular, the literature shows that the performances of SNNs are highly sensitive to their architecture and

¹<https://ourworldindata.org/grapher/artificial-intelligence-parameter-count>

²<https://futureoflife.org/open-letter/pause-giant-ai-experiments/>

³<https://www.ebrains.eu/>

hyperparameters. Consequently, blindly applying the same methodologies to both ANNs and SNNs could negatively affect the performances of the HPO algorithm, and hence the best solution obtained.

In this thesis, we propose a state of the art of the HPO methods used on SNNs. Such a survey has never been done for SNNs. Next, we take a closer look at commonly used HPO algorithms, such as genetic algorithms or Bayesian optimization.

The first contribution concerns the generalization of a family of algorithms based on the decomposition of the search space to optimize a continuous function. We call this family “fractal-based decomposition algorithms”. A software package, named Zellij, has been created to generalize these approaches and facilitate their design. Comparative experiments show the advantages and disadvantages in terms of scalability in dimension of these algorithms. In particular, we are studying deterministic algorithms such as DIRECT, SOO, FDA, and many other versions, all implemented with Zellij.

The following contributions concern large-scale and long-run (100 h each) parallel experiments about HPO applied to SNNs, on the Jean Zay supercomputer. We applied and parallelized a large-scale Bayesian approach to improve the resource management within limited budget experiments. We describe a certain type of SNNs known as silent networks. Experimentation demonstrates the negative impact of these silent networks on the performance and convergence of the HPO algorithm. A solution using early stopping combined with constrained optimization allows improving HPO performances. Comparisons between algorithms and various analyses of the optimization and results will be presented.

Finally, based on the obtained results and to considerably speed up HPO, we propose to combine the previous approach with multi-fidelity optimization. We demonstrate empirically a significant decrease in the required budget while maintaining competitive performances.

Résumé étendu de la thèse

Les Réseaux de Neurones Artificiels (RNAs) sont une technique d'apprentissage machine devenue aujourd'hui incontournable. Par un apprentissage à partir de données, les RNAs permettent de résoudre certaines tâches cognitives complexes. Depuis ces trois dernières décennies, les RNAs ont connu de nombreuses avancées majeures. Notamment avec les réseaux de convolution ou les mécanismes d'attention. Ces avancées ont permis le développement de la reconnaissance d'images, des modèles de langage géants ou de la conversion texte-image. Indéniablement, les RNAs sont devenus un outil précieux pour de nombreuses applications. Comme en chimie avec AlphaFold, la traduction avec DeepL, en archéologie, en santé, ou récemment, en février 2024, la génération de vidéo avec Sora.

En 1943, les travaux de McCulloch et Pitt sur le neurone formel faciliteront la naissance des premiers RNAs appelés perceptrons, et décrits pour la première fois par Rosenblatt en 1958. L'apprentissage machine a par la suite connu des périodes de désintérêt dues à des freins, théoriques comme la NP-complétude des problèmes abordés, technologiques comme la limitation de la puissance de calcul, ou encore budgétaires. Depuis les années 1990 et aidé par la démocratisation des processeurs graphiques (GPUs), nous observons un regain d'intérêt exponentiel⁴ pour les RNAs menant au printemps de l'intelligence artificielle dans les années 2010, et depuis 2020 au boom des modèles générationnels. Cependant, de nouvelles barrières pourraient freiner l'intérêt pour ces modèles. Le premier frein est la fin de la loi de Moore due aux limites physiques atteintes par les transistors. Le second frein est la consommation énergétique. En effet, tandis que la recherche s'est longtemps concentrée en grande partie sur les performances prédictives des RNAs, d'autres aspects ont été relégués au second plan. C'est le cas de l'efficacité énergétique, mais également de la robustesse, de la sécurité, de l'interprétabilité, de la transparence, etc⁵.

C'est pourquoi il faut aller au-delà des architectures de von Neumann qui, aujourd'hui, à cause de la séparation des unités de calcul et de mémoire, ralentissent le débit des calculs. Ainsi, l'approche neuromorphique est un candidat de rupture sérieux afin de réduire l'empreinte énergétique de l'apprentissage machine. En effet, ce mode de calcul repose sur les Réseaux de Neurones à Impulsions (RNIs), plus fidèles au cerveau biologique. Le cerveau humain consomme uniquement 20 watts pour effectuer simultanément de nombreuses tâches cognitives complexes. Ainsi, par biomimétisme, le pari de l'informatique neuromorphique est de réduire considérablement la consommation énergétique des modèles actuels. Le neuromorphique pourrait aussi permettre des avancées dans d'autres domaines de recherche transverses comme les neurosciences⁶.

Aujourd'hui, les RNIs peinent à surpasser les performances des méthodes classiques. Ceux-ci étant plus récents, et donc moins étudiés, une meilleure approche de leur conception pourrait permettre d'allier performances et faible coût énergétique. C'est pourquoi la conception automatique des RNIs est étudiée dans cette thèse. L'intérêt est notamment porté sur l'Optimisation des HyperParamètres (OHP). Un hyperparamètre est un paramètre contrôlant divers aspects de l'apprentissage des RNIs, mais dont la valeur ne peut pas être déterminée par l'apprentissage. Ainsi, nous étudions l'impact de l'OHP sur les RNIs et l'impact des RNIs sur l'OHP. Le but étant d'améliorer les algorithmes utilisés et de mieux comprendre le comportement des RNIs au regard de leurs hyperparamètres.

Dans la littérature, que ce soit pour les RNAs ou RNIs, l'OHP est traité de la même manière. Or, le "No Free Lunch theorem" spécifie qu'il n'existe pas d'algorithme universel

⁴<https://ourworldindata.org/grapher/artificial-intelligence-parameter-count>

⁵<https://futureoflife.org/open-letter/pause-giant-ai-experiments/>

⁶<https://www.ebrains.eu/>

significativement efficace pour tous les problèmes d'optimisation. Une conséquence de ce théorème est que, sans connaissances préalables du problème, il est impossible d'optimiser efficacement. C'est-à-dire que l'on ne peut pas choisir un algorithme d'optimisation avant d'avoir réellement défini le problème. Ainsi, bien que RNAs et RNIs partagent des propriétés communes, les RNIs sont connus pour leurs propriétés uniques. La littérature fait notamment état d'une extrême sensibilité des performances des RNIs par rapport à leur architecture et à leurs hyperparamètres. Par conséquent, appliquer aveuglément les mêmes méthodologies aux RNAs et RNIs pourrait nuire aux performances de l'algorithme d'optimisation, et donc de la meilleure solution obtenue.

Dans cette thèse, nous décrirons dans un premier temps, sous la forme d'un état de l'art, les méthodes d'OHP utilisées sur les RNIs. Un tel exercice n'ayant jamais été fait pour les RNIs. Ensuite, nous décrirons plus précisément les algorithmes d'OHP couramment utilisés, comme les algorithmes génétiques ou l'optimisation Bayésienne.

La première contribution concerne la généralisation d'une famille d'algorithmes se fondant sur la décomposition de l'espace de recherche afin d'optimiser une fonction continue. Nous appelons cette famille "Fractal-based decomposition algorithms". Un logiciel, nommé Zellij, a été créé dans le but de généraliser ces approches et de faciliter leur conception. Des expériences comparatives permettent de montrer les avantages et les inconvénients de ces algorithmes lors du passage à l'échelle en hautes dimensions. Nous étudions notamment des algorithmes déterministes comme DIRECT, SOO, FDA et de nombreuses autres versions, toutes implémentées par Zellij.

Les contributions ultérieures portent sur de longues expériences (100 h) d'OHP parallèle appliqué aux RNIs, à large échelle et effectuées sur le superordinateur Jean Zay. Nous utilisons et parallélisons une approche Bayésienne à large échelle permettant une meilleure gestion des ressources dans le cadre d'un budget d'optimisation limité. De plus, nous décrivons un certain type de RNIs, que l'on appelle "réseaux silencieux". L'expérimentation permet de démontrer l'effet néfaste de ces réseaux silencieux sur les performances et la convergence de l'algorithme d'OHP. Une solution utilisant un arrêt prématuré de l'entraînement, combiné à de l'optimisation contrainte, permet d'améliorer les performances de l'OHP. Diverses comparaisons entre algorithmes et des analyses de l'optimisation et des résultats seront présentées.

Finalement, fort des résultats obtenus, nous proposons de combiner l'approche précédente avec de la multi-fidélité afin d'accélérer considérablement l'OHP. Nous montrons empiriquement une réduction conséquente du budget nécessaire à l'optimisation tout en conservant des performances compétitives.

Contents

Abstract	xv
Remerciements	xvii
Contents	xxiii
Glossary	xxv
List of symbols	xxix
List of Tables	xxxi
List of Figures	xxxiii
1 Introduction to Spiking Neural Networks	1
2 Automated machine learning and spiking neural networks	23
3 Algorithms for Hyperparameter Optimization	53
4 Partition-based global optimization	81
5 Silent networks: a vicious trap for Hyperparameter Optimization	135
6 Accelerating Hyperparameter Optimization with Multi-Fidelity	163
Main conclusion	185
Bibliography	191
A Trust Region and Scalable Constrained Bayesian Optimization	217
B Sensitivity analysis on the stopping criterion	223
C Analysis of the asynchronous parallelization.	225
D Search spaces of chapter 5	231
E Search spaces of chapter 6	239
F Description of all hyperparameters	253
G Computer programs	255
Contents	267

Glossary

- AI** Artificial Intelligence. xxix, 1, 2, 6, 7, 13, 20, 55, 185
- ANN** Artificial Neural Network. 2, 3, 5–7, 11, 12, 14–19, 21, 23, 24, 26–28, 30, 31, 33, 41, 44, 48, 49, 52, 55, 56, 59–61, 74, 76, 138, 143, 185, 187
- ASIC** Application-Specific Integrated Circuit. 2
- AutoML** Automated Machine Learning. xxxi, 23–28, 30, 32–34, 164, 180, 186, 187
- BFS** Best First Search. 105, 106, 115, 117
- BO** Bayesian Optimization. xxix, xxxi, 32, 33, 45, 46, 48, 49, 54–56, 61–63, 67, 68, 70, 72, 73, 75, 81, 82, 134, 136, 141, 163–165, 179, 180, 188, 192
- BOHB** Bayesian Optimization HyperBand. 48, 76–78, 82, 164
- BP** BackPropagation. 9, 10, 17, 18, 20, 37, 139, 141, 143, 162, 164
- BrFS** Breadth First Search. 105
- BS** Beam Search. 106, 115
- CASCBO** Cost Aware Scalable Constrained Bayesian Optimization. xxxi, 167–169, 172–180, 187
- CDTTB** Centered Distance To The Best. 115, 116
- CMA-ES** Covariance Matrix Adaptation Evolution Strategy. 60
- CNN** Convolutional Neural Network. 6, 12, 13, 28, 41, 42, 48, 49
- COCO** COmparing Continuous Optimizers. 32, 60, 83, 116, 126, 186
- CPU** Central Processing Unit. 1–4, 49, 77, 169, 189
- CSNN** Convolutional Spiking Neural Network. 20, 41, 46, 48, 49, 168, 170
- DAG** Directed Acyclic Graph. 31
- DE** Differential Evolution. 46
- DFS** Depth First Search. 105, 106, 110
- DIRECT** DIviding RECTangle. 83, 88, 90–94, 96, 106, 107, 113–115, 117, 120, 124, 128, 129, 133, 185, 186, 234
- DIRECT-L** Locally biased DIviding RECTangle. 88, 114–117, 120
- DIRECT-R** DIviding RECTangle Restart. 88, 114, 115, 117

- DIRMIN** DIRECT-based algorithm with a truncated Newton method. 88
- DL** Deep Learning. 24
- DNN** Deep Neural Network. 6, 7, 23, 30
- DOO** Deterministic Optimistic Optimization. 90
- DTB** Divide-The-Best. 87
- DTTB** Distance To The Best. 91, 94, 107, 115, 116
- DVS** Dynamic Vision Sensor. 4, 16, 142
- EA** Evolutionary Algorithm. 28, 31, 33, 45, 49, 54, 57, 59, 60, 84, 134
- EI** Expected Improvement. 70, 76, 81, 82, 167
- EIpu** Expected Improvement per unit. 165, 167
- EONS** Evolutionary Optimization for Neuromorphic Systems. 49
- ERT** Expecting Running Time. 116, 117
- FBD** Fractal-Based Decomposition. xxx, 83–87, 94, 98, 105, 108, 124, 133, 134, 186, 188
- FDA** Fractal Decomposition Algorithm. xxx, 83, 91, 92, 94, 96, 98, 102, 103, 105, 107, 108, 115–117, 120, 124, 128, 129, 133, 186, 229
- FPA** Field-Programmable Analog Array. 4
- FPGA** Field-Programmable Gate Array. 4
- FRACTOP** Fractal Optimization Approach. 83, 91, 94, 106, 108, 115, 133
- GA** Genetic Algorithm. 46, 49, 58–61, 70, 80, 81, 94, 135, 136, 141
- GAN** Generative Adversarial Network. 6
- GP** Gaussian Process. xxix, xxxi, 26, 33, 61–68, 70, 72, 73, 75, 76, 81, 144, 149, 150, 154, 165, 167, 192, 193, 240
- GPU** Graphical Processing Unit. 1, 3, 4, 6, 49, 52, 55, 67, 77, 78, 135, 143–145, 151, 153, 155, 162, 163, 169, 174, 178, 186, 189, 193, 199
- GRU** Gated Recurrent Units. 12
- GS** Grid Search. xxix, 32, 48, 49, 56, 57, 60, 80, 82
- HERA** Hydrogen Epoch of Reionization Array telescope. 49
- HP** HyperParameter. xxvii–xxix, xxxi, 23, 24, 26–39, 41, 42, 44–46, 48, 49, 52, 53, 55–60, 62, 73–77, 82, 135–139, 141–145, 147, 149–154, 161–165, 167–172, 174, 178–180, 185–188, 193, 197, 199, 205, 206, 213, 218–225
- HPO** HyperParameter Optimization. xxix, xxx, 7, 16, 20, 21, 23, 26–30, 32–36, 41, 42, 44, 46, 48, 49, 52–61, 74, 75, 77–82, 135, 136, 138, 140–142, 145, 147, 155, 162–164, 167–169, 172, 174, 178, 180, 185–189, 240

- IF** Integrate and Fire. 16, 17, 57, 143
- ILS** Intensive Local Search. 91, 94, 105, 108, 110, 111, 115, 120, 128, 129
- IoT** Internet-of-Things. 2
- KDE** Kernel Density Estimation. 75–77
- LCB** Lower Confidence Bound. 70
- LHS** Latin Hypercube Sampling. xxx, 56, 60, 124–129, 133, 134, 186, 236
- LIF** Leaky Integrate and Fire. xxix, 17, 34–37, 48, 49, 57, 142, 143, 168, 169, 172, 174, 179, 187
- LO** Lipschitzian Optimization. 87, 88
- LSTM** Long Short Term Memory. 6, 12, 48
- MCTS** Monte Carlo Tree Search. 93
- MIMD** Multiple Instruction Multiple Data. 2, 3
- ML** Machine Learning. xxix, 1, 4–7, 13–15, 17, 20, 21, 23–33, 37, 43, 46, 55, 59, 61, 63, 66, 74, 77, 135, 185
- MNIST** Mixed National Institute of Standards and Technology. xxxi, xxxii, 15, 19, 42, 45, 46, 48, 49, 135, 138, 141, 143–145, 149, 151, 153, 168, 169, 172, 174, 178, 179, 181, 200, 201
- MOGP** Multi-output Gaussian processes. 165
- MOSFET** MetalOxide-Semiconductor field-effect transistor. 1, 3
- MSE** Mean Squared Error. 25, 38, 56
- MSO** Multi-Scale Optimization. 83–85, 87
- NAS** Neural Architecture Search. 23, 26, 31, 32, 59, 188
- NMNIST** Neuromorphic MNIST. xxxi, 48, 168, 169, 172, 174, 178, 179, 182
- NMSO** Naive Multi-scale Search Optimization. 90, 91, 111–115, 117, 120, 186, 232
- PD** Positive-Definite. 63, 64, 67
- PHS** Promising Hypersphere Search. 91, 108, 115
- PI** Probability of Improvement. 70
- PLIF** Parametric Leaky Integrate and Fire. 34, 39, 141–143, 149, 153, 155, 168, 169, 172, 174, 178–180, 187
- POH** Potentially Optimal Rectangle. xxx, 88, 90, 92, 106, 114, 115
- PSO** Particule Swarm Optimization. 57
- RBF** Radial Basis Function. xxix, 26, 27, 70, 71

- RBM** Restricted Boltzmann Machine. 46
- ReLU** Rectified Linear Unit. 8
- RNN** Recurrent Neural Network. 6, 12, 19, 40, 49
- RS** Random Search. xxix, 46, 49, 56, 57, 60, 80, 82, 168, 169, 178–180, 187
- RSNN** Recurrent Spiking Neural Network. 48
- SA** Simulated Annealing. 57–59, 80, 94, 108
- SCBO** Scalable Constrained Bayesian Optimization. xxxi, 141, 143–145, 147, 151, 153, 162, 167, 170, 180, 186, 187, 191–193, 195, 199
- SGD** Stochastic Gradient Descend. 6, 11, 74, 169, 174
- SH** Successive Halving. 74–76, 82
- SHD** Spiking Heildeberg Digits. xxvii, xxxi, 48, 49, 168, 169, 172, 174, 178–180, 183, 187
- SIMD** Single Instruction Multiple Data. 1–3
- SISD** Single Instruction Single Data. 1–3
- SLAYER** Spike Layer Error Reassignment in Time. 37, 38, 43, 48, 139, 141–143, 153, 169, 172, 174, 179, 180, 187
- SMBO** Surrogate Model Based Optimization. 45, 55, 61, 62, 75, 77, 188
- SNN** Spiking Neural Network. xxvii, xxix, xxxi, 3, 4, 7, 14–21, 23, 24, 26, 27, 30–37, 40–44, 46, 48, 49, 52, 53, 55, 56, 58–60, 62, 63, 78, 135, 136, 138–145, 147, 149, 151, 153, 155, 162–165, 168, 169, 172, 174, 177, 178, 180, 185–189, 191, 193, 197
- SOM** Self Organizing Map. 19, 41, 42, 45, 46, 135, 139, 142, 145, 153, 164
- SOO** Simultaneous Optimistic Optimization. 83, 90–94, 102, 107, 111–117, 120, 124, 128, 129, 133, 186, 231
- SRM** Spike Response Model. 37, 45, 142, 169, 172
- SSC** Spiking Speech Commands. 48
- STDP** Spike Timing Dependent Plasticity. xxix, 6, 17–19, 36, 37, 40, 42, 44–46, 135, 139, 141, 143, 147, 153, 186, 187
- Sto-SOO** Stochastic Simultaneous Optimistic Optimization. 90
- SVM** Support Vector Machine. 6, 27, 43, 46, 61, 142, 147
- TPE** Tree-structure Parzen Estimator. 49, 75–77, 82
- TS** Thompson Sampling. 81, 82, 167, 191–193
- TTFS** Time-To-First Spike. 15, 16, 42, 44, 206
- TuRBO** Trust Region Bayesian Optimization. xxxi, 141, 144, 151, 153, 155, 165, 167, 186, 191, 192, 199
- UCB** Upper Confidence Bound. 73, 127
- WTA** Winner Takes All. 19, 40, 41, 43, 147, 150

List of symbols

\mathbb{R} , set of real numbers.

\mathbb{Z} , set of discrete numbers.

\mathbb{N} , set of natural numbers.

Ω , a search space or search space of hyperparameters.

Θ , learnable parameter space of neural networks.

$[a, b]$, continuous set bounded by a and b .

$\llbracket a, b \rrbracket$, discrete set bounded by a and b .

$\{a, b, c\}$, a set containing elements a , b and c .

$\mathcal{D}_{\text{train}}$, training dataset.

$\mathcal{D}_{\text{valid}}$, validation dataset.

$\mathcal{D}_{\text{test}}$, testing dataset.

λ , a set of hyperparameters, i.e. a single solution from Ω .

Λ , a set of multiple solutions, or a set of sets of hyperparameters.

\sqcap , disjoint at the borders.

x^* , a $*$ superscript indicates an optimal variable returned by an optimization algorithm.

\vec{e}_i , unit vector, 1 at index i , 0 otherwise.

\mathbf{a} , a vector.

\mathbf{A} , a matrix.

\mathbf{A}^\top , transposition of matrix \mathbf{A} .

argmin , argument of the minimum.

argmax , argument of the maximum.

\sup , supremum.

\inf , infimum.

H , Heaviside step function.

$\frac{\partial f}{\partial x}$, partial derivative of function f according to variable x .

$\frac{df}{dx}$, total derivative of function f according to variable x .

$\mathbb{P}(A)$, probability of an event A .

$\mathcal{U}(a,b)$, uniform distribution in $[a,b]$.

$\mathcal{N}(\mu, \sigma^2)$, Gaussian distribution of mean μ and variance σ^2 .

$\mathcal{N}(\boldsymbol{\mu}, \Sigma)$, multivariate Gaussian distribution of mean vector $\boldsymbol{\mu}$ and covariance matrix Σ .

$\mathcal{GP}(f; \mu, K)$, a Gaussian process on a function f , with the mean function μ and kernel K .

\triangleq , definition.

\vee , logical or.

\wedge , logical and.

$a \leftarrow b$, assignment. In algorithmic, value b is assigned to variable a .

$|A|$, cardinality of a set A , also written **card**(A).

List of Tables

2.1	Summary of hyperparameter optimization of STDP trained networks.	46
2.2	Summary of hyperparameter optimization of SNNs trained by surrogate gradient.	48
2.3	Summary of hyperparameter optimization of spiking neural networks with various training algorithms.	50
3.1	Some kernels and their parameters.	67
4.1	Example of fractals and their properties	102
4.2	Instantiated algorithms using Zellij	115
5.1	Setup of all 6 experiments	143
5.2	Architecture details of all 6 experiments.	144
5.3	Proportion of silent networks during HPO	148
5.4	Best performances obtained using SCBO and BindsNet	149
5.5	Best performances obtained using SCBO and LAVA-DL	150
5.6	Best performances obtained using SCBO and SpikingJelly	150
5.7	Best performances obtained using TuRBO and LAVA-DL	157
5.8	Best performances obtained using TuRBO and SpikingJelly	158
5.9	Summary of all long run experiments.	161
6.1	Architecture details of SNNs architectures applied to SHD.	169
6.2	Summary of experiments	170
6.3	Results and performances of best solutions from all experiments	172
D.1	HPs of preliminary experiments	232
D.2	S-STDP-MNIST	233
D.3	S-STDP-DVS	234
D.4	S-SLAY-MNIST	235
D.5	S-SLAY-DVS	236
D.6	S-SuGr-MNIST	237
D.7	S-SuGr-DVS	238
E.1	22-C-SLAY-MNIST,46-C-SLAY-MNIST,22-C-SLAY-NMNIST,46-C-SLAY-NMNIST, 46-R-SLAY-MNIST,46-R-SLAY-NMNIST	240
E.2	21-C-SLAY-SHD,42-C-SLAY-SHD and 42-R-SLAY-SHD	241
E.3	15-C-SuGr-MNIST,46-C-SuGr-MNIST,15-C-SuGr-NMNIST,21-C-SuGr-NMNIST,21-R-SuGr-MNIST and 21-R-SuGr-NMNIST	242
E.4	15-C-SuGr-SHD,21-C-SuGr-SHD and 21-R-SuGr-SHD	243
E.5	Optimized HPs for 22-C-SLAY-MNIST and 22-C-SLAY-NMNIST.	244
E.6	Optimized HPs for 46-C-SLAY-MNIST, 46-C-SLAY-NMNIST, 46-R-SLAY-MNIST and 46-R-SLAY-NMNIST	245
E.7	Optimized HPs for 21-C-SLAY-SHD.	246
E.8	Optimized HPs for 42-C-SLAY-SHD and 42-R-SLAY-SHD.	247

E.9	Optimized HPs for 15-C-SuGr-MNIST and 15-C-SuGr-NMNIST.....	248
E.10	Optimized HPs for 21-C-SuGr-MNIST, 21-C-SuGr-NMNIST, 21-R-SuGr-MNIST and 21-R-SuGr-NMNIST	249
E.11	Optimized HPs for 13-C-SuGr-SHD.	250
E.12	Optimized HPs for 21-C-SuGr-SHD and 21-R-SuGr-SHD.	251
F.1	Hyperparameter descriptions	254

List of Figures

1.1	Flynn's information processing taxonomy.	2
1.2	Number of parameters of major AI models through time. Data are from https://ourworldindata.org/grapher/artificial-intelligence-parameter-count	7
1.3	Basics of neural networks.	8
1.4	Graph of an artificial neuron.	8
1.5	Some activation functions of a neural network.	9
1.6	Graph of a spiking neuron.	17
1.7	1D Dielh & Cook self-organizing map. For clarity, the synapses of the middle neurons are not represented. Dotted lines illustrate inhibitory synapses with a fixed negative weight. While solid lines are excitatory synapses.	20
2.1	Types of features	24
2.2	Modeling "natural" probability distribution of data with ML.	25
2.3	Training - Validation - Test splits of a dataset	27
2.4	Nested cross validation	28
2.5	Example of different network architectures	31
2.6	LIF (left) and Adaptive LIF (right).	36
2.7	Impact of HPs on STDP.	37
2.9	Impact of HPs in different surrogate gradient functions.	38
2.10	$\lambda_{\nabla} = 0.1$	40
2.11	$\lambda_{\nabla} = 0.6$	40
2.12	$\lambda_{\nabla} = 0.025$	40
2.13	$\lambda_{\nabla} = 0.1, \beta_{\nabla} = 0.8$	40
2.14	Feed Forward Architecture	43
2.15	Pruning	43
2.16	Neurons dropout	43
2.17	Usual workflow of HPO applied to SNNs	49
3.1	GS with a grid size of 5.	55
3.2	RS with 64 points.	55
3.3	Illustration of Bayesian inference	61
3.4	GP on a noiseless function.	64
3.5	GP on a noisy function.	64
3.6	Isotropic kernels	68
3.7	RBF with different parameters on $f(\lambda) = \sin^3(\lambda) + 0.1\lambda + \varepsilon$. The dashed line is the mean function.	69
3.8	Flowshart of BO	70
3.9	Master-worker parallelization approach.	77
3.10	Illustration of synchronous parallelization. The communication and evaluations between the master and workers can be done asynchronously. The next iteration of the HPO algorithm is computed once all k evaluations are done.	78

3.11 Illustration of asynchronous parallelization. There is no proper iteration of HPO. The algorithm asynchronously receives and generates <i>on-demand</i> solutions.	78
4.2 Example of a tree-like data structure for bisection-based FBD algorithms, applied to $f(x) = -x\sin(x) - x\cos(2x) - \frac{x}{6}$	82
4.4 Examples of Lipschitz continuity (left) and uniform continuity (right). If the curve crosses the red zone, then the function is not Lipschitz or uniformly continuous.	85
4.6 Three iterations of the Shubert algorithm (left), with the resulting rooted tree (right).	87
4.8 A single refinement of a hyper-rectangle in DIRECT. (the best point is represented in dotted green)	87
4.9 Potentially optimal rectangles according to their size (σ) and their center ($f(c)$). Teal-diamond dots are POH.	88
4.11 A single refinement of a hyper-rectangle in SOO.	88
4.13 A single refinement of a hyper-cube in FRACTOP.	89
4.15 Two refinements in FDA.	90
4.16 Workflow of Zellij.	93
4.17 Example of a cover of Ω by a collection $\mathcal{F} = \{A, B, C, D, E\}$. The solid-green part corresponds to overlapping between fractals. Dotted-gray is the part of fractals outside the search space Ω represented by a thick line.	95
4.18 Example of a 2-refinement of depth 4, with a bisection along the longest side.	96
4.19 2-dimensional illustration of covering and overlap applied on a FDA scheme.	97
4.20 2-dimensional illustrations of definition 25. The hatched-red area corresponds to uncovered space resulting from successive decomposition: $C(\Omega, \cup \mathcal{L}_{\mathcal{T}(d)})$. Solid-green corresponds to overlapping: $O(\mathcal{L}_{\mathcal{T}(d)})$	101
4.21 Various examples of fractals in 2 dimensions: (a) hypercubes, (b) bisections, (c) trisections, (d) hyperspheres, (e) dynamic Voronoï, (f) simplices.	103
4.22 Fractal decomposition with hypercubes applied on 2D Styblinski-Tang function with various tree search: (a) Depth First Search, (b) Breadth First Search, (c) Best First Search, (d) Beam Search, (e) Cyclic Best First Search, (f) Epsilon Greedy Search.	105
4.23 Empirical cumulative distribution according to the budget divided by the dimensions for each functions' subclass and dimension 2, 3 and 5.	117
4.24 Empirical cumulative distribution according to the budget divided by the dimensions for each functions' subclass and dimension 10, 20 and 40.	118
4.25 Empirical cumulative distribution according to the budget divided by the dimensions for all functions and dimensions.	119
4.26 Pair Wise Wilcoxon test for each functions' subclass and dimensions 2, 3 and 5. Solid-Grey : Statistically insignificant ($\alpha > 0.05$). Gridded-Green : Better. Dotted-Red : Worse.	120
4.27 Pair Wise Wilcoxon test for each functions' subclass and dimensions 10, 20 and 40. Solid-Grey : Statistically insignificant ($\alpha > 0.05$). Gridded-Green : Better. Dotted-Red : Worse.	121
4.28 Pair Wise Wilcoxon test for all functions and dimensions. Solid-Grey : Statistically insignificant ($\alpha > 0.05$). Gridded-Green : Better. Dotted-Red : Worse.	122
4.29 LHS with a grid size of 5×5	124
4.30 Nested LHS.	124

4.31	Number of minimum calls s to $\pi_{\text{LHS}}(\Omega)$ multiplied by the number g of sampled fractals for each calls, so that the probability that x^* does not belong to sampled fractals is lower than $1/2$; $\mathbb{P}\left(x^* \notin \bigcup_{i=0}^s \pi_{\text{LHS}}(\Omega) \mid g, n\right) \leq 1/2$ with $g \in \{2, 3, 5, 10, 20, 40\}$	126
4.32	Empirical cumulative distribution according to the budget divided by the dimensions for each functions' subclass and dimension 2, 3 and 5.	129
4.33	Empirical cumulative distribution according to the budget divided by the dimensions for each functions' subclass and dimension 10, 20 and 40.	130
4.34	Empirical cumulative distribution according to the budget divided by the dimensions for all functions and dimensions.	131
4.35	Pair Wise Wilcoxon test for all functions and dimensions. Solid-Grey: Statistically insignificant ($\alpha > 0.05$). Gridded-Green: Better. Dotted-Red: Worse.	131
5.1	Results of preliminary experiments.	137
5.2	Class wise spikes count distribution	140
5.3	Proposed mono- or multi-objectives grey-box HPO of SNNs.	142
5.4	Computation start date to end date compared to accuracy of each SNN.	147
5.5	Single HP sensitivity in S-STDP-MNIST	152
5.6	Single HP sensitivity in S-SLAY-MNIST	153
5.7	Single HP sensitivity in S-SLAY-DVS	153
5.8	Single HP sensitivity in S-SuGr-MNIST	154
5.9	Single HP sensitivity in S-SuGr-DVS	154
5.10	Lengthscales from the GPs	155
5.11	Computation start date to end date compared to accuracy of each SNN optimized by TuRBO.	158
5.12	Allocation of the budget during the optimization on MNIST using LAVA-DL	159
5.13	Allocation of the budget during the optimization on DvsGesture using LAVA-DL	159
5.14	Allocation of the budget during the optimization on MNIST using SpikingJelly	160
5.15	Allocation of the budget during the optimization on DvsGesture using SpikingJelly	160
6.1	Multi-fidelity AutoML on the training dataset. The optimization of λ^i HPs is conjoined to the one of π_{train}	165
6.2	t-SNE applied on all evaluated solutions returned by CASCBO. The lighter or yellower the point, the higher the validation accuracy.	174
6.3	Accuracy according to π_{train} for all CASCBO experiments.	175
6.4	Accuracy according to T for all CASCBO experiments.	176
6.5	Computation start date to end date compared to accuracy of each SNN optimized by CASCBO.	178
6.6	Comparison between results obtained on MNIST between experiments of chapter 5 and 6.	179
6.7	Learning curves summary of the best solutions from two experiments.	180
6.8	Allocation of the budget during multi-fidelity experiments on MNIST.	181
6.9	Allocation of the budget during multi-fidelity experiments on NMNIST.	182
6.10	Allocation of the budget during multi-fidelity experiments on SHD.	183
A.1	Flowshart of BO and TuRBO. The dashed lines illustrate the alternative for TuRBO.	218
A.2	Workflow of SCBO.	221

B.1	Sensitivity analysis of S-SLAY-MNIST. Upper left: number of silent SNNs according to α and β . Upper right: number of evaluated SNNs according to α and β . Lower left: proportion of sampled silent networks according to α and β . Lower right: proportion of the budget spent on computing silent networks according to α and β	224
C.1	Gantt chart for experiments on MNIST using LAVA-DL	226
C.2	Gantt chart for experiments on MNIST using SpikingJelly	227
C.3	Gantt chart for experiments on DvsGesture using LAVA-DL	228
C.4	Gantt chart for experiments on DvsGesture using SpikingJelly	229

Introduction to Spiking Neural Networks

Neuromorphic computing is a recent and interdisciplinary research field ranging from computational neuroscience [90] to electronic engineering [296]. In these past years, Neuromorphic ML has evolved rapidly, but there is still a long way to go to reach predictive performances of usual ML [47, 294, 238]. Our standpoint within the manuscript is from the ML and global optimization communities. Neuromorphic ML is peculiar as it is strongly tied to assumptions and advances of the two previous fields. We need to place ourselves on a spectrum between the biologically plausible and the electronically feasible. But, thanks to the rise of efficient simulators that have quickly evolved since the beginning of this thesis, the ML-practitioner can decide which hypothesis or constraints he can discard. This allows to integrate usual ML techniques without being circumscribed to neurosciences or neuromorphic electronics. Hence, this chapter aims at describing and explaining our approach to neuromorphic ML.

1.1 Beyond von Neumann and silicon

In 1965, Gordon Moore observed that the number of transistors on a microchip would double approximately every two years; this statement was later popularized in 1975 by Carver Mead as the *Moore's law* [279]. Meanwhile, in 1974 Robert H. Dennard pointed out that as transistors shrink, their power density remains constant, enabling faster clock speeds without significantly increasing power consumption [48].

Although manufacturers tend to follow both empirical laws, over the last few years the miniaturization of MOSFET run into fundamental physical limits by reaching atomic scales. Therefore, further gains in traditional CPU speed and energy efficiency become increasingly difficult because of quantum effects and heat dissipation [265].

These incoming hard limitations have driven the development of parallel computing architectures like multicore processors and distributed systems. While improving chip performances by shrinking MOSFET and maintaining competitive energy consumption, today's performances are also obtained by scaling horizontally across multiple processors rather than increasing individual CPU speed. Indeed, the inherent SISD nature of a basic CPU limits its efficacy on certain tasks. For that reason, modern architectures include dedicated components named *accelerators*, especially for AI applications. One of the most widespread accelerators, known as GPU, allows SIMD information processing at a larger scale than multicore CPUs. Over the past 25 years, the popularization of GPUs and SIMD

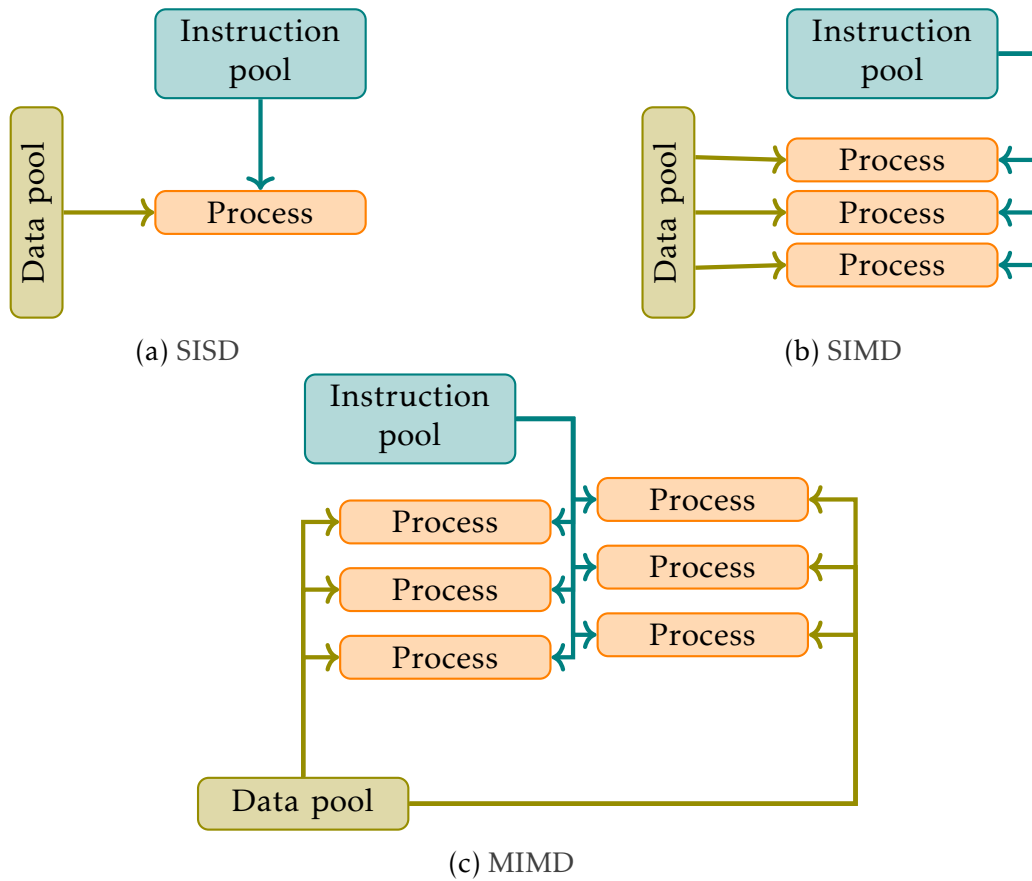


Figure 1.1: Flynn's information processing taxonomy.

processing has accelerated the development of AI, and in particular neural networks where heavy matrix-matrix and matrix-vector operations are essential. Then, with the advent of the IoT and embedded systems, even more specialized chips, such as ASICs, are needed to maximize performances for very specific tasks.

Despite exponential improvements since about 50 years, the organization of hardware components still follows the same *von Neumann* architecture, which was first formalized in 1945 within the *First Draft of a Report on the EDVAC* [278]. The flexibility of this fundamental architecture allowed the design of numerous systems, mostly characterized by physically separated processing and memory units. Today, this separation results in the so-called *von Neumann bottleneck*, where performances in terms of operations and energy [128] are mostly limited by data transfer between the memory and processing units. To mitigate this phenomenon, intermediate smaller but faster memories named *cache* are placed in between the main memory and CPU. However, one has to consider a tradeoff between parallelism and memory, as cache is known to be more energy-consuming [128, 25].

The today's top performances of AI models on cognitive tasks, such as image recognition or text generation, are mainly achieved by ANNs. The popularization of AI solutions and their performances are achieved at the expense of the energy efficiency¹. In March 2023, following the astonishing success of GPT-4, an open letter asked to refocus research on other objectives than predictive performances². As we reach the physical limits of the current paradigm, reducing energy consumption of AI technologies becomes crucial.

Beyond von Neumann systems, more radical approaches are being explored to overcome previous bottlenecks, such as quantum or optical computing. By the end of the 1980s, Carver

¹<https://hai.stanford.edu/news/2023-state-ai-14-charts>

²<https://futureoflife.org/open-letter/pause-giant-ai-experiments/>

Mead introduced another disruptive architecture called neuromorphic computing [238, 175]. This architecture aims to explore brain-inspired analog computing, betting that biomimetic technologies would dramatically improve AI energy efficiency.

Indeed, the human brain can perform various cognitive tasks simultaneously with a power of about 20 watts [227, 296]. Whereas modern GPUs such as an NVIDIA Tesla V100, requires about 300 watts. Therefore, neuromorphic computing consists in processing information using more biologically plausible ANNs, known as SNNs. By mimicking how the brain works, neuromorphic computing also contributes to neuroscience, from which it takes its inspiration to model brain dynamics, neurons, and synapses [90].

Elementary components of a brain are its billions of neurons and trillions of synapses, corresponding to colocated processing and memory units [227, 296]. Information and communications within a brain take the form of action potentials, named *spikes* and are considered as timed events.

Hence, neuromorphic computing and SNNs are bound to some constraints [238, 296]:

- *Collocated processing and memory*: As within the brain, SNNs components can at the same time process and memorize information. Conversely to von Neumann architectures, the memory component is physically local and closely connected to the neuron or synapse. Global information transfer is very limited, if not forbidden.
- *Massively asynchronous parallelism*: All neurons and synapses of a SNN are in theory fully independent and event-based. A neuron integrates potential and does computations only when being woken up by sparse events.
- *Scalability*: As information is processed by brain-like neural networks, scaling the system means adding more neurons and synapses.

1.2 Neuromorphic hardware

Basic CPUs are the versatile component of von Neumann architectures, able to complex operations and SIMD information processing. Some architectures are MIMD such as multicore systems with the Intel Xeon Phi or parallel computers. A GPU is made of numerous simple cores working in a synchronous SIMD fashion, allowing high-throughput computations. The neuromorphic approach could be considered a large-scale MIMD approach with numerous simple colocated computing cores and memory units. Hence, in a SNN, a neuron is a single independent and asynchronous computing unit with colocated memory. However, one major challenge of neuromorphic is the efficient hardware implementation of SNNs while maintaining competitive computation speed and energy efficiency.

Neuromorphic computing is a recent research field trying to overcome von Neumann architecture; many disruptive hardware approaches and components are considered [25, 294, 296, 67, 34]. We distinguish three hardware platforms:

- *Analog*: This approach uses physic properties of silicon to model brain-like behaviors, such as the first approach of Carver Mead in 1980.
- *Digital*: Here usual boolean gates with MOSFET technologies are used to simulate SNNs.
- *Mixed*: A combination of both previous strategies helps to overcome some of their drawbacks.

Usual digital architectures include TrueNorth [4], SpiNNaker [198] or Loihi [42, 41]. While these chips are being discarded, since the beginning of this thesis in 2021, the neuromorphic community gave birth to many other digital technologies. This flourishing dynamic in electronics illustrates that neuromorphic is rapidly evolving.

In 2021, Intel released the Loihi2 [197] with a consumption of about 1 watt. The chip can instantiate up to 1 million neurons and a maximum of 120 million synapses. The same year, SpiNNaker2 was released, allowing to model 152 thousand neurons and 152 million synapses with 152 ARM cores, consuming between 2 and 5 watts. One year later, the Akida 1000 from Brainship was offered for sale ³. During the same year, two chips from SynSense were also released, a 28nm 1000 neurons chip named Xylo [22], and the Speck SoC including an analog DVS sensor and a digital processing unit [292]. Digital chips are more or less flexible, allowing to instantiate various neuron models or certain architectures, such as convolution.

Using analog components for core computations is usually justified by a closer imitation by-design of biological components, while maintaining yet improving computation or energy performances [294, 227]. Implementing biological dynamics directly by the component design allows improving chip compactness. One of the most popular approaches, exploits the subthreshold regime of transistors, where current-voltage properties become exponential. Such a mode allows modeling neuronal dynamics with a higher fidelity [25, 128, 227]. For example, Neurogrid [143] or ROLLS [220] use such a technology. However, these approaches are uncommon as they are prone to errors, newer, noisier, and harder to produce than their digital counterpart [227, 34, 67].

Neuromorphic computing represents an opportunity to explore new electronic components. For now, there is no standard implementation [294], and many other approaches are promising. Non-volatile memories with memristors can model synapses [296], for example with resistive random-access memory. Other disruptive approaches are also considered, such as spintronic [267] or organic electronics [271].

A more flexible, programmable, and popular digital approach are FPGA, or even their analog counterpart FPAA. These highly flexible chips help designing architectures for specific tasks [25, 296, 34, 67]. They use an array of programmable logic blocks that can be connected and configured by using a hardware description language such as VHDL.

1.2.1 Neuromorphic simulator

In this thesis, we mainly focus on the algorithmic part of SNNs. To overcome the hardware bottleneck, one can mimic behaviors of SNNs using simulators on CPUs and GPUs. Depending on the use case, one has to carefully select the most suitable simulator. Thus, we can define different families of simulators. From biologically plausible ones (Brian2[252], NEURON[113], NEST[91]) to ML-based ones (SLAYER [247], Norse[212], snnTorch[64], Bind-snet[105], SpikingJelly [72]). Some are GPU-accelerated (Brian2GeNN, PyTorch-based simulators); others can convert simulated models to hardware implementable ones (Nengo [18], Lava ⁴). Some simulators are designed around specific learning rules, from Hebbian-based ones to gradient-based learning (SLAYER [247], SpikingJelly [72]). Selecting a simulator can also be done according to a certain network topology, such as convolution (CSNN [67], SpykeTorch [186]). Simulators often have a Python interface but can be written in C++ [252, 91], even Scala [24] or based on PyTorch [105, 72]. The more recent ones can also be based on Jax, such as Rockpool ⁵ or Spyx [109].

Thus, selecting a simulator is a tough task for newcomers within the SNNs community, as

³<https://brainchip.com/akida-neural-processor-soc/>

⁴<https://lava-nc.org/>

⁵<https://rockpool.ai/>

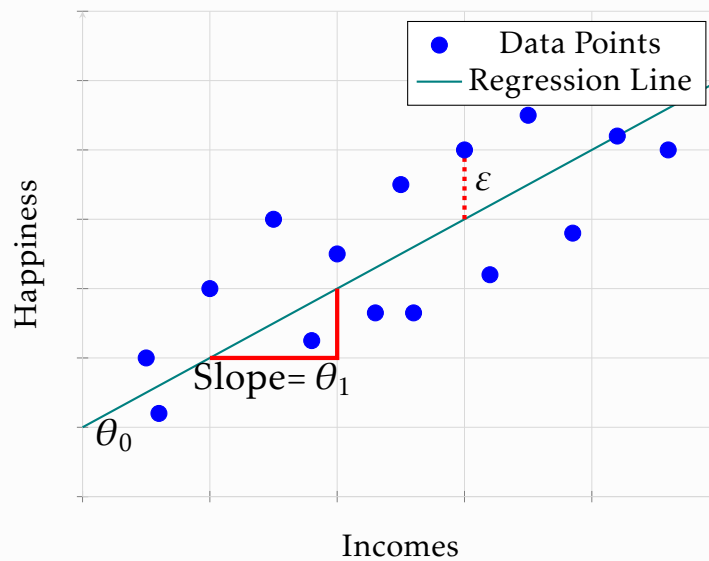
one has to deal with multiple standards [294]. For instance, convolution can be implemented with weight-sharing, which breaks the local memory property of SNNs, or by using a complex topology of excitatory and inhibitory filters [105] instantiable on neuromorphic hardware. Moreover, the flourishing number of simulators shows a growing interest in neuromorphic computing, but at the cost of reproducibility issues. Mastering and understanding various simulators can be a time-consuming task, which can be worsened by a lack of documentation or code standardization. To better select a simulator according to a use case, some simulator benchmarks can be found in [151, 173].

This growing number of simulators demonstrates the richness of the community, but also gives rise to major problems. We have to contend with a lack of standardization, generating difficulties of reproducibility and comparability between different works [294]. A few works tried unifying some simulators and hardware implementation. The PyNN [43] simulator combines Brian, NEST and NEURON simulators within a single descriptive language. A recent work published in August 2024, known as the Neuromorphic Intermediate Representation (NIR) [211], has drawn attention by unifying 7 modern simulators and 4 digital hardware platforms.

1.3 Machine Learning and Artificial Neural Networks

Linear regression is one of the most basic ML models. For example, suppose that money can – linearly – buy happiness. If we sample the incomes of some individuals and ask them to rate their level of happiness, we should see a linear relation between incomes and happiness. Then, a reasonable linear model on the incomes can be written as $f(x) = \theta_0 + \theta_1 x + \varepsilon$, with two *parameters* to be determined, resp. θ_0 the intercept and θ_1 the regression coefficient. The distances between the linear function and sampled points are prediction errors, denoted ε following a zero-mean Gaussian distribution. The values of the parameters θ_0 and θ_1 are *fitted* or *trained* on the samples.

Example: Linear regression



One major challenge is to *break the linearity* of the models when data follows a more complex distribution. To do so, techniques like polynomial regression, basis functions, or splines [131], are partial solutions if data can be explained by polynomial or specific assumptions. Some of these models, like feature maps or Gaussian processes, are discussed

in chapter 3 within the global optimization scope. Furthermore, due to the *curse of dimensionality*, a major challenge arises when a phenomenon is explained by tens or thousands of features, which can have a significant impact on modeling.

ANNs are ML models able to approximate non-linear and high dimensional data. They are computational models initially inspired by biological neural networks. For almost 80 years, the ANN community has been through many ups and downs, dating back to the 1940s [234]. Today, they form the foundation of modern ML, drawing considerable attention because of their exceptional performances in a wide range of applications such as image recognition, natural language processing, translation, etc.

The first use of an artificial neuron to model logical functions, such as AND and OR, dates back to 1943 with Warren McCulloch and Walter Pitts. They demonstrate that networks of binary neurons are at least as powerful as a Turing machine, laying the foundations of artificial neural structures. In 1949, a psychologist named Donald Hebb was the precursor of biological learning and the modern STDP rule. He gave birth to *Hebbian learning* stating that “Neurons that fire together, wire together” [169].

In 1958, Frank Rosenblatt developed the Perceptron, the first network able to do binary classification and able to solve linearly separable problems. The Perceptron was trained by an algorithm developed in 1951 by Herbert Robbins and Sutton Monro. The authors described the early stage of the SGD algorithm. In 1965, Ivakhnenko described the first multilayer neural network, which was later trained using SGD by Shun’ichi Amari in 1967.

But, the limitations of the Perceptron and current neuron models led to a decline of interest within the 1970s, a period referred to as the first *AI winter*. In the 1980s, the backpropagation algorithm, first described in 1970 by Seppo Linnaimaa, was rediscovered and applied to neural networks by researchers such as Geoffrey Hinton, Paul Werbos, David Rumelhart or David Parker. The backpropagation algorithm was applied on CNNs in 1989 by Yann LeCun for image recognition. The early stage of CNNs can be traced back to the Neocognitron proposed by Kunihiro Fukushima in 1979, inspired by works of Hubel and Wiesel on the visual cortex of cats. These works paved the way to modern pattern recognition.

By the end of the 1990s, ANNs suffered from their high complexity compared to other methods such as SVMs. But, the popularization of GPUs for scientific computing in the 2000s allowed significant acceleration of linear algebra computation, which led to the Deep Learning revolution and DNNs. In 1998, one of the first concrete applications was proposed by Yann LeCun with one of the first DNN known as LeNet-5, outperforming other ML approaches for handwritten character recognition.

The 2010s was the decade of the *AI spring*, a period of rapid development of various methods, and during which DNNs achieved breakthrough performances. A pivotal moment came in 2012 when AlexNet, a deep CNN developed by Alex Krizhevsky, Geoffrey Hinton, and their team, obtained the best performances on ImageNet. AlexNet ushered in the widespread use of CNNs in computer vision tasks. From 2012 to 2017, extensive research led to the arrival of generative AI. The popularization of LSTM and RNN gained popularity for handling sequential data like text, speech, and time series. In 2014, Ian Goodfellow suggested a new image generation network named GANs. It relies on the game theory between two adversarial networks, a generator, and a discriminator.

A key turning point in natural language processing occurred in 2017 with the introduction of the Transformer model in the paper “Attention is All You Need”. Unlike previous models, Transformers discarded recurrence and convolutions, relying entirely on attention mechanisms to model relationships between words. Transformers enabled faster training and scaling and became the backbone of numerous neural language processing advancements.

Since 2020, and thanks to the major breakthrough of Transformers, Large Language Models (LLM) have become increasingly popular, leading to a boom in generative models.

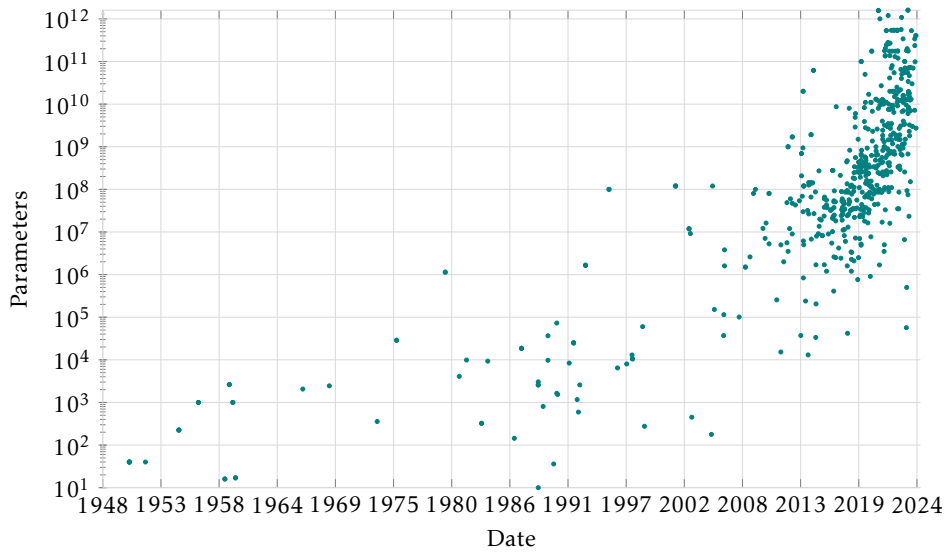


Figure 1.2: Number of parameters of major AI models through time. Data are from <https://ourworldindata.org/grapher/artificial-intelligence-parameter-count>

A frantic race for predictive performance ensued, involving ever larger and more energy-intensive models, which is illustrated in figure 1.2. In 2024, we currently observe a boom in image, video, text, and sound generation.

1.3.1 Breaking the linearity

In this section, we introduce some formalism about ANNs and SNNs, but more details are given in chapter 2 about ML and HPO.

The core idea behind ANNs is to simulate the way neurons in the brain communicate and learn from data. To do so, artificial neurons are simple computational units that receive input signals. They apply a non-linear differentiable function to it and generate an output. Neurons are organized in layers: an input layer receiving data, hidden layers extracting features, and an output layer providing the predictions. Neurons are connected via synapses weighted by w . These weights define the strength of the input and output signals of a neuron. Stacking a certain number of hidden layers gives a DNN capable of extracting complex features from the data and learning intermediate representations. Neural networks are mostly characterized by their weights expressed as matrices $\mathbf{W}^{(i)}$, $\forall i \in \llbracket 1, l \rrbracket$ for l layers. We can already notice that ANNs involves costly matrix-matrix multiplications, which can become a major drawback for large models. The weights have to be fitted on inputs, so to reduce the prediction errors of the model, this phase is called *training*. These fundamentals are illustrated in figure 1.3.

Neurons

A neuron transforms its inputs by passing them into two functions:

- the *transfer* function, usually a sum of all the inputs.
- the *activation* function, a non-linear and continuously differentiable function modifying the input and determining how much the neuron outputs.

In a feed-forward network, at least one neuron has to be non-linear, so the Universal Approximation Theorem holds [98, 117]. The theorem states that a feedforward neural network with a single hidden layer can approximate any continuous function, given sufficient

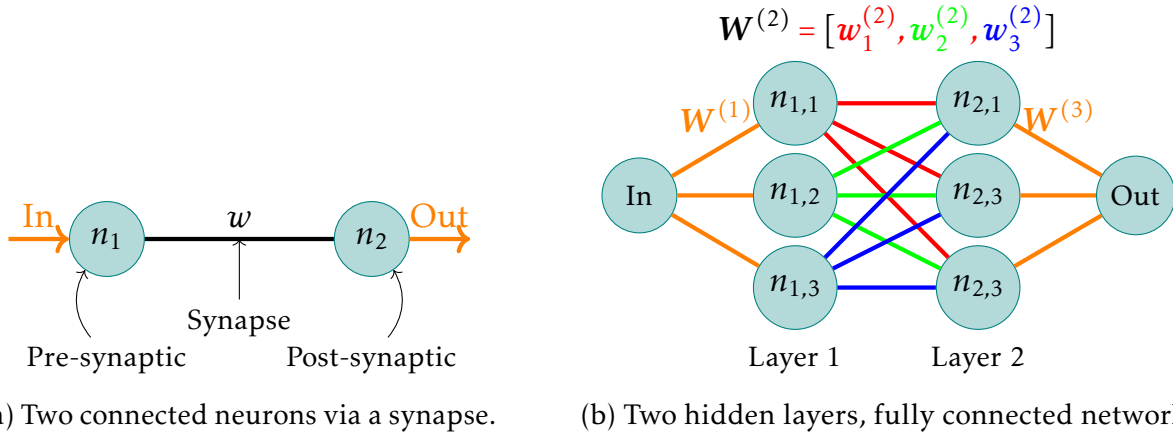


Figure 1.3: Basics of neural networks.

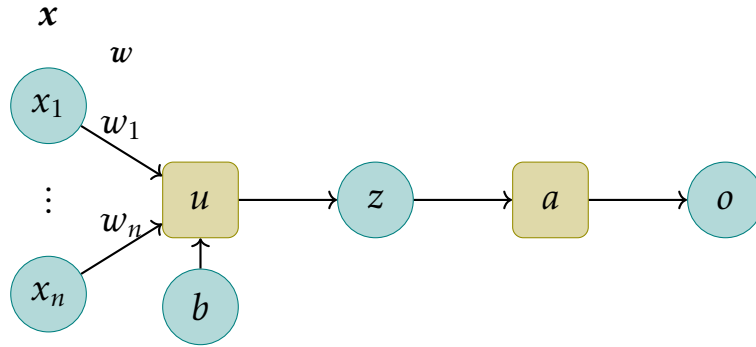


Figure 1.4: Graph of an artificial neuron.

neurons and appropriate activation functions. This theorem demonstrates the theoretical power of neural networks as universal function approximators, meaning they can model a wide variety of complex functions to an arbitrary degree of accuracy.

A neuron can be described as a computational graph of, an activation function a ; the outputs of the transfer function u are denoted z , those of a are written o , and the inputs are denoted x . Then, an activation function of a neuron maps $x \in \mathbb{R}^n$ into \mathbb{R} . A trainable bias b can be added within the transfer function, allowing affine transformation. The artificial neuron is illustrated in figure 1.4. The transfer function u w.r.t. incoming weights w , is usually denoted $u(x) = w^\top x + b$.

Concerning the activation function, a wide range of choices is available with different properties. For example, the *sigmoid* function can help to model probability distributions, while the ReLU is widely used in image recognition [98]. Some activation functions are illustrated in figure 1.5.

Example: Modeling a probability distribution

Let's take a binary classification task. We want to determine the probability of the inputs to be True or False, i.e. $\mathbb{P}(x = \text{True})$. Then because $\mathbb{P}(x = \text{True}) \in [0, 1]$, a convenient activation is the sigmoid function mapping a continuous input into $[0, 1]$:

$$a(x) = \frac{1}{1 + \exp(-x)}.$$

We can write the outputs of a layer k with weights $W^{(k)}$ and bias vector $b^{(k)}$, in a matrix form: $o^{(k)} = a^{(k)}(z^{(k)}) = a^{(k)}(u^{(k)}(o^{(k-1)})) = a^{(k)}(W^{(k)} \cdot o^{(k-1)} + b^{(k)})$. Then, the *forward pass* with inputs x , of a fully connected neural network with l layers can be written as a

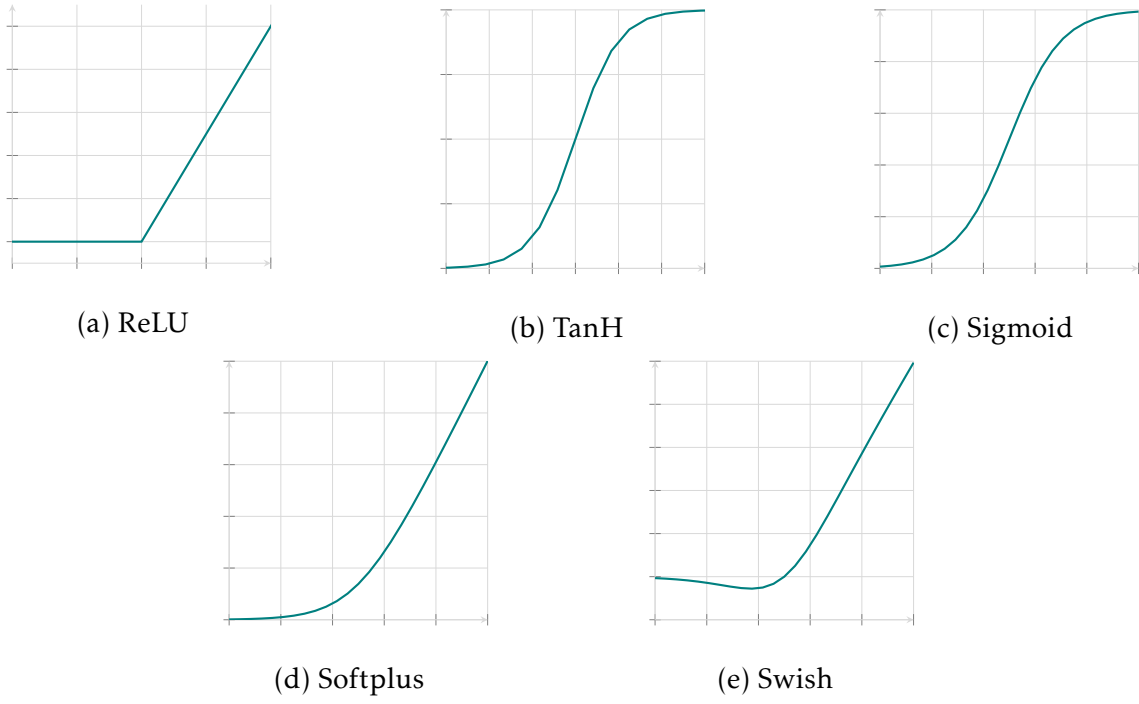


Figure 1.5: Some activation functions of a neural network.

composition of functions:

$$\mathbf{o}^{(l)} = \left(a^{(l)} \circ u^{(l)} \circ a^{(l-1)} \circ u^{(l-1)} \circ \dots \circ a^{(1)} \circ u^{(1)} \right)(\mathbf{x}) ,$$

with $g^{(1)}$ and $u^{(1)}$ the transfer and activation functions of the first layer, and $\mathbf{o}^{(l)}$ are the outputs from the network.

Example: Computations of layer 2 from figure 1.3b

$$\begin{aligned}
 \mathbf{o}^{(2)} &= a^{(2)}(\mathbf{W}^{(2)} \cdot \mathbf{o}^{(1)} + \mathbf{b}^{(2)}) \\
 &= a^{(2)}([\mathbf{w}_1^{(2)}, \mathbf{w}_2^{(2)}, \mathbf{w}_3^{(2)}] \cdot \mathbf{o}^{(1)} + \mathbf{b}^{(2)}) \\
 &= a^{(2)}\left(\begin{bmatrix} \mathbf{w}_{1,1}^{(2)} & \mathbf{w}_{2,1}^{(2)} & \mathbf{w}_{3,1}^{(2)} \\ \mathbf{w}_{1,2}^{(2)} & \mathbf{w}_{2,2}^{(2)} & \mathbf{w}_{3,2}^{(2)} \\ \mathbf{w}_{1,3}^{(2)} & \mathbf{w}_{2,3}^{(2)} & \mathbf{w}_{3,3}^{(2)} \end{bmatrix} \cdot \begin{bmatrix} o_1^{(1)} \\ o_2^{(1)} \\ o_3^{(1)} \end{bmatrix} + \begin{bmatrix} b_1^{(2)} \\ b_2^{(2)} \\ b_3^{(2)} \end{bmatrix}\right)
 \end{aligned}$$

1.3.2 Backpropagation of the errors

Suppose now that inputs \mathbf{x} , e.g. incomes, are associated with a response y , e.g. happiness. We want the neural network to approximate any y according to any \mathbf{x} . Thus, we build d pairs of inputs and responses $\mathcal{X} \times \mathcal{Y} \triangleq [(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_d, y_d)]$, with d sufficiently high to approximate y .

We need a function defining the correctness of an approximation $\hat{y} = \mathbf{o}^{(l)}$, i.e. the outputs of the last layer of a neural network. We write such a function $L : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ and we name it the *loss function*.

Extracting relevant features from input data involves setting the weights \mathbf{W} of a neural network $\mathcal{N}_{\mathbf{W}}$, in order to minimize the approximation errors defined by the expected loss $\mathbb{E}(L | \mathcal{N}_{\mathbf{W}}, \mathcal{X} \times \mathcal{Y})$.

Computing the gradients

To update the weights, the gradient of the prediction errors has to be *backpropagated* through the network, which is possible thanks to the differentiability of the acquisition and the loss functions regarding their inputs. The BP of the gradient is based on the chain rule stating that for two differentiable functions f and g , if $h = f \circ g$ then $h' = (f' \circ g)g'$.

Considering, r_k the number of neurons at layer k , $z_i^{(k)}$ the output of the transfer function, $o_i^{(k)}$ the outputs of the activation function of neuron i at layer k , and $w_{j,i}^{(k)}$ the weight between node i at layer k and node j at layer $k-1$. The gradient of the loss L w.r.t. the weights at a layer $1 \leq k \leq l$ is given by:

$$\frac{\partial L}{\partial w_{j,i}^{(k)}} = \frac{\partial L}{\partial o_i^{(k)}} \cdot \frac{\partial o_i^{(k)}}{\partial z_i^{(k)}} \cdot \frac{\partial z_i^{(k)}}{\partial w_{j,i}^{(k)}} , \quad (1.1)$$

with $\frac{\partial L}{\partial o_i^{(l)}} = L'(y, o_i^{(l)})$ the derivative of the loss function at the last layer l , and $\frac{\partial o_i^{(k)}}{\partial z_i^{(k)}} = a'(z_i^{(k)})$ the derivative of the activation.

For a layer k , we write $\delta_i^{(k)} = \frac{\partial L}{\partial o_i^{(k)}} a'(z_i^{(k)})$, and $\frac{\partial z_i^{(k)}}{\partial w_{j,i}^{(k)}} = \frac{\partial}{\partial w_{j,i}^{(k)}} \left(\sum_{n=1}^{r_{l-1}} w_{n,i}^{(k)} o_n^{(k-1)} \right) = o_j^{(k-1)}$. Then:

$$\frac{\partial L}{\partial w_{j,i}^{(l)}} = \delta_i^{(l)} o_j^{(l-1)} , \quad \frac{\partial L}{\partial b_i^{(l)}} = \delta_i^{(l)} . \quad (1.2)$$

For a neuron i at layer $1 \leq k < l$, we write the errors as:

$$\delta_i^{(k)} = a'(z_i^{(k)}) \sum_{n=1}^{r_{k+1}} \delta_n^{(k+1)} w_{j,i}^{(k+1)} . \quad (1.3)$$

After the *forward pass* the values $\mathbf{o}^{(k)}$ and $\mathbf{z}^{(k)}$ are known. Therefore, by using equations 1.2 and 1.3 we can iteratively compute $\delta^{(k)}$ from the output layer to the input layer; the gradient is backpropagated for all triplets $(\mathbf{x}, \mathbf{y}, \hat{\mathbf{y}})$. This is illustrated in algorithm 1 for a feed-forward network where the previous variables are vectorized.

Algorithm 1 BP for a feed-forward network

- 1: **for each** $\mathbf{y} \in \mathcal{Y}$ **do**
- 2: $\delta^{(l)} \leftarrow L'(\mathbf{y}, \hat{\mathbf{y}}) a'(\mathbf{z}^{(l)})$
- 3: **for** $k \leftarrow l-1$ **to** 1 **do**
- 4: $\delta^{(k)} \leftarrow a'(\mathbf{z}^{(k)}) \delta^{(k+1)} \mathbf{W}^{(k+1)}$
- 5: $\frac{\partial L_{\mathbf{y}}}{\partial \mathbf{W}^{(k)}} \leftarrow \delta^{(k)} \mathbf{o}^{(k-1)}$
- 6: $\frac{\partial L_{\mathbf{y}}}{\partial \mathbf{b}^{(k)}} \leftarrow \delta^{(k)}$
- 7: $\frac{\partial L}{\partial \mathbf{W}^{(k)}} \leftarrow \frac{1}{|\mathcal{Y}|} \sum \frac{\partial L_{\mathbf{y}}}{\partial \mathbf{W}^{(k)}}$
- 8: $\frac{\partial L}{\partial \mathbf{b}^{(k)}} \leftarrow \frac{1}{|\mathcal{Y}|} \sum \frac{\partial L_{\mathbf{y}}}{\partial \mathbf{b}^{(k)}}$
- 9: **Update** $(\mathbf{W}^{(k)}, \mathbf{b}^{(k)})$

Gradient descent algorithm

Gradient descent

In algorithm 1, line 9 updates the weights and biases using the mean of the gradient computed on each pair (\mathbf{x}, \mathbf{y}) and predictions $\hat{\mathbf{y}}$. Because the activation function is differentiable and because we can compute the gradient of the loss w.r.t. the weights and biases for every

neuron, first-order gradient-based optimizers can be applied to iteratively update \mathbf{W} and \mathbf{b} . The basic algorithm of gradient descent at each iteration t is given by:

$$\begin{aligned}\mathbf{W}_{t+1}^{(k)} &= \mathbf{W}_t^{(k)} - \lambda_{\nabla} \frac{\partial L}{\partial \mathbf{W}_t^{(k)}} , \\ \mathbf{b}_{t+1}^{(k)} &= \mathbf{b}_t^{(k)} - \lambda_{\nabla} \frac{\partial L}{\partial \mathbf{b}_t^{(k)}} ,\end{aligned}\tag{1.4}$$

with λ_{∇} the learning rate.

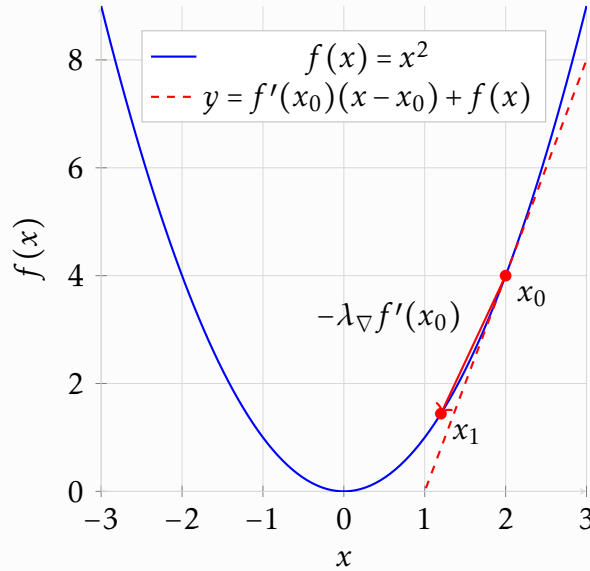
Computing the gradient on all $(\mathbf{x}, y) \in \mathcal{X} \times \mathcal{Y}$ is costly if there is a high number of pairs. For that reason, a solution is mini-batch SGD [23]. Here, instead of updating the weights once the gradient is computed for all pairs, we compute the gradients on subsets of the pairs, i.e. batches. The parameters are updated for each subset $\mathcal{X}' \times \mathcal{Y}' \subset \mathcal{X} \times \mathcal{Y}$, until all gradients for all (\mathbf{x}, y) contributed to the update. Because gradient descent is an iterative algorithm, multiple *forward* and *backward* passes on all batches have to be computed. A single forward and backward pass, followed by parameter updates for all batches, is named an *epoch*.

To accelerate the optimization and go beyond local optima, a momentum β_{∇} can be integrated into the equations. For example, on the weights update:

$$\begin{aligned}\Delta \mathbf{W}_{t+1}^{(k)} &= \beta_{\nabla} \mathbf{W}_t^{(k)} - \lambda_{\nabla} \frac{\partial L}{\partial \mathbf{W}_t^{(k)}} , \\ \mathbf{W}_{t+1}^{(k)} &= \mathbf{W}_t^{(k)} - \lambda_{\nabla} \frac{\partial L}{\partial \mathbf{W}_t^{(k)}} + \beta_{\nabla} \Delta \mathbf{W}_t^{(k)} .\end{aligned}\tag{1.5}$$

State-of-the-art optimizer for ANNs such as ADAM [147] uses two momentum, $\beta_{\nabla 1}, \beta_{\nabla 2}$.

Example: Gradient descent



1.3.3 Architectures and topology

The topology of a network describes how neurons are connected to each others. This was previously described in figure 1.3. Neurons are usually grouped into layers, for which we define the transfer and acquisition functions, the number of neurons, and other properties of that layer.

Feed-forward neural network

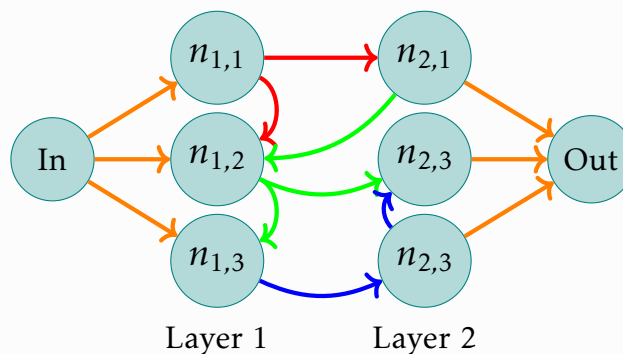
A feed-forward network is a set of layers stacked in such a way that the flow of information goes in one direction, from the inputs to the outputs. This is the most basic neural network, also named *multi-layer perceptron*. There is no cycle, and the forward path is similar to the one given in equation 1.3.1. The number of hidden layers allows finding complex relationships within the inputs. The complexity of the network, i.e. the depth and number of neurons, allows the networks to extract intricate features. It introduces a hierarchy between features; the deeper the network, the more abstract the extracted information. However, adding more hidden layers also increases computational complexity, which can influence the *bias-variance* tradeoff. Therefore, one has to carefully balance the number of layers and neurons. An example of a feed-forward fully connected ANN is given in figure 1.3.

Recurrent neural network

Conversely to feed-forward ANNs, RNNs allows cycles within the topology. RNNs are usually applied to time-series, as they can model some sort of memory. The recurrent connections allow modeling temporal dependencies and relationships in data. However, RNNs becomes less effective with long-term dependencies due to issues like *vanishing gradients*, limiting their ability to capture long-term features.

A solution to this are *cells*, which are a more complex model of neurons. For example, the LSTM or GRU allow learning how to forget information by controlling the impact of previous information on the cell's internal state.

Example: RNN



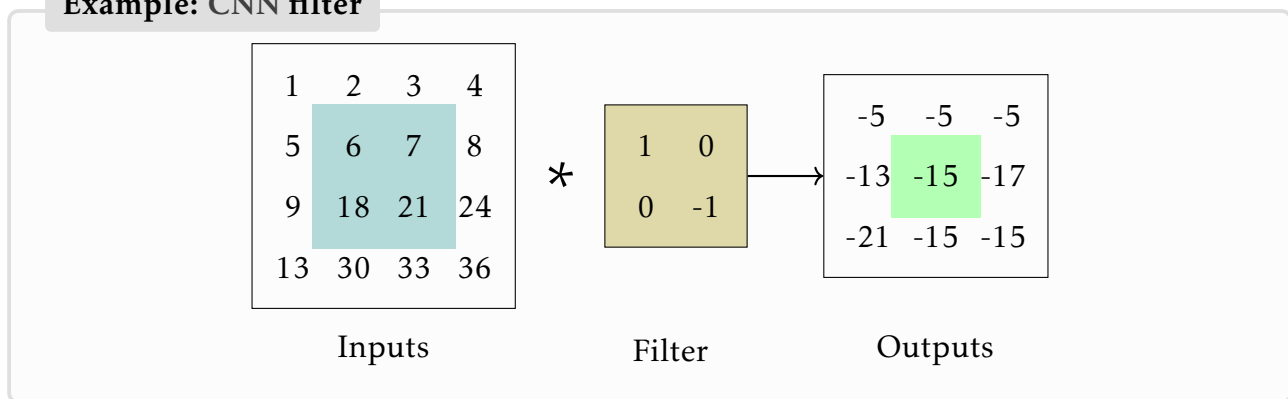
Convolutional neural network

CNNs are widely used in image recognition, and more generally when there is a spatial dependency between inputs. A CNN is mainly defined by its filters, which are small matrices of weights. These filters are rolled over the spatial data to extract patterns automatically. For example, this allows to detect edges, textures, and shapes within an image. CNNs reduce the number of weights by applying the same weights (matrix) for each location where the filter is applied. This is called *weight sharing*. Thus, having multiple filters rolling over the data allows learning and finding different patterns within the whole image regardless of the position. Depending on the property of the CNN layer, the network can act as a dimensionality reduction technique. For example, applying a filter to an image can produce a smaller image containing relevant features.

We can already notice a challenge of CNNs when applied to neuromorphic computing: non-locality of the weight sharing. Indeed, the weights of a CNN involve global information

shared across neurons.

Example: CNN filter



1.3.4 Usual challenges and remarks

ML models come with a set of significant challenges that can impact their performances or scalability. We previously discussed the impact of the energy consumption of usual AI and the necessity to move to another computational paradigm as neuromorphic computing. But most AI problems are NP-hard and rely on big data. In the following sections, we discuss some usual issues of ML such as overfitting, high dimensionality, or computational complexity.

Curse of dimensionality

The *curse of dimensionality* refers to various phenomena arising within high dimensional spaces.

First, high dimensionality has impacts on the geometry and combinatorics [273]. Dividing a d -cube into a grid of size g involves g^d smaller hypercubes of equal size. This property has an impact on some algorithms described in chapter 3. That is why the *grid search* algorithm does not scale with the dimensionality. Another geometrical phenomenon occurs on hyperspheres; their hypervolume (Hausdorff measure) tends to zero [273].

It also has impacts on sampling and distances. Because the hypervolume grows exponentially with the dimension, the number of necessary samples to efficiently approximate the number of dimensions also grows exponentially. But, in ML not all dimensions are independent, and some coordinates are usually collinear [273] which simplifies the geometry. Still, efficiently approximating a high dimensional function implies enough data, leading to Big data issues about storage or computational complexity.

Another phenomenon is the concentration of the norms, causing lower and lower interpretability of some distances [1, 181].

However, sometimes the dimensionality can be used to our advantage, a phenomenon known as the *blessing of dimensionality*. In chapter 3, we describe how *feature mapping* can make non-linearly separable data linearly separable into a higher dimensional space.

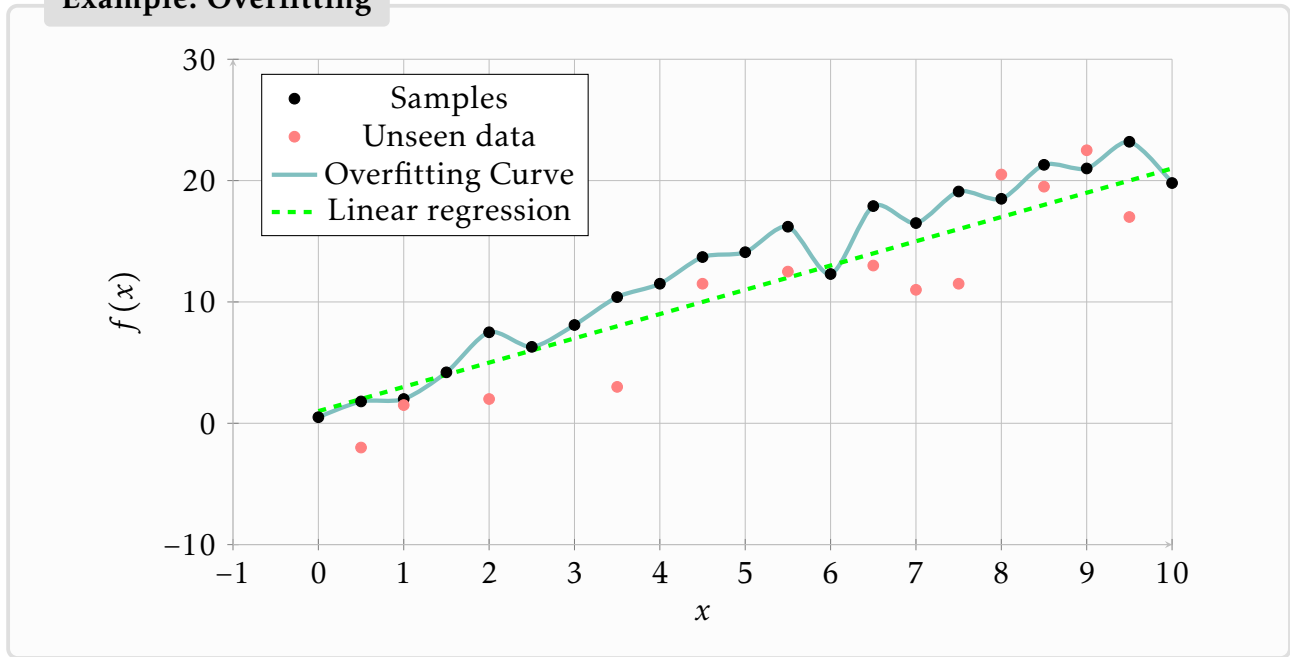
Bias-Variance tradeoff

The *bias-variance* tradeoff, describes the balance between two sources of errors that affect model performances:

- The *bias* describes errors due to an oversimplistic model failing at capturing relevant patterns within the data. This can also be explained by a lack of relevance between inputs and responses.
- The *variance* errors are due to *overfitting*, i.e. the model captures the random noise within data. This can be explained by a model that is too complex, which can make it sensitive to noise and can deteriorate its generalizability.

Achieving an optimal tradeoff between bias and variance is crucial for building models that generalize well to new data. That is why in a ANN balancing, the number of layers and neurons is crucial to prevent under- and overfitting.

Example: Overfitting



No Free Lunch Theorem

In optimization and ML, the No Free Lunch Theorem states that there is not a universal algorithm superior to all others across all possible problems [288]. This happens because algorithms assume certain properties about the problem they are trying to solve, but these assumptions may not be true across all scenarios.

A consequence of this theorem is that optimizing a problem or fitting a ML model requires minimal knowledge about the task. Thus, their performances depend on how well these assumptions align with the properties of the specific problem. This theorem emphasizes how important it is to choose the right algorithm for a given task, as no algorithm can outperform all others in every circumstances.

Supervised and unsupervised learning

In this manuscript, we tackle both supervised and unsupervised learning applied to SNNs.

In *supervised* learning, a model is trained using inputs \mathcal{X} and labels or responses \mathcal{Y} . The objective is to minimize the error between the predicted values and the actual labels by learning a mapping from inputs to outputs. Thus, the ML model has to extract relevant features from the inputs explaining the given labels.

In *unsupervised* learning, there is no explicit label or response. The algorithm is not guided and has to discover hidden patterns, structures, or relationships within the inputs. Since there are no labels, unsupervised learning relies solely on the characteristics of the data to guide its learning process.

1.4 Toward Spiking Neural Networks

SNNs are analog networks embracing a more biologically inspired and event-based approach compared to their counterpart, traditional ANNs. The distinctive characteristics of SNNs, such as in-memory and local computations, make them impractical for execution on usual von Neumann architectures. As previously discussed, they are computed on specialized neuromorphic hardware platforms, such as SpiNNaker [198] or Loihi [41]. SNNs use time as a resource for computations and communications [170]. A spike is a timed event mimicking the action potential, also named nerve impulses. In its simplest version and within digital hardware, a spike can be considered a binary event. Some more biologically accurate approaches can model the event by a bell-like function describing the intensity of the impulse through time. A spiking neuron is defined by ordinary differential equations w.r.t. time. It has a *membrane potential* describing its electrical potential, i.e. its excitation level. Once the membrane potential reaches a *threshold*, the neuron *fires* a spike at its output. These specificities make SNNs energy efficient, massively scalable, and parallelizable due to the theoretical asynchronicity between the neurons. SNNs rely on the dynamics between spikes within the network. Today and within an algorithmic point-of-view, when modeling with SNNs we face four major challenges [294, 238, 246]:

- Although SNNs are theoretically better than ANNs, their predictive performances remain inferior to that of ANNs [170, 214].
- Today, tools for modeling ANNs are well-developed. Even if newcomers like Jax are becoming increasingly popular, libraries like Pytorch or TensorFlow are standards that can be chosen by default by beginners. However, concerning the SNNs, there are dozens of simulators and libraries, mostly because neuromorphic computing ranges within a spectrum from neuroscience to electronics. Even for neuromorphic ML we can count many simulators: snnTorch, spykeTorch, Norse, Lava, Nengo, SpikingJelly, Spyx, Rockpool, Bindnet, Sinabs, Auryn, PySNN, N2S3, CSNN, SNAXX, etc.
- There is no established and satisfying neuromorphic benchmark broadly adopted by the community [294, 238]. Usual benchmarks for ANNs as MNIST, are widely used within the SNNs community. But it is uncertain if these benchmarks really emphasize the type of problems SNNs are made for [294].
- The choice of a training algorithm is still debated [257, 195]. Three main approaches are available, ANN-to-SNN conversion, plasticity rules, or surrogate gradient back-propagation. In-hardware training remains challenging. There is no consensus on the best algorithm to train SNNs, even if gradient-based approaches usually obtain the best performances.

1.4.1 Information coding

Spikes are a crucial component of SNNs. While ANNs compute with integers or floats, spikes are events used for communications and computations within SNNs. [170] A digital spike can be modeled as a binary timed event impacting the computation of weight-signal

multiplications compared to ANNs. To tackle usual digital datasets such as MNIST [153] or CIFAR-10 [149], these have to be converted. Other datasets are naturally neuromorphic such as DvsGesture [10] or DvsLip [263] as they are recorded by neuromorphic sensors.

A spike train can encode information in three major ways inspired by different biological coding [101, 11, 26], such as in the visual cortex [121]:

- *Rate coding*: The information is encoded within the spiking frequency (e.g. count or population rate).
- *Temporal coding*: The information is encoded within the spike timings (e.g. TTFS).
- *Mixed coding*: The information is both encoded by frequency and timings (e.g. burst coding).

When data is not directly encoded into spike trains, it has to be converted into spikes. The most popular way of doing it is Poisson coding [110]. It consists of converting a digital value into a spike frequency. The higher the digital value, the higher the frequency. A major drawback of such an approach is that it necessitates a lot of spiking activity to obtain good predictive performances, whereas fewer spikes can significantly decrease the latency [201].

Temporal encoded data is more challenging as it is more sensitive to noise and often results in lower predictive performances [235]. In frequency coding, the information is averaged through time, while a small perturbation within a spike timing modifies the carried information by temporal coding. However, such a coding method generates much less spiking activity. The most popular temporal conversion method is known as TTFS. Within an encoding time window, TTFS consists in encoding high digital values into earlier spikes, while lower values are converted into later spikes. Another approach, known as rank-order coding, defines the information encoded within the order of the spikes and not necessarily within their precise timing [266].

To prevent any information loss from digital-to-spikes conversions, the best datasets are the ones that are naturally neuromorphic, e.g. obtained using DVS cameras [161] or artificial cochlea [291]. These datasets are the most promising and maybe consensual ones for benchmarking SNNs [294]. The encoding and decoding methods are further discussed in chapter 2 within the HPO scope.

1.4.2 Neuron model

Spiking neurons contribute to modeling biological processes and execute computations in a more brain-like and energy-efficient manner. In a ANN, a neuron always receives digital information that is processed by the transfer and activation functions. Moreover, a ANN neuron always outputs digital information. Whereas, a SNNs incorporates the time dimension within its computations. Therefore, a neuron within a SNN, models temporal dynamics and event-based communications, meaning that a neuron does not always output a signal. Neurons in SNNs are sensitive not only to the strength of the inputs but also to the timing of spikes. The dynamic of a spiking neuron is illustrated in figure 1.6, where on the left are the input spikes, in the middle the membrane dynamic of a spiking neuron, and on the right the spiking outputs of that neuron.

One of the first neuron models, known as the IF and first described by Louis Lapicque in 1908 [152], is one of the simplest models defined by the following differential equation:

$$\begin{cases} C \frac{dV}{dt} = I & , \text{ if } V < V_{th} \\ V = V_{reset} & , \text{ else} \end{cases}, \quad (1.6)$$

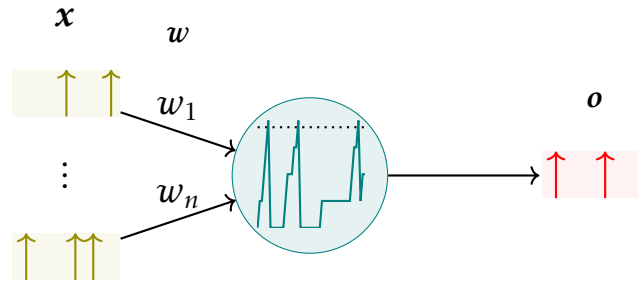


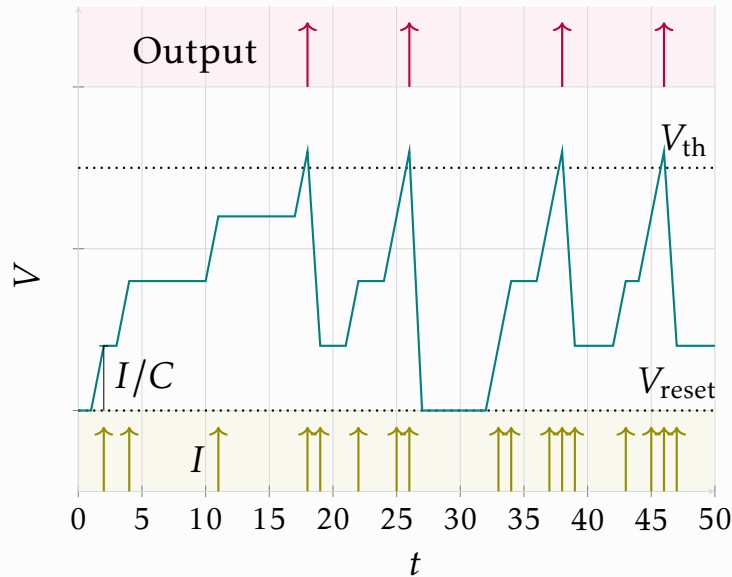
Figure 1.6: Graph of a spiking neuron.

with C the membrane capacitance, I the input current (spikes), V_{th} the threshold and V the membrane potential. Once V reaches V_{th} , the IF neuron fires a spike at its output. We further discuss a more advanced model known as the LIF in chapter 2. The latter is widely used in neuromorphic ML.

One of the most biologically accurate neurons, detailed in 1952 by Hodgkin and Huxley [114], describes the dynamic of the giant axon of a squid. It consists of 4 differential equations; 9 others describe 3 gating variables, and 3 other equations use gating variables to describe ionic flows within the neuron. Due to its complexity, this neuron is untractable to neuromorphic hardware and even within large-scale SNNs simulation.

The Izhikevich neuron [130] is an in-between. The model is simpler than the Hodgkin & Huxley neuron, but sufficiently complex to be more biologically accurate than the IF and LIF neurons. It consists of 2 differential equations and 5 parameters. According to the settings of the parameters, this neuron model describes various behaviors, such as the thalamo-cortical neuron, a resonator, a bursting neuron, etc. Even if the Izhikevich model is simpler than the Hodgkin & Huxley neuron, it is rarely used in neuromorphic ML which is dominated by LIF-like neurons.

Example: IF neuron



1.4.3 Training spiking neural networks

Training a SNN is challenging as the gradient of the dynamic of a spiking neuron w.r.t. a spike train is uninformative. Therefore, usual BP algorithms cannot be applied. We distinguish

three approaches to train a SNN: Hebbian or plasticity rules, surrogate gradient, and ANN-to-SNN conversion. These three approaches have different application cases, and depending on the problem, they have advantages and drawbacks.

Hebbian rules and local learning

Hebbian learning is a biologically plausible unsupervised learning rule relying on the quote “neurons wire together if they fire together” [169]. So, the weights \mathbf{W} of a SNN are updated locally and according to the spiking activity of pre- and post-synaptic neurons. Because these rules are local, they can be easily implemented on neuromorphic hardware.

STDP is an implementation of Hebbian learning [275]. While the standard Hebb rule relies on the exact timing of the pre- and post-synaptic neurons activity, STDP has several properties:

- Weights can grow or decay in the absence of activity.
- Post-synaptic spikes alone have effects.
- Pre-synaptic spikes alone have effects.
- Weights modifications rely on the timing differences between pre- and post-synaptic spikes.
- There is a dependency with the current synaptic weight.

There exists many variations of the STDP rules [275]. Almost all rules reinforce or depress a weight according to $\Delta_t = t_{\text{post}} - t_{\text{pre}}$, with t_{post} the timing of a post-synaptic spike, and t_{pre} the timing of a pre-synaptic spike. If Δ_t is negative, then the weight is decreased, as we consider that a pre-synaptic spike occurring shortly after a post-synaptic spike does not contribute to the firing process of the post-synaptic neuron. Conversely, if Δ_t is positive, then the post-synaptic spike can be partly explained by the pre-synaptic one, so the weight is increased. The closer Δ_t is to 0, the stronger the weight modification, which allows learning unsupervised temporal dependencies. While STDP can be easily instantiated on neuromorphic hardware [134], its performances are lower than the other training approaches. Moreover, to classify labeled data, STDP-based approaches require a decoder to read the outputs of the SNN. Supervised STDP rules, such as the reward-modulated STDP [185] can help overcome an additional training of a decoder. The standard STDP rule is given by:

$$\begin{aligned} \Delta w_{\text{pre,post}} &= \sum_{t_{\text{pre}}} \sum_{t_{\text{post}}} W(t_{\text{post}} - t_{\text{pre}}) \\ W(\Delta_t) &= \begin{cases} A_{\text{pre}} e^{-\Delta_t/\tau_{\text{pre}}} & \Delta_t > 0 \\ A_{\text{post}} e^{\Delta_t/\tau_{\text{post}}} & \Delta_t < 0 \end{cases} \end{aligned} \quad (1.7)$$

with $A_{\text{pre}} > 0$ and $A_{\text{post}} < 0$ the learning rates, τ_{pre} and τ_{post} time constants. One major drawback of this equation is that it needs to iterate over all previous pre- and post-spikes to determine $\Delta w_{\text{pre,post}}$. In chapter 2, we study a practical application of equation 1.7 and the effect of its parameters.

Other type of training

While STDP and Hebbian learning offer ways to perform in-hardware training, the resulting predictive performances are a major drawback. A solution is to perform the training offline

by using a simulator, and then instantiate the trained network within the neuromorphic hardware.

A straightforward approach is to convert an ANN trained via offline BP on spiking data, and then convert it into a hardware-friendly SNN for energy-efficient inference [238, 195, 228]. The challenge here is to find the right mapping between components of the ANN and the ones of the SNN. The conversion process usually assumes that the firing rate of SNNs is proportional to the activation of neurons from a ANN. But the downside is a decrease in performances during conversion, making the performance of SNNs behind that of ANNs.

Another approach is to directly apply the BP algorithm to SNN [247, 193]. However, in a SNN the activation function a of a classical ANN neuron is replaced by the dynamic of a spiking neuron, for which the gradient is uninformative. Then, a surrogate is applied to compute a usable gradient of the output spiking activity. A surrogate function approximates the firing process of a neuron with a differentiable function with informative gradient. This approach enables SNNs to be trained end-to-end on usual tasks directly from the spiking activity. Surrogate gradient training has helped unlock the full potential of SNNs, making them more competitive with conventional ANNs while retaining their advantages in terms of energy efficiency. However, usual BP algorithms cannot be instantiated within neuromorphic hardware, as it is non-local in space and time. We further discuss surrogate gradient and its parameters in chapter 2.

1.4.4 Topologies of spiking neural networks

While usual architectures such as feed-forward or RNN can be equally used with SNNs. Neuromorphic computing offers unique architectures thanks to its properties. For instance, inspired by the biological brain, within a SNN we distinguish two types of neurons:

- The *excitatory* neuron outputs a positive action potential, exciting other connected neurons.
- The *inhibitory* neuron outputs a negative action potential, decreasing the membrane potential of other connected neurons. When the inhibition is sufficiently strong to prevent other neurons from spiking, it can model a WTA mechanism where only one neuron can fire a spike at a time by inhibiting all other neurons.

In 2015, Peter Diehl and Matthew Cook proposed an unsupervised SNN architecture, obtaining competitive performances on MNIST compared to ANNs. The authors used a SOM architecture combined with lateral inhibition, STDP training, and max-spike decoding [51].

The architecture consists of an input layer, an excitatory layer, and an inhibitory layer. Both layers have strictly the same number of neurons. Each excitatory has a respective mirrored inhibitory neuron. The weights between inputs and the excitatory layer are trained by STDP. Two sets of weights connect the excitatory and the inhibitory layers. An excitatory neuron is connected all-to-all with the inhibitory layer, except for its respective mirrored neuron. Inhibitory neurons have fixed weights and one-to-one connections toward their respective excitatory neurons. The excitatory layer is also the output layer, and the dynamic between both layers models a WTA mechanism.

The architecture allows each neuron from the excitatory layer to specialize in a certain pattern of the input data. Indeed, for a given pattern, the specialized excitatory neuron will fire a spike toward its mirrored inhibitory neuron. In return, the excited inhibitory neuron will prevent all other non-specialized excitatory neurons from firing. Because STDP relies on the spiking activity between two neurons, thanks to the WTA a single neuron will fire according to an input pattern. This mechanism enables the reinforcement of the weights

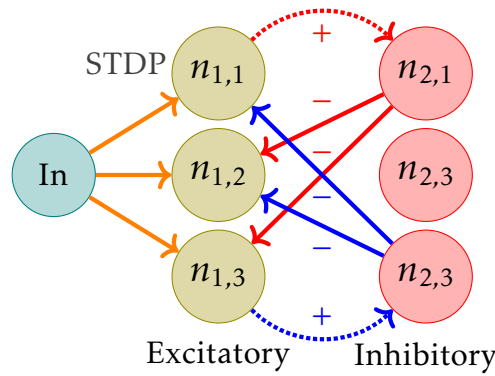


Figure 1.7: 1D Dielh & Cook self-organizing map. For clarity, the synapses of the middle neurons are not represented. Dotted lines illustrate inhibitory synapses with a fixed negative weight. While solid lines are excitatory synapses.

between the inputs and the excitatory layer for that single neuron on the specific pattern. For example, to classify MNIST, each neuron will be specialized into the recognition of a digit. The architecture is illustrated in figure 1.7.

Another unique architecture is reservoir computing [305]. It consists of a RNN with random weights, i.e. the *reservoir*. Then an output layer is connected all-to-all with the reservoir. Only the connection between the output and the layer can be learned. The weights of the reservoir are randomly fixed. This approach is usually applied when no direct training of the SNN is possible. Usually, the output layer is trained via BP, or another ML approach can learn to decode the outputs of the reservoir.

Because convolution involves non-local weight-sharing, CSNNs are challenging. Such architecture can be directly handled by design within the neuromorphic hardware [292]. Or, by using combinations of small Dielh & Cook maps, one can mimic convolution [231, 232]. The major drawback of this approach is the huge number of synapses needed to mimic non-local convolution.

1.5 Motivations

We previously described the baseline of this thesis. While the demand and cost of AI grow significantly, focusing on other objectives than solely on the predictive performances becomes crucial. Therefore, energy-efficient solutions such as SNNs are of the upmost interest. By rethinking the von Neumann architecture and usual silicon chips, SNNs could offer a serious alternative for certain tasks. Today, Neuromorphic ML has two major challenges. This first one concerns electronic engineering and how to design an energy-efficient neuromorphic chip by using various disruptive technologies. The other challenge concerns the algorithmic behind SNNs. In this thesis, we focus on the latter, and we consider that *some current* constraints of the hardware can be bypassed by using simulators. For example, convolution involves weight sharing, but the most recent neuromorphic hardware can instantiate some convolution layers [292, 197, 116].

At the beginning of this thesis in late 2021, the major algorithmic issues bounded to SNNs were:

1. Predictive performances
2. The training algorithm
3. Information encoding and decoding

4. Benchmarking
5. Standardized tools

These challenges were broadly shared among pre-2022 surveys [100, 227, 257, 107, 214, 264, 218, 236, 42, 275, 11, 101, 281, 208, 92, 165]. However, among all previous surveys, there is one blind spot: HPO. We previously described some components of SNNs and their parameters, e.g. the neuron threshold V_{th} . Such parameters that are not learned by the training algorithms are called *hyperparameters*. The major challenge is *how to set these hyperparameters efficiently to optimize given performance metrics*.

In 2008, the authors of [209] suggested a more in-depth investigation of tuning the hyperparameters of a spiking reservoir. Some very early works in 2010 applied HPO to SNN [289, 243]. The first thesis deeply investigating modern HPO applied to SNNs was the one of Maryam Parsa in 2020 [202]. The author applied a Bayesian optimization approach to multiple training algorithms, such as neuroevolution, surrogate gradient, and conversion. It also investigates mono- and multi-objective optimization and hardware implementation. This thesis is a milestone in HPO applied to SNNs as it considers HPO as the main subject of study. However, the search spaces of hyperparameters are limited; they are heavily discretized to reduce the complexity of the HPO. The author investigates the impact of the optimization on the SNNs performance metrics and the impact of some hyperparameters. The thesis compares its optimization approach to grid search, concluding that grid search is the less efficient algorithm. But very little analysis is dedicated to the HPO process itself when applied to SNNs and how it gets to the optimized solution.

1.5.1 Research question

In chapter 2, we review the literature of HPO applied to SNNs. We see that HPO is often considered a secondary task, and very little analysis is performed on the process. HPO of SNNs is performed the same way as HPO of usual ML models and ANNs. Indeed, we usually consider the optimized function as fully black-box, e.g. the predictive performances. That is, only the inputs and outputs of this black-box are used by the HPO algorithm. The inputs are the hyperparameters and data, the inside of the black-box is the SNN, and the outputs are the considered performance metrics.

Today, in the way HPO is applied to SNNs, we could replace the black-box containing the SNN by any other ANNs model applied to the same task without having to modify the HPO algorithm. Only the search space would change. The No Free Lunch theorem indicates that we cannot efficiently do optimization without prior knowledge about the problem.

Here we have an issue, why the usual HPO algorithms applied to ANNs would perform equally when applied to SNNs?

Thus, the research question through this manuscript is:

How can we improve HPO of SNNs by investigating both, the impact of HPO on the performances of SNNs, and the impact of SNNs on the performances of HPO?

1.6 Outline

In chapter 2 we formalize HPO and we discuss the state-of-the-art of HPO applied to SNNs. This chapter is the first attempt to survey the main works investigating HPO of SNNs. We describe almost all hyperparameters that will be later optimized and their local impact on the SNNs dynamics. In chapter 3, we describe more in depth the usual algorithms used for HPO. We start by describing metaheuristics. We then move to Bayesian optimization

and end up with multi-fidelity state-of-the-art HPO approaches. Likewise, we conclude the chapter by discussing the parallelization strategies of previous algorithms within a distributed environment.

Then, the following chapters detail the contributions of the thesis. In chapter 4, we formalize a new family of algorithms named *fractal-based decomposition algorithm*. We introduce a software and theoretical framework called *Zellij*. These algorithms hierarchically decompose the search space to better optimize; they are based on the “divide-and-conquer” principles. We also discuss the relationship between Bayesian optimization and fractal-based decomposition. In chapter 5, we introduce the concept of *silent networks*, and we investigate their impact via large-scale HPO experiments on the Jean Zay supercomputer. We empirically show that SNNs have an impact on the performances of HPO. We study the HPO process, the optimized solutions, and the sensitivity of hyperparameters. Finally, in chapter 6, we leverage silent networks and multi-fidelity to significantly accelerate the search process.

Automated machine learning and spiking neural networks

Selecting a ML model is a tough task. This process relies on the knowledge of the problem and of the user. This phase requires human-in-the-loop interactions, which, beyond its knowledge, also implies intuition and all our human biases prone to errors.

We consider a certain task, for example, predicting the daily electricity consumption of a country. The problem can be described as a time-series regression problem. To predict a daily value, one could look at what previously happened. But how far in the past should we focus our interest 12 hours, 24 hours, weekly, etc. At first glance, one could be tempted to apply a 24-hour time window or any of its multiples (6, 12, 48...). But counter-intuitive values might also result in similar yet better forecasting performances. Meanwhile, add to the historical electricity consumption other features such as meteorological data. Then interactions of the time window with the data become less and less intuitive.

If the time window cannot be learned by our hypothetical ML model, then it is a HP and the process of tuning its value to optimize the quality of the predictions from the model is called HPO.

A ML model can have tens, even hundreds of HPs. In HPO, the human-in-the-loop interaction is reduced to the choice of a ML model, HPO optimizer, the HPs selection, and the definition of the search space. For this reason, HPO offers a higher flexibility in the design of a solution for a given ML task, including exploration of counter-intuitive possibilities.

A higher level problem, known as AutoML, generalizes HPO by automating the selection of an appropriate ML model for a given task.

2.1 Hyperparameter Optimization

AutoML is a recent research domain that was first formalized by Auto-weka and ICML workshops in 2014 [244]. But model selection and HPO are much older (resp. 1970s and 1990s) [244, 123]. Thus, this thesis aims at better understanding HPO and its mechanism when applied to SNNs to better inform and tackle more challenging problems such as NAS or AutoML. AutoML of ANNs can be divided into two subproblems, HPO and NAS [108, 260, 123, 16, 20]. Another sub-problem, known as Meta-learning, i.e. *learning to learn*, is also part of AutoML. It aims at applying, optimizing, and generalizing ML models across problems, tasks, and datasets.

Throughout this manuscript, the focus is on the HPO problem, as a more profound understanding of this problem applied to SNNs is necessary before efficiently tackling harder

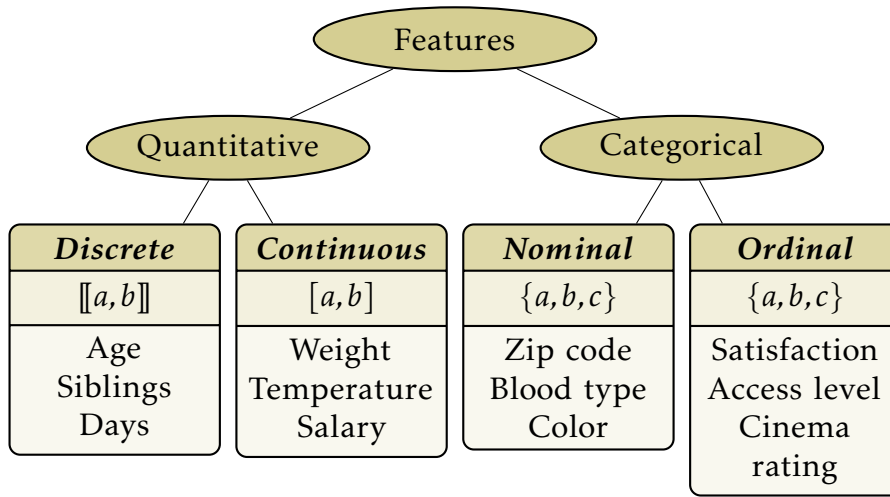


Figure 2.1: Types of features

ones such as NAS. The design of ANNs and DNNs also falls into the range of problems tackled by AutoML [299]. Manual tuning helps to better understand the impact of some HPs but can lead to contradictory results due to the complex interactions between HPs, ANN architecture and the data [249]. Therefore, AutoML can help to inform two of the most important challenges bound to DL which are interpretability and explainability [20].

2.1.1 Formal definition

The formal definitions of the next subsections are mainly inspired by the following books [131, 242, 123].

Abstract machine learning model

In classification or regression problems, a ML model \mathcal{A} is applied to a dataset \mathcal{D} made of rows and columns, resp. *records* and *features*. Features can be of different natures, as described in figure 2.1: continuous, discrete, nominal, or ordinal.

In this thesis, we focus on SNNs applied to classification tasks. So a dataset is divided into a qualitative *response* \mathcal{Y} and p different *predictors* (rows) $\mathcal{X} = X_1, \dots, X_p$, where p is the size of the dataset. Each predictor is a vector of q *features*, $\forall i \in \llbracket 1, p \rrbracket, X_i = (x_1, \dots, x_q)$. Then, a dataset \mathcal{D} is made of pairs $\mathcal{D} \triangleq \mathcal{X} \times \mathcal{Y}$. The response here is made of classes, e.g. $\mathcal{Y} = \{\text{cat, dog, duck, cat}, \dots\}$.

Example: Dataset

\mathcal{X}	x_1	x_2	\dots	x_q		\mathcal{Y}
	Citric acid	Sulphites	\dots	Sugar	Province	
X_1	0.0	7.4	\dots	1.9	Alsace	y_1
X_2	0.0	7.8	\dots	2.6	Mosel	y_2
\vdots	\vdots	\vdots	\ddots	\vdots	\vdots	\vdots
X_p	0.4	7.8	\dots	2.3	Sicily	y_p

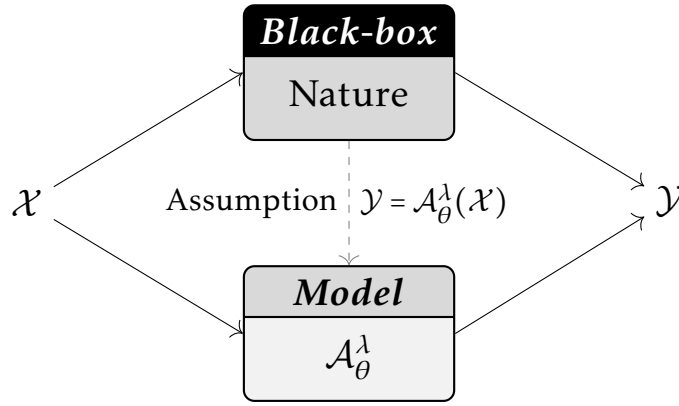


Figure 2.2: Modeling "natural" probability distribution of data with ML.

Definition 1 (Model [131, 16]). Let $\mathcal{D} \triangleq \mathcal{X} \times \mathcal{Y}$ a dataset made of predictors and response. A model \mathcal{A} of \mathcal{X} on \mathcal{Y} parameterized by θ and hyperparameterized by λ defines a relationship between \mathcal{X} and \mathcal{Y} as:

$$\mathcal{Y} = \mathcal{A}_{\theta}^{\lambda}(\mathcal{X}) + \varepsilon, \quad (2.1)$$

where ε is a random error independent of \mathcal{X} .

So, selecting a model $\mathcal{A}_{\theta}^{\lambda}$ is one of the first assumptions made by the user on the natural probability distribution of the data. This is illustrated by figure 2.2. AutoML tends to replace this manual selection by instead automatically choosing a ML model among a set of models and according to a quality metric on the predictions.

To evaluate the prediction errors of $\mathcal{A}_{\theta}^{\lambda}(\mathcal{X})$ on \mathcal{Y} , we define l as a loss function that needs to be maximized, i.e. maximize the quality of the predictions or minimize the prediction errors.

Definition 2 (Generalized Loss Function). Given a domain \mathbb{D} and the set of all possible models \mathbb{A} . A loss function l is defined as

$$l : \mathbb{A} \times \mathbb{D} \rightarrow \mathbb{R}^+, \quad (2.2)$$

then the risk function is the expected loss of a model $\mathcal{A}_{\theta}^{\lambda} \in \mathbb{A}$, w.r.t. a probability distribution \mathbb{D} over \mathbb{A} , namely:

$$\mathcal{L}_{\mathbb{D}} \triangleq \mathbb{E}_{z \sim \mathbb{D}} [l(\mathcal{A}_{\theta}^{\lambda}, z)]. \quad (2.3)$$

So over a given dataset $\mathcal{D} \triangleq \mathcal{X} \times \mathcal{Y} = (z_1, \dots, z_p) = ((X_1, y_1), \dots, (X_p, y_p))$ we have:

$$\mathcal{L}_{\mathbb{D}} \triangleq \frac{1}{p} \sum_{i=1}^p l(\mathcal{A}_{\theta}^{\lambda}, z_i). \quad (2.4)$$

Definition 3 (Model parameters [16]). Model parameters $\theta \in \Theta$ are chosen among a set Θ during the training phase on $\mathcal{D}_{\text{train}}$, of a model \mathcal{A}_{θ} .

Definition 4 (Training). The training problem of a model \mathcal{A}_{θ} can be defined as the maximization of a risk function $\mathcal{L}_{\mathbb{D}}$ over a training dataset sampled from the natural distribution $\mathcal{D}_{\text{train}} \subset \mathbb{D}$. To find an optimal set of parameters $\theta^* \in \Theta$ we write:

$$\theta^* \in \operatorname{argmax}_{\theta \in \Theta} \mathcal{L}_{\mathbb{D}}(\mathcal{A}_{\theta}, \mathcal{D}_{\text{train}}) \quad (2.7)$$

Example: Loss function

In classification problems the accuracy of the predictions is given by:

$$\text{Accuracy}(\mathcal{A}_\theta^\lambda, \mathcal{X}, \mathcal{Y}) = \frac{1}{p} \sum_{i=1}^p \mathbb{1}_{\{\mathcal{A}_\theta^\lambda(X_i) = y_i\}} \quad (2.5)$$

Or the MSE in regression tasks:

$$\text{MSE}(\mathcal{A}_\theta^\lambda, \mathcal{X}, \mathcal{Y}) = \frac{1}{p} \sum_{i=1}^p \|\mathcal{A}_\theta^\lambda(X_i) - y_i\|^2 \quad (2.6)$$

Example: Parameters

Examples of a set of parameters θ could be:

- Coefficients of a linear regression: $y = \theta_0 + \theta_1 x_1$.
- Lengthscales l of a RBF kernel in a GP regression: $K(x, x') = \sigma^2 \exp\left(-\frac{(x-x')^2}{2l^2}\right)$.
- Weights W of a ANN or SNN.
- Split points and node value of a decision tree.
- Centroids in K-means clustering.

Definition 5 (Testing). *To assess the final performances of a trained \mathcal{A}_{θ^*} , a hold-out dataset $\mathcal{D}_{\text{test}} \subset \mathbb{D}$ is used over the risk function $\mathcal{L}_{\mathbb{D}}$ such that:*

$$\mathcal{L}_{\mathbb{D}}(\mathcal{A}_{\theta^*}, \mathcal{D}_{\text{test}}) \quad (2.8)$$

We described the usual training and testing of a ML model without any tuning. When additional steps are involved in the design of a ML solution, an intermediate additional set $\mathcal{D}_{\text{valid}}$ is needed.

Dataset management in AutoML

Usually, data is prepared, cleaned, and sometimes augmented to prevent overfitting [108]. In this work, very little data pre-processing will be applied, as usual high-quality and curated benchmarks will be used. Thus, we are not concerned about data collection, but pre-processing specific to SNN and neuromorphic computing will be discussed in section 2.2.

As previously said, overfitting can be partly caused by a lack of data. In chapter 1 we explain the bias-variance tradeoff of ML as a potential source of overfitting. In AutoML, even when manually tuning a model for a given task, a less known bias is introduced. Indeed, tuning the architecture in NAS or HPs in HPO on the hold-out dataset, can result in a model that overfits the architecture or HPs [20, 120].

Usually, a dataset \mathcal{D} is split into a *training* $\mathcal{D}_{\text{train}}$, and *testing* (a.k.a. *hold-out*) $\mathcal{D}_{\text{test}}$ datasets. Then $\mathcal{D}_{\text{test}}$ should remain untouched until the very final phase of testing the generalizability of the selected and trained model. But, because AutoML requires an additional step between the training and testing, one should not select a ML model and perform AutoML on $\mathcal{D}_{\text{test}}$. This is illustrated in figure 2.3.

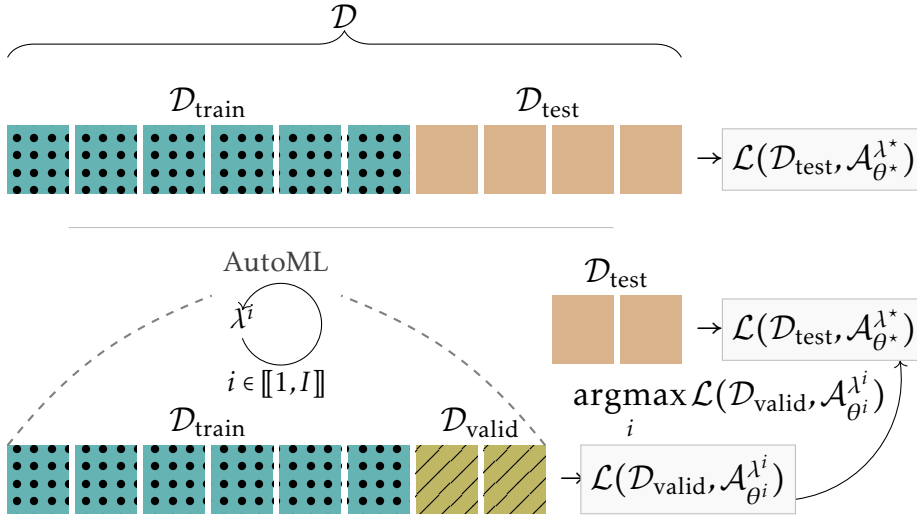


Figure 2.3: Training - Validation - Test splits of a dataset

So, because $\mathcal{D}_{\text{test}}$ must remain unseen during AutoML, a third split, known as the *validation* set $\mathcal{D}_{\text{valid}}$, is introduced. The AutoML (or any tuning) optimizes the performances of the model on the validation dataset. Once done, the performances of the tuned model are finally assessed on $\mathcal{D}_{\text{test}}$. This method does not prevent overfitting but emphasizes it if it exists.

Another approach is *Nested cross validation* [20, 120]. It addresses an issue of cross-validation when used with AutoML, and where the optimization would be still applied on the $\mathcal{D}_{\text{test}}$ folds. Nested cross validation consists in two cross-validation loops; the first one splits the full data \mathcal{D} into $i \in \llbracket 1, K \rrbracket$ folds, $\mathcal{D}_{\text{train},i}$ and $\mathcal{D}_{\text{test},i}$. An inner loop is applied on all $\mathcal{D}_{\text{train},i}$ to build $j \in \llbracket 1, Q \rrbracket$ inner-folds s.t. $\mathcal{D}'_{\text{train},i}$ and $\mathcal{D}_{\text{valid},j}$ (see figure 2.4).

Nested cross validation gives an unbiased estimation of the performances of the AutoML process, but does not return an optimal set of HPs. One can select a combination of HPs among the returned ones and evaluate its performances by retraining and testing it on a full train-test split.

Nonetheless, the major issue with cross-validation and nested cross-validation is that it involves multiple evaluations on different sets of a single solution and multiple inner AutoML. We will see afterward that when the evaluation of a ML model or the AutoML is expensive, these solutions are intractable.

In figures 2.3 and 2.4, the objective is to maximize the losses \mathcal{L} of a ML model \mathcal{A} according to its parameters θ and hyperparameters λ .

Hyperparameter Optimization

Conversely to a parameter, a HP is fixed before the training of $\mathcal{A}_{\theta}^{\lambda}$ and cannot be learned. A HP should have an impact on at least one of the optimized objectives, or on other practical considerations such as hardware memory.

Definition 6 (Hyperparameter). *A hyperparameter is a parameter fixed before the training of a ML model $\mathcal{A}_{\theta}^{\lambda}$ hyperparameterized by a set of HPs $\lambda \in \Omega$, where Ω is the set of all possible HPs combinations.*

Like data, HPs can be of various types, continuous, discrete, categorical, etc. Compared to manual tuning, the advantage of HPO is the higher flexibility offered by the design of Ω .

HPO consists of two nested maximization problems, one on $\mathcal{D}_{\text{train}}$ and the other on $\mathcal{D}_{\text{valid}}$.

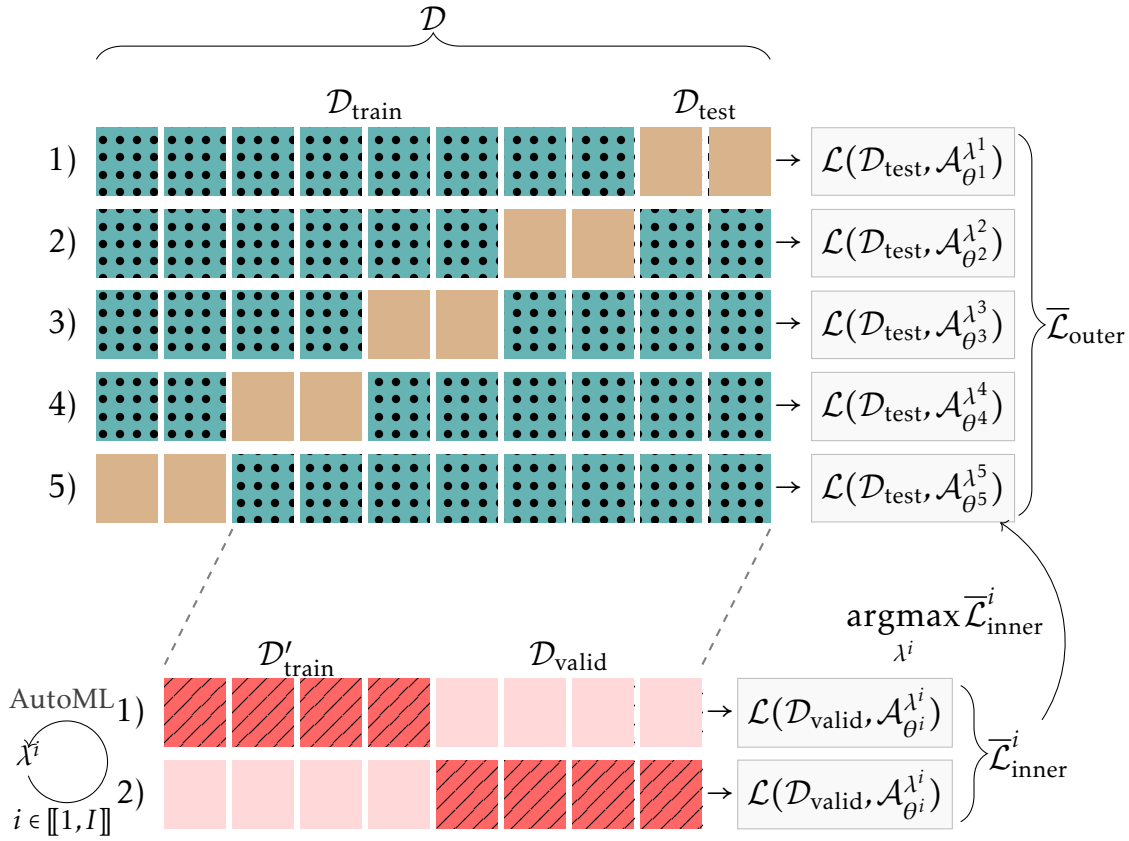


Figure 2.4: Nested cross validation

Definition 7 (HPO). Given a ML model $\mathcal{A}_{\theta}^{\lambda}$, s.t. $\theta \in \Theta$, $\lambda \in \Omega$, a parameter space Θ , a search space Ω , a training dataset $\mathcal{D}_{\text{train}} \subset \mathbb{D}$, a validation dataset $\mathcal{D}_{\text{valid}} \subset \mathbb{D}$ and a single or two different risk functions $\mathcal{L}_{\mathbb{D}}^{(1,2)}$. The mono-objective HPO process can be written as:

$$\lambda^* \in \arg\max_{\lambda \in \Omega} \mathcal{L}_{\mathbb{D}}^{(1)} \left(\mathcal{A}_{\theta^*}^{\lambda} \mid \theta^* \in \arg\max_{\theta \in \Theta} \mathcal{L}_{\mathbb{D}}^{(2)} (\mathcal{A}_{\theta}^{\lambda}, \mathcal{D}_{\text{train}}), \mathcal{D}_{\text{valid}} \right). \quad (2.9)$$

Note that in definition 7, one can use two different loss functions for HPO and the training phase of a neural network. For example in classification tasks, the training can optimize the binary crossentropy, while the HPO optimizes the accuracy.

The algorithm 2 describes sequential HPO using an abstract optimizer denoted as \mathcal{O} . This optimizer takes the previous HPs combination and its associated loss, the search space Ω

Example: Hyperparameter

Example of hyperparameters could be:

- Lagrange multiplier in Ridge regression.
- Kernel type in SVM (RBF, Matérn 5/2, ...).
- Number of neurons of a ANN or SNN.
- Maximum depth of a decision tree.
- Number of neighbors in K-Nearest Neighbors.

and returns the next set of HPs to evaluate. In practice, the line 17 is replaced by a training algorithm, i.e. gradient backpropagation for ANNs. In this algorithm, HPs combinations are evaluated one after each other. Strategies as EA return batches of HPs combinations. Thus, we discuss in chapter 3 parallelization strategies to accelerate the HPO process.

Algorithm 2 Sequential HPO

Inputs:

- | | |
|--|---------------------------------|
| 1: $\mathcal{D}_{\text{train}}$ | <i>Training set</i> |
| 2: $\mathcal{D}_{\text{valid}}$ | <i>Validation set</i> |
| 3: $\mathcal{D}_{\text{test}}$ | <i>Test set</i> |
| 4: $\mathcal{A}_{\emptyset}^{\lambda}$ | <i>ML algorithm to be tuned</i> |
| 5: $\mathcal{L}_{\mathbb{D}}$ | <i>Risk function</i> |
| 6: \mathcal{O} | <i>Optimizer</i> |
| 7: Ω | <i>Search space</i> |

Outputs: $\mathcal{A}_{\theta^*}^{\lambda^*}, \text{loss}_{\text{test}}$

- | | |
|--|-----------------------------|
| 8: $\lambda^* \leftarrow \emptyset$ | <i>Best HPs combination</i> |
| 9: $\theta^* \leftarrow \emptyset$ | <i>Best parameters</i> |
| 10: $\text{best}_{\mathcal{L}} \leftarrow -\infty$ | <i>Best validation loss</i> |
| 11: $\lambda \leftarrow \emptyset$ | |
| 12: $\theta \leftarrow \emptyset$ | |
| 13: $\text{loss} \leftarrow -\infty$ | |
| 14: while stopping criterion not met do | |
| 15: $\lambda \leftarrow \mathcal{O}(\lambda, \text{loss}, \Omega)$ | |
| 16: $\theta' \in \underset{\theta \in \Theta}{\operatorname{argmax}} \mathcal{L}_{\mathbb{D}}(\mathcal{A}_{\theta}^{\lambda}, \mathcal{D}_{\text{train}})$ | <i>Train</i> |
| 17: $\text{loss}_{\text{valid}} \leftarrow \mathcal{L}_{\mathbb{D}}(\mathcal{A}_{\theta'}^{\lambda}, \mathcal{D}_{\text{valid}})$ | |
| 18: if $\text{loss}_{\text{valid}} > \text{best}_{\mathcal{L}}$ then | |
| 19: $\lambda^* \leftarrow \lambda$ | |
| 20: $\theta^* \leftarrow \theta'$ | |
| 21: $\text{best}_{\mathcal{L}} \leftarrow \text{loss}_{\text{valid}}$ | |
| 22: $\text{loss}_{\text{test}} \leftarrow \mathcal{L}_{\mathbb{D}}(\mathcal{A}_{\theta^*}^{\lambda^*}, \mathcal{D}_{\text{test}})$ | |
-

Constrained optimization

Constrained HPO consists of tuning models while being under specific constraints, such as limited computational resources or budget [123, 260]. Unlike unconstrained optimization freely searching over Ω , constrained HPO incorporates these practical limitations directly into the HPO process. This ensures that the resulting ML models achieve high-performance and meet essential real-world requirements. Such approaches help ML to be more applicable and effective in practical deployment.

In practice, some HPs values are incompatible with each other and can result in unfeasible solutions. Thus, modeling constraints helps to improve the coherence of Ω toward technical or real-world requirements.

A constraint can be known and directly modeled within the search space Ω . For example, the parameters of a CNN with stride, padding, and dilation (see section 2.2), result in precise constraints defined by an equation. Other constraints are hidden or black-box [99], i.e. a violation can only be noticed by training or validating a model.

To address constraints, in this work we adopt two approaches. *Penalization* consists in modifying the output of $\mathcal{L}_{\mathbb{D}}(\mathcal{A}_{\theta}^{\lambda}, \mathcal{D}_{\text{valid}})$ so to reflect the value (if measurable) of the violation.

The second method, known as *rejection*, discards solutions violating the constraint. This approach will be applied to hard constraints such as hardware memory limitations. Other approaches can be applied in specific cases [260]. *Repairing* transforms an unfeasible solution into a feasible one, whereas *preserving* adapts the optimizer \mathcal{O} so to sample only valid solutions [260].

In chapters 5 and chapters 6, measurable black-box constraints will be applied to the HPO of SNNs. We consider a set of n real valued constraints $\mathcal{C} \triangleq \{c_1, \dots, c_n\}$. Hence, we can rewrite the definition 7 so to handle these constraints. A constraint c is considered violated if $c \geq 0$:

$$\lambda^* \in \operatorname{argmax}_{\lambda \in \Omega} \mathcal{L}_{\mathbb{D}}^{(1)} \left(\mathcal{A}_{\theta^*}^{\lambda} \mid \theta^* \in \operatorname{argmax}_{\theta \in \Theta} \mathcal{L}_{\mathbb{D}}^{(2)} (\mathcal{A}_{\theta}^{\lambda}, \mathcal{D}_{\text{train}}), \mathcal{D}_{\text{valid}} \right), \text{ s.t. } \forall c \in \mathcal{C}, c < 0 . \quad (2.10)$$

Multi-objective optimization

In section 2.2, works tackling multi-objectives are presented. In real-world applications, optimizing a single criterion is often too simplistic. The usual single-objective HPO maximizes the prediction quality of a ML model. In multi-objective HPO, the optimizer has to achieve a tradeoff among several and often conflicting criteria (accuracy, memory, latency, etc.) [123]. This process typically results in a Pareto front, i.e. a set of non-dominated solutions where no single solution is superior across all objectives. Like constrained optimization, this approach provides greater flexibility and enables the development of models that better meet the diverse requirements of real-world applications.

Considering a set of n objectives written as a risk-like function (definition 2) $\mathcal{F} \triangleq \{f_1, \dots, f_n\}$, where $f_1 = \mathcal{L}_{\mathbb{D}}$ as we always optimize predictions' quality. The focus is on multi-objective HPO and not on the specific training of a ML model. So multi-objectives HPO can be written as:

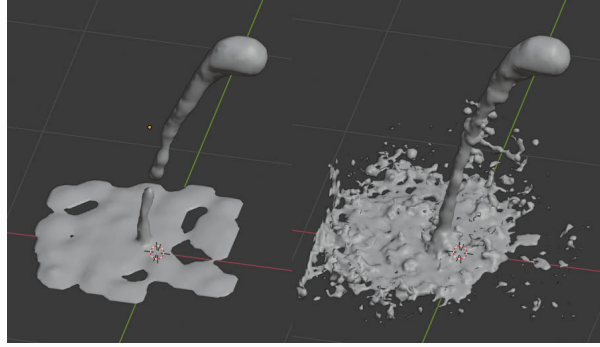
$$\begin{aligned} \lambda^* \in \operatorname{argmax}_{\lambda \in \Omega} (f_1(\mathcal{A}_{\theta^*}^{\lambda}, \mathcal{D}_{\text{valid}}), \dots, f_n(\mathcal{A}_{\theta^*}^{\lambda}, \mathcal{D}_{\text{valid}})), \\ \text{ s.t. } \theta^* \in \operatorname{argmax}_{\theta \in \Theta} \mathcal{L}_{\mathbb{D}}(\mathcal{A}_{\theta}^{\lambda}, \mathcal{D}_{\text{train}}) . \end{aligned} \quad (2.11)$$

Multi-fidelity optimization

Multi-fidelity [20] optimization accelerates the AutoML by considering models with varying degrees of accuracy and fidelity. High-fidelity models are often computationally intensive and time-consuming. Whereas, lower-fidelity models are usually considered less precise but much faster to evaluate. In ANNs and DNNs, fidelity can be explained by the number of epochs or the size of the training set. Then, by allowing the optimization process to have the hand on components or HPs describing fidelities, AutoML incorporates both higher and lower-fidelity models to improve the search strategy. By gathering knowledge from both high- and low-fidelity sources, multi-fidelity optimization can reduce its computational cost, allowing faster convergence toward an optimal solution.

2.1.2 Neural Architecture Search

NAS is a method for automatically building neural network architectures (depicted in figure 2.5) aiming to find an optimal ANN design for a given task [123, 108, 260, 244]. By exploring a search space of possible architectures, NAS algorithms search for the best combi-

Example: Modeling a drop of water.

This non-ML example allows understanding how the fidelity of a simulation impacts its quality. The better the simulation, the higher the computational cost.

nations of layers, topology, and sometimes HPs to maximize $\mathcal{L}_{\mathbb{D}}$. This process often involves reinforcement learning, EA, or gradient-based methods to guide the search efficiently.

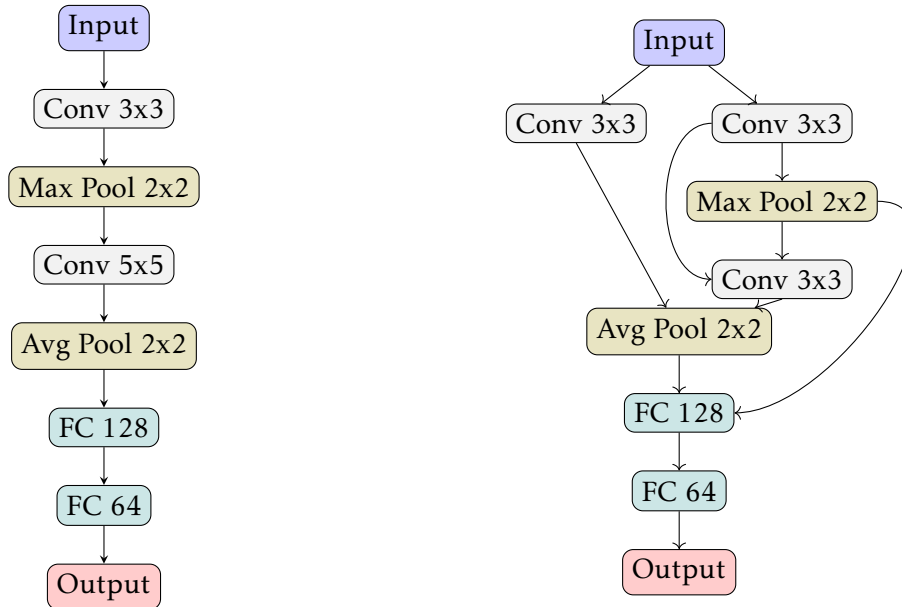


Figure 2.5: Example of different network architectures

An ANN architecture can be modeled by a DAG made of a finite number of nodes. A node represents one operation (convolution, pooling, linear, attention-based, etc.) or a series of operations. So, the search space Ω of a NAS models the set of all possible architectures and topologies among a set of defined layer operations (nodes).

NAS is not addressed in this manuscript. However, it is discussed briefly, as only a limited number of studies have applied NAS to SNNs. Additionally, our contributions could also be applied to NAS.

2.1.3 Advantages, common issues and remarks

Beyond simplifying and improving the design of ML solutions, AutoML allows fair comparisons between models [123, 290]. Because HPs are bounded to specific tasks and problems, comparing performances of a model with default HPs to a model with manually or automatically tuned HPs appears arbitrary [16, 123, 16, 260, 20, 290, 20, 244]. AutoML also

helps to improve ML interpretability by heavily exploring numerous and counter-intuitive configurations and architectures.

However, AutoML, HPO and NAS suffer from multiple drawbacks. While reducing arbitrary user decisions, human-in-the-loop interactions are still necessary within different steps of the process [123, 16, 20, 290, 120, 244]:

- **Algorithm selection:** Selecting the optimizer requires prior knowledge about the problem and is prone to the No Free Lunch theorem. Additionally, the choice should be informed by available resources, budget, model's computational complexity, convergence rate, etc.
- **Search space:** Identifying relevant HPs for given metrics, their bounds, or nodes operations in NAS, is left to the discretion of the user. Wrongly adding an unnecessary HP to a search space exponentially increases its geometrical complexity due to the curse of dimensionality. Moreover, search spaces are complex regarding the heterogeneity of the HPs' nature (discrete, continuous, categorical, etc.).
- **Benchmarking:** Benchmarking AutoML algorithms is a challenging task. Notably, because of the wide variety and computational complexity of the problems. Some initiative exists, such as the COCO [103] benchmark for continuous global optimization. HPOBench [59] could also be considered for specific HPO tasks. Thus, performances of optimizers cannot be ensured for all ML problems. The benchmarking efforts aim to evaluate optimizers on the most common problems, helping to infer what could be the results when an optimizer is applied to similar tasks. Defining comparability metrics and properties is necessary. A tradeoff between convergence speed and solution quality is necessary when dealing with a restricted budget. *Should we prefer heuristics or metaheuristics to algorithms with proved and bounded convergence?*
- **Replicability and randomness:** Results of most ML models are subject to stochasticity. Even optimizers have their own HPs and so are sensitive to certain families of problems. AutoML and ML involve many layers of software and hardware: linear algebra, programming language, operating systems, drivers, data storage, etc. Because of these, exactly replicating results appears to be nearly impossible.
- **Overtuning, overfitting and generalization:** The almost ideal approach would be a systematic use of nested cross validation, which is in practice untractable to expensive problems. Even the train-validation-test splits are not always used, particularly with manual tuning and within the SNNs community [294].
- **Black-box optimization:** Usually AutoML is applied to black-box objective function of ML model where information is almost inaccessible, e.g. no derivatives.
- **Scalability:** AutoML is subject to the curse of dimensionality, making optimizers more or less efficient according to the problem. Scalability also concerns parallelization, which is discussed in chapter 3. For instance, GS scales poorly in dimensions but is embarrassingly parallel, whereas BO scales better but is trickier to parallelize.
- **Stopping criterion and budget:** Defining a sufficient budget or when to stop the optimization algorithm is challenging [20]. For algorithm comparisons, a fixed number of evaluations is usually set, as it is independent of any hardware platform. However, it can appear unfair when algorithms have to do costly computations to identify a new candidate to evaluate (e.g. GP-based BO). Another approach would be to set a minimum required risk function value to reach before stopping, but it involves the knowledge of a

feasible performance. Convergence or stagnation-based stopping involves knowledge of the problem and of the algorithm, so to prevent stopping the optimization too early. For instance, because of an intensive exploration, improvement of the current best-known solution cannot be ensured before exploiting the gathered information. In this thesis, our main stopping criterion is time because the resources are limited, shared among multiple users, and expensive in terms of electricity consumption. The HPO algorithm has to be as fast as possible to ensure minimum performances.

2.2 Hyperparameter Optimization of Spiking Neural Networks

While HPO is a well-established approach in ML where it plays a growing role in the design of efficient models, its application to SNNs introduces unique challenges and opportunities. In the following lines, we describe how usual HPO algorithms are applied indiscriminately to both ANN and SNNs. Thanks to the following state-of-the-art, in chapters 5 and 6 we introduce how the adaptation of HPO strategies improves the process. Indeed, the temporal dynamics and event-driven nature of SNNs demand close attention for tuning HPs. Transitioning HPO methodologies from traditional ML to SNNs involves addressing these complexities to fully harness the potential of neuromorphic computing.

The objective of the following sections is to understand the impact of all HPs at the component scale. Their behavior on the network performances are discussed in chapters 5 and 6.

2.2.1 Common hyperparameters

One of the first steps of HPO described in section 2.1, is the HPs selection and search space design. We saw that AutoML aims at limiting the human-in-the-loop interactions. Hence, the few steps in which user knowledge is needed are crucial [123, 290, 16, 260]. The *garbage-in, garbage out* concept, usually linked to data science, can be transposed to HPO as *optimizing in a poorly defined search space results in worthless optimization*. Then, selecting relevant HPs and defining their domain in such a way that reachable solutions range from intuitive to counter-intuitive, or from unfeasible to feasible, is challenging.

Manual tuning and single HP studies are commonplace in the ANNs community [249] as well as in the SNNs community. Because neuromorphic computing ranges from neuroscience to electronic [238], HPs are often set by default according to biology [88, 298, 130, 114]. However, HPs combinations cannot be generalized to all ML tasks [125] because of the No Free Lunch theorem. Conversely to ANNs, SNNs are known to be very sensitive to their HPs [28, 102, 195]. The complexity of the neuron model and its numerous HPs impact the local and global dynamics of the networks. This underlies the necessity to better explore HPO of SNNs.

HPO of SNNs is sparsely explored in the literature [195], and hardly ever addressed from the AutoML perspective. Parameter (definition 3) and HP (definition 6) are sometimes used interchangeably [293]. The frontier between parameters and HP becomes blurry in some scenarios. For example, in neuroevolution [237], parameters, some HPs and the architecture are jointly optimized via an EA strategy. Hence, some parameters of the EA become the HPs [206, 203] to be optimized, e.g. population size or mutation rate. We consider the synaptic weights \mathbf{W} of a SNN to be its parameters by default. Sometimes other aspects can be learned. This is the case for the synaptic delays in SLAYER [247] or the neuron leakage in PLIF [70].

Some works might employ ambiguous terms and sometimes deviate from the usual HPO standards described in section 2.1. Names of AutoML frameworks, such as NNI [180] or Optuna [3], are sometimes wrongly employed as the optimizer name (see chapter 3). Describing the hardware used, experiment duration, or even computational cost of training is not widely spread. To improve reproducibility and peer review, the community could greatly benefit from systematic information on the technical details. In the HPO standpoint, such information is of the utmost importance, as it allows to better estimate room for maneuver. Indeed, the choice of the optimizer, the allocated resources, or the budget is significantly different if a SNN takes a few minutes or hours to train.

We propose to group SNNs HPs into 5 groups, used in tables 2.1, 2.2 and 2.3:

- G1 : Neuron model
- G2 : Learning rule
- G3 : Architecture
- G4 : Encoding and decoding
- G5 : Resource, system management and regularization HPs

Neuron model

The neuron is the main component of a SNN. It describes how a network integrates asynchronous spikes or action potentials through time. Because neuromorphic computing covers numerous domains, a wide range of neuron models is available depending on the task, bio-plausibility, or hardware [89, 130, 114, 42, 225, 246, 257].

In this work, we focus on the LIF neuron [90, 33, 193]:

$$\begin{cases} \tau_{\text{leak}} \frac{dV}{dt} = -(V - V_{\text{rest}}) + RI & , \text{ if } V < V_{\text{th}} \\ V = V_{\text{reset}} & , \text{ else} \end{cases} . \quad (2.12)$$

The spiking output of this neuron at a time t is modeled by

$$s[t] \triangleq \begin{cases} 1 & , \text{ if } V[t] > V_{\text{th}} \\ 0 & , \text{ else} \end{cases} , \quad (2.13)$$

or sometimes [193] written using the Heaviside step function H :

$$s \triangleq H(V - V_{\text{th}}) . \quad (2.14)$$

Then, equation 2.12 can also be written as:

$$\frac{dV}{dt} \triangleq -\frac{1}{\tau_{\text{leak}}} ((V - V_{\text{rest}}) + RI) + s(V_{\text{reset}} - V_{\text{th}}) \quad (2.15)$$

Parameters are, V the membrane potential, I the synaptic current, t the time, τ_{leak} membrane decay time constant, R membrane resistance, V_{th} membrane threshold, V_{rest} membrane resting potential and V_{reset} the membrane reset potential.

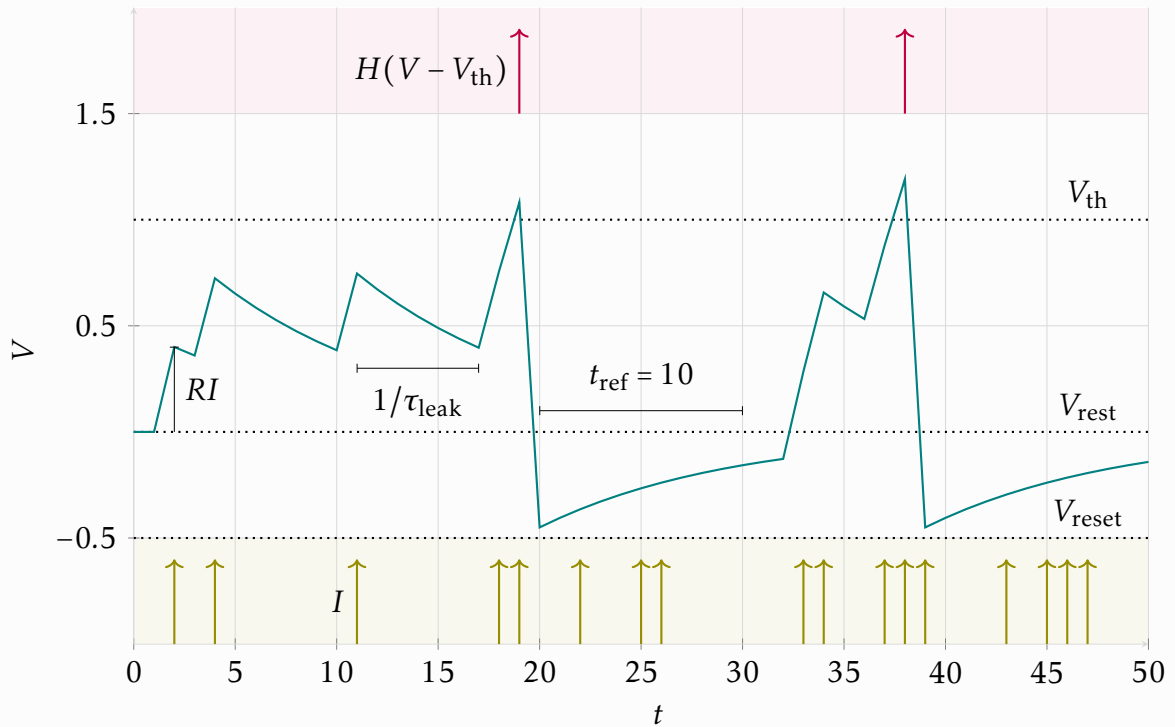
Equations 2.12 and 2.13 describe the most common model of a LIF. In the literature, other notations describe similar dynamics. For instance in [51], the LIF equation uses architecture-specific HPs (e.g, E_{exc} and E_{inh}). Additionally, the way a differential equation is

implemented can differ from a simulator to another. In Brian2 [252] the equation is solved by using numerical methods such as Euler-Maruyama or Runge-Kutta methods [102]. This variability in the implementations may cause reproducibility issues. Furthermore, owing to the sensitivity of the neuron HPs, results can diverge significantly when transitioning from one simulator to another or when transitioning from simulation to hardware implementations. Hence, it is important to have precise documentation about neurons, but also about other SNN components (e.g. learning rules). A challenge when designing the search space for an HPO, is to define the boundaries of these HPs. Then, it is essential to study the behaviors of the selected simulator or hardware to define these bounds.

Nonetheless, we can find common HPs to the different implementations of a LIF. The two most important ones appear to be the threshold V_{th} and the leakage time constant τ_{leak} . If the implementation is more biologically accurate, we can include a reset V_{reset} or a resting V_{rest} potential. These HPs can also be considered as it modifies the dynamic. It also introduces some constraints, as $V_{reset} < V_{th}$ and $V_{rest} < V_{th}$, which complicates the search space definition and HPO process. The ability of LIF to filter noise and act like a filter is one of its main advantages [33]. HPO becomes an interesting approach to adapt LIF HPs to a given noisy problem.

We can extend the dynamic of equation 2.12 by adding a refractory period t_{ref} , i.e. a period during which a neuron that has recently spiked becomes inactive. This HP is often expressed in *milliseconds* or *time steps* depending on the hardware or simulator.

Example: Membrane dynamic of a LIF



The above example allows understanding all HPs of a LIF at the neuron scale. The impacts of HPs at the network scale is discussed in chapters 5 and 6.

To tackle the dead neuron problem, one can also extend equation 2.12 with a certain form of homeostasis called adaptive threshold [257, 51]. This mechanism is modeled by two HPs:

θ_{\oplus} and τ_{θ} . Then, the dynamic of a single neuron becomes:

$$\left. \begin{aligned} \tau \frac{dV}{dt} &= -(V - V_{\text{rest}}) + RI \\ \frac{d\theta}{dt} &= -\frac{\theta}{\tau_{\theta}} \end{aligned} \right\} \text{if } V < V_{\text{th}} + \theta \quad (2.16)$$

$$\left. \begin{aligned} V &= V_{\text{reset}} \\ \theta &= \theta + \theta_{\oplus} \end{aligned} \right\} \text{else}$$

One drawback of adding refractory period or threshold adaptation mechanisms is the increase in memory and computational complexity [25]. It also complicates the search in a HPO scenario, as it increases the dimensionality of the search space. Their implementation might also differ according to the software simulator or hardware platform.

Thanks to SNNs properties, it is theoretically possible to individually tune each neuron HPs. As well as for plasticity rules, which could be locally defined for each synapse. However, it is practically infeasible to tune each neuron and synapse HPs due to the combinatorial explosion of possible solutions. An alternative could be a high dimensional HPO considering HPs by groups of neurons or layer by layer (see chapter 6).

Example: Adaptive LIF

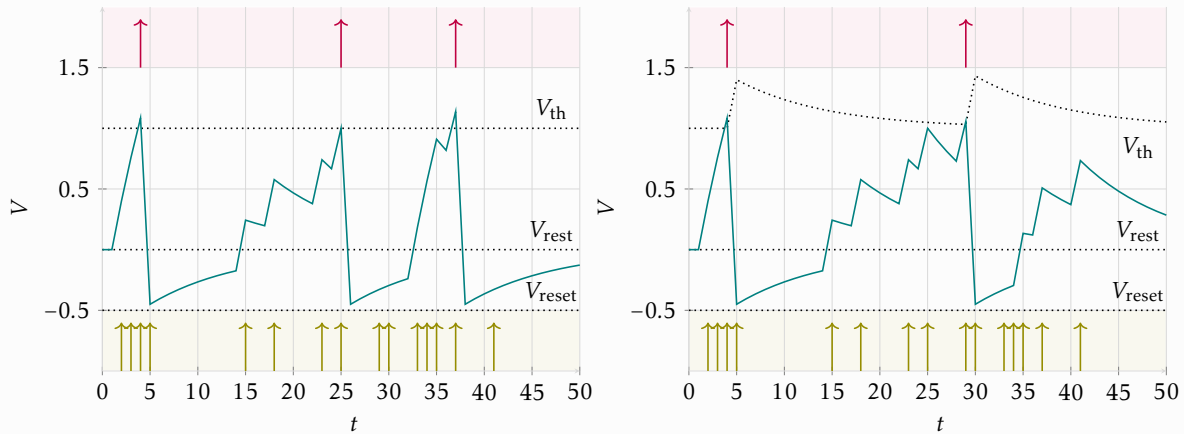


Figure 2.6: LIF (left) and Adaptive LIF (right).

Learning rules

In this manuscript, we consider two learning rules: the standard STDP [90, 275], and surrogate gradient backpropagation [193, 247].

The unsupervised STDP equations described in chapter 1 equation 1.7, are untractable as it implies keeping in memory all spike timings. To approximate theoretical STDP, a trick is to use *traces* of pre- and post-synaptic activity ($a_{\text{pre}}, a_{\text{post}}$) for each neuron. A trace is a time-dependent local memory of the previous spiking activity of a neuron. Many equations and versions of STDP are available.

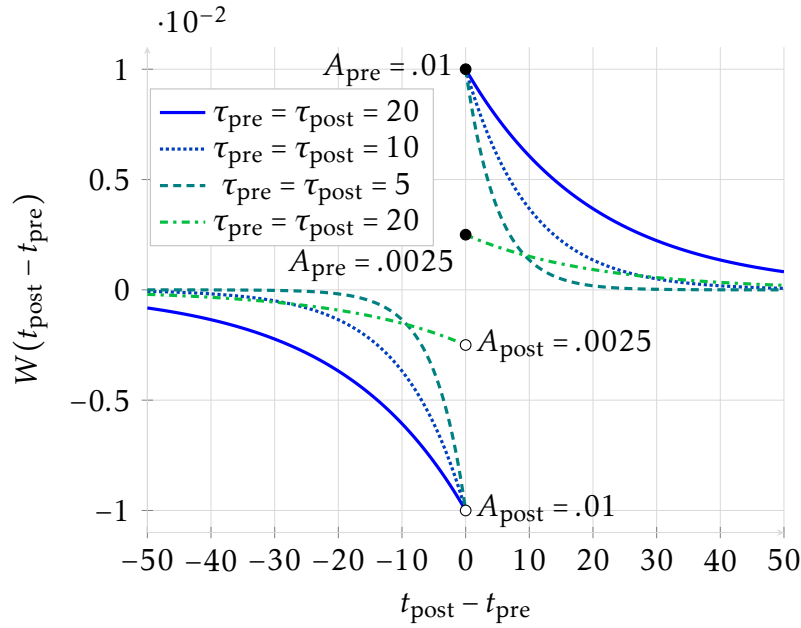


Figure 2.7: Impact of HPs on STDP.

The STDP learning rule (Bindsnet) can be written as:

$$\begin{aligned}
 \tau_{\text{pre}} \frac{da_{\text{pre}}}{dt} &= -a_{\text{pre}} \\
 \tau_{\text{post}} \frac{da_{\text{post}}}{dt} &= -a_{\text{post}} \\
 \left. \begin{aligned} a_{\text{pre}} &= a_{\text{pre}} + A_{\text{pre}} \\ w &= w - \lambda_{\text{pre}} a_{\text{post}} \end{aligned} \right\} &\text{if a pre-synaptic spike occurs} \\
 \left. \begin{aligned} a_{\text{post}} &= a_{\text{post}} + A_{\text{post}} \\ w &= w + \lambda_{\text{post}} a_{\text{pre}} \end{aligned} \right\} &\text{if a post-synaptic spike occurs}
 \end{aligned} \tag{2.17}$$

This learning rule introduces 6 HPs. The scale of the traces ($A_{\text{pre}}, A_{\text{post}}$), i.e. ratios of long-term potentiation. It also introduces, the time constants describing the decay of the neuron traces τ_{pre} and τ_{post} , and $(\lambda_{\text{pre}}, \lambda_{\text{post}})$ the learning rates describing the strength of a weight variation when a pre- or post-synaptic spike occurs. With this learning rule, each neuron must implement an additional differential equation describing the trace of previous spikes. This unsupervised learning rule is often combined with a decoder, e.g. a supervised learning rule or another supervised ML approach. STDP is depicted in figure 2.7 where a weight variation relies on the timing between t_{pre} and t_{post} .

STDP based learning is easily implementable on neuromorphic hardware [134], while gradient-based training is more challenging as it involves non-local memory [304, 38].

In chapter 1 we defined BP as an offline alternative to train SNNs. However, BP cannot be directly and easily applied as the spiking activity $s_j(t)$ at a time t of a neuron j is defined as a Dirac-delta function [193]: $s_j(t) = \sum_{t' \in C_j} \delta(t - t')$, where C_j are the spikes timing of the neuron j . This function is constant everywhere, except at 0 where it is equal to ∞ . Hence, the gradient of the spiking function is uninformative and unusable for BP. Therefore, a differentiable surrogate function is used to instead approximate the spiking function.

In this manuscript, we use two surrogate gradient-based approaches. The first one is known as SLAYER [247]. It was initially applied to SRM neurons. SLAYER2 available within

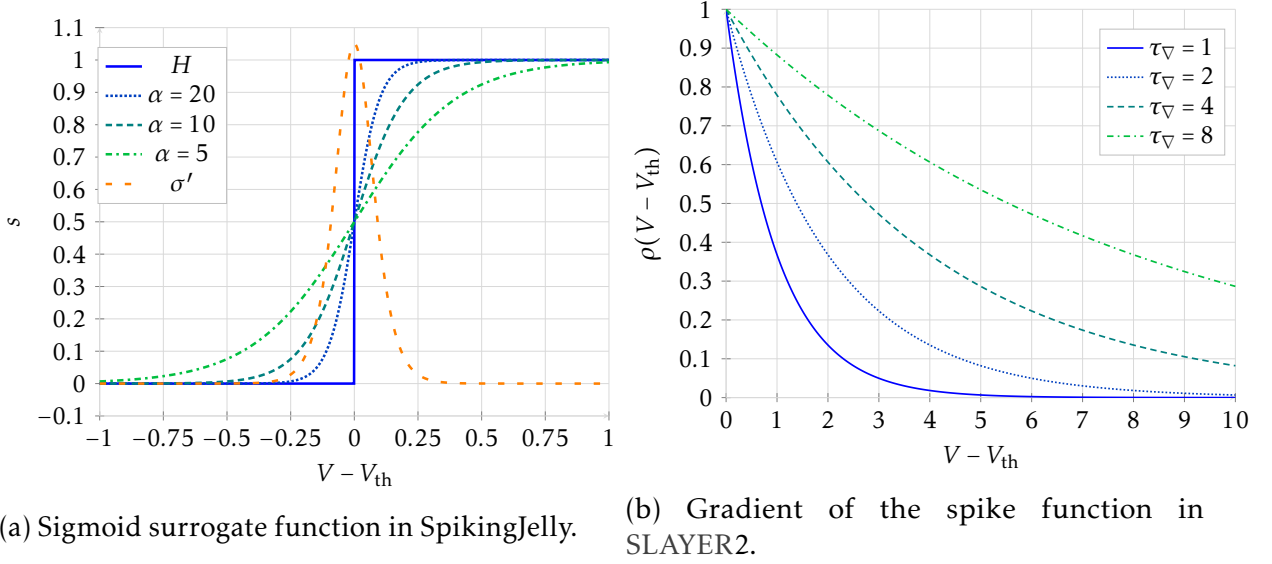


Figure 2.9: Impact of HPs in different surrogate gradient functions.

the LAVA-DL¹ simulator, allows backpropagation with adaptive LIF neurons.

To apply BP to SNNs, it is necessary to define the loss and its partial derivative $\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(l)}}$ w.r.t. the weights $\mathbf{W}^{(l)}$ at a layer l . For a classification task, output neurons are associated with a label and have to output a certain rate or number of spikes at a time t within a time interval $t \in T_{int}$:

$$e^{out}(t) = \int_{T_{int}} s^{out}(t') - \hat{s}(t') dt' . \quad (2.18)$$

With $e^{out}(t)$ the output error at the last layer at a time t . The target spike train \hat{s} is in practice translated into two spike rates: one for the neuron representing the *right* label v_T (True), and another v_F (False) describing a spiking rate for the other output neurons. Actually, equation 2.18 is simply the MSE loss between the output spiking rate of output neurons and the required output rates.

SLAYER adopts a strategy consisting in modeling the *transition probability* between *non-spiking to spiking* states of a neuron. This is done by using a surrogate function at a given time t :

$$\rho(t) = G_{\nabla} \cdot \exp\left(\frac{\min(V - V_{th}, 0)}{\tau_{\nabla}}\right) , \quad (2.19)$$

where G_{∇} scales the gradient by a constant to modify the gradient flow across layers, and tackle vanishing or exploding gradients. The HP τ_{∇} describes the relaxation of the spike function. In SpikingJelly [72], it can be the sigmoid function $\sigma(x) \triangleq \sigma(V - V_{th}) = \frac{1}{1 + \exp(-\alpha_{\nabla}(V - V_{th}))}$ with α_{∇} a HP defining the quality of the approximation. The derivative is given by $\frac{\partial \sigma}{\partial x} = \sigma(x)(1 - \sigma(x))$. Other functions can also be used, such as the *arctangent* [70]. These surrogates and their HPs are illustrated in figure 2.9.

In SLAYER 2, the gradients of the error according to the weights $\mathbf{W}^{(l)}$ at layer l are accumulated through time and expressed by:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(l)}} = \int_0^T \delta^{(l+1)}(t) (\mathbf{o}^{(l)}(t))^{\top} dt , \quad (2.20)$$

where $\mathbf{o}^{(l)}(t)$ is the signal output of layer l . The surrogate is used in the computation of the

¹<https://lava-nc.org/dl.html>

errors:

$$\delta^{(l)}(t) = \rho^{(l)}(t) \mathbf{W}^{(l+1)} \delta^{(l+1)} , \quad (2.21)$$

with $\delta^{\text{out}} \triangleq \frac{\partial \mathcal{L}}{\partial \mathbf{o}^{\text{out}}}$ (e.g. the MSE on the output spiking rates and targeted rates (ν_T, ν_F)).

Now, the gradient is informative and can be backpropagated with usual optimizers such as ADAM [147]. Thus, we therefore return to usual HPs from the gradient descent with the learning rate λ_{∇} or momentum β_{∇} .

To reduce the number of HPs in the neuron model, a few works aim at incorporating some HPs into the gradient computations. This is the case for the PLIF [70] neuron, where τ_{leak} becomes a learnable parameter:

$$s_t = H(\psi_t - V_{\text{th}}) , \quad V_t = \psi_t(1 - S_t) + V_{\text{reset}} S_t , \quad \psi_t = V_{t-1} + \frac{1}{\tau_{\text{leak}}} (-(V_{t-1} - V_{\text{reset}}) + I_t) , \quad (2.22)$$

where ψ_t , S_t , I_t , and V_t are values at a discretized time-step t . The variable V_t is the membrane potential, I_t is the input current, and s_t is the output spike described by the Heaviside step function H . Because τ_{leak} is learned, only an HP τ_{init} has to be set by the user to initialize τ_{leak} .

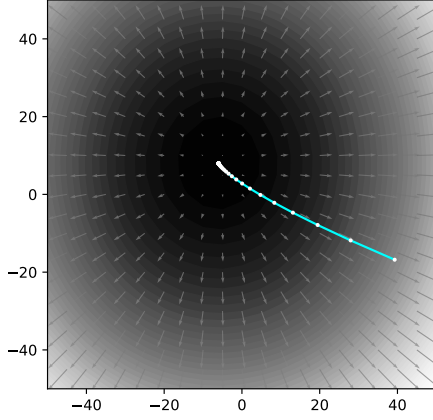
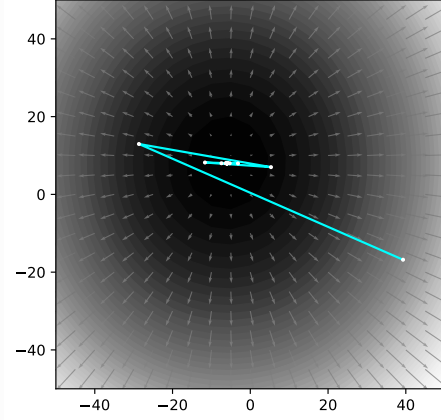
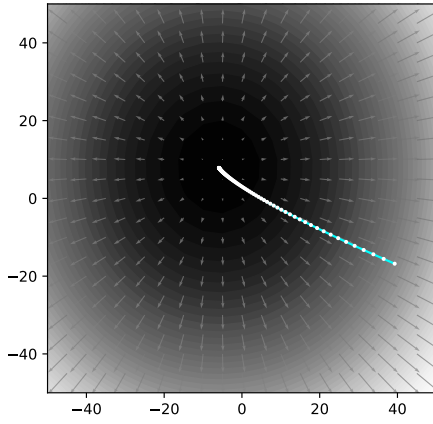
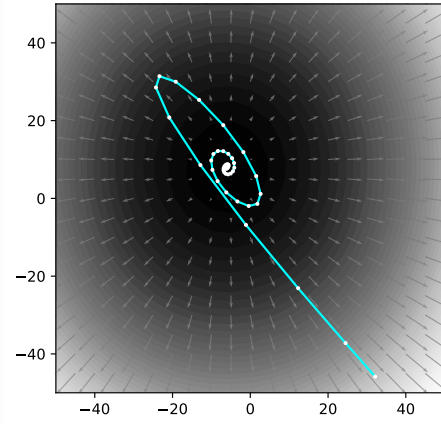
Architecture related hyperparameters

SNNs introduce two types of neurons. The inhibitory neurons emit a negative post-synaptic potential, often modeled by a negative weight between the inhibitory neuron and others. Conversely, excitatory neurons produce a positive post-synaptic potential [257, 51]. These two types of neurons are frequently found in reservoir computing and RNN [305, 218]. They allow modeling more complex dynamics and a higher biological plausibility. In this type of network, the ratio between excitatory and inhibitory neurons can be optimized.

Furthermore, this dynamic becomes essential in most SNNs trained by STDP. It forms the basics to introduce competition between neurons, such as WTA or lateral inhibition mechanisms. It enables certain neurons to only respond to distinct spiking patterns. The WTA is a broader strategy often used to perform the classification task at the last layer of a SNN by allowing only one neuron to spike at a time. The strength of the excitation or inhibition is modeled by fixed weights between layers. These strengths can be optimized within HPO. In a Diehl & Cook like SOM [51] the strength of excitation and inhibition can model whether a soft lateral inhibition [106] or a WTA.

ANNs and SNNs share similar architectures, and so common HPs. The most common one is the feed-forward fully connected network made of $l \in \mathbb{N}^*$ layers, where a layer $1 \leq i \leq l$ is defined by a number of neurons $n^{(i)}$. Here, $n^{(i)}$ can be optimized, and no specific constraints are needed except $n^{(i)} \in \mathbb{N}^*$. The lower and upper bounds can be defined considering memory, computational, or search space complexities [259].

Concerning CNNs, two approaches exist. On the one hand, there is the usual approach with weight sharing during the training, involving non-local shared memory. On the other hand, lateral inhibition can be used in a CSNNs without weight sharing [231, 88]. In a local memory CSNNs, each filter is made of a spiking SOM [231]. It also requires additional inhibitory connections between filters and no weight sharing. This approach greatly increases the memory footprint of a local memory CSNN compared to the usual CNN. Both methods, local-memory CSNN and non-local-memory CSNN can be considered, even on neuromorphic hardware [116, 292, 137]. The two approaches have common HPs, such as the number of filters, the kernel size ($k^{(i)}$), dilation ($d^{(i)}$), stride ($s^{(i)}$), or padding ($p^{(i)}$) for a layer i . While tuning the number of filters is straightforward, $k^{(i)}$, $d^{(i)}$, $s^{(i)}$, and $p^{(i)}$ presents a significant challenge as these HPs are under complex constraints. These involve constrained dynamic

Example: 2D gradient descentFigure 2.10: $\lambda_{\nabla} = 0.1$ Figure 2.11: $\lambda_{\nabla} = 0.6$ Figure 2.12: $\lambda_{\nabla} = 0.025$ Figure 2.13: $\lambda_{\nabla} = 0.1, \beta_{\nabla} = 0.8$

Examples of different learning rates applied to gradient descent on the function $f(x_1, x_2) = \frac{5}{4}(x_1 + 6)^2 + (x_2 - 8)^2$ (from [242]). The arrows show the gradient vector field scaled by its norm. In (2.10), λ_{∇} can be considered acceptable face to the convergence. In (2.11), λ_{∇} is too high as the gradient descent *overshoot* the optimum. In (2.12), λ_{∇} is too low as it converges too slowly. The example (2.13) illustrates gradient descent accelerated by the momentum.

search spaces [259]. Indeed, for each input dimension I of each convolutional layer i , the constraint is:

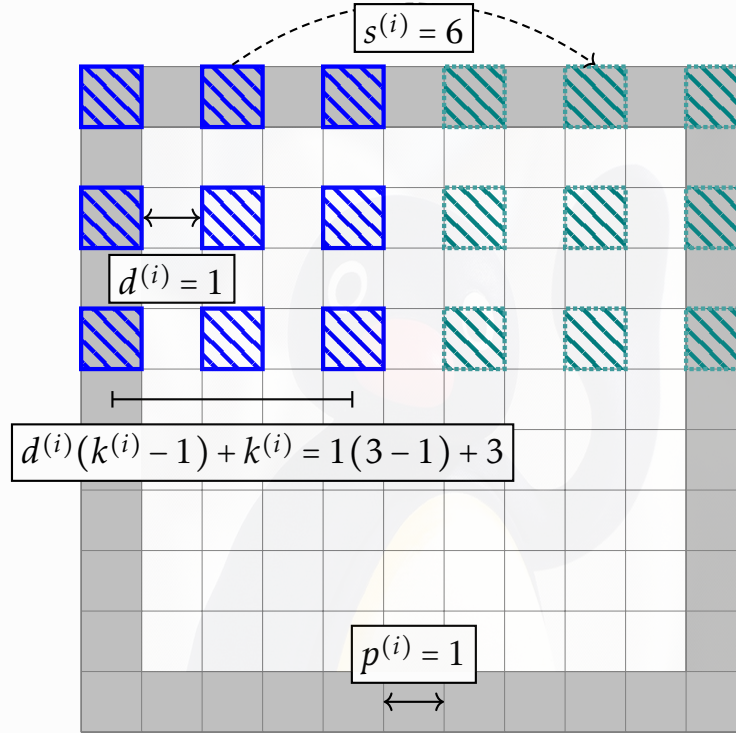
$$\left\lfloor \frac{I + 2p^{(i)} - d^{(i)}(k^{(i)} - 1) - 1}{s^{(i)}} + 1 \right\rfloor \geq 1. \quad (2.23)$$

Hence, $s^{(i)}$, $d^{(i)}$ and $p^{(i)}$ are often fixed and only $k^{(i)}$ is optimized by restraining its bounds to only feasible solutions [88], or by selecting different convolutional architectures [205].

Encoders and decoders

The encoding of input data is a hot topic within the SNNs community. The usage of analog DVS datasets, rate- or latency-encoded numerical datasets is subject to debates [294, 238, 235, 26]. In our HPO standpoint a question remains: *Is the choice of an encoder or preprocessing could be simply solved by HPO which would select the best encoder for a given problem* [213]?

Usually, the MNIST dataset [153] and other numeric datasets, are converted into spikes

Example: CNN filter

Example of a $k^{(i)} = (3 \times 3)$ convolution filter sliding to the right with a step of $s^{(i)} = 6$.

using a Poisson distribution [110]. Other methods exist, such as Time-To-First-Spike, Phase or Rank coding [11]. These methods have a common HP T describing the sampling time, i.e. the maximum time duration a data is exposed to the network or time needed to encode a numerical value into spikes.

In [101], authors investigate the impact of each encoding methods and sampling time on a SOM accuracy, latency, and synaptic operations. They studied the impact of pruning, quantization, noise, synaptic noise, and synaptic fault for four encoding schemes within neuromorphic systems. They trained SNNs on the MNIST and FashionMNIST datasets. Surprisingly, TTFS coding appears to have the best performances, but it is less robust to noise.

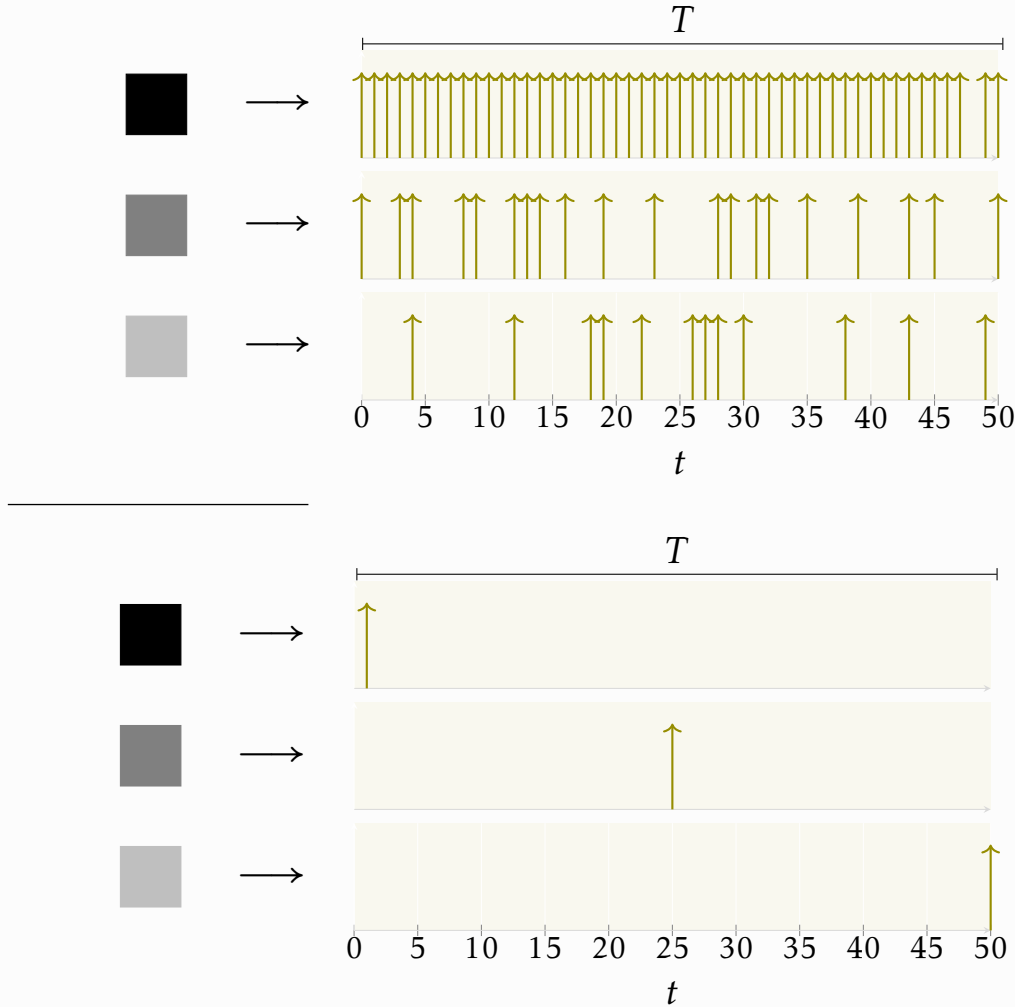
For DvsGesture [10], no encoding is needed. However, due to the high temporal resolutions, a preprocessing is often required before passing data to the network. In [35], authors gathered spikes at a 1 microsecond time resolution, resulting in $T = 150$ timesteps also called frames.

Because the outputs of a SNN are spikes, a decoder is necessary to translate spikes into a workable value. Particularly with SNNs trained by unsupervised STDP. In this case, a supervised decoder needs to be trained.

We focus on classification tasks, but different decoders might be used according to the application [235]. Decoding outputs is challenging as the selected decoder might not be the best one for a given problem or network [150]. Some decoders are bound to the architecture of the SNN. Max spikes or Average spikes work well when the output layer is under a lateral inhibition or WTA mechanism [51]. In [106], a comparison between different approaches is made. It includes unusual ones such as n -grams. Other ML-based methods can be used, but are untractable to neuromorphic hardware. For instance, softmax regression [305] or SVM [68]. In [235], authors showed the impacts and the relation between encoders and

decoders on the quality of SNNs performances for different types of tasks. Concerning gradient-based SNNs applied to a classification task, the training process usually assigns an output neuron to a target class. In SLAYER [247], different loss functions are available to decode outputs such as max spikes, spike rate, or spike timing.

Example: Encoding pixels



Above is an example of Poisson encoding, and below is an example of TTFS encoding of three pixels.

Resource, system management and regularization hyperparameters

In this section, we discuss other HPs that cannot be classified within the previous categories or that are less studied in the SNNs community. We can consider generic HPs grouped within the training pipeline of ANNs and SNNs, e.g. number of epochs or batch size. For STDP based SNNs implementing batch computation is harder due to the online training. Nonetheless, some specific rules or adaptations exist, showing effects on the SNN performances [233, 55]. While for SNNs trained by surrogate gradient, it is straightforward and similar to usual ANN [29].

Other HPs are linked to specific methods, such as weight normalization [51, 194], dropout rate [194], regularization terms [194, 200] or pruning [224, 60]. These HPs can leverage the

homogeneity of the spiking activity within the network, akin to mechanisms of homeostasis. HPs from these methods can also be optimized within HPO because it has an impact on the performances, the spiking activity, and overfitting. Many other HPs we did not mention could also be tuned. These can be related to the hardware [203], quantization, parallelization, multi-fidelity, etc.

Example: Pruning and dropout

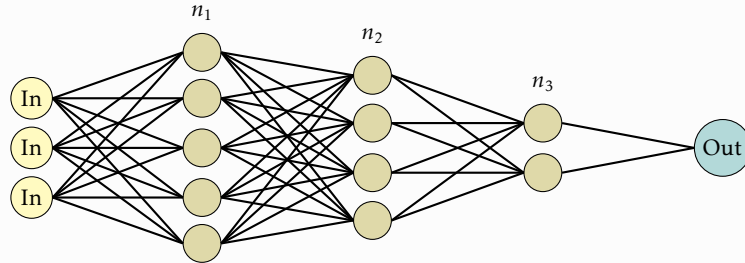


Figure 2.14: Feed Forward Architecture

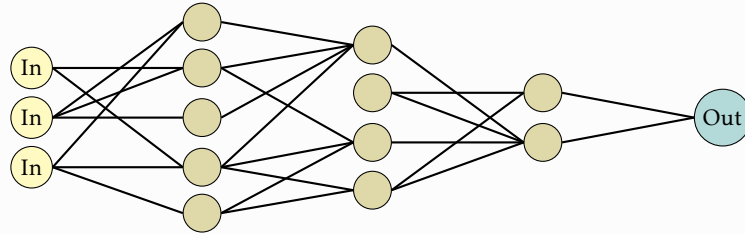


Figure 2.15: Pruning

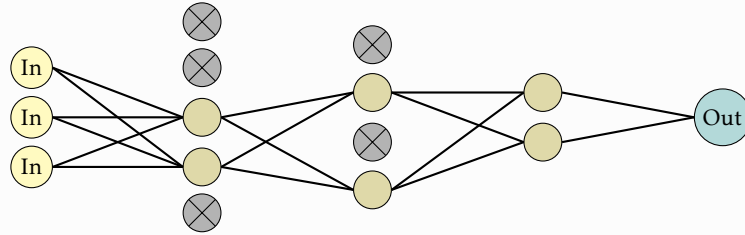


Figure 2.16: Neurons dropout

Example of pruning and dropout applied to a feed-forward network. Here, n_i corresponds to the number of neurons per hidden layer. Pruning is generally described as discarding synapses with a weight lower than a certain value. Dropout can be expressed as a probability to discard a neuron.

2.3 Related works

2.3.1 STDP-based SNNs

In [31], a SMBO approach was applied to a spiking Reservoir made of SRM neurons and trained with STDP. The dataset is made of artificially generated time series. Seven HPs were optimized. Some are linked to STDP, such as both traces of time constant (τ_{pre} , τ_{post}) and ratios of long-term potentiation (A_{pre} , A_{post}). One HP is linked to the neuron model. Three other HPs describing network topology (initial inhibitory and excitatory strength) are also optimized. Experiments indicate that the initial value of inhibitory weights, and the HP of the neuron model have a greater impact on the accuracy than the learning rates of STDP.

However, no analysis of the spiking activity is made to explain these results. The authors suggest a more advanced study and the use of a better BO technique.

EAs are also applied to such problems. In [217], the Diehl & Cook SOM architecture [51] trained using STDP in a federated learning environment is used to classify MNIST and FashionMNIST. The authors applied 5 different metaheuristics, Cuckoo Search, Whale Optimization, Polar Bears, Grasshopper Optimization and Salp Swarm. The best accuracies found were respectively 89% and 82%. Six HPs were optimized: number of neurons, excitatory strength, inhibitory strength, neuron threshold, threshold potential decay, and μ a STDP coefficient specific to their work. Similarly, Guo et al. optimized the same architecture using a GA. To reduce the network latency and maximize its performances, the fitness function is made of the accuracy and the sum of time constants. They optimized θ_{\oplus} for the threshold adaptation, both STDP learning rates, and both pre- and post-synaptic traces time constants. They achieve 86.54% accuracy on Poisson-encoded MNIST. Their work highlights that time constants with high values give higher accuracies but slower convergence. The authors also studied the impact of the approximation made by numerical methods (Euler and third-order Runge-Kutta) on the accuracy and FPGA implementation. They pointed out the high sensitivity of SNNs to their HPs and the computational complexity of simulated SNNs. Their training took up to 11 hours for a simple two layers SOM architecture.

In [241], the authors did not perform an HPO, but have empirically demonstrated the relation between accuracy and the number of output neurons of a RBM trained with STDP. The higher the number of neurons, the better the accuracy. They achieved a maximum of 89.4% of accuracy on MNIST.

STDP-trained SNNs with Locust Lobula Giant Movement Detector (LGMD) model was also optimized in [230]. They used RS, three BO techniques, DE and Self-adaptive DE. Authors were able to optimize up to 18 different HPs of 4 similar architectures implemented using Brian2 [252]. They applied these models to the detection of looming and nonlooming stimuli on an in-situ DVS recorded UAV dataset.

Classifying DvsGesture using STDP is a tough task, notably due to the spatio-temporal features and to the heterogeneous distribution of spikes among classes. In [88], a CSNN combined with a reservoir was used. Outputs are decoded with a logistic regression. The authors compared different networks with hand-tuned HPs linked to the architecture and topology. HPs of STDP, neuron model, and homeostasis are fixed. They achieved a 65% of accuracy with a network of 3.1764 millions of parameters. DvsGesture was also classified in [129]. The authors applied a combination of unsupervised STDP and supervised Tempotron on a SOM like architecture. The model achieved 60.37% of accuracy on an augmented dataset. The authors manually studied the impact of the inhibition strength (see D&H architecture [51, 106]) and the number of neurons. It appears that training with augmented data and a large number of neurons gives the best accuracy.

The work in [262] is more neuroscience-oriented. The authors optimized 6 HPs using cross validation and BO applied on classification of electroencephalography (EEG) measurements. The architecture is made of a reservoir followed by a deSNN. The deSNN encodes spatio-temporal spiking patterns from the reservoir. The output generates a static vector representation, which is passed to a supervised decoder, here a SVM. Most of the HPs (4) are linked to a complex ML-based encoding method, and the two other ones describe the deSNN.

Table 2.1 sums up some of HPO techniques applied to STDP-based learning. HPO algorithms range from Manual (Man.) tuning to BO. HPs are grouped according to their SNNs components. Each group contains the number of HPs tuned during the optimization or manually studied. One can notice that HPs linked to the neuron model (G1) are not often optimized. Usually, the architecture's HPs (G3) are tuned. The learning rates (G2) are sometimes optimized. In the G4 group, i.e. encoding and decoding, the sampling

time T is frequently considered as well as the decoder [106]. It is important to notice that the optimization is typically mono-objective and maximizes the classification accuracy. Sensitivity (Sens), Precision (Prec) and Specificity (Spec) are considered in [230], but not within a multi-objective HPO. The total number of studied HPs is described in the column "#". A question mark '?' indicates the absence or ambiguity of information.

2.3.2 Gradient-based SNNs

As in usual gradient-trained ANNs, HPs are often manually studied one-by-one, trying to infer general behaviors. In [249], some ANNs behaviors are studied for several architectures and datasets. For example, the authors discussed various subjects independently, such as underfitting, overfitting, batch size, learning rate, momentum, and weight decay. Similar works also exist within the SNNs community. In [295], the authors manually reviewed diverse SNN components on multiple architectures and datasets such as SHD, SSC, NMNIST, DvsGesture, MNIST and CIFAR10. A close attention is dedicated to the LIF neuron by performing an ablation study on its leakage, recurrent connections, and reset mechanism. The temporal resolution of input data was also investigated. The authors show that the absence of leakage significantly reduces generalization, especially when temporal resolution is low. Surprisingly, adding recurrent connections between neurons decreases performances. It is noticed that leakage and recurrent connections have a higher impact than the resetting mechanism.

Energy consumption and spiking activity are also of interest for gradient-based SNNs. In [35], the authors discuss the impact of sparse convolutions on the internal spiking activity of the network on the DvsGesture dataset. They have also explored the impact of the sample duration on the accuracy. They concluded that the longer the sample duration, the better the accuracy, until a certain value, after which accuracy remains almost constant. Furthermore, they performed a GS to fix the HP α_{∇} used to approximate the gradient of the Heaviside function.

BOHB was used on CIFAR-100 classified by the S-ResNET38 architecture. The algorithm optimizes a 3 HPs search space made of: leakage τ_{leak} , timesteps T and learning rate λ_{∇} [274]. BOHB [69] is a multi-fidelity state-of-the-art algorithm for HPO [20]. It combines BO[84] and Successive halving [158]. The authors also did a comparison between accuracy and sample duration. They have tested 10, 20, 30, 40, and 50 timesteps and showed a relation between the leakage factor of the LIF and time steps. Empirical results emphasize that by reducing the number of time steps, one can adapt the leakage to counterbalance the accuracy loss due to shorter samples. Moreover, other works based on surrogate gradient show the impact of the neuron threshold on the accuracy [2]. Another work [15] studied the impact of the LIF on performances and proposed a new version of adaptive LIF allowing resonance. The authors showed that the LIF responds to a higher input spike frequency by a higher amplitude of the membrane V . Whereas the adaptive LIF reacts to certain frequencies depending on its HPs. Experimental results are obtained on RSNN and LSTM architectures trained by SLAYER to classify the SHD dataset [39]. HPO with Hyperband [158] was applied on some adaptive-LIF HPs. Using previously optimized solutions, other HPs were manually investigated on different datasets (SSC and ECG). The authors discuss the high impact of the leakage constant τ_{leak} and SLAYER's HPs (G_{∇} and τ_{∇}) on LIF networks. But it is unclear which HPs were optimized.

HPO was also applied on a 3 layers CSNN trained by NormAD [150]. The authors obtained an accuracy of 98.17% on MNIST where pixel intensities are directly converted into input current. The work manually optimized images' presentation time T , number of convolution filters $k^{(i)}$, and learning rates λ_{∇} . For current-based encoded MNIST, the authors

Table 2.1: Summary of hyperparameter optimization of STDP trained networks.

Sim./Hard.	Data	Arch.	Train	Opt.	G1	G2	G3	G4	G5	#	Obj.	Src
Brian2	Custom QUAV DVS	LGMD	STDP	2 EAs, 3 BOs	6	4	8	0	0	18	Acc, Sens, Prec, Spec	[230]
Bindsnet	MNIST, Fash- ionMNIST (P)	SOM	STDP	5 EAs	2	1	3	0	0	6	Acc	[217]
Brian2, FPGA	MNIST	SOM	STDP	GA	1	5	3	0	0	9	Acc, time constants	[102]
Bindsnet	MNIST	SOM	STDP	Man.	0	0	2	1	1	4	Acc	[106]
Bindsnet	DVS-Gesture	CSNN + Reservoir	STDP	Man.	0	0	4	0	0	4	Acc	[88]
?	DVS-Gesture	SOM	STDP + Tempotron	Man.	0	0	3	0	1	4	Acc	[129]
N2S3	MNIST	RBM	STDP	Man.	0	0	1	0	0	1	Acc	[241]
MegaSim, FPGA	MNIST, DVS-Poker	SNN	STDP	Man.	1	1	1	0	1	4	Acc	[297]
Spyker	MNIST	CSNN	STDP	GS	1	0	0	0	0	1	Acc	[245]
Matlab	EEG	Reservoir + deSNN	STDP	BO	0	0	0	4	2	6	Acc	[262]

empirically show that a 100ms exposition time is optimal. Lower and higher values result in lower accuracy.

Manual optimization was also applied on G3 HPs (architecture) [21]. In this work, a hybrid CNN-SNN network of adaptive LIF neurons is optimized to classify the TIMIT dataset [85]. The CNN part is used to encode waveforms into spikes that are then classified by the SNN. The authors investigate each HP separately. They optimized the number of layers and number of neurons for each layer of the SNN. Their experiments indicate that more neurons or layers do not necessarily result in better performances. They optimized the duration in milliseconds of a simulation time step, i.e. time resolution, showing that higher resolution significantly increases the computation cost of the training. However, it does not improve the performances. For a time resolution of 5ms the duration of an epoch is about 21 minutes, while for 1ms the cost is multiplied by 7, reaching 156 minutes per epoch. Furthermore, manual tuning was applied to the number of CNN filters and to the proportion of connectivity between feedforward layers.

In [240], the authors recorded a 19400 radar-based hand gestures dataset and classified it using a CSNN of LIF neurons trained by surrogate gradient. The \arctan function was used to approximate the derivative of the Heaviside function H' . The authors properly divided the dataset into train-validation-test splits and used the Optuna [3] library to perform HPO on 5 HPs: the learning rate of ADAM λ_{∇} , weight decay (L2 regularization [131]), the LIF threshold, and leakage, resp. V_{th} and τ_{leak} . The quality of the gradient approximation α_{∇} is also considered. Unfortunately, the authors did not precise the algorithm used for HPO, but 200 solutions were evaluated to tune these HPs. Similarly, in [199], GS, RS and BO were applied on the learning rate λ_{∇} , the true and false outputs spiking rates, resp. ν_T and ν_F . The work focuses on a CSNN applied to MNIST and audio MNIST encoded into spikes. RS appears to perform better, but too little information about the process is given to conclude strongly. In [222], BO was also used to optimize a 1 hidden layer feed-forward SNN of LIFs to classify SHD. The authors studied meta-learning (MetaL) using the MAML algorithm and applied HPO to tune 5 HPs: the 2 learning rates from MAML, batch size, the number of neurons, and the target firing rates at the SNN output.

HPO of SNNs was investigated in astronomy to detect Radio Frequency Interference [219]. The authors optimized a two fully connected layers (2-l) SNN to classify simulated data based on the HERA dataset. HPO was addressed by the TPE [284] algorithm on 4 HPs: the batch size, number of epochs, number of encoding timesteps T , and the LIF leakage τ_{leak} . The same optimization algorithm was applied in [226] where a CSNN trained on a bank account fraud dataset was tuned. The experiments sampled about 50 to 200 HPs combinations and focused on 7 HPs: learning rate and momentum of Adam λ_{∇} , $\beta_{1\nabla}$, $\beta_{2\nabla}$, LIF threshold V_{th} for each 3 layers, and the leakage τ_{leak} .

2.3.3 Other approaches

EONS [237] is a *neuroevolution* strategy using EA to tune parameters and hyperparameters of SNNs. Here, the application is strongly linked to the neuromorphic hardware (Caspian neuromorphic computing system). Parsa et al. [203] applied BO and GS on EONS. They show that BO performs better than GS in terms of accuracy. BO optimized a search space of 54 432 000 countable solutions made of up to 11 HPs. They have empirically demonstrated that HPs of the encoding method and hardware have a greater impact than HPs of the GA used to train SNNs. The same authors [204] applied H-PABO, a multi-objective BO, on two and three objectives: network performances, energy consumption, and number of synapses.

Tournament Selection was used to optimize a SNNs trained by ANN-to-SNN conversion [144]. The authors optimized an objective function combining accuracy, spiking activity,

Table 2.2: Summary of hyperparameter optimization of SNNs trained by surrogate gradient.

Sim./Hard.	Data	Arch.	Train	Opt.	G1	G2	G3	G4	G5	#	Obj.	Src
slayer-PyTorch	DVS-Gesture	CSNN	SLAYER	H-BO	0	4	2	1	1	8	Acc, Lat	[205]
LAVA-DL	NMNIST	CSNN	SLAYER	lavaBO	3	2	0	0	0	5	Acc	[251]
Pytorch	CIFAR-10, CIFAR-100	S-Resnet	SG	BOHB	1	1	0	1	0	3	Acc	[274]
Custom CUDA-C	MNIST	CSNN	NormAD	Man.	0	1	1	1	0	3	Acc	[150]
Tensorflow	Monkey J Maze	LFADS	SG	PBT	0	1	0	0	4	7	R^2	[139]
snnTorch	FSDD	CSNN	SG	BO, RS	2	0	0	1	0	3	Acc	[156]
snnTorch	Radar Hand Gesture	CSNN	SG	?	2	3	0	0	0	5	Acc	[240]
Pytorch	TIMIT	CNN-SNN	SG	Man.	0	0	4	1	0	5	PER	[21]
?	MNIST, Audio MNIST	CSNN	SG	GS, RS, BO	0	3	0	0	0	3	Acc	[199]
snnTorch	HERA	2-1 SNN	SG	TPE	1	0	0	1	2	4	Acc, MSE, AUROC, AUPRC, F1	[219]
snnTorch	SHD	1-1 SNN	MetaL, SG	BO	0	2	1	0	2	5	Acc	[222]
snnTorch	Bank Account Fraud	CSNN	SG	TPE	4	3	0	0	0	7	FPR, Recall	[226]

and inference time to evaluate the performances of a RNN classifying names according to their origin language. Three HPs were optimized: the neuron threshold V_{th} , input spiking frequency, and T the sample duration. The authors compared optimized HPs to the three selected objectives, and deduced that HPO allows to find more accurate and energy-efficient solutions with lower inference time. ANN-to-SNN conversion was investigated in [207]. The authors optimized the ANN HPs before the conversion to a SNN. However, no analysis of the impact of the HPO is made. They have implemented the network on the Intel Loihi neuromorphic chip [41] by applying network quantization. A power consumption comparison is made between CPUs, GPUs, and Loihi chips [41].

2.3.4 Discussion and remarks

In the previous sections, we presented works applying HPO to SNNs. In figure 2.17, the usual workflow of HPO applied to SNNs is presented. One can distinguish the blackbox containing the data preprocessing, the network, the decoding, a simplified training-validation loop, and most importantly, the input and output spikes. Outside the blackbox are the data, the mono- or multi-objective HPO algorithm, and the computation of the different objectives.

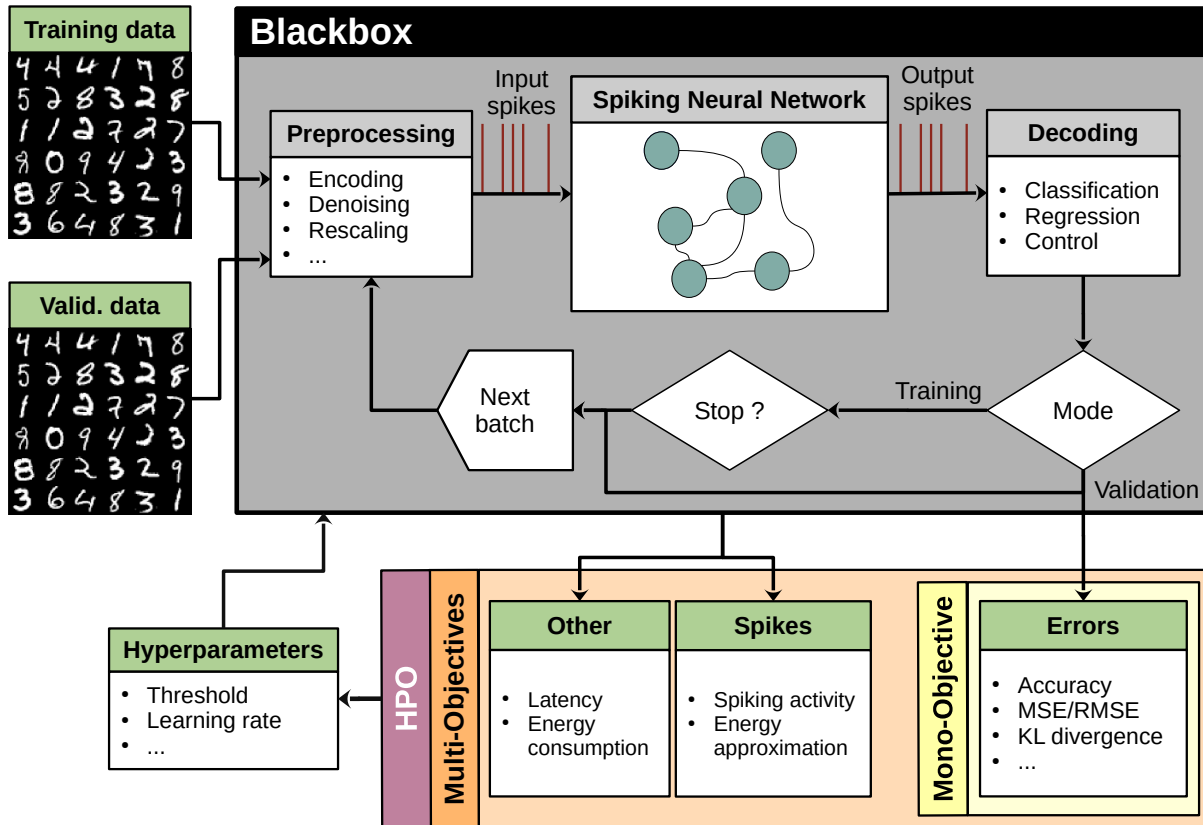


Figure 2.17: Usual workflow of HPO applied to SNNs

To sum up, the HPO of SNNs is a problem characterized by several properties:

- **Black-box:** Information such as the derivatives of the objective function w.r.t. HPs are inaccessible. Data, HPs and prediction errors are the only available information.
- **Expensive:** Evaluating a single solution takes from minutes to hours. It requires specific hardware such as neuromorphic chips or GPU-acceleration. The budget allocated to

Table 2.3: Summary of hyperparameter optimization of spiking neural networks with various training algorithms.

Sim./Hard.	Data	Arch.	Train	Opt.	G1	G2	G3	G4	G5	#	Obj.	Src
NengoDL, Loihi	Cell	U-net	Conv.	?	1	2	0	0	4	?	Acc	[207]
Tensorflow	PASCAL VOC, MS COCO	Tiny YOLO	Conv.	BO	1	0	0	0	0	8	Acc, Spikes, Lat	[144]
TENNLab, DANNA2, mrDANNA	Iris, Radio	EONS	EONS	BO	0	3	0	5	0	8	Acc, En. , Size	[204]
TENNLab, DANNA2	Pole-Balance	EONS	EONS	H-PABO, GS	1	3	0	5	0	11	Acc	[203]
Brian2	MNIST, KTH	Reservoir	\emptyset	SBO, CMA-ES	2	0	6	1	0	9	Acc	[305]

HPO (e.g. time, money, electricity) is limited.

- Noisy: The evaluation of a single HPs combination is stochastic, i.e. evaluating multiple times the same solution returns different objective values.
- High dimensional: The search space is made of tens of HPs. In this manuscript, we focus on automating the search of a wide choice of HPs. Indeed, SNNs have more HPs than ANNs due to a more complex neuron model.
- Sensitive: SNNs are known to be very sensitive to their HPs. Slightly disturbing the value of an HP can result in wholly different results. We can link this phenomenon to a low-conditioned objective function.

In numerous works, HPO was not the main focus of the article. Often, HPO is used to tune some HPs of new approaches or applications. The process itself and its results are rarely studied. This lack of interest in the process results in very little information about it. But more importantly, there is a lack of information about the methodological standard in which the work takes place. In some cases, this lack of information turns out to produce unreliable and irrelevant results.

Here is a non-exhaustive list of recommendations gathered throughout this thesis, that could help to improve readability, reproducibility, peer review, and reliability of HPO applied to SNN:

- Data:
 - Specify the dataset and its repository. If unusual, describe how it is made or give a direct reference to it.
 - Specify the pre-processing steps and its HPs including spike encoding and encoding timesteps. Precising the final shape of data at the input of a SNN helps to understand the process.
 - Specify the splits and their size or proportions: train-valid-test splits, cross-validation or nested cross-validation.
 - If not usual or ambiguous, specify the response and its type. In other words, what is predicted by the SNN and if it is a regression or a classification task.
- Search space:
 - If ambiguous, specify and describe all HPs optimized by HPO and their bounds. Moreover, describing how they are sampled could help improve the HPO.
 - If ambiguous, do not mix within a table, parameters, fixed, and optimized HPs. Describing fixed HPs is a plus, but giving a reliable source code can overcome this.
 - Specify what are the learned parameters and the optimized HPs. In specific cases, some HPs of SNNs can be trained by gradient descent [70].
- Neuron, Architecture, training:
 - Describe the SNN architecture or give a direct reference to it.
 - If there is a baseline configuration, giving its performances, training time, and memory footprint (e.g. number of parameters) greatly helps in designing an HPO.
 - Specify the type of training and the optimized loss.

- Specify or give a direct reference to equations of the neuron model and learning rules.
- Specify how the output spikes are decoded.
- Giving the number of epochs and batch sizes helps designing a HPO. It allows an estimation of the required computational resources.
- Simulator:
 - If used, specify the simulator or a source code of a custom one.
- Hardware:
 - Specify the type of hardware or the fabric if a grid or cluster of resources is used.
 - Specify if any parallelization methods are applied and how they are used to accelerate the HPO. For instance, indicate if CUDA or MPI is needed.
- Software:
 - Indicate the programming language and its version.
 - Specify the major packages, their versions, and what they are used for.
 - Give a repository with the source code and results of the experiments.
- HPO:
 - Specify the optimization algorithm and its configuration. Occasionally, we see a confusion between the HPO software and the optimizer.
 - Indicate what metric is optimized through HPO and what are the objectives. Is it maximization or minimization? Is it mono- or multi-objective? Is it constrained? Is it multi-fidelity? To avoid any ambiguity, mention on which split (validation) the optimization is performed.
 - Specify the stopping criterion: convergence, budget, minimal required performances, etc.
 - Give the experiment duration and the number of sampled solutions. Indeed, from time to time, some works insufficiently optimize very high-dimensional search spaces with less than or a few hundreds of evaluations.
 - If the algorithm has to converge, analyze the convergence of the HPO, e.g. EA or BO. We should be able to answer the following question: *"Are the results found by random or through a converging process?"*
 - If train-validation-test splits are used, we should always select the best solution found by HPO using the performances on the validation set. Then report the final performances on the test set without any further tuning, even manual. The optimized solution can be retrained multiple times to estimate its stochasticity. To clear up any confusion, one can report training, validation, and test performances.
 - Even manual tuning requires at least a train-validation-test split. Once optimized and tested, a solution should not be tuned further.
 - HPO can be very costly and time-consuming. If possible, comparisons between several algorithms on multiple runs are needed to determine which approach is better. However, the computational cost of HPO is a major drag on strong comparisons. Such comparative works are often practically insoluble.

Algorithms for Hyperparameter Optimization

In chapter 2, we described the formalization of HPO. Once the problem is defined, the HPs selected, and the search space built, the choice of an optimization algorithm also has to be informed by the available budget and resources. Among works tackling HPO of SNNs and beyond manual tuning, the usual algorithms applied to HPO of ANNs are similarly deployed on SNNs. The following lines discuss the basic strategies often applied for both HPO of ANNs and SNNs. The algorithms are presented, as well as a description of their parallelization within a distributed environment. In the previous state-of-the-art, we noticed that optimization algorithms used for HPO of SNNs come from various families; the two main ones are *Metaheuristics* [259, 253, 166] and *SMBO* [122, 123]. These algorithms are characterized by a common property: they can be applied to black-box functions. Furthermore, these two families can be distinguished by the usage or not of an additional ML model. Indeed, in SMBO, a trainable model interpolates the objective function, e.g. the accuracy of a SNN w.r.t. its HPs.

The No Free Lunch Theorem drives this chapter, there is no single universal algorithm for all problems. All the following algorithms have their advantages and drawbacks. A thorough description is necessary to better understand the choices in chapters 5 and 6. In this manuscript, we do not tackle hyperheuristics [253, 301], which could help better select an algorithm. For now, these approaches are untractable due to the computational complexity of simulating and training SNNs. However, meta-learning was already applied to some SNNs tasks [222].

In the face of computationally expensive NP-hard problems, parallelizing optimization algorithms is unavoidable. Formalization of parallel metaheuristics dates back to the end of the 1990s to the beginning of the 2000s [6, 95, 176] with software frameworks such as ParadisEO [27]. Technological advances strongly bound this field. In 2008, the first Petascale architecture with 10^{15} double precision operations per second was achieved with the IBM Roadrunner supercomputer¹. Today's challenge is to make optimization algorithms scale to Exascale, first reached in 2022 by the Frontier supercomputer². Recent architectures are converging toward GPU acceleration for AI. Thus, in this manuscript, we mainly focus on distributed parallelization with GPU-accelerated grid computing applied to HPO of SNNs.

This chapter essentially details line 16 of algorithm 2 (the optimizer \mathcal{O}), and how this algorithm could be parallelized according to the choice of \mathcal{O} . As a reminder, a solution to the

¹<https://top500.org/lists/top500/2008/11/>

²<https://top500.org/lists/top500/2022/06/>

HPO problem is a set of HPs $\lambda \in \Omega$, from a search space Ω . Additionally, the *objective function* is optimized by an HPO algorithm (e.g. BO), and a *loss function* is optimized by the training algorithm of a SNN or ANN (e.g. gradient descent). Both *objective function* and *loss function* can sometimes describe the same value. For instance, the training algorithm can optimize the MSE on a training dataset, and an HPO algorithm can minimize the MSE on the validation dataset (see chapter 2 for details).

3.1 Grid search and random search

GS is one of the most popular algorithms for HPO. It is an exhaustive algorithm consisting of discretizing the search space into a finite number of solutions. Box-constrained discrete and continuous values are equidistantly spaced on a grid of dimension d [20, 123, 19]. Thus, GS consists in evaluating all possible solutions without any particular selection strategy. The search space Ω is described by the grid itself. GS is close to a certain form of manual tuning consisting in fixing HPs while varying a single one, as applied to ANNs [249] or SNNs [295, 106].

Usually, GS is used as a baseline comparison with other more advanced techniques [203, 199, 245]. But, GS is outperformed by BO, making BO-based algorithms state-of-the-art in HPO. The major drawback of GS is its poor scalability in dimension. Indeed, if the size of the grid is g , then g^d solutions have to be evaluated. Such computational complexity grows exponentially w.r.t. the dimension d (number of selected HPs).

An alternative to GS is RS. The algorithm consists in randomly selecting independent and identically distributed HPs combinations with no particular strategy. Conversely to GS, RS scales better in dimension and has higher performances [19]. Since the hypervolume (e.g. Lebesgue or Hausdorff measures) grows exponentially, uniformly sampling points to efficiently cover Ω do not scale. Other types of sampling could be employed to better cover Ω with a limited number of samples. For instance, LHS or low discrepancy sequences with Sobol, Kronecker, or Halton sampling [56]. Customized GS can be used to initialize other optimization algorithms. In chapters 5 and 6, we discuss how to define HP-specific sampler to handle multiplicative or additive effects, or to bias the sampling using prior knowledge and toward known promising solutions. Nonetheless, both GS and RS are very basic algorithms, which are described in algorithms 3 and 4. These methods are outperformed by guided algorithmic approaches, discussed in the next sections.

Algorithm 3 Sequential GS

Inputs:

- 1: Ω *Discretized search space*
 - 2: **for** each $\lambda \in \Omega$ **do**
 - 3: **Evaluate**(λ)
-

Algorithm 4 Sequential RS

Inputs:

- 1: Ω *Search space*
 - 2: **while** stopping criterion not met **do**
 - 3: **Evaluate**(**Random**(Ω))
-

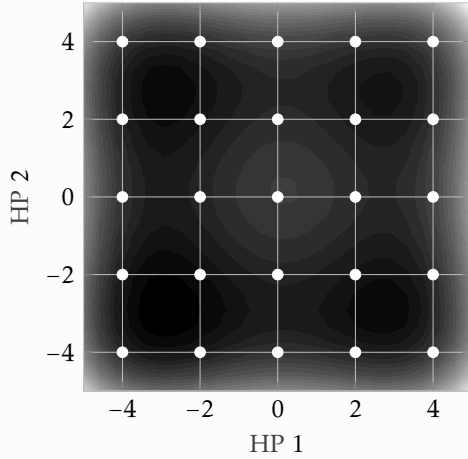
Example: GS and RS

Figure 3.1: GS with a grid size of 5.

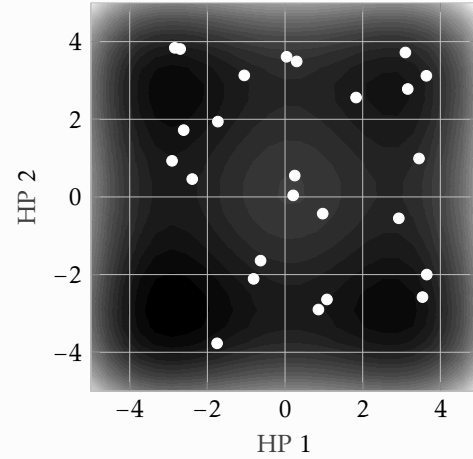


Figure 3.2: RS with 64 points.

Examples of GS and RS applied to the 2-dimensional Styblinski-Tang function.

3.2 Metaheuristic approaches

Today, finding the global optimum of NP-hard problems is practically unsolvable in a reasonable time. Many real-world problems are black-box, noisy, multimodal, non-convex, and high dimensional. Usual optimization algorithms such as convex optimization fail at solving these problems [259]. A *heuristic* is an optimization algorithm specifically designed for a problem or a problem instance. Usually, heuristics do not guarantee convergence toward the global optimum, but they return good-quality solutions within an acceptable budget [253, 259]. The prefix *meta* refers to a higher level of abstraction. Hence, *meta-heuristics* are problem-independent and more general heuristics [253, 259]. Compared to some exact algorithms described in chapter 4, metaheuristics trade exactness for efficiency. Metaheuristics are designed to be flexible, adaptable to various problems (e.g. continuous optimization, combinatorial problems, discrete problems, mixed problems), and capable of finding suitable solutions in high-dimensional search spaces. Such optimization algorithms are one of the favorite candidates for HPO [108, 260, 123, 16, 20].

Metaheuristics are mainly nature-inspired algorithms. Their design can rely on biological evolution (EA), swarm intelligence (PSO) or even physics (SA). Usually, the quality of a solution – the objective value – is named a *fitness*. But we do not employ the term *fitness* to avoid any ambiguities through the chapters.

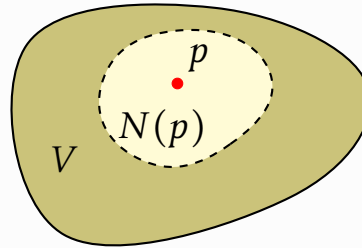
HPs can be of various type, continuous with the neuron threshold V_{th} , discrete with the number of neurons $n^{(lfamily)}$ of a layer l , categorical with the neuron model (IF, LIF, Izhikevitch). So our solution is of mixed types, and Ω is a *mixed searchspace* [261], i.e. $\Omega \triangleq \Omega_1 \times \Omega_2 \times \dots$.

Metaheuristics are characterized by the *exploration-exploitation* tradeoff. Exploration consists in gathering general information about the search space, i.e. trying low-confidence – high-risk solutions, and seeking very diverse solutions to identify promising areas. Exploitation refines this information by improving more thoroughly prior knowledge. For instance, RS is an exploration-only algorithm, while Local search is exploitation-only.

Local search is a hill-climbing heuristic starting from a given solution. Then, it samples new candidates within the *neighborhood* of the current solution of interest. If a neighbor improves the current solution, the algorithm moves to this neighbor. Therefore, the efficacy

of a metaheuristic also relies on the definition of an efficient neighborhood.

Example: Neighborhood



Example of a neighborhood given by the neighborhood function $N(p)$ (the lighter ball) and centered at a point p within a space V .

The high flexibility of metaheuristics allows one to have the hand on many aspects of the algorithms. Indeed, metaheuristics allow easy fine-tuning of memory consumption. This flexibility facilitates the instantiation of the algorithm on a wide range of hardware, from low-memory embedded systems to high-performance computing clusters. For example, single-solution and sequential metaheuristics like SA require less memory than Tabu Search [94]. For instance, population-based methods like GA can be adjusted by tuning the population size so to use more or less memory. Furthermore, practitioners can influence the exploration-exploitation balance of the search process by adjusting the neighborhood structure and the variation operators, i.e. algorithmic components of a metaheuristic (mutation, crossover, or selection in GA). This customization allows for fine-tuning of the metaheuristic's behavior to better suit the problem's properties.

The degree of greediness in a metaheuristic can also be adjusted. It defines how it focuses on improving the current solution versus exploring new areas of the search space. For example, algorithms like Tabu Search allow the incorporation of long and short-term memories that prevent the algorithm from revisiting recently explored solutions. In contrast, other strategies, as SA, might prioritize local improvements. This creates a more greedy approach that quickly converges to good solutions, but which lacks of exploration.

Metaheuristics are inherently suitable for parallelization, which is critical for solving large-scale optimization problems (see section 3.4).

3.2.1 Simulated Annealing

SA [272] is a local search strategy inspired by the annealing process in metallurgy. This metaheuristic is rarely applied to HPO [75, 268, 260], and to our knowledge was never applied to SNNs. Yet, some works try implementing neuromorphic annealing processes [300]. In the context of HPO, SA starts with an initial set of HPs and evaluates its performances, i.e. objective values. It then iteratively explores the HPs space by making small random changes to the current HPs. The key feature of SA is its ability to escape local optima by occasionally accepting worse solutions, according to an acceptance probability. This probability is based on a value named the *temperature* which decreases over time, i.e. the *annealing*.

The annealing or *cooling schedule* modifies the temperature through iteration, allowing SA to balance exploration and exploitation. Initially, the temperature is high, so the algorithm can easily accept worse solutions, enhancing exploration of Ω . As the temperature decreases, the algorithm becomes more conservative. It slowly focuses on exploitation as the acceptance probability decreases, allowing SA to converge toward an optimal or near-optimal set of HPs.

SA can be used in HPO to refine the best solution found by a global optimizer, or to test if a baseline solution is easily optimizable within its neighborhood. The SA pseudocode is described in algorithm 5. The acceptance probability \mathbb{P} at iteration i is usually defined as $\exp\left(-\frac{Y_i - Y_{\text{cur}}}{t_i^\circ}\right)$, with Y_i the objective value from the neighbor λ_i of the current solution λ_{cur} . The current objective value is denoted Y_{cur} , and the current temperature t_i° . The cooling schedule T° is a decreasing function, converging to 0.

Algorithm 5 SA

Inputs:

1: λ_0	<i>Initial solution</i>
2: N	<i>Neighborhood function</i>
3: f	<i>Objective function</i>
4: \mathbb{P}	<i>Acceptance probability</i>
5: \mathcal{U}	<i>Uniform distribution</i>
6: T°	<i>Temperature function</i>
7: $Y_0 \leftarrow f(\lambda_0)$	
8: $i \leftarrow 1$	
9: $\lambda_{\text{cur}} \leftarrow \lambda_0$	<i>Current solution</i>
10: $Y_{\text{cur}} \leftarrow Y_0$	
11: while stopping criterion not met do	
12: $t_i^\circ \leftarrow T^\circ(i)$	<i>Current temperature</i>
13: $\lambda_i \leftarrow N(\lambda_{\text{cur}})$	
14: $Y_i \leftarrow f(\lambda_i)$	
15: if $\mathbb{P}(t_i^\circ, Y_i, Y_{\text{cur}}) < \mathcal{U}(0, 1)$ then	
16: $\lambda_{\text{cur}} \leftarrow \lambda_i$	
17: $Y_{\text{cur}} \leftarrow Y_i$	
18: $i \leftarrow i + 1$	
return best of λ_i	

SA is a local search strategy that can suffer from lack of exploration, and it requires many evaluations to converge. To globally optimize Ω metaheuristics from the EAs family can be considered.

3.2.2 Evolutionary Algorithms

EA is inspired by biological evolution and based on making a *population* of solutions evolves through iterations. For instance, within our HPO problem, a population is made of vectors of HPs. The population has to slowly converge toward exploitation of promising areas of Ω .

Different EAs were applied to address HPO of SNNs [230, 217, 102]. Another application of EAs on SNNs known as *neuroevolution* [77, 61], combines EA with ANN or SNN design and training. Instead of using gradient backpropagation or Hebbian rules to optimize neural networks, neuroevolution evolves simultaneously the architecture and weights of the networks. This approach allows for the automatic and simultaneous discovery of network topologies, optimal parameters, and HPs. It can be considered a combination of HPO, NAS and training. Because EAs are metaheuristics that can be applied to black-box problems, neuroevolution can sometimes be an acceptable first approach to some complex ML models. HPO was also applied on neuroevolutionary SNNs [204, 203] where the optimized HPs are the parameters of the EA.

In this section, we focus on GA [115] which was also applied to HPO of SNNs [102]. Additionally, GA was one of the first unsuccessful algorithms used in this thesis. This is

discussed in chapter 5.

A GA is usually made of 3 components, *selection*, *mutation* and *crossover*. We can also consider a fourth component known as *elitism*. In GAs, a solution is called an *individual*. An element of a solution is a *chromosome*, i.e. a single HP value from a single HPs combination. A batch of solutions is a *population*. A *generation* is an iteration of the algorithm consisting in the creation of a new *population*. The children from the crossover between selected individuals are called the *offsprings*.

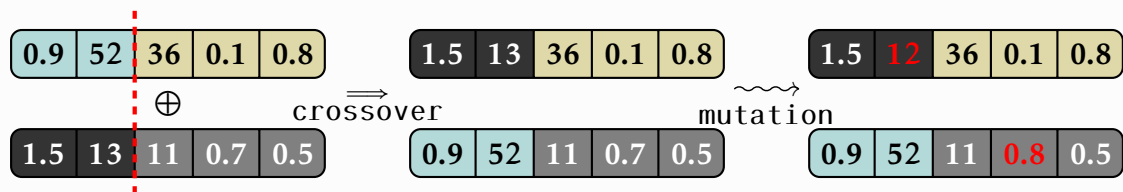
The first step of a GA is to initialize a population of HPs combinations. This initialization is usually done randomly. But as for RS, different samplers can be used, such as LHS, low discrepancy sequences, or even heuristics approaches [259, 260].

The *selection* operator determines which individuals from the population will contribute to the offspring by selecting those with higher objective values. This operator balances exploitation of good solutions and exploration of new HPs combinations. In GAs, the *selection* allows the convergence of the population toward optimal or near-optimal solutions. Various selection methods can be used to ensure that the best solutions have a higher chance of being chosen in the next generation, e.g. roulette or tournament selections. The main difficulty of a good selection is to maintain diversity within the next generations while allowing the algorithm to converge in a reasonable time.

The *crossover* simulates reproduction. The chromosomes from pairs of selected individuals are interchanged to create the offspring, i.e. allowing offspring to share some similarities with their parents. This operator contributes to diversity within the next population. Popular crossover strategies are single-point, multi-point, or uniform crossover. Each of these defines how chromosomes of parents are selected and recombined. By effectively mixing supposedly good chromosomes of the parents, crossover helps GAs to explore the search space more efficiently toward promising areas.

The *mutation* can be considered as an exploration component of a GA. It introduces random changes to the children of the offspring, resulting in a higher diversity of the next population within a certain neighborhood. Mutation prevents the algorithm from becoming trapped in local optima by allowing the exploration of new and local areas of Ω .

Example: Crossover and mutation



The above figure illustrates a one-point crossover (red dashed line), and a random mutation selecting a neighbor of one chromosome (HP).

The *elitism* operator describes how to create the new population from the current one and the offspring. Usually, the k worst individuals of the old population are replaced by the k best individuals from the offspring. Elitism also impacts the convergence and the diversity of the populations.

Algorithm 6 illustrates the workflow and the four operators of a GA. One can notice the simplicity and flexibility of GA offered by the component in lines 7, 8, 9 and 11.

EAs are usually more efficient than GS and RS [20, 160, 9]. Nonetheless, a major drawback of these approaches is the necessity of numerous evaluations. It becomes a bottleneck when the objective function is expensive to evaluate.

Algorithm 6 GA**Inputs:**

1: Ω	<i>Search space</i>
2: f	<i>Objective function</i>
3: G	<i>Number of generations</i>
4: $\mathcal{P}_0 \leftarrow \text{RandomPopulation}(\Omega)$	<i>Initialize population</i>
5: $Y_0 \leftarrow f(\mathcal{P}_0)$	
6: for $i \leftarrow 1$ to g do	
7: $\text{selected} \leftarrow \text{SELECT}(\mathcal{P}_{i-1})$	
8: $\text{offspring} \leftarrow \text{CROSSOVER}(\text{selected})$	
9: $\text{offspring} \leftarrow \text{MUTATE}(\text{offspring})$	
10: $Y_{\text{offspring}} \leftarrow f(\text{offspring})$	
11: $\mathcal{P}_i, Y_i \leftarrow \text{ELITISM}(\mathcal{P}_{i-1}, \text{offspring}, Y_{i-1}, Y_{\text{offspring}})$	
return best of \mathcal{P}_g	

CMA-ES is a state-of-the-art EA designed for continuous optimization problems. Similarly to GA, the algorithm iteratively improves a population of solutions to optimize the search space and objective function. CMA-ES starts with an initial population drawn from a multivariate normal distribution, characterized by a mean vector μ and a covariance matrix Σ . At each iteration, it generates the offspring by sampling from this distribution. The algorithm evaluates their objective values and then updates μ and Σ based on the best-performing solutions. The mean vector is shifted towards better regions of the search space, while the covariance matrix is adapted to capture the correlations between variables and adapt itself to the landscape of Ω . Its top performances are assessed on the COCO benchmark [103]. CMA-ES was also applied to HPO of ANNs [168] but also on SNNs [305].

This algorithm is a good transition to BO which also extracts probabilistic information about evaluated solutions to better optimize.

3.3 Surrogate Model-Based Optimization

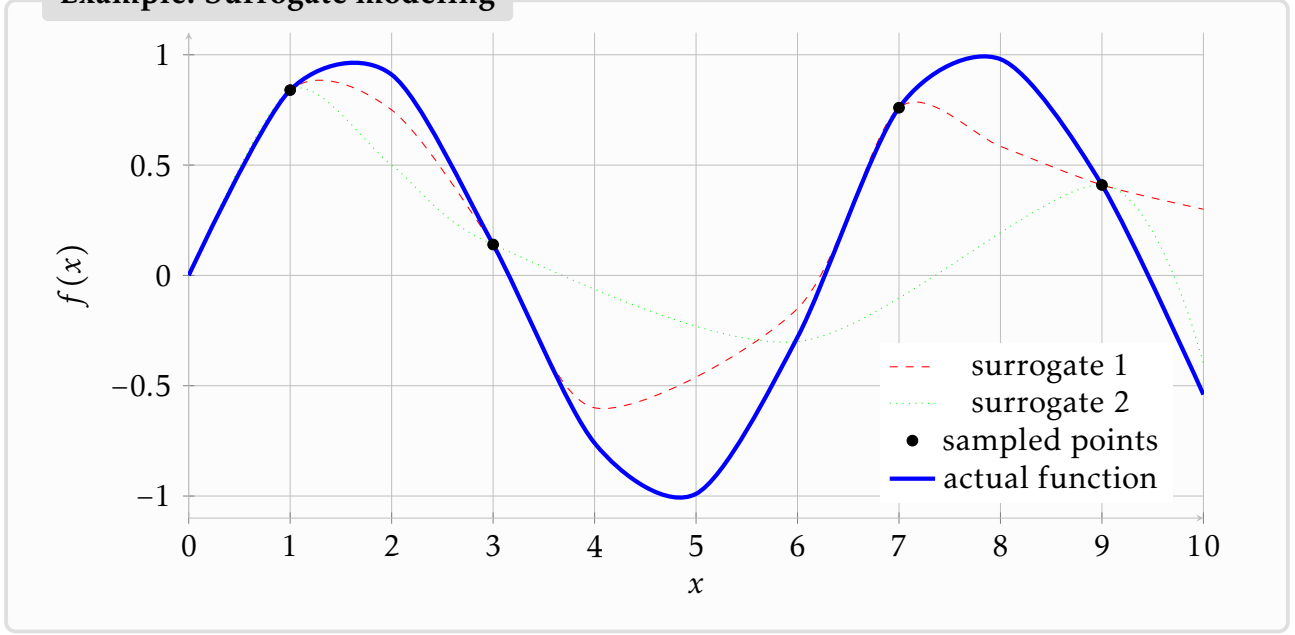
The following content is mainly inspired by this book [84].

SMBO is an optimization approach used when the objective function f is expensive in terms of time, budget, cost, resources availability, etc [122, 221, 123]. For that reason, state-of-the-art HPO algorithms applied to expensive ML are from this family. Instead of directly optimizing the objective function, SMBO builds a surrogate model, i.e. a costless approximation of the actual objective function. This surrogate is faster to evaluate. By optimizing the surrogate instead of directly sampling the actual objective function, SMBO allows to sample more cautiously solutions that will be evaluated on the original function.

Popular surrogate models are GP, Random Forest, Deep GP, SVM, ANN, etc. The optimization process involves iteratively updating the surrogate model based on previously evaluated points. A SMBO algorithm constantly updates its prior knowledge or its belief about the objective function. SMBO efficiently balances exploration and exploitation of the search space by focusing on promising areas while limiting costly evaluations.

3.3.1 Bayesian modeling

Among the SMBO algorithms, we focus on one of the most popular algorithms named GP-based BO, a.k.a. Kriging [84]. BO iteratively creates an *archive* of actual observations to build an *observation model* on the objective function.

Example: Surrogate modeling

The observed value y is noisy and distributed around the value of the objective function $\phi = f(\lambda)$ at λ . If f is noiseless, then $y = \phi$.

In an *observation model*, y is assumed to be explained by a stochastic process relying on ϕ : $p(y | \lambda, \phi)$. We can model noisy observations by $y = \phi + \varepsilon$. This model is known as the *additive Gaussian noise*, with ε a zero-mean Gaussian observation error scaled by σ_{noise} . Hence, the Gaussian observation model becomes

$$p(y | \lambda, \phi, \sigma) = \mathcal{N}(y; \phi, \sigma_{\text{noise}}^2) , \quad (3.1)$$

Example: Validation accuracy of a SNN

Let's consider the accuracy of a SNN $\mathcal{L}_{\mathbb{D}}(\mathcal{A}_{\theta^*}^\lambda, \mathcal{D}_{\text{valid}})$ from a combination of HPs λ . The observed accuracy y at λ is corrupted by noise applied to an ideal noiseless objective function value ϕ at λ . The supposedly Gaussian noise is scaled by σ_{noise} .

In a Bayesian approach, one has to bring knowledge about the distribution of the objective values ϕ . This prior knowledge is then refined by successively observing noisy values y from f at λ . To do so, we define three principles, also illustrated in figure 3.3:

- the *prior*: $p(\phi | \lambda)$ is an assumption, a belief, on the distribution of ϕ .
- the *likelihood*: $p(y | \lambda, \phi)$ describes the distribution of how *likely* we can observe y at λ if distributed around ϕ .
- the *posterior*: $p(\phi | \lambda, y)$ is used to update the prior according to observations y .

Then, the Bayes' theorem is used to update the *posterior* distribution,

$$p(\phi | \lambda, y) = \frac{p(\phi | \lambda)p(y | \lambda, \phi)}{p(y | \lambda)} . \quad (3.2)$$

We consider an archive of known observations defined by observation pairs, s.t. $\mathcal{D} \triangleq [(\lambda_1, y_1), \dots, (\lambda_n, y_n)]$.

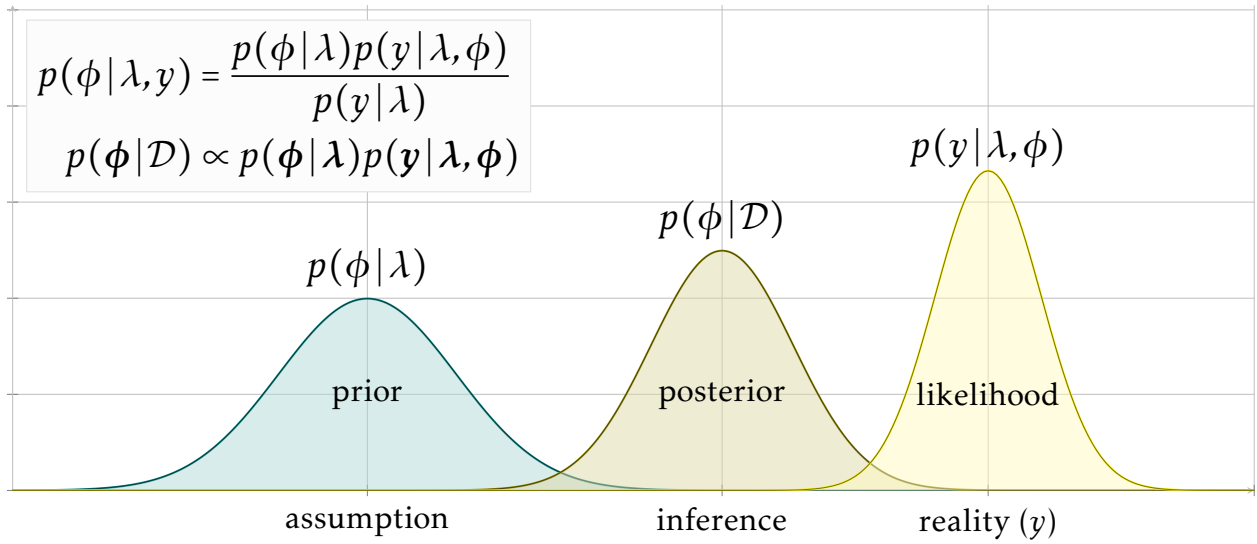


Figure 3.3: Illustration of Bayesian inference

According to Bayes' theorem defined by equation 3.1, the impact of observations on a set of noiseless objective function values ϕ is proportional to the prior and the likelihood

$$p(\phi | \mathcal{D}) \propto p(\phi | \lambda) p(y | \lambda, \phi) . \quad (3.3)$$

Let a *prior process* $p(f)$ be a belief on a function f , and a *posterior process* $p(f | \mathcal{D})$ informed by observations. Then to compute $p(f | \mathcal{D})$, the set ϕ is extended to f such that

$$p(f | \mathcal{D}) = \int p(f | \lambda, \phi) p(\phi | \mathcal{D}) d\phi . \quad (3.4)$$

The approximation of f informed by observations \mathcal{D} relies on the prior process, i.e. the initial assumption about f and the posterior on ϕ .

3.3.2 Gaussian processes

GPs [223] are Bayesian ML models used as surrogates in BO. They are non-parametric as they are *observation models* relying on an archive of observations \mathcal{D} . A GP regressor uses a multivariate normal distribution to model the landscape of the actual objective function, e.g. the accuracy of a SNN on a validation set. Then the *prior process* on f is written

$$p(f) = \mathcal{GP}(f; \mu, K) , \quad (3.5)$$

with μ the mean function $\mu(\lambda) = \mathbb{E}[\phi | \lambda]$ describing the expected objective function value $\phi = f(\lambda)$ at λ . Usually, the observations are standardized to have a zero-mean. The kernel function K gives the covariance matrix computed on each pair of positions (λ, λ') and objective values (ϕ, ϕ') .

The kernel K is a symmetric function defined as $K(\lambda, \lambda') = \text{Cov}(\phi, \phi' | \lambda, \lambda')$, and the resulting covariance matrix has to be PD.

Definition 8 (Symmetric kernels). *Let Λ be a non-empty set. A function $K : \Lambda \times \Lambda \rightarrow \mathbb{R}$ is symmetric if,*

$$\forall (\lambda_1, \lambda_2) \in \Lambda, K(\lambda_1, \lambda_2) = K(\lambda_2, \lambda_1) . \quad (3.6)$$

Definition 9 (Positive definite kernels). *Let Λ be a non-empty set. A symmetric function $K : \Lambda \times \Lambda \rightarrow \mathbb{R}$ is a PD kernel if,*

$$\forall n \in \mathbb{N}, \forall \lambda_1, \dots, \lambda_n \in \Lambda, \forall c_1, \dots, c_n \in \mathbb{R}, \sum_{i=1}^n \sum_{j=1}^n c_i c_j K(\lambda_i, \lambda_j) \geq 0 . \quad (3.7)$$

The multivariate distribution from a GP of a set of positions λ and objective values ϕ is written

$$p(\phi | \lambda) = \mathcal{N}(\phi; \mu, \Sigma) , \quad (3.8)$$

where μ and Σ are the means and covariance computed on all elements of ϕ using $\mu(\lambda)$ and $K(\lambda, \lambda')$.

3.3.3 Inferring with a GP

Marginalization

A n -dimensional random vector of multivariate normal random variables $\lambda = (\lambda_1, \dots, \lambda_n)^\top$ following a multivariate normal distribution s.t. $p(\lambda) = \mathcal{N}(\lambda; \mu, \Sigma)$ can be marginalized. It means that we can *extract* any vector $\lambda' \subset \lambda$ and model its *marginal distribution* by dropping unused elements from the mean vector μ and covariance matrix Σ .

Assuming a partitioning

$$\lambda = \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix} ,$$

with marginal distributions $p(\lambda_1) = \mathcal{N}(\lambda_1; \mu_1, \Sigma_{11})$ and $p(\lambda_2) = \mathcal{N}(\lambda_2; \mu_2, \Sigma_{22})$. We can rewrite the distribution of λ as

$$p(\lambda) = \mathcal{N} \left(\begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix}; \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix} \right) , \quad (3.9)$$

where Σ_{12} and Σ_{21} are cross-covariance between both partitions.

So, any vector of observations y from f with a marginal distribution $p(y) = \mathcal{N}(y; m, C)$ shares a joint multivariate normal distributions with a GP on f

$$p(f, y) = \left(\begin{bmatrix} f \\ y \end{bmatrix}; \begin{bmatrix} \mu \\ m \end{bmatrix}, \begin{bmatrix} K & \kappa^\top \\ \kappa & C \end{bmatrix} \right) , \quad (3.10)$$

with the cross-covariance $\kappa(\lambda) = \text{Cov}[y, \phi | \lambda]$ between observations y and the function value at λ .

Conditioning

Considering a similar partition as in equation 3.3.3, we can condition y_1 according to a known measurement of y_2 by using

$$p(y_1 | y_2) = \mathcal{N}(y_1; \mu_{1|2}, \Sigma_{11|2}) , \quad (3.11)$$

$$\mu_{1|2} = \mu_1 + \Sigma_{12} \Sigma_{22}^{-1} (y_2 - \mu_2) , \quad (3.12)$$

$$\Sigma_{11|2} = \Sigma_{11} + \Sigma_{12} \Sigma_{22}^{-1} \Sigma_{21} . \quad (3.13)$$

Example: Mean and covariance matrices

Considering an archive of 3 observations $\mathbf{y} = (y_1, y_2, y_3)$ at positions λ , and an unknown observation y at position v . Considering a gaussian process \mathcal{GP} on function f described by mean function μ and a kernel K on a multivariate normal distribution, we can write

$$\begin{aligned} p\left(\begin{bmatrix} \lambda \\ v \end{bmatrix}\right) &= \mathcal{N}\left(\begin{bmatrix} \lambda \\ v \end{bmatrix}; \begin{bmatrix} \mu \\ \mu(v) \end{bmatrix}, \begin{bmatrix} \Sigma_{\lambda\lambda}, \Sigma_{\lambda v} \\ \Sigma_{v\lambda}, \Sigma_{vv} \end{bmatrix}\right) \\ &= \mathcal{N}\left(\begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ v \end{bmatrix}; \begin{bmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \\ \mu_v \end{bmatrix}, \begin{bmatrix} \Sigma_{11}, \Sigma_{12}, \Sigma_{13}, \Sigma_{1v} \\ \Sigma_{21}, \Sigma_{22}, \Sigma_{23}, \Sigma_{2v} \\ \Sigma_{31}, \Sigma_{32}, \Sigma_{33}, \Sigma_{3v} \\ \Sigma_{v1}, \Sigma_{v2}, \Sigma_{v3}, \Sigma_{vv} \end{bmatrix}\right), \end{aligned}$$

with $\mu_i = \mu(\lambda_i)$ and $\Sigma_{ij} = K(\lambda_i, \lambda_j)$. By using conditioning, we can deduce the distribution of v according to λ

$$p(y | \mathbf{y}, \lambda, v) = \mathcal{N}(y; \mu_v | \lambda, \Sigma_{vv} | \lambda), \quad \mu_v | \lambda = \mu_v + \Sigma_{v\lambda} \Sigma_{\lambda\lambda}^{-1} (\mathbf{y} - \boldsymbol{\mu}), \quad \Sigma_{vv} | \lambda = \Sigma_{vv} + \Sigma_{v\lambda} \Sigma_{\lambda\lambda}^{-1} \Sigma_{\lambda v}.$$

Exact approximations

In the noise-free case, an observation of f at positions λ reveals the exact value of $\phi = \mathbf{y}$.

Let a function f be modeled with a GP s.t. $p(f) = \mathcal{GP}(f; \mu, K)$, and an archive $\mathcal{D} = (\lambda, \phi)$ of observations sharing a joint distribution with the GP. Using equations 3.11, 3.12 and 3.13, we can condition the GP with \mathcal{D} such that

$$p(f | \mathcal{D}) = \mathcal{GP}(f; \mu_{\mathcal{D}}, K_{\mathcal{D}}), \quad (3.14)$$

$$\mu_{\mathcal{D}}(\lambda) = \mu(\lambda) + K(\lambda, \lambda) \Sigma^{-1} (\phi - \mu), \quad (3.15)$$

$$K_{\mathcal{D}}(\lambda, \lambda') = K(\lambda, \lambda') - K(\lambda, \lambda) \Sigma^{-1} K(\lambda, \lambda'), \quad (3.16)$$

where μ and Σ are respectively the mean vector and covariance matrix of the observations \mathcal{D} . The mean function μ and kernel K are the same as the ones defined in section 3.3.2.

Noisy approximations

In the noisy case, observations $\mathbf{y} = \phi + \varepsilon$ of f at λ are corrupted by homoskedastic noise. The corruption ε is independent of ϕ and is normally distributed around a zero-mean scaled by σ_{noise} . We write the archive of observations $\mathcal{D} = (\lambda, \mathbf{y})$.

Then the marginal distribution of observations \mathbf{y} is given by

$$p(\mathbf{y} | \lambda, \sigma_{\text{noise}}) = \mathcal{N}(\mathbf{y}; \mu, \Sigma + \mathbf{I} \sigma_{\text{noise}}^2). \quad (3.17)$$

By conditioning the GP with noisy observations \mathbf{y} , we obtain the GP posterior

$$p(f | \mathcal{D}, \sigma_{\text{noise}}) = \mathcal{GP}(f; \mu_{\mathcal{D}}, K_{\mathcal{D}}), \quad (3.18)$$

$$\mu_{\mathcal{D}}(\lambda) = \mu(\lambda) + K(\lambda, \lambda) (\Sigma + \mathbf{I} \sigma_{\text{noise}}^2)^{-1} (\mathbf{y} - \mu), \quad (3.19)$$

$$K_{\mathcal{D}}(\lambda, \lambda') = K(\lambda, \lambda') - K(\lambda, \lambda) (\Sigma + \mathbf{I} \sigma_{\text{noise}}^2)^{-1} K(\lambda, \lambda'). \quad (3.20)$$

Now considering a position λ , the potential noisy observation y at λ is given by the

predictive distribution informed by the archive \mathcal{D} and denoted

$$p(y \mid \lambda, \mathcal{D}, \sigma_{\text{noise}}) = \mathcal{N}(y; \mu, \sigma^2 + \sigma_{\text{noise}}^2) , \quad (3.21)$$

with $\mu = \mu_{\mathcal{D}}(\lambda)$ and $\sigma^2 = K_{\mathcal{D}}(\lambda, \lambda)$.

Example: Exact and noisy approximations

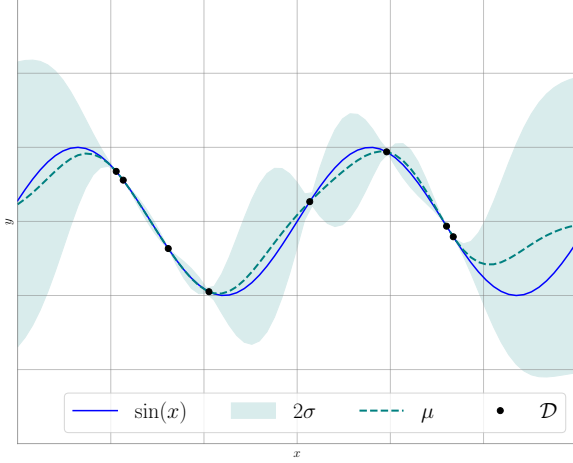


Figure 3.4: GP on a noiseless function.

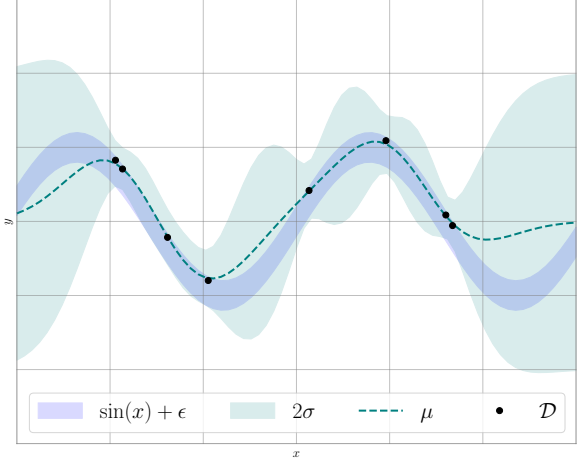


Figure 3.5: GP on a noisy function.

3.3.4 GPs training via marginal likelihood

A GP is a ML model similar to the abstract model $\mathcal{A}_{\theta}^{\lambda}$ described in chapter 2, but applied to continuous optimization. The objective is to fit the GP to a dataset \mathcal{D} , i.e. the archive of the evaluated solutions.

Since, a GP is mainly described by its prior functions μ and K , we can parameterize these functions for a higher flexibility. It allows modeling complex functions dynamically, learning from observations, and reducing the impact of the prior selection. The parameters of GPs are often called *hyperparameters* in the GPs literature, but here we call them *parameters* with the same definition as in chapter 2.

The prior process of a GP parameterized by θ is described as

$$p(f \mid \theta) = \mathcal{GP}(f; \mu(\lambda; \theta), K(\lambda, \lambda'; \theta)) . \quad (3.22)$$

One advantage of GPs is the differentiability of the marginal likelihood w.r.t. the parameters of μ and K . Since, the logarithm function is strictly increasing and we aim at maximizing the likelihood, it is common practice to maximize the log likelihood using

$$\mathcal{L}(\theta) = \log p(y \mid \lambda, \theta) = -\frac{1}{2} [\alpha^{\top} (y - \mu) + \log |V| + n \log 2\pi] , \quad (3.23)$$

with n the number of observations, $V = \Sigma + \sigma_{\text{noise}}^2 I$, and $\alpha = V^{-1}(y - \mu)$. This formula comes from the logarithm of the probability density function of a multivariate normal distribution.

Using the logarithm function also simplifies the computations of the gradient. The partial derivative of $\mathcal{L}(\theta)$ w.r.t. the mean, if not a zero-mean, is given by:

$$\frac{\partial \mathcal{L}}{\partial \theta} = \alpha^{\top} \frac{\partial \mu}{\partial \theta} . \quad (3.24)$$

The partial derivative w.r.t. the variance is written:

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{1}{2} \left[\alpha^\top \frac{\partial V}{\partial \theta} \alpha - \text{Tr} \left(V^{-1} \frac{\partial V}{\partial \theta} \right) \right]. \quad (3.25)$$

If μ and K are differentiable w.r.t. θ , GPs can be trained by usual first-order gradient descent algorithms as described in chapter 1, or second-order quasi-Newton methods such as the L-BFGS algorithm [76].

3.3.5 Numerical instability and computational complexity

In this section, we briefly discuss issues with exact GPs. In practice, efficiently inferring with a GP requires advanced linear algebra to compute matrix-matrix and matrix-vector multiplications. The applied numerical methods need to be accelerated, justifying the use of a dedicated GPU in chapters 5 and 6. We briefly describe how heavy computations are usually performed, but advanced numerical methods for GP are not the focus of this manuscript. More information can be found in [82, 40, 96, 178].

One major drawback of GP-based BO is the scalability of the covariance matrix Σ . Indeed, the size of Σ grows quadratically with the number of observations, so its memory complexity is of $\mathcal{O}(n^2)$. But, its inversion Σ^{-1} is the major challenge. First, the kernel K needs to be symmetric, PD, and is prone to numerical instability. This can result in an unpredicted non-PD matrix during the optimization.

When inferring and training, three operations are costly [82]:

- solving the linear problems, $V^{-1}b$ with b a vector of size n
- computing the log determinant, $\log|V|$
- computing the trace, $\text{Tr}(V^{-1} \frac{\partial V}{\partial \theta})$

Commonly, the Cholesky decomposition is used to compute the inverse of the PD matrix Σ . A PD real matrix A can be decomposed into a lower triangular matrix L and its transpose s.t. $A = LL^\top$, and $A^{-1} = L^{-1}(L^{-1})^\top$. The inversion complexity with Cholesky decomposition is of $\mathcal{O}(n^3)$ with $\frac{1}{2}n^3$ operations [148]. For that reason, it is sometimes preferable to switch to iterative methods, such as conjugate gradient and iterative Lanczos algorithm [82, 178]. It gives an approximation of $b \approx Ay$ without explicitly computing the inverse of Σ . Thus, the computational complexity is reduced at the expense of inference quality.

Due to the numerical instability of iterative methods, a *jitter* can also be added to the diagonal of the covariance matrix. Usually, the jitter is a small value of 10^{-6} .

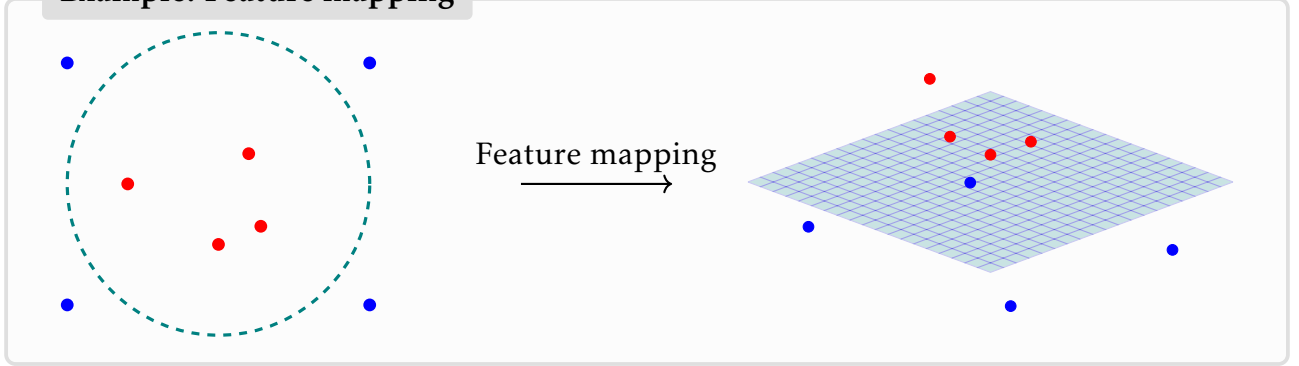
GP-based BO is efficient when the objective function is computationally expensive. However, because of the heavy computations required for inference, the balance between the number of evaluations and time to convergence face to the total optimization budget (e.g. time) is challenging. In the case of a considerable archive of observations, relaxing some GP hypothesis opens the doors to variational and approximate GPs [163, 111]. In variational GPs for instance, a subset of $m \leq n$ points is used for a sparse GP approximation. Typically, Nyström approximations are low-rank approximation methods where the covariance matrix Σ is approximated using a representative subset of the archive, $\mathcal{D}' \subseteq \mathcal{D}$.

3.3.6 Kernel functions

We previously described an abstraction of the mean μ and kernel K functions, both parameterized by θ . We assume a zero-mean $\mu(\lambda) \equiv 0$.

A *feature map* is a function that maps input $\lambda \in \mathbb{R}^d$ to a *feature space* \mathcal{F} ; $\phi : \mathbb{R}^d \rightarrow \mathcal{F}$. Here, the function ϕ has no link to exact observations of a GP. Such an approach can help to classify non-linearly separable data by making them linearly separable into a higher dimensional space.

Example: Feature mapping



A *kernel* is a function relying on the *blessing of dimensionality*; it can be considered a replacement of the dot product [87]. A kernel $K : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ is written:

$$K(\lambda_1, \lambda_2) \triangleq \langle \phi(\lambda_1), \phi(\lambda_2) \rangle, \quad (3.26)$$

with ϕ a nonlinear feature map function and $\langle \cdot, \cdot \rangle$ an inner product. Usually, it is not necessary to explicitly define ϕ and \mathcal{F} . A kernel computes input vectors into a higher dimensional space without explicitly mapping inputs into this higher space.

We can distinguish different types of kernels:

- *Anisotropic stationary* kernels are translation invariant: $K(\lambda_1, \lambda_2) = K_S(\lambda_1 - \lambda_2)$.
- *Isotropic stationary* kernels are functions of the distance between inputs: $K(\lambda_1, \lambda_2) = K_I(\|\lambda_1 - \lambda_2\|)$.
- *Non-stationary* kernels directly depend on the inputs: $K(\lambda_1, \lambda_2)$.

In the context of GPs, kernels are considered a *similarity measure* between points, as depicted in figures 3.6 with isotropic kernels.

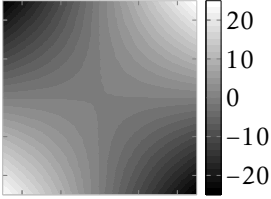
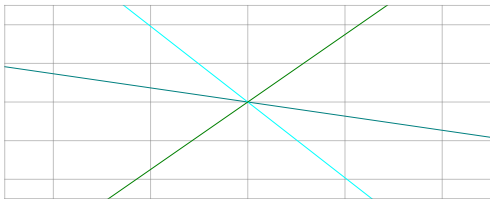
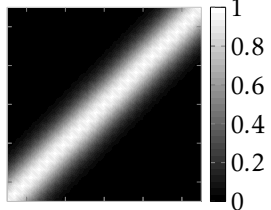
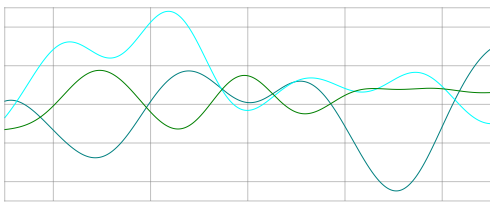
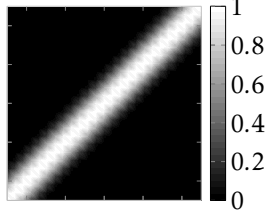
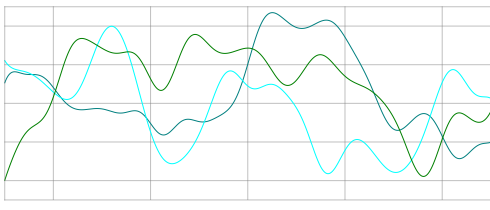
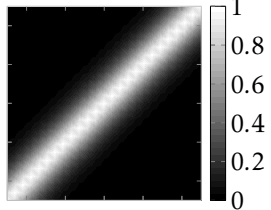
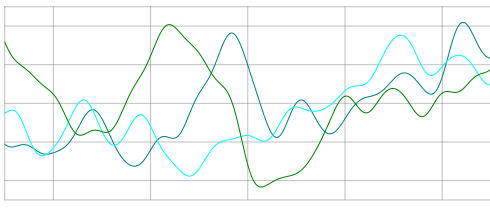
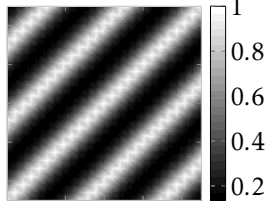
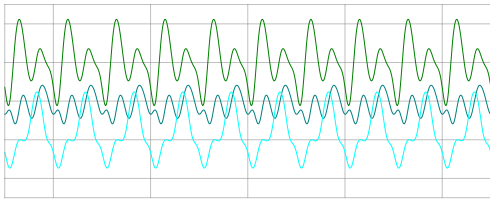
By carefully selecting a kernel, one can inject knowledge about the landscape of the function. To fit prior knowledge to information gathered through iterations, a training phase of the parameters of the GP is necessary.

In table 3.1 a few kernel functions and their parameters are presented. Their effects on the predictive distribution and the covariance matrix are illustrated within the two last columns. Moreover, the impact of the lengthscales and noise on the approximation is depicted in figure 3.7. Finally, kernels can be combined to refine the desired behaviors. One can add or multiply kernels; they can also be decomposed according to dimensions.

3.3.7 Acquisition functions

The principle of BO is to sample more carefully points. To do so, we use a GP shaped with a mean and kernel functions, and trained on an iteratively growing archive of known observations. The acquisition function determines what are the most promising points within the surrogate by balancing the trade-off between exploration and exploitation [84, 282]. In other words, it can determine high- or low-confidence areas of the search space. Instead of directly sampling the posterior process, BO maximizes the acquisition function. By doing so,

Table 3.1: Some kernels and their parameters.

Equation	$K(\lambda, \lambda')$	$p(y) = \mathcal{N}(\lambda; 0, K(\lambda, \lambda'))$
<p>Linear:</p> $\sigma_k \lambda^\top \lambda' ,$ <p>with the variance σ_k.</p>		
<p>RBF:</p> $\exp\left(-\frac{\ \lambda - \lambda'\ _2}{2\ell^2}\right) ,$ <p>with ℓ the lengthscale.</p>		
<p>Matérn3/2:</p> $\left(1 + \frac{\sqrt{3}d}{\ell}\right) \exp\left(-\frac{\sqrt{3}d}{\ell}\right) ,$ <p>with ℓ the lengthscale and $d = \ \lambda - \lambda'\$.</p>		
<p>Matérn5/2:</p> $\left(1 + \frac{\sqrt{5}d}{\ell} + \frac{5d^2}{3\ell^2}\right) \exp\left(-\frac{\sqrt{5}d}{\ell}\right) ,$ <p>with ℓ the lengthscale.</p>		
<p>Periodic:</p> $\exp\left(-2 \sum_i \frac{\sin^2\left(\frac{\pi}{p}(\lambda_i - \lambda'_i)\right)}{\ell}\right) ,$ <p>with p periodic length and ℓ lengthscale.</p>		

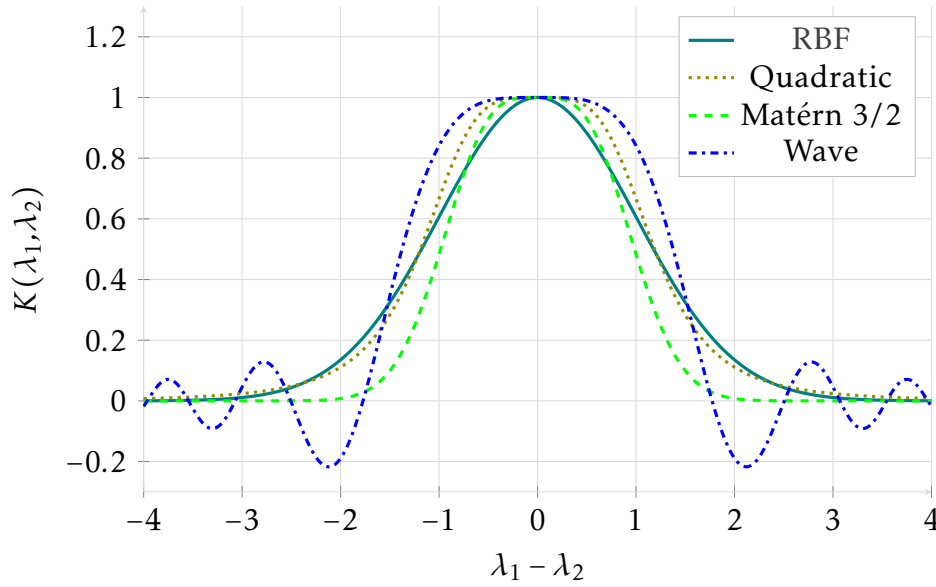


Figure 3.6: Isotropic kernels

BO ensures that new evaluations are sufficiently informative and efficient to progressively refine the GP. Hence, BO is an optimization algorithm with a nested optimization algorithm applied to the acquisition function. Because the cost of evaluating the posterior process requires fewer efforts than the actual objective function, one can optimize the acquisition function with metaheuristics such as GA, gradient-based algorithms if differentiable, or by simply sampling with low discrepancy sequences. The complete workflow of BO is illustrated in figure 3.8 and algorithm 7. We denote the predictions of the predictive distribution from a GP at position λ as $p(y | \lambda, \mathcal{D}, \sigma_{\text{noise}}) = f^*(\lambda)$. Usual acquisition functions are described below.

Probability of improvement

The PI describes the probability of finding a better observation at a position λ compared to the best objective value found so far y_{best} :

$$\begin{aligned} \text{PI}(\lambda; \mathcal{D}) &= \Pr(f^*(\lambda) < y_{\text{best}}) \\ &= \Phi\left(\frac{y_{\text{best}} - \mu_{\mathcal{D}}(\lambda)}{K_{\mathcal{D}}(\lambda, \lambda)}\right) \end{aligned} \quad , \quad (3.27)$$

with $\Phi(\lambda) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\lambda} \exp\left(-\frac{1}{2}t^2\right) dt$, the Gaussian cumulative density function

Expected improvement

The EI gives the expected marginal gain of a sample λ compared to the best objective value found so far y_{best} :

$$\begin{aligned} \text{EI}(\lambda; \mathcal{D}) &= \mathbb{E}[\max(0, f^*(\lambda) - y_{\text{best}})] \\ &= (y_{\text{best}} - \mu_{\mathcal{D}}(\lambda))\Phi\left(\frac{y_{\text{best}} - \mu_{\mathcal{D}}(\lambda)}{K_{\mathcal{D}}(\lambda, \lambda)}\right) + K_{\mathcal{D}}(\lambda, \lambda)\phi\left(\frac{y_{\text{best}} - \mu_{\mathcal{D}}(\lambda)}{K_{\mathcal{D}}(\lambda, \lambda)}\right) \end{aligned} \quad , \quad (3.28)$$

with Φ the Gaussian cumulative density function and ϕ the Gaussian probability density function.

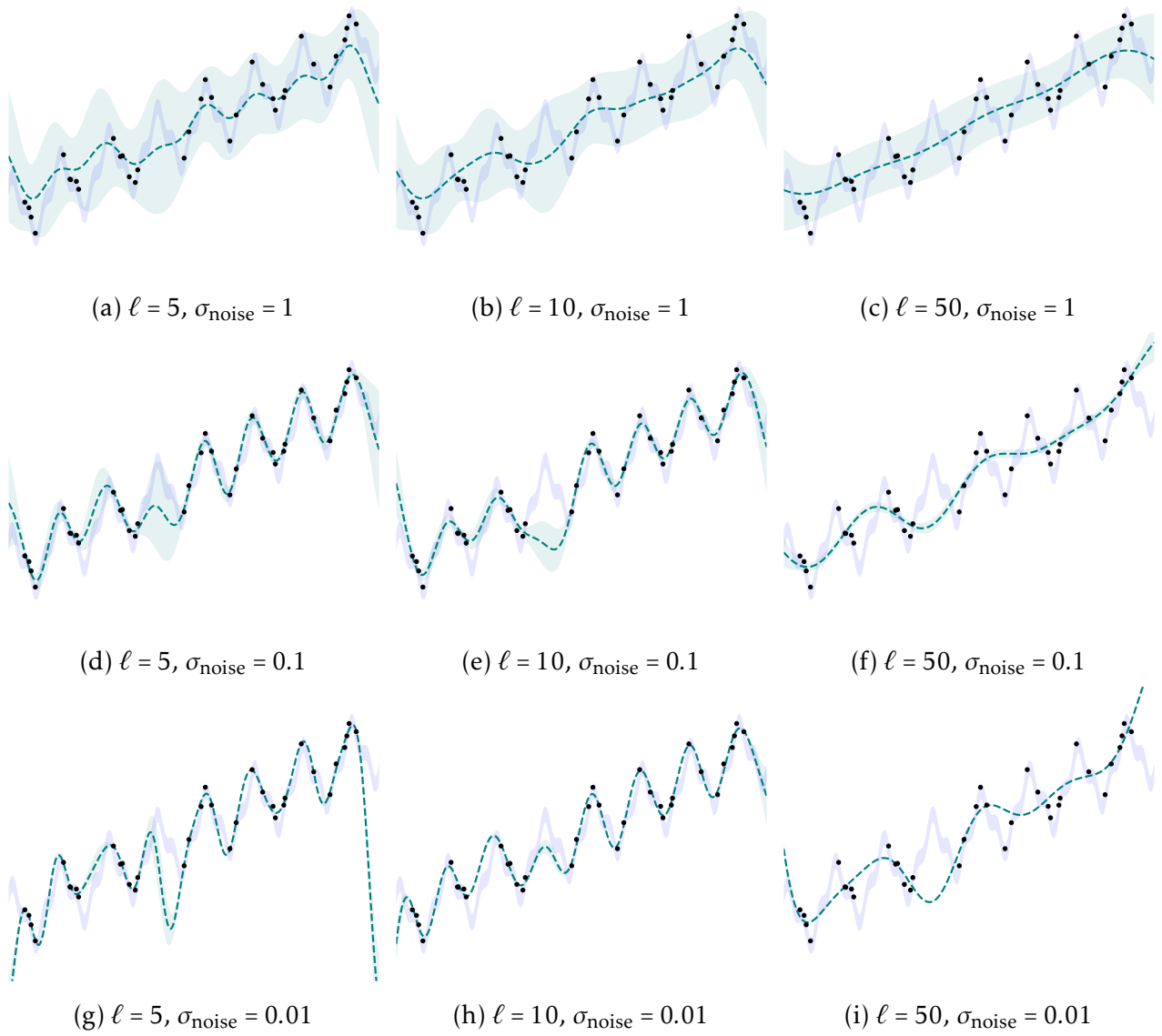


Figure 3.7: RBF with different parameters on $f(\lambda) = \sin^3(\lambda) + 0.1\lambda + \varepsilon$. The dashed line is the mean function.

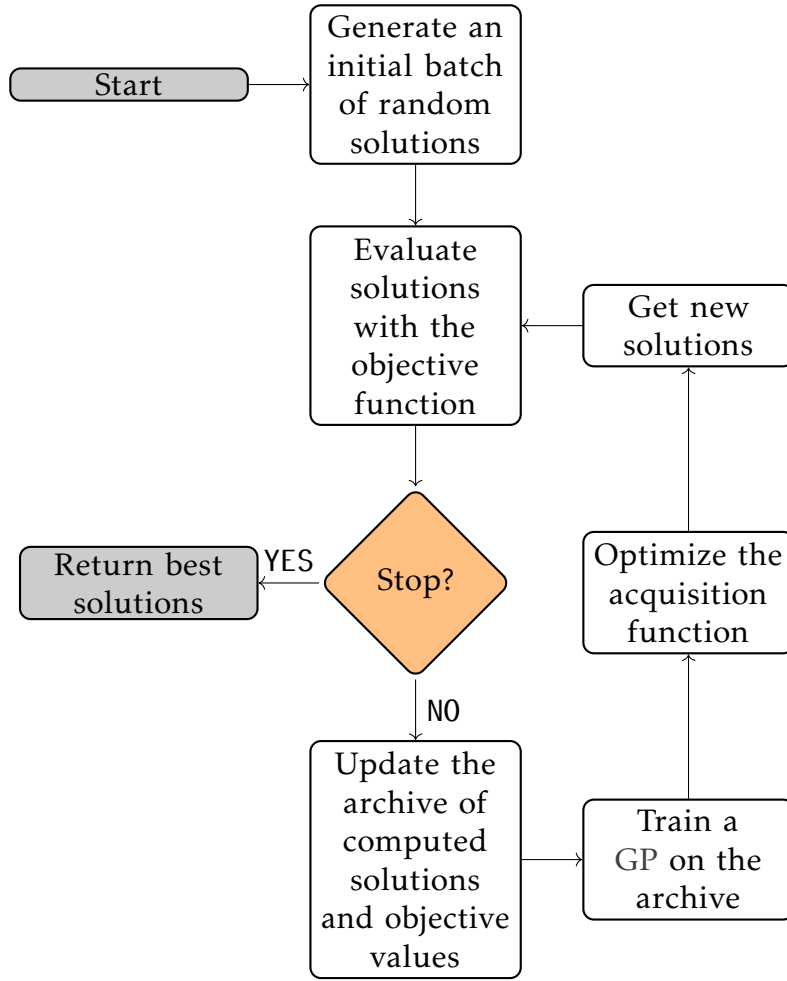


Figure 3.8: Flowchart of BO

Lower and upper confidence bound

The LCB combines the uncertainty and the expected objective value by computing the quantile of the predictive distribution:

$$\begin{aligned}
 \text{LCB}(\lambda; \mathcal{D}) &= q(\pi; \lambda, \mathcal{D}) \\
 &= \inf \{y' \mid \Pr(y \leq y' \mid \lambda, \mathcal{D}) \leq \pi\} \quad , \\
 &= \mu_{\mathcal{D}}(\lambda) - \beta K(\lambda, \lambda)
 \end{aligned} \tag{3.29}$$

with $\pi \in [0, 1]$ the quantile, and β the *exploration* HP describing the exploration-exploitation tradeoff. In the case of a maximization problem, the UCB is given by $\mu_{\mathcal{D}}(\lambda) + \beta K(\lambda, \lambda)$.

3.3.8 Toward multi-fidelity Bayesian optimization for HPO

Usually, in multi-fidelity HPO applied to ML, the fidelity HP to be optimized is the size of the training dataset $\mathcal{D}_{\text{train}}$. To do so, a proportion $\pi_{\text{train}} \in (0, 1]$ of $\mathcal{D}_{\text{train}}$ is extracted s.t. $\pi_{\text{train}} = \frac{|\mathcal{D}'_{\text{train}}|}{|\mathcal{D}_{\text{train}}|}$, with the extracted subset $\mathcal{D}'_{\text{train}} \subseteq \mathcal{D}_{\text{train}}$. A ML model trained iteratively does not require the full training dataset to get an approximation of its performances. For example, the performances of a ANN trained by SGD can be evaluated after each epoch or batch. Thus, in multi-fidelity HPO the algorithm can balance more efficiently the budget attributed to the evaluation of a combination of HPs. The fidelity HP π_{train} has to be tuned by the HPO

Algorithm 7 BO**Inputs:**

1: Ω	<i>Continuous search space</i>
2: f	<i>Objective function</i>
3: $K_{\mathcal{D}}$	<i>Kernel function</i>
4: $\mu_{\mathcal{D}}$	<i>Mean function</i>
5: $\mathcal{D} = \{(\lambda_1, f(\lambda_1)), \dots, (\lambda_n, f(\lambda_n))\}$	<i>Initial Data</i>
6: \mathcal{O}	<i>Inner optimizer/sampler</i>
7: \mathcal{A}_{cqf}	<i>Acquisition function</i>
8: while stopping criterion not met do	
9: $\mu_{\mathcal{D}}, K_{\mathcal{D}} \leftarrow \text{Update}(\mu_{\mathcal{D}}, K_{\mathcal{D}}, \mathcal{D})$	<i>eq. 3.19, 3.20</i>
10: $\mu_{\mathcal{D}}, K_{\mathcal{D}} \leftarrow \text{Train}(\mathcal{GP}(f; \mu_{\mathcal{D}}, K_{\mathcal{D}}), \mathcal{D})$	<i>Fit the GP on the archive with eq. 3.23, 3.24, 3.25</i>
11: $\lambda' \leftarrow \text{Optimize}(\mathcal{A}_{\text{cqf}}(\mu_{\mathcal{D}}, K_{\mathcal{D}}), \mathcal{O})$	<i>eq. 3.21</i>
12: $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\lambda', f(\lambda'))\}$	
return best of $(\lambda^*, f(\lambda^*)) \in \mathcal{D}$	

algorithm. It can be optimized by two main approaches [20]:

- Considering that a higher π_{train} *always* results in a higher objective value.
- Considering that a higher π_{train} *may not necessarily* translate into a higher accuracy, e.g. in case of overfitting.

In this chapter, we focus on the first assumption. The second is discussed in chapter 6. We consider an objective function f , a budget b and a maximum budget b_{max} . An *approximated* and *budgeted* version of f is written $f(\cdot; b)$ s.t. $f(\cdot; b_{\text{max}}) = f(\cdot)$.

Successive Halving

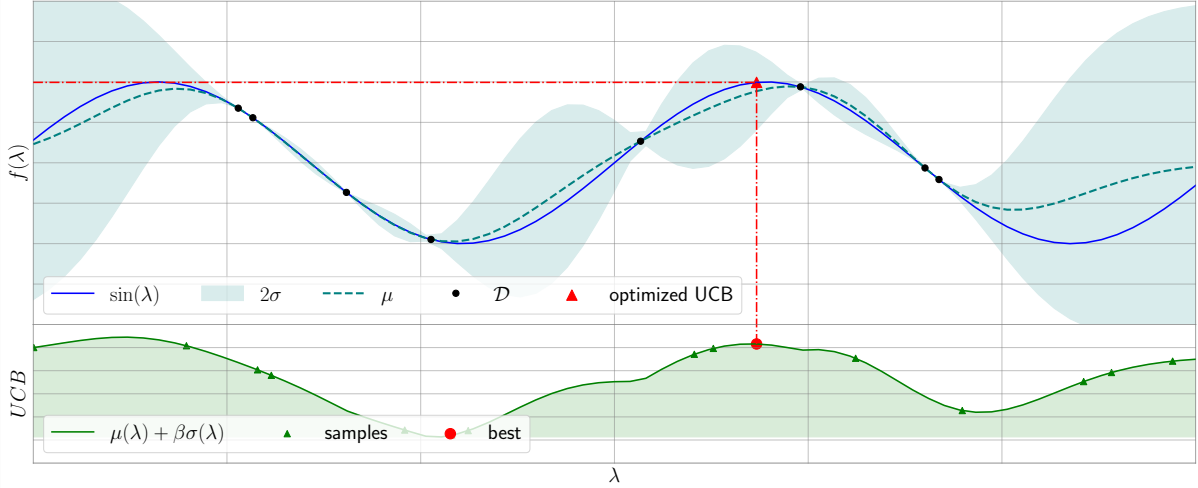
The SH algorithm is a multi-armed bandit algorithm where each arm models a HPs combination λ [132, 158, 69]. If a ML model can be trained iteratively and its performances approximated for each iteration, then SH can be applied with fidelity HPs.

Considering a fixed maximum budget b_{max} , SH balances this budget by favoring promising candidates. At the first iteration, SH starts by evaluating n HPs combinations on the smallest budget. Then, n performances are approximated (e.g. on $\mathcal{D}_{\text{valid}}$) and HPs combinations are sorted by their objective values. Among the n evaluated combinations, a proportion $\frac{1}{\eta}$ of the best ones are selected to be trained further by increasing their dedicated budget by η . The process is repeated until the maximum budget is reached. Thus, SH assigns a higher fidelity to the best HPs combinations, while the algorithm discards the worst ones through iterations. One disadvantage of SH is its strong dependence on the initial sets of n combinations. The process is described in algorithm 8.

Hyperband

Hyperband [158] is an improvement of SH, it is also a multi-armed bandit method for HPO. It uses a nested SH and an adaptive budget to tackle the balance of the budget of SH. Hyperband balances both the following cases:

- A high number of HPs combinations n with low fidelity, i.e. many configurations with low cost.
- A small n with high fidelity, i.e. few configurations with high cost.

Example: UCB on a noiseless function

The UCB was applied to maximize the function $f(\lambda) = \sin(\lambda)$. One can see the posterior process of the GP using an archive of 8 observations (black dots), the optimization of the UCB is performed using random sampling (green triangles). Then the sample maximizing the UCB is selected and evaluated using f (red lines and dots).

Hyperband balances n and b_{\max} by successively applying SH such that each SH run has the same budget. This principle is depicted in algorithm 9.

Tree-structured Parzen Estimator

Considering a minimization problem, TPE is a SMBO algorithm similar to GP-BO [284, 285]. But, instead of modeling $p(y | \lambda, \mathcal{D})$, the TPE algorithm models $p(\lambda | y, \mathcal{D})$. The algorithm models the probability to observe a HPs combination for a given objective value. To do so, TPE uses KDE as surrogates, such as Gaussian mixture models.

The archive \mathcal{D} of observations is partitioned into two groups, i.e. two densities:

$$p(\lambda | y, \mathcal{D}) = \begin{cases} g(\lambda | \mathcal{D}_g), & \text{if } y < y^* \\ b(\lambda | \mathcal{D}_b), & \text{if } y \geq y^* \end{cases}, \quad (3.30)$$

with g the surrogate on *good* observations, b the surrogate on *bad* observations. Observations are partitioned within the *good* archive $\mathcal{D}_g = (\lambda_g, y_g) = \{(\lambda, y) \in \mathcal{D} | f(\lambda) < y^*\}$ and *bad* archive $\mathcal{D}_b = (\lambda_b, y_b) = \{(\lambda, y) \in \mathcal{D} | f(\lambda) \geq y^*\}$. The value y^* is determined according to a quantile γ s.t. $p(y < y^*) = \gamma$. The HP γ is fixed by the user, ensuring that a minimum number of observations is used for \mathcal{D}_g . Both KDEs are given by:

$$\begin{aligned} p(\lambda | \mathcal{D}_g) &= w_g p(\lambda) + w_g^\top K(\lambda, \lambda_g | \ell_g) \\ p(\lambda | \mathcal{D}_b) &= w_b p(\lambda) + w_b^\top K(\lambda, \lambda_b | \ell_b) \end{aligned}, \quad (3.31)$$

with $p(\lambda)$ a prior on the distribution (usually Gaussian) of λ weighted by w_g and w_b . The kernel K is different from the one previously described for GPs. In KDE, a kernel is used to estimate the per-dimension underlying probability distribution of observations λ . A kernel has dimension-wise lengthscales (ℓ_g, ℓ_b) . Good and bad observations are weighted by w_g and w_b . Different kernels can be used for different dimensions.

Algorithm 8 SH

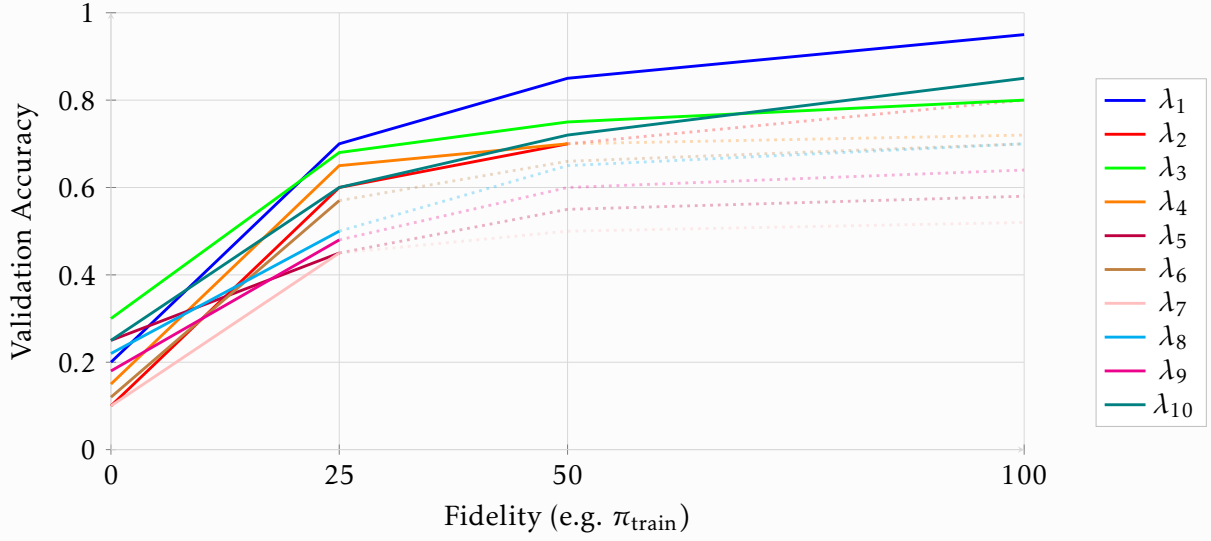
Inputs:

1: Ω	<i>Continuous search space</i>
2: n	<i>Size of the initial batch</i>
3: b_0	<i>Initial budget</i>
4: b_{\max}	<i>Maximum budget</i>
5: η	<i>Scaling</i>
6:	
7: $b \leftarrow b_0$	
8: $L \leftarrow \mathbf{Random}(\Omega, n)$	<i>Sample n configurations</i>
9: while $b \leq b_{\max}$ do	
10: $y \leftarrow f(L; b)$	<i>Evaluate L with budget b</i>
11: $L \leftarrow \text{Top}_k\left(L, y, \left\lfloor \frac{ L }{\eta} \right\rfloor\right)$	
12: $b \leftarrow \eta b$	
return best of (L, y)	

Algorithm 9 Hyperband

Inputs:

1: Ω	<i>Continuous search space</i>
2: b_0	<i>Initial budget</i>
3: b_{\max}	<i>Maximum budget</i>
4: η	<i>Scaling</i>
5:	
6: $s_{\max} \leftarrow \lfloor \log_{\eta} \frac{b_{\max}}{b_0} \rfloor$	
7: for $s \in \{s_{\max}, s_{\max} - 1, \dots, 0\}$ do	
8: $n \leftarrow \lceil \frac{s_{\max} + 1}{s + 1} \eta^s \rceil$	
9: $L \leftarrow \mathbf{Random}(\Omega, n)$	<i>Sample n configurations</i>
10: $b \leftarrow \eta^s b_{\max}$	
11: while $b \leq b_{\max}$ do	<i>Inner bracket (SH)</i>
12: $y \leftarrow f(L; b)$	<i>Evaluate L with budget b</i>
13: $L \leftarrow \text{Top}_k\left(L, y, \left\lfloor \frac{ L }{\eta} \right\rfloor\right)$	
14: $b \leftarrow \eta b$	
return best of among all (L, y)	

Example: SH

The transparent dotted lines are discarded HPs combinations. In this example, $\eta = 2$ and the fidelity hyperparameter is the proportion π_{train} of the training dataset $\mathcal{D}_{\text{train}}$.

Unlike trained kernels for GPs, in TPE the lengthscales ℓ are approximated by heuristics [285] such as the Scott's rule for the univariate Gaussian kernel in BOHB:

$$\forall i \in \{g, b\}, \forall d \in \llbracket 0, D \rrbracket, \ell^{(d)} \approx 1.059 N_i^{-1/5} \min \left(\sigma, \frac{Q1(\lambda_i^{(d)}) - Q3(\lambda_i^{(d)})}{1.34} \right), \quad (3.32)$$

where Q1 and Q3 are the first and third quartiles of observations $\lambda_i^{(d)}$ at dimension d , with $N_i = |\mathcal{D}_i|$.

There is no former training of models g and b using an optimizer as for GPs or ANNs. The lengthscales are determined by a heuristic, and the weights (w_g, w_b) are determined by a uniform heuristic [284]:

$$\begin{aligned} \forall w \in w_g, \quad w &= \frac{1}{N_g + 1} \\ \forall w \in w_b, \quad w &= \frac{1}{N_b + 1} \end{aligned} \quad (3.33)$$

TPE solves a EI maximization problem to efficiently sample HPs combinations s.t.:

$$\text{EI}(\lambda) \propto \frac{g(\lambda | \mathcal{D}_g)}{b(\lambda | \mathcal{D}_b)}. \quad (3.34)$$

So, the next point to evaluate with the actual objective function is selected by sampling HPs combinations s , according to the distribution $g(\lambda | \mathcal{D}_g)$ with w_g and ℓ_g fitted on \mathcal{D}_g . Then, the most promising sample s^* is selected with

$$s^* \in \underset{s}{\operatorname{argmax}} \frac{g(s | \mathcal{D}_g)}{b(s | \mathcal{D}_b)}.$$

The whole TPE process is described in algorithm 10.

Algorithm 10 TPE**Inputs:**

1: Ω	<i>Continuous search space</i>
2: f	<i>Objective function</i>
3: $\mathcal{D} = \{(\lambda_1, f(\lambda_1)), \dots, (\lambda_k, f(\lambda_k))\}$	<i>Initial dataset of k randomly computed solutions</i>
4: γ	<i>Split quantile</i>
5:	
6: while stopping criterion not met do	
7: $\mathcal{D}_g, \mathcal{D}_b \leftarrow \text{Split}(\mathcal{D}, \gamma)$	
8: $w_g, w_b \leftarrow \text{Update}(w_g, w_b, \mathcal{D}_g, \mathcal{D}_b)$	
9: $\ell_g, \ell_b \leftarrow \text{Update}(\ell_g, \ell_b, \mathcal{D}_g, \mathcal{D}_b)$	
10: $g(\lambda \mathcal{D}_g) \leftarrow w_g p(\lambda) + w_g^\top K(\lambda, \lambda_g \ell_g)$	
11: $b(\lambda \mathcal{D}_b) \leftarrow w_b p(\lambda) + w_b^\top K(\lambda, \lambda_b \ell_b)$	
12: Sample ($p(s) = g(\lambda \mathcal{D}_g)$)	
13: $s^* \in \underset{s}{\operatorname{argmax}} \frac{g(s \mathcal{D}_g)}{b(s \mathcal{D}_b)}$	
14: $\mathcal{D} \leftarrow \mathcal{D} \cup (s^*, f(s^*))$	
return best of \mathcal{D}	

Bayesian Optimization HyperBand

BOHB is a state-of-the-art HPO algorithm [69, 20]. It combines Hyperband and TPE to better sample HPs combinations. Hence, BOHB relies on Hyperband to determine the number of solutions to evaluate, and the budget allocated to these solutions. The algorithm combines multi-fidelity and SMBO applied to expensive HPO of ML models. The pseudocode of the sampling used in BOHB is given in algorithm 11; it replaces line 10 of algorithm 9. All evaluations are kept on all budgets, so to enrich the two KDEs computed on the biggest budget. Here the archive of points \mathcal{D} can be grouped by a budget \mathbb{B} , $\mathcal{D}_{\mathbb{B}}$. An archive of observed solution with budget \mathbb{B} , $\mathcal{D}_{\mathbb{B}}$ can be split into *good* and *bad* archives, resp. $\mathcal{D}_{\mathbb{B}g}$ and $\mathcal{D}_{\mathbb{B}b}$.

Algorithm 11 BOHB (sampling)**Inputs:**

1: Ω	<i>Continuous search space</i>
2: f	<i>Objective function</i>
3: $\mathcal{D} = \{(\lambda_1, f(\lambda_1)), \dots, (\lambda_k, f(\lambda_k))\}$	<i>Dataset of n computed solutions</i>
4: γ	<i>Split quantile</i>
5: ρ	<i>Probability of random runs</i>
6: N_{\min}	<i>Minimum number of observations</i>
7:	
8: $\mathbb{B} \leftarrow \max\{\mathbb{B}; \mathcal{D}_{\mathbb{B}} \geq N_{\min} + 2\}$	
9: if $(\mathcal{U}(0, 1) < \rho) \vee (b = \emptyset)$ then	
10: return Random (Ω)	
11: else	
12: $\mathcal{D}_{\mathbb{B}g}, \mathcal{D}_{\mathbb{B}b} \leftarrow \text{Split}(\mathcal{D}_{\mathbb{B}}, \gamma, \mathbb{B})$	
13: $w_{\mathbb{B}g}, w_{\mathbb{B}b} \leftarrow \text{Update}(w_{\mathbb{B}g}, w_{\mathbb{B}b}, \mathcal{D}_{\mathbb{B}g}, \mathcal{D}_{\mathbb{B}b})$	
14: $\ell_{\mathbb{B}g}, \ell_{\mathbb{B}b} \leftarrow \text{Update}(\ell_{\mathbb{B}g}, \ell_{\mathbb{B}b}, \mathcal{D}_{\mathbb{B}g}, \mathcal{D}_{\mathbb{B}b})$	
15: $g(\lambda \mathcal{D}_{\mathbb{B}g}) \leftarrow w_{\mathbb{B}g}p(\lambda) + w_{\mathbb{B}g}^T K(\lambda, \lambda_{\mathbb{B}g} \ell_{\mathbb{B}g})$	
16: $b(\lambda \mathcal{D}_{\mathbb{B}b}) \leftarrow w_{\mathbb{B}b}p(\lambda) + w_{\mathbb{B}b}^T K(\lambda, \lambda_{\mathbb{B}b} \ell_{\mathbb{B}b})$	
17: Sample ($p(S) = g(\lambda \mathcal{D}_{\mathbb{B}g})$)	
18: $s^* \in \underset{s}{\operatorname{argmax}} \frac{g(s \mathcal{D}_{\mathbb{B}g})}{b(s \mathcal{D}_{\mathbb{B}b})}$	
19: return s^*	

3.4 Parallel HPO

In practice, efficiently tackling NP-hard problems involves parallel and distributed optimization. Distributed optimization is challenging, one has to handle memory, homogeneous and heterogeneous computational resources (CPUs, GPUs), network communications, and fault tolerance. The memory can be shared or distributed, involving different parallel programming approaches (OpenMP or MPI). A system can be accelerated or not, for example via GPUs. These methods significantly reduce computation time and enhance scalability. Parallel optimization has several advantages [259]. The main goal of parallelization is to accelerate the optimization process, which can also be used to solve problems that cannot be tackled sequentially in a reasonable amount of time. It can also improve the performances of the optimized solution and the robustness of the algorithm.

In the following sections, we will explore how parallel and distributed optimization can accelerate HPO of previously described algorithms.

Parallelization of metaheuristics can be divided into three levels of granularity [259, 260]:

- **Algorithmic:** Here, different instances of algorithms are executed in parallel. They can be independent or cooperative.
- **Iteration:** This model describes a parallelization of a single iteration of a sequential algorithm.
- **Solution – Objective function:** Here, the evaluation of a single evaluation is accelerated via parallelization of its internal computations.

In this manuscript, we mainly focus on the *iteration* level. Indeed, the evaluation of SNNs is expensive, although GPU-accelerated, and we have access to a few numbers of accelerators

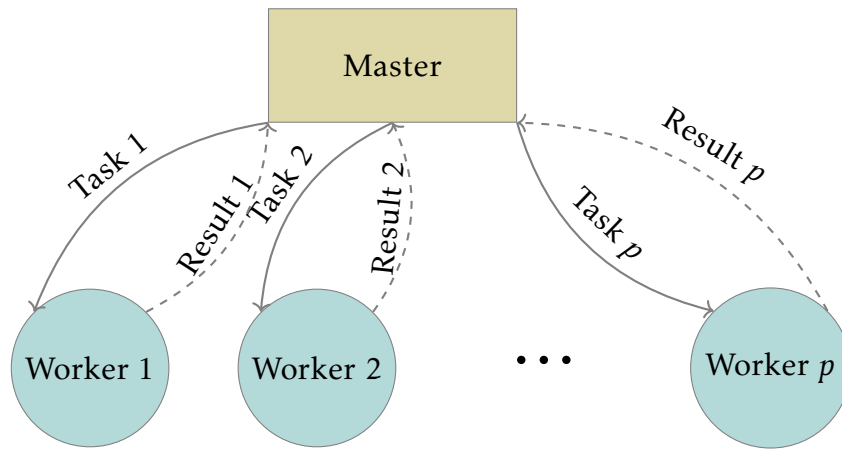


Figure 3.9: Master–worker parallelization approach.

(about 16). Thus, running multiple algorithmic instances in parallel is not reasonable regarding the cost of the problem and limited time budget (see chapters 5 and 6).

Considering the limited number of resources and the cost of the objective function (from minutes to hours), communication time is negligible and the master–worker approach is relevant. This approach is illustrated in figure 3.9. In case of communication bottleneck, other approaches such as work-stealing [53] can be considered.

Two common approaches are synchronous and asynchronous parallelization, each with distinct advantages and tradeoffs [259, 8]. In synchronous parallelization, all worker nodes evaluate candidate solutions simultaneously, and the optimization process waits for all evaluations to complete before proceeding to the next iteration. This approach ensures consistency across updates, but may suffer from idle time if some evaluations take longer than others, leading to inefficiencies in resource utilization. This is illustrated in figure 3.10.

In contrast, asynchronous parallelization allows the optimization process to proceed as soon as any worker finishes its evaluation, without waiting for all others to achieve. This method tends to be more efficient, as it reduces idle time and can adapt to varying evaluation times, but it may introduce inconsistencies in updates as newer solutions may be based on outdated information from slower workers.

3.4.1 Parallel Grid Search and Random Search

The parallelization of GS is straightforward [299, 20, 290] as all points to be evaluated are known in advance and so independent of each other. GS is a “brute-force” – exhaustive search; all points have to be evaluated. Thus, the master-worker approach is not even necessary, as points can be partitioned in advance among processes. In the case of high variability of computation time of the objective function, a master-worker or work-stealing approach could better balance the workload.

RS is even more straightforward than GS [299, 20, 290]. Since all points are i.i.d. and there is no peculiar structure of the bounded search space, RS can be parallelized at the algorithm level by running within each process an instance of RS.

3.4.2 Parallel Simulated Annealing

Parallelizing SA is challenging as it is a sequential S-metaheuristic [259], also named trajectory-based algorithm [7]. The first synchronous approach would be to sample several neighbors of the current solution instead of one. It allows the master-worker approach because a set of solutions is evaluated in parallel. At the algorithm level, the multi-start

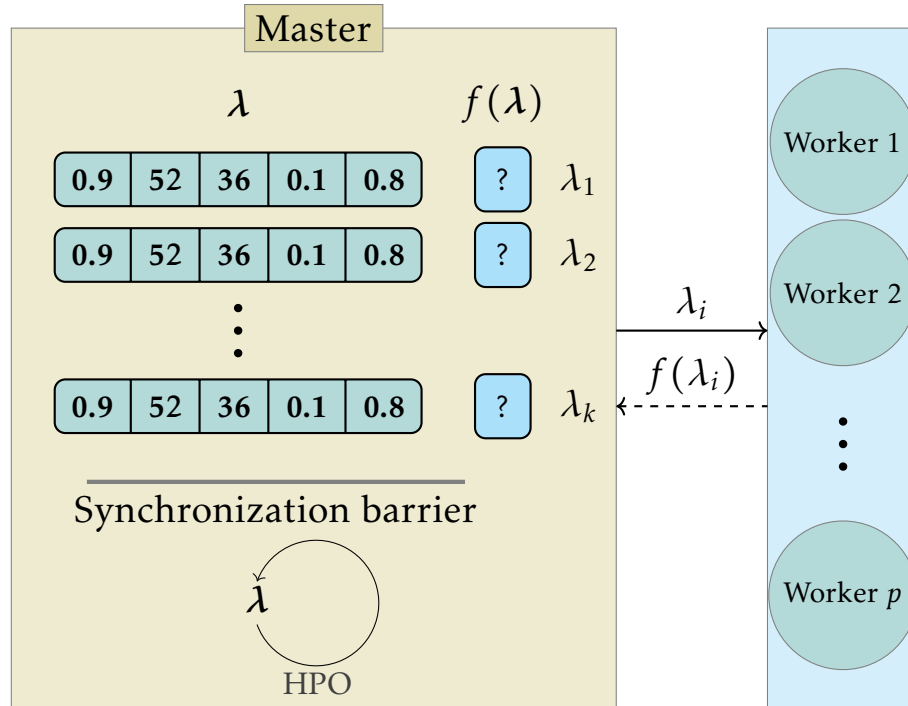


Figure 3.10: Illustration of synchronous parallelization. The communication and evaluations between the master and workers can be done asynchronously. The next iteration of the HPO algorithm is computed once all k evaluations are done.

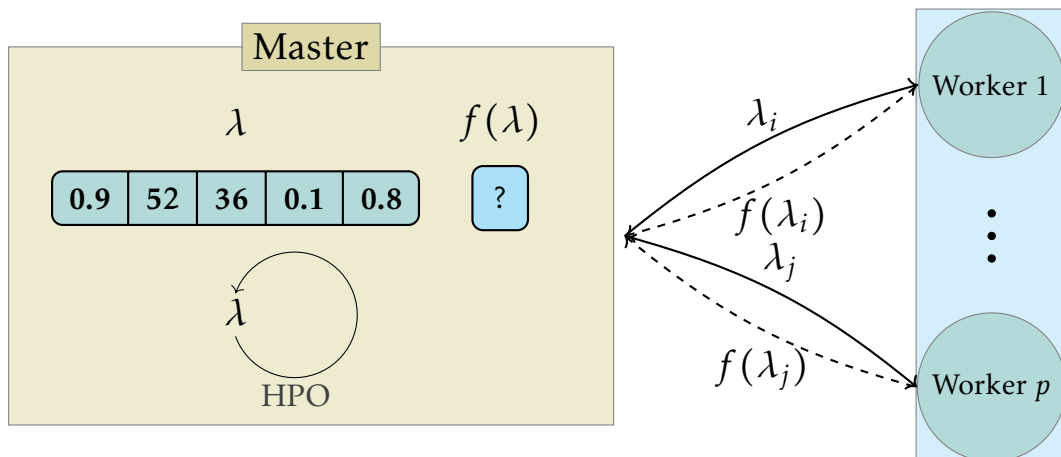


Figure 3.11: Illustration of asynchronous parallelization. There is no proper iteration of HPO. The algorithm asynchronously receives and generates *on-demand* solutions.

strategy consists in launching multiple SA from various starting solutions. Concerning the asynchronous approach, the synchronization barrier of the strategy described earlier could be relaxed. It allows not to wait for the full evaluation of the neighborhood to compute the next *walk*, and eventually to move to a newly better solution from a previous neighborhood [259].

3.4.3 Parallel Genetic Algorithm

Parallelizing GA is also straightforward. It is classified as a P-metaheuristic in [259], and its parallelization has been well documented since the beginning of the 2000s [8, 6, 176, 7]. The sequential nature of the GA operators complicates the parallelization at the iteration level. Within the master-worker approach, the master handles operators and the workload balance, while workers only compute objective values from received solutions.

GA can be synchronously parallelized by sending solutions from the current population to workers. Once all solutions are evaluated, the master can compute the next population using the evolutionary operators. The behavior of the algorithm is not altered with such an approach, which essentially improves the efficiency when the objective function is computationally expensive [7]. To parallelize asynchronously a GA, the steady-state approach alters Line 11 of algorithm 6 by only replacing one individual from the current population. Steady-state GA generates only one offspring on the workers' demand [259, 8].

Other approaches partition the population so it can be distributed among processes. Sparse communications between processes, e.g. migration in island models [7], allows limited interactions between subpopulations. A finer-grain approach, known as cellular algorithms, introduces neighborhood in terms of communications. It maps solutions to processes [259, 7, 8], and solutions can only be altered by its neighborhood. One key component of a cellular algorithm is a sufficiently large population size to ensure diversity, which is an issue considering our expensive HPO problem.

3.4.4 Parallel Gaussian Process-based Bayesian Optimization

GP-BO is hardly parallelizable, as it is an inherently single-solution and sequential algorithm [290].

Instead of sampling a single solution from the acquisition function, batch BO allows sampling a batch of various and optimized points. One of the most popular synchronous batch-BO is the q EI [93, 287, 280, 32] allowing to sample batches of q promising solutions.

Considering an archive \mathcal{D} and a batch of q solutions $\lambda_b = \{\lambda_1, \dots, \lambda_q\}$, q EI can be expressed as:

$$qEI(\lambda_b) = \mathbb{E} \left[\max \left(\max_{\lambda \in \lambda_b} \lambda - y_{\text{best}}, 0 \right) \right]. \quad (3.35)$$

In practice, to prevent computing the equation 3.35 with Monte-Carlo simulations [93], a complex closed-form of this equation allows for fast computations of the q EI [32].

Other approaches exist to sample a batch of sufficiently diversified solutions from the acquisition function.

Local penalization [97] penalizes already selected and maximized samples from the acquisition function. To do so, we suppose that the objective function is Lipschitz continuous (discussed in chapter 4), and we use the output of the GP to estimate two parameters defining the penalization around the current maximized sample. Thus, by iteratively maximizing and penalizing a single solution from the acquisition function, this method builds a batch of various solutions. By penalizing pending evaluations, local penalization can also be used in asynchronous parallelization of batch-BO [78].

Another approach for asynchronous parallelization, known as fantasizing [250], consists in using the GP posterior distribution to impute values of pending evaluations. By conditioning these *fantasized* solutions with the current GP we obtain a fantasy, i.e. a possible future GP if the approximations of the pending evaluations were true. The acquisition function needs to be revisited and redefined as a costly integral. This can be a drawback for large batches of solutions or when fast optimization of the acquisition function is required.

TS is one of the easiest and most straightforward ways to generate batches of solutions and asynchronously parallelize BO [138, 78]. The approach comes from the bandit and reinforcement learning communities. It replaces the acquisition function by the maximization of a batch of candidates sampled from the GP predictive distribution. By using TS no acquisition function is optimized, which makes the approach much cheaper than fantasies and local penalization [78]. The stochasticity of TS ensures an inherent diversity within the batches of candidates.

3.4.5 Parallel multi-fidelity HPO

SH is *embarrassingly parallel* as well as GS and RS [157]. A synchronous way of parallelizing SH and Hyperband is to simply distribute the survivor HPs combinations among processes, and sample a new bracket when some processes become idle [158]. Or, one can also parallelize SH at the algorithmic level by running multiple independent instances. The ASHA algorithm proposes an asynchronous version of SH by allowing incorrect *promotion* of a solution to be evaluated with the next fidelity [157]. If a process has a *non-promotable* solution, then it is discarded, and a new configuration is randomly sampled with a minimum fidelity. Thus, ASHA allows continuously having solutions being evaluated and promoted according to the current state of the algorithm.

In BOHB [69], the parallelism can be tackled within the TPE and Hyperband parts. Similarly to the TS approach for BO, the diversity of a batch of candidates from the EI maximization of TPE is ensured by leveraging the size of the batch of candidate solutions.

The Hyperband algorithm can be parallelized by launching multiple iterations (line 8 of algorithm 9) at the same time, and parallelizing at the iteration level the inner SH loop. So, configurations can be sampled on-demand by multiple parallel Hyperband instances using TPE. Once an inner SH ends, another is launched. This approach is well suited for the master-worker paradigm, as the archive \mathcal{D} of points is shared among all parallel Hyperband runs. Hence, when asking for a batch of solutions to the master, the sampled batch will be up-to-date using all currently evaluated solutions among all parallel processes.

Partition-based global optimization

Continuous relaxation allows tackling the hyperparameter optimization problem by usual continuous Bayesian optimization. This formulation expands the range of algorithm families. One of them, inherited from the divide-and-conquer and branch-and-bound approaches, structures the search space by hierarchically and iteratively decomposing it into progressively smaller subspaces. In the following chapter, we delve into a generalization of some of these approaches, which we name FBD algorithms. We present a unified framework based on a self-similar geometrical object, a fractal, to decompose the search space. This framework is made of four other components: a tree search, a scoring, exploration, and exploitation methods. Within the first experimental section, several algorithms are instantiated and compared using the COCO benchmark. The experimental results emphasize the modularity of the framework and behaviors of such algorithms in terms of efficiency and scalability. Under our framework, we then propose a new FBD algorithm relying on Latin Hypercube Sampling.

4.1 Fractal-based decomposition algorithms

Throughout the following development, we consider a maximization problem, as described in chapters 2 and 3. Conversely to the previous chapter, we do not currently consider expensive functions in terms of computation time, memory, budget, etc. Still, the problem is black-box, non-linear, non-convex, and derivative-free. The optimized function can be written as follows: $f : \Omega \subset \mathbb{R}^d \rightarrow \mathbb{R}$. The search space $\Omega \subset \mathbb{R}^d$ is a compact subset of the n -dimensional continuous set \mathbb{R}^d . Then the maximization consists in finding the global optimum x^* and is defined by:

$$x^* \in \underset{x \in \Omega}{\operatorname{argmin}} f(x) . \quad (4.1)$$

In this section, we investigate a specific class of optimizers, FBD algorithms, inspired by *divide-and-conquer* strategies, which utilize a hierarchical decomposition of the search space. Two families of algorithms partitioning the search space to optimize a function are studied and helped to design the following framework. FBD models heuristic approaches, such as the FRACTOP [46] or FDA [192]. FBD also models some algorithms from Lipschitzian optimization with DIRECT [135, 136] and optimistic optimization with the SOO [188] algorithm.

Existing mathematical frameworks, such as the MSO, already generalize some aspects of Lipschitzian and optimistic optimization partition-based optimizers. FBD aims to include improper partition with overlapping subspaces, allowing to model a broader category of

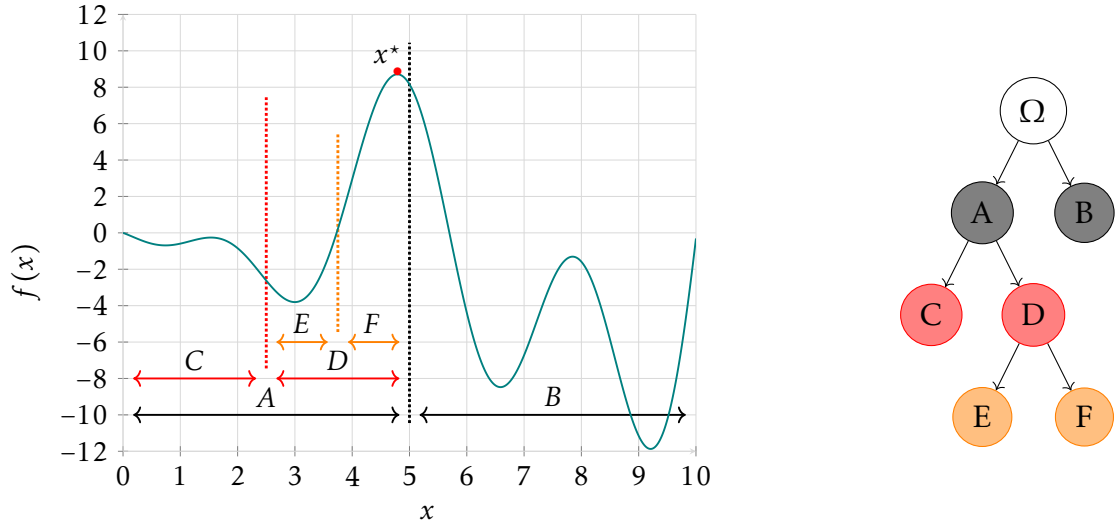


Figure 4.2: Example of a tree-like data structure for bisection-based FBD algorithms, applied to $f(x) = -x \sin(x) - x \cos(2x) - \frac{x}{6}$

algorithms, including heuristics such as FDA. To do so, in FBD we define a *fractal* as a high dimensional *geometrical object* structuring the search space. It is also a *subset* of an initial search space or of another fractal, and a *node* of a tree data structure.

FBD is structured around five search components: fractal, tree search, scoring, exploration, and exploitation. The *fractal geometrical object* partitions the search space in a structured way. Such partitioning approaches, as described in MSO [58], can be modeled via a tree-like data structure. This partitioning process is illustrated in figure 4.2, with a 1- d bisection. This figure highlights the close relation between the partition of the search space and the tree structure. However, the dynamic of the hierarchical partition, i.e. how to decompose the search space while optimizing, is handled by the four other components. This dynamic is first described by the *tree search* algorithms, which at each iteration selects a set of nodes, fractals, to further decompose. This selection relies on a criterion defined as the *scoring* component and describing the probability of a subspace to contain the global optimum. To evaluate how promising a fractal is, an *exploration* component, restrained to this fractal, samples solutions and their objective value. Finally, to overcome a lack of intensification, an *exploitation* component only bounded to the whole search space, is applied to the most promising fractal at a given iteration.

Such a compartmentalized approach allows translating the search components into software bricks within a generic Python framework named *Zellij*¹. Zellij allows the design of new optimization algorithms by combining search components and also the implementation of alternative components thanks to a high level of abstraction. Comparatively to EA [301], by doing so, Zellij opens doors to hyper-heuristic and automatic design of FBD algorithms [254].

4.2 Preliminaries

In this section, we introduce a brief description of the basic concepts and search components from FBD algorithms.

¹The Zellij software is available under GitHub <https://github.com/ThomasFirmin/zellij>.

Definition 10 (Search space). *Let us define a continuous search space Ω of dimension n as a closed and bounded subset of a metric space:*

$$\Omega = L \times U = \prod_{i=1}^n [l_i, u_i] , \quad (4.2)$$

with $L, U \subset \mathbb{R}^n$, the infima and suprema, such that $\forall i \in [1, \dots, n]$, $l_i < u_i$ and $\forall x \in \Omega$, $\forall i \in [1, \dots, n]$, $l_i \leq x_i \leq u_i$.

Assumption 1. *For convenience, we consider a measure set (Ω, Σ, μ) , where Σ is a σ -algebra on the power set of Ω and μ is a valid measure.*

Definition 11 (Fractal). *A fractal F :*

1. *is a closed and bounded subset of a search space : $F \subseteq \Omega$, i.e. it is compact regarding the Heine-Borel theorem.*
2. *is a non-empty set: $F \neq \emptyset$.*
3. *is measurable: $F \in \Sigma$.*
4. *is a child of an ancestor fractal (parent) denoted \mathcal{A}_F .*
5. *has K children, $\mathcal{C}_F \triangleq \{c_1, \dots, c_K\}$. With a finite $K \in \mathbb{N}$.*
6. *is a node of a rooted tree structure: $F \in \mathcal{T}$.*
7. *is the root of \mathcal{T} if $F = \Omega$.*
8. *is characterized by a set of m properties computed using inheritance of \mathcal{A}_F : $P(F, \mathcal{A}_F) \triangleq \{p_1(F, p_1(\mathcal{A}_F, \dots)), \dots, p_m(F, p_m(\mathcal{A}_F, \dots))\}$.*

We first consider the case of a *partial partition*, as defined in the MSO framework. We temporarily define $\beta(F)$ the boundaries of a fractal S . Then, two fractals $F_i, F_j \in \Sigma$ are said to be *disjoint at the borders*, denoted by the $\dot{\cap}$ symbol, if and only if:

$$F_i \dot{\cap} F_j = \emptyset \iff F_i \cap F_j = \beta(F_i) \cap \beta(F_j) \quad (4.3)$$

This property ensures that two fractals are considered disjoint even with a common boundary. Later in this chapter, the notion of boundaries will be discarded to the profit of measures of overlapping and coverage of a set of fractals.

As defined in MSO [58], a collection of disjoint and non-empty subsets of Σ is called a *partial partition*.

Definition 12 (Partial partition). *Considering a measure space (Ω, Σ, μ) , with Ω a search space as defined in 10, and $\Sigma = \mathcal{P}(\Omega)$. A finite collection of fractals, indexed by I , $\mathcal{F} \subseteq \Sigma = \{F_i\}_{i \in I}$ is a *partial partition of Ω* if all paired elements are disjoint at the borders:*

$$\forall i, j \in I, i \neq j \implies F_i \dot{\cap} F_j = \emptyset \quad (4.4)$$

The union of a *partial partition* of Ω is called its *support*. Thus, a *partition* of Ω is a *partial partition* for which the union of its elements is equal to Ω . The search space Ω is the *support* of this *partition*.

Definition 13 (Partition). *Considering a measure space (Ω, Σ, μ) , with Ω a search space as defined in 10, and $\Sigma = \mathcal{P}(\Omega)$. A partial partition, indexed by I , $\mathcal{F} = \{F_i\}_{i \in I}$ is a partition of Ω if its support is Ω :*

$$(\forall i, j \in I, i \neq j \implies F_i \cap F_j = \emptyset) \wedge (\bigcup \mathcal{F} = \Omega) \quad (4.5)$$

We temporarily consider that children \mathcal{C}_F of a fractal F , are a partition of F , i.e. $\bigcup \mathcal{C}_F = F$. As a consequence, a FBD algorithm iteratively builds a partition. To do so, a fractal from a partition can also be partitioned. Thus, a set $\mathcal{G} \subseteq \Sigma$ is a *hierarchy* on Ω if paired elements are disjoint at the borders or nested.

Definition 14 (Hierarchy). *Considering a measure space (Ω, Σ, μ) , with Ω a search space as defined in 10, and $\Sigma = \mathcal{P}(\Omega)$. A collection $\mathcal{G} \subseteq \Sigma$, indexed by I such that $\mathcal{G} = \{F_i\}_{i \in I}$, is a hierarchy on Ω if :*

$$\forall i, j \in I, i \neq j \implies (F_i \cap F_j = \emptyset) \vee (F_i \subseteq F_j) \vee (F_j \subseteq F_i) \quad (4.6)$$

Definition 15 (Leaves). *Considering a measure space (Ω, Σ, μ) , with Ω a search space as defined in 10, and $\Sigma = \mathcal{P}(\Omega)$. A hierarchy on Ω , $\mathcal{G} = \{F_i\}_{i \in I}$. The collection of leaves $\mathcal{L} \subseteq \mathcal{G}$ are fractals that have no child:*

$$\mathcal{L} = \{l \in \mathcal{G} \mid \mathcal{C}_l = \emptyset\} \quad (4.7)$$

Definition 16 (Hierarchy of partitions). *Considering a measure space (Ω, Σ, μ) , with Ω a search space as defined in 10, and $\Sigma = \mathcal{P}(\Omega)$. A hierarchy $\mathcal{G} = \{F_i\}_{i \in I}$ on Ω , indexed by I , is also a hierarchy of partitions if the union of its leaves \mathcal{L} w.r.t 15 is Ω :*

$$(\forall i, j \in I, i \neq j \implies (F_i \cap F_j = \emptyset) \vee (F_i \subseteq F_j) \vee (F_j \subseteq F_i)) \wedge \bigcup \mathcal{L} = \Omega \quad (4.8)$$

It is important to notice the relations between previously described concepts. The leaves \mathcal{L} of a hierarchy of partitions on \mathcal{G} , are also a partition on \mathcal{G} , i.e. $(\forall F_i, F_j \in \mathcal{L}, F_i \neq F_j \implies F_i \cap F_j = \emptyset) \wedge (\bigcup \mathcal{L} = \Omega)$.

A FBD algorithm iteratively and dynamically builds a hierarchy \mathcal{G} on Ω . As illustrated in figure 4.2, a hierarchy \mathcal{G} on Ω , is then modeled by a rooted tree \mathcal{T} , where nodes are fractals. Thus, fractals can be indexed by their level, i.e. depth, within the tree. For a given fractal F , a property is dedicated to the level, $p_{\text{level}}(F, \mathcal{A}_F) = l$, with $1 \leq l \leq D$, where $l = 1$ is the level of the root Ω , and D is the maximum depth of the tree. Considering the K children of \mathcal{A}_F , these can be indexed by $1 \leq i \leq K$. Another property defines the indexing, $p_{\text{index}}(F, \mathcal{A}_F) = i$.

Minimal properties of fractals are given by, $P(F, \mathcal{A}_F) = \{p_{\text{level}}(F, \mathcal{A}_F), p_{\text{index}}(F, \mathcal{A}_F)\}$. As in [58], one can group fractals by properties. For example, $\{F \in \mathcal{L} \mid p_{\text{index}}(F, \mathcal{A}_F) = 1\}$, groups all fractals that are the first child of their ancestor.

To simplify notations, we write a fractal $F_{(l,i)}$, a node from a tree \mathcal{T} of level l , child number i of $\mathcal{A}_{F_{(l,i)}}$.

At each iteration, a FBD algorithm, selects at least one leaf from \mathcal{L} to further partition it by creating its children. Such a process is called a *refinement*. Given a partition, e.g. leaves \mathcal{L} of a hierarchy of partitions \mathcal{G} , a refinement consists in building a finer grained partition \mathcal{L}' by selecting $F \in \mathcal{L}$ and building $\mathcal{L}' = \{\mathcal{L} \setminus F\} \cup \{\mathcal{C}_i \in \mathcal{C}_F \mid 1 \leq i \leq K\}$

Definition 17 (Refinement). *Considering a measure space (Ω, Σ, μ) , with Ω a search space as defined in 10, and $\Sigma = \mathcal{P}(\Omega)$. Given, a partition $\mathcal{F} \subseteq \Omega$, a selected fractal F and its K children \mathcal{C}_F . A refinement \mathcal{F}' of \mathcal{F} is written:*

$$\mathcal{F}' \triangleq (\mathcal{F} \setminus F) \cup \mathcal{C}_F \quad (4.9)$$

Figure 4.2 illustrates a *hierarchy of partitions* on Ω , with $\mathcal{G} = \{\text{A}, \text{B}, \text{C}, \text{D}, \text{E}, \text{F}\}$. Indeed, $\bigcup \mathcal{G} = \Omega$. For instance, nodes **E** and **F** are *disjoint at the borders*, i.e. $\text{E} \cap \text{F} = \emptyset \iff$

$\mathbf{E} \cap \mathbf{F} = \beta(\mathbf{E}) \cap \beta(\mathbf{F}) = \{3.75\}$, and *nested* within its *ancestor*; *parent* node, $\mathcal{A}_{\mathbf{E}} = \mathcal{A}_{\mathbf{F}} = \mathbf{D}$, i.e. $(\mathbf{E} \subset \mathbf{D}) \wedge (\mathbf{F} \subset \mathbf{D})$. The fractal \mathbf{C} is a leaf of \mathcal{G} because $\mathcal{C}_{\mathbf{C}} = \emptyset$. Moreover, the collection of leaves $\mathcal{L} = \{\mathbf{B}, \mathbf{C}, \mathbf{E}, \mathbf{F}\}$ is a *hierarchy of partitions* on (Ω) , because $\bigcup \mathcal{L} = \Omega$.

Considering \mathbf{F} , with $K = 2$ and $\mathcal{C}_{\mathbf{F}} = \{\mathbf{G}, \mathbf{H}\}$. The refinement \mathcal{L}' of \mathcal{L} would be, $\mathcal{L}' = \{\mathbf{B}, \mathbf{C}, \mathbf{E}, \mathbf{G}, \mathbf{H}\}$, and $\mathcal{G}' = \{\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}, \mathbf{E}, \mathbf{F}, \mathbf{G}, \mathbf{H}\}$.

4.3 Background and related works

Since the 1980s, the first mathematical frameworks [215, 118] generalizing algorithms decomposing the search were proposed. One of the first is LO, conjointly suggested in 1972 by Piyavskii [216] and Shubert [248]. In this section, we introduce various frameworks, from the DTB by Sergeyev [239], to the MSO framework of Al-Dujaili [58]. Then, popular FBD algorithms are presented, coming from various communities, as the metaheuristics or LO families.

4.3.1 Piyavskii and Shubert algorithm

Initially, the Piyavskii-Shubert algorithm [216, 248], is a deterministic algorithm from the LO family. It maximizes 1-dimensional continuous search spaces w.r.t. 10. The objective function is assumed to be blackbox and the only assumption is Lipschitz continuity, i.e. a certain smoothness ensuring limited variations of the objective function. This assumption is illustrated in figure 4.4.

A Lipschitz continuous function should not violate the following constraint within the search space Ω :

$$\forall x, x' \in \Omega, |f(x) - f(x')| \leq L|x - x'|, \quad (4.10)$$

where L is a positive constant. There are several benefits to making this assumption. It facilitates proving convergence towards a global optimum. Additionally, the algorithm is deterministic and requires only setting, or knowing L .

In dimension 1, the LO algorithm is a FBD algorithm as it iteratively partitions the closed interval $\Omega = [l, u]$. By replacing x' from equation 4.10, Lipschitz continuity can be applied to

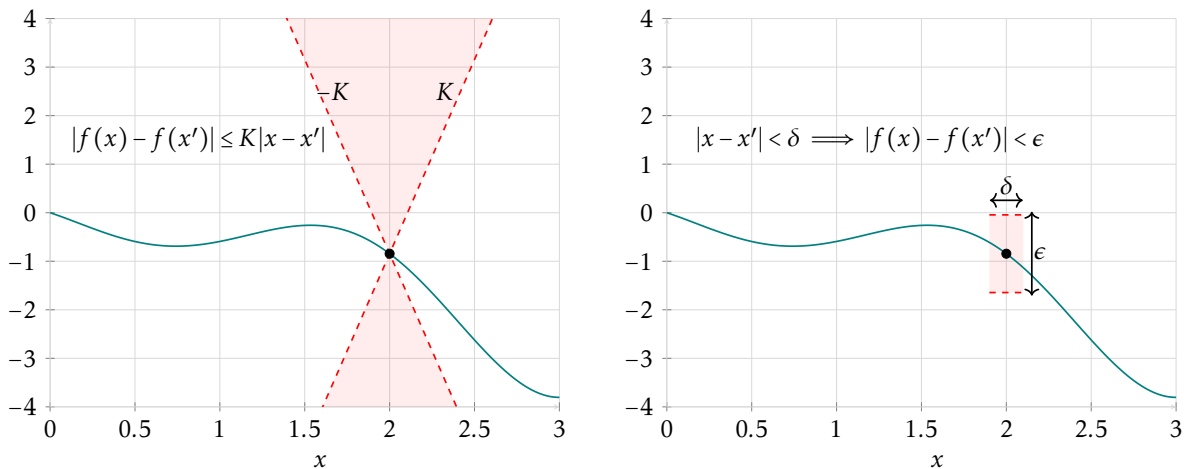


Figure 4.4: Examples of Lipschitz continuity (left) and uniform continuity (right). If the curve crosses the red zone, then the function is not Lipschitz or uniformly continuous.

any closed interval $[a, b] \subseteq \Omega$:

$$\begin{aligned} f(x) &\geq f(a) - L(x - a) \\ f(x) &\geq f(b) + L(x - b) \end{aligned} \tag{4.11}$$

For maximization, at each iteration, for all leaves of the hierarchy of partitions, the next points to evaluate are computed using $X(a, b, f, L) = \frac{a+b}{2} + \frac{f(b)-f(a)}{2L}$. A prediction on $f(X(a, b, f, L))$ is then computed using $B(a, b, f, L) = \frac{f(a)+f(b)}{2} + L(b-a)$. The interval with the lowest prediction, i.e. B-score, is then selected to be further refined. The combination of $X(a, b, f, L)$ and $B(a, b, f, L)$ allows building the cone that might contain the function. The Piyavskii-Shubert algorithm estimates an interval-wise Lipschitz constant by successively refining intervals (fractals) into smaller ones. Three iterations of the algorithm are illustrated in figure 4.6.

However, this algorithm has several drawbacks. Indeed, one has to assume the constant L to be known. If L is mistuned, then the algorithm could converge slowly. The second drawback is its poor scalability. Indeed, within a n – dimensional search space, the algorithm needs to compute all 2^n vertices. These issues will be solved in 1993 with the DIRECT [135] algorithm.

4.3.2 DIRECT: Dividing Rectangles

The DIRECT algorithm, is also part of the LO family. It is initially a modification of the Piyavskii-Shubert algorithm [248, 216], solving its scalability and unknown L issues. It also assumes that the objective function is Lipschitz-continuous, with a positive constant L w.r.t. 4.10.

Conversely to the Piyavskii-Shubert algorithm, the partitioning of DIRECT consists in sampling the center of all sub-hyperrectangles. Thus, at the first iteration, only the center of Ω is sampled, instead of its 2^d vertices. On top of that, DIRECT assumes L to be unknown and iteratively estimates it.

To do so, equation 4.11 is modified to focus the search around the center c of the subspaces, instead of its boundaries:

$$\begin{aligned} f(x) &\geq f(a) - L(x - c), \quad \text{if } x \leq c, \\ f(x) &\geq f(b) + L(x - c), \quad \text{otherwise} . \end{aligned} \tag{4.12}$$

Then, the upper bound for a given subspace F can be written: $B(F, c, f, L) = f(c) + L\sigma(F)$, where σ measures the size of F .

To build the hierarchy of partitions, the algorithm refines fractals by a series of trisections on the longest sides. The choice of the cutting sides relies on the quality of previously sampled centers. The partitioning strategy is illustrated in figure 4.8.

One can notice that in the computation of 4.12 and B , L still must be known. To solve these issues, DIRECT introduces the concept of POH, a strategy that selects the most promising fractals to be refined [135]. POH can be considered the computation of a Pareto front between the size of hyper-rectangles and their fitness values, i.e. a convex hull, preventing a lack of exploration due to over-dividing small subspaces. The POH is illustrated in figure 4.9, where the Pareto front is depicted in dotted blue. Non-POH are symbolized by square-red points.

Despite the improvement of Piyavskii-Shubert algorithm, DIRECT has several drawbacks. There is a poor balance between exploration and exploitation, and the computational complexity is subject to the *curse of dimensionality*. Many modifications of DIRECT have been proposed in the literature to counterbalance some of these drawbacks [136]. For instance, a

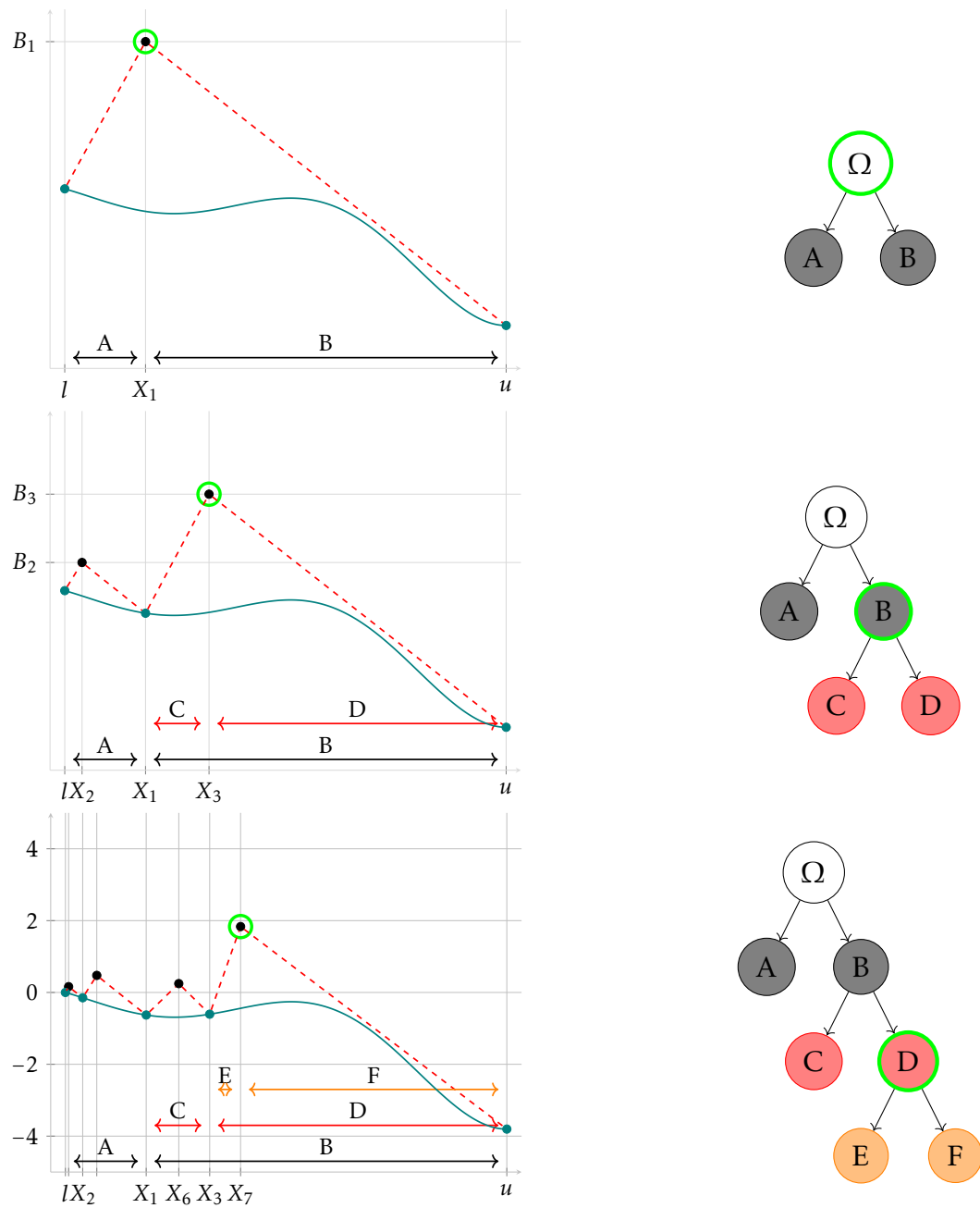


Figure 4.6: Three iterations of the Shubert algorithm (left), with the resulting rooted tree (right).

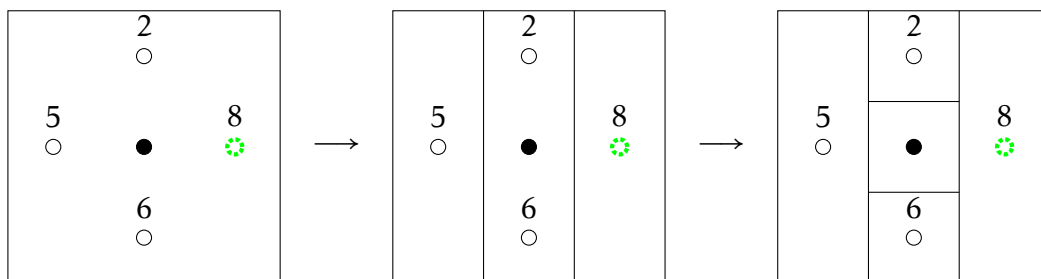


Figure 4.8: A single refinement of a hyper-rectangle in DIRECT. (the best point is represented in dotted green)

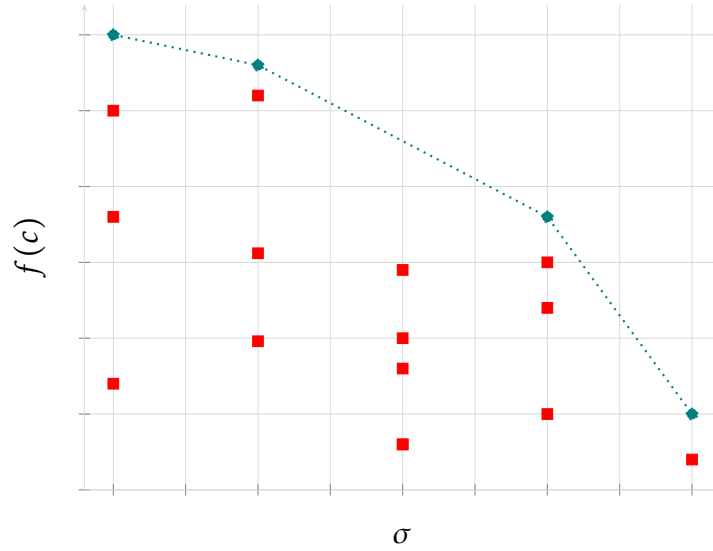


Figure 4.9: Potentially optimal rectangles according to their size (σ) and their center ($f(c)$). Teal-diamond dots are POH.

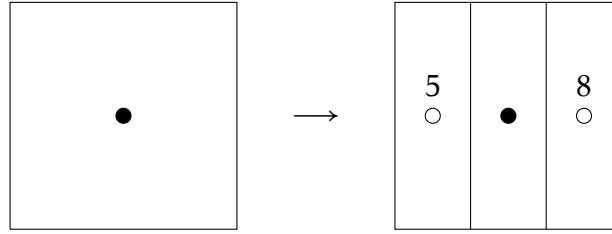


Figure 4.11: A single refinement of a hyper-rectangle in SOO.

DIRMIN [164] tackles the lack of local optimizer by applying a truncated Newton method after a certain number of iterations. The DIRECT-L algorithm introduced by Gablonsky [81], reduces the *global drag*, a phenomenon consisting in over-refinement of local solutions. Another version, DIRECT-R [73], improves the exploration-exploitation tradeoff by making the parameter ϵ adaptive. Additionally, modifying the geometry of the partition itself can help improve DIRECT depending on the problem. For instance, by using simplices [306] or Voronoï cells [162].

All these modifications try to overcome DIRECT's lower performances in high dimensions, low convergence rate when trapped by local optima, or lack of a local optimizer.

4.3.3 DOO, SOO, NMSO: Optimistic Optimization

DOO and SOO generalizes the DIRECT algorithm [188, 189]. These algorithms make a strong assumption on the existence of a semi-metric l . This assumption simplifies the Lipschitz-continuous property by assuming a *local smoothness* around the global optimum x^* [188] :

$$\forall x \in \Omega, f(x^*) - f(x) \leq l(x^*, x) .$$

DOO is used when l is known; otherwise, SOO is more adapted. The strength of these algorithms, is their low number of parameters and the proof of a convergence bound. Both algorithms are deterministic. At each iteration and at each level of the partition tree, the best fractal is selected according to the evaluation of a representative solution inside it (e.g. center). Both algorithms use a trisection to partition Ω , but applied differently compared to DIRECT, as seen in figure 4.11

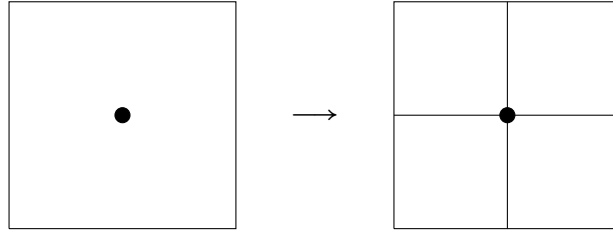


Figure 4.13: A single refinement of a hyper-cube in FRACTOP.

In SOO, the balance between exploration and exploitation relies on the tree search algorithm. It consists in selecting, within the tree and in descending order, the best fractal at each level only if no other fractals from previous levels are better. DOO and SOO are linked to the multi-armed bandit problem and Monte Carlo Tree Search [189]. This is why these algorithms apply *the optimism in the face of uncertainty* principle. It consists of favoring poorly explored areas where the uncertainty is the highest. In addition, a stochastic version, inspired by the UCB-1 algorithm [174], named Sto-SOO has been designed for noisy loss function, where each fractal has to be evaluated multiple times [270].

NMSO [57], based on SOO, modifies the tree search algorithm, i.e. the fractal selection strategy, focusing on depth-wise expansions, and where fractals can become *unpromising* under certain criteria. It performs well on black-box optimization with expensive functions and a low budget. It uses a trisection, and more generally, k -section, to partition the space into hyper-rectangular subspaces. NMSO computes the center of each hyper-rectangle as its representative point. NMSO, compared to SOO, tends to be more exploitive by favoring deep trees. However, compared to DIRECT and SOO, NMSO has more parameters, a total of four, impacting its sensitivity, the exploration-exploitation tradeoff, and the partition size.

4.3.4 FRACTOP, FDA and PolyFRAC: FBD metaheuristics

FRACTOP is one of the first metaheuristic based on fractal decomposition [46]. It uses hypercubes (figure 4.13) to decompose the search space, a genetic algorithm (see 6) to explore each fractal, and simulated annealing (see 5) to exploit a promising fractal.

The algorithm implements a fuzzy measure, named *belief*, to determine the fitness of a fractal. Let's define $p_{\text{score}}(F, \mathcal{A}_F) = \text{Belief}(F, \mathcal{A}_F, X_F, \hat{x})$, a property of a fractal F defining its score, with X_F the points sampled in F and \hat{x} the current best solution found:

$$\text{Belief}(F, \mathcal{A}_F) = \gamma \text{Belief}(\mathcal{A}_F, \mathcal{A}_{\mathcal{A}_F}) + (1 - \gamma) \frac{f(X_F)}{f(\hat{x})} \exp\left(1 - \frac{f(X_F)}{f(\hat{x})}\right), \quad (4.13)$$

with $\gamma \in [0, 1]$ the influence of the ancestor on the child's score, and $\text{Belief}(\Omega, \emptyset) = 0$.

One major drawback of this algorithm is its poor scalability in high dimensions due to an exponential complexity (2^d) to build a d -dimensional partition of equal-sized hypercubes.

The FDA metaheuristic partly solves the *curse of dimensionality* problem of FRACTOP [192] at the expense of full coverage of the search space, and so to a convergence toward the global optimum. Indeed, the *improper* partitioning of Ω is made by $2d$ hyper-spheres, as seen in figure 4.15.

FDA has a lower partitioning complexity compared to FRACTOP, but at the cost of overlapping fractals due to an inflation ratio α . This ratio partially reduces the lack of space coverage implied by the hyperspheres decomposition. FDA quickly acquires information about fractals by applying the PHS algorithm to each of them. It computes three points: the center of the hypersphere and two opposite points equidistant to the center. To score an

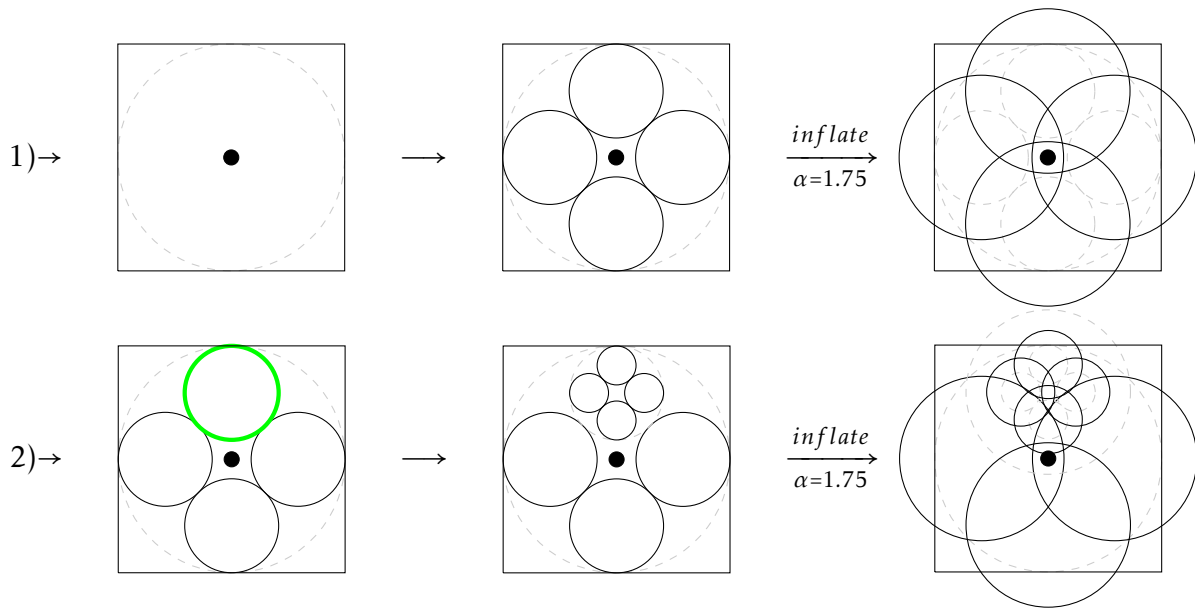


Figure 4.15: Two refinements in FDA.

explored fractal, FDA uses the DTTB solution found so far.

To counterbalance the lack of coverage, and improve the intensification phase of FDA, an algorithm only bounded by Ω and named ILS, allows reaching some uncovered space. The ILS is similar to a coordinate descent algorithm starting from the deepest and most promising fractal.

The polyFRAC algorithm is a modification of FDA [142]. Rather than using hyperspheres, polyFRAC partitions the space using Voronoï cells. The algorithm computes an approximation of these fractals, since finding the vertices (i.e. d -faces) of a Voronoï cell is a complex procedure that does not scale in dimension.

4.3.5 Frameworks for partition-based algorithms

One of the first frameworks proposed in 1987 by Horst and Tuy [118, 119], precursors of branch-and-bound [166], describes a common structure. The authors give a proof of convergence for different partitioning algorithms, such as the Shubert algorithm [248] or ones within the Pintér's class [215]. They suggested a *conceptual algorithm*, including a definition of a *partition* of a subset $F \subseteq \Omega$ based on boundaries of F ; $\beta(F)$. They proposed an indirect definition of a *refinement*, i.e. a nested partition, using two sets, \mathcal{P} containing selected subsets to be refined and \mathcal{R} the remaining ones. This approach was later improved with the *Divide-the-Best Algorithm* (DBA) framework [239]. DBA suggests a higher level of abstraction by including parameters of subsets, additional information about evaluations (derivatives), and an improved abstraction of the selection strategy, allowing to refine multiple subsets at the same iteration. Later, in [58], a K -ary tree structure is proposed to model a hierarchical partition of the search space. The authors proposed the *Multi-Scale Optimization* (MSO) framework and an analysis of its convergence using the Hölder condition, boundedness and sphericity of subspaces.

Other more specific studies [255, 254], proposed to break down DIRECT-based algorithms into a *partitioning strategy*, i.e. the partition and evaluation of a subset, and *selection scheme*, i.e. our tree search component. The authors proposed a different combination of three selection and partitioning strategies. The three investigated selection strategies are an improvement of the POH from DIRECT, an aggressive one, and a two-step-based Pareto selection. Concerning the partition, the selection can be combined with bisections, 1-dimensional trisections (e.g.

SOO) and d -dimensional trisections (e.g. DIRECT). The proposed algorithms can sample the center of the hyperrectangles, two diagonal points, or two vertices. By combining these components, they designed 12 different DIRECT-type algorithms and showed that a proper combination of algorithmic components results in different behaviors and performances on specific problems and situations.

These works and frameworks pave the way toward hyperheuristics and automated design of optimization algorithms based on the partition of the search space. Decomposing this family of algorithms into search components might result in similar works about hyperheuristics applied to population-based metaheuristics [301].

4.4 Zellij: An algorithmic framework for FBD algorithms

This section introduces the basic concepts and search components of the *Zellij* framework. Conversely to previously described works, we cannot propose a theoretical convergence within our framework, as we cannot ensure, for all cases described in section 4.4.1, that the global optima x^* is always reachable. Modeling heuristics comes at the expense of ensuring a proof of global convergence.

In Zellij, a *fractal* F , is a self-similar geometrical object that does not depend on any information besides the nature of its ancestor. The inheritance from the parent \mathcal{A}_F during the creation of F is the only acceptable transfer of information between fractals. This is modeled by the properties $P(F, \mathcal{A}_F)$. For example, the $\text{Belief}(F, \mathcal{A}_F)$ w.r.t. 4.13, can be partly computed as the score of the parent $\text{Belief}(\mathcal{A}_F, \mathcal{A}_{\mathcal{A}_F})$ can be inherited, but the current best solution found \hat{x} must be shared across fractal.

Thus, to prevent combinatorial explosion in terms of memory, a child F can only compute its properties by considering \mathcal{A}_F and cannot append $P(\mathcal{A}_F, \mathcal{A}_{\mathcal{A}_F})$, and so on. This is one of the drawback of the inheritance-only property, as we cannot model algorithms based on shared information between fractals. For example, the Multilevel Coordinate Search algorithm [124] cannot be modeled. Indeed, points are sampled after the creation of a fractal and at the borders of hyperrectangles, so they can belong to different fractals, involving the transfer of information between already created fractals. Therefore, the number of properties m is globally fixed for all fractals. The inheritance prevents communication overhead during parallelization of any fractal-based algorithm.

In 4.2, we described *proper* partitions and hierarchy of partitions on Ω . To model heuristic and metaheuristic approaches, we need a less restrictive definition of a partition w.r.t. 13. To that end, in the following lines we will consider *covers* and *improper K -covers*. Five properties characterize an improper K -cover on Ω : partition size, building complexity, coverage, overlap, and memory complexity (see Table 4.1 and Figure 4.21). Unlike existing mathematical frameworks [239, 58, 118], some parts of fractals can be outside their parents and can overlap. Two fractals with overlapping boundaries can be considered a special case of negligible overlapping. Therefore, in Zellij, we do not define the boundaries of a fractal F , often written as a function of F : ∂F [118], $\delta(F)$ [239], $\beta(F)$ [58]...

In Zellij, exploring and exploiting the rooted tree defined in 4.2 is essential to thoroughly selecting the next fractals to refine. This is done by a *tree search algorithm*, sometimes referred to as *selection strategy* in the literature. Many tree search algorithms can be used, from MCTS [189] to usual search algorithms, such as *Best First Search* (BFS) [45], Iterative deepening Depth-First Search [49] or *Epsilon Greedy Search* [269]. Considering practical and hardware implementations, reducing the search space or tackling memory issues can be done by pruning strategies, such as *Beam Search* [80]. In Zellij, such an approach consists in deleting fractals from the tree.

In global optimization, high performance requires a trade-off between the exploration of the search space and the exploitation of the acquired knowledge. One has to find the best strategy for this dilemma [259, 253, 166, 183]. In Zellij, non-explored fractals should be visited to ensure that all fractals are evenly explored and that search is not confined to a reduced number of fractals. Exploitation (i.e. intensification) into a reduced region of the search space uses that knowledge (e.g. best-found fractals) to improve it. The promising fractals are searched more thoroughly in the hope of finding better solutions.

Sampling in high dimensional search spaces is a critical task, as it is not always easily tractable to all geometries. Moreover, one does not sample in the same way in a hypercube or in a hypersphere. In DIRECT [135] and SOO [188], only the centers of the hyperrectangles are sampled. In our framework, we consider both deterministic and stochastic sampling. We also consider unique or multiple points methods to sample inside a fractal, such as in FRACTOP. The *exploration* can be done in a passive way (e.g. Markov Chain Monte Carlo sampling, low discrepancy sequences [56]) or in an active way (e.g. metaheuristics [258, 253, 166], surrogate-based optimization [83]).

Once a fractal has been explored, information about its fitness is available. The *scoring* component uses this knowledge to compute a score defining how promising a fractal is. This score is then used by the tree search, with other properties of a fractal. Different scoring components are used in the literature, such as *minimum*, *mean*, *median*, DTTB [192, 142], or *belief* [46].

According to certain criteria, e.g. no more improvement or high confidence, a more intensive search can be applied to a fractal; this will be the *exploitation* component of Zellij. The component is not necessarily restrained to a fractal F , such as in FDA with the ILS only bounded by Ω . One can use local search strategies such as gradient-based algorithms or SA [258, 253, 166].

The *Zellij* workflow is described in Figure 4.16. One can identify the five search components, their interactions, and their algorithmic behaviors. The two *For each* instructions correspond to line 14 and line 16 of Algorithm 16. Two tests are made at each iteration; the stopping criterion corresponds to the *while* loop at line 13, and the maximum depth test of the tree \mathcal{T} to the line 17 in Algorithm 16.

Because each search component is independent of another, it allows instantiating various strategies for fractals, tree search, exploration, exploitation, and scoring search components. One can for instance reproduce FRACTOP by using *Hypercubes*, *Best First Search*, *Belief*, a *GA* for the exploration and a *SA* for the exploitation. Some components can be optional, notably in DIRECT or SOO, for which there is no explicit exploitation or scoring components. Other search components and instantiation of various FBD algorithms will be described throughout the next sections.

The purpose of Zellij is to propose a modular framework to facilitate the design of FBD algorithms through a high-level generalization, and thanks to the five independent search components implemented as software bricks. The following paragraphs will further describe the components and introduce some theoretical background to FBD algorithms.

4.4.1 Geometrical fractal object

The fractal search component within Zellij framework allows structuring high dimensional search spaces to better explore and exploit them. Several types of fractals can be used in the decomposition of the search space, going from hyperspheres [192], hypercubes [46, 135, 188] to more complex structures such as Voronoï cells [142, 162] (see Figure 4.21). This geometrical object has a great impact on the behaviors of FBD algorithms.

In 4.2 we described a *partition* 13, a *hierarchy* 14, a *hierarchy of partitions* 16 and a

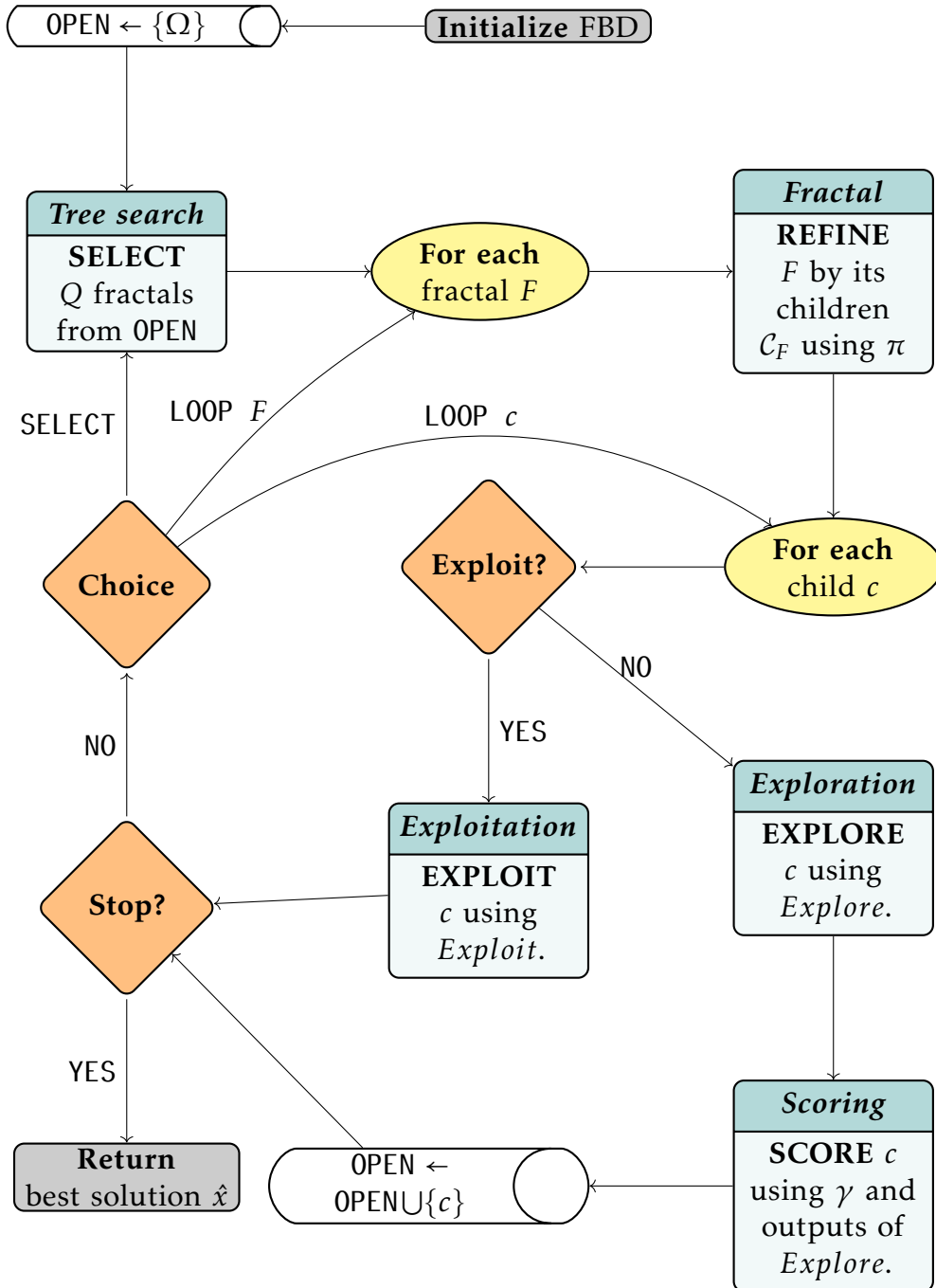


Figure 4.16: Workflow of Zellij.

refinement 17, which can also be found in usual frameworks [118, 239, 58]. To model heuristic approaches such as FDA, we extend these previous frameworks to *covers* and *improper K-covers*.

To extend the flexibility of the frameworks, we denote by $\pi : F, P(F, \mathcal{A}_F) \rightarrow \mathcal{C}_F$ the *partition operator*, i.e. a function taking a fractal F , its properties, and returning a collection of its K children \mathcal{C}_F . Thus, we can redefine the partition according to π . Further properties of π will be detailed by the end of this section.

Definition 18 (*K-partition*). Consider a measure space (Ω, Σ, μ) , with Ω a search space as defined in 10, and $\Sigma = \mathcal{P}(\Omega)$. A *K-partition* of a fractal $F \subseteq \Omega$ by its K children \mathcal{C}_F obtained with a function $\pi : F, P(F, \mathcal{A}_F) \rightarrow \mathcal{C}_F$, is written:

$$\begin{aligned} F &= \bigcup \pi(F, P(F, \mathcal{A}_F)) \\ &= \bigcup \mathcal{C}_F \\ &= \bigcup \{c \in \mathcal{C}_F \mid \forall s, c \in \mathcal{C}_F, s \neq c, c \cap s = \emptyset\} \end{aligned} \quad (4.14)$$

Nonetheless, this definition is still too restrictive to model algorithms like FDA. First, we need to allow overlapping, i.e. dropping the *disjoint at the borders* (\cap) condition. A collection of fractals, for which Ω is included in the union of its elements, is called a *cover*.

Definition 19 (*Cover*). Consider a measure space (Ω, Σ, μ) , with Ω a search space as defined in 10, and $\Sigma = \mathcal{P}(\Omega)$. A *cover* of Ω is a finite collection of fractals, indexed by I , $\mathcal{F} = \{F_i\}_{i \in I}$ such that their union includes Ω :

$$\Omega \subseteq \bigcup_{i \in I} F_i \quad (4.15)$$

We can extend this definition to the partition operator π and any fractal, so to a *K-cover*:

Definition 20 (*K-cover*). Consider a measure space (Ω, Σ, μ) , with Ω a search space as defined in 10, and $\Sigma = \mathcal{P}(\Omega)$. A *K-cover* of a fractal $F \subseteq \Omega$ by its K children \mathcal{C}_F obtained with a function $\pi : F, P(F, \mathcal{A}_F) \rightarrow \mathcal{C}_F$, is written:

$$\begin{aligned} F &\subseteq \bigcup \pi(F, P(F, \mathcal{A}_F)) \\ &\subseteq \bigcup \mathcal{C}_F \end{aligned} \quad (4.16)$$

Figure 4.17, illustrates a *K-cover* of a 2-dimensional Ω with rectangles.

Notice how in our framework, π and any component can make use of any additional information about a fractal F with $P(F, \mathcal{A}_F)$. For instance, a measure of the size of a fractal. In DIRECT [81] the σ -function can be of different types, such as σ_2 or σ_∞ . Thus in DIRECT, $p_{\text{size}}(F, \mathcal{A}_F) = \sigma_\infty(F) \in P(F, \mathcal{A}_F)$.

Using a given fractal F , one can build a recursive *K-cover* of Ω . Indeed, the *refinement* described in 17, is still applicable to *K-covers* using the partition operator π . For readability, we write $\pi(F, P(F, \mathcal{A}_F))$ as $\pi(F)$.

Definition 21 (*Hierarchical K-refinement*). Consider a measure space (Ω, Σ, μ) , with Ω a search space as defined in 10, and $\Sigma = \mathcal{P}(\Omega)$. Let D be the number of successive refinements of a fractal $F_{(l,j,i)} \subseteq \Omega$ obtained by a function $\pi : F, P(F, \mathcal{A}_F) \rightarrow \mathcal{C}_F$.

We write $j \in \llbracket 1, E_l \rrbracket$, with E_l the number of fractals that have been refined l times, and $i \in \llbracket 1, K \rrbracket$ the i th set of a *K-cover* of a superset j . A *hierarchical K-refinement* of $F_{(l,j,i)}$ of maximum level D

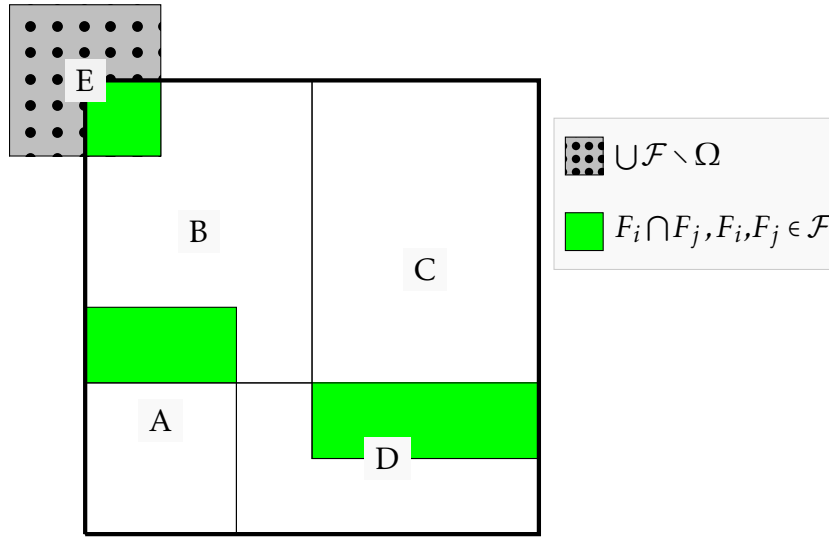


Figure 4.17: Example of a cover of Ω by a collection $\mathcal{F} = \{A, B, C, D, E\}$. The solid-green part corresponds to overlapping between fractals. Dotted-gray is the part of fractals outside the search space Ω represented by a thick line.

(i.e. *depth*) is written as:

$$\begin{aligned} \forall l \in \llbracket 2, D-1 \rrbracket, \exists (x, y, j) \in (\llbracket 1, E_{l-1} \rrbracket, \llbracket 1, K \rrbracket, \llbracket 1, E_l \rrbracket), F_{(l,x,y)} \subseteq \bigcup \pi(F_{(l,x,y)}) \\ \subseteq \bigcup_{i=1}^K F_{(l+1,j,i)} \end{aligned} \quad (4.17)$$

Here, a subset identified by (l, j, i) corresponds to the fractal at level l , child number i of the fractal j at level $l-1$. An appropriate data structure used to model Definition 21 is a K -ary rooted tree. Hence, one can rewrite the initial search space Ω as the root of this tree: $F_{(1,0,1)} = \Omega$, where $E_0 = \emptyset$. A fractal is now considered as a node of a K -ary rooted tree \mathcal{T} . Figure 4.18 shows an example of a 2-refinement of depth 4, of a 2D square using a bisection, drawn as a red dotted line, along the longest side.

For a fixed $l \in \llbracket 1, D-1 \rrbracket$, one can write the set $\mathcal{A}^{(l)}$ of supersets (ancestors) at level l that have children at level $l+1$. Let us denote a leaf at the level l of a tree \mathcal{T} as $\circ[l, j, i]$. An interesting property of using a K -ary tree on a K -refinement that covers Ω is that the initial search space is a subset of the union of all the leaves.

Theorem 1. Consider a measure space (Ω, Σ, μ) , with Ω a search space as defined in 10, and $\Sigma = \mathcal{P}(\Omega)$. Let Ω be the root of a K -ary tree \mathcal{T} of maximum depth D . For all $1 < l \leq D$, the union of all leaves $\mathcal{L} = \{\circ[l, j, i] \in \mathcal{T} \mid \mathcal{C}_{\circ[l, j, i]} = \emptyset\}$, children of nodes numbered by $\llbracket 1, E_{l-1} \rrbracket$ as defined in Definition 21, contains Ω :

$$\Omega = F_{(1,0,1)} \subseteq \bigcup_{l=2}^D \bigcup_{j=1}^{E_{l-1}} \bigcup_{i=1}^K \circ[l, j, i] \quad (4.18)$$

Proof. We consider a k -ary rooted tree $\mathcal{T}^{(d)}$ of depth d , with $1 < d \leq D$. Considering $\mathcal{A}^{(l)}$ the set of fractals at level l having children at level $l+1$: $\bigcup \mathcal{A}^{(d-1)} \subseteq \bigcup_{j=1}^{E_{d-1}} \bigcup_{i=1}^k \circ[d, j, i]$

If we remove all $\circ[d, j, i]$ from $\mathcal{T}^{(d)}$, we obtain a tree $\mathcal{T}^{(d-1)}$. So, the leaves of $\mathcal{T}^{(d-1)}$ at level $d-1$ can be written as $\{\circ[d-1, j, i], \mathcal{A}^{(d-1)}\}$.

Thus, $\bigcup \mathcal{A}^{(d-2)} \subseteq \bigcup_{j=1}^{E_{d-2}} \bigcup_{i=1}^k \circ[d-1, j, i] \cup \mathcal{A}^{(d-1)}$

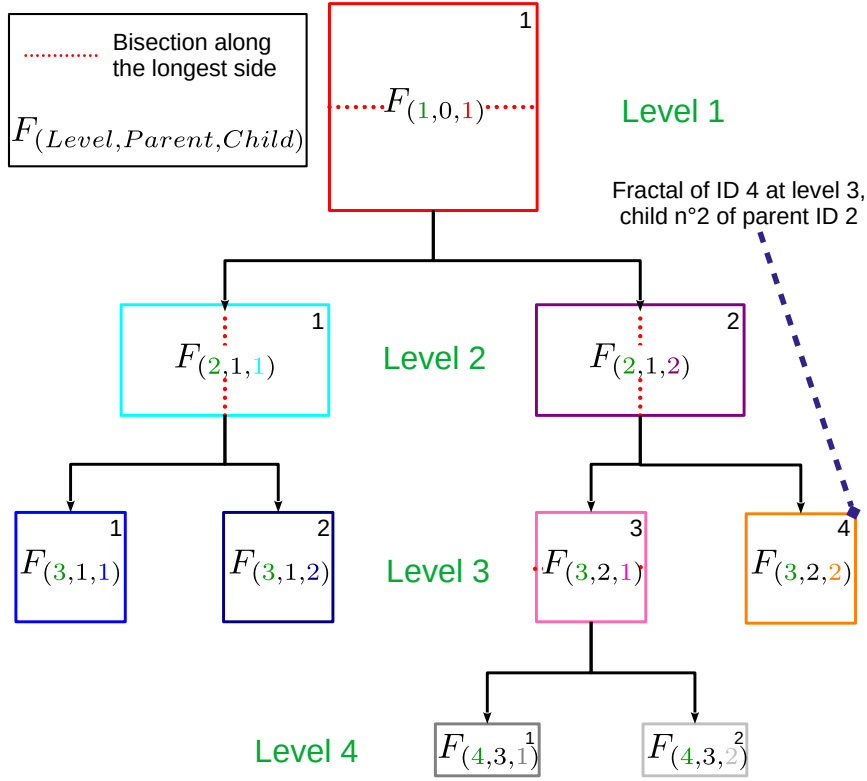


Figure 4.18: Example of a 2-refinement of depth 4, with a bisection along the longest side.

and so on, until $\mathcal{T}^{(1)}$:

$$\Omega \subseteq \mathcal{A}^{(1)} \subseteq \bigcup_{i=1}^k \circ[2, 1, i] \cup \mathcal{A}^{(2)}$$

□

For simplicity, one can write all leaves $\circ[l, j, i]$ of $\mathcal{T}^{(d)}$, a tree \mathcal{T} of depth d , as $\mathcal{L}_{\mathcal{T}^{(d)}}$.

Theorem 1 is valid for a K -cover as defined in 20, allowing overlapping fractals. Moreover, a part of a fractal can lie outside Ω . In Zellij only parts inside Ω are considered as the fractals are trimmed to Ω .

Nonetheless, the current framework does not allow modeling algorithms like FDA, which uses hyperspheres [192] and where fractals do not necessarily fully cover Ω .

We previously described the basic principles of fully covering fractal-based decomposition algorithms. To extend our framework to *improper covers*, i.e. a collection of fractals that only covers a significant part of Ω , we have to measure what the overlap and coverage are for FBD algorithms.

By considering a tree $\mathcal{T}^{(d)}$ and (Ω, Σ) a measurable set, we can write $\mathcal{L}_{\mathcal{T}^{(d)}} \subseteq \Sigma$ and $\mathcal{L}_{\mathcal{T}^{(d)}} \in \Sigma$. We define a measure, $\mu: \Sigma \rightarrow [0, +\infty]$. A stricter condition is applied to μ , $\mu(F) = 0 \iff F = \emptyset$. Hence, the result of a hierarchical K -refinement cannot be made of null sets. Because π cannot produce null sets, and because the maximum tree depth D is finite, we can define a measure of the coverage of Ω by $\mathcal{L}_{\mathcal{T}^{(d)}}$ and between fractals, i.e. coverage of certain areas by at least two fractals.

Definition 22 (Coverage). Let (Ω, Σ, μ) a measure space, and two fractals $A, B \subseteq \Omega$, $A, B \in \Sigma$ and $B \subseteq A$. A measure of the coverage of A by B can be written as:

$$C(A, B) = \mu\left(\bigcup_A B\right) \quad (4.19)$$

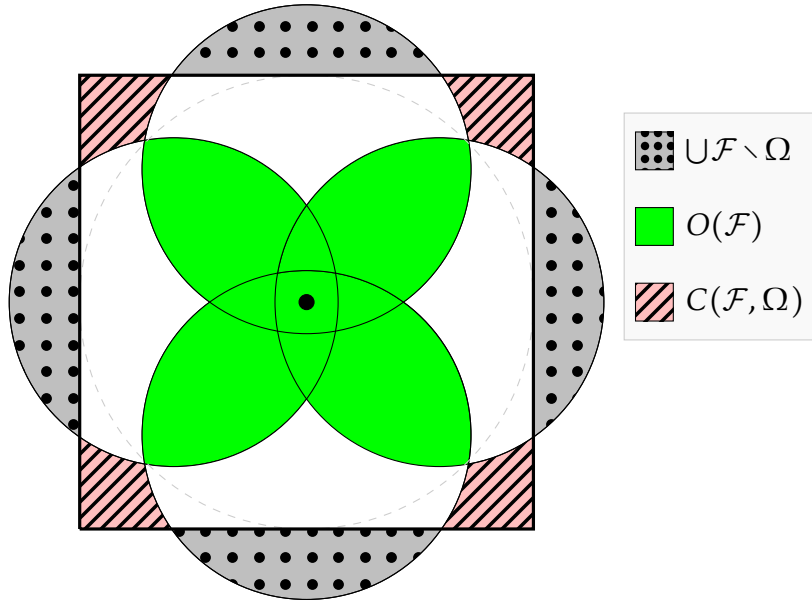


Figure 4.19: 2-dimensional illustration of covering and overlap applied on a FDA scheme.

However if $B \not\subseteq A$ such as in FDA:

$$C(A, B) = \mu(A \setminus (A \cap B)) \quad (4.20)$$

From Equation 4.20 we can infer $B \setminus A$, the part of B outside A .

One can distinguish two types of coverage: $C(\Omega, \cup \mathcal{L}_{\mathcal{T}(d)})$ measures the coverage of the search space by $\mathcal{L}_{\mathcal{T}(d)}$, and $C(F_{(l,j,i)}, \cup \pi(F_{(l,j,i)}))$ measures the coverage of a fractal by its children.

Definition 23 (Overlap). Let (Ω, Σ, μ) a measure space, and $A \neq \emptyset$ a countable collection of fractals such that $\forall n \in \mathbb{N}, \forall a_n \in A : a_n \in \Sigma$ and $\bigcup_{n \in \mathbb{N}} a_n \in \Sigma$. A measure of the overlap between all a_n can be written as:

$$O(A) = \mu \left(\bigcup_{i=1}^{|A|-1} \bigcup_{j=i+1}^{|A|} (a_i \cap a_j) \right) \quad (4.21)$$

Coverage and overlap properties are illustrated in Figure 4.19 as a 2-dimensional FDA scheme. One can see, in hatched-red, the uncovered space and in solid-green the overlap between fractals. Moreover, because of the inflation of hyperspheres, the dotted-gray space represents parts of the fractals outside Ω .

As the monotonic property of measures is not strict, $A \subseteq B \implies \mu(A) \leq \mu(B)$, one cannot say that $\mathcal{L}_{\mathcal{T}(d)}$ covers Ω by only looking at $C(\Omega, \cup \mathcal{L}_{\mathcal{T}(d)})$. Thus, several assumptions are made. For continuous dimensions, we consider that $\cup \mathcal{L}_{\mathcal{T}(d)}$ covers Ω if $C(\Omega, \cup \mathcal{L}_{\mathcal{T}(d)}) = 0$, even if $\cup \mathcal{L}_{\mathcal{T}(d)}$ does not include Ω boundaries. To describe the necessary assumptions of an improper K -cover, we have to clearly define the *partition operator* π .

Definition 24 (Partition operator). Consider a measure space (Ω, Σ, μ) , with Ω a search space as defined in 10, and $\Sigma = \mathcal{P}(\Omega)$. A partition operator is a function $\pi : F, \mathcal{P}(F, \mathcal{A}_F) \rightarrow \mathcal{C}_F$, describing how to create children $\mathcal{C}_F \triangleq \{c_1, c_2, \dots, c_K\}$ of a given fractal $F \subseteq \Omega$ such that :

1. Children cannot be empty nor null sets:

$$\forall i \in [1, \dots, K], c_i \in \mathcal{C}_F, (c_i \neq \emptyset) \wedge (\mu(c_i) > 0)$$

2. The union of the children cannot be empty, and its measure is significant:

$$(\bigcup \mathcal{C}_F \neq \emptyset) \wedge (\mu(\bigcup \mathcal{C}_F) > 0)$$

3. The intersection between children must be part of the search space or can be empty:

$$(\bigcap \mathcal{C}_F \subseteq \Omega) \vee (\bigcap \mathcal{C}_F = \emptyset)$$

4. Two children are strictly different subsets, and a child cannot be a subset of another:

$$\forall i, j \in [1, \dots, K], i \neq j, c_i, c_j \in \mathcal{C}_F (c_i \neq c_j) \wedge (c_i \not\subseteq c_j)$$

5. A subset of a child must be part of its ancestor and the measure of their intersection must be significant:

$$\forall i \in [1, \dots, K], c_i \in \mathcal{C}_F, (c_i \cap \mathcal{A}_{c_i} \neq \emptyset) \wedge (\mu(c_i \cap \mathcal{A}_{c_i}) > 0)$$

6. A child must be significantly smaller than its ancestor:

$$\forall i \in [1, \dots, K], c_i \in \mathcal{C}_F, \mu(c_i) < \mu(\mathcal{A}_{c_i})$$

Even if, definition 24 allows modeling many *improper* K -covers, open questions remain about stricter conditions that could be applied to the definition of the *partition operator*. Notably, about enforcing the part of a child outside its parent to be significantly smaller than the part inside it; $\mu(c_i \cap \mathcal{A}_{c_i}) \gg \mu(c_i \setminus \mathcal{A}_{c_i})$. Which could also be applied to all children; $\mu(\bigcup \mathcal{C}_F \cap \mathcal{A}_{\mathcal{C}_F}) \gg \mu(\bigcup \mathcal{C}_F \setminus \mathcal{A}_{\mathcal{C}_F})$.

By using definitions 11, 22, 23 and 24, we can now modify Theorem 1 for an improper K -cover.

Definition 25. Let Ω be a search space, $\Sigma = \mathcal{P}(\Omega)$ a σ -algebra on Ω , and (Ω, Σ, μ) a measure space. Ω is the root of a K -ary partition tree $\mathcal{T}^{(d)}$ of depth d and $\mathcal{L}_{\mathcal{T}^{(d)}}$ are its leaves. $C(.,.)$ and $O(.,.)$ are measures of the coverage and overlapping between fractals. We suppose that $\mathcal{L}_{\mathcal{T}^{(d)}}$ is not a null set and $\mu(A) = 0 \iff (A = \emptyset) \vee (\bigcup A = \emptyset)$. Then, a given decomposition-based algorithm is said to be:

1. **Preservative:** If the union of the leaves covers the search space;

$$\forall d > 1, C(\Omega, \bigcup \mathcal{L}_{\mathcal{T}^{(d)}}) = 0 \text{ and:}$$

(a) *Proper:* If the fractals never significantly overlap;

$$O(\mathcal{L}_{\mathcal{T}^{(d)}}) = 0$$

(b) *Improper:* If the fractals significantly overlap;

$$O(\mathcal{L}_{\mathcal{T}^{(d)}}) > 0$$

Then,

$$\mu(\Omega) = \mu(\bigcup \mathcal{L}_{\mathcal{T}^{(d)}}) \quad (4.22)$$

2. **Sacrificial:** If the union of the leaves does not cover the search space;

$$\forall d > 1, C(\Omega, \bigcup \mathcal{L}_{\mathcal{T}^{(d)}}) > 0 \text{ and:}$$

(a) *Lowly:* If children do not cover their ancestor for the first decomposition of the search space;

$$\forall l > 1, C(F_{(l,i,j)}, \bigcup \pi(F_{(l,i,j)})) = 0 \text{ and } C(\Omega, \bigcup \pi(\Omega)) > 0. \text{ Then,}$$

$$\mu(\Omega) > \mu(\bigcup \mathcal{L}_{\mathcal{T}^{(d)}}) \quad (4.23)$$

with $\forall d > 1, \mu(\bigcup \mathcal{L}_{\mathcal{T}^{(d)}}) \leq \mu(\bigcup \mathcal{L}_{\mathcal{T}^{(d+1)}})$

- (b) *Highly*: If after every refinement children do not cover their ancestor;
 $\forall l > 1, C(F_{(l,i,j)}, \bigcup \pi(F_{(l,i,j)})) > 0$ and $\bigcup \pi(F_{(l,i,j)}) \setminus F_{(l,i,j)} = \emptyset$. Then, $\forall d > 1$,

$$\mu(\Omega) > \mu(\bigcup \mathcal{L}_{\mathcal{T}^{(d)}}) > \mu(\bigcup \mathcal{L}_{\mathcal{T}^{(d+1)}}) \quad (4.24)$$

- (c) *Unbounded*: If after every refinement a part of the children are outside their ancestor;
 $\forall l > 1, C(F_{(l,i,j)}, \bigcup \pi(F_{(l,i,j)})) > 0$ and $\bigcup \pi(F_{(l,i,j)}) \setminus F_{(l,i,j)} \neq \emptyset$. Then, we cannot infer any relations between measures of the leaves:

$$\mu(\Omega) > \mu(\bigcup \mathcal{L}_{\mathcal{T}^{(d)}}) \not\leq \mu(\bigcup \mathcal{L}_{\mathcal{T}^{(d+1)}}) \quad (4.25)$$

Proof. We consider a K -ary rooted tree \mathcal{T} of depth d , with $1 < d \leq D$. This tree is denoted as $\mathcal{T}^{(d)}$, and the leaves of $\mathcal{T}^{(d)}$ as $\mathcal{L}_{\mathcal{T}^{(d)}}$. One can consider a partition operator π defined by definition 24.

Preservative:

If $\forall d > 1, C(\Omega, \bigcup \mathcal{L}_{\mathcal{T}^{(d)}}) = 0$.

Then $\Omega = \bigcup \mathcal{L}_{\mathcal{T}^{(d)}}$, because by construction $\bigcup \mathcal{L}_{\mathcal{T}^{(d)}} \subseteq \Omega$ and

$$\begin{aligned} C(A, B) = 0 &\iff \mu(A \setminus (A \cap B)) = 0 \\ &\iff A = B \end{aligned}$$

So, we have:

$$\Omega = \bigcup \mathcal{L}_{\mathcal{T}^{(d)}} \implies \mu(\Omega) = \mu(\bigcup \mathcal{L}_{\mathcal{T}^{(d)}})$$

Lowly sacrificial:

If $\forall d > 1, C(\Omega, \bigcup \mathcal{L}_{\mathcal{T}^{(d)}}) > 0$ and $C(F_{(d,j,i)}, \bigcup \pi(F_{(d,j,i)})) = 0$. Then, $\Omega \supset \bigcup \mathcal{L}_{\mathcal{T}^{(d)}} \implies \mu(\Omega) > \mu(\bigcup \mathcal{L}_{\mathcal{T}^{(d)}})$

and $\bigcup \mathcal{L}_{\mathcal{T}^{(d)}} \subseteq \bigcup \mathcal{L}_{\mathcal{T}^{(d+1)}}$.

So we have,

$$\begin{aligned} \forall a \in \mathcal{L}_{\mathcal{T}^{(d)}}, \pi(a) \in \mathcal{L}_{\mathcal{T}^{(d+1)}} \\ \implies \mu(a) \leq \mu(\bigcup \pi(a)) \\ \implies \mu(\bigcup \mathcal{L}_{\mathcal{T}^{(d)}}) \leq \mu(\bigcup \mathcal{L}_{\mathcal{T}^{(d+1)}}) \end{aligned}$$

Highly sacrificial:

If $\forall d > 1, C(\Omega, \bigcup \mathcal{L}_{\mathcal{T}^{(d)}}) > 0, C(F_{(d,j,i)}, \pi(F_{(d,j,i)})) > 0$ and $\bigcup \pi(A) \setminus A = \emptyset$ (Subsets cannot have solutions outside their parents).

Then, $\Omega \supset \bigcup \mathcal{L}_{\mathcal{T}^{(d)}} \implies \mu(\Omega) > \mu(\bigcup \mathcal{L}_{\mathcal{T}^{(d)}})$

and $\bigcup \mathcal{L}_{\mathcal{T}^{(d)}} \supset \bigcup \mathcal{L}_{\mathcal{T}^{(d+1)}}$.

So we have,

$$\begin{aligned} \forall a \in \mathcal{L}_{\mathcal{T}^{(d)}}, \pi(a) \in \mathcal{L}_{\mathcal{T}^{(d+1)}} \\ \implies \mu(a) > \mu(\bigcup \pi(a)) \\ \implies \mu(\bigcup \mathcal{L}_{\mathcal{T}^{(d)}}) > \mu(\bigcup \mathcal{L}_{\mathcal{T}^{(d+1)}}) \end{aligned}$$

Unbounded sacrificial:

If $\forall d > 1, C(\Omega, \bigcup \mathcal{L}_{\mathcal{T}^{(d)}}) > 0, C(F_{(d,j,i)}, \pi(F_{(d,j,i)})) > 0$ and $\bigcup \pi(A) \setminus A \neq \emptyset$.

Then, $\Omega \supset \bigcup \mathcal{L}_{\mathcal{T}^{(d)}} \implies \mu(\Omega) > \mu(\bigcup \mathcal{L}_{\mathcal{T}^{(d)}})$

and $\bigcup \mathcal{L}_{\mathcal{T}^{(d)}} \supset \bigcup \mathcal{L}_{\mathcal{T}^{(d+1)}}$. If $\mu(\bigcup \pi(A) \setminus A) = 0$, then the part of $\pi(A)$ outside A is negligible, so

behaviors are similar to highly sacrificial:

$$\begin{aligned}
& \forall a \in \mathcal{L}_{\mathcal{T}^{(d)}}, \pi(a) \in \mathcal{L}_{\mathcal{T}^{(d+1)}} \\
& \implies \mu(a) > \mu(\pi(a) \cap a) + \mu(\bigcup \pi(A) \setminus A) \\
& \implies \mu(a) > \mu(\bigcup \pi(a)) \\
& \implies \mu(\bigcup \mathcal{L}_{\mathcal{T}^{(d)}}) > \mu(\bigcup \mathcal{L}_{\mathcal{T}^{(d+1)}})
\end{aligned}$$

Otherwise, if $\mu(\bigcup \pi(A) \setminus A) > 0$, then we need additional information or constraints to determine the inequalities. \square

This theorem describes behaviors of an improper Hierarchical K -cover where leaves might overlap or not fully cover Ω .

Preservative algorithms describe a partition that covers Ω , no matter whether there is overlap between leaves or not, as we consider $\bigcup \mathcal{L}_{\mathcal{T}^{(d)}}$. The proper preservative situation, is comparable to previous frameworks [118, 239, 58], where two fractals A and B are disjoint if and only if: $A \cap B$, with negligible overlapping on the borders.

However, lowly sacrificial algorithms describe leaves of the tree that do not fully cover Ω , but where children of the fractal cover its parent. In this case, a part of the search space is lost only at the first refinement of Ω .

For highly sacrificial-based algorithms, one only considers fractals that cannot expand outside their parents. If so, it becomes hard to define the notion of sacrifice, as we can imagine a decomposition where successive children do not fully cover their parents and shift outside an ancestor fractal. This can be the case in unbounded situations, where children may cover space outside their ancestors. Therefore, the choice of π and its combination with the other search components is crucial.

These different situations are described in figure 4.20. Proper-preservative in figure 4.20a based on SOO preserves the initial search space Ω , no space is lost. Figure 4.20b illustrate a case of an improper-preservative algorithm where overlapping fractals fully cover Ω . Low-sacrifice fractals in figure 4.20c are here based on Sierpinski triangles. Such fractals sacrifice space of Ω to use a geometry based on triangles. And finally, the examples on figure 4.20d and 4.20e, based on an FDA scheme, sacrifice space after each refinement. The unbounded sacrificial example shows how difficult it can be to describe overlapping and coverage when children can expand outside their ancestors.

In FDA [192], the algorithm considers the inscribed hypersphere as the initial search space. The algorithm "sacrifices" points between the hypercube and the inscribed hypersphere; then children do not necessarily cover their ancestors and can shift outside because of the inflation ratio. Therefore, FDA can be considered as an unbounded sacrificial algorithm. Nonetheless, if the inflation ratio is not applied, then FDA becomes a highly sacrificial algorithm.

Five properties can inform the choice of the fractal component: the coverage, the overlapping, the partition size K , the computational complexity $\mathcal{O}(\pi)$ of the partition operator π , and the data structure of a fractal. These properties are illustrated in table 4.1 with various example from the literature.

According to the partition size K , the hypercube [46] and simplices [306] are not scalable for high dimensional search spaces. Because K grows exponentially, $\mathcal{O}(\pi)$ does not scale either. Even so K scales, such as with the Voronoï cell where the number of seeds (centers) is a parameter, π is not always adapted to high dimensions. Indeed, popular algorithms computing the Voronoï diagram are well studied for 2D and 3D (e.g. QuickHull [14], Fortune's algorithm [79], Bowyer-Watson algorithm [286]). However, they suffer from the curse of dimensionality. Hence, one has to use heuristics to approximate Voronoï cells in high dimensions [162, 229, 182, 276, 141], such as SpokeDart with hyperplane sampling [182].

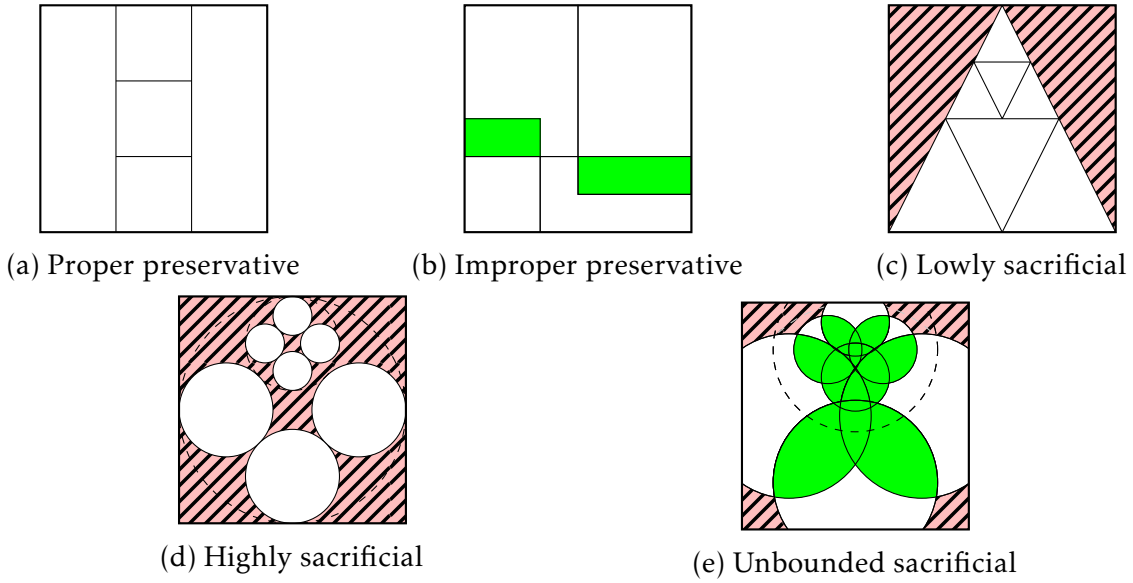


Figure 4.20: 2-dimensional illustrations of definition 25. The hatched-red area corresponds to uncovered space resulting from successive decomposition: $C(\Omega, \cup \mathcal{L}_{\mathcal{T}(d)})$. Solid-green corresponds to overlapping: $O(\mathcal{L}_{\mathcal{T}(d)})$.

A drawback of such approaches is that the partition becomes dynamic. When refining a fractal, new cells are created, and then the whole diagram should be re-computed to consider children's boundaries. Other hypervolumes showed in Figure 4.21 are easier to compute, and further details can be found in [135, 46, 192, 188].

Now, considering the coverage and overlap properties, the major drawback of hyperspheres compared to other hypervolumes is the poor coverage of Ω by $\mathcal{L}_{\mathcal{T}(d)}$. If $C(\Omega, \mathcal{L}_{\mathcal{T}(d)})$ and $C(F_{(l,j,i)}, \cup \pi(F_{(l,j,i)}))$, are strictly superior to 0, then the partition does not fully cover the search space or a parent fractal. According to [192], the inflation ratio applied on hyperspheres can reduce the substantial lack of coverage by increasing the overlap. However, we have to mention the impact of the curse of dimensionality on such objects. Indeed, if we look at the Hausdorff measure of a unit hypersphere, the hyper-volume and the hyper-surface tend to zero as the dimension tends to infinity. This can explain the empirical results obtained in [192], indicating that the deeper the decomposition tree, the lower FDA performs.

Figure 4.21 illustrates that 2D representations are misleading. Even if they are intuitive and allow to better understand basic principles, we cannot infer behaviors in high dimensions by only looking at 2D or 3D drawings. Thus, the choice of a fractal can be informed by carefully examining the properties defined in Table 4.1.

4.4.2 Tree search

Before each refinement, a fractal candidate has to be selected to be further refined. As previously described, a K -hierarchical refinement can be modeled by a K -ary rooted tree. The expansion of this tree relies on a tree search algorithm, written as a function τ selecting Q unique fractals among all $\mathcal{L}_{\mathcal{T}(d)}$:

$$\tau : \mathcal{L}_{\mathcal{T}(d)}, P(\mathcal{L}_{\mathcal{T}(d)}), p_{\text{score}}(\mathcal{L}_{\mathcal{T}(d)}) \rightarrow \{e_1, \dots, e_Q\}, \quad (4.26)$$

where $\forall q \in [1, \dots, Q] : e_q \in \mathcal{L}_{\mathcal{T}(d)}, 1 \leq Q \leq s$. The property vector $p_{\text{score}}(\mathcal{L}_{\mathcal{T}(d)}) \subset \mathbb{R}^s$ of size $s = |\mathcal{L}_{\mathcal{T}(d)}|$, describes the scores of leaves. The properties of the leaves are denoted $P(\mathcal{L}_{\mathcal{T}(d)}) = \{P(F, \mathcal{A}_F) \mid \forall F \in \mathcal{L}_{\mathcal{T}(d)}\}$ and $p_{\text{score}}(\mathcal{L}_{\mathcal{T}(d)}) = \{p_{\text{score}}(F, \mathcal{A}_F) \mid \forall F \in \mathcal{L}_{\mathcal{T}(d)}\}$.

Table 4.1: Example of fractals and their properties

	$S_{l,j,i}$	n -cube	Trisection	Bisection	n -sphere	Voronoi	Simplex
Partition size	k	2^n	3	2	$2n$	c^a	$n!$
Partition building complexity	$\mathcal{O}(\pi)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(2^n)^b$	$\mathcal{O}(n!)$
Coverage of Ω	$C(\Omega, \cup \mathcal{L}_{\mathcal{T}^{(d)}})$	0	0	0	> 0	0	0
Children coverage	$C(A, \cup \pi(A))$	0	0	0	> 0	0	0
Overlap	$O(\mathcal{L}_{\mathcal{T}^{(d)}})$	0	0	0	> 0	0 or $> 0^c$	0
Data structure	\emptyset	center and side length	2 points of size n	2 points of size n	center and radius	See c	$n + 1$ points of size n
Examples	\emptyset	[46]	[188]	[210]	[192]	[162, 142]	[306]

^aNumber of centroids defined by the user.

^bValid for usual algorithms, we can reduce this complexity by approximating the Voronoi diagram in high dimensions. This complexity, also depends on c , the number of centroids. But here we consider the complexity depending on the dimension n .

^cIt depends on the algorithm used to compute the diagram. It can be a set of vertices for the QuickHull algorithm or a set of hyperplanes for sampling methods. An exact algorithm or a heuristic approach impacts the properties.

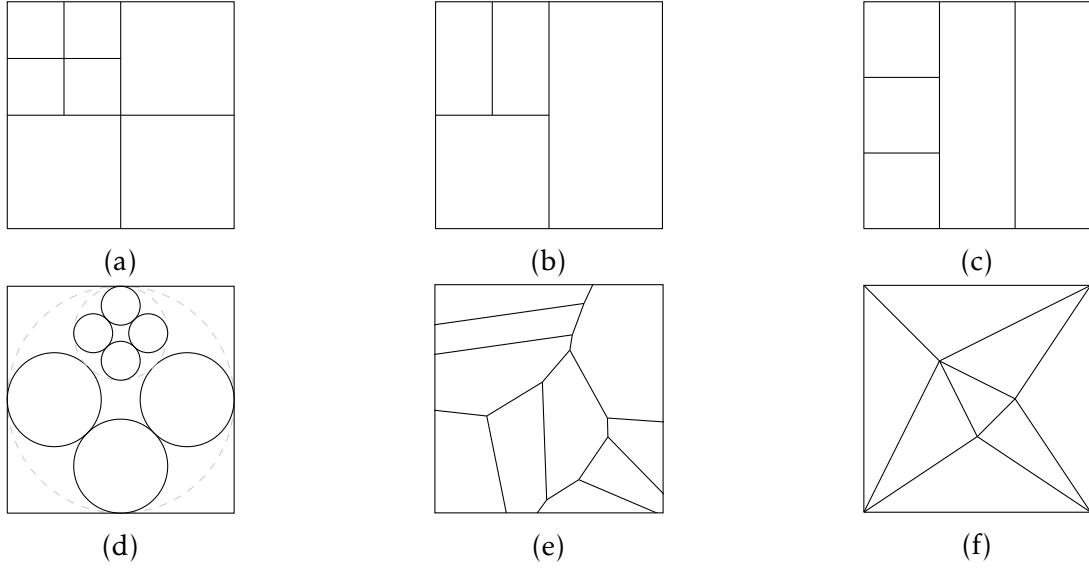


Figure 4.21: Various examples of fractals in 2 dimensions: (a) hypercubes, (b) bisections, (c) trisections, (d) hyperspheres, (e) dynamic Voronoï, (f) simplices.

One can instantiate tree search algorithms using an OPEN and CLOSED lists [45], as described in algorithm 12. The OPEN list contains unexplored or unexploited fractals. The CLOSED list contains expanded fractals. Thus, the tree search component is a rule defining how to append non-expanded fractals to these lists and how to select them. The CLOSED list is generally used to prevent the algorithm from selecting expanded fractals.

Algorithm 12 OPEN-CLOSE

Inputs:

- 1: $\text{OPEN} = \mathcal{L}_{\mathcal{T}^{(d)}}$
- 2: CLOSE
- 3: Q
- 4: $P(\text{OPEN})$

Outputs: \mathcal{F}_Q

- 5: $\mathcal{F}_Q \leftarrow \text{SELECT}(\text{OPEN}, \text{CLOSE}, Q, P(\text{OPEN}))$
 - 6: $\text{OPEN} \leftarrow \text{OPEN} \setminus \mathcal{F}_Q$
 - 7: $\text{CLOSE} \leftarrow \text{CLOSE} \cup \mathcal{F}_Q$
 - 8: **return** \mathcal{F}_Q
-

Properties of OPEN
Selected fractal

As the exploration and exploitation tradeoff is a major challenge in global optimization, the selection of the tree search algorithm requires a peculiar attention for a K -hierarchical refinement. For example, on one hand, BrFS is mostly inefficient within a FBD algorithm. Indeed, all fractals at a certain level will be decomposed before selecting fractals of the next level. See line 5 of algorithm 13.

On the other hand, and conversely to BrFS, DFS always selects the deepest fractals, which is illustrated in line 5 of algorithm 14. Hence, DFS can be considered a greedy *exploitation only*, whereas BrFS is a greedy *exploration only* algorithm [65]. Both strategies are ineffective, as the scores given to fractals do not impact the selection. We lose the notion of hierarchy between fractals. Their behaviors are illustrated in figure 4.22 for different tree search algorithms.

In FDA, a sorted DFS, called Move-up, favor the selection of deep fractals to rapidly apply the ILS. By using a more exploitive tree search, FDA emphasizes greedy exploitation of deep

Algorithm 13 BrFS**Inputs:**1: $\text{OPEN} \triangleq \mathcal{L}_{\mathcal{T}(d)}$

2: CLOSE

3: $p_{\text{level}}(\text{OPEN})$ *Level property of fractals***Outputs:** \mathcal{F}_Q *Selected fractal*4: $\mathcal{F}_Q \leftarrow \underset{F \in \text{OPEN}}{\text{argmin}} p_{\text{level}}(F, \mathcal{A}_F)$ 5: $\text{OPEN} \leftarrow \text{OPEN} \setminus \mathcal{F}_Q$ 6: $\text{CLOSE} \leftarrow \text{CLOSE} \cup \mathcal{F}_Q$ 7: **return** \mathcal{F}_Q **Algorithm 14** DFS**Inputs:**1: $\text{OPEN} \triangleq \mathcal{L}_{\mathcal{T}(d)}$

2: CLOSE

3: $p_{\text{level}}(\text{OPEN})$ *Level property of fractals***Outputs:** \mathcal{F}_Q *Selected fractal*4: $\mathcal{F}_Q \leftarrow \underset{F \in \text{OPEN}}{\text{argmax}} p_{\text{level}}(F, \mathcal{A}_F)$ 5: $\text{OPEN} \leftarrow \text{OPEN} \setminus \mathcal{F}_Q$ 6: $\text{CLOSE} \leftarrow \text{CLOSE} \cup \mathcal{F}_Q$ 7: **return** \mathcal{F}_Q

fractals, which can lead to a lack of exploration.

To replace DFS or BrFS, an efficient and greedy candidate could be the BFS algorithm [45]. Instead of focusing on $p_{\text{level}}(F, \mathcal{A}_F)$ fractal's property, BFS selects the fractal with the best score property; $p_{\text{score}}(F, \mathcal{A}_F)$.

Algorithm 15 BFS**Inputs:**1: $\text{OPEN} \triangleq \mathcal{L}_{\mathcal{T}(d)}$

2: CLOSE

3: $p_{\text{score}}(\text{OPEN})$ *Score property of fractals***Outputs:** \mathcal{F}_Q *Selected fractal*4: $\mathcal{F}_Q \leftarrow \underset{F \in \text{OPEN}}{\text{argmax}} p_{\text{score}}(F, \mathcal{A}_F)$ 5: $\text{OPEN} \leftarrow \text{OPEN} \setminus \mathcal{F}_Q$ 6: $\text{CLOSE} \leftarrow \text{CLOSE} \cup \mathcal{F}_Q$ 7: **return** \mathcal{F}_Q

Some of these tree search algorithms are more complex and can be stochastic (e.g. Epsilon Greedy Search, Diverse Best First Search [126]), others allow a parameterization of the exploration-exploitation tradeoff (e.g. Cyclic best First Search [184]).

In the DIRECT algorithm, a complex algorithm is used. The POH strategy, iteratively estimates the Lipschitzian constant L by grouping fractals according to their score $p_{\text{score}}(F, \mathcal{A}_F)$ and sizes $p_{\text{size}}(F, \mathcal{A}_F)$. In DIRECT, the score corresponds to the center of F and its size is determined by a σ function, which can be the Euclidean distance between the center and one of the fractal's vertex. Many other variations of DIRECT are based on this selection strategy [81, 73, 136].

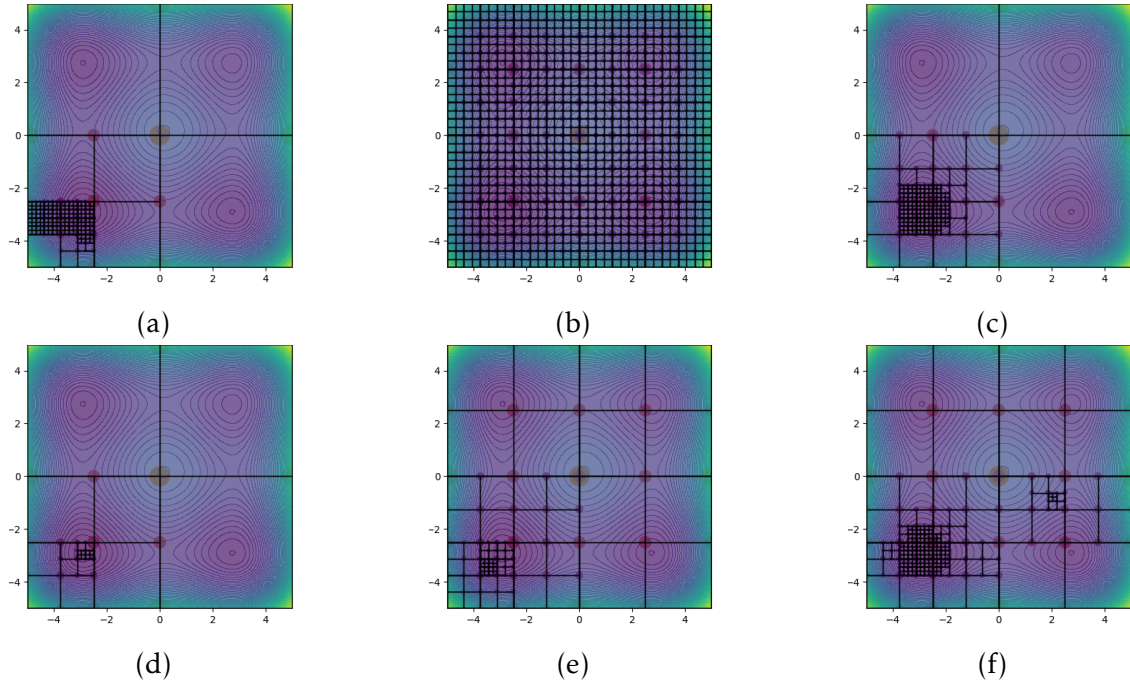


Figure 4.22: Fractal decomposition with hypercubes applied on 2D Styblinski-Tang function with various tree search: (a) Depth First Search, (b) Breadth First Search, (c) Best First Search, (d) Beam Search, (e) Cyclic Best First Search, (f) Epsilon Greedy Search.

The notion of sacrifice defined in definition 25 also concerns tree search algorithms and pruning techniques. For time and memory complexity reasons, it can be necessary to prune some leaves. For instance, in the BS algorithm [80], only a given number of leaves are stored and selected.

4.4.3 Scoring fractals

We previously defined $p_{\text{score}}(F, \mathcal{A}_F)$, a real value defining the quality of a fractal. This value summarizes gathered information about F . To compute $p_{\text{score}}(F, \mathcal{A}_F)$, we define a scoring method γ assigning a quality value to a given fractal F and a finite set of solutions \mathcal{S} restricted to F . Such as in [118, 239], γ is similar to the *characteristic value* defining the probability that a fractal contains the global optimum:

$$\gamma : F, P(F, \mathcal{A}_F), \mathcal{S}, f|_F(\mathcal{S}) \rightarrow \mathbb{R} \quad (4.27)$$

The scoring method γ takes a fractal F , a sample of solutions \mathcal{S} restricted to the region of F , and their corresponding objective values $f|_F(\mathcal{S})$. The function returns a score within \mathbb{R} which defines the quality of F . The properties of the leaves, denoted $P(F, \mathcal{A}_F)$, can also be considered. For example, concerning FRACTOP and the *Belief*, in equation 4.13, the score of \mathcal{A}_F is used to compute $\gamma(F, P(F, \mathcal{A}_F), \mathcal{S}, f|_F(\mathcal{S}))$.

In DIRECT and SOO, γ can be written as $\gamma(F, P(F, \mathcal{A}_F), c, f|_F(c)) = f|_F(c)$, where c is the center of F .

In FDA, the algorithm maximizes the DTTB solution found so far among all sampled solutions. So $\gamma(F, P(F, \mathcal{A}_F), \mathcal{X}, f|_F(\mathcal{X})) = \max_{x \in \mathcal{X}} \frac{f|_F(x)}{\|x - \text{BSF}\|}$, where $\mathcal{X} = \left\{ c - \alpha \frac{r_F}{\sqrt{n}}, c, c + \alpha \frac{r_F}{\sqrt{n}} \right\}$. Here, c is the center of the hypersphere F , r_F its radius, α the inflation ratio [192], BSF is the best solution found so far, and n is the dimension of the problem.

The combination between τ (tree search) and γ (scoring) is essential, and has different

purposes concerning the exploration and exploitation tradeoff. Finally, τ and γ can use additional information modeled within properties $P(F, \mathcal{A}_F)$, such as a measure of the size of a fractal. For example, the σ function in DIRECT, with σ_2 or σ_∞ , measures the size of hyperrectangles according to their level or the length of their longest side [81].

4.4.4 Exploration and exploitation strategies

The scoring component γ requires information about the landscape of F , i.e. \mathcal{S} and $f|_F(\mathcal{S})$. The exploration component gathers this information, and so the quality of the scoring and the exploration-exploitation tradeoff also relies on this. As previously defined in the sections 4.3 and 4.4, once the confidence within the quality of a fractal reaches a certain criterion, one can exploit or intensify, the search within and around this fractal.

The exploration *Explore* and exploitation *Exploit* components are defined by the following functions applied on a fractal F :

$$\begin{aligned} \text{Explore} : F, P(F, \mathcal{A}_F) &\rightarrow \mathcal{S}, f|_F(\mathcal{S}) \\ \text{Exploit} : F, P(F, \mathcal{A}_F) &\rightarrow \mathcal{S}, f(\mathcal{S}) \end{aligned} \quad (4.28)$$

where \mathcal{S} is a finite set of sampled points restricted to the region of the fractal F . Notice that *Exploit* is not restricted by F .

Sampling in hyperrectangles or hypercubes is a simple procedure. One can apply all sampling methods or metaheuristics using search spaces as defined by 10. For example, low discrepancy sequences such as Sobol, Halton, and Kronecker methods can be used [56]. Sampling in a hypersphere requires a few tricks to satisfy the equation of a n -ball. The Box-Muller method can be a solution [277, 187, 104]. However, sampling inside a Voronoï cell (i.e. polytope) is a complex procedure. One could use methods to approximate the Lebesgue measure [86], hit-and-run sampling [36], hyperplanes sampling [182], or MCMC sampling [30]. For active algorithms, such as metaheuristics (e.g. local search, evolutionary algorithms, swarm optimization), one must adapt the search operators (e.g. neighborhood, mutation, crossover, velocity update) to the fractal geometry.

Finally, if the algorithm lacks of exploitation, one could apply an optional exploitation algorithm within leaves. This is the case for FRACTOP or FDA, with resp. a SA or ILS.

It is a thorough task to select an exploration and an exploitation search components. The budget allocated to *Explore* and *Exploit*, can be informed by the cost of the objective function, the maximum depth D of the tree, or the partition size K . A budget that is too high for *Explore* can result in a low exploitation phase. Whereas, a budget that is too low for the exploration phase can result in expensive exploitation within areas of low confidence.

Algorithm 16 Fractal-based decomposition algorithm**Inputs:**

- | | |
|-------------------|---------------------------------------|
| 1: Ω | <i>Initial search space</i> |
| 2: D | <i>Maximum depth</i> |
| 3: <i>Explore</i> | <i>Exploration strategy</i> |
| 4: <i>Exploit</i> | <i>Exploitation strategy</i> |
| 5: π | <i>Fractal decomposition function</i> |
| 6: τ | <i>Tree search</i> |
| 7: γ | <i>Scoring</i> |

Outputs: \hat{x} *Best solution found*

```

8:  $\hat{x} \leftarrow \infty$ 
9: OPEN  $\leftarrow \{\Omega\}$ 
10: CLOSED  $\leftarrow \{\cdot\}$ 
11: current  $\leftarrow \{\Omega\}$ 
12: scores  $\leftarrow \{+\infty\}$ 
13: while stopping criterion not reached do
14:   for each leaf  $\in$  current do
15:     children  $\leftarrow F(\text{leaf})$ 
16:     for each child  $\in$  children do
17:       if level(child)  $< D$  then
18:          $R, \text{values} \leftarrow \text{Explore}(\text{child})$ 
19:         score  $\leftarrow \gamma(\text{child}, R, \text{values})$ 
20:         Append child to OPEN
21:         Append score to scores
22:         if min(values)  $< \hat{x}$  then
23:            $\hat{x} \leftarrow \min(\text{values})$ 
24:       else
25:         values  $\leftarrow \text{Exploit}(\text{child})$ 
26:         if min(values)  $< \hat{x}$  then
27:            $\hat{x} \leftarrow \min(\text{values})$ 
28:       Append leaf to CLOSED
29:       index  $\leftarrow$  Index of leaf in OPEN
30:       Remove element at index from OPEN
31:       Remove element at index from scores
32:   current  $\leftarrow \tau(\text{OPEN}, \text{scores})$ 
return  $\hat{x}$ 

```

4.5 Instantiating algorithms within Zellij

This section introduces how to instantiate some popular FBD algorithms within Zellij. Moreover, it shows how one can extend these algorithms using various search strategies for the different components. These algorithms are presented per search component in table 4.2.

4.5.1 FDA

Fractal

FDA decomposes the search space Ω by using $2n$ hyperspheres, with n the dimensionality. The initial center C_1 of Ω is computed by $C_1 = \frac{U+L}{2}$, with U and L the upper and lower bounds. The radius of the hypersphere of center C_1 is $r_1 = \frac{U-L}{2}$. Then the region of $F_{(l,i,j)}$ is defined by the hypersphere of center $C_{l,i,j}$ and its radius r_l at level l . To simplify notation, only $C_{,...}$ are considered instead of the region $F_{(,...)}$. Centers are points of size n from the search space, $C_{(...)} \in \Omega$. The partition operator π is defined by:

$$\pi(C_{l,...}, P(C_{l,...}, \mathcal{A}_{C_{l,...}})) = \{C_{l+1,...,j} = C_{l,...} + (-1)^j (r_l - r_{l+1}) \vec{e}_j\}_{j \in \llbracket 1, n \rrbracket} ,$$

where \vec{e}_j is the unit vector at dimension j and $r_{l+1} = \frac{r_l}{1+\sqrt{2}}$.

Exploration

The PHS, i.e. the exploration component, computes the center of the current hypersphere and two symmetrical points as previously defined in section 4.4.3. So, $Explore: F_{(l,i,j)}, P(F_{(l,i,j)}, \mathcal{A}_{F_{(l,i,j)}}) \rightarrow \mathcal{X}, f|_{F_{(l,i,j)}}(\mathcal{X})$, where

$$\mathcal{X} = \left\{ C_{l,i,j} - \alpha \frac{r_l}{\sqrt{n}}, C_{l,i,j}, C_{l,i,j} + \alpha \frac{r_l}{\sqrt{n}} \right\} ,$$

where $\alpha > 1$ is the inflation ratio. Points that could be sampled outside Ω because of the inflation, are trimmed to the borders of the search space.

Scoring

From the 3 previously sampled points, the scoring method is defined by taking the maximum of a slope such that,

$$\gamma : F, P(F, \mathcal{A}_F), \mathcal{X}, f|_F(\mathcal{X}) \rightarrow \max_{x \in \mathcal{X}} \frac{f|_F(x)}{\|x - \text{BSF}\|} ,$$

where BSF is the best solution found so far. One issue with BSF, is that it is a global information that varies through iterations, and that needs to be shared across all fractals. This information cannot be shared through inheritance.

Tree search

Concerning the tree search component, i.e. Move-Up, it is comparable to a sorted DFS algorithm. Here, the best fractal of maximum current depth d is selected at each iteration, only if it is not a fractal of maximum depth D . The Move-Up algorithm is described in Algorithm 17.

Algorithm 17 Move-Up**Inputs:**

- | | |
|---|--------------------------|
| 1: OPEN $\triangleq \mathcal{L}_{\mathcal{T}(d)}$ | <i>Leaves</i> |
| 2: d | <i>Current depth</i> |
| 3: D | <i>Maximum depth</i> |
| 4: γ | <i>Scoring component</i> |
| 5: | |

Outputs: A *Selected fractal*

```

6: if  $d = D$  then
7:   depth  $\leftarrow d - 1$ 
8: else
9:   depth  $\leftarrow d$ 
10:  $A \leftarrow \emptyset$ 
11: while  $(A = \emptyset) \wedge (\text{depth} > 1)$  do
12:   leaf =  $\{F_{(\text{depth}, i, j)} \in \text{OPEN}\}_{\forall i, j}$ 
13:    $A \leftarrow \underset{a \in \text{leaf}}{\text{argmin}}(\gamma(a))$ 
14:   depth  $\leftarrow \text{depth} - 1$ 
15: OPEN  $\leftarrow \text{OPEN} \setminus A$ 
16: CLOSE  $\leftarrow \text{CLOSE} \cup A$ 
17: return  $A$ 

```

Exploitation

The ILS is similar to a greedy coordinate search. It is defined in [192, 191] and in algorithm 18. The algorithm iterates through all n axes until a stopping criterion is met. At each iteration, the ILS samples two symmetrical points on the current axis, centered on the best current point. The first considered point is the center of the fractal in which the ILS is executed. At each iteration, the two sampled points are determined by a step ω from the current best point. Then, if one of these points is better than the current one, the ILS moves toward this point and considers it as the new current best point. When the ILS has iterated through all axes, and if no improvement of the center occurred, then ω is decreased and the algorithm restarts iterating through the axes.

4.5.2 SOO and NMSO**Fractal**

SOO and NMSO divide the search space Ω by using trisections ($K = 3$) along the current longest side of the ancestor \mathcal{A}_F . Similarly to the Definition 10, here the region of a fractal $F_{(l, i, j)}$ is defined by a hyperrectangle of bounds $L_{l, i, j}$ and $U_{l, i, j}$. So, a fractal $F_{(l, i, j)}$ is defined by the pair $L_{l, i, j}, U_{l, i, j}$. The partition operator can be written,

$$\pi((L_{l, \dots}, U_{l, \dots}), P(F_{(l, \dots)}, \mathcal{A}_{F_{(l, \dots)}})) = \{(L_{l, \dots} + k \cdot \ell \cdot \vec{e}_\ell, U_{l, \dots} - (K - k - 1) \cdot \ell \cdot \vec{e}_\ell)\}_{k \in \llbracket 0, K-1 \rrbracket},$$

where ℓ is the index of the longest dimension – or side – of $F_{(l, i, j)}$, and \vec{e}_ℓ is the unit vector at index ℓ .

Algorithm 18 ILS**Inputs:**

1:	F	<i>A fractal</i>
2:	C	<i>Center of F</i>
3:	r_f	<i>Radius of F</i>
4:	ρ	<i>Reduction factor</i>
5:	d	<i>Dimension</i>
6:	$\omega \leftarrow r_f$	
7:	center $\leftarrow C$	
8:	improvement $\leftarrow \text{False}$	
9:	while stopping criterion not met do	
10:	for $i \leftarrow 1$ to d do	
11:	$\Delta \leftarrow \omega \times \vec{e}_i$	<i>Step</i>
12:	$\mathcal{X} \leftarrow \{\text{center} - \Delta, \text{center}, \text{center} + \Delta\}$	
13:	best $\leftarrow \underset{x \in \mathcal{X}}{\text{argmax}} f(x)$	
14:	if $f(\text{center}) < f(\text{best})$ then	
15:	center $\leftarrow \text{best}$	
16:	improvement $\leftarrow \text{True}$	
17:	if improvement then	
18:	improvement $\leftarrow \text{False}$	
19:	else	
20:	$\omega \leftarrow \rho \times \omega$	
	return center	

Exploration

The exploration component of SOO and NMSO, consists in computing the center of each fractal. So, $\text{Explore} : F_{(l,i,j)}, P(F_{(l,i,j)}, \mathcal{A}_{F_{(l,i,j)}}) \rightarrow \mathcal{S}, f|_{F_{(l,i,j)}}(\mathcal{S})$, where

$$\mathcal{S} = \left\{ \frac{U_{l,i,j} + L_{l,i,j}}{2} \right\},$$

Scoring

Then, the scoring component is the objective value of the center $\gamma(F, P(F, \mathcal{A}_F), \mathcal{S}, f|_F(\mathcal{S})) = f|_F(\mathcal{S})$. Here, the scoring component does not have much impact on the algorithm. The exploration-exploitation tradeoff is mainly explained by the tree search algorithm.

Tree search

The tree search of SOO consists in selecting, within the tree $\mathcal{T}^{(d)}$ in a top-down manner, the best fractal at each level only if it is better than all fractals of previous levels. This component is described in algorithm 19.

The tree search of NMSO is more complex; we give here an overview of the algorithm; more details about the implementation are found in [57] and the source code ². This component balances in width and depth, the exploration of $\mathcal{T}^{(d)}$. To do so, the algorithm builds sequences of depth first search selection. Then, according to a criterion based on the slopes between the centers of the children of a fractal and their sizes, NMSO restarts a

²<https://github.com/ash-aldujaili/NMSO/tree/master>

Algorithm 19 SOO tree search**Inputs:**

- | | | |
|--|--|--------------------------|
| 1: OPEN $\triangleq \mathcal{L}_{\mathcal{T}^{(d)}}$ | | <i>Leaves</i> |
| 2: d | | <i>Current depth</i> |
| 3: D | | <i>Maximum depth</i> |
| 4: γ | | <i>Scoring component</i> |
| 5: | | |

Outputs: A *Selected fractals*

```

6: depth  $\leftarrow 1$ 
7:  $v_{\max} \leftarrow +\infty$ 
8:  $A \leftarrow \{\cdot\}$ 
9: while depth < min( $d, D$ ) do
10:   leaf =  $\{S_{\text{depth}, i, j} \in \text{OPEN}\}$ 
11:    $\alpha \leftarrow \underset{a \in \text{leaf}}{\text{argmin}}(\gamma(a))$ 
12:   if ( $\alpha \neq \emptyset$ )  $\wedge$  ( $\gamma(\alpha) \leq v_{\max}$ ) then
13:      $A \leftarrow A \cup \{\alpha\}$ 
14:      $v_{\max} \leftarrow \gamma(\alpha)$ 
15:   depth  $\leftarrow$  depth + 1
16: OPEN  $\leftarrow$  OPEN  $\setminus$  A
17: CLOSE  $\leftarrow$  CLOSE  $\cup$  A
18: return A

```

new sequence from one of the highest (low level), non-expanded fractals. The fractals from stopped sequences are put in a *basket*. If after a certain number of iterations, these leaves from the basket are visited – and not refined – a certain number of times according to their quality, then they are selected to resume the stopped sequences.

Exploitation

SOO and NMSO do not have any exploitation component. The balance between exploration and exploitation is mainly guided by the tree search component. As the depth of the tree is supposed to be infinite, a part of the budget could be allocated to an exploitation component applied to the best fractal once SOO or NMSO are stopped.

4.5.3 DIRECT

The instantiation of DIRECT within *Zellij* is based on the following user guide [74].

Fractal and Exploration

Like SOO and NMSO, DIRECT is based on trisections applied to hyperrectangle of bounds $L_{l,\dots}$ and $U_{l,\dots}$ for a level l . Unlike other algorithms, the partition operator first requires the exploration of the fractal in order to sort the axis – or sides of the fractal – of maximum length. Then, a series of trisections along all axes of maximal sizes is iteratively applied to the resulting central hyperrectangles.

The exploration of a fractal $F_{(l,\dots)}$ can be written as: $\text{Explore} : F_{(l,\dots)}, P(F_{(l,\dots)}), \mathcal{A}_{F_{(l,\dots)}} \rightarrow \mathcal{S}, f|_{F_{(l,\dots)}}(\mathcal{S})$, where

$$\mathcal{S} = \{C_{l,\dots} - \delta \vec{e}_j, C_{l,\dots} + \delta \vec{e}_j\}_{j \in I} ,$$

with the center of a fractal $C_{l,...} = \frac{U_{l,...} - L_{l,...}}{2}$, the set I containing the indices of the axis of maximum size for a given fractal, $I = \operatorname{argmax}_{i \in \llbracket 1, n \rrbracket} (U_{l,...} \times \vec{e}_i - L_{l,...} \times \vec{e}_i)$, and δ is equal to one third of the longest side of a given fractal, $\delta = \max(U_{l,...} - L_{l,...}) / 3$. Once a hyperrectangle has been explored, the resulting points along all axes of maximum sizes are used to iteratively trisects the fractal; refer to [74] for more details. The combination of the partitioning factor and *Explore* is detailed in algorithm 20. Lines 6 to 10 can be considered the *Explore* search component.

Algorithm 20 DIRECT partition operator

Inputs:

- | | | |
|----------|--|-----------------------------|
| 1: F | | <i>Fractal</i> |
| 2: L_F | | <i>Fractal lower bounds</i> |
| 3: U_F | | <i>Fractal upper bounds</i> |
| 4: n | | <i>dimension</i> |

Outputs: \mathcal{C}_F

- | | | |
|---|--|--|
| 5: $\mathcal{C}_F \leftarrow \{\cdot\}$ | | <i>Children</i> |
| 6: $c \leftarrow \frac{U_F + L_F}{2}$ | | <i>Center</i> |
| 7: $L_c, U_c \leftarrow L_F, U_F$ | | |
| 8: $\delta \leftarrow \max\left(\frac{U_F - L_F}{3}\right)$ | | <i>1/3 of maximum side length</i> |
| 9: $I \leftarrow \operatorname{argmax}_{i \in \llbracket 1, n \rrbracket} (U_F - L_F) \times \vec{e}_i$ | | <i>Longest axis</i> |
| 10: $w \leftarrow \{w_i = \min(f(c - \delta \vec{e}_i), f(c + \delta \vec{e}_i))\}_{i \in I}$ | | |
| 11: $I' \leftarrow \mathbf{Reorder}(I, w)$ | | <i>Sort I in a descending order according to w</i> |
| 12: for $i \in I'$ do | | |
| 13: $\text{child}_1 \leftarrow (L_c, U_c - 2\delta \times \vec{e}_i)$ | | |
| 14: $\text{child}_2 \leftarrow (L_c + 2\delta \times \vec{e}_i, U_c)$ | | |
| 15: $L_c, U_c \leftarrow L_c + \delta \times \vec{e}_i, U_c - \delta \times \vec{e}_i$ | | |
| 16: $c \leftarrow (L_c, U_c)$ | | |
| 17: $\mathcal{C}_F \leftarrow \mathcal{C}_F \cup \{\text{child}_1, \text{child}_2\}$ | | |
-

Tree Search

The tree search algorithm of DIRECT is complex and bounds the Lipschitz constant of leaves $\mathcal{L}_{\mathcal{T}^{(d)}}$ by building the set of POH, i.e. a tradeoff between the score of a fractal and its size. The selection of POH is described in Algorithm 21 and is based on [74]. For each leaf, the algorithm builds three sets of fractals according to their size. One set is made of leaves that are of equal size, and two others are made of leaves that are bigger, respectively smaller, than the current leaf. Then, by computing different inequalities, the algorithm determines whether the Lipschitz constant can be bound or not. So, here we have to include the size of fractals to their properties, $p_{\text{size}}(F_{(l,...)}, \mathcal{A}_{F_{(l,...)}}) = \sigma(F_{(l,...)})$, where σ is a measure of the size of a hyperrectangle [81]. In the pseudocode, the best fractal found so far (the best center) is denoted BSF, $\epsilon > 0$ is a small value defining how a score of a fractal should exceed BSF to be considered better. As in SOO and NMSO the scoring of a fractal is the objective value of its center, and there is no exploitation component. Most of the algorithm relies on the partitioning and the tree search. Two extensions of DIRECT, known as DIRECT-L [81], and DIRECT-R [73], slightly modify the algorithm 21 and the measure σ , to obtain different behaviors.

Algorithm 21 POHs

Inputs:

- | | |
|--|----------------------------|
| 1: $\text{OPEN} \triangleq \mathcal{L}_{\mathcal{T}(d)}$ | <i>Leaves</i> |
| 2: d | <i>Current depth</i> |
| 3: D | <i>Maximum depth</i> |
| 4: γ | <i>Scoring component</i> |
| 5: ϵ | |
| 6: BSF | <i>Best solution found</i> |
| 7: | |

Outputs: A *POHs*

- | | |
|--|-------------------|
| 8: $A \leftarrow \{\cdot\}$ | |
| 9: for each $a \in \text{OPEN}$ do | |
| 10: $I_1 = \{b \in \text{OPEN} \mid \sigma(b) < \sigma(a)\}$ | |
| 11: $I_2 = \{b \in \text{OPEN} \mid \sigma(b) > \sigma(a)\}$ | |
| 12: $I_3 = \{b \in \text{OPEN} \mid \sigma(b) = \sigma(a)\}$ | |
| 13: if $\gamma(a) \leq \gamma(b), \forall b \in I_3$ then | |
| 14: $\max_{I_1} = \max_{b \in I_1} \left(\frac{\gamma(a) - \gamma(b)}{\sigma(a) - \sigma(b)} \right)$ | |
| 15: $\min_{I_2} = \min_{b \in I_2} \left(\frac{\gamma(b) - \gamma(a)}{\sigma(b) - \sigma(a)} \right)$ | |
| 16: if BSF = 0 then | |
| 17: $\text{err} = -\epsilon + \frac{\gamma(\text{BSF}) - \gamma(a)}{ \gamma(\text{BSF}) } + \frac{\sigma(a) \cdot \min_{I_2}}{ \gamma(\text{BSF}) }$ | |
| 18: else | |
| 19: $\text{err} = -\gamma(a) + \sigma(a) \cdot \min_{I_2}$ | |
| 20: if $(\min_{I_2} - \max_{I_1} > 0) \wedge (\text{err} \geq 0)$ then | |
| 21: $A \leftarrow A \cup \{a\}$ | <i>a is a POH</i> |
| 22: $\text{OPEN} \leftarrow \text{OPEN} \setminus A$ | |
| 23: $\text{CLOSE} \leftarrow \text{CLOSE} \cup A$ | |
| 24: return A | |
-

4.6 Experimental setup

The objective of these comparisons is to illustrate the workability of our software framework by instantiating several popular algorithms within the *Zellij* framework. One can also evaluate their scalability and their sensitivity according to the different search components: fractal, tree search, scoring, exploration, and exploitation components. We summed up all 11 implemented algorithms and their different search components in table 4.2.

To illustrate the flexibility of *Zellij* we modified some search components from the previous algorithms. The Move-Up algorithm of FDA is replaced by a BFS consisting, at each iteration, of selecting the Q best nodes from $\mathcal{L}_{\mathcal{T}(d)}$. FDA-based algorithms using BFS are denoted, FDA-BFS and FDA-DBFS.

Similarly, BFS was applied to SOO(SOO-BFS). The advantage of using *BFS* instead of *BS* is that all fractals are kept. So in SOO-BFS, the entirety of the search space remains accessible; however, BFS might modify its rate of convergence.

We also test FDA and FDA-DBFS, a version with a deeper tree. The maximum depth D was set to 5 for FDA and FDA-BFS and to 10 for FDA-D and FDA-BFS. Additionally, we replaced the DTTB (γ) of FDA by a centered version, CDTTB, of it (FDA-C). The measure is centered on the best solution found so far,

$$\gamma : F, \mathcal{X}, f|_F(\mathcal{X}), P(F, \mathcal{A}_F) \rightarrow \min_{x \in \mathcal{X}} \left(\frac{f(x) - f(\text{BSF})}{\|x - \text{BSF}\|} \right) .$$

The optimization algorithms were evaluated on the BBOB benchmark from the Comparing Continuous Optimizer framework [103]. This benchmark is made of 24 functions with peculiar properties such as, separability, ill-conditioning, multi-modality and weak structured multi-modality. Each function has 15 different instances, and are available for 6 dimensions, $n \in \{2, 3, 5, 10, 20, 40\}$. COCO compares algorithms on the number of solved problems, under a given tolerance $\Delta f = 10^{-8}$, such that,

$$f(\hat{x}) \leq f(x^*) + \Delta f , \quad (4.29)$$

where x^* is the known global optimum and $f(\hat{x})$ is the acceptable optimum to consider a problem as solved. Then COCO computes a metric called the ERT, based on the number of problems solved and the budget of the optimization. Here, the budget, `budget`, is the number of function evaluations, and it depends on the dimensionality of the problem, `budget` = $10^4 \cdot n$, as in [57].

COCO is based on the number of problems solved. Hence, to extend the comparison, we also analyze the best solutions found for all problems, which might not follow the equation 4.29. To do so, we use the two-sided Wilcoxon signed-rank test and corresponding mean ranks for each of the 11 algorithms. We perform this test for each subclass of functions and dimension. Thus, we were able to see their reliability. We define an error rate, p -value, of 5% for the statistical test. The two hypotheses are:

- **H0:** The two samples come from the same distribution.
- **H1:** The two samples come from different distributions.

The two-sided Wilcoxon signed-rank test is based on the mean of the ranking of all 11 algorithms for each subclass of functions and for each dimension. Summing these means and taking the lowest results is not enough to determine if an algorithm is better than another. Indeed, giving a rank to an algorithm for a function is arbitrary. An algorithm ranked second is not necessarily and significantly worse than the first one. We resumed the comparison

Table 4.2: Instantiated algorithms using Zellij

Algo- rithm	π	τ	<i>Explor</i>	<i>Exploit</i>	γ	Depth	Source
FRAC- TOP	n -cube	BFS	GA	SA	Belief	4	[46]
FDA	n -sphere	Move-Up	PHS	ILS	DTTB	5	[192]
FDA-BFS	n -sphere	BFS ^a	PHS	ILS	DTTB	5	This work
FDA-C	n -sphere	Move-Up	PHS	ILS	CDTTB	5	This work
FDA-D	n -sphere	Move-Up	PHS	ILS	DTTB	10	[192]
FDA- DBFS	n -sphere	BFS ^a	PHS	ILS	DTTB	10	This work
SOO	3-section	alg.19 ^b	Center	\emptyset	\emptyset	h_{\max} ^c	[188]
SOO-BFS	3-section	BFS ^a	Center	\emptyset	\emptyset	h_{\max} ^c	This work
NMSO	3-section	See [57]	Center	\emptyset	\emptyset	600	[188]
DIRECT	3-section	All POH	Center	\emptyset	\emptyset	600 ^d	[135]
DIRECT- L	3-section	1 POH per level	Center	\emptyset	\emptyset	600 ^d	[81]
DIRECT- R	3-section	Adaptive POH	Center	\emptyset	\emptyset	600 ^d	[73]

^aAt each iteration $Q = n(\text{dimension})$ nodes are returned.

^bIf the best fractal at a given level is worse than one from previous levels, then it is not selected [188].

^cHere $h_{\max} = 10\sqrt{\log(n10^4)^3}$ [50].

^dThe maximum depth is set to 600, as the `maxdeep` variable in the original FORTRAN implementation. In the original code, a `maxdiv` variable, set to 3000, limits the number of successive decomposition of a fractal. In *Zellij*, when the difference between an infimum and a supremum is inferior to a value ϵ set to $1e-16$, the fractal cannot be decomposed anymore.

between ranks and statistical test in Figures 4.26, Figure 4.27 and Figure 4.28. These figures can be read column by column. Each column represents comparisons of an algorithm (column label) with all other 10 algorithms (row labels). As an example, in Figure 4.28, for dimensions 3, if we focus on DIRECT-L (last column), we can compare it with SOO (sixth row). The color indicates that there is statistical evidence that DIRECT-L is better than SOO. In Figure 4.26, Figure 4.27 and Figure 4.28, there are three color codes:

- **Solid-Grey:** $\alpha > 0.05$, we cannot reject the null hypothesis.
- **Gridded-Green:** $\alpha \leq 0.05$ and $\text{rank}(\text{column}_i) < \text{rank}(\text{row}_j)$. We can reject the null hypothesis, and the algorithm with the label of the column i has a lower rank (is better) than the algorithm of the row j .
- **Dotted-Red:** $\alpha \leq 0.05$ and $\text{rank}(\text{column}_i) > \text{rank}(\text{row}_j)$. We can reject the null hypothesis, and the algorithm with the label of the column i has a higher rank (is worse) than the algorithm of the row j .

All experiments were carried-out on Grid'5000 [13], a large-scale and flexible testbed for experiment-driven research. We used a multi-CPU cluster containing Intel Xeon Gold 5220 of 18 cores, each CPU has 96 GiB of RAM.

4.7 Results analysis and discussion

4.7.1 Sensitivity to the dimensionality

The initial observations from, figures 4.24, 4.25, 4.27 and 4.28, reveal that for dimensions from 2 to 40, while FDA-based algorithms perform poorly on low dimensional problems, they can maintain certain performances when dimension scales. Except for FDA-D and FDA-DBFS always performing worse than FDA algorithms with a shallow tree. This confirms results obtained in [192], illustrating that the deeper the tree, the lower the performances of FDA. For dimension 40, performances of FDA-based algorithms are comparable to SOO, SOO-BFS and NMSO.

DIRECT-based algorithms are one of the best performing ones in low dimensionalities, with performances comparable to SOO and NMSO. However, it scales poorly, except for multi-modal functions, where all DIRECT algorithms perform better than FDA ones. This can be explained by the overconfidence of FDA to exploit fractals. DIRECT remains worse than SOO and NMSO.

It is important to notice the low number of successes in solving high dimensional problems in figures 4.24 and 4.25, 4.27. Therefore, the two-sided Wilcoxon signed-rank tests illustrated in figures 4.26, 4.28, and 4.28, provide additional in-depth analysis.

So, the algorithms that scale the best up to dimension 40, appear to be SOO, NMSO and FDA. However, because FDA and SOO-BFS are greedy algorithms, they perform poorly in lower dimensions because they seem to be easily trapped into local optima.

4.7.2 Sensitivity to the tree search

In the following lines, we focus on FDA-BFS, SOO-BFS and FDA-BFS. Concerning, SOO-BFS, it is clear that the original tree search algorithm performs better than the BFS. So, replacing this search component from SOO by one that is too greedy has a significant impact on the performances of the algorithms. The same observations can be made for FDA, FDA-BFS and FDA-DBFS.

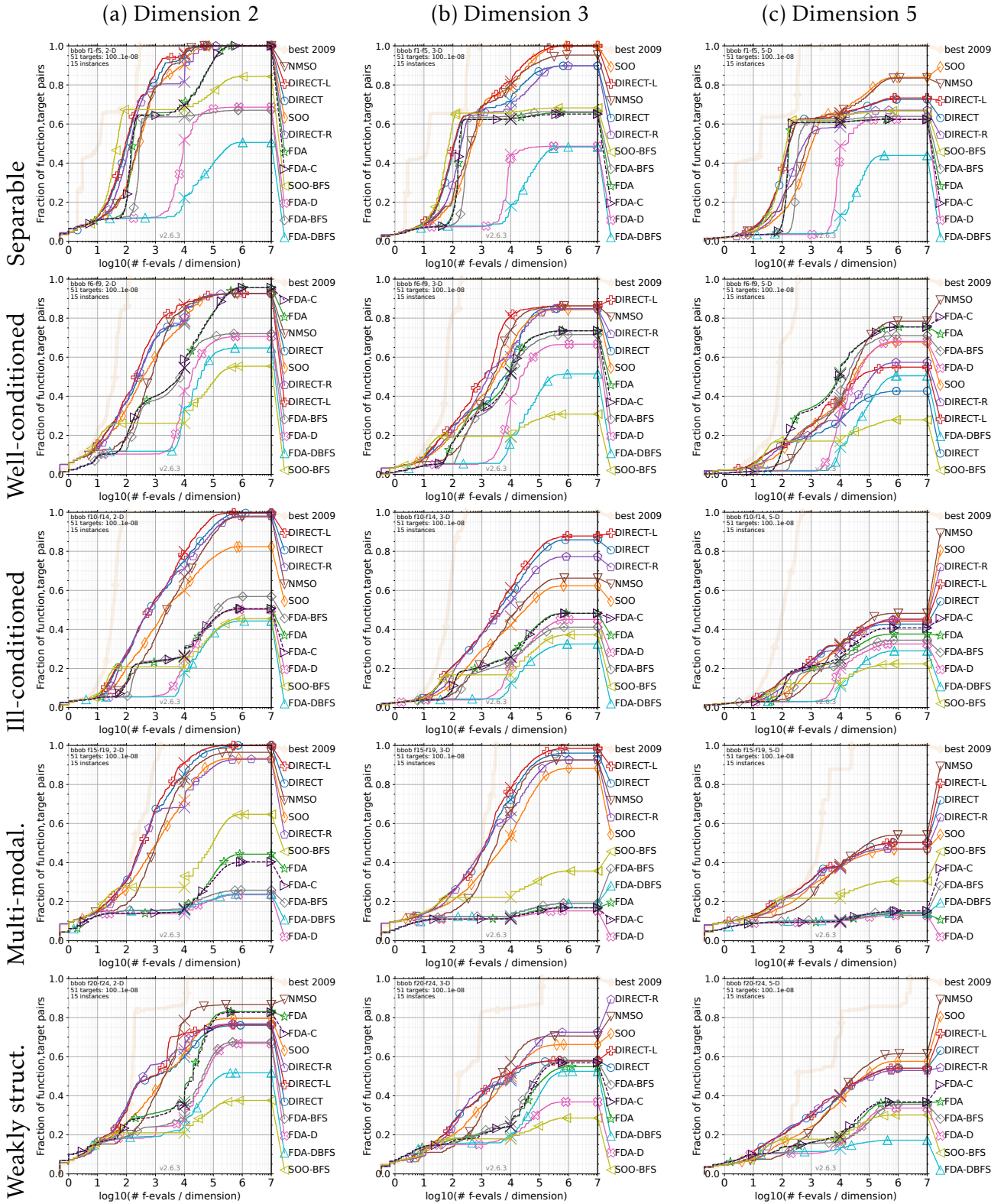


Figure 4.23: Empirical cumulative distribution according to the budget divided by the dimensions for each functions' subclass and dimension 2, 3 and 5.

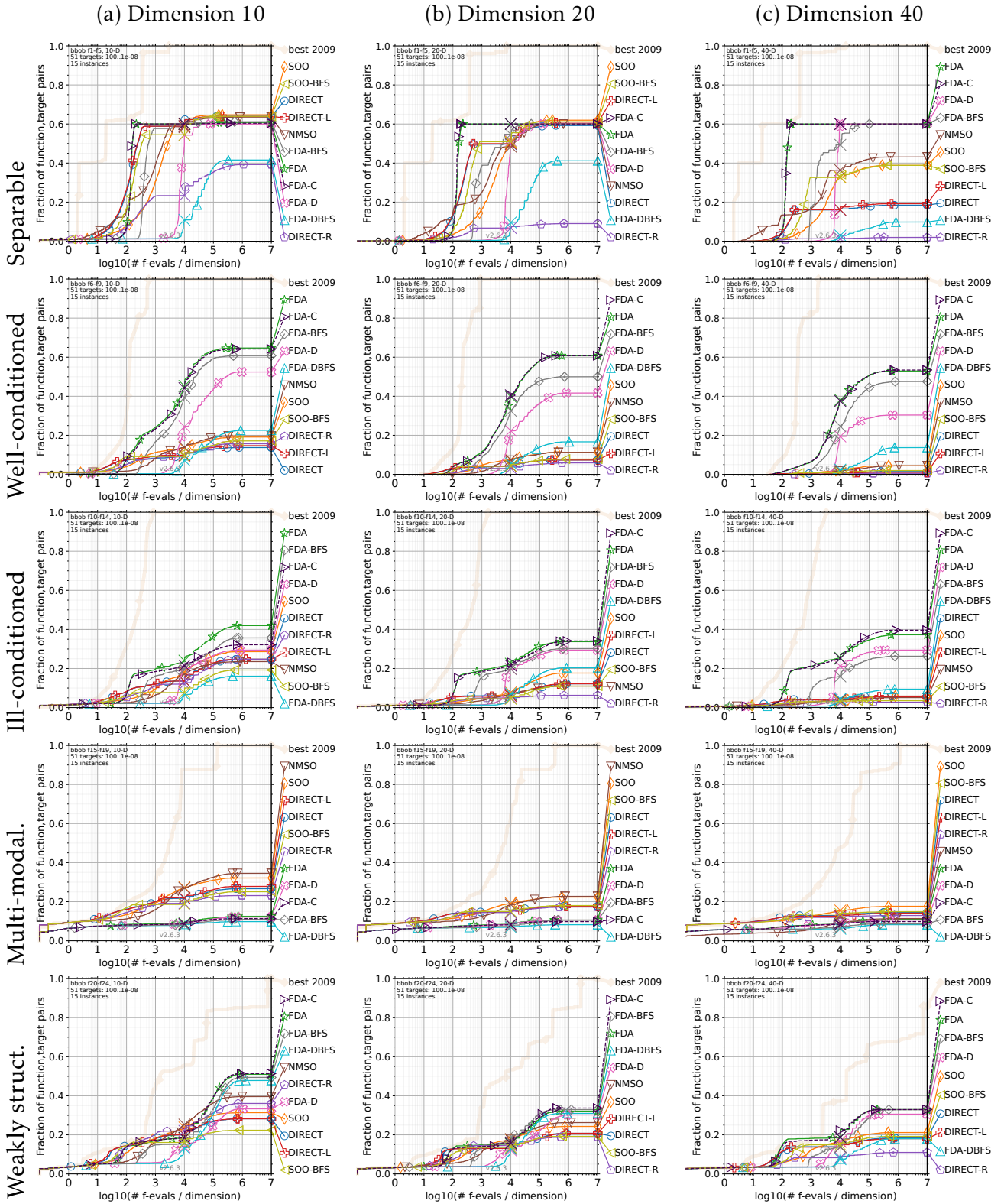


Figure 4.24: Empirical cumulative distribution according to the budget divided by the dimensions for each functions' subclass and dimension 10, 20 and 40.

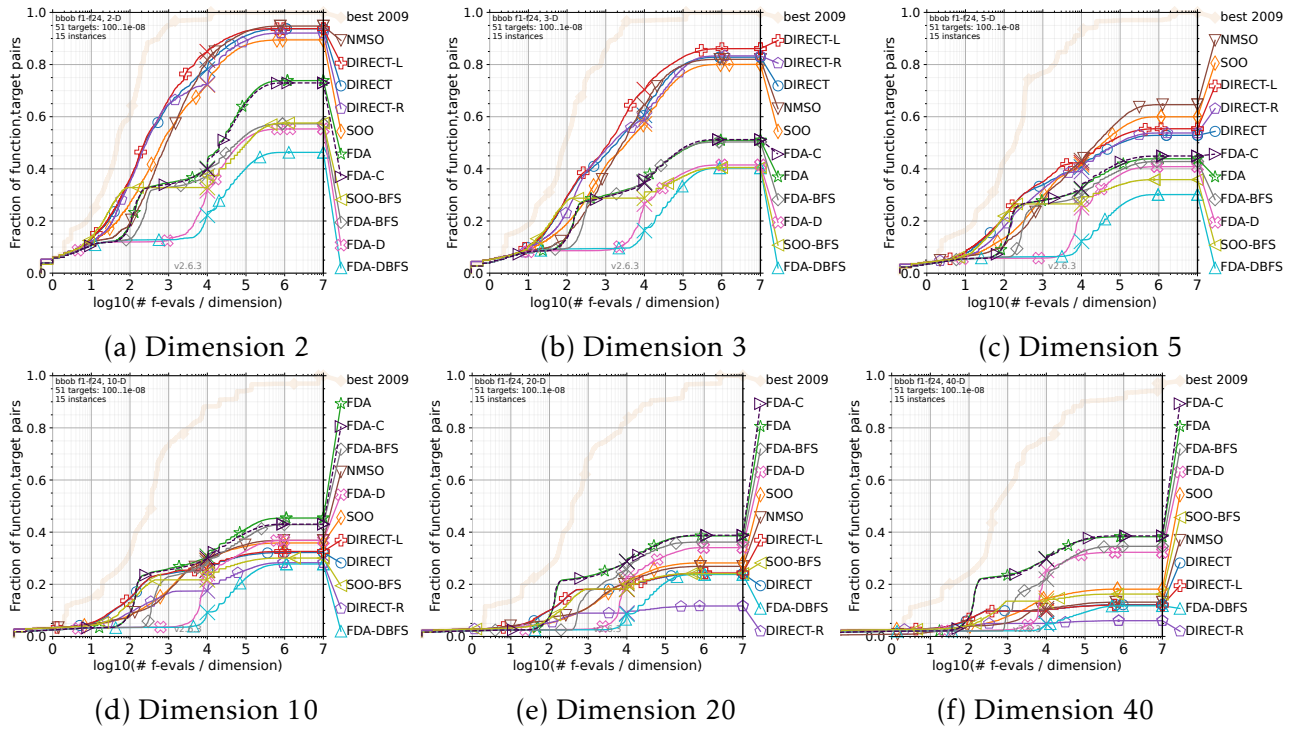


Figure 4.25: Empirical cumulative distribution according to the budget divided by the dimensions for all functions and dimensions.

Concerning DIRECT, DIRECT-R and DIRECT-L, it appears that biasing DIRECT toward exploitation (DIRECT-L) helps to scale the algorithm to higher dimensions. While in Figure 4.25, DIRECT-L solves almost always more problems than DIRECT, their performances (in ERT according to budget) for dimension 40 are comparable. To break the tie, the two-sided Wilcoxon signed-rank tests in figures 4.26, 4.28 show that DIRECT-L is more likely to find a better solution than DIRECT, even if it cannot always solve the problem. However, DIRECT-R seems to be worse than DIRECT and DIRECT-L.

This section illustrates how fractal-based decomposition algorithms are sensitive to the tree search algorithm and its parametrization. So, particular attention is needed to the design of this search component.

4.7.3 Sensitivity to function properties

Concerning *separable* functions, NMSO and SOO are two of the top algorithms, no matter the dimensions. When comparing the ERT in Figures 4.23 and 4.24, their performances in solving low dimensional problems are comparable to DIRECT-L and DIRECT. However, when dimensions grow, FDA-based algorithms, except FDA-D and FDA-DBFS, benefit from their exploitation components to scale. When ERT and statistical tests are compared together, the exploitation component makes FDA competitive with NMSO and SOO, yet NMSO seems slightly superior.

For low dimensional (2, 3 and 5) and *well-conditioned* problems, SOO, DIRECT, NMSO and FDA seems to perform similarly with a dominance of NMSO, SOO and DIRECT-L, in figure 4.23. However, when the dimension scales, FDA becomes better, except for dimension 40, for which SOO, FDA and NMSO have similar behaviors.

Ill-conditioned problems are harder, and algorithms have difficulties scaling when the dimension grows. NMSO, DIRECT-L and SOO seem to dominate low dimensional problems. And, once again, for higher dimensions, FDA scales better. For dimension 40, FDA, SOO

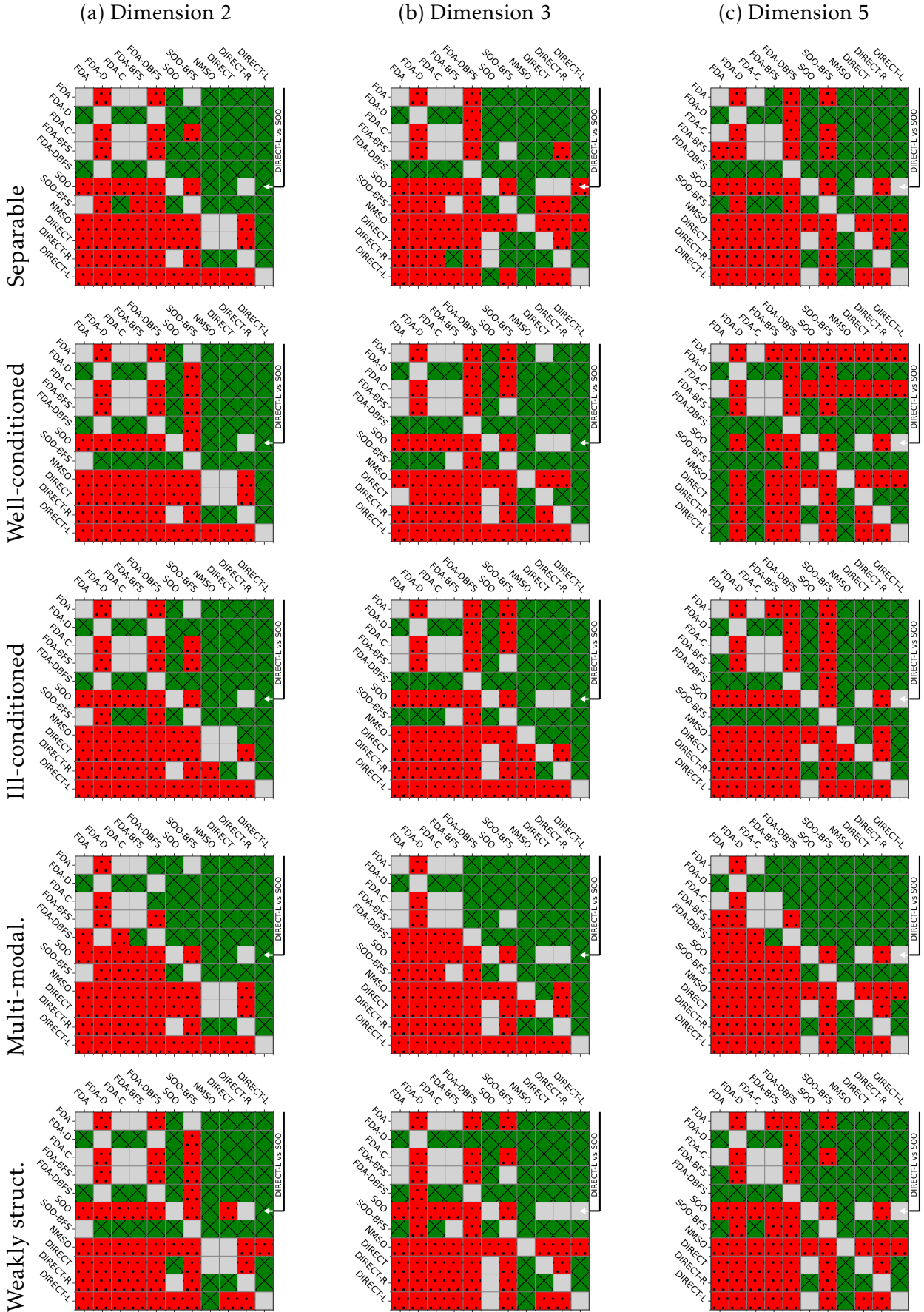


Figure 4.26: Pair Wise Wilcoxon test for each functions' subclass and dimensions 2, 3 and 5. **Solid-Grey:** Statistically unsignificative ($\alpha > 0.05$). **Gridded-Green:** Better. **Dotted-Red:** Worse.



Figure 4.27: Pair Wise Wilcoxon test for each functions' subclass and dimensions 10, 20 and 40. **Solid-Grey:** Statistically unsignificative ($\alpha > 0.05$). **Gridded-Green:** Better. **Dotted-Red:** Worse.

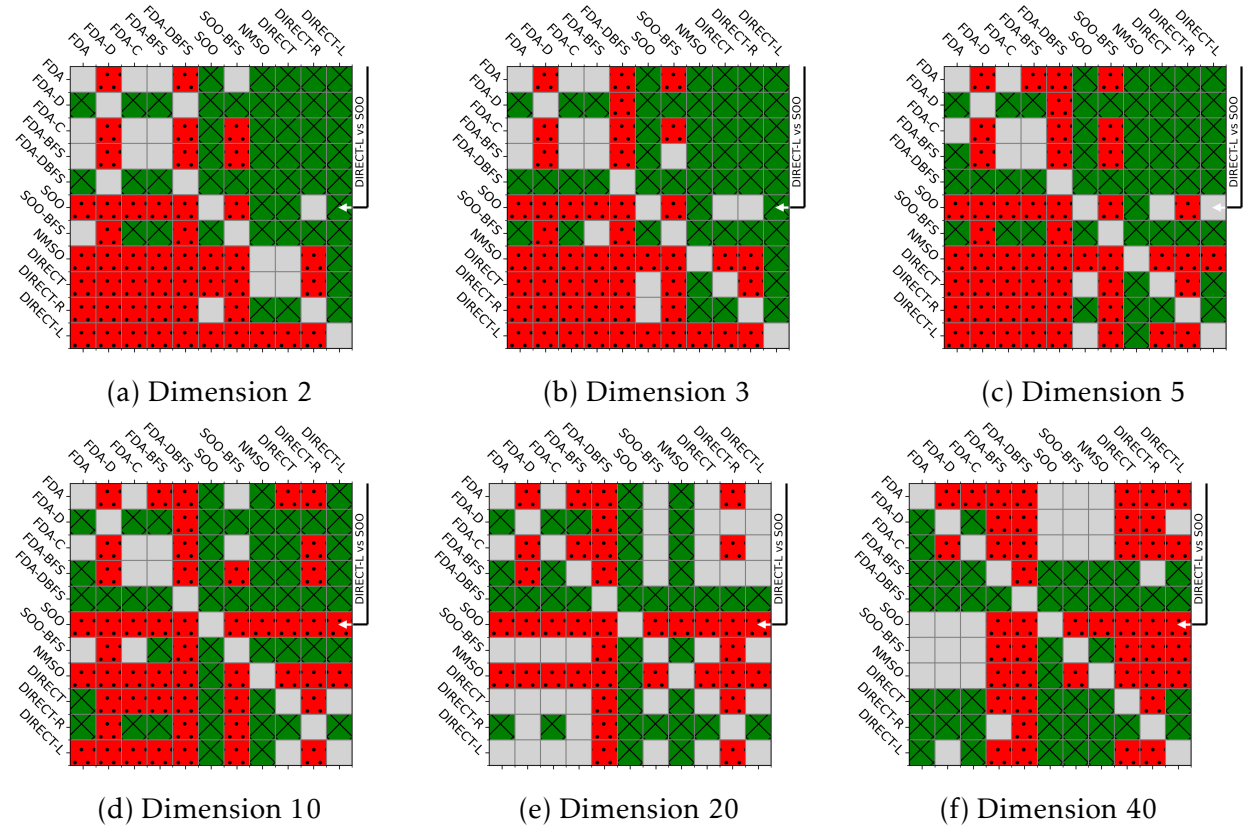


Figure 4.28: Pair Wise Wilcoxon test for all functions and dimensions. **Solid-Grey**: Statistically insignificant ($\alpha > 0.05$). **Gridded-Green**: Better. **Dotted-Red**: Worse.

and NMSO have similar performances, even if they solve fewer problems than FDA. This can be explained by the exploitation component of FDA, allowing the algorithm a faster convergence.

As mentioned in [192], the ILS of FDA seems to take advantage of separable functions, and can be easily dragged toward local optima. The *multi-modal* problems illustrate these behaviors, while SOO and NMSO are the best performing algorithms in high dimensions (10, 20, and 40), followed by DIRECT-based algorithms. The same behaviors are observable for *weakly structured multi-modal problems*, except that FDA performs better than DIRECT-based algorithms for dimension 40 and has similar performances compared to SOO and NMSO.

Finally, it appears that to design an efficient fractal-based algorithm, an exploitation component allows to significantly improve performances and convergence on certain problems. However, a greedy exploitation component, unable to escape from local optima, can decrease the performance of an algorithm. Thus, the balance between exploration and exploitation seems to rely on combining an exploitation component, an efficient partition of the search space, and a not too greedy tree search.

4.8 Fractal decomposition based on Latin Hypercubes

In this section, we introduce a new FBD algorithm based on a LHS decomposition. In the previous sections, we noticed that there is no perfect fractal geometry to partition the search space Ω . A hypercubic partition involves 2^n children, while FDA with hyperspheres and despite the inflation, covers less and less space w.r.t. higher dimensionality. SOO or DIRECT are proper preservative algorithms, but they lack scalability because of the trisection partitioning, which requires deeper and deeper 3-ary rooted trees to scale.

Thus, LHS partitioning could be an alternative to hyperspheres. It keeps the geometrical properties of hypercubes, while a child reduces exponentially the space covered by the ancestor. Indeed, in SOO a child only reduces one dimension of its ancestor by 3, whereas a child hypercube from a 3-dimensional Cartesian 3-grid reduces all dimensions by 3 at once. However, it highly sacrifices ancestor space, as it only selects $g > 1$ hypercubes among g^n hypercubes from a grid of size g at dimension n .

4.8.1 Latin Hypercube Sampling

LHS is a stratified Monte Carlo sampling method [127]. In LHS, the range of each variable is divided into equally probable intervals, forming a n -dimensional grid of size g . A value is then sampled randomly within each interval. LHS ensures that each interval for every variable is sampled only once. This reduces the likelihood of clustering and enhances coverage across the entire input space. The result is a stratified sampling method where the points are distributed more uniformly than in simple random sampling, making LHS an efficient sampling method for high-dimensional spaces with a limited number of samples.

A LHS within the unit hypercube $[0,1]^n$ of dimension n , can be described by d i.i.d. random permutations σ_k of $\{0, \dots, g-1\}$, with $1 \leq k \leq d$, $g \in \llbracket 2, N \rrbracket$ and N a finite upper bound [167]. Then the bounds $L_i \times U_i$ of all sampled hypercube $1 \leq i \leq g$ are given by:

$$\forall 1 \leq i \leq g, \begin{cases} L_i &= \frac{1}{g} \times [\sigma_1[i], \dots, \sigma_d[i]]^\top \\ U_i &= \frac{1}{g} + L_i \end{cases}. \quad (4.30)$$

By using LHS fractals, we have the hand on the partition size and the size of the sampled hypercubes, as g is a tunable parameter. Usually in LHS, a single i.i.d. point X_i is uniformly sampled within each sampled hypercube such that $X_i \sim \mathcal{U}(L_i, U_i)$. But, for the application within a FBD algorithm, we consider this part as the *Explor* component, and we only evaluate the center of the sampled hypercube.

4.8.2 Nested Latin Hypercube Partition

We now consider nested LHS as a FBD algorithm prototype. Then, the bounds of g children from permutations σ_k w.r.t. the bound of their direct ancestor \mathcal{A} , regarding the length of the ancestor hypercube $l_{\mathcal{A}}$, and its lower bounds $L_{\mathcal{A}}$, are given by:

$$\forall 1 \leq i \leq g, \begin{cases} L_i &= L_{\mathcal{A}} + \frac{l_{\mathcal{A}}}{g} \times [\sigma_1[i], \dots, \sigma_d[i]]^\top \\ U_i &= \frac{l_{\mathcal{A}}}{g} + L_i \end{cases}. \quad (4.31)$$

We can extend equation 4.31 to a tree $\mathcal{T}^{(d)}$ of depth d . Thus, for a fixed grid size g from the unit hypercube (root), the length of a single Latin hypercube at depth d is given by $l = 1 \div g^d$. At each refinement, we are generating a grid of g^n hypercubes among which only g

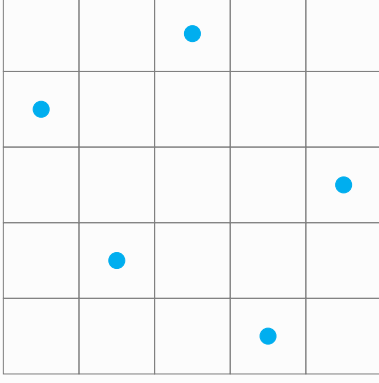
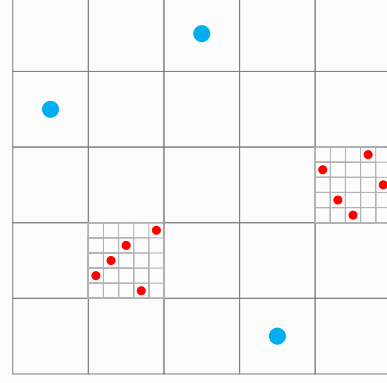
Example: LHS and nested LHSFigure 4.29: LHS with a grid size of 5×5 .

Figure 4.30: Nested LHS.

(Left) Example of a LHS with $g = 5$, the g permutations are $\{\sigma_1, \sigma_2\} = \{\{0, 1, 2, 3, 4\}, \{4, 2, 5, 1, 3\}\}$. (Right) Example of 2 nested LHS with $g = 5$.

hypercubes are selected. Then at each refinement, a proportion of $1 - \frac{1}{g^{(n-1)}}$ of the ancestor space is sacrificed. Through a hierarchical refinement, all the sides of the root fractal Ω are reduced exponentially, while exponentially sacrificing space.

We assume continuity of the function $f : \Omega \rightarrow \mathbb{R}$, and Ω is a non-empty compact metric space from \mathbb{R}^n , as stated in previous sections. By construction, any fractal is also a compact from \mathbb{R}^n . Consequently, by the extreme value theorem, the function f is bounded within any fractal F ; $\exists l, u \in F$, s.t. $f(l) = \inf_{x \in F} f(x)$ and $f(u) = \sup_{x \in F} f(x)$. By definition, the global optimum x^* within $F_{(1,0,1)} = \Omega$, is defined by $x^* \triangleq \sup_{x \in \Omega} f(x)$. For simplicity, we suppose that x^* cannot be on a border. Otherwise, x^* can belong to all fractals from $\pi_{\text{grid}}(\Omega)$. LHS divides the search space into a n -grid of size g . The first step is to partition Ω with g^n closed and bounded hypercubes which are *disjoint at the borders*; we write the grid-partitioning function π_{grid} , and $\forall F_i, F_j \in \pi_{\text{grid}}(\Omega)$, $F_i \neq F_j$, $(F_i \cap F_j = \emptyset)$. Then, the optimal cell F^* from π_{grid} is defined by:

$$\exists! F^* \in \pi_{\text{grid}}(\Omega), \sup_{x \in F^*} f(x) = f(x^*) .$$

The partitioning function of a LHS decomposition, π_{LHS} , selects a subset from π_{grid} ; $\pi_{\text{LHS}}(F) \subset \pi_{\text{grid}}(F)$. If we consider x^* as unique and without prior knowledge about the distribution of x^* w.r.t. Ω , the probability that a fractal $F \in \pi_{\text{grid}}(\Omega)$ contains x^* , is given by:

$$\mathbb{P}(x^* \in F \mid F \in \pi_{\text{grid}}(\Omega), g, n) = \frac{1}{g^n} . \quad (4.32)$$

Because selecting all g^n fractals from π_{grid} does not scale with n , LHS samples g fractals from π_{grid} with the function π_{LHS} . In LHS each cell has the same probability of being selected. Thus, the probability that x^* belongs to one fractal from $\pi_{\text{LHS}}(\Omega)$ is given by:

$$\mathbb{P}(x^* \in \pi_{\text{LHS}}(\Omega) \mid g, n) = \frac{1}{g^{n-1}} . \quad (4.33)$$

Proof. Considering a LHS built by n permutations $\sigma_{[1,n]}$ of the set $\{0, \dots, g-1\}$. We write the matrix formed by stacking all σ_k as $\Sigma = [\sigma_1, \dots, \sigma_n]^\top$, where each column describes a single hypercube. Each permutation σ_k has $g!$ different designs, and thus Σ has $(g!)^{n-1}$ designs, as

the first permutation σ_1 can be fixed. For a given cell – hypercube – F_k described by a column within Σ , the position of this column is not relevant to describe F_k . Then, for a fixed Σ , all rearrangements of the columns describe the same LHS configuration. We have $g!$ columns rearrangement. So, the total number of LHS designs is given by:

$$\text{card}(\text{All}) = \frac{(g!)^{n-1}}{g!}.$$

To determine the probability of selecting a cell, we can fix this cell, i.e. forbid a value of the sets $\{0, \dots, g-1\}$ for the d permutations. Then, the number of LHS configurations, including this fixed cell, is given by:

$$\text{card}(1 \text{ fixed cell}) = \frac{((g-1)!)^{n-1}}{g!}.$$

Finally, the probability of a fixed cell $F \in \pi_{\text{grid}}(\Omega)$ to be selected by $\pi_{\text{LHS}}(\Omega)$ is given by:

$$\mathbb{P}(F \in \pi_{\text{LHS}}(\Omega) \mid F \in \pi_{\text{grid}}(\Omega), g, n) = \frac{\text{card}(1 \text{ fixed cell})}{\text{card}(\text{All})} = \frac{1}{g^{n-1}}.$$

□

For now, without any refinement, we can notice that a design based on LHS partitioning cannot scale in high dimensions, as $\lim_{n \rightarrow \infty} 1/g^{n-1} = 0$. We consider the optimal fractal F^* from the first refinement of Ω as the coarse-grain neighborhood of x^* . The probability of being within the biggest possible neighborhood of a nested LHS partition, decreases exponentially regarding the dimensionality. We can easily deduce for a nested LHS after d refinements, that the probability of x^* of being within a fractal $F_{(d, \dots)}$ at level d , and child of a fractal $F_{(d-1, \dots)}$, converges toward 0 even faster. Indeed, we suppose that at each level we know which fractal from this level is optimal, and that we directly refine it, without refining non-optimal fractals. Therefore, the probability that π_{LHS} samples the optimal fractal at each level is given by:

$$\mathbb{P}(x^* \in \pi_{\text{LHS}}(F_{(d-1, \dots)}) \subset \pi_{\text{LHS}}(F_{(d-2, \dots)}) \subset \dots \subset \pi_{\text{LHS}}(F_{(1,0,1)}) \mid g, n) = \left(\frac{1}{g^{n-1}} \right)^d. \quad (4.34)$$

To counterbalance this, instead of using a destructive partition function, meaning that the ancestor is removed from the OPEN list of the tree search component, suppose we can apply multiple times π_{LHS} to Ω . We compute the probability of **not** selecting the optimal fractal F^* with s multiple i.i.d. calls to $\pi_{\text{LHS}}(\Omega)$ with:

$$\mathbb{P}\left(x^* \notin \bigcup_{i=0}^s \pi_{\text{LHS}}(\Omega) \mid g, n\right) = \left(1 - \frac{1}{g^{n-1}}\right)^s.$$

We now consider, the minimal grid size $g = 2$, i.e. the biggest coarse-grain neighborhood around x^* . We want to determine s such that the probability of not selecting F^* with $\pi_{\text{LHS}}(\Omega)$ is inferior to $1/2$; $\mathbb{P}\left(x^* \notin \bigcup_{i=0}^s \pi_{\text{LHS}}(\Omega) \mid g = 2, n\right) \leq 1/2$. The goal is to determine a minimal budget proportional to s for a given dimension n . This is illustrated within figure 4.31 for different values of g . The black-dotted line illustrates the budget used in the previous experimental section with COCO. We suppose that only one point is sampled within each fractal.

Hence, we can deduce that at the second level (the first refinement of Ω), sampling the

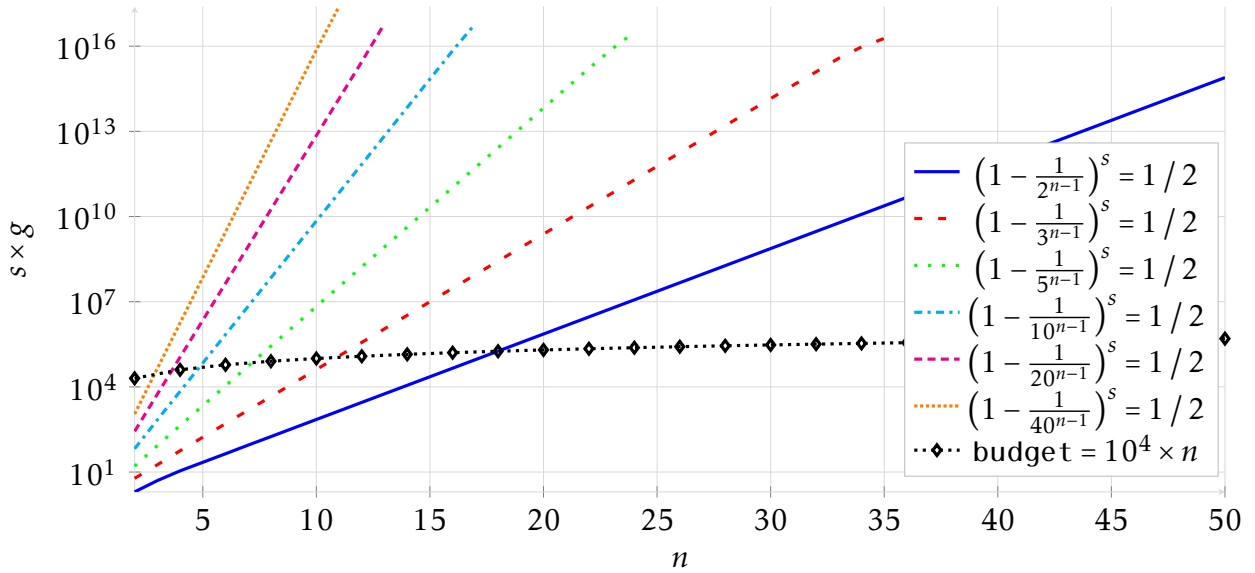
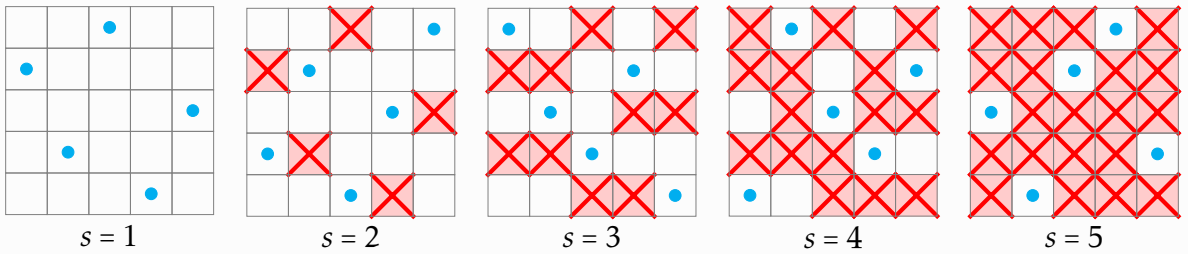


Figure 4.31: Number of minimum calls s to $\pi_{\text{LHS}}(\Omega)$ multiplied by the number g of sampled fractals for each calls, so that the probability that x^* does not belong to sampled fractals is lower than $1/2$; $\mathbb{P}\left(x^* \notin \bigcup_{i=0}^s \pi_{\text{LHS}}(\Omega) \mid g, n\right) \leq 1/2$ with $g \in \{2, 3, 5, 10, 20, 40\}$.

biggest possible fractal containing x^* is far from being trivial, and does not scale with the dimensionality as it requires an exponential number of calls to π_{LHS} . A solution to bound s would be to consider forbidden positions within the grid. These positions are determined by the previously sampled fractals from π_{LHS} . We can consider LHS as the g -non-attacking rooks problem [5, 179]. The problem of iteratively sampling LHS while including forbidden positions is known as the problem of *arrangement with forbidden positions*. However, such an approach is untractable. First, including forbidden positions would involve a bottleneck as the number of possible fractals within a grid is exponential, while we are only sampling $s \cdot g$ fractals. Secondly, including forbidden positions would complicate the LHS sampling, as it involves saving an exponential number of positions to sample without collisions. Computing a probability of collision w.r.t. to s involves knowing all previous sampled fractals in order to use the inclusion-exclusion principle [179].

Example: Forbidden positions



4.8.3 Other components

Because we need an exponential number of fractals to maximize the probability that x^* belongs to at least one of the fractals at level 2, we have decided to only sample the center of each fractal. Furthermore, because at each iteration g fractals are evaluated, we can use the information gathered among fractals and share it using inheritance. The information

is leveraged to compute a form of UCB inspired by [283]. Regarding a child fractal $c_i \in \mathcal{C}_F$, $\mathcal{C}_F = \pi_{\text{LHS}}(F)$, we write the center of c_i as \mathcal{X}_{c_i} :

$$\mathcal{X}_{c_i} = \left\{ \frac{U_{c_i} + L_{c_i}}{2} \right\},$$

with U_{c_i} and L_{c_i} as the upper and lower bounds of c_i . So, the scoring of c_i is given by:

$$\forall c_i \in \mathcal{C}_F, \gamma(c_i, f(\mathcal{X}_{c_i})) = f(\mathcal{X}_{c_i}) + \sqrt{2 \log \left(\frac{\pi^2 N^2}{6\eta} \right) \text{Var}(\{f(x)\}_{x \in \text{Explor}(\pi_{\text{LHS}}(F))})},$$

with $\eta \in (0, 1)$ and N the number of function evaluations. The objective is to build a heuristic inferring the variance within a child fractal according to the variance within the ancestor. This variance relies on all points computed within all child fractals of $\pi_{\text{LHS}}(\Omega)$. Here, as we consider a maximization problem, the scoring is maximized. Otherwise, the variance should be subtracted to $f(\mathcal{X}_{c_i})$.

As we cannot efficiently ensure that x^* belongs to a fractal, we adopt the same exploitation strategy as FDA, i.e. the ILS which is only bounded to Ω and not to a child fractal. Instead of applying the ILS once a fractal has reached the maximum depth D of the tree $\mathcal{T}^{(D)}$, we divide the algorithm into a partition–exploration and exploitation phases. We define a balance of the budget between these two phases. Once the partition–exploration phase ends, the fractals are sorted, and the ILS is iteratively applied to the best ones until no budget is left. We fixed the budget allocated to the partition–exploration phase at 20%.

We use the same tree search as SOO described in algorithm 19, except that Ω is always added by default to the selected fractals, so the algorithm can generate children of Ω at each iteration. Furthermore, we fix the maximum depth D of the tree with $D = \left\lceil \log \left(\frac{l}{g} \right) \right\rceil$, with l the minimum acceptable length of the deepest fractal. This allows us to decide what is an acceptable neighborhood for the ILS according to g . For the following experiments, we fixed $l = 10^{-4}$.

4.8.4 Experimentation

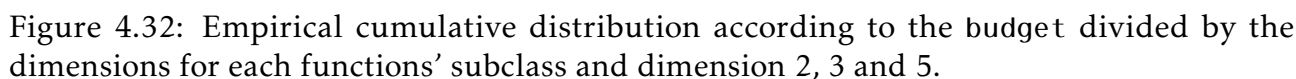
To ensure the relevance of our approach, we compare it to a random multi-start ILS. Meaning that the partition–exploration is replaced by random sampling with the same budget as the LHS-based approach. We compare it to an exploration–exploitation budget tradeoff of 1%, 20%, and 70% (1RndILS, 20RndILS, 70RndILS). The approach is also compared to FDA, SOO and DIRECT. We are testing 4 different grid sizes, $g \in \{2, 6, 10, 20\}$, and different values of $\eta \in \{0.05, 0.5, 0.99\}$. The experiment names are encoded as follows: 20g2HI, for 20% exploration–exploitation tradeoff, g2 for $g = 2$ grid size, and HI for High Impact η (LI: Low Impact, MI: Medium Impact).

In figures 4.32, 4.33, 4.34 and 4.35, we clearly notice how our LHS-based approach scales. At low dimensions $n \in \{2, 3, 5\}$, the LHS-based algorithms with a small grid size $g \in \{2, 6\}$ perform better than FDA and random multi-start ILS. We can assume that nested LHS performs better than FDA, as it does not definitely discard space. All the space is still accessible, but stochastically. But for these dimensions, the LHS-based algorithms have weaker performances than DIRECT and SOO. At dimension $n = 10$, LHS-based algorithms become better than DIRECT, LHS algorithms are still better than random multi-start ILS and FDA. For dimensions $n \in \{20, 40\}$, LHS-based algorithms have statistically equivalent performances compared to SOO. They are still better than FDA, DIRECT, 1RndILS and 70RndILS. But there is no statistical difference with 20RndILS. In other words, for such high

dimensions, the LHS-based approach is losing its ability to efficiently cover relevant subspaces of Ω . The fact that 1RndILS and 70RndILS perform worse, illustrates the difficult tradeoff between exploration–exploitation or partition–exploration–exploitation. These results are coherent with the theoretical behaviors depicted in figures 4.31 where at the first refinement and with low grid sizes, the probability that $x^* \in \pi_{\text{LHS}}$ with a probability of 50% becomes almost unreachable within the given budget after dimension 10.

Concerning the number of problems solved in figures 4.32, 4.33 and 4.34, we clearly distinguish that FDA performs the worst between $n = 2$ and $n = 5$. Then, for dimensions $n \geq 10$, we start distinguishing the lower performances of SOO and DIRECT. It is worth noticing that FDA and LHS-based algorithms perform well on convex–separable functions and well-conditioned ones, with 50 to 60% of these problems solved. Meaning that these algorithms struggle when small variations of a point x induce high variations of the response $f(x)$ (less than 40% problems solved for ill-conditioned functions). The most challenging functions are multi-modal ones; this could indicate that these algorithms are easily trapped by global optima. Indeed, as FDA and LHS-based algorithms heavily rely on the naive ILS algorithm, if this algorithm gets trapped, then it is less likely to reach the global optimum. This also explains the good performances on convex–separable functions, as reaching x^* can be done via a naive hill-climbing algorithm.

Concerning the grid size, it is clear that a too high g results in lower performances, which can also be explained by the previous theoretical investigations of nested Latin Hypercube partitioning.



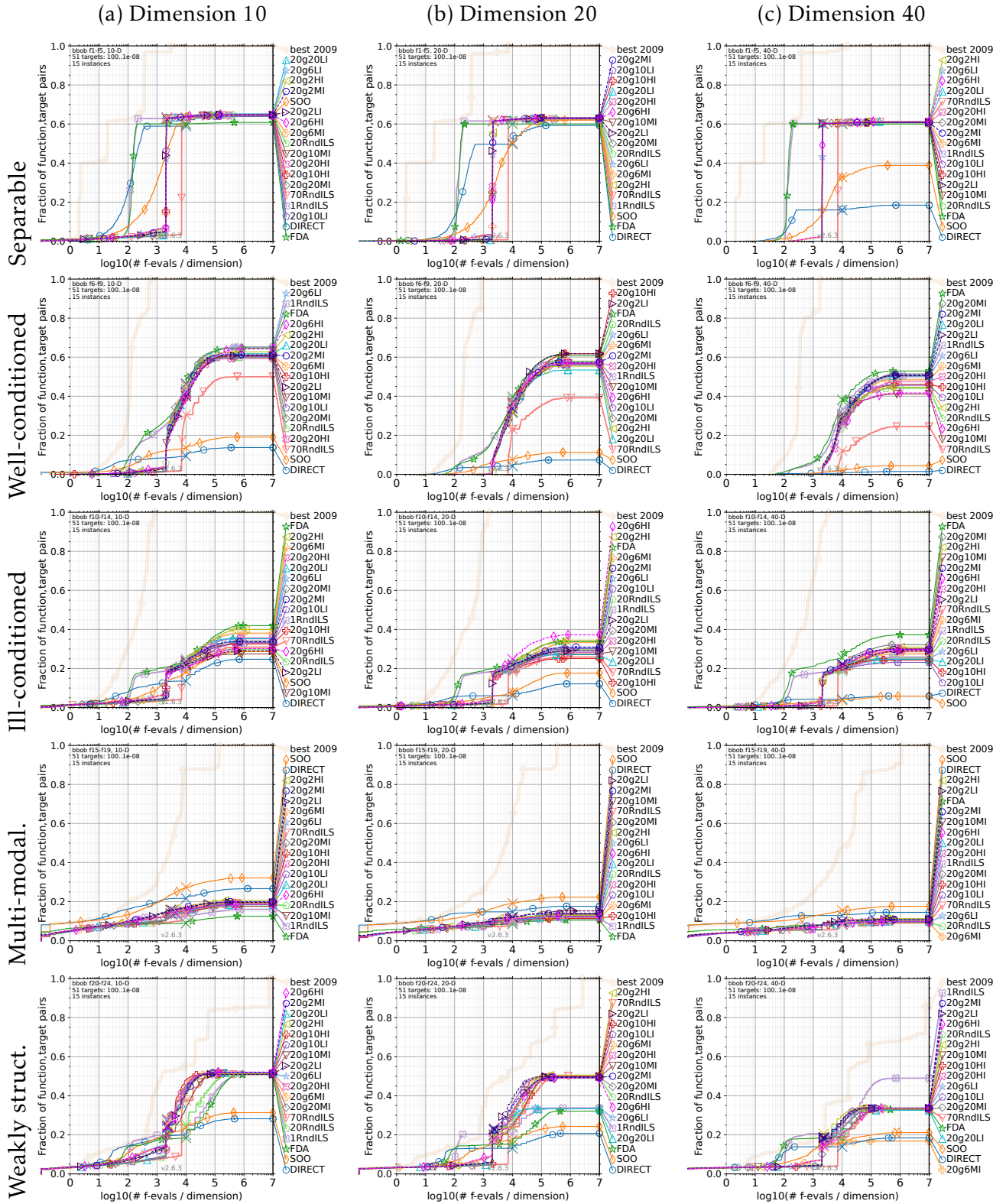


Figure 4.33: Empirical cumulative distribution according to the budget divided by the dimensions for each functions' subclass and dimension 10, 20 and 40.

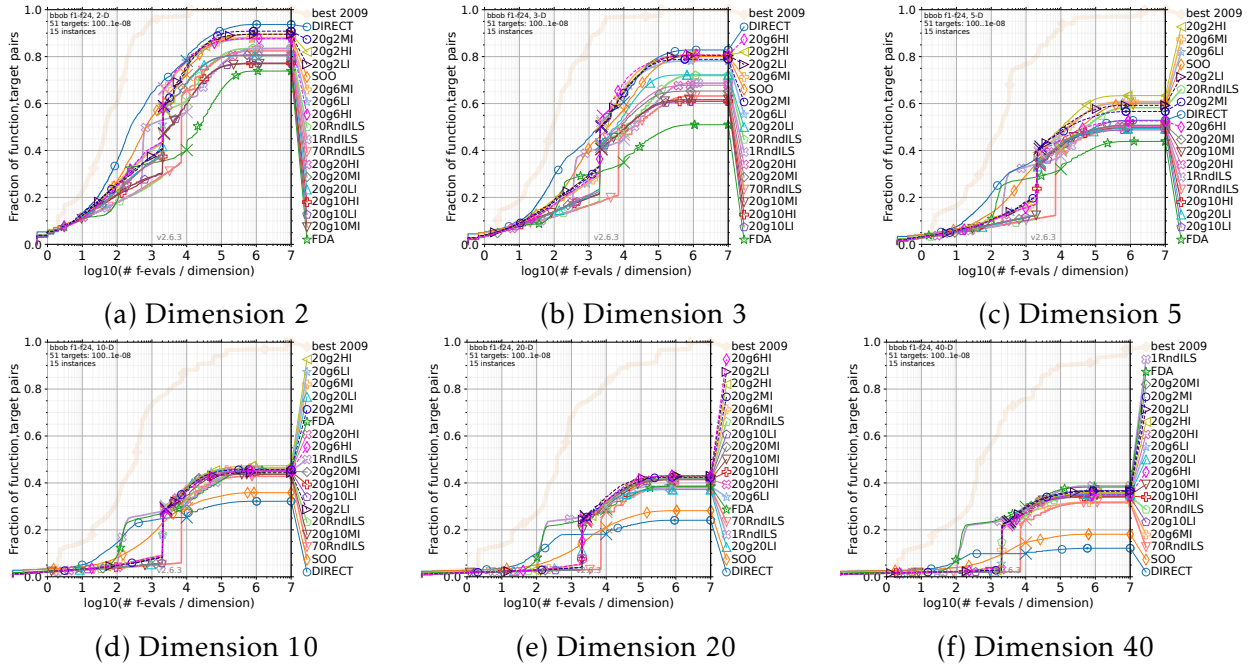


Figure 4.34: Empirical cumulative distribution according to the budget divided by the dimensions for all functions and dimensions.

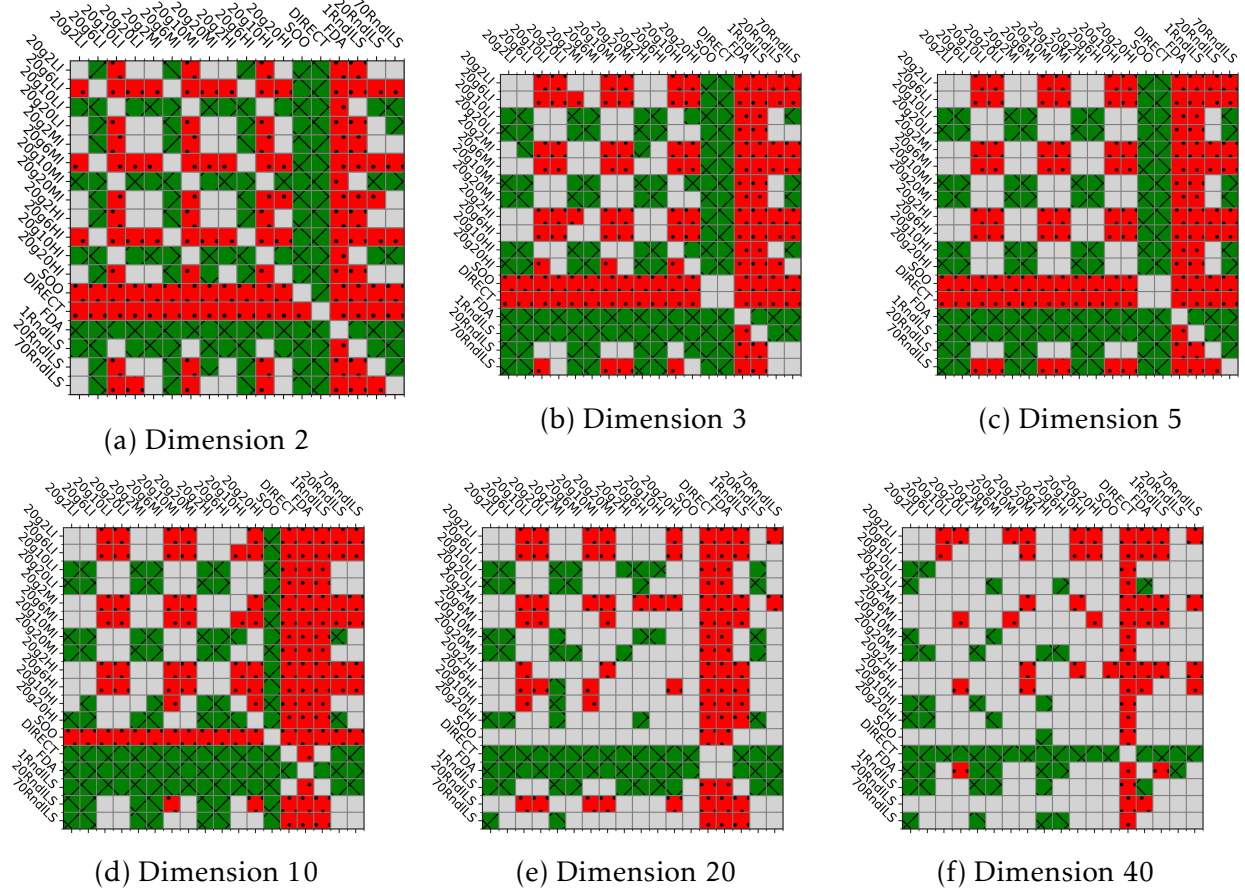


Figure 4.35: Pair Wise Wilcoxon test for all functions and dimensions. **Solid-Grey:** Statistically insignificant ($\alpha > 0.05$). **Gridded-Green:** Better. **Dotted-Red:** Worse.

4.9 Conclusion

In this chapter, we introduced *Zellij*³, a general, unified, and flexible framework for FBD algorithms. A FBD algorithm is made of five search components, i.e. software bricks:

- **Fractal:** Describes the geometry used for a K -hierarchical cover of the search space. The partition operator π generates children of a given fractal and ensures the self-similarity.
- **Tree search:** Allows a certain dynamic in the iterative creation of the K -ary rooted tree modeling the K -hierarchical cover. This dynamic is ensured by a function τ selecting the next fractals to refine, balancing the exploration-exploitation tradeoff, and the width-depth balance of the tree expansion.
- **Scoring:** The result of this component is used by the tree search and allows summarizing knowledge about a fractal into a single value describing its quality, i.e. how promising it is. To do so, the function γ uses the output of the exploration component to return a quality value.
- **Exploration:** The exploration component is an optimization algorithm, a metaheuristic, or a sampling method allowing to quickly retrieve knowledge about a fractal. Thus, the *Explore* function takes a fractal and returns a list of sampled solutions and their objective values.
- **Exploitation:** Once the confidence within a fractal quality is high enough, an intensive local search can be applied to the fractal and its neighborhood. The goal of the *Exploit* function is to refine the knowledge about a fractal to improve the best solution known.

Thanks to the modularity provided by *Zellij*, allowing us to prototype new algorithms quickly, we were able to model various decomposition-based algorithms such as DIRECT, SOO, FRACTOP, FDA and much more. The obtained experimental results illustrate the role of all five search components and their impacts on the performances and the exploration-exploitation tradeoff.

In a long term, the automatic design of FBD algorithms could be tackled by using hyper-heuristic approaches to find the best combination of search components and for specific problems [254]. Such approaches were already applied to population-based algorithms [301].

Concerning, LHS-based algorithms, we have empirically illustrated that we can perform similarly, yet better than FDA across all studied dimensions. However, the algorithm suffers from the curse of dimensionality, as it is less and less able to cover enough space to ensure covering x^* with a certain probability. But conversely to FDA, the sacrifice of the search space is stochastic and not linked to the fractal geometry itself. It is important to mention that in high dimensions, the performances of the algorithms rely heavily on the exploitation component *Exploi*. In low dimensions ($n \leq 10$), the partitioning of the search clearly has some advantages, as it allows detecting confidence regions where to start a local search. However, while the dimension grows $n \geq 20$, the relevance of the partition is greatly diminished as the algorithms perform the same as a random multi-start local search. Thus, our LHS-based algorithms offer an alternative to the n -sphere partition component of FDA for both low and high dimensions. Our stochastic sacrificial LHS-algorithm is not a perfect solution to tackle high dimensional problems, but it opens doors to improvements and other investigations. As we consider the objective function continuous with a bounded searchspace, applying the Heine-Cantor theorem could allow investigating FBD algorithms under the scope of surrogate

³<https://github.com/ThomasFirmin/zellij>

models based on high dimensional piecewise constant approximations of functions, similarly to Riemann integration. Because FBD algorithms are modeled by a K -ary rooted tree, we could also see FBD algorithms from a regression tree perspective. Some early works suggest moving toward FBD-surrogate approaches [44, 177, 283]. Indeed, the approach could be improved by considering a Bayesian strategy [177], as with the BaMSOO algorithm [283]. In LHS-based algorithms, instead of uniformly sampling fractals, one could integrate previous knowledge to model a posterior distribution, $\mathbb{P}(F \in \pi_{\text{LHS}}(\mathcal{A}_F) \mid \mathcal{D})$, with \mathcal{D} an archive of previously sampled fractals or points, associated to their scores or function values. Then, instead of producing i.i.d. and uniformly distributed fractals, π_{LHS} could produce a set of g fractals, optimized according to an acquisition function informed by a posterior distribution.

Furthermore, combining FBD and BO could also open doors to new parallelization techniques of BO algorithms. Indeed, in chapter 3 we described how difficult it is to parallelize BO. Therefore, by partitioning the search space into sub-problems, and because fractals are fully independent sub-spaces, one could parallelize a BO–FBD algorithm by generating diverse batch of fractals or solutions. At the algorithmic levels, we could also think of multiple and collaborative distributed instances of a BO–FBD algorithms on distinct fractals, like the island model for EA algorithms [259, 6, 7].

Silent networks: a vicious trap for Hyperparameter Optimization

Designing an efficient search space Ω is difficult. The first step is to select relevant HPs. The peculiar characteristics of SNNs contribute to a rich choice of HPs, notably concerning the neuron model [89]. However, a counterpart is that SNNs are known to be very sensitive to their numerous HPs [102, 203, 205], making their HPOs challenging.

In chapter 2 we described a few SNN-specific HPs, all having impacts on the 5 described groups. Furthermore, we noticed that within the HPO of SNNs literature, many search spaces had a limited number of HPs. Occasionally, the search spaces are heavily discretized to reduce the complexity at the expense of the flexibility. The HPs of the neuron model are rarely optimized, while they actively contribute to the network dynamic. Then our objective is to optimize a higher dimensional and flexible search space with more HPs, especially those of the neuron model.

In conjunction with chapter 2, we write the ML model as $\mathcal{A}_\theta^\lambda = \mathcal{N}_\theta^\lambda$, a SNN \mathcal{N} parameterized by θ (weights \mathbf{W} and biases \mathbf{b}), and hyperparameterized by λ .

5.1 Preliminary experiments

The following experiments were one of the first done during this thesis. They were run on 16 NVIDIA GTX 1080 TI GPUs for 62 hours on Grid'5000, a large-scale and flexible computation grid [13]. The first architecture to be optimized was the Diehl & Cook SOM, trained by STDP on Poisson encoded MNIST, which was described in chapter 1. The dataset is divided into $\mathcal{D}_{\text{train}}$, $\mathcal{D}_{\text{valid}}$ and $\mathcal{D}_{\text{test}}$ (48000, 12000, 10000 images). The reported accuracies are on $\mathcal{D}_{\text{valid}}$. For these preliminary –failed– experiments, we did not go as far as the test phase. We used the BindsNET [105] simulator because it handles GPU acceleration.

We tried to optimize 15 HPs; most of them are described in chapter 2, and summed up in table D.1. The fixed HPs are given in 5.4. There are two layers, $\{\text{exc}, \text{inh}\}$, a HP can be indexed by the layer. The strength of excitation and inhibition between the two layers are denoted by $f^{(\text{exc})}$ and $f^{(\text{inh})}$. The HP “Norm” describes weight normalization, consisting in making the sum of all pre-synaptic weights of a neuron equal to the HP value. We can already notice that we are optimizing the highest number of HPs from the G1 group (8); within the literature, the maximum is 6 (see chapter 2).

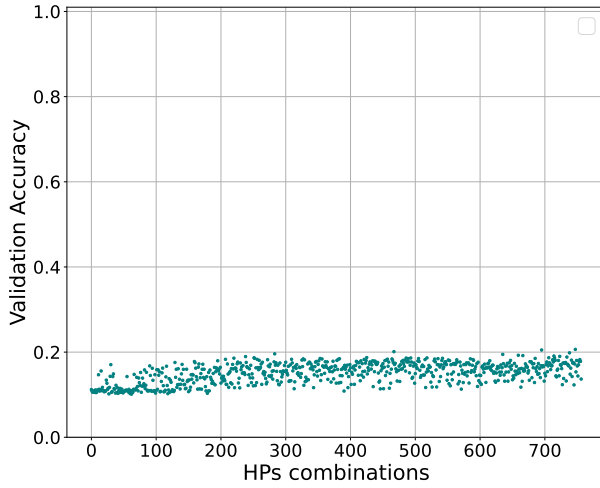
We applied a GA to the problem. The maximum training and validation accuracy are respectively equal to 15.1% and 20.6%. In 62 hours, about 758 solutions were tested, but most of them have an accuracy between 10% and 20%. The evaluated solutions are illustrated

in figure 5.1a and 5.1c. The HPs combinations are sorted by evaluation order; the first combination was the first evaluated, and the last one was the last evaluated. Even if the algorithm seems to hardly converge toward 20%, there is something wrong.

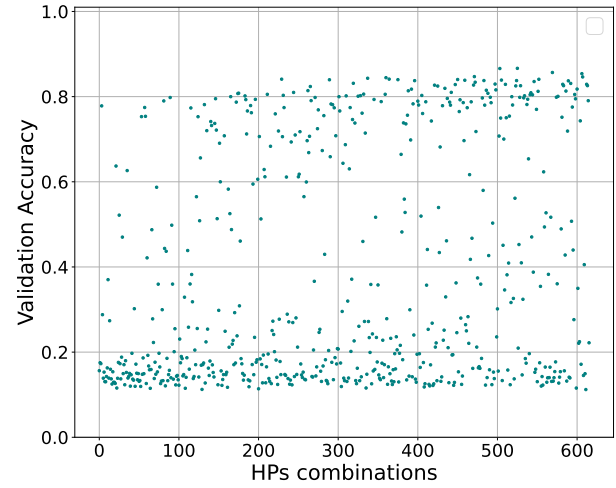
In figure 5.1e, we plot the validation accuracy w.r.t. to the number of spikes produced during the validation phase. It appears that there is a connection between accuracy and spiking activity.

Based on this experience, we modified the GA parameters and operators to sample solutions likely to emit a sufficient number of spikes. Now, the maximum validation accuracy reached 86.6%, and results are presented in figures 5.1b, 5.1d and 5.1f. But, despite 617 evaluations, the algorithm did not converge and the spiking activity is still an issue. The computation time of a single solution ranges from 11.6 minutes to 4.5 hours, with an average of 2 hours per evaluation. For the rest of the manuscript, BO approaches are used for the reasons discussed in chapter 2. Especially because the evaluation of a solution is expensive, and the budget is limited. To observe convergence of the HPO algorithm, we need more resources and longer experiments.

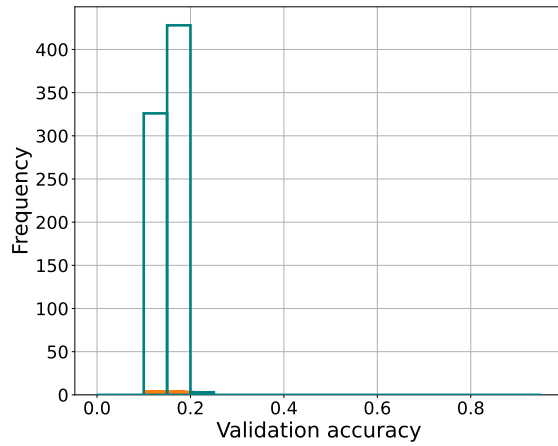
What is the impact of the spiking activity on the HPO?



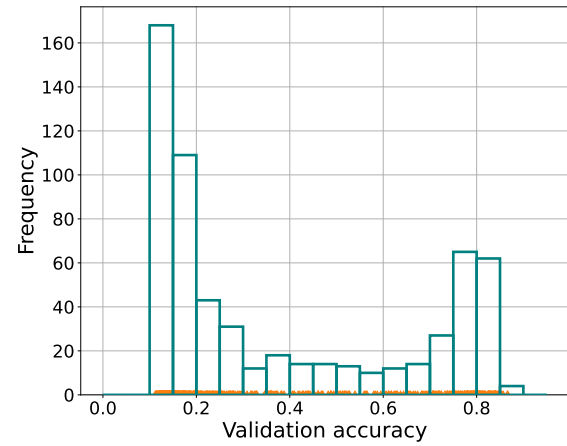
(a) Sampled HPs combinations and accuracy from the first experiment.



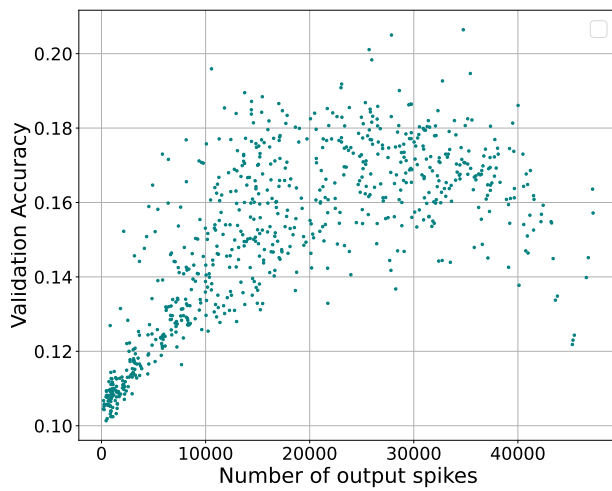
(b) Sampled HPs combinations and accuracy from the second experiment.



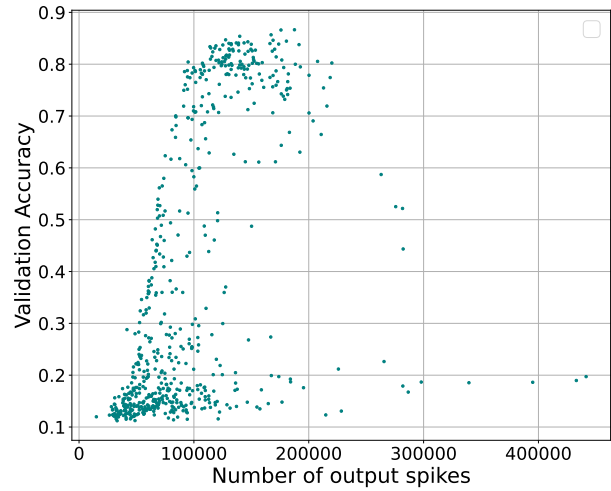
(c) Histogram of accuracies from the first experiment.



(d) Histogram of accuracies from the second experiment.



(e) Accuracy according the spiking activity from the first preliminary experiment.



(f) Accuracy regarding the spiking activity from the second preliminary experiment.

Figure 5.1: Results of preliminary experiments.

5.2 Infeasible solutions, early stopping and black-box constraints

The search spaces contain infeasible solutions that output no or only a few spikes during the training or validation phases; we call such a mode a *silent network*. Finding them is difficult, as many HPs are highly correlated to the architecture and to the dataset. The following sections describe the strategies used to leverage these silent networks within the HPO process.

5.2.1 Silent networks and infeasible solutions

Another challenge of SNNs is the signal loss problem induced by a decreasing firing rate through layers [302, 145]. This phenomenon limits the depth of SNNs since too deep SNNs might result in networks unable to output spikes [35]. In this work, we extend this problem to deep and shallow networks outputting an insufficient spiking activity to perform a specific task. We call these infeasible solutions “silent networks”, as the lack of spiking activity is not only explained by the depth of the network but also by mistuned HPs or architecture.

Explaining silent networks by only considering the depth of a network is misleading, especially when performing HPO. Indeed, one can imagine a binary classification problem in which no output spikes correspond to class 0, and an output spiking activity to class 1. The input data is made of spikes and contains both classes. One can solve this problem using a shallow network of two layers (inputs and outputs) made of LIF neurons, with a resting potential set to 0. Then, the neuron threshold of the output layer can be set to infinity because nothing prohibits this. Considering a finite number of input spikes, neurons, and synapses with finite weights, then we obtain a shallow network unable to spike – a silent network. This network is an *infeasible solution* as output spikes were expected.

We just framed one challenge of HPO, which is designing a viable search space, and finding bounds of the decision variables. In the previous thought experiment, the lower bound of the threshold is straightforward, while the upper one is more challenging. To define it, information about the inputs, topology, spike frequency, or other HPs is needed. Even with this information, a question remains:

Does the threshold itself solely explain the silent network?

We can extend this question to broader architectures, neuron models and HPs.

In a mono- or multi-objectives HPO, a SNN, its training, validation, computations of the errors, and other information, are considered as a fully blackbox. The optimization algorithm has only access to the outputs of this blackbox. This process was described in chapter 2 in figure 2.17. In a classification task, the output of the blackbox is usually the accuracy of the network on the validation dataset. In this case, the decoder (e.g. average spike or max spike) becomes a trap for HPO. Indeed, the decoder can create a link between non-spiking outputs and a class, value, or action. If we look at the previous binary classification problem with a silent network, then the average accuracy will be about 50% (random). Here, the error is to consider a 50% accuracy *silent* network, similar to a 50% accuracy *spiking* network. This is what the HPO algorithm is doing. Because of the blackbox, spiking and non-spiking networks with equal objectives are considered the same.

Conversely to ANNs that always output information, SNNs can output nothing.

This *nothing* is the absence of events. It can be considered as information if a class is attributed to *nothing*, or an absence of information – which is an information – if the SNN is silent.

Moreover, in many works, the minimization of the energy consumption of a network is tackled by considering the spiking activity within a multi-objectives context where the number of spikes or number of neurons has to be minimized [204, 144, 52]. The trap in this context is more vicious. Here, a silent network doing nothing can be considered better than a SNN with low spiking activity.

A simple solution to avoid silent networks during HPO, would be to sufficiently restrain the bounds of the search space to only consider non-silent networks. However, by doing so, one forgets the complex correlations of HPs with accuracy and spiking activity. Indeed, by strongly restraining the search space, one can reject many suitable solutions. This becomes even harder in high dimensional HPO for deep SNNs. This problem is even more crucial and difficult when energy consumption or spiking activity has to be minimized. Indeed, one needs to find the frontier between silent and low-spiking networks.

5.2.2 Early stopping

A way to accelerate HPO process is to early detect silent networks and to prevent useless, expensive computations. Indeed, in the current workflow, no matter if a network is spiking or not, samples are presented until the whole dataset is processed and until all epochs are computed. No matter the outputs, the equations of neurons and learning algorithm are still computed. This phenomenon gets worse when the network is trained with Hebbian based rules:

If neurons do not fire, then synapses do not wire together.

In this work, we define an early stopping criterion based on the percentage of images that did not emit at least α spikes, during the presentation of a sample from the dataset. A network can sometimes be only active for a few data points, since it can be made of an imbalanced number of spikes. In figure 5.2a, one can see that class 1 of MNIST produces fewer spikes than others. Figure 5.2b shows that for DvsGesture, these disparities are more pronounced and are seen among data of the same class. For example, wider or narrower gestures produce more or less spikes. The training pipeline of a SNN under early stopping is described in algorithm 22; it can be easily extended to epochs and batches of larger size. This early stopping allows interrupting the training if a certain percentage β of data does not emit at least α spikes. For example, during the training phase, if at least $\beta = 5\%$ of the data have not emitted at least $\alpha = 1$ spike when presented to the network, the training is interrupted.

We can extend this stopping criterion to other layers, such as the inhibitory layer in a Diehl & Cook architecture [51]. This layer enables neurons from the excitatory layer to specialize into a certain spiking pattern. Thus, to learn, the architecture requires spikes at both the output and at the inhibitory layers.

The α and β HPs can be set according to input data, architecture, or training algorithm. For instance, in SLAYER, the outputs require a certain rate of spikes, and the BP can enforce the network to spike. Whereas in a SOM trained by STDP, if there are no output spikes, then there is neither training nor weights update.

So for SLAYER, β can be greater than for networks trained by STDP. For SNNs trained by SLAYER, α can be set according to the number of expected output spikes, with a certain tolerance since the gradient can enforce spiking activity. It is also important to define β according to the training time. Spending time training a silent network is time that cannot be spent on training promising networks with minimum spiking activity.

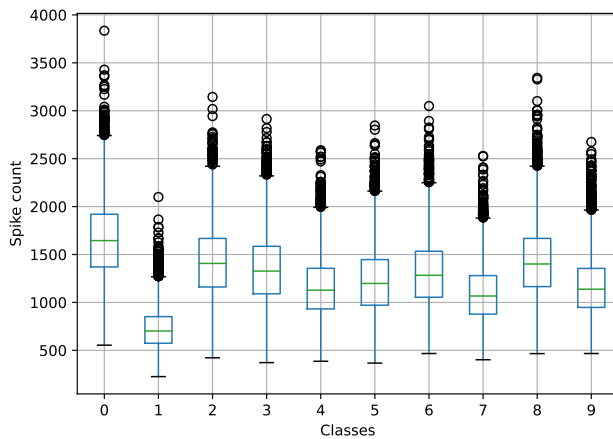
We now have a more practical definition of what silent networks are, depending on the values of α and β . The objective is to prevent a network from being stopped during training because of a lack of spiking activity on certain data.

Algorithm 22 SNN training pipeline with early stopping for 1 epoch and no batch**Inputs:**

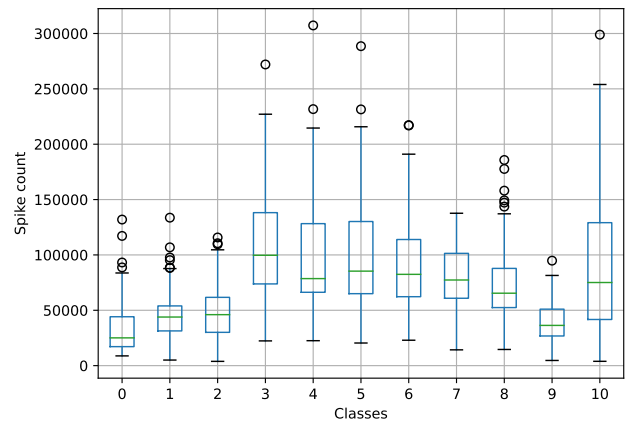
- | | |
|-----------------------|------------------------------------|
| 1: \mathcal{N} | <i>Network</i> |
| 2: X_{train} | <i>Training data</i> |
| 3: Y_{train} | <i>Training labels</i> |
| 4: S | <i>Number of samples</i> |
| 5: α | <i>Minimum spiking activity</i> |
| 6: β | <i>Maximum of non spiking data</i> |
| 7: | |

Outputs: \mathcal{N} *Trained network*

- | | |
|---|-----------------------------------|
| 8: $i \leftarrow 1$ | |
| 9: $\text{count} \leftarrow 0$ | <i>Number of non spiking data</i> |
| 10: $\text{out} \leftarrow \emptyset$ | <i>Output spikes</i> |
| 11: while $(\text{count}/S \leq \beta) \wedge (i \leq S)$ do | |
| 12: $\text{out} \leftarrow \text{Train}(\mathcal{N}, X_{\text{train}}[i], Y_{\text{train}}[i])$ | |
| 13: if $\text{SUM}(\text{out}) < \alpha$ then | |
| 14: $\text{count} \leftarrow \text{count} + 1$ | |
| 15: $i \leftarrow i + 1$ | |
| return \mathcal{N} | |



(a) 100 frames Poisson encoded MNIST



(b) 100 frames preprocessed DvsGesture

Figure 5.2: Class wise spikes count distribution

5.2.3 Black-box constraints

The early stopping criterion allows detecting silent networks, and so prevents useless and costly computations. By spending less time on silent networks, the HPO can evaluate more networks within a given runtime. However, we can improve this by considering silent networks within the HPO process. Indeed, to accelerate the exploration of the search space, one could forecast the spiking activity of a network to avoid sampling within areas of the search space containing silent networks. However, predicting the exact spiking activity of a SNNs is in practice difficult. We can only get the exact spiking activity by passing data to the network and retrieving its outputs.

There are various means to handles constraints in HPO, rejecting, penalizing, repairing, or preserving [259]. Because of the stochasticity and unpredictability of the spiking activity of SNNs, repairing, and preserving strategies are ruled out. Therefore, constraints on the spiking activity are blackbox as we cannot formally model them via an equation. In our specific context, it is not advisable to systematically reject silent networks. Indeed, a network that has been early stopped does not necessarily have low performances. Multi-fidelity shows that we can obtain good performances with a reduced dataset [106]. Thus, the validation accuracy of a silent network that does not follow constraints is still computed, as some samples of the data might output spikes.

We apply a penalization by rewriting the early stopping to create a violation value:

$$\sum_{c \in \mathcal{C}} \max\left(\frac{\text{count}_c}{d} - \beta_c, 0\right), \quad (5.1)$$

where d is the size of the training dataset, and \mathcal{C} is a set of constraints on different layers. If $\exists c \in \mathcal{C} : \frac{\text{count}_c}{d} > \beta_c$ for at least one epoch, the training is interrupted and penalized by at least a positive value, which slightly varies depending on the number of non-spiking data within batches. So, when constraints are met, it describes an acceptable proportion of silent data during training. Thanks to black-box constraints, one of the objectives of the HPO is to avoid sampling silent networks to prevent the training from being stopped because of a lack of spiking activity.

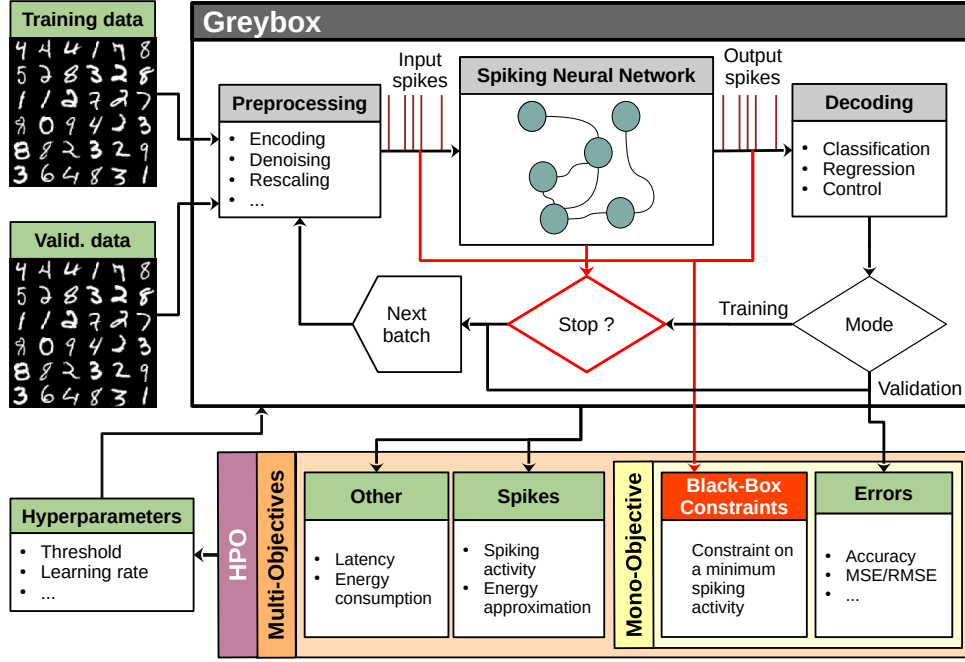


Figure 5.3: Proposed mono- or multi-objectives grey-box HPO of SNNs.

5.3 Experimental setup

5.3.1 High dimensional and constrained Bayesian Optimization

In section 5.2, we described what silent networks are. We designed a spike-based early stopping criterion and associated black-box constraints. Then, the following sections describe how we tackle black-box constrained optimization described in chapter 2.

The preliminary experiments with a GA failed due to the high number of necessary evaluations to make the algorithm converge within a limited budget. Therefore, BO approaches are now preferred. To handle black-box constraints, we have selected SCBO [62], an algorithm based on TuRBO [63]. Both algorithms overcome the scalability issues of BO with high dimensional search spaces. Both TuRBO and SCBO are based on BO [84], which was described in chapter 3. These two new algorithms use trust regions, i.e. a subset of the search space Ω . The scalability is ensured by a zoom-in-zoom-out strategy according to the best current solution. Both algorithms and their parallelization are described in appendix A.

5.3.2 Search spaces definition

All experiments are summed up in table 5.1. The codes of experiments names are read as follows: optimization algorithm - training algorithm - dataset, where the optimization algorithm can be S or T for SCBO or TuRBO (used in section 5.5). The training algorithm is described by STDP for STDP, SLAY for SLAYER/LAVA-DL and SuGr (Surogate Gradient) for BP/SpikingJelly. The datasets are MNIST for Poisson encoded MNIST and DVS for DvsGesture.

We optimized up to 21 HPs, by designing 6 experiments, and so 6 different search spaces, to study behaviors of HPO on SNNs trained with STDP, SLAYER and BP with PLIF neurons, on digital Poisson encoded MNIST and the neuromorphic DvsGesture datasets.

The search spaces and best HPs found by the optimization are summed up in appendix D

Table 5.1: Setup of all 6 experiments

Experiment	Dataset	Data shape	Architecture	Training	Simulator	HPs	c_i
S-STDP-MNIST	MNIST	B.100.1.28.28	SOM	STDP	BindsNET	18	2
S-STDP-DVS	DvsGesture	B.100.2.128.128	SOM	STDP	BindsNET	19	2
S-SLAY-MNIST	MNIST	B.25.1.28.28	CSNN [247]	SLAYER	LAVA-DL	20	1
S-SLAY-DVS	DvsGesture	B.50.2.128.128	CSNN [247]	SLAYER	LAVA-DL	21	1
S-SuGr-MNIST	MNIST	B.25.1.28.28	CSNN [70]	SG	SpikingJelly	12	1
S-SuGr-DVS	DvsGesture	B.50.2.128.128	CSNN [70]	SG	SpikingJelly	13	1

and categorized into five HPs groups, from G1 to G5. In all 6 experiments, we optimized HPs of the neuron model (G1), training algorithm (G2), architecture (G3), decoder or loss (G4), and training pipeline (G5).

HPs of the encoders are fixed. Indeed, preliminary experiments optimizing the encoding method (Poisson Encoded vs. Time-To-First-Spike) and the presentation time (T) ranging from 10 to 300ms, showed a very high dependency of the accuracy on these HPs. This strong correlation greatly affected the impact of more subtle HPs. These encoding HPs could be optimized in a multi-objective context, where number of spikes and accuracy are concurrent objectives. Moreover, T has also a great impact on the training time, memory complexity, and latency of the network. The greater T , the more expensive are the computations and memory usage. That is why T is optimized in chapter 6.

For experiments S-STDP-MNIST and S-STDP-DVS, most of the HPs of the neuron model are optimized for both layers. The list of optimized and fixed HPs, their bounds, and optimized values are given in appendix D.

The architecture of S-STDP-MNIST is the SOM found in [51]. The architecture of S-STDP-DVS, is slightly different and instantiates a distance-based soft lateral inhibition. Neurons that are close together strongly inhibit each other, and conversely [106, 129]. In S-STDP-DVS because DvsGesture is a more challenging dataset, SVM and Log Regression can learn to decode the spikes of the network outputs if selected by the HPO algorithm. These two decoders are often used in the literature, even if it is not considered a fully spiking solution. This experiment also introduces the “Reset interval” HP [129], which prevents too high temporal dependency on previous frames. During the presentation of DVS data, neuron parameters are periodically reset to their initial state. All architectures are summed up in table 5.2.

Concerning experiments S-SLAY-MNIST and S-SLAY-DVS based on SLAYER, both used architecture can be found in [247], and are detailed in table 5.2. Instead of SRM neurons used in the original paper, in this work we have decided to use LIF neurons with adaptive threshold. The architecture used for S-SuGr-MNIST and S-SuGr-DVS are the same as described in [70], with two N_{down} blocks, i.e., the convolution layer, batch normalization layer, and pooling layer. A dropout layer was added after the last convolutional block.

In experiments S-STDP-MNIST and S-SLAY-DVS, and considering both the excitatory and inhibitory layers, the training of the networks is under two stopping criteria and their corresponding constraints. For S-STDP-MNIST, SNNs are expected to output at least $\alpha = 5$ spikes at the excitatory layer and at least $\alpha = 1$ spike at the inhibitory layer for at least 90% ($\beta = 10\%$) of the training dataset. In S-STDP-DVS, because of the higher complexity of DvsGesture, we allow a higher flexibility, $\alpha = 1$ and $\beta = 30\%$ for both stopping criteria.

In experiments S-SLAY-MNIST and S-SuGr-MNIST, the parameters of the stopping criteria

Table 5.2: Architecture details of all 6 experiments.

Experiment	Architecture	Neuron model	Encoding
S-STDP-MNIST	Inputs \rightarrow Outputs \leftrightarrow Inhibitory	Adaptive LIF (chapter 2)	Poisson
S-STDP-DVS	Inputs \rightarrow Outputs \leftrightarrow Inhibitory	Adaptive LIF (chapter 2)	DVS
S-SLAY-MNIST	Inputs \rightarrow Convolution \rightarrow Avg.Pooling \rightarrow Convolution \rightarrow Avg.Pooling \rightarrow Outputs	Adaptive LIF (chapter 2)	Poisson
S-SLAY-DVS	Inputs \rightarrow Avg.Pooling \rightarrow Convolution \rightarrow Avg.Pooling \rightarrow Convolution \rightarrow Avg.Pooling \rightarrow FeedForward \rightarrow Outputs	Adaptive LIF (chapter 2)	DVS
S-SuGr-MNIST	Inputs \rightarrow Convolution \rightarrow Normalization \rightarrow Avg.Pooling \rightarrow Convolution \rightarrow Normalization \rightarrow Avg.Pooling \rightarrow Outputs	PLIF (chapter 2)	Poisson
S-SuGr-DVS	Inputs \rightarrow Avg.Pooling \rightarrow Convolution \rightarrow Normalization \rightarrow Avg.Pooling \rightarrow Convolution \rightarrow Normalization \rightarrow Avg.Pooling \rightarrow FeedForward \rightarrow Outputs	PLIF (chapter 2)	DVS

are set to $\alpha = 3$ and $\beta = 5\%$. In S-SLAY-DVS and S-SuGr-DVS, $\alpha = 1$ and $\beta = 30\%$. We allow a higher flexibility as the number of samples within DvsGesture is much lower than MNIST.

The boundaries of the HPs heavily implied in the spiking activity, such as, V_{th} (neuron's threshold), V_{reset} (neuron's reset potential), τ_{leak} (leakage) or t_{ref} (refractory period) were defined by uniformly sampling random combinations of HPs on a reduced subset of MNIST and DvsGesture, until some networks present a minimal spiking activity. However, we did not try to prevent silent networks, so to define a more general search space. The boundaries of the HPs having an impact on the memory complexity (e.g., number of neurons, number of kernels, kernel size, or batch size) were set according to the available memory on a single GPUs. The number of epochs was set according to computation time of a single solution and the budget of one experiment. To bias the initial combinations of HPs for SCBOs, these are sampled differently according to their additive or multiplicative effect, or to bias the sampling toward known suitable solutions. For instance, the leakage τ_{leak} of a neuron, because of its multiplicative effect, is sampled according to a log-uniform distribution. When values closer to the upper bound need to be sampled with a higher probability, the log-uniform distribution is reversed (R-LogUniform).

5.3.3 Simulators and datasets

To overcome the hardware bottleneck, one can mimic the behaviors of SNNs by using simulators. Depending on the use case, the selection of the most suitable simulator has to be made carefully. Thus, we have selected BindsNET [105] since it handles biologically inspired training algorithms (2-factor STDP, 3-factor STDP), it is based on PyTorch, so it can be accelerated on GPU. BindsNET also implements different neuron models, such as the

IF, LIF or Izhikevich models. One can also find different encoding methods such as Poisson or Rank encoding and decoding methods such as Average Spikes [71], Max Spikes [241], or n -gram [106]. BindsNET is used for experiments S-STDP-MNIST and S-STDP-DVS; the HPs are presented in tables D.2 and D.3. We also used a second simulator, LAVA-DL, as it is also based on PyTorch, and it implements the SLAYER [247] BP-based training algorithm. Experiments S-SLAY-MNIST and S-SLAY-DVS summed-up in tables D.4 and D.5 were simulated with LAVA-DL. Moreover, to study the effect on silent networks of including certain HPs that impact the spiking activity within the gradient backpropagation, we also used SpikingJelly [72]. Then, for experiments S-SuGr-MNIST and S-SuGr-DVS presented in tables D.6 and D.7, the PLIF neurons were used, allowing to train the membrane time constant τ_{leak} .

Performances of SNNs are often assessed on standard ANNs classification datasets, such as the Poisson encoded MNIST [305, 150, 51, 106, 231]. But these benchmarks should be considered a proof-of-concept for SNNs, and spiking analog dataset should be preferred to test SNNs performances [294, 214, 172]. Therefore, we have selected two benchmarks: Poisson encoded MNIST [153] and DvsGesture [10].

In this work, MNIST was encoded within 100 frames, the shape of the data is the following: $B.T.C.H.W$, for batch size, frames, channels, height, and width ($B.100.1.28.28$). The number of frames is 3.5 time lower than in [51]. No other transformation was made, such as denoising or centering. For S-SLAY-MNIST and S-SuGr-MNIST, T was set to 25 [247].

Concerning DvsGesture, spikes were accumulated within 100 or 50 frames, overlapping spikes during this process are considered as a single spike. Data was denoised using a 20000ms temporal neighborhood. Both ON and OFF channels are considered, so the shape is $B.100.2.128.128$. The Tonic [155] Python package was used to process the data. In experiments, S-SuGr-DVS and S-SuGr-DVS, T was set to 50. Due to its higher pixel and temporal resolutions, the DvsGesture dataset presents a more challenging problem by increasing the complexity in terms of topology, memory usage, and computational demands.

Both datasets were divided into training, validation, and testing datasets of respective sizes 48000, 12000 and 10000 for MNIST. DvsGesture is divided into sets of sizes 862, 215 and 264. The optimized accuracy is the one obtained on the validation dataset, and the final results of the best solution found are assessed on the testing datasets.

5.3.4 Hardware and software specifications

Long-run experiments were carried out on the GPU partition of the Jean Zay supercomputer. Each experiment lasted for 100h, and 15 NVIDIA Tesla V100 with 32 GB of RAM were dedicated to the computation of SNNs, one additional GPU was used to compute the GPs of SCBO or TuRBO. A single experiment represents a total of 1600 GPU hours, 100 GPU hours dedicated to SCBO or TuRBO, and 1500 GPU hours to train SNNs. The 16 GPUs are grouped by clusters of 4, containing 2 Intel Cascade Lake 6248 processors of 20 cores each; a single cluster cumulates a total of 160 GB of RAM.

The experiments were parallelized using OpenMPI interfaced by the python library `mpi4py`. BindsNET and SpikingJelly are fully based on PyTorch, while LAVA-DL also compiles custom CUDA code, all three can easily be run onto NVIDIA GPUs.

5.3.5 Algorithmic details

SCBO and TuRBO were implemented and instantiated using `Zellij`¹ and `BoTorch`[12]. Apart from the constraint part, both algorithms share the same parameters. We used a zero-mean function. For the covariance function, different combinations of Matérn kernel ($1/2$, $3/2$

¹<https://github.com/ThomasFirmin/zellij>

and 5/2) were tested to model different landscapes of the covariance function. We kept the Matérn5/2 since it was the most suitable one. Additive models were discarded as they generated numerical instability, particularly for the constraint models.

5.4 Computational results on large-scale experiments

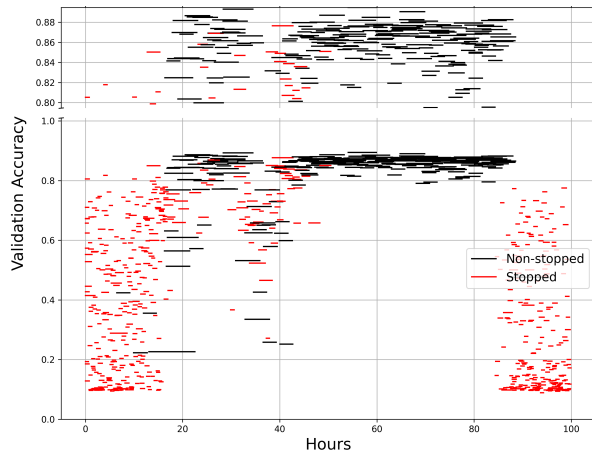
5.4.1 Analysis of the HPO process

In the following lines, an analysis of the impact of silent networks on SCBO is made. In figures 5.4a to 5.4f, a single horizontal line corresponds to the starting and ending dates of the evaluation of a single SNN. The upper part of these graphs, after a break of the y-axis, zooms on the best computed accuracies. This representation emphasizes the ability of SCBO to detect silent networks and focus on fully trained SNNs with high accuracy. The observed drops in accuracy and computation time for S-STDP-MNIST, S-SLAY-MNIST, S-SLAY-DVS and S-SuGr-MNIST, are explained by the reset of the trust region once it shrank to its limits [63, 62]. Then, new random points are sampled, computed, added to the list of existing ones, and SCBO restarts the process. Additionally, one can notice, thanks to the zoom-in, a multiscale convergence of the optimization, particularly in figure 5.4c. It illustrates that improving the best-known HPs combination becomes more and more difficult, and requires more and more resources. For example, in figure 5.4d, reaching 80% validation accuracy can be done in less than 5 hours, while to reach 90% accuracy, the optimization needed about 40 hours, and to improve this solution by 4% (up to 94%) about 40 additional hours were required. This phenomenon is even more emphasized during experiments S-SLAY-MNIST and S-SuGr-MNIST where improvements are about 0.1% accuracy. This illustrates how difficult it can be to reach state-of-the-art accuracy. Moreover, these state-of-the-art accuracy is not always computed using a train-validation-test split, meaning that some could be biased by manual tuning [120].

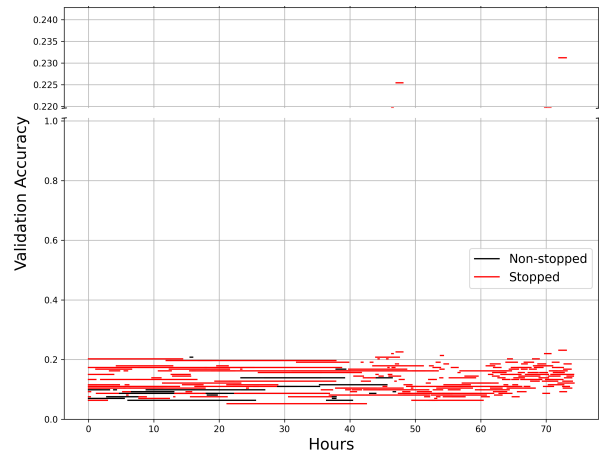
The impact of silent networks on the sampling and budget are presented in table 5.3. The second column represents the total number of sampled solutions, i.e. HPs combinations. The third describes the proportion of silent networks among all sampled solutions. This third column should be compared with the fourth column describing the proportion of the budget, in GPU hours, spent on computing silent networks. During S-STDP-MNIST, one can observe in table 5.3 that while almost 73% of the evaluated networks were stopped, silent networks only consumed about 36% of the 1500 GPU hours. So, for S-STDP-MNIST, the early stopping criterion and constraints prevented significant worthless and expensive computations of silent networks. Indeed, figure 5.4a emphasizes the focus of SCBO on fully trained networks after evaluating numerous silent networks, resulting in high validation accuracies.

Concerning S-STDP-DVS, the HPO failed to focus on feasible solutions. Only 407 networks were computed, and among them about 92% were stopped. Conversely to other experiments, about 86% of the time budget was spent on computing silent networks. These can be explained by the expensive computation time, which can reach up to 50 hours for a shallow network of 2 layers on BindsNET. The SOM architecture might not be suited for such a task, as it involves about 10^7 parameters for only two layers.

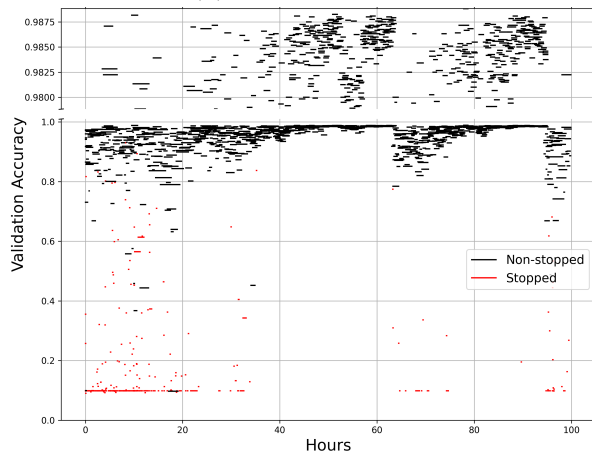
Similar behaviors to S-STDP-MNIST are observed in S-SLAY-MNIST, S-SLAY-DVS, S-SuGr-MNIST and S-SuGr-DVS. These experiments converge much faster toward better solutions than S-STDP-MNIST. Because LAVA-DL and SpikingJelly appear to be less time-consuming than BindsNET, more networks can be computed during the optimization process. In table 5.3, by considering silent networks, their low impact on the budget is even more emphasized compared to S-STDP-MNIST. In S-SLAY-MNIST, 47.1% of the trainings were stopped, but only consumed 1.7% of the total GPU hours. Similarly, for S-SLAY-DVS, about 40.4% of the



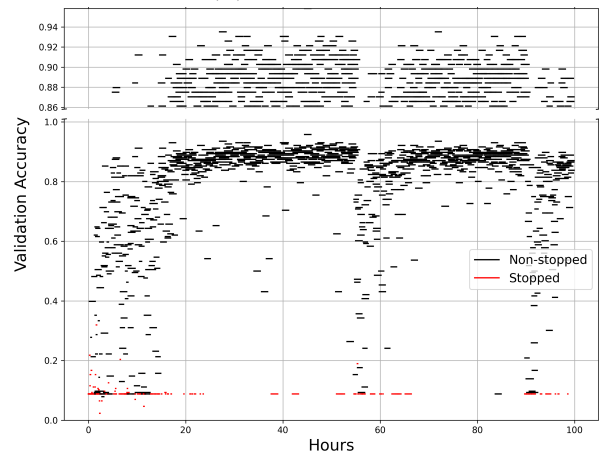
(a) S-STDP-MNIST



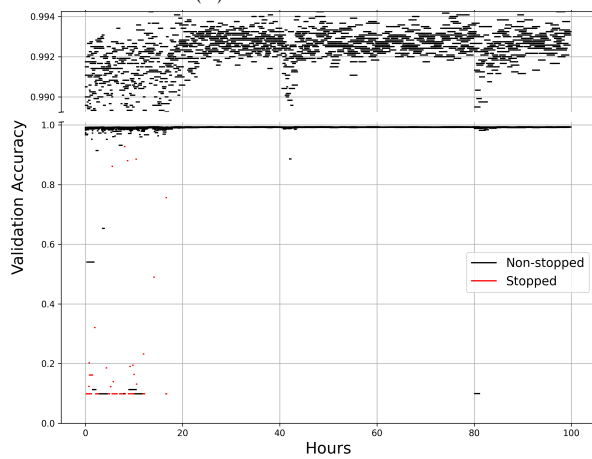
(b) S-STDP-DVS



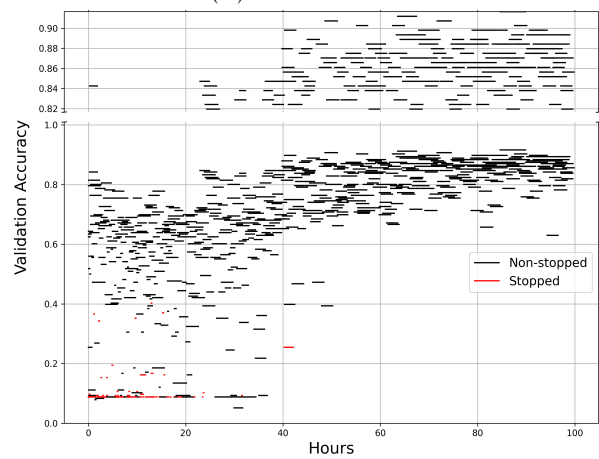
(c) S-SLAY-MNIST



(d) S-SLAY-DVS



(e) S-SuGr-MNIST



(f) S-SuGr-DVS

Figure 5.4: Computation start date to end date compared to accuracy of each SNN.

Table 5.3: Proportion of silent networks during HPO

Experiment	# eval.	% silent	% budget
S-STDP-MNIST	868	72.7	36.3
S-STDP-DVS	407	92.1	85.9
S-SLAY-MNIST	2392	47.1	1.7
S-SLAY-DVS	2063	40.4	3.7
S-SuGr-MNIST	2333	28.1	0.2
S-SuGr-DVS	1044	27.5	2.1

networks were stopped because of a low spiking activity; their computation costed about 3.7% of the total budget.

Regarding S-SuGr-MNIST, obtaining good accuracy seems easier compared to S-STDP-MNIST and S-SLAY-MNIST, resulting in a faster convergence of SCBO. So, we can suppose that integrating additional parameters (τ_{leak}) into the training improves the results. However, once these good first accuracies are obtained, it becomes much harder to improve them. Stabilizing and improving solutions by only 0.01% requires intensive computations, which can question the usage of MNIST to compare SNNs. Such a stagnation observed in figure 5.4e might be explained by limitations of the model or of the search space. Even if τ_{leak} is learned, some networks are still stopped because of a lack of spiking activity; about 28.1% of sampled solutions were stopped. Nonetheless, their impact on the total budget is almost negligible, 0.2%. This might emphasize the performances of SpikingJelly compared to their counterparts. Lower impacts are also observed in S-SuGr-DVS, where 27.5% of the networks were stopped, consuming about 2.1% of the budget. This higher impact on the budget, compared to the one in S-SuGr-MNIST, can be explained by a higher $\beta = 30\%$, so silent networks are trained longer before being stopped.

In S-SuGr-DVS, fewer networks were sampled compared to S-SLAY-DVS, as it appears that SCBO focused the optimization on more expensive solutions. According to table 5.6, the best solution found for S-SuGr-DVS has 10 times more parameters than S-SLAY-DVS. This has an impact on the duration of computing one epoch: 5.56 minutes for the best solution from S-SLAY-DVS, while the one obtained in S-SuGr-DVS lasts for about 10.62 minutes. This can explain that S-SLAY-DVS converged faster than S-SuGr-DVS.

In S-STDP-MNIST, S-SLAY-MNIST, S-SLAY-DVS, S-SuGr-MNIST and S-SuGr-DVS the optimization algorithm was able to find suitable solutions with good accuracy by considering silent networks as a part of the HPO process. Although we have more general search spaces with higher dimensionalities and wider bounds compared to similar works [203, 204, 205, 31, 217, 102]. SCBO was able to scale in dimension, and by modeling black-box constraints, SCBO prevented sampling silent networks. An interesting point emphasized by all 6 experiments and shown in figure 5.4 is that early stopped networks, visible by a red horizontal line, can quickly have acceptable accuracies. This confirms that multi-fidelity HPO, i.e. training on smaller subsets of the training dataset, can be applied to SNNs.

Table 5.4: Best performances obtained using SCBO and BindsNet

S-STDP-MNIST			
Decoder	Train %	Valid %	Test %
MaxSpike	86.0 ± 1.1	86.3 ± 1.7	86.8 ± 1.6
AverageSpike	86.4 ± 0.8	86.6 ± 1.5	87.1 ± 1.3
2-gram	86.8 ± 1.6	88.2 ± 1.7	88.8 ± 1.4
LogReg	89.5 ± 0.5	89.9 ± 1.1	90.5 ± 1.0
SVM	90.1 ± 0.4	90.2 ± 1.1	90.8 ± 1.2
Epochs (Retrained)	8	Parameters	1 895 266
Time/Epoch	138 mins	Trainable parameters	624 848
S-STDP-DVS			
Decoder	Train %	Valid %	Test %
MaxSpike	16.7	23.1	∅
AverageSpike	15.6	23.1	∅
2-gram	14.2	15.6	∅
LogReg	9.4	21.4	∅
SVM	9.7	18.5	∅
Epochs (Retrained)	∅	Parameters	11 508 864
Time/Epoch	Stopped	Trainable parameters	11 272 192

5.4.2 Analysis of the best solutions found

Information about the best solution found by SCBO is presented in table 5.4, 5.5 and 5.6. Optimized HPs combinations are summed up in appendix D. Concerning S-STDP-MNIST and S-STDP-DVS accuracies for all decoders are presented.

The best solutions found for S-STDP-MNIST, S-SLAY-MNIST and S-SLAY-DVS, were re-trained on a larger number of epochs, and then, the final classification on the untouched testing dataset was made to assess solutions' performances and stochasticity. Concerning S-STDP-DVS, that failed, we decided to not intensively retrain the best network since it was stopped during training; it has a high computation time and low performances (23%). This emphasizes the difficulty of obtaining acceptable results in a reasonable amount of time with STDP on DvsGesture. This can be explained by an unsuitable architecture or by the sensitivity of the HPs and the difficulty of defining a viable search space. S-STDP-DVS was run several times with different configurations; by reducing the search space or modifying the preprocessing of input data, similar results were obtained. Comparatively, S-SLAY-DVS and S-SuGr-DVS, achieved better accuracies, with much fewer efforts and less computation time.

In S-STDP-MNIST, considering the optimized decoder (MaxSpike), the best testing accu-

Table 5.5: Best performances obtained using SCBO and LAVA-DL

S-SLAY-MNIST			
Loss	Train %	Valid %	Test %
Rate	98.91 ± 0.12	98.57 ± 0.25	98.80 ± 0.19
Epochs (Retrained)	100	Parameters	161 852
Time/Epoch	5.14 mins	Trainable parameters	161 844
S-SLAY-DVS			
Loss	Train %	Valid %	Test %
Rate	100.00 ± 0.0	89.12 ± 4.86	84.66 ± 4.36
Epochs (Retrained)	50	Parameters	48 046 540
Time/Epoch	5.56mins	Trainable parameters	48 046 528

Table 5.6: Best performances obtained using SCBO and SpikingJelly

S-SuGr-MNIST			
Loss	Train %	Valid %	Test %
Rate	99.96 ± 0.01	99.29 ± 0.07	99.47 ± 0.10
Epochs (Retrained)	100	Parameters	2 900 457
Time/Epoch	3.84 mins	Trainable parameters	2 900 457
S-SuGr-DVS			
Loss	Train %	Valid %	Test %
Rate	99.71 ± 0.29	90.51 ± 3.01	83.52 ± 5.11
Epochs (Retrained)	50	Parameters	125 901 468
Time/Epoch	10.62mins	Trainable parameters	125 901 468

racy is equal to 88.4% which is similar to AverageSpike. Considering other decoders, the maximum testing and validation accuracies are achieved by the SVM with respectively 92% and 90.3%. The 2-gram decoder has a maximum of 90.2% on the testing dataset, which is the best accuracy among low complexity decoders. The values of the optimized HPs $f^{(\text{exc})}$ and $f^{(\text{inh})}$ are high; this indicates a convergence of the HPO toward a WTA mechanism with MaxSpike. Decoding outputs using spike counts and complex machine learning methods gives better results, at the expense of higher complexity and non-neuromorphic solutions. Compared to [51], we were able to achieve similar performances considering the number of neurons, but with 3.5 times less exposition time ($T = 100$), and so lower latency and spikes. In [51], the authors achieved 87.0% with 400 neurons, 91.9% with 1600, and 95.0% with 6400, using a $T = 350\text{ms}$ exposition time. However, our training dataset was smaller due to the additional testing set. Moreover, in our case, a single image is not passed multiple times to the network until reaching a minimum of 5 output spikes [51]. This condition was modeled by a black-box constraint on the outputs and was met for 97.34% of the images during training.

Compared to S-STDP-MNIST, S-SLAY-MNIST achieves a better accuracy of $98.80\% \pm 0.19$ with convolutions, pooling, and about 11 times fewer parameters. The training time per epoch is much cheaper, about 5 minutes for a single epoch, while the other architecture on BindsNET needs 138 minutes for one epoch. In S-SLAY-MNIST, we were able to easily obtain better accuracy, with a sampling duration (T) 4 times lower than S-STDP-MNIST. Compared to SpikingJelly in S-SuGr-MNIST, with almost 18 times more parameters and a shorter computation time of about 4 minutes, the best accuracy obtained on Poisson encoded MNIST in this paper is about 99.47 ± 0.10 . This is close to the baseline architecture [70] with 99.63% accuracy obtained on a testing set. The baseline is trained for 1000 epochs, while our solution is trained for 100 epochs; if it was retrained for 1000 epochs, it would have taken about 64 hours.

Compared to S-STDP-DVS, in S-SLAY-DVS, it is much easier to obtain good accuracies in less time, with more epochs and parameters. We found a testing accuracy of $84.66\% \pm 4.36$. However, we faced overfitting during intensive training of the best solution found; the training accuracy is at 100% while validation and testing are much lower. This can be partly explained by the smaller training set (862 samples) compared to some other datasets. By using PLIF and SpikingJelly in S-SuGr-DVS, equivalent performances were obtained but with 3 times more parameters and a cost of about 10.62 minutes per epoch, while the solution from S-SLAY-DVS has a cost of about 5.56 minutes per epoch.

5.4.3 Analysis of the hyperparameters

We can extract the lengthscales of the GPs to interpret the sensitivity of the HPs w.r.t. the validation accuracy, and constraint on the spiking outputs. However, the GPs are trained on a batch of solutions biased toward the area of convergence of the algorithm, amplified by the trust region. Moreover, HPs are optimized regarding a dataset, a SNN architecture, and search space; hence, extracting general behaviors is delicate. Besides, there is a link between the GP modeling the constraints and the GP modeling accuracy, since enough spiking activity is necessary to obtain good accuracies. Therefore, the analysis of the lengthscales helps to understand some choices, but is far from being absolute. As illustrated in chapter 3, the lengthscales (ℓ) define the “width” of the covariance function. Thus, for a Matérn5/2 kernel, a short lengthscale results in high variability for a small HP value variation, and conversely for a long lengthscale.

In figure 5.10, the first observation is that HPs are more or less sensitive w.r.t. the simulator, the architecture, and dataset. The neuron threshold V_{th} appears to be sensitive

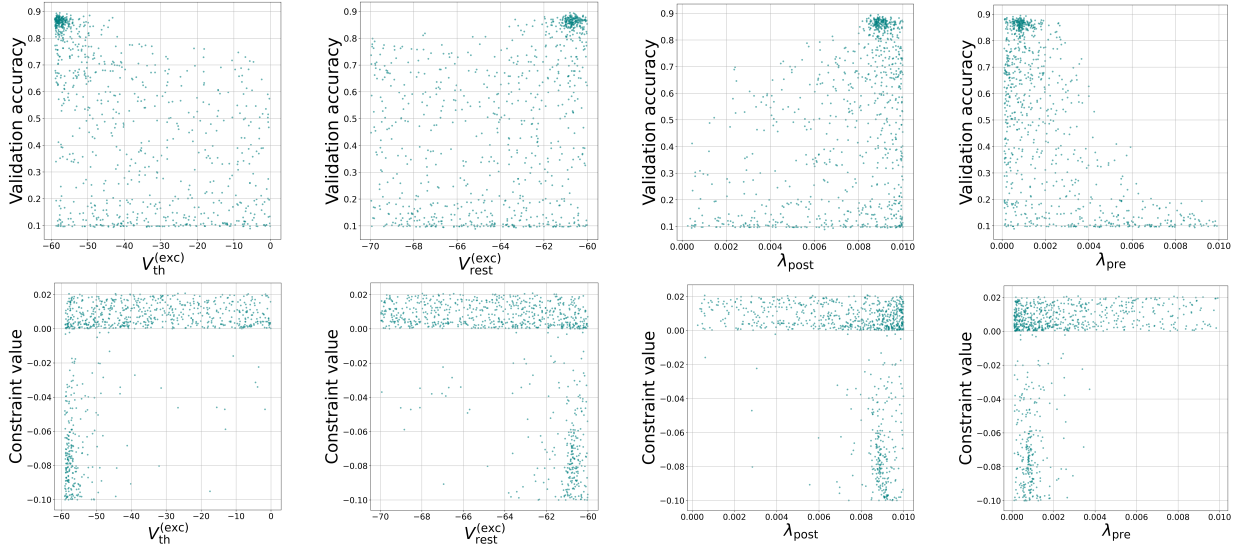


Figure 5.5: Single HP sensitivity in S-STDP-MNIST

across all experiments. Even if for some experiment it is unclear. In S-SLAY-MNIST, V_{th} is sensitive w.r.t. the constraint and so to the spiking activity, while in S-SLAY-DVS, V_{th} is sensitive w.r.t. the accuracy. But, for the reasons previously discussed, these behaviors might be the two faces of the same coin, as enough spiking activity is necessary to obtain good accuracy. Moreover, because of this constraint–accuracy duality, the GPs can learn hazardous rules, such as in S-SLAY-MNIST with the number of epochs being sensitive w.r.t. constraint but not to the accuracy. These anomalous behaviors illustrate that the GPs are not learning causalities, but correlations yet coincidence.

But by combining some lengthscales with single HP–Accuracy and HP–Constraint plots, we can have some insights about HPs sensitivity. For example, in S-STDP-MNIST, the learning rates λ_{pre} and λ_{post} are highly sensitive to both accuracy and constraint. Which is also illustrated in figure 5.5. We can notice that a high λ_{post} and low λ_{pre} are preferred. Meaning that a weight is strongly reinforced if a post-synaptic spike occurs after a pre-synaptic spike, and that a weight is weakly decreased if a pre-synaptic spike occurs after a post-synaptic spike. Lastly for S-STDP-MNIST, we can notice that the optimization favored low $V_{th}^{(exc)}$ and high $V_{rest}^{(exc)}$, the spiking activity is regulated by a refractory period t_{ref} of about 5, a high τ_{leak} (slow leakage), high excitatory and inhibitory strengths ($f^{(exc)}$, $f^{(inh)}$). Thus, the best performing networks are made of a highly spiking excitatory layer with a strong WTA mechanism.

For S-SLAY-MNIST there is less to say; the problem appears to be easier as there are less sensitive HPs. The momentum $\beta_{\nabla 1}$ and $\beta_{\nabla 2}$ do not seem very sensitive (within the boundaries), as well as the learning rate, with efficient values in $[0.001, 0.04]$ and lower ones in $[0.4, 0.1]$. Concerning ν_T , it also has a low sensitivity, and optimal values are in $[0.5, 0.9]$. For the sensitive HPs w.r.t. constraints, it is an all-or-nothing. That is, if there are enough spikes, then the constraints are met for almost all images. The constraint value is almost always minimal (-0.05) if the network is not silent. This means that the gradient appears to be efficient to enforce spiking activity. Whereas, compared to S-STDP-MNIST, minimizing the constraint seems harder. The HPs sensitivity of V_{th} , τ_u is depicted in figure 5.6. We also show τ_{∇} and G_{∇} describing the quality of the gradient surrogate, and that seems sensible w.r.t. the accuracy.

Regarding S-SLAY-DVS and because DvsGesture is a harder task, the sensitivity is even more emphasized. We illustrate it in figure 5.7 with V_{th} , τ_u , τ_{∇} and G_{∇} . The comparison

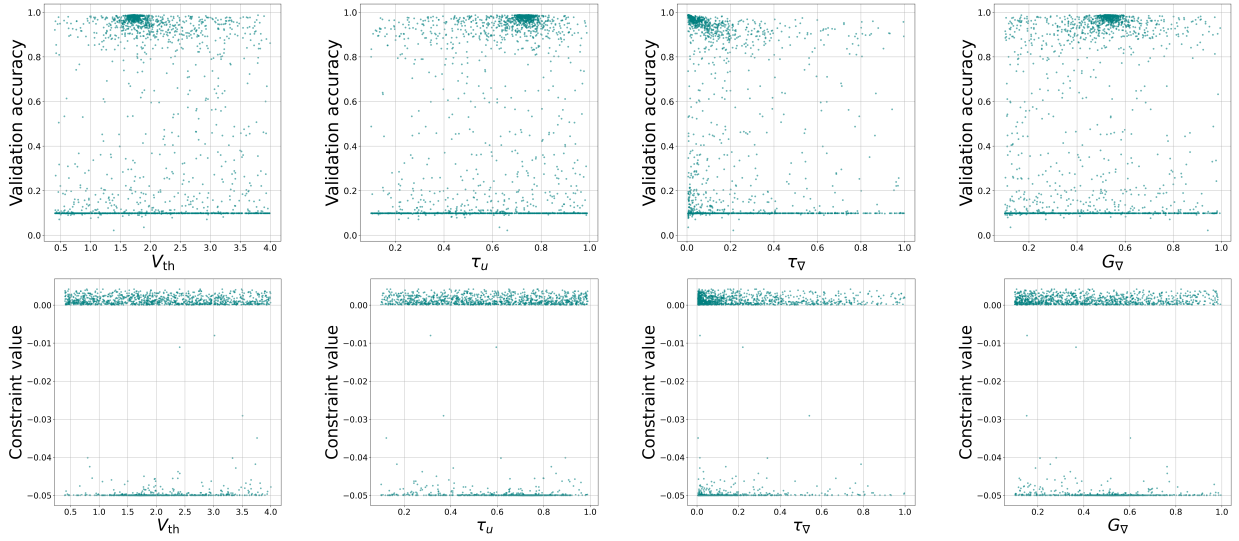


Figure 5.6: Single HP sensitivity in S-SLAY-MNIST

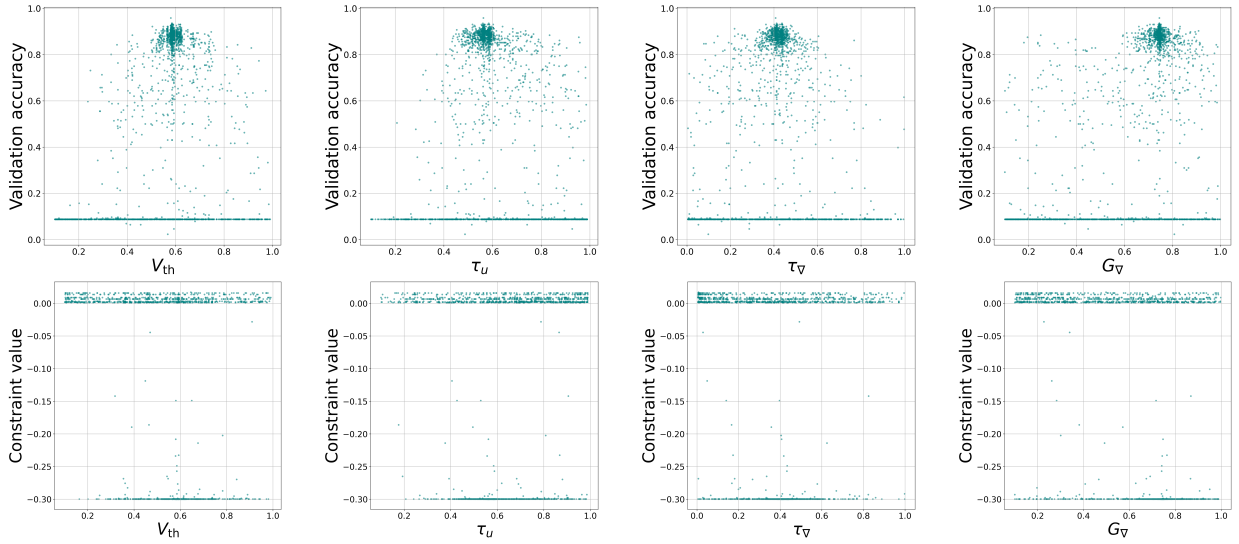


Figure 5.7: Single HP sensitivity in S-SLAY-DVS

between this figure and figure 5.6 illustrates that HP are sensitive regarding the dataset.

Although integrating τ_{leak} as a learnable parameter should reduce the sensitivity toward silent networks, within S-SuGr-MNIST and S-SuGr-DVS we still observe a certain sensitivity of the HPs with the accuracy and spiking activity. We illustrate this by selecting V_{th} , τ_{init} the initial value of τ_{leak} , λ_{∇} the learning rate, and $\beta_{\nabla 2}$ the second momentum of the ADAM optimizer. Once again, we observe a higher sensitivity of the HPs on DvsGesture compared to S-SuGr-MNIST. This is illustrated in figure 5.8 and 5.9.

To conclude, it is clear that it is easier to reach high accuracy with surrogate gradient-based experiments on MNIST, as the HPs appear less sensitive, even if many silent networks are still generated.

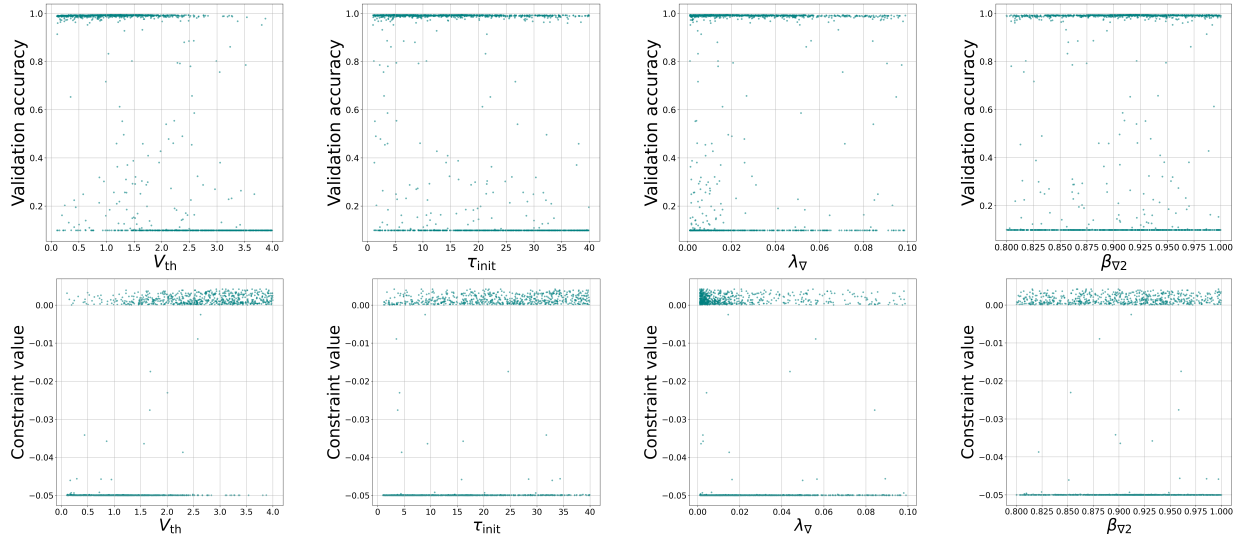


Figure 5.8: Single HP sensitivity in S-SuGr-MNIST

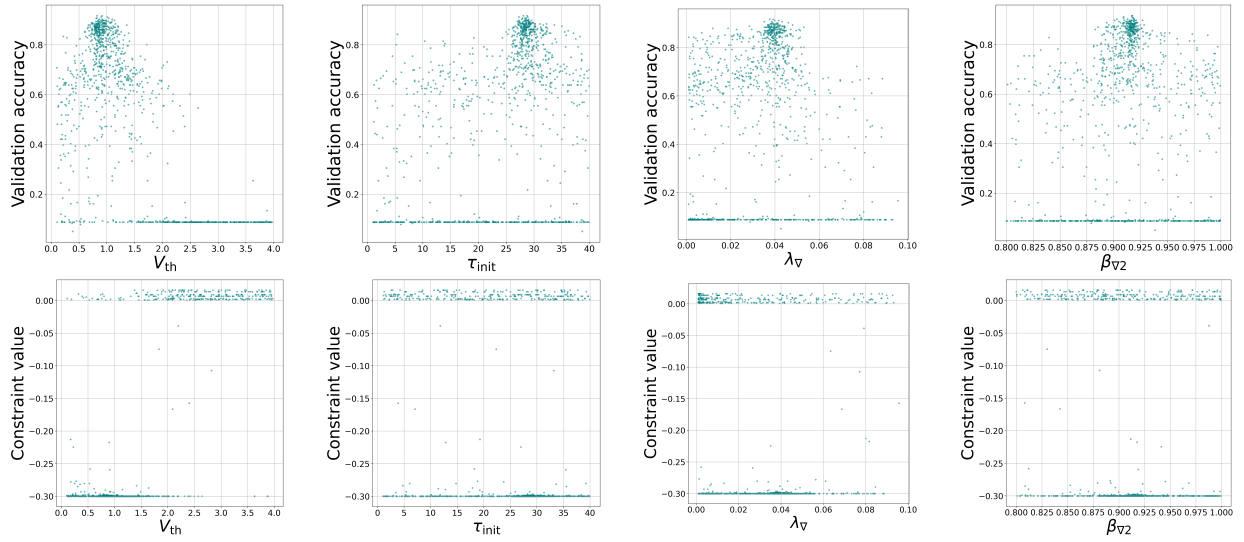


Figure 5.9: Single HP sensitivity in S-SuGr-DVS

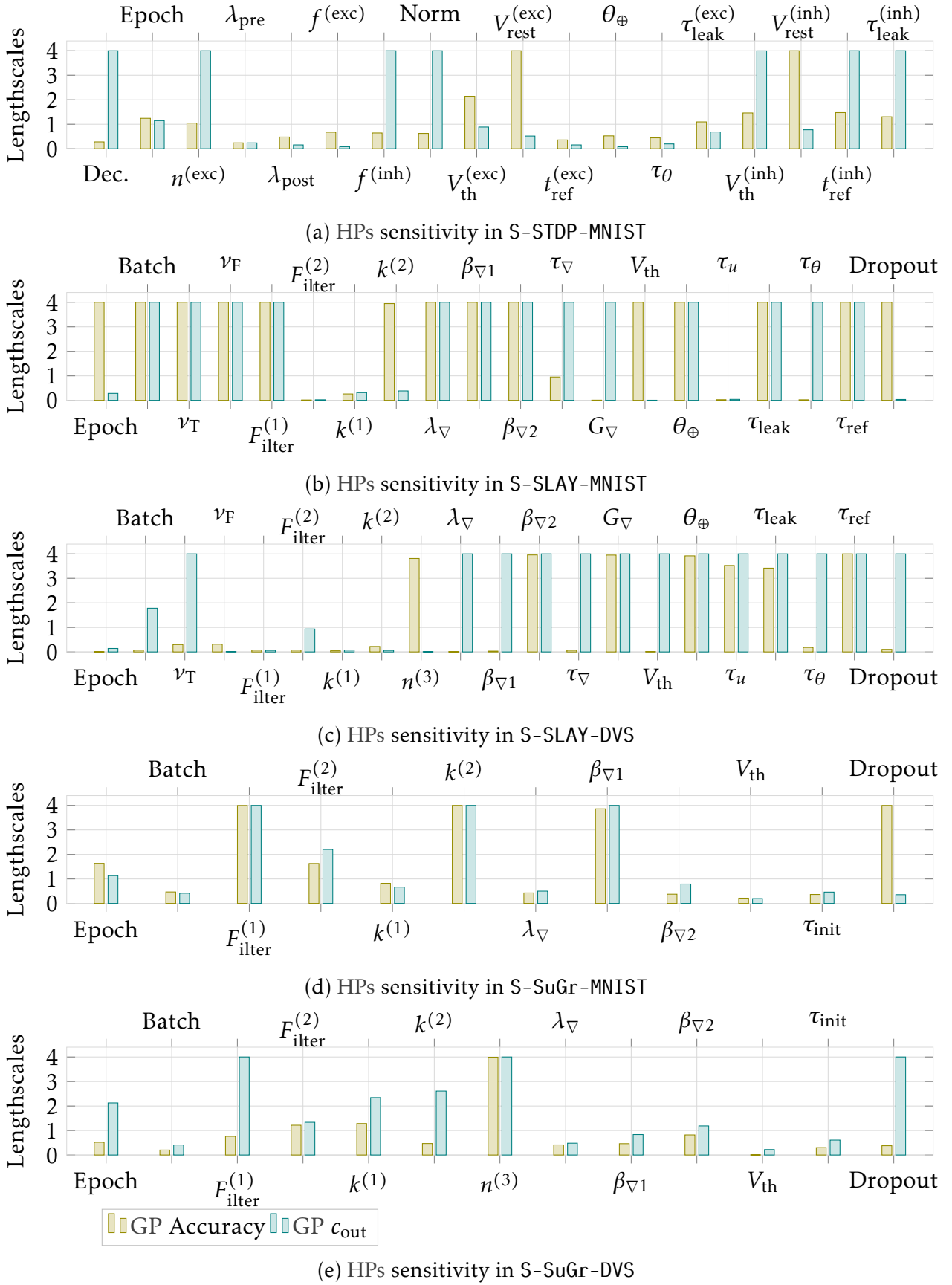


Figure 5.10: Lengthscales from the GPs

5.5 Ablation studies, sensitivity analysis and discussions

In this section, the experimental setup and search spaces are the same as in section 5.3. The following studies demonstrate the contributions of the previously described approach. Only the sensitivity analysis is computed on 16 NVIDIA A100 during 15 hours on a cluster of GPUs within Grid'5000 [13].

5.5.1 Ablation studies

The following experiment serves as an ablation study and a baseline comparison. Indeed, we compare our result obtained by asynchronous SCBO with the synchronous TuRBO from BoTorch [12, 63]. The hyperparameters between experiments in section 5.3 and the following ones are the same. The only difference is that in TuRBO there is no black-box constraint, and so SNNs cannot be stopped. However, we simulate early stopping to detect a network that should have been stopped according to the same α and β described in section 5.3. So in the figures 5.11a, 5.11b, 5.11c and 5.11d, networks marked as *stopped* are not actually stopped; it only indicates that at a given moment of their training, they were considered as *silent*. We compute these supplementary experiments on SLAYER-LAVA-DL and PLIF-SpikingJelly, as the gradient can enforce a spiking activity conversely to STDP. Our goal is to demonstrate that even with gradient-based training, silent networks still appear. With BindsNET on the SOM architecture, if there are no spikes, then there is no training. Another objective of the following experiments, T-SLAY-MNIST, T-SLAY-DVS, T-SuGr-MNIST and T-SuGr-DVS, is to determine if considering silent networks within the optimization process has a positive or negative impact on the performances and on the budget.

To evaluate the impact on the budget, new figures (5.12, 5.13, 5.14, 5.15) show the proportions of the consumed budget by trained SNNs, categorized by validation accuracies, at a given point of the optimization process. Thus, one can see after x GPU hours which proportions of these x GPU hours were allocated to compute SNNs, grouped in increments of 10% validation accuracy.

When comparing convergence between SCBO and TuRBO, figure 5.11 illustrates how silent networks can impact the convergence of the optimization algorithm. It is clear that in T-SLAY-MNIST and figure 5.11a, the algorithm did not have the necessary budget to converge. However, the obtained accuracy is still competitive with S-SLAY-MNIST. Once again, this indicates that MNIST might be too simple a problem for this architecture. Regarding the allocation of the budget for S-SLAY-MNIST, as shown in figure 5.12, at 300 GPU hours, the worst-performing silent networks, identified by a validation accuracy between 0 and 20%, consumed only about 3 to 4% of the total 300 GPU hours. More than 50% and up to 76% were spent on high-performing networks with an accuracy greater than 90%. While during T-SLAY-MNIST about 35 to 43% of the budget was constantly dedicated to computing low-performing networks that could have been stopped.

Interesting behaviors can be seen in figures 5.13 and 5.11b. Both S-SLAY-DVS and T-SLAY-DVS were able to converge toward similar accuracy. Nevertheless, in S-SLAY-DVS, the end of the first exploitation phase happened after about 58 hours, while this exploitation phase for T-SLAY-DVS ended after 80 hours. This can be explained by figure 5.13 and after 300 GPU hours, where T-SLAY-DVS allocated more than 50% of the resources to compute networks having about 10% accuracy, and that could have been stopped. At the same period, only 10% of the budget was dedicated to $> 80\%$ accuracy SNNs. Whereas S-SLAY-DVS at 300 GPU hours, spent between 20 and 25% of the budget on computing silent networks, and between 30 and 40% was used for computing SNNs of about 80 to 90% validation accuracy. In both cases, as the algorithms converge, the impact of the bad decisions at the beginning

Table 5.7: Best performances obtained using TuRBO and LAVA-DL

T-SLAY-MNIST			
Loss	Train %	Valid %	Test %
Rate	98.24 ± 1.07	97.98 ± 0.95	98.38 ± 0.73
Epochs (Retrained)	100	Parameters	105 034
Time/Epoch	2.02 mins	Trainable parameters	105 026
T-SLAY-DVS			
Loss	Train %	Valid %	Test %
Rate	100.00 ± 0.0	92.59 ± 3.24	87.69 ± 3.98
Epochs (Retrained)	50	Parameters	585 767 541
Time/Epoch	5.60mins	Trainable parameters	585 767 529

of S-SLAY-DVS and T-SLAY-DVS decreases. In the end, almost 15% of the budget was lost on worthless computations for T-SLAY-DVS and only about 5% for S-SLAY-DVS. So in this case, considering silent networks and implementing an early stopping, allows to better balance the resources. Indeed, in both S-SLAY-DVS and T-SLAY-DVS, networks considered non-silent (non-stopped) are almost always guaranteed to have high accuracies $\geq 80\%$. For non-silent networks, the impact on the budget of bad solutions ($< 20\%$ accuracy) is negligible.

Regarding PLIF-based SNNs using SpikingJelly with T-SuGr-MNIST and T-SuGr-DVS, silent networks are still observed, but with a lower impact on the budget. Both experiments converged toward a network that should have been stopped, i.e. considered silent. This shows limitations and potential improvements of the proposed stopping criterion and constraints. Still, their convergences seem slower than the ones observed in S-SuGr-MNIST and S-SuGr-DVS. In figures 5.14 and 5.15, it is clear that among networks that should have been stopped, some of them have an impact on the budget while having low accuracies $\leq 20\%$. For T-SuGr-MNIST and at 300 GPU hours, between 10 and 15% of the budget was spent on worthless computations, regarding T-SuGr-DVS it is about 35%. Once again, because of the convergence, this impact tends to decrease through time.

So, to increase the efficacy of the optimization algorithm during HPO of SNNs, considering silent networks helps in increasing the convergence speed. Indeed, by better allocating resources to SNNs having a sufficient spiking activity, it increases the chances of obtaining high accuracy.

However, one of the major drawbacks of our approach, illustrated in figures 5.11, 5.12, 5.13, 5.14 and 5.15, is that the early stopping can discard a non-negligible amount of good solutions. These are considered silent at a certain step of the training process. While waiting for, a few additional training steps can make these misclassified silent networks non-silent. Therefore, future works could improve the approach by considering the evolution of the spiking activity during training or by being more patient and careful before triggering the early stopping. Even if in S-SLAY-MNIST, S-SLAY-DVS, S-SuGr-MNIST and S-SuGr-DVS, SNNs considered as silent could have resulted in high accuracies, no significant loss of performances in terms of validation accuracies was seen compared to experiments using TuRBO.

The previous experiments and the ones from sections 5.4 are summed up in table 5.9. The

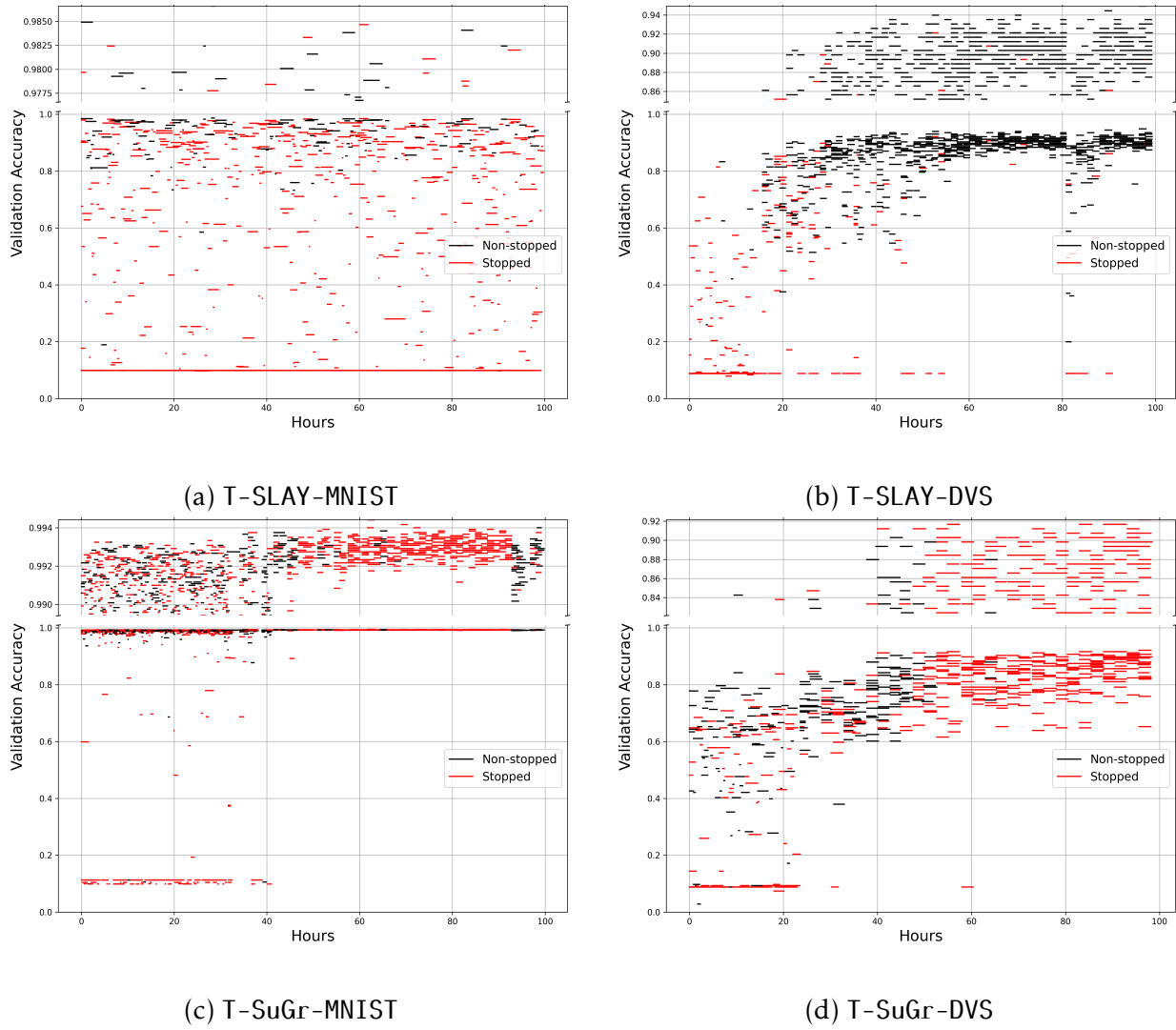


Figure 5.11: Computation start date to end date compared to accuracy of each SNN optimized by TuRBO.

Table 5.8: Best performances obtained using TuRBO and SpikingJelly

T-SuGr-MNIST			
Loss	Train %	Valid %	Test %
Rate	99.97 ± 0.01	99.35 ± 0.10	99.47 ± 0.10
Epochs (Retrained)	100	Parameters	2 939 421
Time/Epoch	2.31 mins	Trainable parameters	2 939 421
T-SuGr-DVS			
Loss	Train %	Valid %	Test %
Rate	99.88 ± 0.12	93.75 ± 2.22	87.12 ± 2.65
Epochs (Retrained)	50	Parameters	146 273 535
Time/Epoch	10.57mins	Trainable parameters	146 273 535

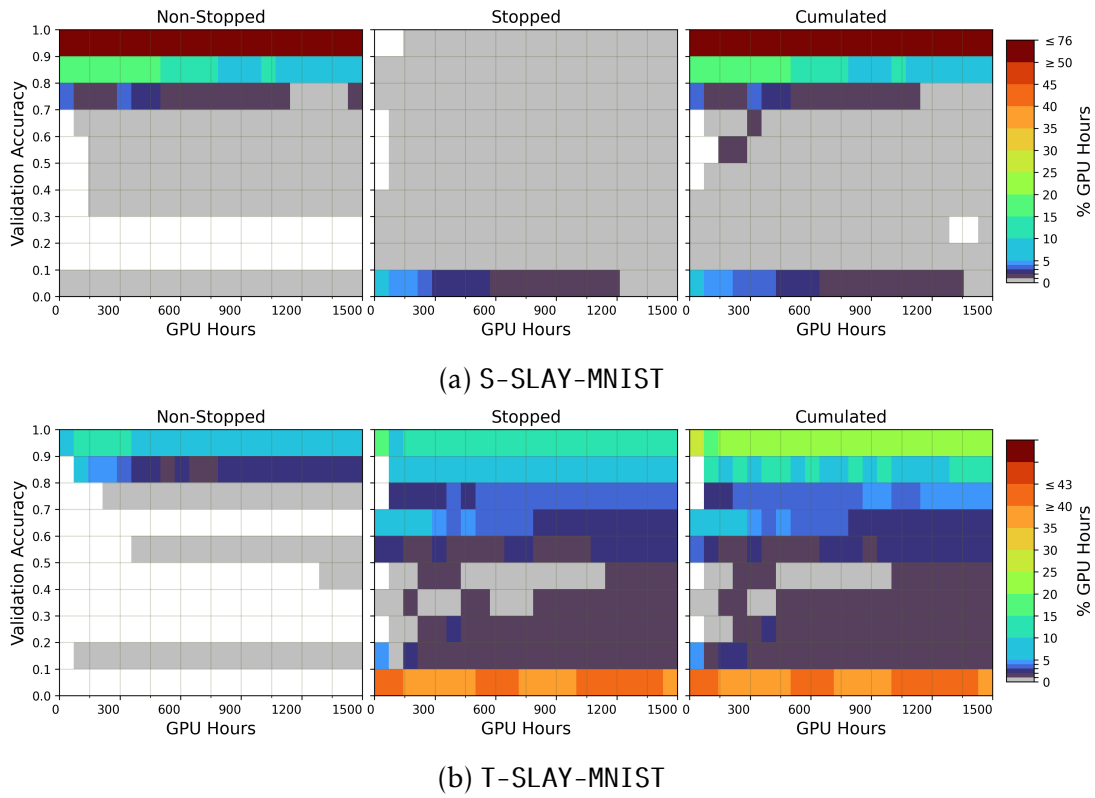


Figure 5.12: Allocation of the budget during the optimization on MNIST using LAVA-DL

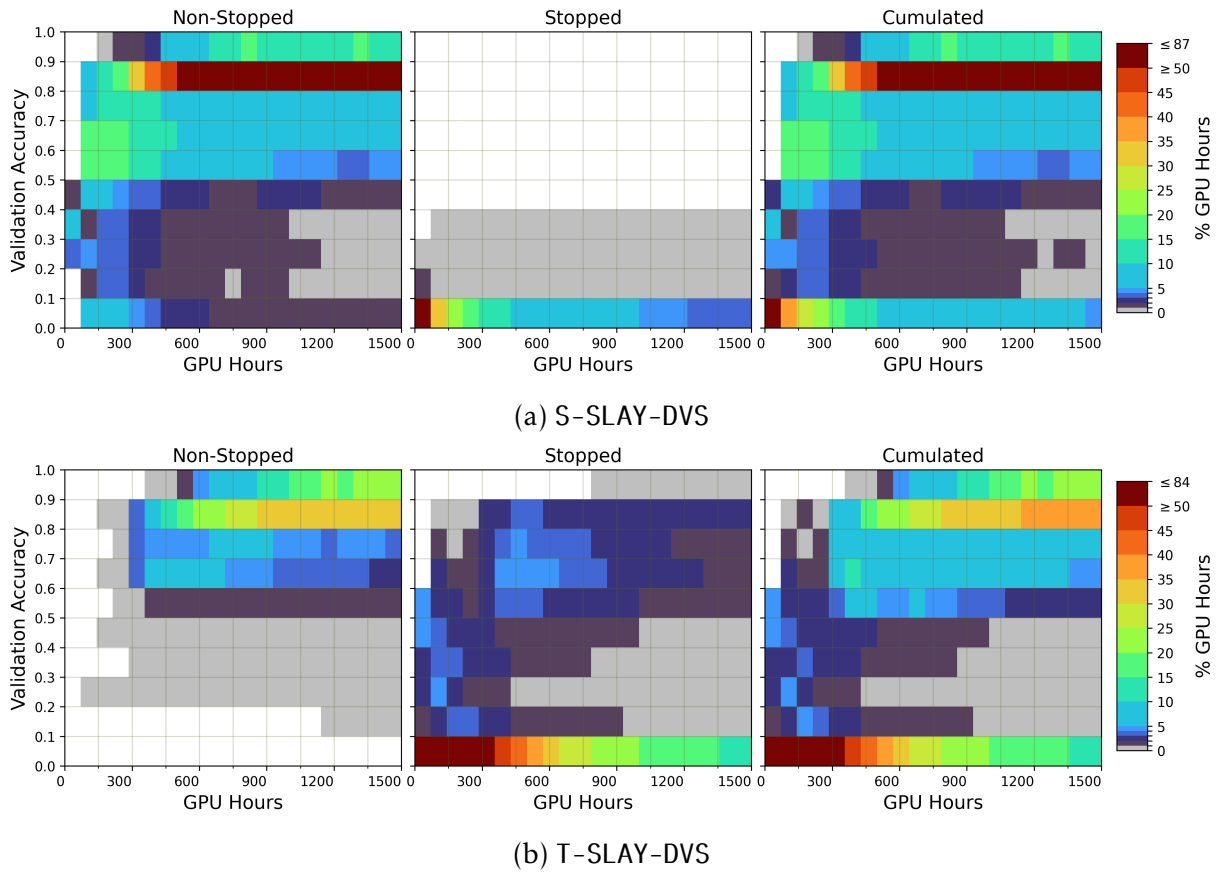


Figure 5.13: Allocation of the budget during the optimization on DvsGesture using LAVA-DL

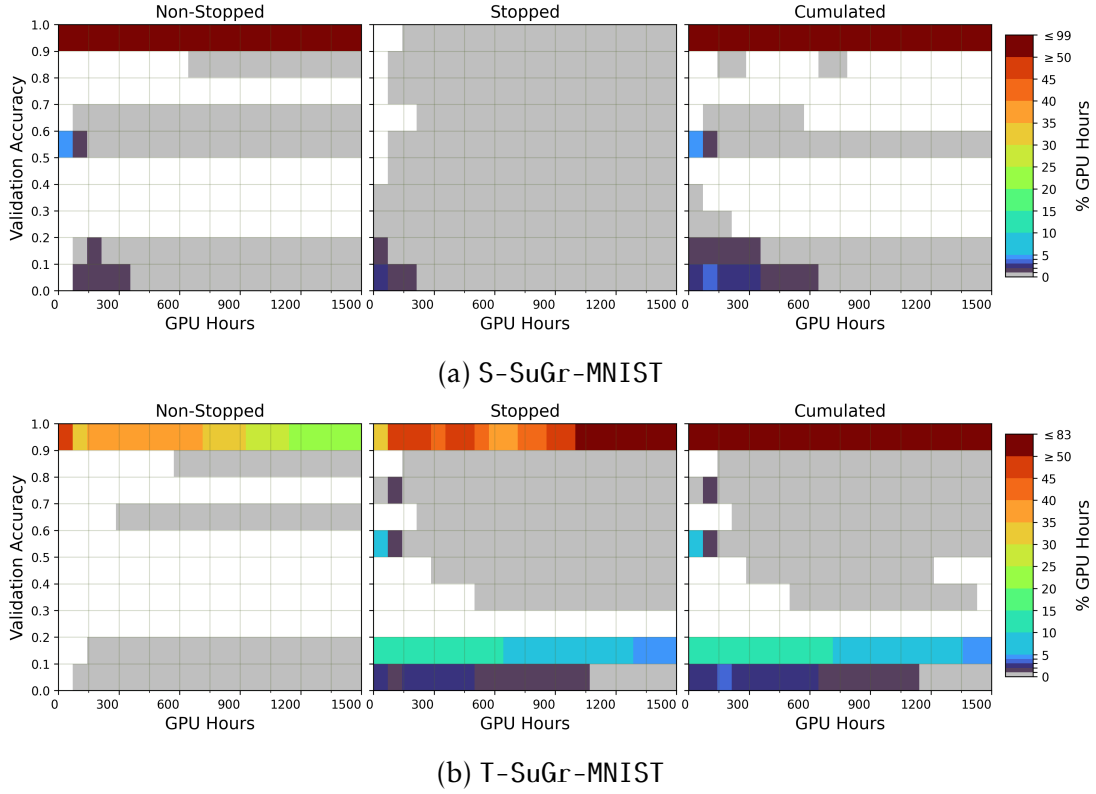


Figure 5.14: Allocation of the budget during the optimization on MNIST using SpikingJelly

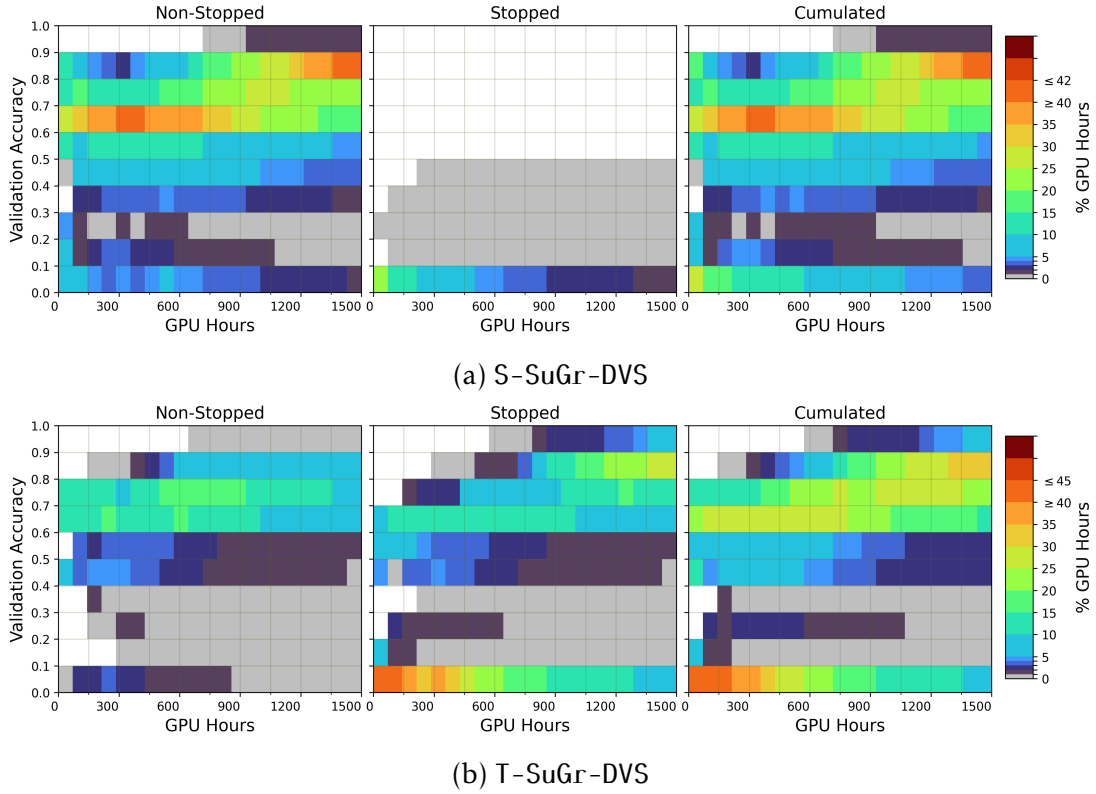


Figure 5.15: Allocation of the budget during the optimization on DvsGesture using SpikingJelly

Experiment	Archi.	Simulator	HPs	c_i	Acc.	#	% silent	Hrs	%budget
With constraints and early stopping									
S-STDP-MNIST	SOM	BindsNET	18	2	90.8 ± 1.2	868	72.7	100	36.3
S-STDP-DVS	SOM	BindsNET	19	2	FAIL (~ 23.1)	407	92.1	100	85.0
S-SLAY-MNIST	CSNN	LAVA-DL	20	1	98.80 ± 0.19	2392	47.1	100	1.7
S-SLAY-DVS	CSNN	LAVA-DL	21	1	84.66 ± 4.36	2063	40.4	100	3.7
S-SuGr-MNIST	CSNN	SpikingJelly	12	1	99.47 ± 0.10	2333	28.1	100	0.2
S-SuGr-DVS	CSNN	SpikingJelly	13	1	83.52 ± 5.11	1044	27.5	100	2.1
Without constraints and early stopping									
T-SLAY-MNIST	CSNN	LAVA-DL	20	0	98.38 ± 0.73	951	85.5	100	88.0
T-SLAY-DVS	CSNN	LAVA-DL	21	0	87.69 ± 3.98	1305	30.5	100	27.0
T-SuGr-MNIST	CSNN	SpikingJelly	12	0	99.47 ± 0.10	1693	68.7	100	73.3
T-SuGr-DVS	CSNN	SpikingJelly	13	0	87.12 ± 2.65	695	69.6	100	72.6

Table 5.9: Summary of all long run experiments.

column c_i gives the number of constraints, *Acc.* is the accuracy on the **hold out** test dataset $\mathcal{D}_{\text{test}}$, *HPs* is the number of optimized HPs. The column *Hrs* is the budget of the experiment in hours, and *%budget* is the proportion of the budget spent on computing silent networks. The column # is the number of evaluated HPs combinations, and *% silent* is the proportion of silent networks.

5.6 Conclusion

HPO of SNNs is challenging. This problem is often solved by thinking of the SNN as a fully blackbox. The spiking activity of SNNs is usually considered within a multi-objective context, where it is minimized so to obtain better energy efficiency.

However, ignoring that a SNN needs a minimal spiking activity, is ignoring the silent network problem explained by mistuned HPs or architecture. Therefore, during the HPO, SNNs should be instead considered as a graybox, where information about its behaviors is used to better sample solutions.

Regularly, search spaces are low dimensional and strongly bound to only good solutions. In this work, we have shown that we can define general and scalable high dimensional search spaces containing many silent networks while maintaining performances of the HPO process for expensive SNNs. By leveraging infeasible solutions, we can increase the efficiency of the exploration of search spaces. This is done by a combination of a spike-based early stopping criterion and its associated black-box constraints. The early stopping criteria prevent useless computation by interrupting the training when a certain proportion of the data did not output enough spikes. The SCBO algorithm ensures scalability in high dimension and models the constraints to prevent sampling silent networks. Our approach was generalized to the two most popular families of training algorithms known as plasticity rules and surrogate gradient BP. For both cases, experimental results emphasize the value of our strategy, which maintains good performances within a generalized, high dimensional search space. Moreover, because of the early stopping criterion, computation time of a single SNN is stochastic. This stochasticity is handled by the asynchronous parallelization of the optimization algorithm on a heterogeneous multi-node and multi-GPU Petascale architecture.

The next step is to even more accelerate HPO. Therefore, in chapter 6 we leverage silent networks and multi-fidelity optimization.

Accelerating Hyperparameter Optimization with Multi-Fidelity

In chapter 5, we emphasized the negative impact of silent networks on HPO. By detecting them via a heuristic, we can better balance the workload focusing on networks with a minimal spiking activity, preventing the costly and worthless computation of some low-accuracy networks. To do so, we applied an early stopping criterion and integrated specific blackbox constraints within the optimization algorithm.

We described *silent networks*, which are SNNs unable to output enough spikes for a given task because of mistuned HPs or architecture. This concept is a generalization of the signal loss problem [302, 145] explained by a too-deep SNN. Because spiking datasets have heterogeneous spiking activity for and within each class, early stopping is based on per-sample spiking activity. If one can detect that a proportion β_{train} of samples outputting less than α spikes, is greater than a given proportion β ($\beta_{\text{train}} \geq \beta$), then the training is stopped and the SNN is considered a silent network. We combine the early stopping with blackbox constraints, indirectly measuring the spiking activity: $c_{\text{out}} \leq 0 \iff \beta_{\text{train}} - \beta \leq 0$. A negative or positive value of c_{out} is an indication that the training phase has been stopped or fully completed. Therefore, the constraints force the HPO algorithm to find HPs combinations carrying out the training process.

By doing so, we can accelerate the convergence of the algorithm by sometimes a few tens of hours. But it still requires considerable efforts to obtain convergence and good performances.

The environmental impact of previous long-run experiments on Jean Zay, is estimated to 0.435 tons of CO₂ emissions for the 1500 GPU hours. The electricity consumption of each long-run experiment costed about, in May 2024, 576 EUR, 625 USD or 4521 CNY. The total emissions, including experiments in section 5.3, experiment design, test, and debugging, are about 1.126 tons of CO₂, for a cost of about 14893 EUR, 16160 USD or 116872 CNY.

To reduce the budget of HPO and so its cost, we can combine the early stopping and blackbox constraints with multi-fidelity optimization. Furthermore, previous results emphasize that some very early stopped SNNs can have acceptable performances, leading the way to even more accelerated HPO by considering multi-fidelity, i.e., training on smaller subsets of $\mathcal{D}_{\text{train}}$. In this chapter, we apply a more general approach to multi-fidelity, known as *cost-aware* BO. The following sections explain why and how we can scale the optimization up to 46 HPs while reducing the budget drastically and maintaining performances. This is the highest number of HPs optimized concerning HPO of SNNs in the literature.

Moreover in this chapter, we discard the BindsNET simulator due to its notably lower computational performances compared to LAVA-DL and SpikingJelly. Calibrating and

designing networks on BindsNET is also harder than with LAVA-DL and SpikingJelly. Lastly, the performances on DvsGesture are deceptive compared to the laborious work put in designing various experiments for this dataset using BindsNET. Surrogate-based BP offers undeniably better performances with much fewer efforts. Because we discard the SOM architecture, we now focus on a single constraint c_{out} on the output of the SNN.

6.1 Multi-fidelity and spiking neural networks

In this section, we quickly review some works presented in chapter 2, but under a multi-fidelity point-of-view.

Hyperparameters definitely have an impact on the computation time of SNNs. In [205], the *cost* was considered within a multi-objective approach, so to maximize the accuracy while minimizing the training time per epoch. The authors have demonstrated that high accuracy can be obtained in less training time. As a result, intensive and expensive training is not necessary to obtain competitive accuracy.

Another work [274] used a multi-fidelity BO algorithm known as BOHB, based on the Hyperband algorithm [69]; both algorithms are described in chapter 2. The authors optimized a 3 HPs search space (leakage, time-steps, learning rate) to classify CIFAR-10 and CIFAR-100 with the S-Resnet38 architecture. However, the impact of the multi-fidelity on the optimization process (the training set size π_{train}) was not investigated. BOHB maximizes the fidelity HP, i.e. π_{train} the size of the training subset from $\mathcal{D}_{\text{train}}$. BOHB assumes that higher fidelity always results in better performances [20]. However, as described in [20], multi-fidelity HPO can be divided into two groups, one for which the previous assumption holds, and another one where higher fidelity does not necessarily result in better performances, notably in the case of overfitting. We illustrate multi-fidelity on $\mathcal{D}_{\text{train}}$ in figure 6.1.

In our SNNs problem, specifically when they are simulated, several HPs can control a certain fidelity, i.e. the complexity of the simulator [105, 252]. In BindsNET [105], the temporal granularity, named dt , in milliseconds, controls the number of time steps during the simulation. In Brian2[252], one can choose between different numerical integration methods such as Euler or Runge-Kutta algorithms, which also affects the performances [102]. Usually, and for non-event-driven simulators, spikes of a sample are accumulated within a certain number of frames T . This HP also influences the performances of the SNN. The higher the number of frames, the higher the computation time, but not necessarily the better the performances [150, 159].

6.1.1 Improved early stopping and constraints

The early stopping criterion described in chapter 5, is based on two HPs, α describing the minimum number of output spikes for a single sample from $\mathcal{D}_{\text{train}}$, and β the maximum acceptable proportion of samples outputting less than α spikes. During the training, after each batch, the proportion β_{train} of non-spiking samples is computed. If $\beta_{\text{train}} \geq \beta$, the training is stopped. We can extend this to $\pi_{\text{train}} \in (0, 1]$, a HP defining the proportion of the initial $\mathcal{D}_{\text{train}}$ used for training. To improve the detection of silent networks, β now describes the proportion of samples from the subset $\mathcal{D}'_{\text{train}} \subseteq \mathcal{D}_{\text{train}}$, outputting less than α spikes. So, as described in algorithm 23, which can be easily extended to batches of data, the early stopping is now based on the subset of $\mathcal{D}_{\text{train}}$ proportional to π_{train} . Thus, selecting a convenient β relies on the minimum value of π_{train} and on the size of $\mathcal{D}_{\text{train}}$. Moreover, instead of computing a constraint on the proportion of samples outputting **less than** α spikes, $c_{\text{out}} \triangleq \beta_{\text{train}} - \beta$, we rewrite it as the proportion β'_{train} of samples that have outputted **at least** α spikes before the network being stopped:

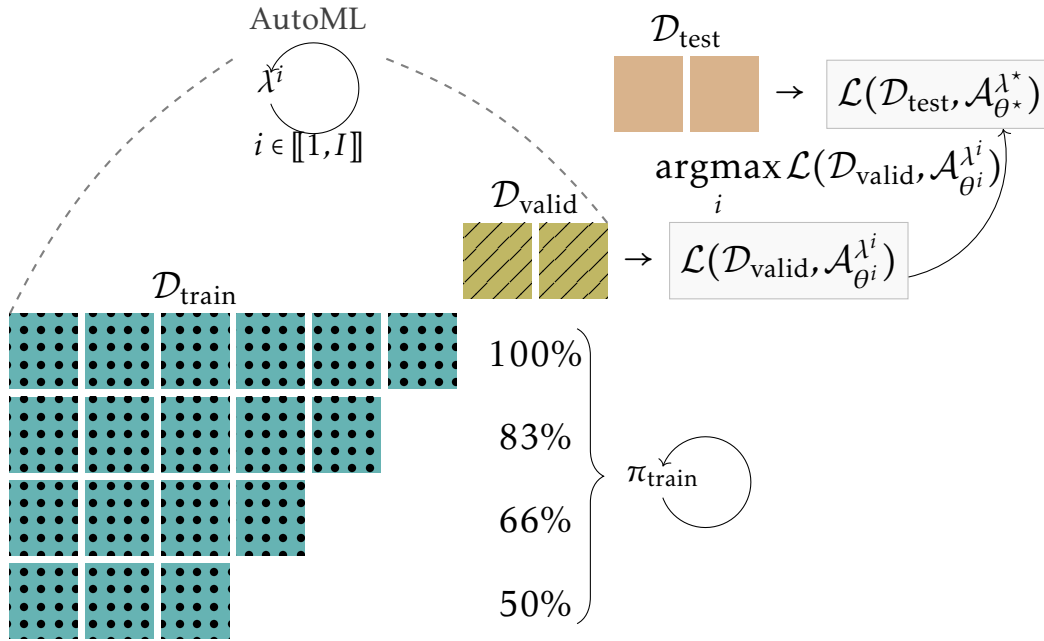


Figure 6.1: Multi-fidelity AutoML on the training dataset. The optimization of λ^i HPs is conjoined to the one of π_{train}

$$c_{\text{out}} \triangleq 1 - \beta'_{\text{train}} - \beta \quad . \quad (6.1)$$

So we obtain a stochastic value (depending on the shuffling of the dataset), describing how far the training of \mathcal{N} went before being stopped. If the training of \mathcal{N} was not stopped, then $c_{\text{out}} < 0$. A hypergeometric distribution can model the probability of encountering non-spiking samples within the first n samples from $\mathcal{D}'_{\text{train}}$ during training. If X is the number of non-spiking samples encountered after n draws, then $X \sim \text{Hypergeometric}(n, K, |\mathcal{D}'_{\text{train}}|)$. However, we cannot know a-priori the values of K , i.e., the number of non-spiking samples in $\mathcal{D}'_{\text{train}}$. Moreover, this value K may vary through epochs. Thus, in probability, the higher K , the lower β'_{train} and the sooner \mathcal{N} should be stopped. Therefore, one of the objectives of the optimization algorithm is to minimize c_{out} to ensure minimal per-sample spiking activity.

6.1.2 Cost aware Bayesian optimization

Difficulties arise when dealing with multi-fidelity. A solution to this is MOGP, allowing to jointly model with a single GP, all discrete fidelities [78]. MOGPs allow the transfer of information between levels of fidelities. In [78], the authors proposed a multi-fidelity approach to TuRBO implying discrete fidelities, modeled by a MOGP outputting a M -dimensional mean and a $M \times M$ co-variance matrix for each sample from the posterior, where M is the number of fidelities. A major drawback of this approach is the discretization of a single HP describing the fidelity. We saw that when SNNs are simulated, the fidelity can be expressed as a combination of different HPs that can have large bounds or be continuous. Moreover, some fidelity HPs (e.g. T) might sometimes result in lower performances when their value is increased. Thus, a solution to our problem is to consider a generalization of multi-fidelity BO, known as Cost-Apportioned Bayesian Optimization (CARBO) [154]. Here, the idea is to train two GPs, one for estimating the actual objective function, and a second to impute the computational cost of a HPs combination. The algorithm then optimizes an

Algorithm 23 SNN training with improved constraint

Inputs: $\mathcal{N}_\theta^\lambda$ *Network* $\mathcal{D}_{\text{train}}$ *Training data* π_{train} *Proportion of $\mathcal{D}_{\text{train}}$*

epochs

Number of epochs α *Minimum spiking activity* β *Maximum proportion of non spiking samples***Outputs:** $\mathcal{N}_{\theta^*}^\lambda, \beta'_{\text{train}}$ $\mathcal{D}'_{\text{train}} \leftarrow \text{SUBSET}(\mathcal{D}_{\text{train}}, \pi_{\text{train}})$ *Extract π_{train} % from $\mathcal{D}_{\text{train}}$* out $\leftarrow \emptyset$ *Output spikes* $i, e \leftarrow 1$ **while** $(\beta_{\text{train}} \leq \beta) \wedge (e \leq \text{epochs})$ **do** nscout $\leftarrow 0$ *Number of non spiking samples* scout $\leftarrow 0$ *Number of spiking samples* **while** $(\beta_{\text{train}} \leq \beta) \wedge (i \leq |\mathcal{D}'_{\text{train}}|)$ **do** out $\leftarrow \text{Train}(\mathcal{N}_\theta^\lambda, \mathcal{D}'_{\text{train}}[i])$ **if** SUM(out) $< \alpha$ **then***Number of output spikes* nscout $\leftarrow \text{nscout} + 1$ $\beta_{\text{train}} \leftarrow \frac{\text{nscout}}{|\mathcal{D}'_{\text{train}}|}$ *Ratio of non spiking samples* **else** scout $\leftarrow \text{scout} + 1$ $\beta'_{\text{train}} = \frac{\text{scout}}{|\mathcal{D}'_{\text{train}}|}$ *Ratio of spiking samples* $i \leftarrow i + 1$ $e \leftarrow e + 1$ **return** $\mathcal{N}_{\theta^*}^\lambda, \beta'_{\text{train}}$

acquisition function known as the cost-cooling EIpu:

$$\text{EI-cool}(\lambda) \triangleq \frac{\text{EI}(\lambda)}{\tilde{C}(\lambda)^\kappa} , \quad (6.2)$$

where λ is a HPs combination and \tilde{C} the positive mean from the GP posterior distribution (on the costs) for a given sample λ . The annealing parameter κ , a function decreasing from 1 to 0, describes the contribution of the cost on the EIpu. This allows the algorithm to focus on cheap computations at the beginning and on expensive ones by the end of the remaining budget. When $\kappa \rightarrow 0$, the acquisition function becomes the usual EI. Unfortunately, a major drawback of EIpu is its difficult tractability toward asynchronous parallelization.

6.1.3 Cost aware Thompson sampling

In the following section, we describe our modification of the SCBO algorithm, specifically designed for our HPO problem, so to handle the cost of evaluations. We named this algorithm CASCBO. The following description is based on appendix A.

In SCBO [62], TS replaces the acquisition function, and is adapted to black-box constraints. In this work, the accuracy, the constraint on the outputs, and the cost are modeled by their respective GP. For simplicity, we write the performances of a network as $f(\lambda) \triangleq \text{Accuracy}(\mathcal{N}_{\theta^*}^\lambda, \mathcal{D}_{\text{valid}})$, the value of the constraint for the HPs combination λ is written $c_{\text{out}}(\lambda)$, and $\text{cost}(\lambda)$ is the positive computational cost of training \mathcal{N}^λ . Therefore, CASCBO maintains 3 Gaussian Processes for the 3 previous functions. The resulting prior distributions are written :

$$\begin{aligned} p(f(\lambda) | \mathcal{D}^{(\text{acc})}) &= \mathcal{GP}(\mu_{\mathcal{D}^{(\text{acc})}}, K_{\mathcal{D}^{(\text{acc})}}) , \\ p(c_{\text{out}}(\lambda) | \mathcal{D}^{(c)}) &= \mathcal{GP}(\mu_{\mathcal{D}^{(c)}}, K_{\mathcal{D}^{(c)}}) , \\ p(\text{cost}(\lambda) | \mathcal{D}^{(\text{cost})}) &= \mathcal{GP}(\mu_{\mathcal{D}^{(\text{cost})}}, K_{\mathcal{D}^{(\text{cost})}}) , \end{aligned} \quad (6.3)$$

where $\mathcal{D}^{(\text{acc})}$, $\mathcal{D}^{(c)}$, and $\mathcal{D}^{(\text{cost})}$ are the archives of the accuracy, constraint values, and costs regarding the computed solutions λ . The mean functions are denoted $\mu_{\mathcal{D}}^{(\cdot)}$, and $K_{\mathcal{D}}^{(\cdot)}$ are the covariance matrices. For simplicity, we write a realization of $p(f(\lambda) | \mathcal{D}^{(\text{acc})})$, $p(c_{\text{out}}(\lambda) | \mathcal{D}^{(c)})$, and $p(\text{cost}(\lambda) | \mathcal{D}^{(\text{cost})})$ as r_f , $r_{c_{\text{out}}}$ and r_{cost} . TuRBO, SCBO and CASCBO, build batches of realizations B , and extract a potentially optimal subset $S \subset B$ by redefining the relation *is better than* ($>$) between two potentially optimal candidates λ_1 and λ_2 (see A).

Inspired by the EIpu [154], described by equation 6.2, and to include cost in equation A.6, we propose to compute a greedy improvement based on the current best solution λ_{best} , with $\Delta(\lambda) = f(\lambda_{\text{best}}) - f(\lambda)$ or $\Delta_r(\lambda) = f(\lambda_{\text{best}}) - r_f(\lambda)$. Then, according to the sign of $\Delta(\lambda)$, the improvement will be weighted or penalized by $\mathbb{E}[\text{cost}(\lambda) | \mu_{\mathcal{D}^{(\text{cost})}}, K_{\mathcal{D}^{(\text{cost})}}]$. If c_{out} is positive, then it should be penalized by $\mathbb{E}[\text{cost}(\lambda) | \mu_{\mathcal{D}^{(\text{cost})}}, K_{\mathcal{D}^{(\text{cost})}}]$. We define the score functions to be minimized for a given HPs combination $s: \lambda \rightarrow \mathbb{R}$ for actual sampled values in the archive, and $s_r: \lambda \rightarrow \mathbb{R}$ for realizations, such that:

$$s(\lambda) = \begin{cases} \Delta(\lambda) / \text{cost}(\lambda)^{\kappa(\Delta_t)} , & \text{if } (\Delta(\lambda) < 0) \wedge (c_{\text{out}}(\lambda) < 0) \\ \Delta(\lambda) \times \text{cost}(\lambda)^{\kappa(\Delta_t)} , & \text{if } (\Delta(\lambda) \geq 0) \wedge (c_{\text{out}}(\lambda) < 0) \\ c_{\text{out}}(\lambda) \times \text{cost}(\lambda)^{\kappa(\Delta_t)} , & \text{if } c_{\text{out}}(\lambda) < 0 \end{cases} . \quad (6.4)$$

$$s_r(\lambda) = \begin{cases} \Delta_r(\lambda) / r_{\text{cost}}(\lambda)^{\kappa(\Delta_t)} , & \text{if } (\Delta_r(\lambda) < 0) \wedge (r_{\text{cout}}(\lambda) < 0) \\ \Delta_r(\lambda) \times r_{\text{cost}}(\lambda)^{\kappa(\Delta_t)} , & \text{if } (\Delta_r(\lambda) \geq 0) \wedge (r_{\text{cout}}(\lambda) < 0) \\ r_{\text{cout}}(\lambda) \times r_{\text{cost}}(\lambda)^{\kappa(\Delta_t)} , & \text{if } r_{\text{cout}}(\lambda) < 0 \end{cases} , \quad (6.5)$$

with Δ_t the remaining budget at a time t and $\kappa : \mathbb{R} \rightarrow [0, 1]$ the annealing function strictly decreasing to 0.

Then, $\lambda_1 > \lambda_2$ if $(c_{\text{out}}(\lambda_1) < 0) \wedge (c_{\text{out}}(\lambda_2) \geq 0)$; otherwise, $\lambda_1 > \lambda_2$ if $s(\lambda_1) < s(\lambda_2)$. For a realization, $\lambda_1 > \lambda_2$ if $(r_{\text{cout}}(\lambda_1) < 0) \wedge (r_{\text{cout}}(\lambda_2) \geq 0)$; otherwise, $\lambda_1 > \lambda_2$ if $s_r(\lambda_1) < s_r(\lambda_2)$. In other words, if λ_1 follows the constraints while λ_2 violates at least one of them, then $\lambda_1 > \lambda_2$. But if both λ_1 and λ_2 follow or violate the constraints, then the scoring functions are used. So at each iteration, CASCBO returns a batch of candidates S , containing HPs combinations λ with the best greedy improvement per cost unit if $\Delta_r(\lambda)$ is negative. Otherwise, it should return the less costly λ compared to the potential loss of performances. If none of the realizations are feasible, then it will return the solutions with the minimum constraint violation penalized by the cost. The best solution found so far λ_{best} , is not solely based on the validation accuracy but is determined by equations 6.4 applied to the archives of actually sampled and computed HPs combinations.

6.2 Experimental setup

In this study, 18 distinct experiments were conducted. We optimize a search space of 22 HPs applied globally to the network, and search spaces of up-to 46 HPs applied layer-wise. We classify MNIST [153] and NMNIST [196], with the same CSNN architecture as described in chapter 5. Furthermore, we also optimized the architecture from [17] made of 3 hidden feed-forward layers (see table 6.1) to classify the SHD dataset with a search space of 13 HPs applied globally and another one of 21 HPs applied layer-wise. The experiments using SpikingJelly have less HPs since the neuron model is simpler than the one of LAVA-DL. We add two new HPs, $\pi_{\text{train}} \in (0, 1]$ the proportion of $\mathcal{D}_{\text{train}}$ used to extract the training set $\mathcal{D}'_{\text{train}} \subseteq \mathcal{D}_{\text{train}}$. And we add the number of frames (duration T) to encode a sample to the list of fidelity HPs. Other HPs, also have an impact on the evaluation cost, such as the batch size or the number of epochs. We discarded the DvsGesture dataset, as it has too few samples to efficiently apply multi-fidelity HPO using the π_{train} HP.

We use LIF and PLIF neurons with respectively the LAVA-DL and SpikingJelly simulators. We use the previously described CASCBO algorithm for HPO, and we compare the algorithm to RS on the search spaces with the highest number of HPs.

Batches from the datasets have the following shape : $B.T.C.H.W$, i.e. batch size, frames, channels, height, and width. The batch size and number of frames are HPs to be tuned. The MNIST, NMNIST and SHD datasets follow the same pattern, respectively $(B.T.1.28.28)$, $(B.T.2.34.34)$ and $(B.T.1.700)$. The Tonic [155] Python package was used to load and convert DVS data into frames. Both MNIST and NMNIST have the same number of samples. They are divided into training ($\mathcal{D}_{\text{train}}$), validation ($\mathcal{D}_{\text{valid}}$), and test ($\mathcal{D}_{\text{test}}$) sets of respective sizes 48000, 12000, and 10000. The SHD dataset contains fewer samples than previous datasets but was still divided into three subsets of sizes 6524, 1632, and 2264. All validation sets are randomly extracted from the training sets while keeping class proportions. A cluster of GPU from Grid5000 [13] was used for computations. A total of 16 NVIDIA A-100 (40 GB) were used; one of them was dedicated to the computation of CASCBO. Each node contains 4 GPUs, a 32-core AMD EPYC 7513 (Zen 3) CPU and 512 GiB of RAM. CASCBO was parallelized

Table 6.1: Architecture details of SNNs architectures applied to SHD.

Experiment	Architecture	Neuron model	Encoding
21-C-SLAY-SHD	Inputs \rightarrow FeedForward \rightarrow FeedForward \rightarrow FeedForward \rightarrow Outputs	Adaptive LIF (chapter 2)	LAUSCHER [37]
42-C-SLAY-SHD			
21-R-SLAY-SHD			
42-R-SLAY-SHD			
13-C-SuGr-SHD		PLIF (chapter 2)	
21-C-SuGr-SHD			
13-R-SuGr-SHD			
21-R-SuGr-SHD			

using OpenMPI, instantiated with Zellij¹ and BoTorch [12].

We adopt a similar experiment name coding as in chapter 5: number of HPs-algorithm-training-dataset. Thus, the experiments are, 22-C-SLAY-MNIST, 46-C-SLAY-MNIST, 15-C-SuGr-MNIST, 21-C-SuGr-MNIST, 22-C-SLAY-NMNIST, 46-C-SLAY-NMNIST, 15-C-SuGr-NMNIST, 21-C-SuGr-NMNIST, 21-C-SLAY-SHD, 42-C-SLAY-SHD, 21-C-SuGr-SHD and 21-C-SuGr-SHD. For RS we have, 46-R-SLAY-MNIST, 46-R-SLAY-NMNIST, 42-R-SLAY-SHD, 21-R-SuGr-MNIST, 21-R-SuGr-NMNIST, and 21-R-SuGr-SHD. These experiments are summed up in table 6.2. Experiments on NMNIST last for 40 hours, as NMNIST is more computationally expensive than MNIST. The other experiments have a duration of 14 hours. A practical reason for these HPO time budgets is the availability of resources that are easily accessible for 14 hours straight, while resources for longer experiments are harder to schedule. All search spaces are available in appendix E.

¹<https://github.com/ThomasFirmin/zellij>

Table 6.2: Summary of experiments

Experiments	α, β	Budget	HPs	Simulator	Archi.	Dataset	Source
Cost Aware SCBO							
22-C-SLAY-MNIST	5,3%	224	22	LAVA-DL	CSNN	MNIST	99.36 \pm 0.05 [247]
46-C-SLAY-MNIST	5,3%	224	46	LAVA-DL	CSNN	MNIST	
15-C-SuGr-MNIST	5,3%	224	15	SpikingJelly	CSNN	MNIST	
21-C-SuGr-MNIST	5,3%	224	21	SpikingJelly	CSNN	MNIST	
22-C-SLAY-NMNIST	5,3%	640	22	LAVA-DL	CSNN	NMNIST	99.20 \pm 0.02 [247]
46-C-SLAY-NMNIST	5,3%	640	46	LAVA-DL	CSNN	NMNIST	
15-C-SuGr-NMNIST	5,3%	640	15	SpikingJelly	CSNN	NMNIST	
21-C-SuGr-NMNIST	5,3%	640	21	SpikingJelly	CSNN	NMNIST	
21-C-SLAY-SHD	10,1%	224	21	LAVA-DL	3-layers	SHD	70.58 \pm 1.9 [17]
42-C-SLAY-SHD	10,1%	224	42	LAVA-DL	3-layers	SHD	
13-C-SuGr-SHD	10,1%	224	13	SpikingJelly	3-layers	SHD	
21-C-SuGr-SHD	10,1%	224	21	SpikingJelly	3-layers	SHD	
Random Search							
46-R-SLAY-MNIST	5,3%	224	46	LAVA-DL	CSNN	MNIST	99.36 \pm 0.05 [247]
21-R-SuGr-MNIST	5,3%	224	21	SpikingJelly	CSNN	MNIST	
46-R-SLAY-NMNIST	5,3%	640	46	LAVA-DL	CSNN	NMNIST	99.20 \pm 0.02 [247]
21-R-SuGr-NMNIST	5,3%	640	21	SpikingJelly	CSNN	NMNIST	
42-R-SLAY-SHD	10,1%	224	42	LAVA-DL	3-layers	SHD	70.58 \pm 1.9 [17]
21-R-SuGr-SHD	10,1%	224	21	SpikingJelly	3-layers	SHD	

6.3 Results and discussion

6.3.1 Analysis of the best solutions

Results presented in Table 6.2 are selected according to the best accuracy on $\mathcal{D}_{\text{valid}}$ found by CASCBO, and final accuracy is computed on $\mathcal{D}_{\text{test}}$. To evaluate stochasticity of the best solutions, SNNs are retrained multiple times during 100 epochs. To our knowledge, in the literature of HPO applied to SNNs, we optimized the highest number of HPs while maintaining competitive accuracies on standard benchmarks. Although experiments 46-C-SLAY-MNIST, 46-C-SLAY-NMNIST, 42-C-SLAY-SHD, 21-C-SuGr-MNIST, 21-C-SuGr-NMNIST, 21-C-SuGr-SHD, are more difficult due to the higher number of HPs, but this does not always translate into lower accuracies. Due to potential overfitting noticed in chapter 5, we replaced the ADAM optimizer by SGD and a learning rate schedule for image recognition tasks. Despite unsuccessful experiments with regularization methods to handle overfitting, such as L2 or neuron dropout, we replaced ADAM by SGD, since some works indicate that SGD generalizes better than ADAM for image recognition tasks [140, 303].

In Table 6.2, one can even observe better performances when HPs are optimized layer-wise. However, it is not always the case, and this difference tends to be less pronounced for PLIF-SpikingJelly experiments, maybe because more parameters are optimized ($\mathbf{W}, \theta, \tau_{\text{leak}}$).

Concerning experiments 22-C-SLAY-MNIST, 46-C-SLAY-MNIST, 22-C-SLAY-NMNIST, 46-C-SLAY-NMNIST, and compared to the baseline results trained by SLAYER [247], we can notice that our results are more stochastic, even if they are close to the baseline, and slightly better for 46-C-SLAY-NMNIST. We assume this stochasticity might be explained by the usage of adaptive LIF rather than SRM neurons. An additional source of uncertainty may arise from the difference in simulators used. The baseline was computed using the original SLAYER simulator², whereas we used SLAYER-2 from LAVA-DL. Moreover, our networks are trained on fewer data points than the given baseline. Indeed, in chapter 2 we discussed a good practice [120, 20] in HPO consisting of having three subsets of data, $\mathcal{D}_{\text{train}}$, $\mathcal{D}_{\text{valid}}$ and $\mathcal{D}_{\text{test}}$, to prevent overfitting the HPs and to guarantee a certain degree of generalizability. When doing HPO, a bias is introduced because of an additional step in the design of a SNN.

Concerning PLIF-SpikingJelly experiments, 15-C-SuGr-MNIST and 21-C-SuGr-MNIST are better than the SLAYER baseline [247], but despite the same architecture, more parameters are optimized. The experiments 15-C-SuGr-NMNIST and 21-C-SuGr-NMNIST obtained far better performances (99.33 ± 0.10 , 99.24 ± 0.29) than the baseline at 99.20 ± 0.02 . In these experiments, switching from SRM to PLIF neurons improves the performances.

Additionally, for all MNIST and NMNIST experiments we observe better performances on $\mathcal{D}_{\text{test}}$ than on the validation set $\mathcal{D}_{\text{valid}}$, which is a clue that the approach generalizes well, despite higher performances on $\mathcal{D}_{\text{train}}$.

Concerning experiments 21-C-SLAY-SHD, 42-C-SLAY-SHD, the results indicate that by tuning HPs, one could significantly improve the accuracy of handcrafted architecture presented in [17]. The baseline obtained a validation accuracy of 70.58 ± 1.9 with SLAYER and 78.01 ± 0.2 with EXODUS (a SLAYER-based alternative), while we obtained up to 92.2% testing accuracy. We also observe an indication of potential overfitting since the differences between training, validation, and test accuracies are clear. We notice a loss of about 9% accuracy between the validation and the testing accuracies. Indeed, for 21-C-SLAY-SHD the training accuracy reaches 100% with a maximum validation accuracy of 99.20%, the test accuracy drops to a maximum of 91.29%. A similar behavior is observed for 42-C-SLAY-SHD.

Although performances on SHD with PLIF-SpikingJelly are still better than the baseline, we clearly observe overfitting. This overfitting can be explained by more training parameters

²<https://bitbucket.org/bamsumit/slayer>

Table 6.3: Results and performances of best solutions from all experiments

Experiments	HPs	Train	Valid	Test	#	%sil.	Hrs	%bud.	Cost
With constraints and early stopping									
22-C-SLAY-MNIST	22	97.62 \pm 0.94	97.36 \pm 0.74	97.56 \pm 0.70	2073	55.3	14	2.2	1229
46-C-SLAY-MNIST	46	99.66 \pm 0.12	99.06 \pm 0.13	99.06 \pm 0.14	1823	70.5	14	1.6	6106
15-C-SuGr-MNIST	15	99.80 \pm 0.02	99.24 \pm 0.09	99.37 \pm 0.06	2932	40.8	14	1.0	631
21-C-SuGr-MNIST	21	99.92 \pm 0.01	99.33 \pm 0.07	99.41 \pm 0.05	2263	61.4	14	1.3	1628
22-C-SLAY-NMNIST	22	94.75 \pm 2.81	94.51 \pm 2.75	94.96 \pm 2.26	2291	57.0	40	0.6	4255
46-C-SLAY-NMNIST	46	98.94 \pm 0.46	98.18 \pm 0.34	98.85 \pm 0.38	3666	55.9	40	4.0	1523
15-C-SuGr-NMNIST	15	99.89 \pm 0.02	99.18 \pm 0.06	99.33 \pm 0.10	6461	27.1	40	0.6	636
21-C-SuGr-NMNIST	21	99.76 \pm 0.04	99.06 \pm 0.09	99.24 \pm 0.29	8492	17.6	40	0.4	345
21-C-SLAY-SHD	21	94.21 \pm 0.67	98.62 \pm 0.58	89.55 \pm 1.74	3469	45.5	14	1.6	437
42-C-SLAY-SHD	42	99.99 \pm 0.01	98.47 \pm 0.67	90.19 \pm 2.43	2367	49.2	14	2.1	1029
13-C-SuGr-SHD	13	99.97 \pm 0.03	94.24 \pm 2.51	78.67 \pm 4.81	1775	69.9	14	2.3	1805
21-C-SuGr-SHD	21	99.95 \pm 0.03	93.90 \pm 4.81	78.40 \pm 2.39	2199	85.5	14	3.0	3571
Without constraints and early stopping									
46-R-SLAY-MNIST	46	68.44 \pm 9.16	82.82 \pm 9.30	85.95 \pm 8.76	689	82.9	14	82.7	2914
21-R-SuGr-MNIST	21	72.22 \pm 19.35	76.22 \pm 19.36	76.35 \pm 19.77	686	99.7	14	99.8	2577
46-R-SLAY-NMNIST	46	94.44 \pm 1.76	95.32 \pm 1.83	95.37 \pm 1.93	1925	87.7	40	84.7	5231
21-R-SuGr-NMNIST	21	94.86 \pm 0.08	95.45 \pm 0.92	95.50 \pm 0.81	2042	99.9	40	99.9	1732
42-R-SLAY-SHD	42	98.63 \pm 0.79	93.38 \pm 3.25	85.31 \pm 4.53	595	94.8	14	93.9	1641
21-R-SuGr-SHD	21	99.93 \pm 0.04	94.45 \pm 1.13	78.91 \pm 1.88	366	80.1	14	74.0	4938

and by the dataset itself. Indeed, we split the original training dataset into $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{valid}}$, while we kept the original $\mathcal{D}_{\text{test}}$ untouched for better comparison with the literature. However, in the way SHD is made, the sound records in the original testing dataset are made with different speakers than the original training dataset. Thus, the training and validation splits are made of the same speakers, while the test dataset $\mathcal{D}_{\text{test}}$ is made with different speakers than $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{valid}}$. Therefore, SHD could be extended with a validation dataset with new speakers for HPO and manual tuning. Future neuromorphic datasets should consider this possibility to perform fair and better HPO.

In figure 6.2, we used the t-SNE algorithm [171] to apply dimensionality reduction on evaluated solutions produced by CASCBO. One can see there exist different areas (clusters of points) of the search space containing good solutions, meaning that there is not a unique suitable HPs combination. The darkest and biggest clusters, in all figures, indicate groups of solutions that are mostly silent networks. We can deduce that some part of the search space, distinct from the ones with high-accuracy solutions, contains many infeasible solutions, i.e. silent networks. So, multi-fidelity combined with constraints and early stopping allows for quick escape from areas where silent networks are located.

For experiments 13-C-SuGr-SHD and 21-C-SuGr-SHD, in figure 6.2 we distinguish large clusters of bad solutions. This illustrates the difficulty of doing HPO for these experiments. Whereas for their LIF-SLAYER counterparts, 21-C-SLAY-SHD and 42-C-SLAY-SHD, we can clearly distinguish clusters of bad and good solutions.

For NMNIST experiments 15-C-SuGr-NMNIST and 21-C-SuGr-NMNIST, we clearly distinguish multiple clusters of good solutions, while there is one very compact cluster of bad solutions. According to the colors, there is clearly a binary partition of the sampled solutions for these two experiments. This is an additional clue comforting that for PLIF-SpikingJelly experiments on MNIST-NMNIST, as soon as there is enough spiking activity we can ensure high accuracy. For LIF-SLAYER experiments on NMNIST, the bad clusters appear less compact. We can better distinguish the gradient of colors between multiple clusters, meaning that minimum spiking activity is necessary to obtain suitable solutions, but this does not solely explain top accuracy.

Concerning the π_{train} HP, as illustrated in figure 6.3, it is not always necessary to use the full training dataset to obtain good accuracies. With proper parameter tuning, we can reach about 96% of accuracy with only using 10% of $\mathcal{D}_{\text{train}}$ on MNIST. Most of the dataset is then used to refine the accuracy to reach about 99% validation accuracy. Similar behaviors can be observed on NMNIST. For the SHD dataset, we clearly distinguish the loss of accuracy when π_{train} is lower.

Concerning the HP describing the number of frames T , and illustrated in figure 6.4. For experiments 22-C-SLAY-MNIST, 46-C-SLAY-MNIST, 22-C-SLAY-NMNIST, and 46-C-SLAY-NMNIST, the minimal number of frames to obtain validation accuracies $> 95\%$ is of about 30. However, for PLIF-SpikingJelly experiments, even 10 frames gives acceptable results. For 21-C-SLAY-SHD, when $T > 300$, the validation accuracy decreases drastically, while in 42-C-SLAY-SHD the best solutions are around $325 < T < 400$. This illustrates the intricate relationships between HPs, emphasizing that the performances of SNNs rely on complex combinations of HPs.

6.3.2 Analysis of CASCBO

In this section, we analyze the behaviors of CASCBO during the optimization process. The table 6.2 sums up the number of evaluated HPs combinations (#), the proportion of sampled silent networks (%sil.), their influence on the budget in GPU hours (%budg.), and the training cost in seconds, of the best solution found.

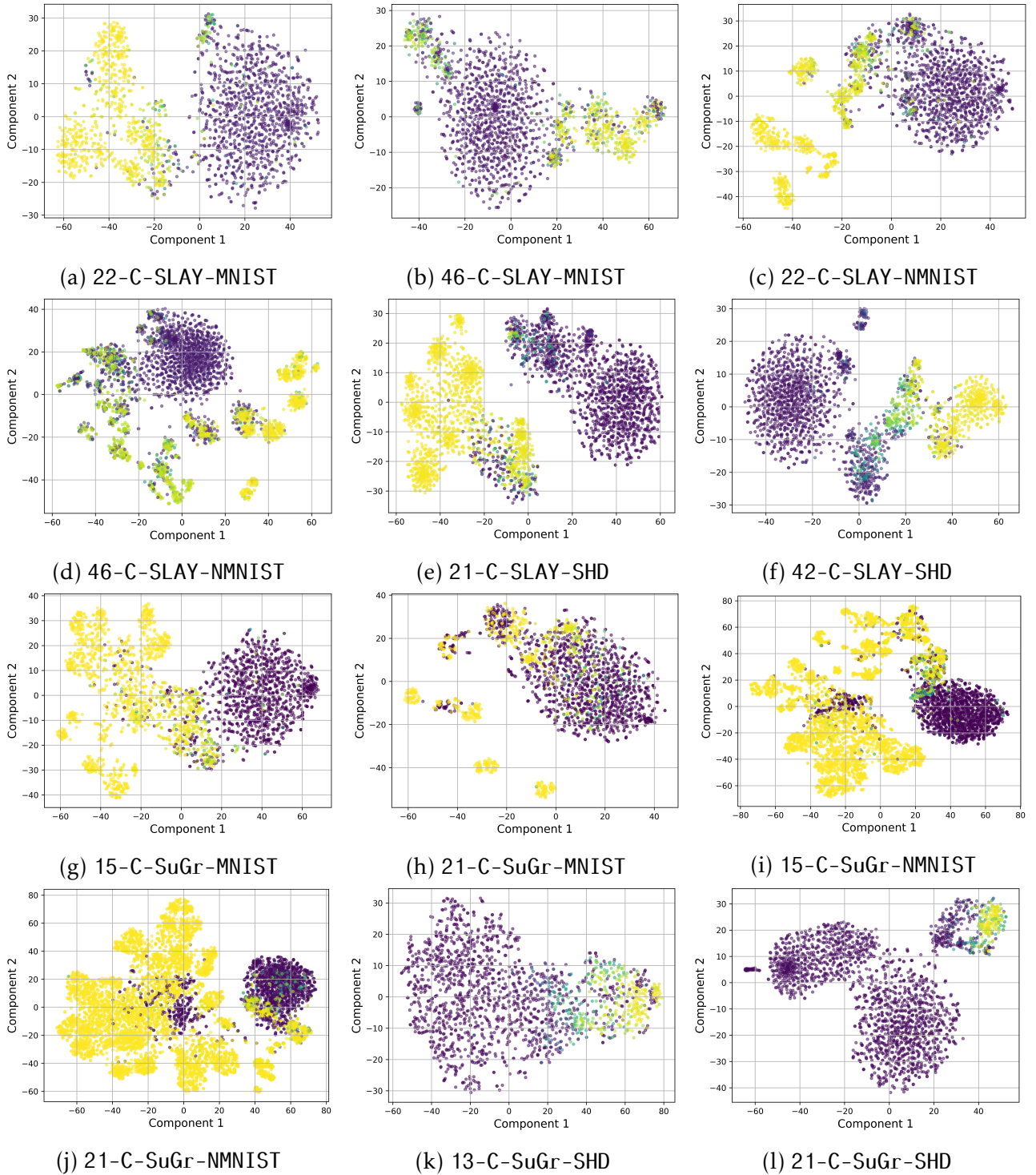


Figure 6.2: t-SNE applied on all evaluated solutions returned by CASCBO. The lighter or yellower the point, the higher the validation accuracy.

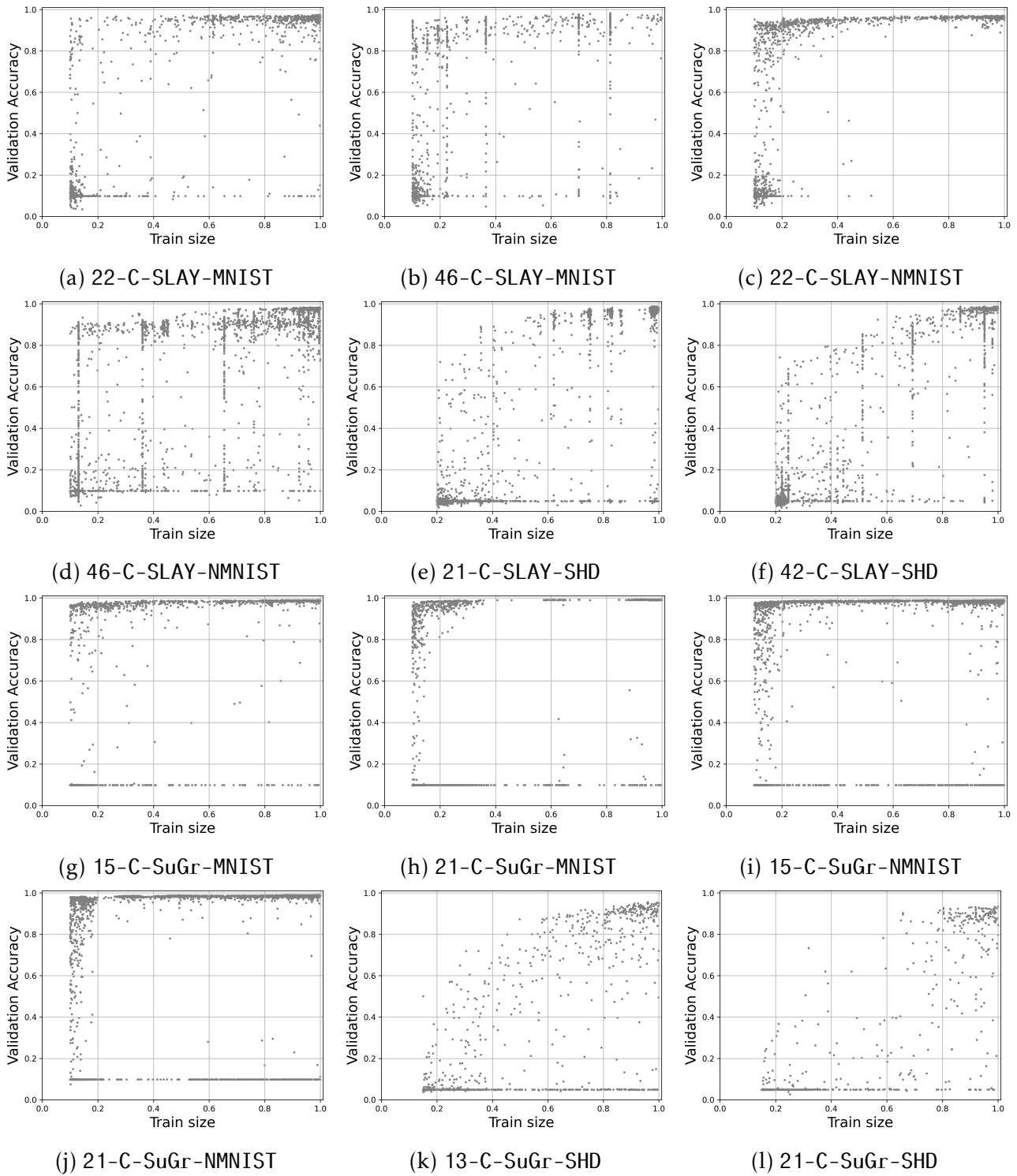
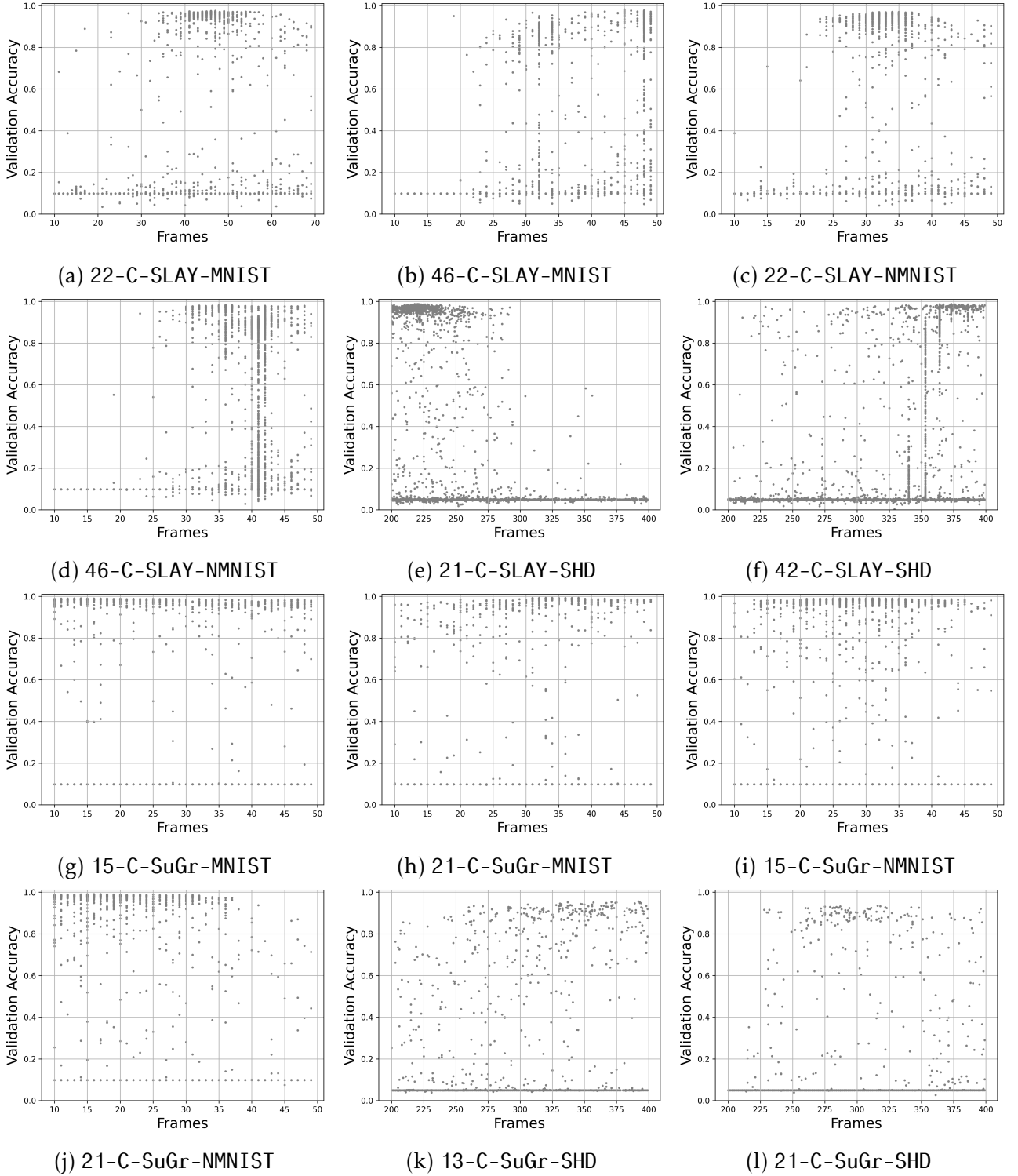


Figure 6.3: Accuracy according to π_{train} for all CASCBO experiments.

Figure 6.4: Accuracy according to T for all CASCBO experiments.

Concerning LIF-SLAYER experiments, one can see that while almost half of the sampled solutions are silent networks (70% for 46-C-SLAY-MNIST), their impact on the budget is greatly diminished, from 0.6% up to only 4%. Additionally, figure 6.5 illustrates that most of the silent networks are sampled at the very beginning of the optimization (red lines) when π_{train} and the costs are low. Then, CASCBO can quickly converge toward fully trained networks (black lines).

In chapter 5, the HPO on MNIST with a budget of 1600 GPU hours, the LIF-SLAYER experiment obtained a maximum of 98.99% accuracy, and 99.57% for PLIF-SpikingJelly. According to the results presented in table 6.3, here with 7.6 times less budget (224 GPU hours), we obtained for search spaces of 22 HPs, a maximum of 98.26% and 99.43%. Whereas, for the search spaces of 46 HPs, we get 99.20% and 99.46% validation accuracies. Despite a slight decrease in performances, we still obtained competitive accuracy with much less budget. Moreover, the switch from ADAM to SGD did not help to improve the performances and seems to have generated noisier solutions.

Figure 6.6 illustrates the acceleration between experiments from chapter 5 on MNIST, and the multi-fidelity experiments. The experiments 22-C-SLAY-MNIST and 15-C-SuGr-NMNIST are rescaled within a 100 hours timescale. We clearly see the advantage of the cost-aware approach. Indeed, 22-C-SLAY-MNIST converges in 11 hours and 15-C-SuGr-MNIST in 9 hours, while S-SLAY-MNIST converges in 62 hours and S-SuGr-MNIST in 40 hours. For 46-C-SLAY-MNIST and 21-C-SuGr-MNIST, in figure 6.5 the convergence is slower, but the search space is much bigger, making the optimization noisier.

Concerning PLIF-SpikingJelly experiments, for 15-C-SuGr-MNIST and 21-C-SuGr-MNIST, similar behaviors are observed, compared to their LIF-SLAYER counterparts. Between 40% and 60% of silent networks are sampled, but they only consumed about 1% of the budget. However, for 15-C-SuGr-NMNIST and 21-C-SuGr-NMNIST, only 17% to 27% of silent networks are generated and consumed less than 1% of the budget. This can be explained by faster convergence with the SpikingJelly simulator and a focus on cheap networks. Therefore, 14 hours (224 GPU hours) are sufficient to obtain top-performing accuracy with SpikingJelly on NMNIST. In these two experiments, about 2.5 hours are needed to reach about 98.75% validation accuracy, and about 7.5 additional hours are needed to obtain 99.00% accuracy.

However, while SpikingJelly performs better than LIF-SLAYER on MNIST and NMNIST, it is clear that the convergence is more challenging on SHD. Even if the number of optimized HPs is lower than for LIF-SLAYER experiments. It can be explained by a higher rate of silent networks within the search spaces of SpikingJelly experiments on SHD, and a focus on expensive networks. We have:

- 21-C-SLAY-SHD: 45.5% of evaluations are silent networks, and costed 437 GPU hours.
- 13-C-SuGr-SHD: 69.9% of evaluations are silent networks, and costed 1805 GPU hours.
- 42-C-SLAY-SHD: 49.2% of evaluations are silent networks, and costed 1029 GPU hours.
- 21-C-SuGr-SHD: 85.5% of evaluations are silent networks, and costed 3571 GPU hours.

This illustrates the sensitivity of SNNs regarding the classification task, the HPs, the architecture, and simulators.

The differences in number of evaluated solutions between all experiments can be explained by a focus of CASCBO on more or less costly solutions; this is explained by a higher or lower number of epochs, batch sizes, or frames T .

Finally, these experiments illustrate how difficult it is to improve performances by about 0.25% accuracy, as depicted in figure 6.5. Moreover, experiments on MNIST and NMNIST can quickly obtain very high accuracy, while we see slower convergence for SHD experiments.

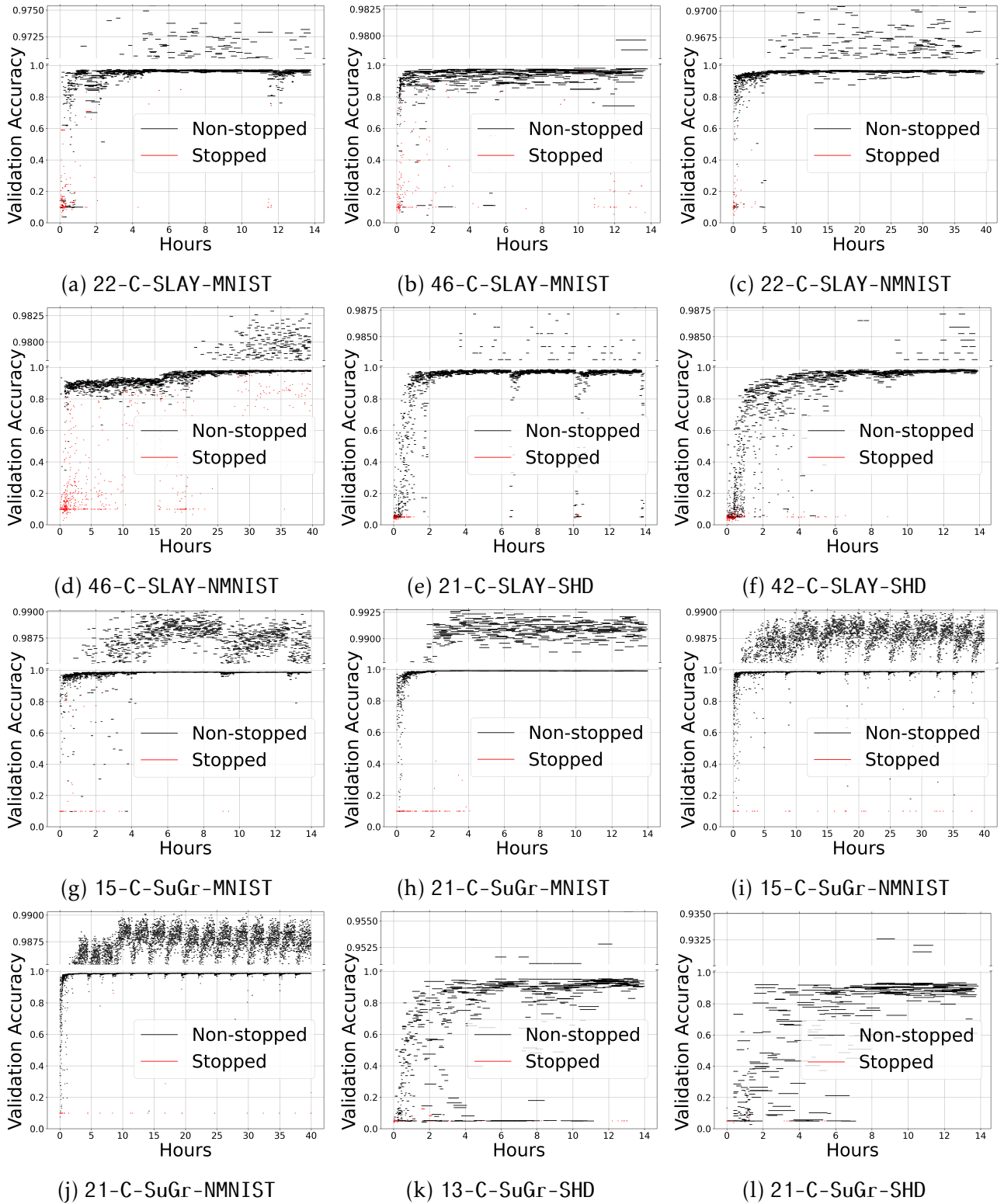


Figure 6.5: Computation start date to end date compared to accuracy of each SNN optimized by CASCBO.

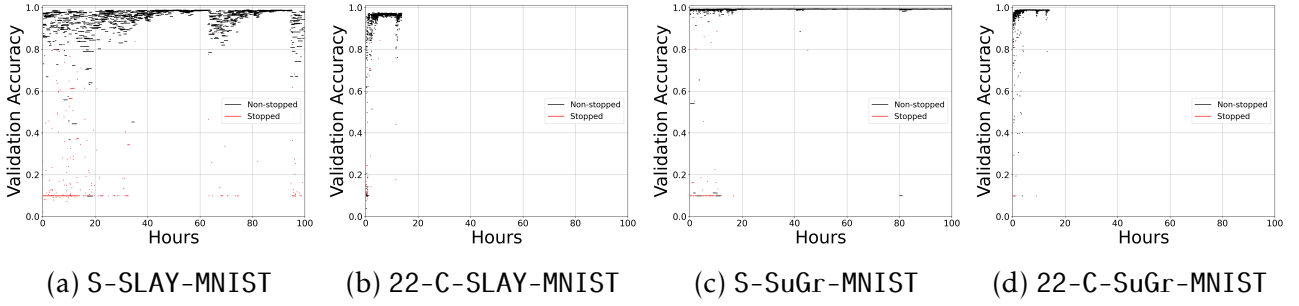


Figure 6.6: Comparison between results obtained on MNIST between experiments of chapter 5 and 6.

These challenges emphasize even more our approach when compared to baseline networks, which are designed and trained on solely a training and test datasets. While we have to train on a smaller training set by splitting the original training dataset into $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{valid}}$.

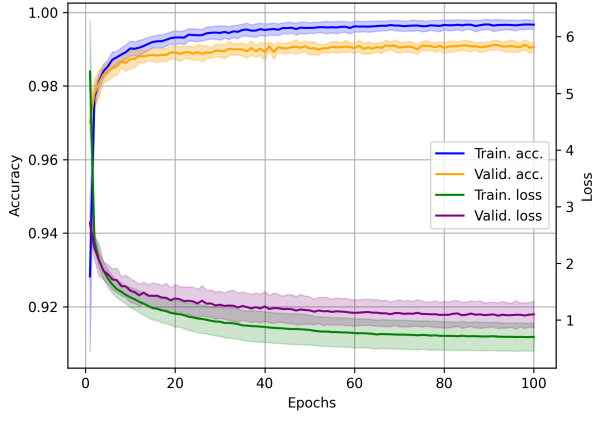
6.3.3 Comparison with Random Search

We applied the RS algorithm to the largest search spaces to compare the relevancy of CASCBO. It allows us to determine if the optimized results obtained with CASCBO are explained by random or by a converging process. We also investigate how resources and budgets are distributed in RS in order to determine whether good solutions are common or not.

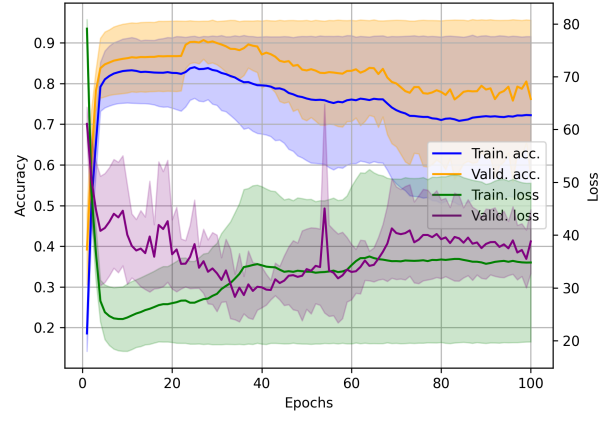
According to table 6.3, we can say that 21-C-SuGr-SHD failed. Indeed, the maximum test accuracies for RS and CASCBO are of about 80.79%, indicating that CASCBO is as good as random on SHD with PLIF-SpikingJelly. These results can reinforce our doubts about overfitting due to more parameters with PLIF-SpikingJelly. For experiments 13-C-SuGr-SHD and 21-C-SuGr-SHD, to explain lower performances, a hypothesis could be that the higher number of parameters decreases the impact of HPs, resulting in overfitting. Because RS does not try to maximize the validation accuracy, it can result in better generalization, explaining its better performances. Thus, it reinforces our previous discussions about the SHD dataset used in HPO. But, it is clear that the 42-C-SLAY-SHD was a success as the maximum test accuracy is equal to 92.62%, while 42-R-SLAY-SHD with RS obtained 89.84%. Nonetheless, compared to 21-C-SuGr-SHD, 42-C-SLAY-SHD and 42-R-SLAY-SHD have a much bigger search space. Such a high dimensionality makes the optimization by random sampling harder. We refer to chapter 1 and our discussions about the *curse of dimensionality*.

All other CASCBO experiments performed better than RS. The 46-C-SLAY-MNIST experiment obtained a minimum accuracy of 98.92%, and reaching up-to 99.20%, while 46-R-SLAY-MNIST obtained a maximum of 94.71% with a much noisier solution. Additionally, we observe an overfitting for the best solution returned by RS, as depicted in figure 6.7. The same goes for 46-C-SLAY-NMNIST with a minimum test accuracy of 98.47% and a maximum of 99.23%, while 46-R-SLAY-NMNIST obtained a maximum of 97.30. To better understand the comparisons, we refer to the previous sections discussing how hard it is for the optimization to improve a solution by 0.25%.

For PLIF-SpikingJelly, the differences between CASCBO and RS on MNIST and NMNIST, tend to be less pronounced. First because the training optimizes more parameters, and secondly because the search spaces have less HPs than LIF-SLAYER experiments. However, the results obtained by CASCBO are still better. The RS experiment 21-R-SuGr-MNIST obtained a maximum of 96.12%, while 21-C-SuGr-MNIST was able to achieve a minimum test accuracy of 99.36% with a maximum of 99.46%. We notice that the best solution found by RS is much noisier than the one found by CASCBO. We can explain that CASCBO finds



(a) 46-S-SLAY-MNIST.



(b) 46-R-SLAY-MNIST.

Figure 6.7: Learning curves summary of the best solutions from two experiments.

less noisy solutions because BO naturally handles noisy evaluations within the kernel (see chapter 3).

Concerning 21-R-SuGr-NMNIST, the maximum test accuracy is equal to 96.31%, while the maximum accuracy for the CASCBO counterpart 21-C-SuGr-NMNIST reaches 99.53%. For MNIST and NMNIST, we clearly observe these 0.25% validation accuracy gaps, which are also observed on the test accuracy. These gaps are also observed on SHD between 42-R-SLAY-SHD and 42-C-SLAY-SHD, illustrating the advantages of optimization compared to RS. Furthermore, we notice that most of the sampled solutions by RS are silent networks; the proportions of networks that should have been stopped range from 80% to 99.99%. Thus, we can question how the resources and budget are used by RS. This is illustrated by figures 6.8, 6.9, and 6.10.

For MNIST experiments, we clearly distinguish a focus on accuracies $> 90\%$ by CASCBO. While for RS, less than 5% of the budget is used for this accuracy bin. It is even worse for 21-R-SuGr-MNIST where at least 60% of the budget is spent on computing networks with accuracies $< 20\%$. We observe the same phenomenon in NMNIST experiments.

Concerning SHD experiments, it is less pronounced. For 21-C-SLAY-SHD we clearly observe that reaching accuracies $> 90\%$ is harder than on MNIST. For 21-R-SLAY-SHD, it is even worse; not a single sampled solution has a validation accuracy higher than 90%. We can clearly assess that accuracies $> 30\%$ are very uncommon. Furthermore, in 21-R-SLAY-SHD, we can say that having enough spiking activity is not enough to ensure high accuracy according to non-stopped networks. While for 46-R-SLAY-MNIST and 46-R-SuGr-NMNIST, a high spiking activity almost ensures accuracies $> 60\%$ with random HPs, as soon as this randomness generates output spikes. This confirms that for these datasets, the major challenge is to refine accuracies $> 99\%$. For 21-C-SuGr-SHD, we better understand the difficulties to obtain high accuracy solutions, but in 21-R-SuGr-SHD, we also notice how uncommon good SNNs are.

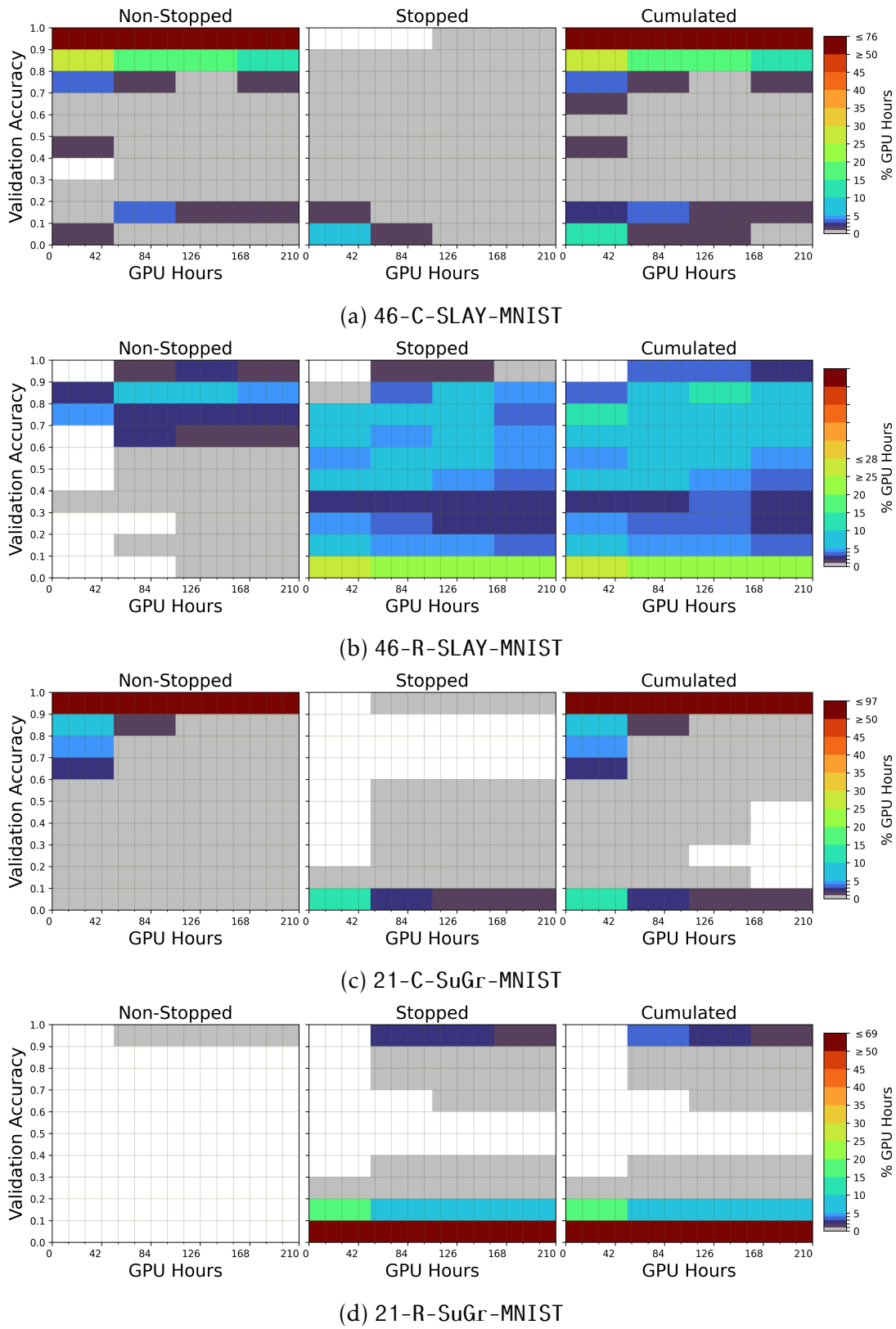


Figure 6.8: Allocation of the budget during multi-fidelity experiments on MNIST.

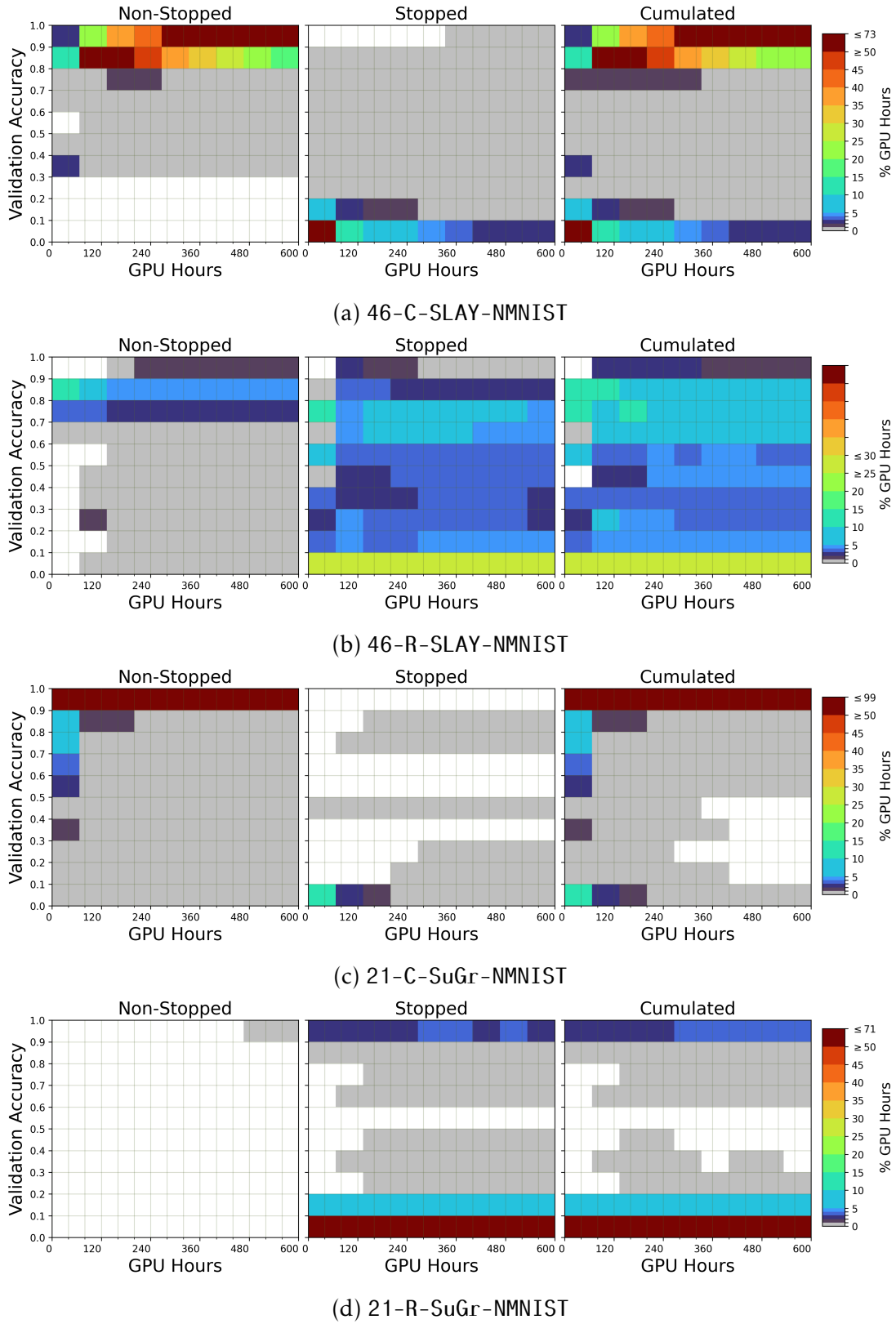


Figure 6.9: Allocation of the budget during multi-fidelity experiments on MNIST.

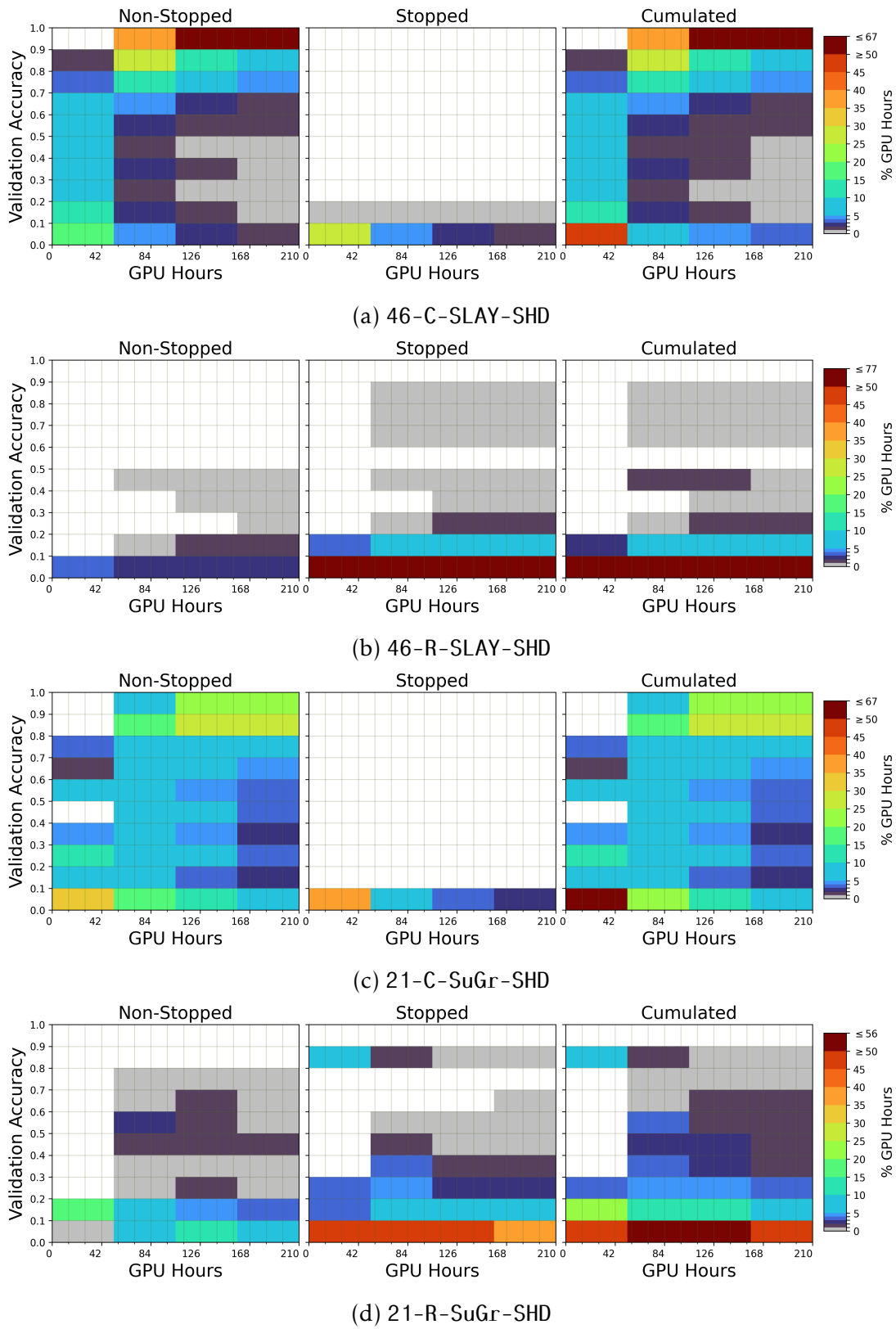


Figure 6.10: Allocation of the budget during multi-fidelity experiments on SHD.

6.4 Conclusion

In this chapter, we leveraged multi-fidelity optimization to handle HPs influencing the cost of training a SNN. The expensiveness of SNNs can be explained by diverse HPs, such as the temporal resolution of the simulator, the size of the network, or the size of the training dataset. We designed a new BO approach called CASCBO, based on SCBO [62], allowing to consider the cost of evaluations, constraints, and high dimensional search spaces. To improve the detection of *silent networks* and reduce their negative influence on the budget, we redefined specific constraints. These are based on the number of samples that output at least, and not at most, α spikes for a certain layer. This new definition follows a hypergeometric law, giving a stronger quantitative and probabilistic meaning to the constraints.

Experiments indicate that CASCBO can optimize high dimensional and constrained search spaces, including fidelity HPs. These HPs have an effect on the training time of SNNs, and so on the budget allocated to silent networks. The proposed methodology makes the impact of silent networks negligible, allowing to define a more general and flexible search space. The optimization process can quickly stabilize the high sensitivity of SNNs to their HPs before tackling expensive evaluations. Our approach empirically indicates that expensive evaluation on the full training dataset is not necessary to obtain high accuracy. Despite the complexity of the problem, the presented results remain competitive compared to their baseline accuracies, even better. On SHD, we demonstrate that we can considerably improve a default solution by doing some tuning. Moreover, compared to the PLIF-SpikingJelly approach, we also illustrate how sensitive the performances are regarding the simulator, neuron model, and training algorithm. While SpikingJelly obtained about 80% accuracy on SHD, SLAYER reached almost 93%. We illustrate that for most of the experiments, CASCBO performs better than RS. Moreover, because CASCBO handles noisy evaluations by design, some clues indicate that the algorithm can find more robust solutions than RS by selecting less noisy HPs combinations while obtaining better accuracies.

Additionally, when doing HPO and to prevent overfitting the HPs [120], a more rigorous approach is necessary. It includes train, validation, and test subsets, meaning that we train on fewer data compared to the baselines, which improves the generalizability of our solutions. Thus, further studies are necessary to tackle this overfitting, including better neuromorphic datasets for HPO and AutoML, including at least a train-validation-test split.

Finally, it is clear that compared to results from chapter 5, we can drastically accelerate the convergence by using multi-fidelity while dividing by 7 the necessary budget.

Main conclusion

This thesis explores hyperparameter optimization for spiking neural networks through both algorithmic and neuromorphic-ML approaches. The first three chapters of this manuscript give the rich and necessary background to understand both HPO and SNNs. We motivate the investigation on neuromorphic computing by the necessity to study other aspects of modern AI technologies. Among these other considerations that have been put aside for too long, the energy consumption of predictive models is becoming a crucial challenge, and SNNs could efficiently balance predictive performances and energy efficiency. We first described how usual synchronous ANNs work, before shifting toward the unique properties of SNNs. These peculiar models are closer to the biology. They mimic asynchronous neurons by replacing the activation function with a complex and highly hyperparameterized ordinary differential equation regarding time. According to Maass [170], SNNs use time as a resource for communications and computations. They replace floating-point information by timed events known as spikes. The dynamics within a SNN rely on the asynchronous transfer and the integration of these spikes between neurons. Consequently, considering SNNs similar to ANNs in the sense they are both blackboxes mimicking some sort of brain-like structure is a pitfall. This led to the Neurobench initiative [294], questioning the benchmarking of SNNs, and asking the following question: “*Can we fairly compare SNNs with ANNs on usual digital benchmarks?*”. While SNNs are known by their high sensitivity to their HPs and the difficulty of their tuning [217, 28, 102, 195], in the literature the HPO of SNNs does not differ from the HPO of ANNs. Although the HPO algorithm is selected a priori regarding the knowledge about ANNs, we forgot to clearly define the specificities of SNNs from the HPO standpoint. Therefore, we ended chapter 1 with a discussion on the No Free Lunch Theorem, implying the necessity to have prior knowledge about the problem before selecting an optimization algorithm. From these discussions, we coined the following research question:

How can we improve HPO of SNNs by investigating both the impact of HPO on the performances of SNNs, and the impact of SNNs on the performances of HPO?

In chapter 2, we propose the first work reviewing HPO of SNNs, tackled automatically or manually. We conclude that most of the works investigate very few HPs. The search spaces are limited, even discretized. The maximum search space size was of 18 HPs with an HPO of an architecture applied on an in-situ dataset. Moreover, the numerous HPs of the neuron model are rarely optimized altogether. Hence, our objective was to optimize more HPs from all five HPs groups: neurons model, learning rule, architecture, encoding–decoding, and HPs of the training pipeline. In chapter 3, we described the usual algorithms for HPO, ranging from metaheuristics to multi-fidelity Bayesian optimization. Then, the thesis contribution begins at chapter 4.

Thesis contributions

In chapter 4, we described a new family of algorithms named fractal-based decomposition algorithms. After reviewing the usual divide-and-conquer algorithms, we introduced a gen-

eral and flexible framework unifying three families of algorithms: DIRECT-based algorithms from the global optimization community, SOO from the multi-armed bandits family, and FDA from the metaheuristic community. A FBD algorithm is defined by five search components: fractal, tree search, scoring, and exploration. These components are translated into software bricks within Zellij. Zellij is a Python library helping to instantiate FBD algorithms and create new search components. We reproduced many popular FBD algorithms, including DIRECT and some of its other versions, SOO and its improvement NMSO, and FDA. We empirically demonstrate the behaviors of the search component on the COCO benchmark, and we focus on the scalability of these algorithms. Not only that, but we observed that while DIRECT and SOO are proper preservative algorithms, unbounded sacrificial algorithms like FDA can scale and reach top performances thanks to a local search counterbalancing the sacrificed space. Furthermore, we proposed to leverage LHS to partition the search space, and while the proposed algorithm can perform better than FDA in low and high dimensions, we observed that the algorithm struggles to scale. Further investigations, discussed in the future works sections, could help to improve this new highly sacrificial algorithm. Therefore, we also proposed a new highly sacrificial algorithm based on a Latin Hypercube decomposition. The algorithm presents some limitations but is better than FDA in low dimensions while being at least as efficient in higher dimensions.

In chapter 5 we tackled HPO of SNNs. The preliminary naive experiments indicate that during HPO many solutions have a very low validation accuracy, close to a random classification. We empirically demonstrate that there is a relationship between these bad accuracies and the spiking activity. We defined a new type of SNNs named *silent networks*. These networks output whether noise or no spikes. This lack of spiking activity can be explained by mistuned HPs. But, the usual blackbox workflow associated with the training of SNNs can learn to decode these noisy spikes or the absence of events by wrongly associating a class to *nothing*. We ended these discussions on preliminary experiments with some remarks and questions:

- *What is the impact of SNNs on HPO?*
- Is there a difference between a spiking SNN and a silent SNN, both with random accuracies?
- Concerning STDP learning, *if neurons do not fire, then synapses do not wire together.*

To leverage silent networks, we designed an indirect spike-based early stopping criterion to prevent worthless and costly computations. While stopping criteria are usually employed to prevent overfitting, this new one does not improve the generalizability of SNNs but aims at accelerating the convergence of HPO. Additionally to the early stopping, we have associated blackbox constraints to prevent the HPO from sampling silent networks with a high risk of being stopped. Therefore, the constraints force the HPO to sample SNNs for which the training can be carried out. Because of the early stopping, the evaluations of HP combinations become highly stochastic, which is a major challenge for parallelization of the HPO algorithm. Indeed, the training of a SNN can now last from a few tens of seconds to hours. Moreover, the problem is expensive, blackbox-constrained and high-dimensional. Hence, we have selected the SCBO algorithm that we have asynchronously parallelized on the GPU partition of the Jean Zay supercomputer. We performed long-run experiments with a budget of 1600 GPU hours, and we compared our approach to the synchronous TuRBO algorithm, the unconstrained version of SCBO.

The results strongly suggest that we can accelerate the convergence of about hundreds of GPU hours, while maintaining the performances. We clearly illustrate the negative impact of

silent networks on the optimization, where more than 50% of the budget can be spent on computing networks that could have been stopped with the early stopping. We have also illustrated that we can ensure a minimum accuracy as soon as there is enough spiking activity. But, depending on the dataset and simulator, the spiking activity does not solely explain the validation accuracy. As discussed in chapter 2, AutoML and HPO have their own scientific interests and benefits [20, 16, 123, 290, 19, 259] compared to manual tuning. Therefore, beyond usual AutoML, investigating HPO of SNNs and tailoring optimization algorithms to this peculiar problem allows to more efficiently design SNNs with a reduced cost. Chapter 5 is a step toward tailored HPO applied to SNNs. To improve HPO of SNNs by considering silent networks, a tradeoff must be made between the optimal allocation of resources and the potential loss of high-risk–high-reward solutions that may be misclassified as silent. Indeed, we noticed that our early stopping might wrongly detect silent networks, indicating that our approach can be improved.

In chapter 6, we investigate multi-fidelity HPO of SNNs. The experiments in chapter 5 required an enormous budget to converge. The bill for the electricity consumption of the experiments in chapter 5 reaches about 15000. In that regard, it is necessary to reduce the budget of HPO to make it affordable. We noticed in chapter 5 that some early stopped networks can have acceptable validation accuracies. Meaning that a full training is not necessary to retrieve some information about a HPs combination. Therefore, multi-fidelity HPO leverage low-cost – low-fidelity information to sample more efficiently high-cost – high-fidelity HPs combinations. But, because in this thesis SNNs are simulated, the fidelity can be explained by numerous HPs. For this reason, we adapted the asynchronous SCBO algorithm to a generalization of multi-fidelity known as cost-aware optimization. With this approach, we were able to reduce by 7 the budget necessary for convergence while maintaining competitive performances. With such a reduced budget, we performed 18 experiments with search spaces of up to 46 HPs, and we compared CASCBO to RS. We have empirically demonstrated that random HPs combinations in such high dimensional spaces generate mostly silent networks with low validation accuracies. We illustrate that HPs combinations with a minimum spiking activity are outliers, and that high accuracy SNNs in such high dimensional search spaces are uncommon.

Through chapters 5 and 6, we illustrate the major differences in terms of performances between simulators. We had to face some failures. We failed at obtaining and reproducing results on DvsGesture with an architecture trained by STDP on BindsNET. The necessary efforts to obtain suitable results on BindsNET with STDP to classify DvsGesture were such that we discarded this approach for the rest of the thesis. There is no doubt that modern surrogate gradient approaches outperform STDP in terms of performances, but also ease of design. However, local learning and neuromorphic-hardware-friendly learning rules should be deeper investigated; SNNs are unique models not only by their neurons, but also by their learning rules, clearly different from ANNs. In chapter 6, we have illustrated on the SHD dataset that according to the simulator and the approach, we can obtain entirely different performances. While the PLIF-SpikingJelly trains the neuron leakage, we observed overfitting and a clear loss of accuracy compared to the LIF-SLAYER approach.

Lastly, SNNs are clearly sensitive to their HPs, which can generate infeasible solutions. But SNNs are also sensitive to the simulator, their instantiation, and to the dataset. Before discarding a new approach, learning rule, neuron model, or simulator having bad performances, maybe it requires a prior HPO to identify a suitable baseline. Moreover, comparing two SNNs approaches without the same level of HPO is unfair. On SHD, we have demonstrated that with a HPO of 14 hours, we can improve a baseline architecture by almost 20% test accuracy. Thus, stating that a new SNNs approach is better than another, while they did not receive the

same efforts in terms of design, is misleading regarding SNNs sensitivity.

HPO can improve significantly the predictive performances of SNNs, and silent SNNs have a significant impact on the HPO.

Future works

Because fractals from FBD algorithms are fully independent subsets of the original search space, we claim that FBD are a serious candidate for massive parallelization within distributed Peta- or Exa-scale architectures. Thus, future work should focus on finding common parallelization techniques for FBD algorithms. Some early works [191, 133, 256, 66] have already investigated limited parallelization of some FBD algorithms, but it is now necessary to scale these on modern large-scale distributed architectures. Investigating sacrifice-based approaches is also of interest, and could help improve FBD algorithms regarding convergence time or memory limitation.

In the long term, the automatic design of FBD algorithms could be of interest. Indeed, thanks to the modularity of Zellij, we could tailor FBD algorithms to specific tasks, notably by using hyper-heuristic approaches to find the best combination of search components [254]. Such approaches were already applied to population-based algorithms [301]. Adapting FBD algorithms to HPO is challenging. State-of-the-art HPO algorithms are mostly from the SMBO family. Some works [177, 283] combine both FBD and BO algorithms, showing a decrease in the number of necessary evaluations to converge. Hence, leveraging these BO-FBD algorithms could help design for expensive HPO. Investigating such an algorithmic paradigm could also enable new methods for parallelizing Bayesian optimization. Therefore, tackling HPO of SNNs with FBD algorithms first requires further investigations within two research axes:

- Combining BO and FBD algorithms, such as with the BaMSOO algorithm [283]. Such a combination could help BO scales in dimension. Furthermore, it could also open doors to new ways of parallelizing BO. Indeed, as discussed in chapter 3, one has to face major challenges to synchronously or asynchronously parallelize BO.
- Large-scale parallelization of FBD algorithms on Exascale architectures. Some properties of a FBD approach, such as the independence and inheritance mechanism of fractals, could help design new optimization approaches while considering by design its parallelization.

Concerning HPO of SNN. Future works will focus on multi-objective optimization, where finding the frontier between low-spiking and silent networks is crucial. Indeed, the described constraints could be considered secondary objectives to be minimized. Applying our methodology to cost-aware multi-objective HPO of SNNs where minimizing the number of spikes is important while maintaining a minimum spiking activity. Finally, the strategy could also be considered within a NAS framework. Only a few works have tackled this problem applied to SNNs [190, 146]. Indeed, designing feasible solutions is difficult. Hence, being able to quickly determine whether a given architecture is silent or not could improve the optimization process.

Future works could also focus on improving the constraints. Indeed, in chapters 5 and 6 we observed that some SNNs are misclassified as silent networks, while having a high accuracy. Occasionally, because the gradient can force the network to have a spiking activity, the constraint could implement a patience, waiting to see if the spiking activity

is increasing, thus preventing SNNs misclassified as silent. It could also be interesting to study the impact on the learning curves of these misclassified silent networks. We noticed that some of these SNNs converge slower. Our hypothesis is that the gradient has to first compensate for the lack of spiking activity before focusing on refining the weights regarding the loss. We could also combine the evolution of the spiking activity with HPO based on learning curve interpolation [54]. Moreover, investigating a Meta-learning [222] approach leveraging constraints and early stopping could help to determine common non-silent HPs combinations or architectures across tasks and problems.

Finally, investing on hybrid CPU-GPU-Neuromorphic clusters could definitely help to do on-hardware HPO of SNNs, and go beyond with hardware-aware HPO.

Acknowledgments

Experiments presented in this work were carried out using the Grid’5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see <https://www.grid5000.fr>).

This work was granted access to the HPC resources of IDRIS under the allocation 2023-AD011014347 made by GENCI.

This work has been supported by the University of Lille, the ANR-20-THIA-0014 program AI_PhD@Lille and the ANR PEPR AI and Numpex. It was also supported by IRCICA(CNRS and Univ. Lille USR-3380).

Code availability

The code of this thesis is available at <https://gitlab.cristal.univ-lille.fr/tfirmin/mythesis>.

Bibliography

- [1] Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. “On the Surprising Behavior of Distance Metrics in High Dimensional Space”. en. *Database Theory ICDT 2001*. Ed. by Gerhard Goos et al. Vol. 1973. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 420–434. ISBN: 978-3-540-41456-8 978-3-540-44503-6. DOI: [10.1007/3-540-44503-X_27](https://doi.org/10.1007/3-540-44503-X_27). URL: http://link.springer.com/10.1007/3-540-44503-X_27 (visited on 08/12/2022).
- [2] Hedyeh Aghabarar, Kourosh Kiani, and Parviz Keshavarzi. “Improvement of pattern recognition in spiking neural networks by modifying threshold parameter and using image inversion”. en. *Multimedia Tools and Applications* (July 2023). ISSN: 1380-7501, 1573-7721. DOI: [10.1007/s11042-023-16344-3](https://doi.org/10.1007/s11042-023-16344-3). URL: <https://link.springer.com/10.1007/s11042-023-16344-3> (visited on 08/31/2023).
- [3] Takuya Akiba et al. “Optuna: A Next-generation Hyperparameter Optimization Framework”. en. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. Anchorage AK USA: ACM, July 2019, pp. 2623–2631. ISBN: 978-1-4503-6201-6. DOI: [10.1145/3292500.3330701](https://doi.org/10.1145/3292500.3330701). URL: <https://dl.acm.org/doi/10.1145/3292500.3330701> (visited on 08/22/2024).
- [4] Filipp Akopyan et al. “Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip”. *IEEE transactions on computer-aided design of integrated circuits and systems* 34.10 (2015). Publisher: IEEE, pp. 1537–1557.
- [5] Feryal Alayont and Nicholas Krzywonos. “Rook polynomials in three and higher dimensions”. en. *Involve, a Journal of Mathematics* 6.1 (June 2013), pp. 35–52. ISSN: 1944-4184, 1944-4176. DOI: [10.2140/involve.2013.6.35](https://doi.org/10.2140/involve.2013.6.35). URL: <http://msp.org/involve/2013/6-1/p04.xhtml> (visited on 07/10/2024).
- [6] Enrique Alba. *Parallel metaheuristics: a new class of algorithms*. John Wiley & Sons, 2005.
- [7] Enrique Alba, Gabriel Luque, and Sergio Nesmachnow. “Parallel metaheuristics: recent advances and new trends”. en. *International Transactions in Operational Research* 20.1 (Jan. 2013), pp. 1–48. ISSN: 0969-6016, 1475-3995. DOI: [10.1111/j.1475-3995.2012.00862.x](https://doi.org/10.1111/j.1475-3995.2012.00862.x). URL: <https://onlinelibrary.wiley.com/doi/10.1111/j.1475-3995.2012.00862.x> (visited on 09/02/2024).
- [8] Enrique Alba and José M. Troya. “Analyzing synchronous and asynchronous parallel distributed genetic algorithms”. en. *Future Generation Computer Systems* 17.4 (Jan. 2001), pp. 451–465. ISSN: 0167739X. DOI: [10.1016/S0167-739X\(99\)00129-6](https://doi.org/10.1016/S0167-739X(99)00129-6). URL: <https://linkinghub.elsevier.com/retrieve/pii/S0167739X99001296> (visited on 02/21/2023).

- [9] Hussain Alibrahim and Simone A. Ludwig. “Hyperparameter Optimization: Comparing Genetic Algorithm against Grid Search and Bayesian Optimization”. en. *2021 IEEE Congress on Evolutionary Computation (CEC)*. Kraków, Poland: IEEE, June 2021, pp. 1551–1559. ISBN: 978-1-72818-393-0. DOI: [10.1109/CEC45853.2021.9504761](https://doi.org/10.1109/CEC45853.2021.9504761). URL: <https://ieeexplore.ieee.org/document/9504761/> (visited on 09/26/2024).
- [10] Arnon Amir et al. “A Low Power, Fully Event-Based Gesture Recognition System”. en. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI: IEEE, July 2017, pp. 7388–7397. ISBN: 978-1-5386-0457-1. DOI: [10.1109/CVPR.2017.781](https://doi.org/10.1109/CVPR.2017.781). URL: <https://ieeexplore.ieee.org/document/8100264/> (visited on 08/30/2023).
- [11] Daniel Auge et al. “A Survey of Encoding Techniques for Signal Processing in Spiking Neural Networks”. en. *Neural Processing Letters* (July 2021). ISSN: 1370-4621, 1573-773X. DOI: [10.1007/s11063-021-10562-2](https://doi.org/10.1007/s11063-021-10562-2). URL: <https://link.springer.com/10.1007/s11063-021-10562-2> (visited on 10/13/2021).
- [12] Maximilian Balandat et al. “BOTORCH: a framework for efficient monte-carlo Bayesian optimization”. *Proceedings of the 34th International Conference on Neural Information Processing Systems. NIPS’20*. Vancouver, BC, Canada: Curran Associates Inc., 2020, p. 15. ISBN: 9781713829546.
- [13] Daniel Balouek et al. “Adding Virtualization Capabilities to Grid’5000”. en ().
- [14] C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. “The Quickhull Algorithm for Convex Hulls”. *ACM Trans. Math. Softw.* 22.4 (Dec. 1996), pp. 469–483. ISSN: 0098-3500. DOI: [10.1145/235815.235821](https://doi.org/10.1145/235815.235821). URL: <https://doi.org/10.1145/235815.235821>.
- [15] Maximilian Baronig et al. *Advancing Spatio-Temporal Processing in Spiking Neural Networks through Adaptation*. en. arXiv:2408.07517 [cs]. Aug. 2024. URL: <http://arxiv.org/abs/2408.07517> (visited on 08/22/2024).
- [16] Eva Bartz et al., eds. *Hyperparameter Tuning for Machine and Deep Learning with R: A Practical Guide*. en. Citation KKey: hpobook_2023. Singapore: Springer Nature Singapore, 2023. ISBN: 978-981-19516-9-5 978-981-19517-0-1. DOI: [10.1007/978-981-19-5170-1](https://doi.org/10.1007/978-981-19-5170-1). URL: <https://link.springer.com/10.1007/978-981-19-5170-1> (visited on 04/18/2024).
- [17] Felix C. Bauer et al. “EXODUS: Stable and efficient training of spiking neural networks”. en. *Frontiers in Neuroscience* 17 (Feb. 2023), p. 1110444. ISSN: 1662-453X. DOI: [10.3389/fnins.2023.1110444](https://doi.org/10.3389/fnins.2023.1110444). URL: <https://www.frontiersin.org/articles/10.3389/fnins.2023.1110444/full> (visited on 03/26/2024).
- [18] Trevor Bekolay et al. “Nengo: a Python tool for building large-scale functional brain models”. en. *Frontiers in Neuroinformatics* 7 (2014). ISSN: 1662-5196. DOI: [10.3389/fninf.2013.00048](https://doi.org/10.3389/fninf.2013.00048). URL: <http://journal.frontiersin.org/article/10.3389/fninf.2013.00048/abstract> (visited on 09/30/2024).
- [19] James Bergstra and Yoshua Bengio. “Random Search for Hyper-Parameter Optimization”. en ().
- [20] Bernd Bischl et al. “Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges”. en. *WIREs Data Mining and Knowledge Discovery* 13.2 (Mar. 2023), e1484. ISSN: 1942-4787, 1942-4795. DOI: [10.1002/widm.1484](https://doi.org/10.1002/widm.1484). URL: <https://wires.onlinelibrary.wiley.com/doi/10.1002/widm.1484> (visited on 10/03/2023).

- [21] Alexandre Bittar and Philip N. Garner. *Exploring neural oscillations during speech perception via surrogate gradient spiking neural networks*. en. arXiv:2404.14024 [cs, q-bio]. Apr. 2024. URL: <http://arxiv.org/abs/2404.14024> (visited on 08/22/2024).
- [22] Hannah Bos and Dylan Muir. *Sub-mW Neuromorphic SNN audio processing applications with Rockpool and Xylo*. en. arXiv:2208.12991 [cs, eess]. Sept. 2022. URL: <http://arxiv.org/abs/2208.12991> (visited on 09/29/2024).
- [23] Léon Bottou. “Large-scale machine learning with stochastic gradient descent”. *Proceedings of COMPSTAT’2010: 19th International Conference on Computational Statistics Paris France, August 22-27, 2010 Keynote, Invited and Contributed Papers*. Springer, 2010, pp. 177–186.
- [24] Pierre Boulet et al. “N2s3, an open-source scalable spiking neuromorphic hardware simulator”. PhD Thesis. Université de Lille 1, Sciences et Technologies; CRISTAL UMR 9189, 2017.
- [25] Maxence Bouvier et al. “Spiking Neural Networks Hardware Implementations and Challenges: A Survey”. en. *ACM Journal on Emerging Technologies in Computing Systems* 15.2 (June 2019), pp. 1–35. ISSN: 1550-4832, 1550-4840. DOI: [10.1145/3304103](https://doi.org/10.1145/3304103). URL: <https://dl.acm.org/doi/10.1145/3304103> (visited on 10/13/2021).
- [26] Romain Brette. “Philosophy of the Spike: Rate-Based vs. Spike-Based Theories of the Brain”. en. *Frontiers in Systems Neuroscience* 9 (Nov. 2015). ISSN: 1662-5137. DOI: [10.3389/fnsys.2015.00151](https://doi.org/10.3389/fnsys.2015.00151). URL: <http://journal.frontiersin.org/Article/10.3389/fnsys.2015.00151/abstract> (visited on 12/06/2021).
- [27] S. Cahon, N. Melab, and E.-G. Talbi. “ParadisEO: A Framework for the Reusable Design of Parallel and Distributed Metaheuristics”. en. *Journal of Heuristics* 10.3 (May 2004), pp. 357–380. ISSN: 1381-1231. DOI: [10.1023/B:HEUR.0000026900.92269.ec](https://doi.org/10.1023/B:HEUR.0000026900.92269.ec). URL: <http://link.springer.com/10.1023/B:HEUR.0000026900.92269.ec> (visited on 10/06/2021).
- [28] Biswadeep Chakraborty et al. “Brain-Inspired Spatiotemporal Processing Algorithms for Efficient Event-Based Perception”. en. *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Antwerp, Belgium: IEEE, Apr. 2023, pp. 1–6. DOI: [10.23919/DAT56975.2023.10136914](https://doi.org/10.23919/DAT56975.2023.10136914). URL: <https://ieeexplore.ieee.org/document/10136914/> (visited on 09/07/2023).
- [29] Kaiwei Che et al. “Differentiable hierarchical and surrogate gradient search for spiking neural networks”. en ().
- [30] Yuansi Chen et al. “Fast MCMC Sampling Algorithms on Polytopes”. *J. Mach. Learn. Res.* 19.1 (Jan. 2018), pp. 2146–2231. ISSN: 1532-4435. DOI: [10.5555/3291125.3309617](https://doi.org/10.5555/3291125.3309617).
- [31] Alexey Chernyshev. “Bayesian Optimization of Spiking Neural Network Parameters to Solving the Time Series Classification Task”. *Biologically Inspired Cognitive Architectures (BICA) for Young Scientists*. Ed. by Alexei V. Samsonovich, Valentin V. Klimov, and Galina V. Rybina. Cham: Springer International Publishing, 2016, pp. 39–45. ISBN: 978-3-319-32554-5.

- [32] Clément Chevalier and David Ginsbourger. “Fast Computation of the Multi-Points Expected Improvement with Applications in Batch Selection”. en. *Learning and Intelligent Optimization*. Ed. by Giuseppe Nicosia and Panos Pardalos. Vol. 7997. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 59–69. ISBN: 978-3-642-44972-7 978-3-642-44973-4. DOI: [10.1007/978-3-642-44973-4_7](https://doi.org/10.1007/978-3-642-44973-4_7). URL: http://link.springer.com/10.1007/978-3-642-44973-4_7 (visited on 08/22/2022).
- [33] Sayeed Shafayet Chowdhury, Chankyu Lee, and Kaushik Roy. “Towards understanding the effect of leak in Spiking Neural Networks”. en. *Neurocomputing* 464 (Nov. 2021), pp. 83–94. ISSN: 09252312. DOI: [10.1016/j.neucom.2021.07.091](https://doi.org/10.1016/j.neucom.2021.07.091). URL: <https://linkinghub.elsevier.com/retrieve/pii/S0925231221011760> (visited on 11/02/2021).
- [34] Loïc Cordone. “Performance of spiking neural networks on event data for embedded automotive applications”. fr ().
- [35] Loïc Cordone, Benoît Miramond, and Sonia Ferrante. *Learning from Event Cameras with Sparse Spiking Convolutional Neural Networks*. en. arXiv:2104.12579 [cs]. Apr. 2021. URL: <http://arxiv.org/abs/2104.12579> (visited on 05/11/2023).
- [36] Mario Vazquez Corte and Luis V. Montiel. “Novel matrix hit and run for sampling polytopes and its GPU implementation”. *Computational Statistics* (Sept. 2023). DOI: [10.1007/s00180-023-01411-y](https://doi.org/10.1007/s00180-023-01411-y). URL: <https://doi.org/10.1007/s00180-023-01411-y>.
- [37] B. Cramer et al. “The Heidelberg Spiking Data Sets for the Systematic Evaluation of Spiking Neural Networks”. *IEEE Transactions on Neural Networks and Learning Systems* (2020), pp. 1–14. ISSN: 2162-2388. DOI: [10.1109/TNNLS.2020.3044364](https://doi.org/10.1109/TNNLS.2020.3044364).
- [38] Benjamin Cramer et al. “Surrogate gradients for analog neuromorphic computing”. en. *Proceedings of the National Academy of Sciences* 119.4 (Jan. 2022), e2109194119. ISSN: 0027-8424, 1091-6490. DOI: [10.1073/pnas.2109194119](https://doi.org/10.1073/pnas.2109194119). URL: <https://pnas.org/doi/full/10.1073/pnas.2109194119> (visited on 10/13/2023).
- [39] Benjamin Cramer et al. “The Heidelberg Spiking Data Sets for the Systematic Evaluation of Spiking Neural Networks”. en. *IEEE Transactions on Neural Networks and Learning Systems* 33.7 (July 2022). Citation KKey: shd, pp. 2744–2757. ISSN: 2162-237X, 2162-2388. DOI: [10.1109/TNNLS.2020.3044364](https://doi.org/10.1109/TNNLS.2020.3044364). URL: <https://ieeexplore.ieee.org/document/9311226/> (visited on 04/23/2024).
- [40] Biswa Nath Datta. *Numerical linear algebra and applications*. SIAM, 2010.
- [41] Mike Davies et al. “Loihi: A Neuromorphic Manycore Processor with On-Chip Learning”. en. *IEEE Micro* 38.1 (Jan. 2018), pp. 82–99. ISSN: 0272-1732, 1937-4143. DOI: [10.1109/MM.2018.112130359](https://doi.org/10.1109/MM.2018.112130359). URL: <https://ieeexplore.ieee.org/document/8259423/> (visited on 11/13/2023).
- [42] Mike Davies et al. “Advancing Neuromorphic Computing With Loihi: A Survey of Results and Outlook”. en. *Proceedings of the IEEE* 109.5 (May 2021), pp. 911–934. ISSN: 0018-9219, 1558-2256. DOI: [10.1109/JPROC.2021.3067593](https://doi.org/10.1109/JPROC.2021.3067593). URL: <https://ieeexplore.ieee.org/document/9395703/> (visited on 08/23/2024).
- [43] Andrew P Davison et al. “PyNN: a common interface for neuronal network simulators”. *Frontiers in neuroinformatics* 2 (2009), p. 388.

- [44] Erick G. G. De Paz et al. "A Regression Tree as Acquisition Function for Low-Dimensional Optimisation". en. *Pattern Recognition*. Ed. by Efrén Mezura-Montes et al. Vol. 14755. Series Title: Lecture Notes in Computer Science. Cham: Springer Nature Switzerland, 2024, pp. 23–33. ISBN: 978-3-031-62835-1 978-3-031-62836-8. DOI: [10.1007/978-3-031-62836-8_3](https://doi.org/10.1007/978-3-031-62836-8_3). URL: https://link.springer.com/10.1007/978-3-031-62836-8_3 (visited on 07/10/2024).
- [45] Rina Dechter and Judea Pearl. "Generalized Best-First Search Strategies and the Optimality of A*". *J. ACM* 32.3 (July 1985), pp. 505–536. ISSN: 0004-5411. DOI: [10.1145/3828.3830](https://doi.org/10.1145/3828.3830). URL: <https://doi.org/10.1145/3828.3830>.
- [46] Melek et al Demirhan. "FRACTOP: A Geometric Partitioning Metaheuristic for Global Optimization". en. *Journal of Global Optimization* 14.4 (1999), pp. 415–436. ISSN: 09255001. DOI: [10.1023/A:1008384329041](https://doi.org/10.1023/A:1008384329041). URL: <http://link.springer.com/10.1023/A:1008384329041> (visited on 03/01/2022).
- [47] Lei Deng, Huajin Tang, and Kaushik Roy. "Editorial: Understanding and Bridging the Gap Between Neuromorphic Computing and Machine Learning". en. *Frontiers in Computational Neuroscience* 15 (Mar. 2021), p. 665662. ISSN: 1662-5188. DOI: [10.3389/fncom.2021.665662](https://doi.org/10.3389/fncom.2021.665662). URL: <https://www.frontiersin.org/articles/10.3389/fncom.2021.665662/full> (visited on 09/23/2021).
- [48] R H Dennard et al. "Design of ion-implanted MOSFET's with very small physical dimensions". en (). Citation ey: dennard.
- [49] "Depth-first iterative-deepening: An optimal admissible tree search". en. *Artificial Intelligence* 28.1 (Feb. 1986), p. 123. ISSN: 00043702. DOI: [10.1016/0004-3702\(86\)90035-4](https://doi.org/10.1016/0004-3702(86)90035-4). URL: <https://linkinghub.elsevier.com/retrieve/pii/S0004370286900354> (visited on 07/25/2024).
- [50] Bilel Derbel and Philippe Preux. "Simultaneous optimistic optimization on the noiseless BBOB testbed". en. *2015 IEEE Congress on Evolutionary Computation (CEC)*. Sendai, Japan: IEEE, May 2015, pp. 2010–2017. ISBN: 978-1-4799-7492-4. DOI: [10.1109/CEC.2015.7257132](https://doi.org/10.1109/CEC.2015.7257132). URL: <http://ieeexplore.ieee.org/document/7257132/> (visited on 05/22/2024).
- [51] Peter U. Diehl and Matthew Cook. "Unsupervised learning of digit recognition using spike-timing-dependent plasticity". en. *Frontiers in Computational Neuroscience* 9 (Aug. 2015). ISSN: 1662-5188. DOI: [10.3389/fncom.2015.00099](https://doi.org/10.3389/fncom.2015.00099). URL: <http://journal.frontiersin.org/Article/10.3389/fncom.2015.00099/abstract> (visited on 10/13/2021).
- [52] Mihaela Dimovska et al. "Multi-Objective Optimization for Size and Resilience of Spiking Neural Networks". en. *2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*. New York City, NY, USA: IEEE, Oct. 2019, pp. 0433–0439. ISBN: 978-1-72813-885-5. DOI: [10.1109/UEMCON47517.2019.8992983](https://doi.org/10.1109/UEMCON47517.2019.8992983). URL: <https://ieeexplore.ieee.org/document/8992983/> (visited on 09/23/2021).
- [53] James Dinan et al. "Scalable work stealing". en. *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. Portland Oregon: ACM, Nov. 2009, pp. 1–11. ISBN: 978-1-60558-744-8. DOI: [10.1145/1654059.1654113](https://doi.org/10.1145/1654059.1654113). URL: <https://dl.acm.org/doi/10.1145/1654059.1654113> (visited on 01/31/2023).
- [54] Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. "Speeding up Automatic Hyperparameter Optimization of Deep Neural Networks by Extrapolation of Learning Curves". en ().

- [55] Yiting Dong et al. “An unsupervised STDP-based spiking neural network inspired by biologically plausible learning rules and connections”. en. *Neural Networks* 165 (Aug. 2023), pp. 799–808. ISSN: 08936080. DOI: [10.1016/j.neunet.2023.06.019](https://doi.org/10.1016/j.neunet.2023.06.019). URL: <https://linkinghub.elsevier.com/retrieve/pii/S0893608023003301> (visited on 10/17/2023).
- [56] Michael Drmota and Robert F. Tichy. *Sequences, Discrepancies and Applications*. Berlin Heidelberg: Springer, 1997. DOI: [10.1007/bfb0093404](https://doi.org/10.1007/bfb0093404).
- [57] Abdullah Al-Dujaili and S. Suresh. “A Naive multi-scale search algorithm for global optimization problems”. en. *Information Sciences* 372 (Dec. 2016). Citaaaaation Key: nmso, pp. 294–312. ISSN: 00200255. DOI: [10.1016/j.ins.2016.07.054](https://doi.org/10.1016/j.ins.2016.07.054). URL: <https://linkinghub.elsevier.com/retrieve/pii/S0020025516305370> (visited on 05/05/2024).
- [58] Abdullah Al-Dujaili, S. Suresh, and N. Sundararajan. “MSO: a framework for bound-constrained black-box global optimization algorithms”. en. *Journal of Global Optimization* 66.4 (Dec. 2016), pp. 811–845. ISSN: 0925-5001, 1573-2916. DOI: [10.1007/s10898-016-0441-5](https://doi.org/10.1007/s10898-016-0441-5). URL: <http://link.springer.com/10.1007/s10898-016-0441-5> (visited on 03/22/2024).
- [59] Katharina Eggersperger et al. *HPOBench: A Collection of Reproducible Multi-Fidelity Benchmark Problems for HPO*. en. arXiv:2109.06716 [cs]. Oct. 2022. URL: <http://arxiv.org/abs/2109.06716> (visited on 03/10/2023).
- [60] Hammouda Elbez et al. “Progressive compression and weight reinforcement for spiking neural networks”. en. *Concurrency and Computation: Practice and Experience* 34.11 (May 2022), e6891. ISSN: 1532-0626, 1532-0634. DOI: [10.1002/cpe.6891](https://doi.org/10.1002/cpe.6891). URL: <https://onlinelibrary.wiley.com/doi/10.1002/cpe.6891> (visited on 10/18/2023).
- [61] Daniel Elbrecht and Catherine Schuman. “Neuroevolution of Spiking Neural Networks Using Compositional Pattern Producing Networks”. en. *International Conference on Neuromorphic Systems 2020*. Oak Ridge TN USA: ACM, July 2020, pp. 1–5. ISBN: 978-1-4503-8851-1. DOI: [10.1145/3407197.3407198](https://doi.org/10.1145/3407197.3407198). URL: <https://dl.acm.org/doi/10.1145/3407197.3407198> (visited on 09/04/2024).
- [62] David Eriksson and Matthias Poloczek. *Scalable Constrained Bayesian Optimization*. en. arXiv:2002.08526 [cs, stat]. Feb. 2021. URL: <http://arxiv.org/abs/2002.08526> (visited on 10/20/2023).
- [63] David Eriksson et al. *Scalable Global Optimization via Local Bayesian Optimization*. en. arXiv:1910.01739 [cs, stat]. Feb. 2020. URL: <http://arxiv.org/abs/1910.01739> (visited on 10/20/2023).
- [64] Jason K. Eshraghian et al. “Training Spiking Neural Networks Using Lessons From Deep Learning”. en. *Proceedings of the IEEE* 111.9 (Sept. 2023), pp. 1016–1054. ISSN: 0018-9219, 1558-2256. DOI: [10.1109/JPROC.2023.3308088](https://doi.org/10.1109/JPROC.2023.3308088). URL: <https://ieeexplore.ieee.org/document/10242251/> (visited on 09/30/2024).
- [65] Shimon Even. *Graph Algorithms*. Ed. by Guy Editor Even. 2nd ed. Cambridge: Cambridge University Press, 2011. DOI: [10.1017/CB09781139015165](https://doi.org/10.1017/CB09781139015165).
- [66] Yuri Evtushenko, Mikhail Posypkin, and Israel Sigal. “A framework for parallel large-scale global optimization”. en. *Computer Science - Research and Development* 23.3-4 (June 2009), pp. 211–215. ISSN: 1865-2034, 1865-2042. DOI: [10.1007/s00450-009-0083-7](https://doi.org/10.1007/s00450-009-0083-7). URL: <http://link.springer.com/10.1007/s00450-009-0083-7> (visited on 05/05/2024).

- [67] Pierre Falez. “Improving Spiking Neural Networks Trained with Spike Timing Dependent Plasticity for Image Recognition”. en ().
- [68] Pierre Falez et al. “Unsupervised visual feature learning with spike-timing-dependent plasticity: How far are we from traditional feature learning approaches?” en. *Pattern Recognition* 93 (Sept. 2019), pp. 418–429. ISSN: 00313203. DOI: [10.1016/j.patcog.2019.04.016](https://doi.org/10.1016/j.patcog.2019.04.016). URL: <https://linkinghub.elsevier.com/retrieve/pii/S0031320319301621> (visited on 09/07/2023).
- [69] Stefan Falkner, Aaron Klein, and Frank Hutter. *BOHB: Robust and Efficient Hyperparameter Optimization at Scale*. en. arXiv:1807.01774 [cs, stat]. July 2018. URL: <http://arxiv.org/abs/1807.01774> (visited on 09/01/2023).
- [70] Wei Fang et al. *Incorporating Learnable Membrane Time Constant to Enhance Learning of Spiking Neural Networks*. en. arXiv:2007.05785 [cs]. Aug. 2021. URL: <http://arxiv.org/abs/2007.05785> (visited on 05/28/2024).
- [71] Wei Fang et al. *Incorporating Learnable Membrane Time Constant to Enhance Learning of Spiking Neural Networks*. en. arXiv:2007.05785 [cs]. Aug. 2021. URL: <http://arxiv.org/abs/2007.05785> (visited on 09/01/2023).
- [72] Wei Fang et al. *SpikingJelly: An open-source machine learning infrastructure platform for spike-based intelligence*. en. arXiv:2310.16620 [cs]. Oct. 2023. URL: <http://arxiv.org/abs/2310.16620> (visited on 05/28/2024).
- [73] D E Finkel and C T Kelley. “An Adaptive Restart Implementation of DIRECT”. en (), p. 16. URL: <https://repository.lib.ncsu.edu/bitstream/handle/1840.4/461/crsc-tr04-30.pdf?sequence=1>.
- [74] Daniel E. Finkel. “Direct optimization algorithm user guide”. 2003. URL: <https://api.semanticscholar.org/CorpusID:6899005>.
- [75] Matteo Fischetti and Matteo Stringher. *Embedded hyper-parameter tuning by Simulated Annealing*. en. arXiv:1906.01504 [cs, math, stat]. June 2019. URL: <http://arxiv.org/abs/1906.01504> (visited on 09/05/2024).
- [76] Roger Fletcher. *Practical methods of optimization*. John Wiley & Sons, 2000.
- [77] Dario Floreano, Peter Dürri, and Claudio Mattiussi. “Neuroevolution: from architectures to learning”. en. *Evolutionary Intelligence* 1.1 (Mar. 2008), pp. 47–62. ISSN: 1864-5909, 1864-5917. DOI: [10.1007/s12065-007-0002-4](https://doi.org/10.1007/s12065-007-0002-4). URL: <http://link.springer.com/10.1007/s12065-007-0002-4> (visited on 09/04/2024).
- [78] Jose Pablo Folch et al. *Combining Multi-Fidelity Modelling and Asynchronous Batch Bayesian Optimization*. en. arXiv:2211.06149 [cs, stat]. Feb. 2023. URL: <http://arxiv.org/abs/2211.06149> (visited on 03/08/2024).
- [79] Steven Fortune. “A sweepline algorithm for Voronoi diagrams”. en (), p. 22. DOI: [10.1007/BF01840357](https://doi.org/10.1007/BF01840357).
- [80] Nikolaus Frohner et al. “Parallel Beam Search for Combinatorial Optimization (Extended Abstract)”. en. *Proceedings of the International Symposium on Combinatorial Search* 15.1 (July 2022), pp. 273–275. ISSN: 2832-9163, 2832-9171. DOI: [10.1609/socs.v15i1.21783](https://doi.org/10.1609/socs.v15i1.21783). URL: <https://ojs.aaai.org/index.php/SOCS/article/view/21783> (visited on 01/31/2023).
- [81] J M Gablonsky and C T Kelley. “A Locally-Biased form of the DIRECT Algorithm”. en (), p. 12. DOI: [10.1023/A:1017930332101](https://doi.org/10.1023/A:1017930332101).

- [82] Jacob R. Gardner et al. *GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration*. en. arXiv:1809.11165 [cs, stat]. June 2021. URL: <http://arxiv.org/abs/1809.11165> (visited on 09/13/2024).
- [83] Roman Garnett. *Bayesian Optimization*. Cambridge: Cambridge University Press, 2023. DOI: [10.1017/9781108348973](https://doi.org/10.1017/9781108348973).
- [84] Roman Garnett. *Bayesian optimization*. Cambridge University Press, 2023.
- [85] John Garofolo et al. *TIMIT Acoustic-Phonetic Continuous Speech Corpus*. Version V1. 1993. DOI: [11272.1/AB2/SWVENO](https://doi.org/10.11272.1/AB2/SWVENO). URL: <https://hdl.handle.net/11272.1/AB2/SWVENO>.
- [86] Cunjing Ge and Feifei Ma. “A Fast and Practical Method to Estimate Volumes of Convex Polytopes”. *Frontiers in Algorithmics*. Ed. by Jianxin Wang and Chee Yap. Cham: Springer International Publishing, 2015, pp. 52–65. ISBN: 978-3-319-19647-3. DOI: [10.1007/978-3-319-19647-3_6](https://doi.org/10.1007/978-3-319-19647-3_6).
- [87] Marc G Genton. “Classes of Kernels for Machine Learning: A Statistics Perspective”. en ().
- [88] Arun M. George et al. “A Reservoir-based Convolutional Spiking Neural Network for Gesture Recognition from DVS Input”. en. *2020 International Joint Conference on Neural Networks (IJCNN)*. Glasgow, United Kingdom: IEEE, July 2020, pp. 1–9. ISBN: 978-1-72816-926-2. DOI: [10.1109/IJCNN48605.2020.9206681](https://doi.org/10.1109/IJCNN48605.2020.9206681). URL: <https://ieeexplore.ieee.org/document/9206681/> (visited on 05/11/2023).
- [89] Wulfram Gerstner and Werner M. Kistler. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. 1st ed. Cambridge University Press, Aug. 2002. ISBN: 978-0-521-81384-6 978-0-521-89079-3 978-0-511-81570-6. DOI: [10.1017/CB09780511815706](https://doi.org/10.1017/CB09780511815706). URL: <https://www.cambridge.org/core/product/identifier/9780511815706/type/book> (visited on 12/02/2021).
- [90] Wulfram Gerstner et al. *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. en. Cambridge: Cambridge University Press, 2014. ISBN: 978-1-107-44761-5. DOI: [10.1017/CB09781107447615](https://doi.org/10.1017/CB09781107447615). URL: <http://ebooks.cambridge.org/ref/id/CB09781107447615> (visited on 12/06/2021).
- [91] Marc-Oliver Gewaltig and Markus Diesmann. “NEST (neural simulation tool)”. *Scholarpedia* 2 (Jan. 2007), p. 1430. DOI: [10.4249/scholarpedia.1430](https://doi.org/10.4249/scholarpedia.1430).
- [92] Samanwoy Ghosh-Dastidar and Hojjat Adeli. “Third Generation Neural Networks: Spiking Neural Networks”. en. *Advances in Computational Intelligence*. Ed. by Janusz Kacprzyk, Wen Yu, and Edgar N. Sanchez. Vol. 116. Series Title: Advances in Intelligent and Soft Computing. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 167–178. ISBN: 978-3-642-03155-7 978-3-642-03156-4. DOI: [10.1007/978-3-642-03156-4_17](https://doi.org/10.1007/978-3-642-03156-4_17). URL: http://link.springer.com/10.1007/978-3-642-03156-4_17 (visited on 10/04/2024).
- [93] David Ginsbourger, Rodolphe Le Riche, and Laurent Carraro. “A Multi-points Criterion for Deterministic Parallel Global Optimization based on Gaussian Processes”. en (2008).
- [94] Fred Glover and Manuel Laguna. *Tabu search I*. Vol. 1. Jan. 1999. ISBN: 978-0-7923-9965-0. DOI: [10.1287/ijoc.1.3.190](https://doi.org/10.1287/ijoc.1.3.190).
- [95] Fred W Glover and Gary A Kochenberger. *Handbook of metaheuristics*. Vol. 57. Springer Science & Business Media, 2003.
- [96] Gene H Golub and Charles F Van Loan. *Matrix computations*. JHU press, 2013.

- [97] Javier González et al. “Batch Bayesian Optimization via Local Penalization”. en. *arXiv:1505.08052 [stat]* (Oct. 2015). arXiv: 1505.08052. URL: <http://arxiv.org/abs/1505.08052> (visited on 01/26/2022).
- [98] Ian Goodfellow. *Deep learning*. 2016.
- [99] Robert B. Gramacy and Herbert K. H. Lee. *Optimization Under Unknown Constraints*. en. arXiv:1004.4027 [stat]. July 2010. URL: <http://arxiv.org/abs/1004.4027> (visited on 08/19/2024).
- [100] Andre Gruning and Sander M Bohte. “Spiking Neural Networks: Principles and Challenges”. en. *Computational Intelligence* (2014), p. 10.
- [101] Wenzhe Guo et al. “Neural Coding in Spiking Neural Networks: A Comparative Study for Robust Neuromorphic Systems”. en. *Frontiers in Neuroscience* 15 (Mar. 2021), p. 638474. ISSN: 1662-453X. DOI: [10.3389/fnins.2021.638474](https://doi.org/10.3389/fnins.2021.638474). URL: <https://www.frontiersin.org/articles/10.3389/fnins.2021.638474/full> (visited on 12/08/2021).
- [102] Wenzhe Guo et al. “Toward the Optimal Design and FPGA Implementation of Spiking Neural Networks”. en. *IEEE Transactions on Neural Networks and Learning Systems* 33.8 (Aug. 2022), pp. 3988–4002. ISSN: 2162-237X, 2162-2388. DOI: [10.1109/TNNLS.2021.3055421](https://doi.org/10.1109/TNNLS.2021.3055421). URL: <https://ieeexplore.ieee.org/document/9353400/> (visited on 10/15/2023).
- [103] Nikolaus Hansen et al. “COCO: A Platform for Comparing Continuous Optimizers in a Black-Box Setting”. en. *Optimization Methods and Software* 36.1 (Jan. 2021). arXiv:1603.08785 [cs, math, stat], pp. 114–144. ISSN: 1055-6788, 1029-4937. DOI: [10.1080/10556788.2020.1808977](https://doi.org/10.1080/10556788.2020.1808977). URL: <http://arxiv.org/abs/1603.08785> (visited on 04/09/2024).
- [104] Radoslav Harman and Vladimír Lacko. “On decompositional algorithms for uniform sampling from n-spheres and n-balls”. *Journal of Multivariate Analysis* 101.10 (2010), pp. 2297–2304. ISSN: 0047-259X. DOI: <https://doi.org/10.1016/j.jmva.2010.06.002>. URL: <https://www.sciencedirect.com/science/article/pii/S0047259X10001211>.
- [105] Hananel Hazan et al. “BindsNET: A Machine Learning-Oriented Spiking Neural Networks Library in Python”. en. *Frontiers in Neuroinformatics* 12 (Dec. 2018), p. 89. ISSN: 1662-5196. DOI: [10.3389/fninf.2018.00089](https://doi.org/10.3389/fninf.2018.00089). URL: <https://www.frontiersin.org/article/10.3389/fninf.2018.00089/full> (visited on 12/15/2021).
- [106] Hananel Hazan et al. “Unsupervised Learning with Self-Organizing Spiking Neural Networks”. en. *2018 International Joint Conference on Neural Networks (IJCNN)*. Rio de Janeiro: IEEE, July 2018, pp. 1–6. ISBN: 978-1-5090-6014-6. DOI: [10.1109/IJCNN.2018.8489673](https://doi.org/10.1109/IJCNN.2018.8489673). URL: <https://ieeexplore.ieee.org/document/8489673/> (visited on 12/16/2021).
- [107] Weihua He et al. “Comparing SNNs and RNNs on neuromorphic vision datasets: Similarities and differences”. en. *Neural Networks* 132 (Dec. 2020), pp. 108–120. ISSN: 08936080. DOI: [10.1016/j.neunet.2020.08.001](https://doi.org/10.1016/j.neunet.2020.08.001). URL: <https://linkinghub.elsevier.com/retrieve/pii/S0893608020302902> (visited on 09/23/2021).
- [108] Xin He, Kaiyong Zhao, and Xiaowen Chu. “AutoML: A survey of the state-of-the-art”. en. *Knowledge-Based Systems* 212 (Jan. 2021), p. 106622. ISSN: 09507051. DOI: [10.1016/j.knosys.2020.106622](https://doi.org/10.1016/j.knosys.2020.106622). URL: <https://linkinghub.elsevier.com/retrieve/pii/S0950705120307516> (visited on 08/14/2024).

- [109] Kade M Heckel and Thomas Nowotny. “Spyx: A Library for Just-In-Time Compiled Optimization of Spiking Neural Networks”. *arXiv preprint arXiv:2402.18994* (2024).
- [110] David Heeger et al. “Poisson model of spike generation”. *Handout, University of Stanford* 5.1-13 (2000), p. 76.
- [111] James Hensman, Nicolo Fusi, and Neil D Lawrence. “Gaussian Processes for Big Data”. en ().
- [112] José Miguel Hernández-Lobato et al. “Parallel and distributed Thompson sampling for large-scale accelerated exploration of chemical space”. *Proceedings of the 34th International Conference on Machine Learning - Volume 70. ICML’17*. Sydney, NSW, Australia: JMLR.org, 2017, pp. 1470–1479.
- [113] Michael Hines. “NEURON A Program for Simulation of Nerve Equations”. *Neural Systems: Analysis and Modeling*. Ed. by Frank H. Eeckman. Citation Kkey: neuron. Boston, MA: Springer US, 1993, pp. 127–136. ISBN: 978-1-4615-3560-7. DOI: [10.1007/978-1-4615-3560-7_11](https://doi.org/10.1007/978-1-4615-3560-7_11). URL: https://doi.org/10.1007/978-1-4615-3560-7_11.
- [114] A. L. Hodgkin and A. F. Huxley. “Currents carried by sodium and potassium ions through the membrane of the giant axon of *Loligo*”. en. *The Journal of Physiology* 116.4 (Apr. 1952), pp. 449–472. ISSN: 0022-3751, 1469-7793. DOI: [10.1113/jphysiol.1952.sp004717](https://onlinelibrary.wiley.com/doi/10.1113/jphysiol.1952.sp004717). URL: <https://onlinelibrary.wiley.com/doi/10.1113/jphysiol.1952.sp004717> (visited on 11/02/2021).
- [115] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. en. The MIT Press, Apr. 1992. ISBN: 978-0-262-27555-2. DOI: [10.7551/mitpress/1090.001.0001](https://direct.mit.edu/books/book/2574/Adaptation-in-Natural-and-Artificial-Systems). URL: <https://direct.mit.edu/books/book/2574/Adaptation-in-Natural-and-Artificial-Systems> (visited on 09/04/2024).
- [116] Sebastian Höppner et al. *The SpiNNaker 2 Processing Element Architecture for Hybrid Digital Neuromorphic Computing*. en. arXiv:2103.08392 [cs]. Aug. 2022. URL: <http://arxiv.org/abs/2103.08392> (visited on 08/28/2024).
- [117] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer feedforward networks are universal approximators”. *Neural Networks* 2.5 (1989), pp. 359–366. ISSN: 0893-6080. DOI: [https://doi.org/10.1016/0893-6080\(89\)90020-8](https://doi.org/10.1016/0893-6080(89)90020-8). URL: <https://www.sciencedirect.com/science/article/pii/0893608089900208>.
- [118] R. Horst and H. Tuy. “On the convergence of global methods in multiextremal optimization”. en. *Journal of Optimization Theory and Applications* 54.2 (Aug. 1987), pp. 253–271. ISSN: 0022-3239, 1573-2878. DOI: [10.1007/BF00939434](http://link.springer.com/10.1007/BF00939434). URL: <http://link.springer.com/10.1007/BF00939434> (visited on 04/03/2024).
- [119] Reiner Horst and Hoang Tuy. *Global Optimization*. en. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996. ISBN: 978-3-642-08247-4 978-3-662-03199-5. DOI: [10.1007/978-3-662-03199-5](http://link.springer.com/10.1007/978-3-662-03199-5). URL: <http://link.springer.com/10.1007/978-3-662-03199-5> (visited on 05/06/2024).
- [120] Mahan Hosseini et al. “I tried a bunch of things: The dangers of unexpected overfitting in classification of brain data”. en. *Neuroscience & Biobehavioral Reviews* 119 (Dec. 2020), pp. 456–467. ISSN: 01497634. DOI: [10.1016/j.neubiorev.2020.09.036](https://linkinghub.elsevier.com/retrieve/pii/S0149763420305868). URL: <https://linkinghub.elsevier.com/retrieve/pii/S0149763420305868> (visited on 12/06/2021).
- [121] David H Hubel and Torsten N Wiesel. “Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex”. *The Journal of physiology* 160.1 (1962), p. 106.

- [122] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. “Sequential Model-Based Optimization for General Algorithm Configuration”. en. *Learning and Intelligent Optimization*. Ed. by Carlos A. Coello Coello. Vol. 6683. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 507–523. ISBN: 978-3-642-25565-6 978-3-642-25566-3. DOI: [10.1007/978-3-642-25566-3_40](https://doi.org/10.1007/978-3-642-25566-3_40). URL: http://link.springer.com/10.1007/978-3-642-25566-3_40 (visited on 09/02/2024).
- [123] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, eds. *Automated Machine Learning: Methods, Systems, Challenges*. en. The Springer Series on Challenges in Machine Learning. Cham: Springer International Publishing, 2019. ISBN: 978-3-030-05317-8 978-3-030-05318-5. DOI: [10.1007/978-3-030-05318-5](https://doi.org/10.1007/978-3-030-05318-5). URL: <http://link.springer.com/10.1007/978-3-030-05318-5> (visited on 04/02/2024).
- [124] Waltraud Huyer and Arnold Neumaier. “Global Optimization by Multilevel Coordinate Search”. en ().
- [125] Tasbiha Ibad et al. “Hyperparameter Optimization of Evolving Spiking Neural Network for Time-Series Classification”. en. *New Generation Computing* 40.1 (Apr. 2022), pp. 377–397. ISSN: 0288-3635, 1882-7055. DOI: [10.1007/s00354-022-00165-3](https://doi.org/10.1007/s00354-022-00165-3). URL: <https://link.springer.com/10.1007/s00354-022-00165-3> (visited on 05/11/2023).
- [126] Tatsuya Imai and Akihiro Kishimoto. “A Novel Technique for Avoiding Plateaus of Greedy Best-First Search in Satisficing Planning.” Vol. 2. Jan. 2011. DOI: [10.1609/socs.v2i1.18208](https://doi.org/10.1609/socs.v2i1.18208).
- [127] Ronald L. Iman. “Latin Hypercube Sampling”. en. *Encyclopedia of Quantitative Risk Analysis and Assessment*. Ed. by Edward L. Melnick and Brian S. Everitt. 1st ed. Wiley, July 2008. ISBN: 978-0-470-03549-8 978-0-470-06159-6. DOI: [10.1002/9780470061596.risk0299](https://doi.org/10.1002/9780470061596.risk0299). URL: <https://onlinelibrary.wiley.com/doi/10.1002/9780470061596.risk0299> (visited on 05/05/2024).
- [128] Giacomo Indiveri and Shih-Chii Liu. “Memory and information processing in neuromorphic systems”. en. *Proceedings of the IEEE* 103.8 (Aug. 2015). arXiv:1506.03264 [cs], pp. 1379–1397. ISSN: 0018-9219, 1558-2256. DOI: [10.1109/JPROC.2015.2444094](https://doi.org/10.1109/JPROC.2015.2444094). URL: <http://arxiv.org/abs/1506.03264> (visited on 09/28/2024).
- [129] Laxmi R. Iyer and Yansong Chua. “Classifying Neuromorphic Datasets with Tempotron and Spike Timing Dependent Plasticity”. en. *2020 International Joint Conference on Neural Networks (IJCNN)*. Glasgow, United Kingdom: IEEE, July 2020, pp. 1–8. ISBN: 978-1-72816-926-2. DOI: [10.1109/IJCNN48605.2020.9207474](https://doi.org/10.1109/IJCNN48605.2020.9207474). URL: <https://ieeexplore.ieee.org/document/9207474/> (visited on 09/08/2023).
- [130] E.M. Izhikevich. “Simple model of spiking neurons”. en. *IEEE Transactions on Neural Networks* 14.6 (Nov. 2003), pp. 1569–1572. ISSN: 1045-9227. DOI: [10.1109/TNN.2003.820440](https://doi.org/10.1109/TNN.2003.820440). URL: <http://ieeexplore.ieee.org/document/1257420/> (visited on 11/02/2021).
- [131] G James. *An Introduction to Statistical Learning*. 2013.
- [132] Kevin Jamieson and Amee Talwalkar. *Non-stochastic Best Arm Identification and Hyperparameter Optimization*. 2015. arXiv: [1502.07943](https://arxiv.org/abs/1502.07943) [cs.LG]. URL: <https://arxiv.org/abs/1502.07943>.

- [133] Jian He et al. "Performance Modeling and Analysis of a Massively Parallel DirectPart 2". en. *The International Journal of High Performance Computing Applications* 23.1 (Feb. 2009), pp. 29–41. ISSN: 1094-3420, 1741-2846. DOI: [10.1177/1094342008098463](https://doi.org/10.1177/1094342008098463). URL: <http://journals.sagepub.com/doi/10.1177/1094342008098463> (visited on 03/22/2024).
- [134] Xin Jin et al. "Implementing spike-timing-dependent plasticity on SpiNNaker neuro-morphic hardware". en. *The 2010 International Joint Conference on Neural Networks (IJCNN)*. Barcelona, Spain: IEEE, July 2010, pp. 1–8. ISBN: 978-1-4244-6916-1. DOI: [10.1109/IJCNN.2010.5596372](https://doi.org/10.1109/IJCNN.2010.5596372). URL: <http://ieeexplore.ieee.org/document/5596372/> (visited on 10/13/2023).
- [135] D. R. Jones, C. D. Perttunen, and B. E. Stuckman. "Lipschitzian optimization without the Lipschitz constant". en. *Journal of Optimization Theory and Applications* 79.1 (Oct. 1993), pp. 157–181. ISSN: 0022-3239, 1573-2878. DOI: [10.1007/BF00941892](https://doi.org/10.1007/BF00941892). URL: <http://link.springer.com/10.1007/BF00941892> (visited on 01/27/2022).
- [136] Donald R. Jones and Joaquim R. R. A. Martins. "The DIRECT algorithm: 25 years Later". en. *Journal of Global Optimization* 79.3 (Mar. 2021), pp. 521–566. ISSN: 0925-5001, 1573-2916. DOI: [10.1007/s10898-020-00952-6](https://doi.org/10.1007/s10898-020-00952-6). URL: <https://link.springer.com/10.1007/s10898-020-00952-6> (visited on 03/01/2022).
- [137] Chetan Kadway et al. "Low Power & Low Latency Cloud Cover Detection in Small Satellites Using On-board Neuromorphic Processors". en. *2023 International Joint Conference on Neural Networks (IJCNN)*. Gold Coast, Australia: IEEE, June 2023, pp. 1–8. ISBN: 978-1-66548-867-9. DOI: [10.1109/IJCNN54540.2023.10191569](https://doi.org/10.1109/IJCNN54540.2023.10191569). URL: <https://ieeexplore.ieee.org/document/10191569/> (visited on 08/28/2024).
- [138] Kirthivasan Kandasamy et al. "Parallelised Bayesian Optimisation via Thompson Sampling". *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*. Ed. by Amos Storkey and Fernando Perez-Cruz. Vol. 84. Proceedings of Machine Learning Research. PMLR, Apr. 2018, pp. 133–142. URL: <https://proceedings.mlr.press/v84/kandasamy18a.html>.
- [139] Mohammad Reza Keshtkaran and Chethan Pandarinath. "Enabling hyperparameter optimization in sequential autoencoders for spiking neural data". en ().
- [140] Nitish Shirish Keskar and Richard Socher. *Improving Generalization Performance by Switching from Adam to SGD*. en. arXiv:1712.07628 [cs]. Dec. 2017. URL: <http://arxiv.org/abs/1712.07628> (visited on 10/13/2024).
- [141] Leonid Khachiyan et al. "Generating All Vertices of a Polyhedron Is Hard". en. *Discrete & Computational Geometry* 39.1-3 (Mar. 2008), pp. 174–190. ISSN: 0179-5376, 1432-0444. DOI: [10.1007/s00454-008-9050-5](https://doi.org/10.1007/s00454-008-9050-5). URL: <https://link.springer.com/10.1007/s00454-008-9050-5> (visited on 03/01/2022).
- [142] Ghazaleh Khodabandelou and Amir Nakib. "H -polytope decomposition-based algorithm for continuous optimization". en. *Information Sciences* 558 (May 2021), pp. 50–75. ISSN: 00200255. DOI: [10.1016/j.ins.2020.12.090](https://doi.org/10.1016/j.ins.2020.12.090). URL: <https://linkinghub.elsevier.com/retrieve/pii/S0020025521000232> (visited on 03/01/2022).
- [143] Dion Khodagholy et al. "NeuroGrid: recording action potentials from the surface of the brain". en. *Nature Neuroscience* 18.2 (Feb. 2015). Citation Kkey: neurogrid, pp. 310–315. ISSN: 1097-6256, 1546-1726. DOI: [10.1038/nn.3905](https://doi.org/10.1038/nn.3905). URL: <https://www.nature.com/articles/nn.3905> (visited on 09/29/2024).

- [144] Seijoon Kim et al. “Towards Fast and Accurate Object Detection in Bio-Inspired Spiking Neural Networks Through Bayesian Optimization”. en. *IEEE Access* 9 (2021), pp. 2633–2643. ISSN: 2169-3536. DOI: [10.1109/ACCESS.2020.3047071](https://doi.org/10.1109/ACCESS.2020.3047071). URL: <https://ieeexplore.ieee.org/document/9306772/> (visited on 05/11/2023).
- [145] Youngeun Kim and Priyadarshini Panda. “Optimizing Deeper Spiking Neural Networks for Dynamic Vision Sensing”. en. *Neural Networks* 144 (Dec. 2021), pp. 686–698. ISSN: 08936080. DOI: [10.1016/j.neunet.2021.09.022](https://doi.org/10.1016/j.neunet.2021.09.022). URL: <https://linkinghub.elsevier.com/retrieve/pii/S0893608021003841> (visited on 10/15/2023).
- [146] Youngeun Kim et al. *Neural Architecture Search for Spiking Neural Networks*. en. arXiv:2201.10355 [cs, eess]. July 2022. URL: <http://arxiv.org/abs/2201.10355> (visited on 08/31/2023).
- [147] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. en. arXiv:1412.6980 [cs]. Jan. 2017. URL: <http://arxiv.org/abs/1412.6980> (visited on 08/26/2024).
- [148] Aravindh Krishnamoorthy and Deepak Menon. “Matrix Inversion Using Cholesky Decomposition”. en ().
- [149] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. “CIFAR-10 (Canadian Institute for Advanced Research)” (). URL: <http://www.cs.toronto.edu/~kriz/cifar.html>.
- [150] Shruti R. Kulkarni and Bipin Rajendran. “Spiking neural networks for handwritten digit recognitionSupervised learning and network optimization”. en. *Neural Networks* 103 (July 2018), pp. 118–127. ISSN: 08936080. DOI: [10.1016/j.neunet.2018.03.019](https://doi.org/10.1016/j.neunet.2018.03.019). URL: <https://linkinghub.elsevier.com/retrieve/pii/S0893608018301126> (visited on 05/11/2023).
- [151] Shruti R. Kulkarni et al. “Benchmarking the performance of neuromorphic and spiking neural network simulators”. en. *Neurocomputing* 447 (Aug. 2021), pp. 145–160. ISSN: 09252312. DOI: [10.1016/j.neucom.2021.03.028](https://doi.org/10.1016/j.neucom.2021.03.028). URL: <https://linkinghub.elsevier.com/retrieve/pii/S0925231221003969> (visited on 09/23/2021).
- [152] L Lapicque. “Recherches quantitatives sur l'excitation électrique des nerfs”. *J. Physiol. Paris* 9 (1907), pp. 620–635.
- [153] Yann LeCun and Corinna Cortes. “MNIST handwritten digit database” (2010). URL: <http://yann.lecun.com/exdb/mnist/>.
- [154] Eric Hans Lee et al. *Cost-aware Bayesian Optimization*. en. arXiv:2003.10870 [cs, stat]. Mar. 2020. URL: <http://arxiv.org/abs/2003.10870> (visited on 01/30/2024).
- [155] Gregor Lenz et al. *Tonic: event-based datasets and transformations*. July 2021. DOI: [10.5281/zenodo.5079802](https://doi.org/10.5281/zenodo.5079802). URL: <https://doi.org/10.5281/zenodo.5079802>.
- [156] Cong Sheng Leow, Wang Ling Goh, and Yuan Gao. “Sparsity Through Spiking Convolutional Neural Network for Audio Classification at the Edge”. en. *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*. Monterey, CA, USA: IEEE, May 2023, pp. 1–4. ISBN: 978-1-66545-109-3. DOI: [10.1109/ISCAS46773.2023.10181974](https://doi.org/10.1109/ISCAS46773.2023.10181974). URL: <https://ieeexplore.ieee.org/document/10181974/> (visited on 09/07/2023).
- [157] Liam Li et al. *A System for Massively Parallel Hyperparameter Tuning*. en. arXiv:1810.05934 [cs, stat]. Mar. 2020. URL: <http://arxiv.org/abs/1810.05934> (visited on 09/18/2024).
- [158] Lisha Li et al. *Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization*. en. arXiv:1603.06560 [cs, stat]. June 2018. URL: <http://arxiv.org/abs/1603.06560> (visited on 09/01/2023).

- [159] Yuhang Li et al. *SEENN: Towards Temporal Spiking Early-Exit Neural Networks*. 2023. arXiv: [2304.01230](https://arxiv.org/abs/2304.01230) [cs.NE].
- [160] Petro Liashchynskiy and Pavlo Liashchynskiy. *Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS*. en. arXiv:1912.06059 [cs, stat]. Dec. 2019. URL: <http://arxiv.org/abs/1912.06059> (visited on 09/26/2024).
- [161] P. Lichtsteiner, C. Posch, and T. Delbruck. “A 128 X 128 120db 30mw asynchronous vision sensor that responds to relative intensity change”. en. *2006 IEEE International Solid State Circuits Conference - Digest of Technical Papers*. San Francisco, CA, USA: IEEE, 2006, pp. 2060–2069. ISBN: 978-1-4244-0079-9. DOI: [10.1109/ISSCC.2006.1696265](https://doi.org/10.1109/ISSCC.2006.1696265). URL: <https://ieeexplore.ieee.org/document/1696265/> (visited on 10/03/2024).
- [162] Haitao Liu et al. “A global optimization algorithm for simulation-based problems via the extended DIRECT scheme”. en. *Engineering Optimization* 47.11 (Nov. 2015), pp. 1441–1458. ISSN: 0305-215X, 1029-0273. DOI: [10.1080/0305215X.2014.971777](https://doi.org/10.1080/0305215X.2014.971777). URL: <http://www.tandfonline.com/doi/full/10.1080/0305215X.2014.971777> (visited on 01/27/2022).
- [163] Haitao Liu et al. “When Gaussian Process Meets Big Data: A Review of Scalable GPs”. en. *IEEE Transactions on Neural Networks and Learning Systems* 31.11 (Nov. 2020), pp. 4405–4423. ISSN: 2162-237X, 2162-2388. DOI: [10.1109/TNNLS.2019.2957109](https://doi.org/10.1109/TNNLS.2019.2957109). URL: <https://ieeexplore.ieee.org/document/8951257/> (visited on 09/13/2024).
- [164] G. Liuzzi, S. Lucidi, and V. Piccialli. “A DIRECT-based approach exploiting local minimizations for the solution of large-scale global optimization problems”. en. *Computational Optimization and Applications* 45.2 (Mar. 2010), pp. 353–375. ISSN: 0926-6003, 1573-2894. DOI: [10.1007/s10589-008-9217-2](https://doi.org/10.1007/s10589-008-9217-2). URL: <http://link.springer.com/10.1007/s10589-008-9217-2> (visited on 09/19/2023).
- [165] Jesus L. Lobo et al. “Spiking Neural Networks and online learning: An overview and perspectives”. en. *Neural Networks* 121 (Jan. 2020), pp. 88–100. ISSN: 08936080. DOI: [10.1016/j.neunet.2019.09.004](https://doi.org/10.1016/j.neunet.2019.09.004). URL: <https://linkinghub.elsevier.com/retrieve/pii/S0893608019302655> (visited on 10/04/2024).
- [166] Marco Locatelli and Fabio Schoen. “(Global) Optimization: Historical notes and recent developments”. en. *EURO Journal on Computational Optimization* 9 (2021), p. 100012. ISSN: 21924406. DOI: [10.1016/j.ejco.2021.100012](https://doi.org/10.1016/j.ejco.2021.100012). URL: <https://linkinghub.elsevier.com/retrieve/pii/S2192440621001398> (visited on 05/05/2024).
- [167] Wei-Liem Loh. “ON LATIN HYPERCUBE SAMPLING”. en ().
- [168] Ilya Loshchilov and Frank Hutter. *CMA-ES for Hyperparameter Optimization of Deep Neural Networks*. en. arXiv:1604.07269 [cs]. Apr. 2016. URL: <http://arxiv.org/abs/1604.07269> (visited on 09/04/2024).
- [169] Siegrid Lowel and Wolf Singer. “Selection of Intrinsic Horizontal Connections in the Visual Cortex by Correlated Neuronal Activity”. en (1992).
- [170] Wolfgang Maass. “Networks of spiking neurons: The third generation of neural network models”. en. *Neural Networks* 10.9 (Dec. 1997), pp. 1659–1671. ISSN: 08936080. DOI: [10.1016/S0893-6080\(97\)00011-7](https://doi.org/10.1016/S0893-6080(97)00011-7). URL: <https://linkinghub.elsevier.com/retrieve/pii/S0893608097000117> (visited on 11/04/2021).
- [171] Laurens van der Maaten and Geoffrey Hinton. “Visualizing Data using t-SNE”. *Journal of Machine Learning Research* 9.86 (2008), pp. 2579–2605. URL: <http://jmlr.org/papers/v9/vandermaaten08a.html>.

- [172] Kai Malcolm and Josue Casco-Rodriguez. *A Comprehensive Review of Spiking Neural Networks: Interpretation, Optimization, Efficiency, and Best Practices*. en. arXiv:2303.10780 [cs, eess]. Mar. 2023. URL: <http://arxiv.org/abs/2303.10780> (visited on 05/11/2023).
- [173] Davide Liberato Manna et al. *Frameworks for SNNs: a Review of Data Science-oriented Software and an Expansion of SpykeTorch*. en. arXiv:2302.07624 [cs]. Feb. 2023. URL: <http://arxiv.org/abs/2302.07624> (visited on 09/07/2023).
- [174] Nobuhito Manome, Shuji Shinohara, and Ung-il Chung. “Simple Modification of the Upper Confidence Bound Algorithm by Generalized Weighted Averages”. en ().
- [175] Carver A Mead and Misha A Mahowald. “A silicon model of early visual processing”. *Neural networks* 1.1 (1988), pp. 91–97.
- [176] Nouredine Melab et al. “Parallel metaheuristics: Models and frameworks”. *Parallel combinatorial optimization* (2006), pp. 149–162.
- [177] Erich Merrill et al. “An Empirical Study of Bayesian Optimization: Acquisition Versus Partition”. en ().
- [178] Gérard Meurant. *The Lanczos and Conjugate Gradient Algorithms*. Society for Industrial and Applied Mathematics, 2006. DOI: [10.1137/1.9780898718140](https://doi.org/10.1137/1.9780898718140). eprint: <https://epubs.siam.org/doi/pdf/10.1137/1.9780898718140>. URL: <https://epubs.siam.org/doi/abs/10.1137/1.9780898718140>.
- [179] John G Michaels. “Arrangements with Forbidden Positions”. en ().
- [180] Microsoft. *Neural Network Intelligence*. Version 2.0. Jan. 2021. URL: <https://github.com/microsoft/nni>.
- [181] Evgeny M. Mirkes, Jeza Allohibi, and Alexander Gorban. “Fractional Norms and Quasinorms Do Not Help to Overcome the Curse of Dimensionality”. en. *Entropy* 22.10 (Sept. 2020). Citation KKey: distance2, p. 1105. ISSN: 1099-4300. DOI: [10.3390/e22101105](https://doi.org/10.3390/e22101105). URL: <https://www.mdpi.com/1099-4300/22/10/1105> (visited on 02/04/2023).
- [182] Scott A. Mitchell et al. “Spoke-Darts for High-Dimensional Blue-Noise Sampling”. en. *ACM Transactions on Graphics* 37.2 (Apr. 2018), pp. 1–20. ISSN: 0730-0301, 1557-7368. DOI: [10.1145/3194657](https://doi.org/10.1145/3194657). URL: <https://dl.acm.org/doi/10.1145/3194657> (visited on 01/28/2022).
- [183] Bernardo Morales-Castañeda et al. “A better balance in metaheuristic algorithms: Does it exist?” *Swarm and Evolutionary Computation* 54 (2020), p. 100671. ISSN: 2210-6502. DOI: <https://doi.org/10.1016/j.swevo.2020.100671>. URL: <https://www.sciencedirect.com/science/article/pii/S2210650219304080>.
- [184] David R. Morrison et al. “Cyclic best first search: Using contours to guide branch-and-bound algorithms”. English (US). *Naval Research Logistics Quarterly* 64.1 (Feb. 2017), pp. 64–82. ISSN: 0028-1441. DOI: [10.1002/nav.21732](https://doi.org/10.1002/nav.21732).
- [185] Milad Mozafari et al. “First-spike-based visual categorization using reward-modulated STDP”. *IEEE transactions on neural networks and learning systems* 29.12 (2018). Publisher: IEEE, pp. 6178–6190.
- [186] Milad Mozafari et al. “Spyketorch: Efficient simulation of convolutional spiking neural networks with at most one spike per neuron”. *Frontiers in neuroscience* 13 (2019). Publisher: Frontiers Media SA, p. 625.

- [187] Mervin E. Muller. “A Note on a Method for Generating Points Uniformly on N-Dimensional Spheres”. *Commun. ACM* 2.4 (Apr. 1959), pp. 19–20. ISSN: 0001-0782. DOI: [10.1145/377939.377946](https://doi.org/10.1145/377939.377946). URL: <https://doi.org/10.1145/377939.377946>.
- [188] Rémi Munos. “Optimistic Optimization of a Deterministic Function without the Knowledge of its Smoothness”. en (), p. 9. DOI: [10.5555/2986459.2986547](https://doi.org/10.5555/2986459.2986547).
- [189] Rémi Munos. “Optimistic Optimization of a Deterministic Function without the Knowledge of its Smoothness”. en (), p. 9.
- [190] Byunggook Na et al. *AutoSNN: Towards Energy-Efficient Spiking Neural Networks*. en. arXiv:2201.12738 [cs]. June 2022. URL: <http://arxiv.org/abs/2201.12738> (visited on 08/31/2023).
- [191] A. Nakib, L. Souquet, and E.-G. Talbi. “Parallel fractal decomposition based algorithm for big continuous optimization problems”. en. *Journal of Parallel and Distributed Computing* 133 (Nov. 2019), pp. 297–306. ISSN: 07437315. DOI: [10.1016/j.jpdc.2018.06.002](https://doi.org/10.1016/j.jpdc.2018.06.002). URL: <https://linkinghub.elsevier.com/retrieve/pii/S0743731518304003> (visited on 03/01/2022).
- [192] A. Nakib et al. “Deterministic metaheuristic based on fractal decomposition for large-scale optimization”. en. *Applied Soft Computing* 61 (Dec. 2017), pp. 468–485. ISSN: 15684946. DOI: [10.1016/j.asoc.2017.07.042](https://doi.org/10.1016/j.asoc.2017.07.042). URL: <https://linkinghub.elsevier.com/retrieve/pii/S1568494617304623> (visited on 03/01/2022).
- [193] Emre O. Neftci, Hesham Mostafa, and Friedemann Zenke. *Surrogate Gradient Learning in Spiking Neural Networks*. en. arXiv:1901.09948 [cs, q-bio]. May 2019. URL: <http://arxiv.org/abs/1901.09948> (visited on 02/21/2023).
- [194] Daniel Neil, Michael Pfeiffer, and Shih-Chii Liu. “Learning to be efficient: algorithms for training low-latency, low-compute deep spiking neural networks”. en. *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. Pisa Italy: ACM, Apr. 2016, pp. 293–298. ISBN: 978-1-4503-3739-7. DOI: [10.1145/2851613.2851724](https://doi.org/10.1145/2851613.2851724). URL: <https://dl.acm.org/doi/10.1145/2851613.2851724> (visited on 10/18/2023).
- [195] Joao D. Nunes et al. “Spiking Neural Networks: A Survey”. en. *IEEE Access* 10 (2022), pp. 60738–60764. ISSN: 2169-3536. DOI: [10.1109/ACCESS.2022.3179968](https://doi.org/10.1109/ACCESS.2022.3179968). URL: <https://ieeexplore.ieee.org/document/9787485/> (visited on 08/23/2024).
- [196] Garrick Orchard et al. “Converting Static Image Datasets to Spiking Neuromorphic Datasets Using Saccades”. en. *Frontiers in Neuroscience* 9 (Nov. 2015). ISSN: 1662-453X. DOI: [10.3389/fnins.2015.00437](https://doi.org/10.3389/fnins.2015.00437). URL: <http://journal.frontiersin.org/Article/10.3389/fnins.2015.00437/abstract> (visited on 03/08/2024).
- [197] Garrick Orchard et al. “Efficient neuromorphic signal processing with loihi 2”. *2021 IEEE Workshop on Signal Processing Systems (SiPS)*. IEEE. 2021, pp. 254–259.
- [198] Eustace Painkras et al. “SpiNNaker: A 1-W 18-Core System-on-Chip for Massively-Parallel Neural Network Simulation”. en. *IEEE Journal of Solid-State Circuits* 48.8 (Aug. 2013), pp. 1943–1953. ISSN: 0018-9200, 1558-173X. DOI: [10.1109/JSSC.2013.2259038](https://doi.org/10.1109/JSSC.2013.2259038). URL: <http://ieeexplore.ieee.org/document/6515159/> (visited on 05/10/2022).
- [199] Jin Seon Park and Choong Seon Hong. “Improving the Multimodal Classification Performance of Spiking Neural Networks Through Hyper-Parameter Optimization”. en. *2024 International Conference on Information Networking (ICOIN)*. Ho Chi Minh City, Vietnam: IEEE, Jan. 2024, pp. 182–186. ISBN: 9798350330946. DOI: [10.1109/ICOIN59985.2024.10572194](https://doi.org/10.1109/ICOIN59985.2024.10572194). URL: <https://ieeexplore.ieee.org/document/10572194/> (visited on 08/23/2024).

- [200] Seongsik Park and Sungroh Yoon. “Training Energy-Efficient Deep Spiking Neural Networks with Time-to-First-Spike Coding”. en. *arXiv:2106.02568 [cs]* (June 2021). arXiv: 2106.02568. URL: <http://arxiv.org/abs/2106.02568> (visited on 12/08/2021).
- [201] Seongsik Park et al. “T2FSNN: deep spiking neural networks with time-to-first-spike coding”. *Proceedings of the 57th ACM/EDAC/IEEE Design Automation Conference. DAC '20*. Virtual Event, USA: IEEE Press, 2020. ISBN: 9781450367257.
- [202] Maryam Parsa. “Bayesian-based Multi-Objective Hyperparameter Optimization for Accurate, Fast, and Efficient Neuromorphic System Designs”. PhD Thesis. Purdue University, 2020.
- [203] Maryam Parsa et al. “Bayesian-based Hyperparameter Optimization for Spiking Neuromorphic Systems”. en. *2019 IEEE International Conference on Big Data (Big Data)*. Los Angeles, CA, USA: IEEE, Dec. 2019, pp. 4472–4478. ISBN: 978-1-72810-858-2. DOI: [10.1109/BigData47090.2019.9006383](https://doi.org/10.1109/BigData47090.2019.9006383). URL: <https://ieeexplore.ieee.org/document/9006383/> (visited on 09/23/2021).
- [204] Maryam Parsa et al. “Bayesian Multi-objective Hyperparameter Optimization for Accurate, Fast, and Efficient Neural Network Accelerator Design”. en. *Frontiers in Neuroscience* 14 (July 2020), p. 667. ISSN: 1662-453X. DOI: [10.3389/fnins.2020.00667](https://doi.org/10.3389/fnins.2020.00667). URL: <https://www.frontiersin.org/article/10.3389/fnins.2020.00667/full> (visited on 05/19/2023).
- [205] Maryam Parsa et al. “Accurate and Accelerated Neuromorphic Network Design Leveraging A Bayesian Hyperparameter Pareto Optimization Approach”. en. *International Conference on Neuromorphic Systems 2021*. Knoxville TN USA: ACM, July 2021, pp. 1–8. ISBN: 978-1-4503-8691-3. DOI: [10.1145/3477145.3477160](https://doi.org/10.1145/3477145.3477160). URL: <https://dl.acm.org/doi/10.1145/3477145.3477160> (visited on 09/11/2023).
- [206] Maryam Parsa et al. “Multi-Objective Hyperparameter Optimization for Spiking Neural Network Neuroevolution”. en. *2021 IEEE Congress on Evolutionary Computation (CEC)*. Kraków, Poland: IEEE, June 2021, pp. 1225–1232. ISBN: 978-1-72818-393-0. DOI: [10.1109/CEC45853.2021.9504897](https://doi.org/10.1109/CEC45853.2021.9504897). URL: <https://ieeexplore.ieee.org/document/9504897/> (visited on 09/23/2021).
- [207] Kinjal Patel et al. *A Spiking Neural Network for Image Segmentation*. en. arXiv:2106.08921 [cs]. June 2021. URL: <http://arxiv.org/abs/2106.08921> (visited on 05/11/2023).
- [208] Hélène Paugam-Moisy and Sander Bohte. “Computing with Spiking Neuron Networks”. en. *Handbook of Natural Computing*. Ed. by Grzegorz Rozenberg, Thomas Bäck, and Joost N. Kok. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 335–376. ISBN: 978-3-540-92909-3 978-3-540-92910-9. DOI: [10.1007/978-3-540-92910-9_10](https://doi.org/10.1007/978-3-540-92910-9_10). URL: http://link.springer.com/10.1007/978-3-540-92910-9_10 (visited on 10/04/2024).
- [209] Hélène Paugam-Moisy, Régis Martinez, and Samy Bengio. “Delay learning and polychronization for reservoir computing”. en. *Neurocomputing* 71.7-9 (Mar. 2008), pp. 1143–1158. ISSN: 09252312. DOI: [10.1016/j.neucom.2007.12.027](https://doi.org/10.1016/j.neucom.2007.12.027). URL: <https://linkinghub.elsevier.com/retrieve/pii/S0925231208000507> (visited on 10/04/2024).

- [210] Remigijus Paulavicius, Lakhdar Chiter, and Julius ilinskas. “Global optimization based on bisection of rectangles, function values at diagonals, and a set of Lipschitz constants”. *Journal of Global Optimization* 71.1 (Dec. 2016), pp. 5–20. DOI: [10.1007/s10898-016-0485-6](https://doi.org/10.1007/s10898-016-0485-6). URL: <https://doi.org/10.1007/s10898-016-0485-6>.
- [211] Jens E. Pedersen et al. “Neuromorphic intermediate representation: A unified instruction set for interoperable brain-inspired computing”. en. *Nature Communications* 15.1 (Sept. 2024), p. 8122. ISSN: 2041-1723. DOI: [10.1038/s41467-024-52259-9](https://doi.org/10.1038/s41467-024-52259-9). URL: <https://www.nature.com/articles/s41467-024-52259-9> (visited on 09/30/2024).
- [212] Christian-Gernot Pehle and Jens Egholm Pedersen. *Norse - A deep learning library for spiking neural networks*. Version 0.0.5. Jan. 2021. DOI: [10.5281/zenodo.4422025](https://doi.org/10.5281/zenodo.4422025). URL: <https://doi.org/10.5281/zenodo.4422025>.
- [213] Balint Petro, Nikola Kasabov, and Rita M. Kiss. “Selection and Optimization of Temporal Spike Encoding Methods for Spiking Neural Networks”. en. *IEEE Transactions on Neural Networks and Learning Systems* 31.2 (Feb. 2020), pp. 358–370. ISSN: 2162-237X, 2162-2388. DOI: [10.1109/TNNLS.2019.2906158](https://doi.org/10.1109/TNNLS.2019.2906158). URL: <https://ieeexplore.ieee.org/document/8689349/> (visited on 12/06/2021).
- [214] Michael Pfeiffer and Thomas Pfeil. “Deep Learning With Spiking Neurons: Opportunities and Challenges”. en. *Frontiers in Neuroscience* 12 (Oct. 2018), p. 774. ISSN: 1662-453X. DOI: [10.3389/fnins.2018.00774](https://doi.org/10.3389/fnins.2018.00774). URL: <https://www.frontiersin.org/article/10.3389/fnins.2018.00774/full> (visited on 10/21/2021).
- [215] J. Pinter. “Globally convergent methods for n -dimensional multiextremal optimization”. en. *Optimization* 17.2 (Jan. 1986), pp. 187–202. ISSN: 0233-1934, 1029-4945. DOI: [10.1080/02331938608843118](https://doi.org/10.1080/02331938608843118). URL: <https://www.tandfonline.com/doi/full/10.1080/02331938608843118> (visited on 05/06/2024).
- [216] S.A. Piyavskii. “An algorithm for finding the absolute extremum of a function”. en. *USSR Computational Mathematics and Mathematical Physics* 12.4 (Jan. 1972), pp. 57–67. ISSN: 00415553. DOI: [10.1016/0041-5553\(72\)90115-2](https://doi.org/10.1016/0041-5553(72)90115-2). URL: <https://linkinghub.elsevier.com/retrieve/pii/0041555372901152> (visited on 07/22/2024).
- [217] Dawid Poap et al. “A heuristic approach to the hyperparameters in training spiking neural networks using spike-timing-dependent plasticity”. en. *Neural Computing and Applications* 34.16 (Aug. 2022), pp. 13187–13200. ISSN: 0941-0643, 1433-3058. DOI: [10.1007/s00521-021-06824-8](https://doi.org/10.1007/s00521-021-06824-8). URL: <https://link.springer.com/10.1007/s00521-021-06824-8> (visited on 05/11/2023).
- [218] Filip Ponulak and Andrzej Kasiski. “Introduction to spiking neural networks: Information processing, learning and applications”. en (). Citation Key : ponulak, p. 25.
- [219] Nicholas J. Pritchard et al. *Supervised Radio Frequency Interference Detection with SNNs*. en. arXiv:2406.06075 [astro-ph]. June 2024. URL: <http://arxiv.org/abs/2406.06075> (visited on 08/22/2024).
- [220] Ning Qiao et al. “A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128K synapses”. en. *Frontiers in Neuroscience* 9 (Apr. 2015). ISSN: 1662-453X. DOI: [10.3389/fnins.2015.00141](https://doi.org/10.3389/fnins.2015.00141). URL: <http://journal.frontiersin.org/article/10.3389/fnins.2015.00141/abstract> (visited on 09/29/2024).

- [221] Nestor V. Queipo et al. "Surrogate-based analysis and optimization". en. *Progress in Aerospace Sciences* 41.1 (Jan. 2005), pp. 1–28. ISSN: 03760421. DOI: [10.1016/j.paerosci.2005.02.001](https://doi.org/10.1016/j.paerosci.2005.02.001). URL: <https://linkinghub.elsevier.com/retrieve/pii/S0376042105000102> (visited on 09/05/2024).
- [222] Karthik Charan Raghunathan et al. "Hardware-aware Few-shot Learning on a Memristor-based Small-world Architecture". en. *2024 Neuro Inspired Computational Elements Conference (NICE)*. La Jolla, CA, USA: IEEE, Apr. 2024, pp. 1–8. ISBN: 9798350390582. DOI: [10.1109/NICE61972.2024.10548824](https://doi.org/10.1109/NICE61972.2024.10548824). URL: <https://ieeexplore.ieee.org/document/10548824/> (visited on 08/22/2024).
- [223] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian processes for machine learning*. en. Adaptive computation and machine learning. Cambridge, Mass: MIT Press, 2006. ISBN: 978-0-262-18253-9.
- [224] Nitin Rathi, Priyadarshini Panda, and Kaushik Roy. "STDP-Based Pruning of Connections and Weight Quantization in Spiking Neural Networks for Energy-Efficient Recognition". en. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38.4 (Apr. 2019), pp. 668–677. ISSN: 0278-0070, 1937-4151. DOI: [10.1109/TCAD.2018.2819366](https://doi.org/10.1109/TCAD.2018.2819366). URL: <https://ieeexplore.ieee.org/document/8325325/> (visited on 10/07/2021).
- [225] Nitin Rathi et al. "Exploring Neuromorphic Computing Based on Spiking Neural Networks: Algorithms to Hardware". en. *ACM Computing Surveys* 55.12 (Dec. 2023), pp. 1–49. ISSN: 0360-0300, 1557-7341. DOI: [10.1145/3571155](https://doi.org/10.1145/3571155). URL: <https://dl.acm.org/doi/10.1145/3571155> (visited on 08/23/2024).
- [226] Bernardete Ribeiro et al. "Convolutional Spiking Neural Networks targeting learning and inference in highly imbalanced datasets". en. *Pattern Recognition Letters* (Aug. 2024), S0167865524002344. ISSN: 01678655. DOI: [10.1016/j.patrec.2024.08.002](https://doi.org/10.1016/j.patrec.2024.08.002). URL: <https://linkinghub.elsevier.com/retrieve/pii/S0167865524002344> (visited on 08/22/2024).
- [227] Kaushik Roy, Akhilesh Jaiswal, and Priyadarshini Panda. "Towards spike-based machine intelligence with neuromorphic computing". en. *Nature* 575.7784 (Nov. 2019), pp. 607–617. ISSN: 0028-0836, 1476-4687. DOI: [10.1038/s41586-019-1677-2](https://doi.org/10.1038/s41586-019-1677-2). URL: <http://www.nature.com/articles/s41586-019-1677-2> (visited on 09/23/2021).
- [228] Bodo Rueckauer et al. "Conversion of Continuous-Valued Deep Networks to Efficient Event-Driven Networks for Image Classification". en. *Frontiers in Neuroscience* 11 (Dec. 2017), p. 682. ISSN: 1662-453X. DOI: [10.3389/fnins.2017.00682](https://doi.org/10.3389/fnins.2017.00682). URL: <http://journal.frontiersin.org/article/10.3389/fnins.2017.00682/full> (visited on 10/03/2024).
- [229] Ahmad A. Rushdi et al. "VPS: VORONOI PIECEWISE SURROGATE MODELS FOR HIGH-DIMENSIONAL DATA FITTING". en. *International Journal for Uncertainty Quantification* 7.1 (2017), pp. 1–21. ISSN: 2152-5080. DOI: [10.1615/Int.J.UncertaintyQuantification.2016018697](https://doi.org/10.1615/Int.J.UncertaintyQuantification.2016018697). URL: <http://www.dl.begellhouse.com/journals/52034eb04b657aea,7bd16ae14fe9cbcf,1c04fb773044c729.html> (visited on 01/27/2022).
- [230] Llewyn Salt et al. "Parameter Optimization and Learning in a Spiking Neural Network for UAV Obstacle Avoidance Targeting Neuromorphic Processors". en. *IEEE Transactions on Neural Networks and Learning Systems* 31.9 (Sept. 2020), pp. 3305–3318. ISSN: 2162-237X, 2162-2388. DOI: [10.1109/TNNLS.2019.2941506](https://doi.org/10.1109/TNNLS.2019.2941506). URL: <https://ieeexplore.ieee.org/document/8867860/> (visited on 05/11/2023).

- [231] Daniel J. Saunders et al. "STDP Learning of Image Patches with Convolutional Spiking Neural Networks". en. *2018 International Joint Conference on Neural Networks (IJCNN)*. Rio de Janeiro: IEEE, July 2018, pp. 1–7. ISBN: 978-1-5090-6014-6. DOI: [10.1109/IJCNN.2018.8489684](https://doi.org/10.1109/IJCNN.2018.8489684). URL: <https://ieeexplore.ieee.org/document/8489684/> (visited on 08/22/2022).
- [232] Daniel J. Saunders et al. "Locally connected spiking neural networks for unsupervised feature learning". en. *Neural Networks* 119 (Nov. 2019), pp. 332–340. ISSN: 08936080. DOI: [10.1016/j.neunet.2019.08.016](https://doi.org/10.1016/j.neunet.2019.08.016). URL: <https://linkinghub.elsevier.com/retrieve/pii/S0893608019302333> (visited on 10/12/2022).
- [233] Daniel J. Saunders et al. "Minibatch Processing for Speed-up and Scalability of Spiking Neural Network Simulation". en. *2020 International Joint Conference on Neural Networks (IJCNN)*. Glasgow, United Kingdom: IEEE, July 2020, pp. 1–8. ISBN: 978-1-72816-926-2. DOI: [10.1109/IJCNN48605.2020.9207452](https://doi.org/10.1109/IJCNN48605.2020.9207452). URL: <https://ieeexplore.ieee.org/document/9207452/> (visited on 10/17/2023).
- [234] Juergen Schmidhuber. "Deep Learning in Neural Networks: An Overview". en. *Neural Networks* 61 (Jan. 2015). arXiv:1404.7828 [cs], pp. 85–117. ISSN: 08936080. DOI: [10.1016/j.neunet.2014.09.003](https://doi.org/10.1016/j.neunet.2014.09.003). URL: <http://arxiv.org/abs/1404.7828> (visited on 09/30/2024).
- [235] Catherine Schuman et al. "Evaluating Encoding and Decoding Approaches for Spiking Neuromorphic Systems". en. *Proceedings of the International Conference on Neuromorphic Systems 2022*. Knoxville TN USA: ACM, July 2022, pp. 1–9. ISBN: 978-1-4503-9789-6. DOI: [10.1145/3546790.3546792](https://doi.org/10.1145/3546790.3546792). URL: <https://dl.acm.org/doi/10.1145/3546790.3546792> (visited on 10/03/2023).
- [236] Catherine D. Schuman et al. *A Survey of Neuromorphic Computing and Neural Networks in Hardware*. en. arXiv:1705.06963 [cs]. May 2017. URL: <http://arxiv.org/abs/1705.06963> (visited on 08/23/2024).
- [237] Catherine D. Schuman et al. "Evolutionary Optimization for Neuromorphic Systems". en. *Proceedings of the Neuro-inspired Computational Elements Workshop*. Heidelberg Germany: ACM, Mar. 2020, pp. 1–9. ISBN: 978-1-4503-7718-8. DOI: [10.1145/3381755.3381758](https://doi.org/10.1145/3381755.3381758). URL: <https://dl.acm.org/doi/10.1145/3381755.3381758> (visited on 09/23/2021).
- [238] Catherine D. Schuman et al. "Opportunities for neuromorphic computing algorithms and applications". en. *Nature Computational Science* 2.1 (Jan. 2022), pp. 10–19. ISSN: 2662-8457. DOI: [10.1038/s43588-021-00184-y](https://doi.org/10.1038/s43588-021-00184-y). URL: <https://www.nature.com/articles/s43588-021-00184-y> (visited on 10/03/2023).
- [239] Yaroslav D. Sergeyev. "On convergence of "divide the best" global optimization algorithms". en. *Optimization* 44.3 (Jan. 1998), pp. 303–325. ISSN: 0233-1934, 1029-4945. DOI: [10.1080/02331939808844414](https://doi.org/10.1080/02331939808844414). URL: <http://www.tandfonline.com/doi/abs/10.1080/02331939808844414> (visited on 04/03/2024).
- [240] Ahmed Shaaban et al. "RT-SCNNs: real-time spiking convolutional neural networks for a novel hand gesture recognition using time-domain mm-wave radar data". en. *International Journal of Microwave and Wireless Technologies* (Jan. 2024), pp. 1–13. ISSN: 1759-0787, 1759-0795. DOI: [10.1017/S1759078723001575](https://doi.org/10.1017/S1759078723001575). URL: https://www.cambridge.org/core/product/identifier/S1759078723001575/type/journal_article (visited on 08/22/2024).

- [241] Mahyar Shahsavari, Pierre Falez, and Pierre Boulet. “Combining a volatile and nonvolatile memristor in artificial synapse to improve learning in Spiking Neural Networks”. *2016 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*. 2016, pp. 67–72. DOI: [10.1145/2950067.2950090](https://doi.org/10.1145/2950067.2950090).
- [242] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.
- [243] V. Sharma and D. Srinivasan. “A spiking neural network based on temporal encoding for electricity price time series forecasting in deregulated markets”. en. *The 2010 International Joint Conference on Neural Networks (IJCNN)*. Barcelona, Spain: IEEE, July 2010, pp. 1–8. ISBN: 978-1-4244-6916-1. DOI: [10.1109/IJCNN.2010.5596676](https://doi.org/10.1109/IJCNN.2010.5596676). URL: <http://ieeexplore.ieee.org/document/5596676/> (visited on 10/04/2024).
- [244] Zhenqian Shen et al. *Automated Machine Learning: From Principles to Practices*. en. arXiv:1810.13306 [cs, stat]. Feb. 2024. URL: <http://arxiv.org/abs/1810.13306> (visited on 08/21/2024).
- [245] Shahriar Rezghi Shirsavar and Mohammad-Reza A. Dehaqani. *A Faster Approach to Spiking Deep Convolutional Neural Networks*. en. arXiv:2210.17442 [cs, q-bio]. Oct. 2022. URL: <http://arxiv.org/abs/2210.17442> (visited on 04/14/2023).
- [246] Amar Shrestha et al. “A Survey on Neuromorphic Computing: Models and Hardware”. en. *IEEE Circuits and Systems Magazine* 22.2 (2022), pp. 6–35. ISSN: 1531-636X, 1558-0830. DOI: [10.1109/MCAS.2022.3166331](https://doi.org/10.1109/MCAS.2022.3166331). URL: <https://ieeexplore.ieee.org/document/9782767/> (visited on 08/23/2024).
- [247] Sumit Bam Shrestha and Garrick Orchard. “SLAYER: Spike Layer Error Reassignment in Time”. en. arXiv:1810.08646 [cs, stat] (Sept. 2018). arXiv: 1810.08646. URL: <http://arxiv.org/abs/1810.08646> (visited on 11/02/2021).
- [248] Bruno O. Shubert. “A Sequential Method Seeking the Global Maximum of a Function”. en. *SIAM Journal on Numerical Analysis* 9.3 (Sept. 1972), pp. 379–388. ISSN: 0036-1429, 1095-7170. DOI: [10.1137/0709036](https://doi.org/10.1137/0709036). URL: <http://epubs.siam.org/doi/10.1137/0709036> (visited on 09/29/2022).
- [249] Leslie N. Smith. *A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay*. en. arXiv:1803.09820 [cs, stat]. Apr. 2018. URL: <http://arxiv.org/abs/1803.09820> (visited on 05/05/2024).
- [250] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. “Practical Bayesian Optimization of Machine Learning Algorithms”. en. arXiv:1206.2944 [cs, stat] (Aug. 2012). arXiv: 1206.2944. URL: <http://arxiv.org/abs/1206.2944> (visited on 11/29/2021).
- [251] Shay Snyder, Sumedh R. Risbud, and Maryam Parsa. *Neuromorphic Bayesian Optimization in Lava*. en. arXiv:2305.11060 [cs]. May 2023. URL: <http://arxiv.org/abs/2305.11060> (visited on 10/12/2023).
- [252] Marcel Stimberg, Romain Brette, and Dan FM Goodman. “Brian 2, an intuitive and efficient neural simulator”. en. *eLife* 8 (Aug. 2019), e47314. ISSN: 2050-084X. DOI: [10.7554/eLife.47314](https://doi.org/10.7554/eLife.47314). URL: <https://elifesciences.org/articles/47314> (visited on 12/02/2021).
- [253] Jörg Stork, A. E. Eiben, and Thomas Bartz-Beielstein. “A new taxonomy of global optimization algorithms”. en. *Natural Computing* 21.2 (June 2022), pp. 219–242. ISSN: 1567-7818, 1572-9796. DOI: [10.1007/s11047-020-09820-4](https://doi.org/10.1007/s11047-020-09820-4). URL: <https://link.springer.com/10.1007/s11047-020-09820-4> (visited on 05/05/2024).

- [254] Linas Stripinis and Remigijus Paulaviius. *An empirical study of various candidate selection and partitioning techniques in the DIRECT framework*. en. arXiv:2109.14912 [cs, math]. May 2022. URL: <http://arxiv.org/abs/2109.14912> (visited on 03/22/2024).
- [255] Linas Stripinis and Remigijus Paulaviius. “Lipschitz-inspired HALRECT algorithm for derivative-free global optimization”. en. *Journal of Global Optimization* 88.1 (Jan. 2024), pp. 139–169. ISSN: 0925-5001, 1573-2916. DOI: [10.1007/s10898-023-01296-7](https://link.springer.com/10.1007/s10898-023-01296-7). URL: <https://link.springer.com/10.1007/s10898-023-01296-7> (visited on 05/05/2024).
- [256] Roman G Strongin and Yaroslav D Sergeyev. “Global Optimization: Fractal Approach and Non-redundant Parallelism”. en. *GLOBAL OPTIMIZATION* ().
- [257] Aboozar Taherkhani et al. “A review of learning in biologically plausible spiking neural networks”. en. *Neural Networks* 122 (Feb. 2020), pp. 253–272. ISSN: 08936080. DOI: [10.1016/j.neunet.2019.09.036](https://linkinghub.elsevier.com/retrieve/pii/S0893608019303181). URL: <https://linkinghub.elsevier.com/retrieve/pii/S0893608019303181> (visited on 10/13/2021).
- [258] El-Ghazali Talbi. *Metaheuristics*. Hoboken: Wiley, 2009. DOI: [10.1002/9780470496916](https://doi.org/10.1002/9780470496916).
- [259] El-Ghazali Talbi. *Metaheuristics: from design to implementation*. en. Hoboken, N.J: John Wiley & Sons, 2009. ISBN: 978-0-470-27858-1.
- [260] El-Ghazali Talbi. “Automated Design of Deep Neural Networks: A Survey and Unified Taxonomy”. en. *ACM Computing Surveys* 54.2 (Mar. 2022), pp. 1–37. ISSN: 0360-0300, 1557-7341. DOI: [10.1145/3439730](https://dl.acm.org/doi/10.1145/3439730). URL: <https://dl.acm.org/doi/10.1145/3439730> (visited on 08/22/2022).
- [261] Prof Talbi et al. “Metaheuristics for (variable-size) mixed optimization problems: A unified taxonomy and survey”. *arXiv preprint arXiv:2401.03880* (2024).
- [262] Clarence Tan, Marko arlija, and Nikola Kasabov. “NeuroSense: Short-term emotion recognition and understanding based on spiking neural network modelling of spatio-temporal EEG patterns”. en. *Neurocomputing* 434 (Apr. 2021), pp. 137–148. ISSN: 09252312. DOI: [10.1016/j.neucom.2020.12.098](https://linkinghub.elsevier.com/retrieve/pii/S0925231220320105). URL: <https://linkinghub.elsevier.com/retrieve/pii/S0925231220320105> (visited on 08/23/2024).
- [263] Ganchao Tan et al. “Multi-grained Spatio-Temporal Features Perceived Network for Event-based Lip-Reading”. en. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. New Orleans, LA, USA: IEEE, June 2022, pp. 20062–20071. ISBN: 978-1-66546-946-3. DOI: [10.1109/CVPR52688.2022.01946](https://ieeexplore.ieee.org/document/9879993/). URL: <https://ieeexplore.ieee.org/document/9879993/> (visited on 10/03/2024).
- [264] Amirhossein Tavanaei et al. “Deep Learning in Spiking Neural Networks”. en. *Neural Networks* 111 (Mar. 2019). arXiv: 1804.08150, pp. 47–63. ISSN: 08936080. DOI: [10.1016/j.neunet.2018.12.002](http://arxiv.org/abs/1804.08150). URL: <http://arxiv.org/abs/1804.08150> (visited on 10/13/2021).
- [265] Thomas N. Theis and H.-S. Philip Wong. “The End of Moore’s Law: A New Beginning for Information Technology”. en. *Computing in Science & Engineering* 19.2 (Mar. 2017), pp. 41–50. ISSN: 1521-9615. DOI: [10.1109/MCSE.2017.29](http://ieeexplore.ieee.org/document/7878935/). URL: <http://ieeexplore.ieee.org/document/7878935/> (visited on 09/28/2024).
- [266] Simon Thorpe and Jacques Gautrais. “Rank Order Coding”. en. *Computational Neuroscience*. Ed. by James M. Bower. Boston, MA: Springer US, 1998, pp. 113–118. ISBN: 978-1-4613-7190-8 978-1-4615-4831-7. DOI: [10.1007/978-1-4615-4831-7_19](http://link.springer.com/10.1007/978-1-4615-4831-7_19). URL: http://link.springer.com/10.1007/978-1-4615-4831-7_19 (visited on 12/09/2021).

- [267] Jacob Torrejon et al. “Neuromorphic computing with nanoscale spintronic oscillators”. en. *Nature* 547.7664 (July 2017), pp. 428–431. ISSN: 0028-0836, 1476-4687. DOI: [10.1038/nature23011](https://doi.org/10.1038/nature23011). URL: <https://www.nature.com/articles/nature23011> (visited on 09/28/2024).
- [268] Chun-Wei Tsai et al. “Optimizing hyperparameters of deep learning in predicting bus passengers based on simulated annealing”. en. *Applied Soft Computing* 88 (Mar. 2020), p. 106068. ISSN: 15684946. DOI: [10.1016/j.asoc.2020.106068](https://doi.org/10.1016/j.asoc.2020.106068). URL: <https://linkinghub.elsevier.com/retrieve/pii/S1568494620300089> (visited on 09/05/2024).
- [269] Richard Valenzano and Fan Xie. “On the Completeness of Best-First Search Variants That Use Random Exploration”. *Proceedings of the AAAI Conference on Artificial Intelligence* 30.1 (Feb. 2016). DOI: [10.1609/aaai.v30i1.10081](https://doi.org/10.1609/aaai.v30i1.10081). URL: <https://ojs.aaai.org/index.php/AAAI/article/view/10081>.
- [270] Michal Valko, Alexandra Carpentier, and Rémi Munos. “Stochastic Simultaneous Optimistic Optimization”. en (), p. 9. DOI: [10.5555/3042817.3042896](https://doi.org/10.5555/3042817.3042896).
- [271] Yoeri Van De Burgt et al. “Organic electronics for neuromorphic computing”. en. *Nature Electronics* 1.7 (July 2018), pp. 386–397. ISSN: 2520-1131. DOI: [10.1038/s41928-018-0103-3](https://doi.org/10.1038/s41928-018-0103-3). URL: <https://www.nature.com/articles/s41928-018-0103-3> (visited on 09/28/2024).
- [272] Peter JM Van Laarhoven et al. *Simulated annealing*. Springer, 1987.
- [273] Michel Verleysen and Damien François. “The Curse of Dimensionality in Data Mining and Time Series Prediction”. en. *Computational Intelligence and Bioinspired Systems*. Ed. by David Hutchison et al. Vol. 3512. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 758–770. ISBN: 978-3-540-26208-4 978-3-540-32106-4. DOI: [10.1007/11494669_93](https://doi.org/10.1007/11494669_93). URL: http://link.springer.com/10.1007/11494669_93 (visited on 10/02/2024).
- [274] Alex Vicente-Sola et al. “Keys to accurate feature extraction using residual spiking neural networks”. en. *Neuromorphic Computing and Engineering* 2.4 (Dec. 2022), p. 044001. ISSN: 2634-4386. DOI: [10.1088/2634-4386/ac8bef](https://doi.org/10.1088/2634-4386/ac8bef). URL: <https://iopscience.iop.org/article/10.1088/2634-4386/ac8bef> (visited on 05/11/2023).
- [275] Alex Vigneron and Jean Martinet. “A critical survey of STDP in Spiking Neural Networks for Pattern Recognition”. en. *2020 International Joint Conference on Neural Networks (IJCNN)*. Glasgow, United Kingdom: IEEE, July 2020, pp. 1–9. ISBN: 978-1-72816-926-2. DOI: [10.1109/IJCNN48605.2020.9207239](https://doi.org/10.1109/IJCNN48605.2020.9207239). URL: <https://ieeexplore.ieee.org/document/9207239/> (visited on 10/07/2021).
- [276] Alejandro Villagran et al. “Non-parametric Sampling Approximation via Voronoi Tessellations”. en. *Communications in Statistics - Simulation and Computation* 45.2 (Feb. 2016), pp. 717–736. ISSN: 0361-0918, 1532-4141. DOI: [10.1080/03610918.2013.870798](https://doi.org/10.1080/03610918.2013.870798). URL: <https://www.tandfonline.com/doi/full/10.1080/03610918.2013.870798> (visited on 01/28/2022).
- [277] Aaron Voelker, Jan Gosmann, and Terrence Stewart. “Efficiently sampling vectors and coordinates from the n-sphere and n-ball” (Jan. 2017). DOI: [10.13140/RG.2.2.15829.01767/1](https://doi.org/10.13140/RG.2.2.15829.01767/1).
- [278] John Von Neumann. “First Draft of a Report on the EDVAC”. *IEEE Annals of the History of Computing* 15.4 (1993), pp. 27–75.

- [279] M Mitchell Waldrop. “The chips are down for Moores law”. *Nature News* 530.7589 (2016), p. 144.
- [280] Jialei Wang et al. *Parallel Bayesian Global Optimization of Expensive Functions*. en. arXiv:1602.05149 [math, stat]. May 2019. URL: <http://arxiv.org/abs/1602.05149> (visited on 09/23/2024).
- [281] Xiangwen Wang, Xianghong Lin, and Xiaochao Dang. “Supervised learning in spiking neural networks: A review of algorithms and evaluations”. en. *Neural Networks* 125 (May 2020), pp. 258–280. ISSN: 08936080. DOI: [10.1016/j.neunet.2020.02.011](https://doi.org/10.1016/j.neunet.2020.02.011). URL: <https://linkinghub.elsevier.com/retrieve/pii/S0893608020300563> (visited on 10/13/2021).
- [282] Xilu Wang et al. *Recent Advances in Bayesian Optimization*. en. arXiv:2206.03301 [cs, math]. Nov. 2022. URL: <http://arxiv.org/abs/2206.03301> (visited on 09/18/2024).
- [283] Ziyu Wang et al. *Bayesian Multi-Scale Optimistic Optimization*. en. arXiv:1402.7005 [cs, stat]. Feb. 2014. URL: <http://arxiv.org/abs/1402.7005> (visited on 12/08/2023).
- [284] Shuhei Watanabe. *Tree-Structured Parzen Estimator: Understanding Its Algorithm Components and Their Roles for Better Empirical Performance*. en. arXiv:2304.11127 [cs]. May 2023. URL: <http://arxiv.org/abs/2304.11127> (visited on 07/10/2024).
- [285] Shuhei Watanabe. *Tree-Structured Parzen Estimator: Understanding Its Algorithm Components and Their Roles for Better Empirical Performance*. en. arXiv:2304.11127 [cs]. May 2023. URL: <http://arxiv.org/abs/2304.11127> (visited on 09/19/2024).
- [286] D. F. Watson. “Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes*”. *The Computer Journal* 24.2 (Jan. 1981), pp. 167–172. ISSN: 0010-4620. DOI: [10.1093/comjnl/24.2.167](https://doi.org/10.1093/comjnl/24.2.167). eprint: <https://academic.oup.com/comjnl/article-pdf/24/2/167/967258/240167.pdf>. URL: <https://doi.org/10.1093/comjnl/24.2.167>.
- [287] James T. Wilson et al. “The reparameterization trick for acquisition functions”. en. *arXiv:1712.00424 [cs, math, stat]* (Dec. 2017). arXiv: 1712.00424. URL: <http://arxiv.org/abs/1712.00424> (visited on 01/26/2022).
- [288] D.H. Wolpert and W.G. Macready. “No free lunch theorems for optimization”. en. *IEEE Transactions on Evolutionary Computation* 1.1 (Apr. 1997), pp. 67–82. ISSN: 1089778X. DOI: [10.1109/4235.585893](https://doi.org/10.1109/4235.585893). URL: <http://ieeexplore.ieee.org/document/585893/> (visited on 10/02/2024).
- [289] F. Wyffels and B. Schrauwen. “A comparative study of Reservoir Computing strategies for monthly time series prediction”. en. *Neurocomputing* 73.10-12 (June 2010), pp. 1958–1964. ISSN: 09252312. DOI: [10.1016/j.neucom.2010.01.016](https://doi.org/10.1016/j.neucom.2010.01.016). URL: <https://linkinghub.elsevier.com/retrieve/pii/S0925231210000962> (visited on 10/04/2024).
- [290] Li Yang and Abdallah Shami. “On hyperparameter optimization of machine learning algorithms: Theory and practice”. en. *Neurocomputing* 415 (Nov. 2020), pp. 295–316. ISSN: 09252312. DOI: [10.1016/j.neucom.2020.07.061](https://doi.org/10.1016/j.neucom.2020.07.061). URL: <https://linkinghub.elsevier.com/retrieve/pii/S0925231220311693> (visited on 09/07/2023).
- [291] Minhao Yang et al. “A 0.5 V 55 μW 64 \times 2 Channel Binaural Silicon Cochlea for Event-Driven Stereo-Audio Sensing”. en. *IEEE Journal of Solid-State Circuits* 51.11 (Nov. 2016), pp. 2554–2569. ISSN: 0018-9200, 1558-173X. DOI: [10.1109/JSSC.2016.2604285](https://doi.org/10.1109/JSSC.2016.2604285). URL: <http://ieeexplore.ieee.org/document/7574309/> (visited on 10/03/2024).

- [292] Man Yao et al. “Spike-based dynamic computing with asynchronous sensing-computing neuromorphic chip”. en. *Nature Communications* 15.1 (May 2024), p. 4464. issn: 2041-1723. doi: [10.1038/s41467-024-47811-6](https://doi.org/10.1038/s41467-024-47811-6). url: <https://www.nature.com/articles/s41467-024-47811-6> (visited on 08/28/2024).
- [293] Alper Yegenoglu et al. “Exploring Parameter and Hyper-Parameter Spaces of Neuroscience Models on High Performance Computers With Learning to Learn”. en. *Frontiers in Computational Neuroscience* 16 (May 2022), p. 885207. issn: 1662-5188. doi: [10.3389/fncom.2022.885207](https://doi.org/10.3389/fncom.2022.885207). url: <https://www.frontiersin.org/articles/10.3389/fncom.2022.885207/full> (visited on 08/22/2024).
- [294] Jason Yik et al. *NeuroBench: A Framework for Benchmarking Neuromorphic Computing Algorithms and Systems*. en. arXiv:2304.04640 [cs]. Jan. 2024. url: <http://arxiv.org/abs/2304.04640> (visited on 08/21/2024).
- [295] Huifeng Yin et al. *Understanding the Functional Roles of Modelling Components in Spiking Neural Networks*. en. arXiv:2403.16674 [cs]. Mar. 2024. url: <http://arxiv.org/abs/2403.16674> (visited on 08/22/2024).
- [296] Aaron R. Young et al. “A Review of Spiking Neuromorphic Hardware Communication Systems”. en. *IEEE Access* 7 (2019), pp. 135606–135620. issn: 2169-3536. doi: [10.1109/ACCESS.2019.2941772](https://doi.org/10.1109/ACCESS.2019.2941772). url: <https://ieeexplore.ieee.org/document/8843969/> (visited on 09/25/2024).
- [297] Amirreza Yousefzadeh et al. “On Practical Issues for Stochastic STDP Hardware With 1-bit Synaptic Weights”. en. *Frontiers in Neuroscience* 12 (Oct. 2018), p. 665. issn: 1662-453X. doi: [10.3389/fnins.2018.00665](https://doi.org/10.3389/fnins.2018.00665). url: <https://www.frontiersin.org/article/10.3389/fnins.2018.00665/full> (visited on 09/07/2023).
- [298] Qiang Yu et al. “A brain-inspired spiking neural network model with temporal encoding and learning”. en. *Neurocomputing* 138 (Aug. 2014), pp. 3–13. issn: 09252312. doi: [10.1016/j.neucom.2013.06.052](https://doi.org/10.1016/j.neucom.2013.06.052). url: <https://linkinghub.elsevier.com/retrieve/pii/S0925231214003452> (visited on 12/02/2021).
- [299] Tong Yu and Hong Zhu. *Hyper-Parameter Optimization: A Review of Algorithms and Applications*. en. arXiv:2003.05689 [cs, stat]. Mar. 2020. url: <http://arxiv.org/abs/2003.05689> (visited on 09/07/2023).
- [300] Chenhui Zhao, Zenan Huang, and Donghui Guo. “Spiking neural network dynamic system modeling for computation of quantum annealing and its convergence analysis”. en. *Quantum Information Processing* 20.2 (Feb. 2021), p. 70. issn: 1570-0755, 1573-1332. doi: [10.1007/s11128-021-03003-5](https://doi.org/10.1007/s11128-021-03003-5). url: <https://link.springer.com/10.1007/s11128-021-03003-5> (visited on 09/05/2024).
- [301] Qi Zhao et al. *AutoOptLib: Tailoring Metaheuristic Optimizers via Automated Algorithm Design*. en. arXiv:2303.06536 [cs]. Nov. 2023. url: <http://arxiv.org/abs/2303.06536> (visited on 05/23/2024).
- [302] Hanle Zheng et al. “Going Deeper With Directly-Trained Larger Spiking Neural Networks”. en. *Proceedings of the AAAI Conference on Artificial Intelligence* 35.12 (May 2021), pp. 11062–11070. issn: 2374-3468, 2159-5399. doi: [10.1609/aaai.v35i12.17320](https://doi.org/10.1609/aaai.v35i12.17320). url: <https://ojs.aaai.org/index.php/AAAI/article/view/17320> (visited on 08/31/2023).
- [303] Pan Zhou et al. “Towards Theoretically Understanding Why SGD Generalizes Better Than ADAM in Deep Learning”. en ().

- [304] Peng Zhou et al. "Gradient-Based Neuromorphic Learning on Dynamical RRAM Arrays". en. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 12.4 (Dec. 2022), pp. 888–897. ISSN: 2156-3357, 2156-3365. DOI: [10.1109/JETCAS.2022.3224071](https://doi.org/10.1109/JETCAS.2022.3224071). URL: <https://ieeexplore.ieee.org/document/9961206/> (visited on 10/13/2023).
- [305] Yan Zhou, Yaochu Jin, and Jinliang Ding. "Surrogate-Assisted Evolutionary Search of Spiking Neural Architectures in Liquid State Machines". en. *Neurocomputing* 406 (Sept. 2020), pp. 12–23. ISSN: 09252312. DOI: [10.1016/j.neucom.2020.04.079](https://doi.org/10.1016/j.neucom.2020.04.079). URL: <https://linkinghub.elsevier.com/retrieve/pii/S0925231220306640> (visited on 09/23/2021).
- [306] J. ilinskas. "BRANCH AND BOUND WITH SIMPLICIAL PARTITIONS FOR GLOBAL OPTIMIZATION". en. *Mathematical Modelling and Analysis* 13.1 (Mar. 2008), pp. 145–159. ISSN: 1392-6292, 1648-3510. DOI: [10.3846/1392-6292.2008.13.145-159](https://doi.org/10.3846/1392-6292.2008.13.145-159). URL: <https://journals.vgtu.lt/index.php/MMA/article/view/6998> (visited on 03/01/2022).

Trust Region and Scalable Constrained Bayesian Optimization

A.1 Trust Region Bayesian Optimization

To clear up any ambiguity in the context of TuRBO and SCBO, \mathcal{D} describes the archive of computed solutions. The validation accuracy computed after training and validation of a SNN is written as the objective function f . So $f(\lambda) = \mathcal{L}_{\mathbb{D}}(\mathcal{N}_{\theta^*}^{\lambda}, \mathcal{D}_{\text{valid}})$ (see chapter 2).

SCBO and TuRBO instantiate trust regions to restrict the search within regions where the surrogate model is considered reliable, helping to focus on promising areas. These trust regions increase the scalability in high dimensional search spaces. In TuRBO and SCBO a trust region L is a hypercubic subspace of a continuous search space Ω , of side length l ; $L \subseteq \Omega$. The trust region is always centered on the best current solution λ_{best} , and at each iteration, the algorithm returns a batch of s solutions sampled within L using TS and the posterior distribution; $S = \{\lambda_1, \dots, \lambda_s\}$ such that:

$$\forall \lambda \in S, p(y \mid \lambda, \mathcal{D}, \sigma_{\text{noise}}) = \mathcal{N}(y; \mu_{\mathcal{D}}(\lambda), K_{\mathcal{D}}(\lambda, \lambda) + \sigma_{\text{noise}}^2) . \quad (\text{A.1})$$

TS is a bandit algorithm. Initially, we create a larger batch $B \supset S$ of b random solutions. Then, instead of optimizing the acquisition function, we sample n possible outcomes (realizations) from the posterior and for each element of B . We write $r(\lambda)$ a single realization sampled with the posterior $p(y \mid \lambda, \mathcal{D}, \sigma_{\text{noise}})$. We define the relation operator *is better than* as $>$. Then for two solutions λ_1 and λ_2 we have:

$$\lambda_1 > \lambda_2 \iff f(\lambda_1) > f(\lambda_2), \text{ or} \quad (\text{A.2})$$

$$\lambda_1 > \lambda_2 \iff r(\lambda_1) > r(\lambda_2) . \quad (\text{A.3})$$

$$(\text{A.4})$$

Then, to extract $S \subset B$ s.t. $|S| = s$, we select the top s with equation A.3 solutions according to their maximum possible outcome among n realizations.

The equation A.2 is used to update the trust region L . We define n_{success} and n_{fail} the number of successes or fails at improving the best current solution λ_{best} . At each iteration, a batch S is generated and if $\max f(S) > f(\lambda_{\text{best}})$, then a counter successes is increased,

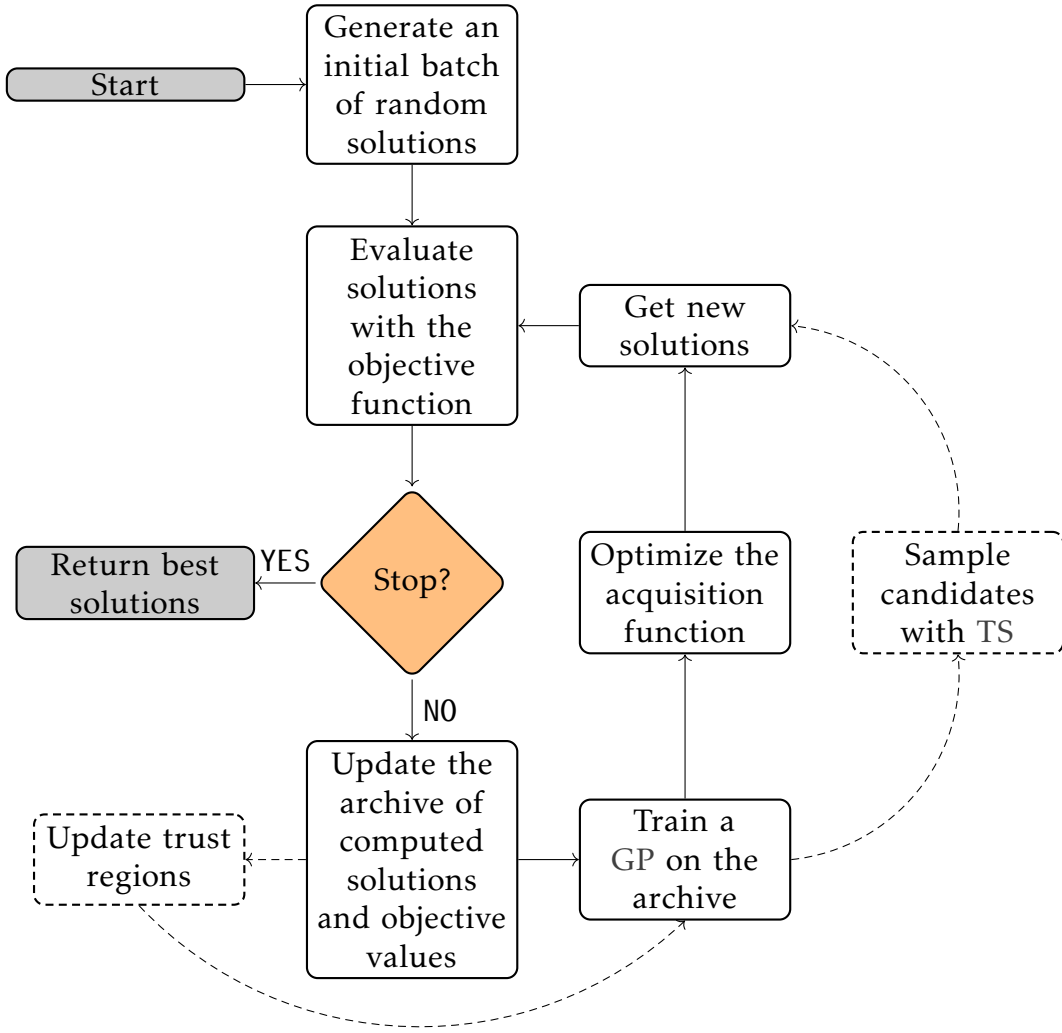


Figure A.1: Flowchart of BO and TuRBO. The dashed lines illustrate the alternative for TuRBO.

otherwise `fails` is increased. So, the length l of a trust region is updated according to:

$$\begin{aligned} l &\leftarrow l \times 2, \text{ if } \text{success} \geq n_{\text{success}} \\ l &\leftarrow l / 2, \text{ if } \text{fails} \geq n_{\text{fail}} \end{aligned} \quad (\text{A.5})$$

Once the trust region becomes sufficiently small, i.e. $l \leq \epsilon$, it is reset to its default size, still centered on λ_{best} . The update of a trust region L is described in algorithm 24. The alternative workflow of TuRBO is illustrated in A.1.

A.2 Scalable Constrained Bayesian Optimization

SCBO introduces a black-box approach, in this work the constraints are the ones defined in equation 5.1. SCBO models each constraint by a dedicated GP. So, we write $c(\lambda)$ the actual constraint value of λ , and $r_c(\lambda)$ a realization from the posterior distribution of the GP on c .

Therefore, the algorithm needs to redefine the comparison between two solutions, λ_1 and

λ_2 . We say that λ_1 is better than λ_2 ; $\lambda_1 \succ \lambda_2$, if:

$$\left\{ \begin{array}{l} f(\lambda_1) > f(\lambda_2) \quad , \text{ if } \forall c \in C, (c(\lambda_1) < 0) \wedge (c(\lambda_2) < 0) \\ v(\lambda_1) < v(\lambda_2) \quad , \text{ if } (v(\lambda_1) \geq 0) \wedge (v(\lambda_2) \geq 0) \\ v(\lambda_2) \geq 0 \end{array} \right. , \quad (A.6)$$

$$\left\{ \begin{array}{l} r(\lambda_1) > r(\lambda_2) \quad , \text{ if } \forall c \in C, (r_c(\lambda_1) < 0) \wedge (r_c(\lambda_2) < 0) \\ v_c(\lambda_1) < v_c(\lambda_2) \quad , \text{ if } (v_c(\lambda_1) \geq 0) \wedge (v_c(\lambda_2) \geq 0) \\ v_c(\lambda_2) \geq 0 \end{array} \right.$$

with v the total constraint violation of a given λ such that $v(\lambda) \triangleq \sum_{c \in C} \max(c(\lambda), 0)$ and $v_c(\lambda) \triangleq \max(r_c(\lambda), 0)$.

Based on the equation A.6, the update of the trust region L is described in algorithm 24. At each iteration and according to a potential improvement of the current λ_{best} , SCBO will shrink or expand the trust region L after n_{fail} failures or n_{success} successes at improving λ_{best} .

A.3 SCBO with asynchronous trust regions

The parallelization of SCBO is necessary since the early stopping criterion generates a high variability in the computation time of SNNs. The asynchronous strategy consists in maximizing the workload of processes by keeping a continuous flow of solutions. Thus, TS was used to replace the acquisition function, since it can be parallelized asynchronously [112]. The asynchronous parallelization of SCBO is described in figure A.2 and algorithm 25.

The main data structure used for asynchronous parallelization consists in maintaining a FIFO queue with candidates solutions (λ) waiting to be evaluated within idle worker processes (one process per GPU). A master process has to compute SCBO, send candidates to workers, receive evaluations from the workers, and store these evaluations. If the size of the FIFO queue decreases to a certain threshold, the master worker updates the GPs of SCBO with newly computed solutions. This process is described in figure A.2. However, due to the stochasticity of the evaluations, induced by the early stopping described in section 5.2 and other HPs (e.g. batch size, epochs, etc.), we cannot ensure that trust regions are always updated with the same number of evaluated solutions. Consequently, the trust region may be updated while expensive evaluations of previous batches of candidates are still pending. As a result, at a given iteration i , some solutions from previous batches may be located outside the current trust region, as it has been updated with less costly solutions.

To solve the previously described asynchronous problem, we propose to temporally link candidates to the trust regions it were sampled in. We write L_I the trust region at the iteration I , and L_i the trust region at a given previous iteration $i < I$. We rewrite the batch of candidates S as \mathcal{B}_I the batch of solutions at I . This batch is made of pairs $(\lambda_{i,j}, L_i)$, where $\lambda_{i,j}$ is an evaluated solution (i.e. accuracies and black-box constraints are known) and the j^{th} candidate from Thompson sampling at a previous iteration i . So, at iteration I , $\mathcal{B}_I := \{(\lambda_{i,j}, L_i) | i < I, 0 \leq j \leq b\} \setminus \mathcal{B}_{0,\dots,I-1}$, are the evaluated solutions returned by the master process when the size of the FIFO queue is under the threshold. The new trust region update is described in algorithm 25. The main idea is that while costly solutions are being computed, SCBO can exploit a cheaper trust region. If a costly solution, once evaluated, improves the current best solution and is outside the current trust region, then L_I is restored to the state

Algorithm 24 Trust region update**Inputs:**

- 1: $S = \{\lambda_1, \dots, \lambda_b\}$ *batch of computed solutions*
- 2: f *Objective function (validation accuracy)*
- 3: λ_{best}
- 4: l *Length of current trust region*
- 5: n_{success}
- 6: **successes** *Current number of successes*
- 7: n_{fail}
- 8: **fails** *Current number of fails*
- 9:

Outputs: l , **fails**, **successes**, λ_{best}

- 10: $\text{best} \leftarrow \underset{\lambda \in S}{\operatorname{argmax}} f(\lambda)$
- 11: **if** $\text{best} > \lambda_{\text{best}}$ **then** *Use eq.A.2, A.3 or A.6*
- 12: $\lambda_{\text{best}} \leftarrow \text{best}$
- 13: **successes** \leftarrow **successes** + 1
- 14: **else**
- 15: **fails** \leftarrow **fails** + 1
- 16: **if** **fails** $\geq n_{\text{fail}}$ **then**
- 17: $l \leftarrow l/2$
- 18: **successes** \leftarrow 0
- 19: **fails** \leftarrow 0
- 20: **if** **successes** $\geq n_{\text{success}}$ **then**
- 21: $l \leftarrow l \times 2$
- 22: **successes** \leftarrow 0
- 23: **fails** \leftarrow 0

Algorithm 25 Asynchronous Trust region update**Inputs:**

- 1: $\mathcal{B}_I = \{\dots, (\lambda_{i,j}, L_i), \dots\}$ *batch of computed solutions*
- 2: f *Objective function (validation accuracy)*
- 3: λ_{best}
- 4: L, l *Current trust region and length*
- 5: n_{success}
- 6: **successes** *Current number of successes*
- 7: n_{fail}
- 8: **fails** *Current number of fails*
- 9:

Outputs: L, l , **fails**, **successes**, λ_{best}

- 10: $\text{best} \leftarrow \underset{\lambda \in \mathcal{B}}{\operatorname{argmax}} f(\lambda)$
- 11: **if** $\text{best} > \lambda_{\text{best}}$ **then** *Use eq.A.2, A.3, or A.6*
- 12: $\lambda_{\text{best}} \leftarrow \text{best}$
- 13: **if** $\text{best} \in L$ **then**
- 14: **successes** \leftarrow **successes** + 1
- 15: **else**
- 16: $L \leftarrow L_{\text{best}}$
- 17: $l \leftarrow l_{\text{best}}$
- 18: **successes** \leftarrow **successes**_{best} + 1
- 19: **fails** \leftarrow 0
- 20: **else**
- 21: **if** $\exists \lambda \in \mathcal{B}, \lambda \in L$ **then**
- 22: **fails** \leftarrow **fails** + 1
- 23: **if** **successes** $\geq n_{\text{success}}$ **then**
- 24: $l \leftarrow l \times 2$
- 25: **successes** \leftarrow 0
- 26: **fails** \leftarrow 0
- 27: **if** **fails** $\geq n_{\text{fail}}$ **then**
- 28: $l \leftarrow l/2$
- 29: **successes** \leftarrow 0
- 30: **fails** \leftarrow 0

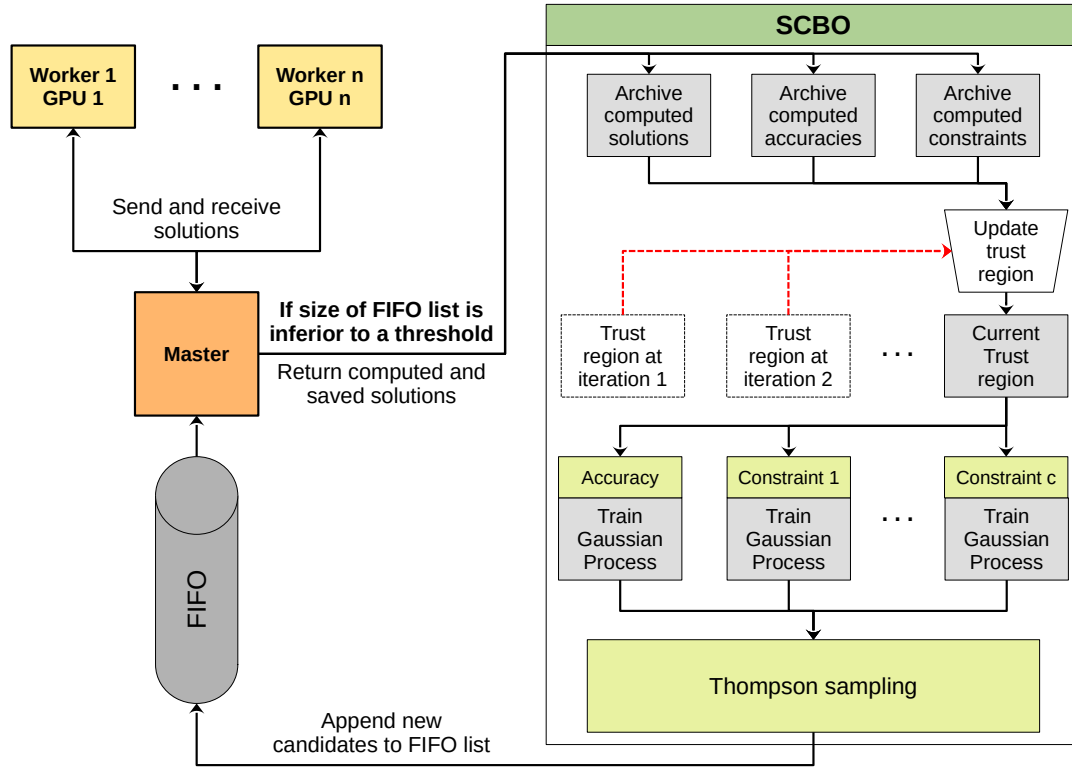


Figure A.2: Workflow of SCBO.

at which the best costly solution was sampled. This allows asynchronous SCBO to retrieve cheap and relevant information while computing expensive solutions. In algorithm 25, l_i corresponds to the length of L_i and successes_i corresponds to the number of successes of L_i .

Sensitivity analysis on the stopping criterion

To better evaluate the impact of the stopping criterion and its HPs α and β , the following section discusses additional experiments using the same configuration as S-SLAY-MNIST. These experiments were conducted over 15 hours with 16 different configurations; $\alpha \in \{1, 5, 10, 20, 40\}$ and $\beta \in [0.05, 0.1, 0.2, 0.4]$. Understanding how these new HPs work may help improve the approach and avoid misclassification of silent networks.

In figure B.1, one can see that when the minimum required number of spike per data points (α) is high, and the proportion β is low, the number of sampled silent networks increases, impacting the budget significantly. This can be explained by the fact that enforcing high spiking activity without allowing a certain tolerance causes many networks to fail to meet the requirement within a few training steps. Indeed, when β is low, only a few samples have to emit less than α spikes to make the SNN stop. When β increases with high $\alpha = 20$, we can notice a decrease in the proportion of silent networks. Conversely, when α is high and β is low, the number of silent networks decreases, along with their impact on the budget. So mid-values appears to be a reasonable tradeoff between the number of detected silent networks and proportion of the budget dedicated to them.

Further studies with neuromorphic datasets could help better understand their behaviors. A reasonable assumption could be that when high spiking activity is required, a higher tolerance for the proportion of non-spiking samples is necessary. This would allow the gradient to have sufficient budget to meet the requirement before being interrupted by early stopping. However, higher tolerance can also mean a higher budget spent on silent networks.

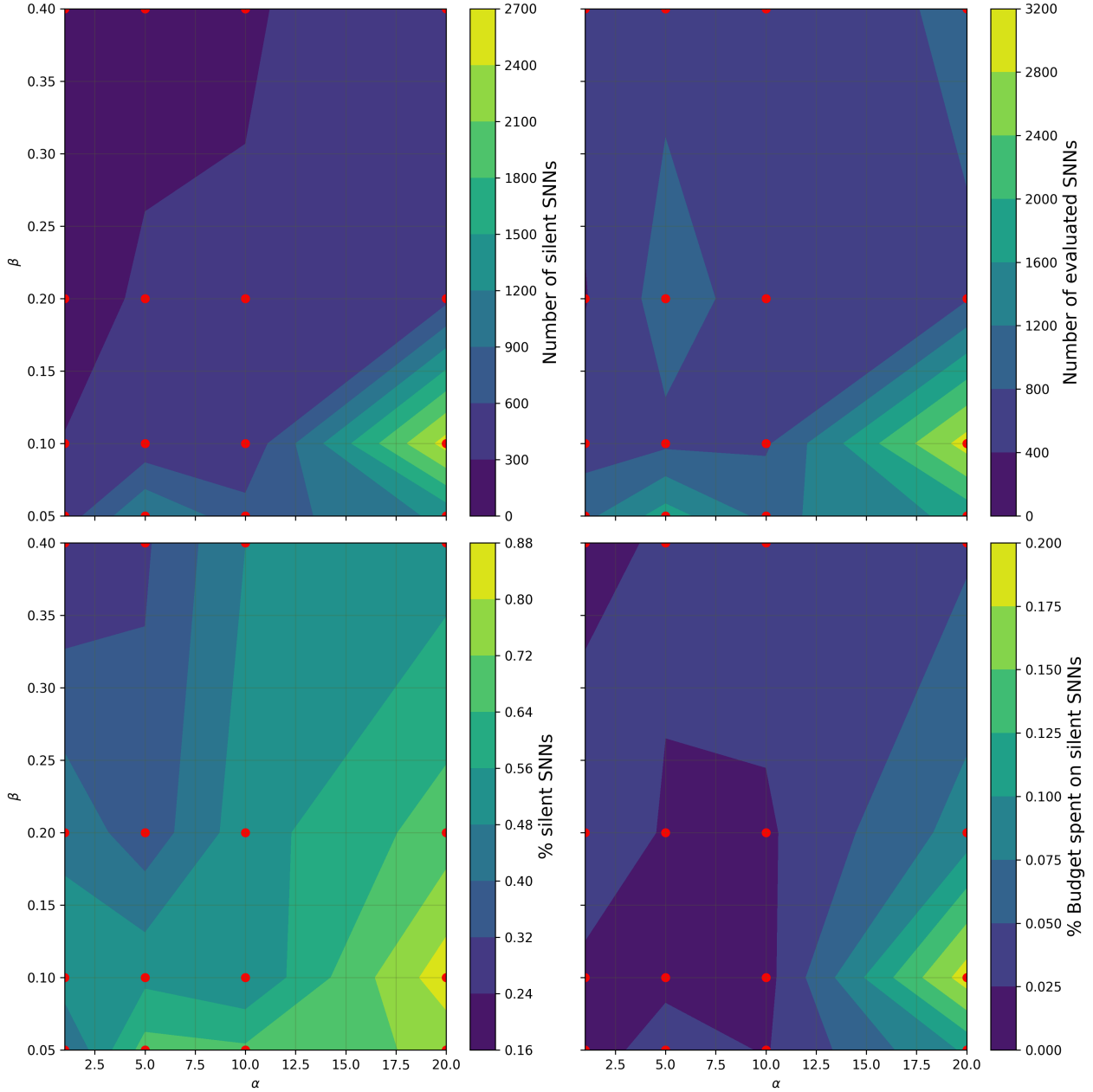
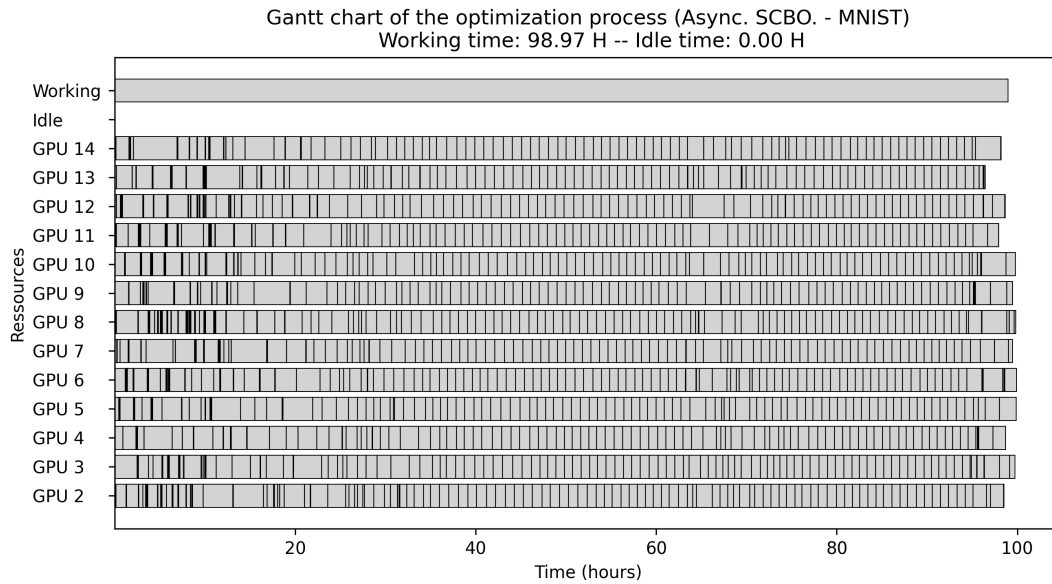


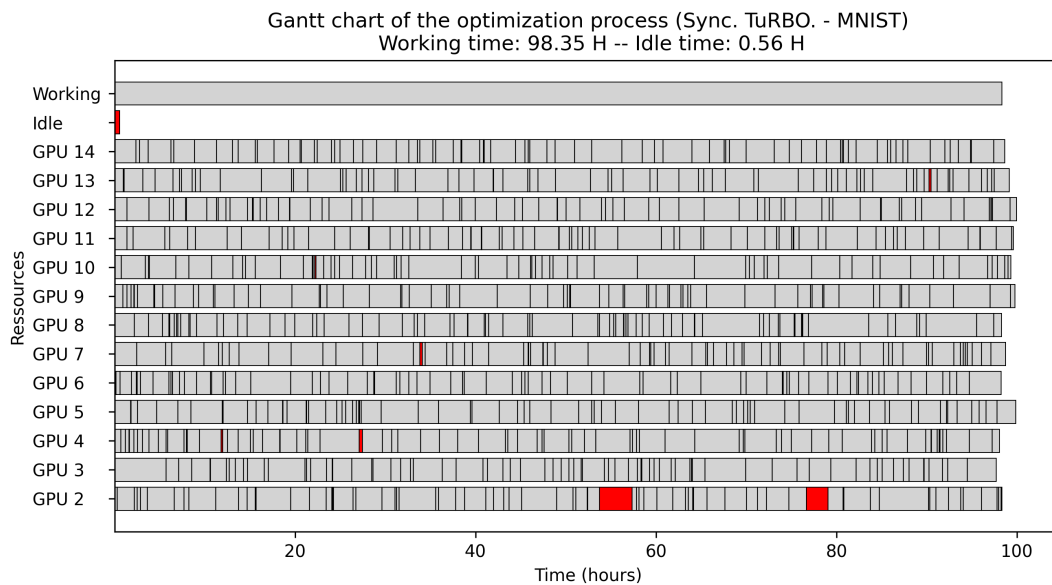
Figure B.1: Sensitivity analysis of S-SLAY-MNIST. Upper left: number of silent SNNs according to α and β . Upper right: number of evaluated SNNs according to α and β . Lower left: proportion of sampled silent networks according to α and β . Lower right: proportion of the budget spent on computing silent networks according to α and β

Analysis of the asynchronous parallelization.

In this appendix we analyse how the asynchronous parallelization of SCBO better balance the workload. In the figures C.1 to C.4, we illustrate the load balancing among GPUs using a Gantt chart. We accumulated the total working and idle time of the processes. We compare asynchronous SCBO with the synchronous TuRBO, which have the same performances as discussed in chapter 5. Because of the synchronization barrier in synchronous TuRBO, we clearly see that during the 100 hours experiments, all the GPUs are idle for about 10 hours. Thus, about 150 GPUs hours are lost by doing nothing. Because of the high stochasticity between evaluations of HPs combinations processes have to wait for each other before computing the next iteration of TuRBO. While, considering asynchronous parallelization of SCBO, the idle time of processes is almost equal to 0 hour.

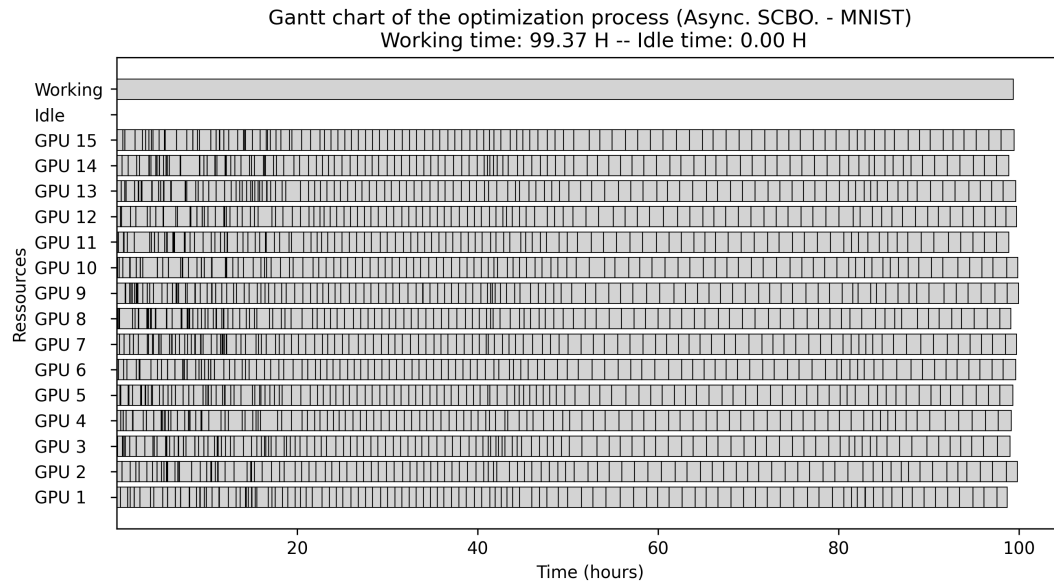


(a) S-SLAY-MNIST

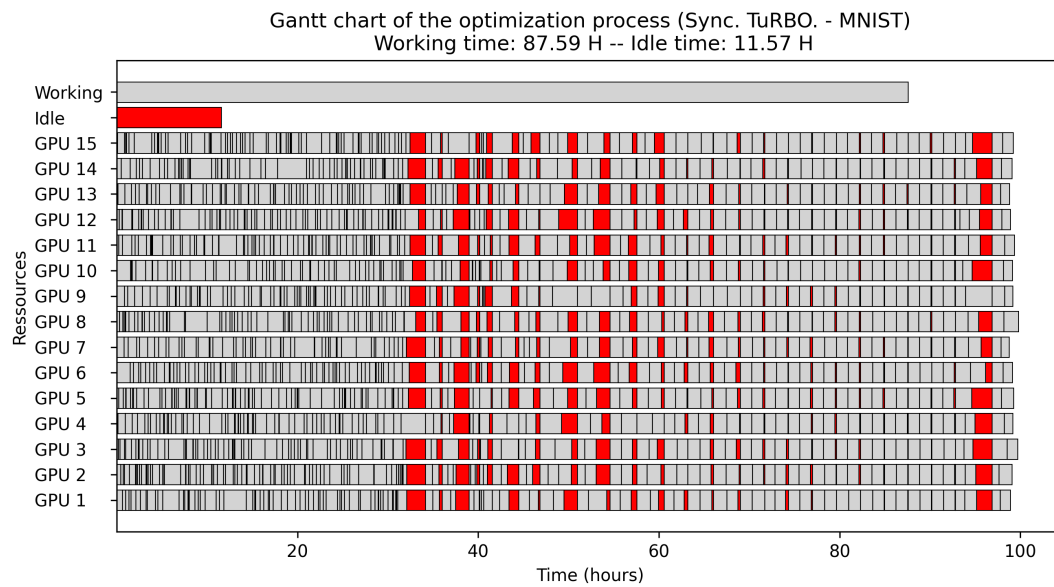


(b) T-SLAY-MNIST

Figure C.1: Gantt chart for experiments on MNIST using LAVA-DL

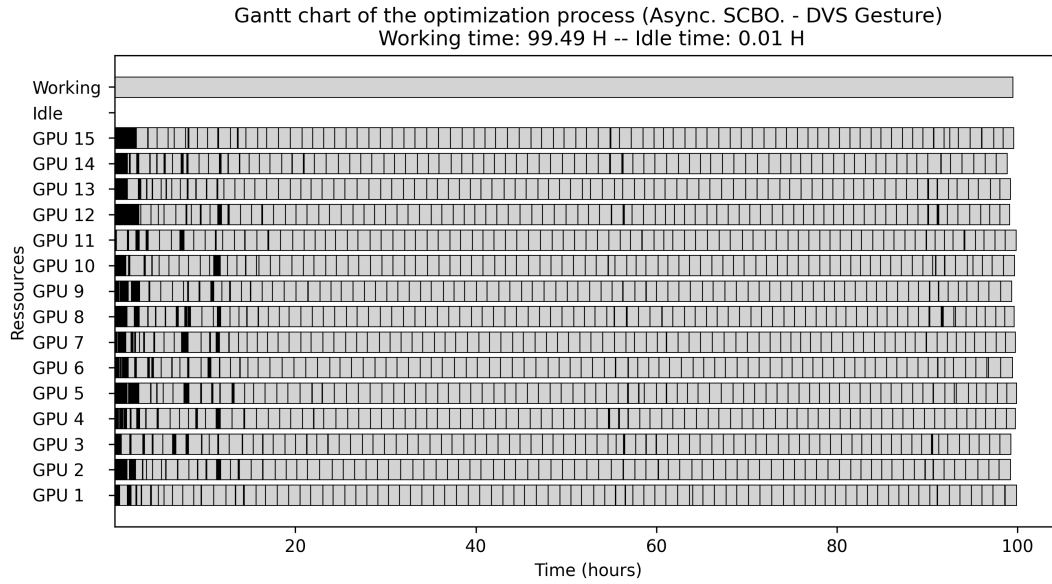


(a) S-SuGr-MNIST

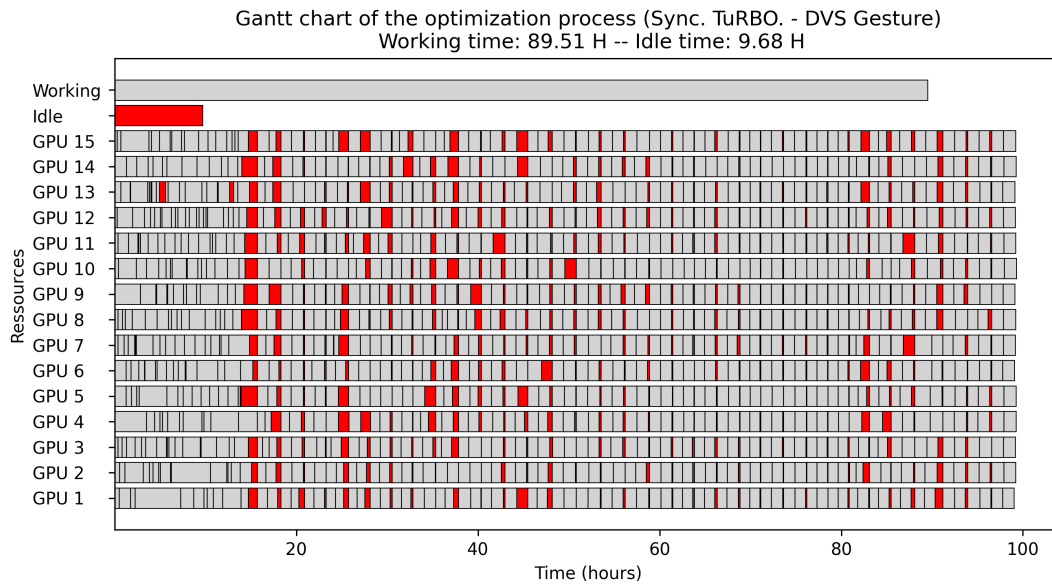


(b) T-SuGr-MNIST

Figure C.2: Gantt chart for experiments on MNIST using SpikingJelly

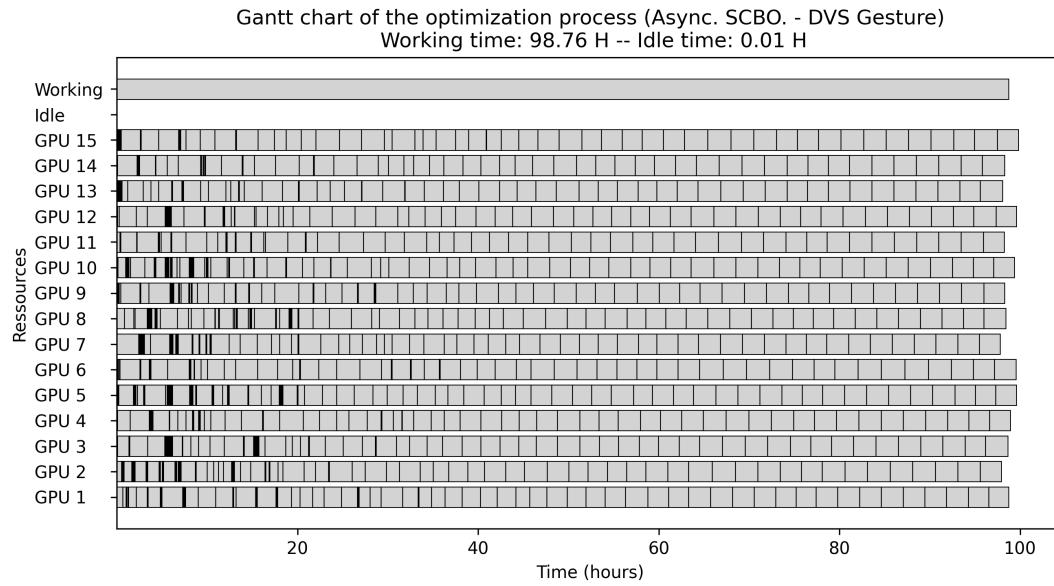


(a) S-SLAY-DVS

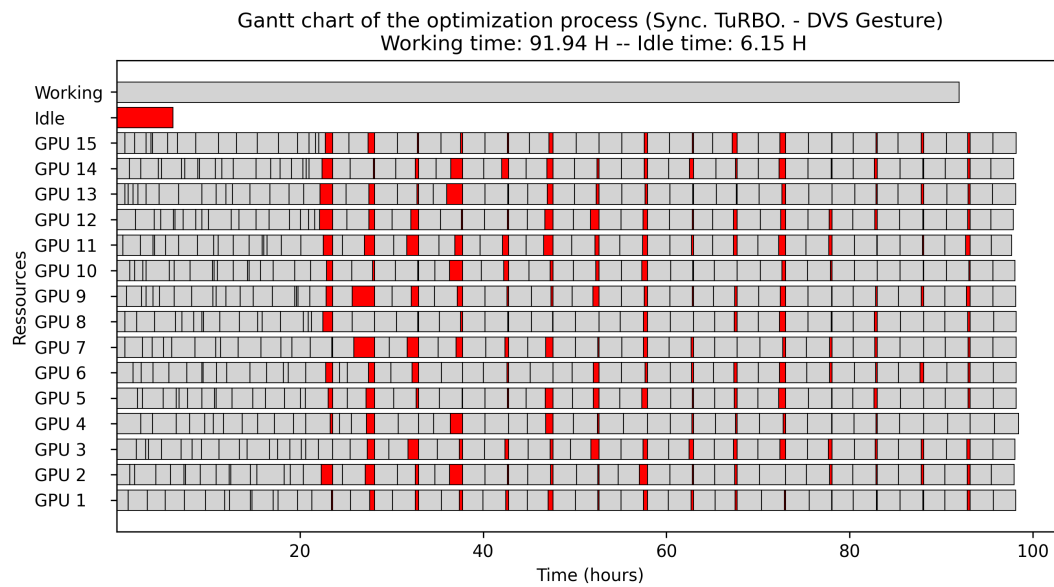


(b) T-SLAY-DVS

Figure C.3: Gantt chart for experiments on DvsGesture using LAVA-DL



(a) S-SuGr-DVS



(b) T-SuGr-DVS

Figure C.4: Gantt chart for experiments on DvsGesture using SpikingJelly

Search spaces of chapter 5

In this appendix we present all search spaces Ω of all experiments from chapter 5, the bounds of the HPs and their optimized values.

Table D.1: HPs of preliminary experiments

Group	HPs	Bounds
G1	$V_{\text{th}}^{(\text{exc})}$	$[-50, -0.05]$
	$t_{\text{ref}}^{(\text{exc})}$	$\llbracket 0, 20 \rrbracket$
	$\theta_{\oplus}^{(\text{exc})}$	$[0.001, 0.5]$
	$\tau_{\theta}^{(\text{exc})}$	$[1e^6, 1e^7]$
	$\tau_{\text{leak}}^{(\text{exc})}$	$[10, 200]$
	$V_{\text{th}}^{(\text{inh})}$	$[-40, -0.1]$
	$t_{\text{ref}}^{(\text{inh})}$	$\llbracket 0, 20 \rrbracket$
	$\tau_{\text{leak}}^{(\text{inh})}$	$[1, 20]$
G2	λ_{pre}	$[1e^{-4}, 1e^{-2}]$
	λ_{post}	$[1e^{-4}, 1e^{-2}]$
G3	$n^{(\text{exc})}$	$\llbracket 30, 2000 \rrbracket$
G4	T	$\llbracket 10, 250 \rrbracket$
	Encoder	$\{\text{Poisson}, \text{TTFS}\}$

Table D.2: S-STDP-MNIST

HP	Bounds	Optimized	Sampler	Type	Group
λ_{post}	$[1e-4, 1e-2]$	0.00084	R-LogUniform	Continuous	G2
λ_{pre}	$[1e-4, 1e-2]$	0.0088	LogUniform	Continuous	G2
$n^{(\text{exc})}$	$\llbracket 20, 2000 \rrbracket$	797	Uniform	Discrete	G3
Decoder	{Average, Max, 2-gram, 3-gram}	Max	Random choice	Categorical	G4
Epochs	$\llbracket 1, 3 \rrbracket$	2	Uniform	Discrete	G5
Norm	$[78.4, 784]$	123.38	Uniform	Continuous	G5
Excitatory layer					
$V_{\text{th}}^{(\text{exc})}$	$[-59, 0]$	-57.6	Uniform	Continuous	G1
$V_{\text{rest}}^{(\text{exc})}$	$[-70, -60]$	-60.8	Uniform	Continuous	G1
$\tau_{\text{leak}}^{(\text{exc})}$	$[5, 5000]$	4166.8	LogUniform	Continuous	G1
$t_{\text{ref}}^{(\text{exc})}$	$\llbracket 0, 20 \rrbracket$	6	Uniform	Discrete	G1
θ_{\oplus}	$[0.001, 0.5]$	0.044	LogUniform	Continuous	G1
τ_{θ}	$[1e6, 1e7]$	2041798	LogUniform	Continuous	G1
$f^{(\text{exc})}$	$[0.5, 500]$	356.9	LogUniform	Continuous	G3
Inhibitory layer					
$V_{\text{th}}^{(\text{inh})}$	$[-40, 0]$	-22.8	Uniform	Continuous	G1
$V_{\text{rest}}^{(\text{inh})}$	$[-60, -45]$	-51	Uniform	Continuous	G1
$\tau_{\text{leak}}^{(\text{inh})}$	$[5, 5000]$	1516.15	LogUniform	Continuous	G1
$t_{\text{ref}}^{(\text{inh})}$	$\llbracket 0, 20 \rrbracket$	19	Uniform	Discrete	G1
$f^{(\text{inh})}$	$[0.5, 500]$	433.6	LogUniform	Continuous	G3
Fixed					
$V_{\text{reset}}^{(\text{exc})}$	-60			Continuous	G1
$V_{\text{reset}}^{(\text{inh})}$	-45			Continuous	G1
τ_{pre}	20			Continuous	G2
τ_{post}	20			Continuous	G2
Early stopping					
HP	Excitatory		Inhibitory	Type	Group
α	5		1	Discrete	G5
β	0.1		0.1	Continuous	G5

Table D.3: S-STDP-DVS

HP	Bounds	Optimized	Sampler	Type	Group
λ_{post}	$[1e-4, 1e-2]$	0.00491	R-LogUniform	Continuous	G2
λ_{pre}	$[1e-4, 1e-2]$	0.00977	LogUniform	Continuous	G2
$n^{(\text{exc})}$	$\llbracket 20, 1000 \rrbracket$	344	Uniform	Discrete	G3
Decoder	{Average, Max, 2-gram, SVM, Log}	Max	Random choice	Categorical	G4
Epochs	$\llbracket 1, 3 \rrbracket$	1	Uniform	Discrete	G5
Norm	$[3276.8, 32768]$	17522.71	Uniform	Continuous	G5
Reset interval	$\llbracket 5, 100 \rrbracket$	20	Uniform	Discrete	G5
Excitatory layer					
$V_{\text{th}}^{(\text{exc})}$	$[-59, 60]$	-27.4	Uniform	Continuous	G1
$V_{\text{rest}}^{(\text{exc})}$	$[-140, -60]$	-111.0	Uniform	Continuous	G1
$\tau_{\text{leak}}^{(\text{exc})}$	$[5, 5000]$	2475.24	LogUniform	Continuous	G1
$t_{\text{ref}}^{(\text{exc})}$	$\llbracket 0, 40 \rrbracket$	11	Uniform	Discrete	G1
θ_{\oplus}	$[0.1, 1]$	0.972	LogUniform	Continuous	G1
τ_{θ}	$[1e6, 1e7]$	16783940	LogUniform	Continuous	G1
$f^{(\text{exc})}$	$[1, 500]$	22.32	Uniform	Continuous	G3
Inhibitory layer					
$V_{\text{th}}^{(\text{inh})}$	$[-40, 40]$	2.81	Uniform	Continuous	G1
$V_{\text{rest}}^{(\text{inh})}$	$[-120, -45]$	-75.85	Uniform	Continuous	G1
$\tau_{\text{leak}}^{(\text{inh})}$	$[5, 5000]$	761.38	LogUniform	Continuous	G1
$t_{\text{ref}}^{(\text{inh})}$	$\llbracket 0, 40 \rrbracket$	36	Uniform	Discrete	G1
$f^{(\text{inh})}$	$[1, 500]$	66.05	Uniform	Continuous	G3
Fixed					
$V_{\text{reset}}^{(\text{exc})}$	-60			Continuous	G1
$V_{\text{reset}}^{(\text{inh})}$	-45			Continuous	G1
τ_{pre}	20			Continuous	G2
τ_{post}	20			Continuous	G2
Early stopping					
HP		Excitatory	Inhibitory	Type	Group
α		1	1	Discrete	G5
β		0.1	0.3	Continuous	G5

Table D.4: S-SLAY-MNIST

HP	Bounds	Optimized	Sampler	Type	Group
V_{th}	$[0.4, 4]$	1.727	Uniform	Continuous	G1
τ_{leak}	$[0.01, 0.2]$	0.0674	LogUniform	Continuous	G1
θ_{\oplus}	$[0.001, 0.25]$	0.089	LogUniform	Continuous	G1
τ_{θ}	$[0.01, 0.5]$	0.327	LogUniform	Continuous	G1
τ_{ref}	$[0.1, 0.99]$	0.388	R-LogUniform	Continuous	G1
τ_u	$[0.1, 0.99]$	0.7100	Uniform	Continuous	G1
λ_{∇}	$[1e-3, 1e-1]$	0.0151	LogUniform	Continuous	G2
$\beta_{\nabla 1}$	$[0.8, 1.0]$	0.86	LogUniform	Continuous	G2
$\beta_{\nabla 2}$	$[0.8, 1.0]$	0.91	R-LogUniform	Continuous	G2
G_{∇}	$[0.1, 1]$	0.52	LogUniform	Continuous	G2
τ_{∇}	$[0.005, 1]$	0.0058	LogUniform	Continuous	G2
ν_{T}	$[0.1, 0.9]$	0.47	LogUniform	Continuous	G4
ν_{F}	$[0.01, 0.09]$	0.059	LogUniform	Continuous	G4
Epochs	$\llbracket 1, 40 \rrbracket$	22	Uniform	Discrete	G5
Batch	$\llbracket 20, 200 \rrbracket$	114	R-LogUniform	Discrete	G5
Dropout	$[0.01, 0.90]$	0.356	LogUniform	Continuous	G5
1st Convolutional layer					
$F_{\text{filter}}^{(1)}$	$\llbracket 1, 128 \rrbracket$	26	Uniform	Discrete	G3
$k^{(1)}$	$\llbracket 4, 12 \rrbracket$	5	Uniform	Discrete	G3
2nd Convolutional layer					
$F_{\text{filter}}^{(2)}$	$\llbracket 1, 128 \rrbracket$	40	Uniform	Discrete	G3
$k^{(2)}$	$\llbracket 4, 12 \rrbracket$	4	Uniform	Discrete	G3
Fixed					
Padding	0			Discrete	G3
Stride	1			Discrete	G3
Dilation	1			Discrete	G3
Pooling	See [247]				G3
Early stopping					
HP	Outputs		Type		Group
α	3		Discrete		G5
β	0.05		Continuous		G5

Table D.5: S-SLAY-DVS

HP	Bounds	Optimized	Sampler	Type	Group
V_{th}	$[0.1, 1]$	0.58	Uniform	Continuous	G1
τ_{leak}	$[0.01, 0.9]$	0.8571	LogUniform	Continuous	G1
θ_{\oplus}	$[0.001, 0.4]$	0.3163	LogUniform	Continuous	G1
τ_{θ}	$[0.01, 0.5]$	0.3163	LogUniform	Continuous	G1
τ_{ref}	$[0.1, 0.99]$	0.1397	R-LogUniform	Continuous	G1
τ_u	$[0.1, 0.99]$	0.5621	Uniform	Continuous	G1
λ_{∇}	$[1e-3, 1e-1]$	0.0769	LogUniform	Continuous	G2
$\beta_{\nabla 1}$	$[0.8, 1.0]$	0.8077	LogUniform	Continuous	G2
$\beta_{\nabla 2}$	$[0.8, 1.0]$	0.8292	R-LogUniform	Continuous	G2
G_{∇}	$[0.1, 1]$	0.7449	LogUniform	Continuous	G2
τ_{∇}	$[0.005, 1]$	0.4073	LogUniform	Continuous	G2
ν_{T}	$[0.1, 0.9]$	0.46	LogUniform	Continuous	G4
ν_{F}	$[0.01, 0.09]$	0.085	LogUniform	Continuous	G4
Epochs	$\llbracket 1, 15 \rrbracket$	14	Uniform	Discrete	G5
Batch	$\llbracket 1, 20 \rrbracket$	6	R-LogUniform	Discrete	G5
Dropout	$[0.01, 0.90]$	0.0198	LogUniform	Continuous	G5
1st Convolutional layer					
$F_{\text{ilter}}^{(1)}$	$\llbracket 1, 36 \rrbracket$	5	Uniform	Discrete	G3
$k^{(1)}$	$\llbracket 4, 48 \rrbracket$	21	Uniform	Discrete	G3
2nd Convolutional layer					
$F_{\text{ilter}}^{(2)}$	$\llbracket 1, 128 \rrbracket$	13	Uniform	Discrete	G3
$k^{(2)}$	$\llbracket 4, 48 \rrbracket$	19	Uniform	Discrete	G3
Dense layer					
$n^{(3)}$	$\llbracket 64, 2048 \rrbracket$	446	Uniform	Discrete	G3
Fixed					
Padding	0			Discrete	G3
Stride	1			Discrete	G3
Dilation	1			Discrete	G3
Pooling	See [247]				G3
Early stopping					
HP	Outputs			Type	Group
α	1			Discrete	G5
β	0.3			Continuous	G5

Table D.6: S-SuGr-MNIST

HP	Bounds	Optimized	Sampler	Type	Group
V_{th}	$[0.1, 4]$	1.0855	Uniform	Continuous	G1
τ_{init}	$[1, 40]$	16.91	LogUniform	Continuous	G1
θ_{\oplus}	$[0.001, 0.25]$	0.1419	LogUniform	Continuous	G1
λ_{∇}	$[1e-3, 1e-1]$	0.0178	LogUniform	Continuous	G2
$\beta_{\nabla 1}$	$[0.8, 1.0]$	0.8025	LogUniform	Continuous	G2
$\beta_{\nabla 2}$	$[0.8, 1.0]$	0.9202	R-LogUniform	Continuous	G2
Epochs	$\llbracket 1, 40 \rrbracket$	27	Uniform	Discrete	G5
Batch	$\llbracket 20, 200 \rrbracket$	127	R-LogUniform	Discrete	G5
Dropout	$[0.01, 0.90]$	0.427	LogUniform	Continuous	G5
1st Convolutional layer					
$F_{\text{filter}}^{(1)}$	$\llbracket 1, 128 \rrbracket$	53	Uniform	Discrete	G3
$k^{(1)}$	$\llbracket 4, 12 \rrbracket$	4	Uniform	Discrete	G3
2nd Convolutional layer					
$F_{\text{filter}}^{(2)}$	$\llbracket 1, 128 \rrbracket$	100	Uniform	Discrete	G3
$k^{(2)}$	$\llbracket 4, 12 \rrbracket$	8	Uniform	Discrete	G3
Fixed					
Padding	0			Discrete	G3
Stride	1			Discrete	G3
Dilation	1			Discrete	G3
Pooling		See [247]			G3
Early stopping					
HP	Outputs		Type		Group
α	3		Discrete		G5
β	0.05		Continuous		G5

Table D.7: S-SuGr-DVS

HP	Bounds	Optimized	Sampler	Type	Group
V_{th}	$[0.1, 4]$	0.8223	Uniform	Continuous	G1
τ_{init}	$[1, 40]$	28.11	LogUniform	Continuous	G1
θ_{\oplus}	$[0.001, 0.25]$	0.1419	LogUniform	Continuous	G1
λ_{∇}	$[1e-3, 1e-1]$	0.0376	LogUniform	Continuous	G2
$\beta_{\nabla 1}$	$[0.8, 1.0]$	0.8510	LogUniform	Continuous	G2
$\beta_{\nabla 2}$	$[0.8, 1.0]$	0.9170	R-LogUniform	Continuous	G2
Epochs	$\llbracket 1, 15 \rrbracket$	14	Uniform	Discrete	G5
Batch	$\llbracket 1, 20 \rrbracket$	3	R-LogUniform	Discrete	G5
Dropout	$[0.01, 0.90]$	0.389	LogUniform	Continuous	G5
1st Convolutional layer					
$F_{\text{ilter}}^{(1)}$	$\llbracket 1, 36 \rrbracket$	12	Uniform	Discrete	G3
$k^{(1)}$	$\llbracket 4, 48 \rrbracket$	4	Uniform	Discrete	G3
2nd Convolutional layer					
$F_{\text{ilter}}^{(2)}$	$\llbracket 1, 36 \rrbracket$	17	Uniform	Discrete	G3
$k^{(2)}$	$\llbracket 4, 48 \rrbracket$	24	Uniform	Discrete	G3
Dense layer					
$n^{(3)}$	$\llbracket 64, 2048 \rrbracket$	697	Uniform	Discrete	G3
Fixed					
Padding	0			Discrete	G3
Stride	1			Discrete	G3
Dilation	1			Discrete	G3
Pooling		See [247]			G3
Early stopping					
HP		Outputs		Type	Group
α		1		Discrete	G5
β		0.3		Continuous	G5

Search spaces of chapter 6

In this appendix we present all search spaces Ω of all experiments from chapter 6. Here, HPs with a superscript indicate that it can be applied to the layer. And convolution layer is denoted by c_i , a average pooling layer by a_i , a linear layer by i , and the ouput layer by o .

Table E.1: 22-C-SLAY-MNIST,46-C-SLAY-MNIST,22-C-SLAY-NMNIST,46-C-SLAY-NMNIST, 46-R-SLAY-MNIST,46-R-SLAY-NMNIST

HP	Bounds	Sampler	Type	Group
π_{train}	[0.1, 1.0]	R-LogUniform	Continuous	G5
Epochs	[[1, 20]]	Uniform	Discrete	G5
Batch	[[30, 300]]	R-LogUniform	Discrete	G5
T	[[10, 50]]	LogUniform	Discrete	G4
$F_{\text{filter}}^{(1,2)}$	[[1, 48]]	Uniform	Discrete	G3
$k^{(c_1, c_2)}$	[[4, 12]]	Uniform	Discrete	G3
ν_{T}	[0.1, 0.9]	LogUniform	Continuous	G2
ν_{F}	[0.01, 0.09]	LogUniform	Continuous	G2
τ_{∇}	[0.005, 1]	LogUniform	Continuous	G2
G_{∇}	[0.1, 1]	LogUniform	Continuous	G2
λ_{∇}	[0.001, 0.1]	LogUniform	Continuous	G2
β_{∇}	[0.8, 0.999]	R-LogUniform	Continuous	G2
γ_{∇}	[0.9, 1.0]	R-LogUniform	Continuous	G2
$V_{\text{th}}^{(c_1, c_2, a_1, a_2, a_3, o)}$	[0.5, 8]	R-LogUniform	Continuous	G1
$\tau_{\theta}^{(1, 2, 3, o)}$	[0.01, 0.5]	LogUniform	Continuous	G1
$\theta_{\oplus}^{(c_1, c_2, a_1, a_2, a_3, o)}$	[0.05, 0.4]	R-LogUniform	Continuous	G1
$\tau_u^{(c_1, c_2, a_1, a_2, a_3, o)}$	[0.05, 0.5]	LogUniform	Continuous	G1
$\tau_{\text{leak}}^{(c_1, c_2, a_1, a_2, a_3, o)}$	[0.01, 0.2]	LogUniform	Continuous	G1
$\tau_{\text{ref}}^{(c_1, c_2, a_1, a_2, a_3, o)}$	[0.05, 0.5]	LogUniform	Continuous	G1
$\text{Dropout}^{(a_1, a_2, a_3)}$	[0.01, 0.90]	LogUniform	Continuous	G5
Early stopping				
HP	Excitatory	Inhibitory	Type	Group
α	5	1	Discrete	G5
β	0.03	0.3	Continuous	G5

Table E.2: 21-C-SLAY-SHD, 42-C-SLAY-SHD and 42-R-SLAY-SHD

HP	Bounds	Sampler	Type	Group
π_{train}	[0.2, 1.0]	R-LogUniform	Continuous	G5
Epochs	[[1, 35]]	Uniform	Discrete	G5
Batch	[[20, 300]]	R-LogUniform	Discrete	G5
T	[[200, 400]]	LogUniform	Discrete	G4
$n^{(1,2,3)}$	[100, 1000]	Uniform	Discrete	G3
ν_{T}	[0.1, 0.9]	LogUniform	Continuous	G2
ν_{F}	[0.01, 0.09]	LogUniform	Continuous	G2
τ_{∇}	[0.005, 1]	LogUniform	Continuous	G2
G_{∇}	[0.1, 1]	LogUniform	Continuous	G2
λ_{∇}	[0.001, 0.1]	LogUniform	Continuous	G2
$\beta_{\nabla 1}$	[0.8, 0.999]	LogUniform	Continuous	G2
$\beta_{\nabla 2}$	[0.8, 0.999]	R-LogUniform	Continuous	G2
$V_{\text{th}}^{(1,2,3,o)}$	[0.5, 10]	R-LogUniform	Continuous	G1
$\tau_{\theta}^{(1,2,3,o)}$	[0.01, 0.5]	LogUniform	Continuous	G1
$\theta_{\oplus}^{(1,2,3,o)}$	[0.05, 0.5]	R-LogUniform	Continuous	G1
$\tau_u^{(1,2,3,o)}$	[0.05, 0.5]	LogUniform	Continuous	G1
$\tau_{\text{leak}}^{(1,2,3,o)}$	[0.01, 0.2]	LogUniform	Continuous	G1
$\tau_{\text{ref}}^{(1,2,3,o)}$	[0.05, 0.5]	LogUniform	Continuous	G1
Dropout $^{(a_1, a_2, a_3, o)}$	[0.01, 0.90]	LogUniform	Continuous	G5
Early stopping				
HP	Excitatory	Inhibitory	Type	Group
α	10	1	Discrete	G5
β	0.1	0.3	Continuous	G5

Table E.3: 15-C-SuGr-MNIST,46-C-SuGr-MNIST,15-C-SuGr-NMNIST,21-C-SuGr-NMNIST,21-R-SuGr-MNIST and 21-R-SuGr-NMNIST

HP	Bounds	Sampler	Type	Group
π_{train}	[0.1, 1.0]	R-LogUniform	Continuous	G5
Epochs	[[1, 20]]	Uniform	Discrete	G5
Batch	[[20, 200]]	R-LogUniform	Discrete	G5
T	[[10, 50]]	LogUniform	Discrete	G4
$F_{\text{filter}}^{(1,2)}$	[[1, 128]]	Uniform	Discrete	G3
$k^{(c_1, c_2)}$	[[4, 12]]	Uniform	Discrete	G3
λ_{∇}	[0.001, 0.1]	LogUniform	Continuous	G2
$\alpha_{\nabla}^{(1,2,o)}$	[2, 10]	Uniform	Continuous	G2
β_{∇}	[0.8, 0.999]	R-LogUniform	Continuous	G2
γ_{∇}	[0.9, 1.0]	R-LogUniform	Continuous	G2
$V_{\text{th}}^{(1,2,o)}$	[0.1, 4]	LogUniform	Continuous	G1
$\tau_{\text{init}}^{(1,2,o)}$	[1, 40]	LogUniform	Continuous	G1
Dropout	[0.01, 0.90]	LogUniform	Continuous	G5
Early stopping				
HP	Excitatory	Inhibitory	Type	Group
α	5	1	Discrete	G5
β	0.03	0.3	Continuous	G5

Table E.4: 15-C-SuGr-SHD, 21-C-SuGr-SHD and 21-R-SuGr-SHD

HP	Bounds	Sampler	Type	Group
π_{train}	$[0.15, 1.0]$	R-LogUniform	Continuous	G5
Epochs	$\llbracket 1, 35 \rrbracket$	Uniform	Discrete	G5
Batch	$\llbracket 20, 300 \rrbracket$	R-LogUniform	Discrete	G5
T	$\llbracket 200, 400 \rrbracket$	LogUniform	Discrete	G4
$n^{(1,2,3)}$	$\llbracket 50, 1000 \rrbracket$	LogUniform	Discrete	G3
λ_{∇}	$[0.001, 0.1]$	LogUniform	Continuous	G2
$\alpha_{\nabla}^{(1,2,3,o)}$	$[2, 10]$	Uniform	Continuous	G2
$\beta_{1\nabla}$	$[0.8, 0.999]$	LogUniform	Continuous	G2
$\beta_{2\nabla}$	$[0.8, 0.999]$	R-LogUniform	Continuous	G2
$V_{\text{th}}^{(1,2,3,o)}$	$[0.1, 2]$	R-LogUniform	Continuous	G1
$\tau_{\text{init}}^{(1,2,3,o)}$	$[1, 40]$	LogUniform	Continuous	G1
Dropout $^{(1,2,3)}$	$[0.01, 0.90]$	LogUniform	Continuous	G5
Early stopping				
HP	Excitatory	Inhibitory	Type	Group
α	1	1	Discrete	G5
β	0.2	0.3	Continuous	G5

Table E.5: Optimized HPs for 22-C-SLAY-MNIST and 22-C-SLAY-NMNIST.

HP	22-C-SLAY-MNIST	22-C-SLAY-NMNIST
π_{train}	0.9980	0.9716
T	47	35
Epochs	5	17
Batch	31	27
ν_{T}	0.7945	0.8114
ν_{F}	0.0656	0.0853
$F_{\text{filter}}^{(1)}$	34	20
$F_{\text{filter}}^{(2)}$	19	32
$k^{(c_1)}$	4	4
$k^{(c_2)}$	5	4
λ_{∇}	0.0335	0.0172
β_{∇}	0.9156	0.9333
γ_{∇}	0.9408	0.9468
τ_{∇}	0.0314	0.2552
G_{∇}	0.5481	0.6122
V_{th}	6.0349	1.6787
θ_{\oplus}	0.1616	0.3283
τ_u	0.1446	0.3081
τ_{leak}	0.0791	0.01288
τ_{θ}	0.1057	0.2304
τ_{ref}	0.3211	0.1328
Dropout	0.1712	0.1378

Table E.6: Optimized HPs for 46-C-SLAY-MNIST, 46-C-SLAY-NMNIST, 46-R-SLAY-MNIST and 46-R-SLAY-NMNIST

HP	46-C-SLAY-MNIST	46-C-SLAY-NMNIST	46-R-SLAY-MNIST	46-R-SLAY-NMNIST
π_{train}	0.8129	0.9974	0.8327	0.9901
T	45	36	48	39
Epochs	19	8	15	15
Batch	207	120	290	32
ν_{T}	0.8546	0.7561	0.5110	0.5526
ν_{F}	0.0227	0.0101	0.0502	0.0163
$F_{\text{ilter}}^{(1)}$	32	40	28	34
$F_{\text{ilter}}^{(2)}$	44	34	45	47
$k^{(c_1)}$	4	4	7	4
$k^{(c_2)}$	5	5	7	9
λ_{∇}	0.0507	0.0569	0.0481	0.0043
β_{∇}	0.8853	0.8333	0.9193	0.9658
γ_{∇}	0.9829	0.9869	0.9745	0.9926
τ_{∇}	0.0085	0.0065	0.5646	0.3314
G_{∇}	0.5937	0.3181	0.7996	0.1992
$V_{\text{th}}^{(c_1)}$	5.7737	0.6141	1.0663	0.6598
$\theta_{\oplus}^{(c_1)}$	0.3850	0.1612	0.0804	0.1446
$\tau_u^{(c_1)}$	0.1165	0.1205	0.2442	0.0682
$\tau_{\text{leak}}^{(c_1)}$	0.1907	0.0144	0.0444	0.0354
$\tau_{\theta}^{(c_1)}$	0.2594	0.0101	0.3344	0.3510
$\tau_{\text{ref}}^{(c_1)}$	0.3564	0.1593	0.0638	0.1964
Dropout $^{(c_1)}$	0.0147	0.0276	0.4312	0.2249
$V_{\text{th}}^{(c_2)}$	4.0787	5.0541	6.9742	3.0413
$\theta_{\oplus}^{(c_2)}$	0.3219	0.2588	0.0683	0.1438
$\tau_u^{(c_2)}$	0.4200	0.1635	0.0593	0.1569
$\tau_{\text{leak}}^{(c_2)}$	0.1726	0.1449	0.0239	0.1838
$\tau_{\theta}^{(c_2)}$	0.3021	0.1738	0.3776	0.3900
$\tau_{\text{ref}}^{(c_2)}$	0.1156	0.1681	0.2708	0.1758
Dropout $^{(c_2)}$	0.5381	0.0276	0.7616	0.2092
$V_{\text{th}}^{(a_1)}$	0.9959	7.8059	2.5748	4.3888
$\theta_{\oplus}^{(a_1)}$	0.3350	0.3625	0.1796	0.3996
$\tau_u^{(a_1)}$	0.4860	0.3943	0.3185	0.3978
$\tau_{\text{leak}}^{(a_1)}$	0.0830	0.0392	0.1968	0.1194
$\tau_{\theta}^{(a_1)}$	0.3243	0.1786	0.2354	0.2558
$\tau_{\text{ref}}^{(a_1)}$	0.4394	0.4677	0.1907	0.4713
$V_{\text{th}}^{(a_2)}$	3.0561	3.1575	4.4542	2.1897
$\theta_{\oplus}^{(a_2)}$	0.3304	0.3676	0.2252	0.3095
$\tau_u^{(a_2)}$	0.1401	0.1724	0.4450	0.1319
$\tau_{\text{leak}}^{(a_2)}$	0.0770	0.0796	0.1225	0.1673
$\tau_{\text{ref}}^{(a_2)}$	0.1239	0.0223	0.3727	0.3043
$\tau_{\theta}^{(a_2)}$	0.0583	0.1806	0.2357	0.1477
$V_{\text{th}}^{(o)}$	7.4354	4.3996	3.1200	6.7648
$\theta_{\oplus}^{(o)}$	0.2106	0.3537	0.2244	0.1225
$\tau_u^{(o)}$	0.2735	0.4544	0.0853	0.1356
$\tau_{\text{leak}}^{(o)}$	0.1510	0.1120	0.1264	0.1721
$\tau_{\theta}^{(o)}$	0.2587	0.2669	0.3257	0.3871
$\tau_{\text{ref}}^{(o)}$	0.3480	0.2498	0.1404	0.3081

Table E.7: Optimized HPs for 21-C-SLAY-SHD.

HP	21-C-SLAY-SHD
π_{train}	0.99239
T	218
Epochs	23
Batch	23
ν_{T}	0.83802
ν_{F}	0.08633
$n^{(1)}$	812
$n^{(2)}$	916
$n^{(3)}$	756
λ_{∇}	0.03604
$\beta_{\nabla 1}$	0.92042
$\beta_{\nabla 2}$	0.97725
τ_{∇}	0.20942
G_{∇}	0.59046
V_{th}	1.84118
θ_{\oplus}	0.14305
τ_u	0.19730
τ_{leak}	0.19234
τ_{θ}	0.05070
τ_{ref}	0.07131
Dropout	0.05741

Table E.8: Optimized HPs for 42-C-SLAY-SHD and 42-R-SLAY-SHD.

HP	42-C-SLAY-SHD	42-R-SLAY-SHD
π_{train}	0.9879	0.5883
T	373	297
Epochs	33	30
Batch	28	50
ν_{T}	0.8237	0.3272
ν_{F}	0.0188	0.0265
$n^{(1)}$	481	511
$n^{(2)}$	400	865
$n^{(3)}$	500	523
λ_{∇}	0.02592	0.0859
$\beta_{\nabla 1}$	0.86872	0.8953
$\beta_{\nabla 2}$	0.94445	0.9403
τ_{∇}	0.01319	0.1442
G_{∇}	0.94366	0.9937
$V_{\text{th}}^{(1)}$	2.48522	3.0269
$\theta_{\oplus}^{(1)}$	0.37921	0.3187
$\tau_u^{(1)}$	0.36796	0.1685
$\tau_{\text{leak}}^{(1)}$	0.14137	0.1445
$\tau_{\theta}^{(1)}$	0.38948	0.1554
$\tau_{\text{ref}}^{(1)}$	0.08658	0.3111
Dropout ⁽¹⁾	0.02790	0.3502
$V_{\text{th}}^{(2)}$	5.85936	7.9735
$\theta_{\oplus}^{(2)}$	0.39584	0.2537
$\tau_u^{(2)}$	0.28018	0.4443
$\tau_{\text{leak}}^{(2)}$	0.19229	0.1637
$\tau_{\theta}^{(2)}$	0.21494	0.0721
$\tau_{\text{ref}}^{(2)}$	0.44235	0.2434
Dropout ⁽²⁾	0.15651	0.0130
$V_{\text{th}}^{(3)}$	4.34834	3.1136
$\theta_{\oplus}^{(3)}$	0.25340	0.0812
$\tau_u^{(3)}$	0.45007	0.1472
$\tau_{\text{leak}}^{(3)}$	0.08682	0.1552
$\tau_{\theta}^{(3)}$	0.27034	0.2326
$\tau_{\text{ref}}^{(3)}$	0.49979	0.4342
Dropout ⁽³⁾	0.16562	0.6449
$V_{\text{th}}^{(o)}$	7.07539	5.3926
$\theta_{\oplus}^{(o)}$	0.12721	0.1460
$\tau_u^{(o)}$	0.41306	0.3875
$\tau_{\text{leak}}^{(o)}$	0.03961	0.1626
$\tau_{\theta}^{(o)}$	0.39940	0.1979
$\tau_{\text{ref}}^{(o)}$	0.12302	0.0860

Table E.9: Optimized HPs for 15-C-SuGr-MNIST and 15-C-SuGr-NMNIST.

HP	15-C-SuGr-MNIST	15-C-SuGr-NMNIST
π_{train}	0.9606	0.9707
T	14	29
Epochs	12	7
Batch	20	24
$F_{\text{filter}}^{(1)}$	43	28
$F_{\text{filter}}^{(2)}$	30	28
$k^{(c_1)}$	7	4
$k^{(c_2)}$	5	11
λ_{∇}	0.0359	0.0825
β_{∇}	0.9915	0.9836
γ_{∇}	0.9700	0.9873
V_{th}	0.2770	0.4731
τ_{init}	8.4894	3.6402
Dropout	0.4552	0.0301
α_{∇}	6.6500	9.6322

Table E.10: Optimized HPs for 21-C-SuGr-MNIST, 21-C-SuGr-NMNIST, 21-R-SuGr-MNIST and 21-R-SuGr-NMNIST

HP	21-C-SuGr-MNIST	21-C-SuGr-NMNIST	21-R-SuGr-MNIST	21-R-SuGr-NMNIST
π_{train}	0.9356066393	0.8861	0.5495	0.8155
T	33	16	25	31
Epochs	16	7	17	14
Batch	24	23	20	84
$F_{\text{ilter}}^{(1)}$	44	41	36	37
$F_{\text{ilter}}^{(2)}$	47	31	26	26
$k^{(c_1)}$	5	4	6	7
$k^{(c_2)}$	6	7	5	7
λ_{∇}	0.0817	0.0801	0.0927	0.0597
β_{∇}	0.9887	0.9822	0.9421	0.9836
γ_{∇}	0.9881	0.9252	0.9114	0.9810
$V_{\text{th}}^{(1)}$	0.4475	0.9799	0.7130	2.8845
$\tau_{\text{init}}^{(1)}$	23.7849	5.0497	7.4296	1.8623
$\alpha_{\nabla}^{(1)}$	7.5882	8.3793	8.5698	6.8077
$V_{\text{th}}^{(2)}$	1.0771	0.7824	1.4678	0.7025
$\tau_{\text{init}}^{(2)}$	2.2235	3.6341	16.4193	28.1352
$\alpha_{\nabla}^{(2)}$	8.6832	5.6028	7.5396	4.7738
$V_{\text{th}}^{(o)}$	0.6145	0.1828	0.1519	0.5309
$\tau_{\text{init}}^{(o)}$	15.0652	3.8413	16.0378	6.7250
$\alpha_{\nabla}^{(o)}$	5.3247	5.8989	5.1481	4.7587
Dropout	0.4762	0.2472	0.0842	0.7477

Table E.11: Optimized HPs for 13-C-SuGr-SHD.

HP	13-C-SuGr-SHD
π_{train}	0.9603
T	392
Epochs	30
Batch	110
λ_{∇}	0.0844
$\beta_{\nabla 1}$	0.8120
$\beta_{\nabla 2}$	0.9879
$f^{(1)}$	671
$f^{(2)}$	708
$f^{(3)}$	834
V_{th}	0.9289
τ_{init}	36.0365
Dropout	0.0681

Table E.12: Optimized HPs for 21-C-SuGr-SHD and 21-R-SuGr-SHD.

HP	21-C-SuGr-SHD	21-R-SuGr-SHD
π_{train}	0.9998	0.9853
T	316	379
Epochs	24	11
Batch	25	24
λ_{∇}	0.0548	0.0197
$\beta_{\nabla 1}$	0.8211	0.9182
$\beta_{\nabla 2}$	0.9408	0.9114
$f^{(1)}$	670	843
$f^{(2)}$	443	410
$f^{(3)}$	625	145
$V_{\text{th}}^{(1)}$	0.7758	1.1965
$\tau_{\text{init}}^{(1)}$	22.5689	4.1225
Dropout ⁽¹⁾	0.0255	0.2998
$V_{\text{th}}^{(2)}$	1.1974	0.9027
$\tau_{\text{init}}^{(2)}$	28.6416	15.6550
Dropout ⁽²⁾	0.1092	0.1864
$V_{\text{th}}^{(3)}$	0.7724	1.7210
$\tau_{\text{init}}^{(3)}$	32.4351	5.9083
Dropout ⁽³⁾	0.5234	0.0763
$V_{\text{th}}^{(o)}$	1.1812	1.9450
$\tau_{\text{init}}^{(o)}$	16.8020	32.7867

Description of all hyperparameters

In this appendix we describe all optimized hyperparameters.

Table F.1: Hyperparameter descriptions

HP	Description	Group
V_{th}	Neuron threshold.	G1
V_{rest}	Neuron resting potential.	G1
V_{reset}	Neuron reset potential.	G1
t_{ref}	Refractory period in time steps.	G1
θ_{\oplus}	Added value to the threshold, for threshold adaptation mechanism.	G1
τ_{leak}	Neuron leakage time constant.	G1
τ_{init}	Initial Neuron leakage time constant for PLIF.	G1
τ_{θ}	Threshold adaptation time constant. Makes the threshold decrease through time.	G1
τ_{ref}	Refractory period time constant for SLAYER.	G1
τ_u	Time constant for current decay.	G1
$(\tau_{pre}, \tau_{post})$	Traces time constant of STDP.	G2
$(\lambda_{pre}, \lambda_{post})$	Learning rate of pre- and post- synaptic spikes in STDP.	G2
λ_{∇}	Learning rate of gradient descent.	G2
$(\beta_{\nabla 1}, \beta_{\nabla 2})$	Momentum of ADAM.	G2
τ_{∇}	Relaxation of the spike function gradient in SLAYER.	G2
G_{∇}	Controls the gradient flow across layers, and handles vanishing or exploding gradient in SLAYER.	G2
α_{∇}	Quality of the gradient approximation in SpikingJelly.	G2
$n^{(i)}$	Number of neurons for layer i	G3
$F_{filter}^{(i)}$	Number of filters in a convolution layer i .	G3
$k^{(i)}$	Size of filters in a convolution layer i .	G3
Padding	Padding HP for convolution layer i .	G3
Stride	Stride HP for convolution layer i .	G3
Dilation	Dilation HP for convolution layer i .	G3
ν_T	Output spiking rate for true neurons in SLAYER	G4
ν_F	Output spiking rate for false neurons in SLAYER	G4
Decoder	Algorithm used to decode outputs of Hebbian-trained SNNs.	G4
T	Encoding time window.	G4
Epochs	Number of epochs.	G5
Batch	Batch size.	G5
Dropout	Probability of removing neurons and their connections during training.	G5
Norm.	Weight normalization. Value at which the sum of neuron weights must be equal.	G5
Reset interval	Time between two resets of neurons to their initial state.	G5
π_{train}	Proportion of the training dataset used in multi-fidelity optimization.	G5
α	Number of minimum output spikes for early stopping.	G5
β	Proportion of non-spiking outputs for early stopping.	G5

Computer programs

Listing G.1: FDA with Zellij

```
1 from zellij.core import (  
2     ArrayVar,  
3     FloatVar,  
4     Loss,  
5     Experiment,  
6     Threshold,  
7     Minimizer,  
8     IThreshold,  
9 )  
10 from zellij.strategies.fractals import ILS, DBA  
11 from zellij.strategies.fractals.sampling import PHS  
12 from zellij.strategies.tools import Hypersphere, DistanceToTheBest,  
13     MoveUp  
14 from zellij.utils.converters import FloatMinMax, ArrayDefaultC  
15 from zellij.utils.benchmarks import Rosenbrock  
16  
17 dim = 5 # Dimensionality of the problem  
18 fun = Rosenbrock() # Raw objective function  
19  
20 # Wrap the objective function within the Loss object  
21 # to extend its functionalities  
22 lf = Loss(  
23     objective=[Minimizer("obj")], # Define the objective to  
24     optimize  
25 )(fun)  
26  
27 # Define the decision variables as an Array of variables  
28 # Use a converter as we are in [0,1]  
29 values = ArrayVar(converter=ArrayDefaultC())  
30 for i in range(dim): # Define variables for all dimensions  
31     values.append(  
32         FloatVar(  
33             f"float_{i+1}",
```



```

33         fun.lower,
34         fun.upper,
35         converter=FloatMinMax(),
36     )
37 ) # Use a converter for each FloatVar to redefine the range in
    [0,1]
38
39 # Build the search space using previously defined variables
40 # With save_points=True
41 # The search space is a fractal of type Hypersphere
42 # A fractal will "remember" computed points within it
43 sp = Hypersphere(values, save_points=True)
44
45 # Define the exploration as the Potential optimal Hypersphere
    Search
46 explor = PHS(sp, inflation=1.75)
47 # Define the exploitation as the Intensive Local Search
48 exploi = ILS(sp, inflation=1.75)
49
50 # Define the stopping criterion for PHS which computes only 3
    points at each round
51 # if the attribute of PHS "current_calls" > 3, then stop sampling
    with PHS
52 stop1 = Threshold(
53     None, "current_calls", 3
54 ) # set target to None, DBA will automatically assign it.
55
56 # Define the stopping criterion for the ILS
57 # if the attribute of ILS "step" < 1e-16, then stop sampling with
    ILS
58 stop2 = IThreshold(exploi, "step", 1e-16)
59
60 # Define the stopping criterion of the experiments
61 # if the number of calls to the loss function > dim*10^4 then stop
62 stop3 = Threshold(lf, "calls", dim * 10**4)
63
64 # Define the tree search component as MoveUp with a maximum depth
    of 5
65 ts = MoveUp(sp, 5)
66
67 # Combine all component into a DBA algorithm
68 dba = DBA(
69     sp,
70     MoveUp(sp, 5),
71     (explor, stop1),
72     (exploi, stop2),
73     scoring=DistanceToTheBest(lf), # Define the scoring component
74 )
75

```

```
76 # Define the experiments by combining the optimizer, loss function,  
    stopping  
77 exp = Experiment(  
78     dba, lf, stop3, verbose=True, save="testfda"  
79 ) # save results into a folder  
80 exp.run()  
81 print(f"Best solution: {lf({lf.best_point})={lf.best_score}")
```

Listing G.2: SOO with Zellij

```

1  from zellij.core import (
2      ArrayVar,
3      FloatVar,
4      Loss,
5      Experiment,
6      Threshold,
7      Minimizer,
8  )
9  from zellij.strategies.fractals import DBASampling, CenterS00
10 from zellij.strategies.tools import Section, Min, SooTreeSearch
11 from zellij.utils.converters import FloatMinMax, ArrayDefaultC
12 from zellij.utils.benchmarks import Rosenbrock
13
14
15 dim = 5
16 fun = Rosenbrock()
17
18 lf = Loss(objective=[Minimizer("obj")])(fun)
19
20 values = ArrayVar(converter=ArrayDefaultC())
21 for i in range(dim):
22     values.append(
23         FloatVar(
24             f"float_{i+1}",
25             fun.lower,
26             fun.upper,
27             converter=FloatMinMax(),
28         )
29     )
30
31 # The search space is a fractal of type Section
32 # here we are using a 3-section
33 sp = Section(values, section=3)
34
35 # The explore component samples the center of the fractal
36 explor = CenterS00(sp)
37 # Exploitation is optional
38
39 # Experiment stopping criterion
40 stop2 = Threshold(lf, "calls", dim * 10**4)
41
42 # Use DBASampling, a variation of DBA
43 # Allows to sample in one shot all points from
44 # one iteration of explor applied to all selected fractals
45 dba = DBASampling(sp, SooTreeSearch(sp, float("inf")), explor,
46     scoring=Min())
47 exp = Experiment(dba, lf, stop2, save="soo")
48 exp.run()
49 print(f"Best_solution:f({lf.best_point})={lf.best_score}>={fun.

```

```
| optimum*dim} " )
```

Listing G.3: NMSO with Zellij

```

1  from zellij.core import (
2      ArrayVar,
3      FloatVar,
4      Loss,
5      Experiment,
6      Threshold,
7      Minimizer,
8      BooleanStop,
9  )
10 from zellij.strategies.fractals import DBA, CenterS00
11 from zellij.strategies.tools import NMSOSection, Min,
12     NMSOTreeSearch
13 from zellij.utils.converters import FloatMinMax, ArrayDefaultC
14 from zellij.utils.benchmarks import Rosenbrock
15
16 dim = 5
17 fun = Rosenbrock()
18
19 lf = Loss(objective=[Minimizer("obj")])(fun)
20
21 values = ArrayVar(converter=ArrayDefaultC())
22 for i in range(dim):
23     values.append(
24         FloatVar(
25             f"float_{i+1}",
26             fun.lower,
27             fun.upper,
28             converter=FloatMinMax(),
29         )
30     )
31
32 # The search space is a fractal of type Section
33 # here we are using a 3-section
34 sp = NMSOSection(values, section=3)
35
36 # The explore component samples the center of the fractal
37 explor = CenterS00(sp)
38 # Stopping criterion of CenterS00
39 # If it was already computed for a given fractal, then stop
40 # sampling this fractal
41 stop1 = BooleanStop(explor, "computed")
42
43 # Experiment stopping criterion
44 stop2 = Threshold(lf, "calls", dim * 10**4)
45
46 basket = sp.size
47 alpha = 1e-1 * sp.size
48 beta = alpha

```

```
49
50 dba = DBA(
51     sp,
52     NMS0TreeSearch(sp, float("inf"), V=basket, alpha=alpha, beta=
53         beta),
54     (explor, stop1),
55     # Exploitation is optional
56     (None, None),
57     # Score a fractal with the minimum computed value
58     scoring=Min(),
59 )
60 exp = Experiment(dba, lf, stop2, verbose=False)
61 exp.run()
62 print(f"Best_solution:f({lf.best_point})={fun.optimum*dim-lf.
63     best_score}")
```

Listing G.4: DIRECT with Zellij

```

1  from zellij.core import (
2      ArrayVar,
3      FloatVar,
4      Loss,
5      Experiment,
6      Threshold,
7      Minimizer,
8      BooleanStop,
9  )
10 from zellij.strategies.fractals import DBADirect, DirectSampling
11 from zellij.strategies.tools import Direct, Nothing,
12     PotentiallyOptimalRectangle, Sigma2
13 from zellij.utils.converters import FloatMinMax, ArrayDefaultC
14 from zellij.utils.benchmarks import Rosenbrock
15
16 dim = 5
17 fun = Rosenbrock()
18 lf = Loss(objective=[Minimizer("obj")])(fun)
19
20 values = ArrayVar(converter=ArrayDefaultC())
21 for i in range(dim):
22     values.append(
23         FloatVar(
24             f"float_{i+1}",
25             fun.lower,
26             fun.upper,
27             converter=FloatMinMax(),
28         )
29     )
30
31 # measurement defines how to measure the size
32 # of a Direct fractal
33 sp = Direct(values, measurement=Sigma2())
34
35 explor = DirectSampling(sp)
36 stop1 = BooleanStop(
37     explor, "computed"
38 ) # set target to None, DBA will automatically assign it.
39
40 stop2 = Threshold(lf, "calls", 1000)
41
42 # Use DBADirect algorithm, as in Direct the exploration
43 # is made BEFORE the partitioning
44 dba = DBADirect(
45     sp,
46     PotentiallyOptimalRectangle(sp, 600),
47     (explor, stop1),
48     (None, None),

```

```
49     scoring=Nothing(),
50 )
51 exp = Experiment(dba, lf, stop2, save="test_direct")
52 exp.run()
53 print(f"Best solution: f({lf.best_point})={lf.best_score}")
```


Listing G.5: LHS with Zellij

```

1  from zellij.core import (
2      ArrayVar,
3      FloatVar,
4      Loss,
5      Experiment,
6      Threshold,
7      Minimizer,
8  )
9  from zellij.strategies.fractals.dba import DBALHS
10 from zellij.strategies.tools.scoring import UCB
11 from zellij.strategies.fractals.sampling import Center
12 from zellij.strategies.tools import LatinHypercubeUCB, S00UCB
13 from zellij.utils.converters import FloatMinMax, ArrayDefaultC
14 from zellij.utils.benchmarks import Rosenbrock
15
16 import numpy as np
17
18 #####
19 # No extra ILS HERE see provided code within chapter 4 #
20 # Here we describe the partition-exploration part      #
21 #####
22
23 dim = 5 # Dimensionality of the problem
24 fun = Rosenbrock() # Raw objective function
25
26 # Wrap the objective function within the Loss object
27 # to extend its functionalities
28 lf = Loss(
29     objective=[Minimizer("obj")], # Define the objective to
        optimize
30 )(fun)
31
32 # Define the decision variables as an Array of variables
33 # Use a converter as we are in [0,1]
34 values = ArrayVar(converter=ArrayDefaultC())
35 for i in range(dim): # Define variables for all dimensions
36     values.append(
37         FloatVar(
38             f"float_{i+1}",
39             fun.lower,
40             fun.upper,
41             converter=FloatMinMax(),
42         )
43     ) # Use a converter for each FloatVar to redefine the range in
        [0,1]
44
45
46 eps = 1e-4
47 gsize = 20

```

```
48 nu = 0.5
49
50 eps = 1 / eps
51 levels = int(np.ceil(np.log(eps) / np.log(gsize)))
52 if 1 / gsize**levels < eps:
53     levels -= 1
54
55 sp = LatinHypercubeUCB(values, levels, grid_size=gsize, strength=1)
56
57 sample = Center(sp)
58
59 dba = DBALHS(sp, S00UCB(sp, levels, nu=nu), sample, scoring=UCB())
60
61 stop3 = Threshold(lf, "calls", dim * 10**4)
62 exp = Experiment(
63     dba, lf, stop3, verbose=True, save="testlhs"
64 ) # save results into a folder
65 exp.run()
66 print(f"Best solution: {lf({lf.best_point})={lf.best_score}")
```


Contents

Abstract	xv
Remerciements	xvii
Extended thesis abstract	xix
Résumé étendu de la thèse	xxi
Contents	xxiii
Glossary	xxv
List of symbols	xxix
List of Tables	xxxi
List of Figures	xxxiii
1 Introduction to Spiking Neural Networks	1
1.1 Beyond von Neumann and silicon	1
1.2 Neuromorphic hardware	3
1.2.1 Neuromorphic simulator	4
1.3 Machine Learning and Artificial Neural Networks	5
1.3.1 Breaking the linearity	7
1.3.2 Backpropagation of the errors	9
1.3.3 Architectures and topology	11
1.3.4 Usual challenges and remarks	13
1.4 Toward Spiking Neural Networks	15
1.4.1 Information coding	15
1.4.2 Neuron model	16
1.4.3 Training spiking neural networks	17
1.4.4 Topologies of spiking neural networks	19
1.5 Motivations	20
1.5.1 Research question	21
1.6 Outline	21
2 Automated machine learning and spiking neural networks	23
2.1 Hyperparameter Optimization	23
2.1.1 Formal definition	24
2.1.2 Neural Architecture Search	30
2.1.3 Advantages, common issues and remarks	31
2.2 Hyperparameter Optimization of Spiking Neural Networks	33

2.2.1	Common hyperparameters	33
2.3	Related works	43
2.3.1	STDP-based SNNs	43
2.3.2	Gradient-based SNNs	45
2.3.3	Other approaches	47
2.3.4	Discussion and remarks	49
3	Algorithms for Hyperparameter Optimization	53
3.1	Grid search and random search	54
3.2	Metaheuristic approaches	55
3.2.1	Simulated Annealing	56
3.2.2	Evolutionary Algorithms	57
3.3	Surrogate Model-Based Optimization	59
3.3.1	Bayesian modeling	59
3.3.2	Gaussian processes	61
3.3.3	Inferring with a GP	62
3.3.4	GPs training via marginal likelihood	64
3.3.5	Numerical instability and computational complexity	65
3.3.6	Kernel functions	65
3.3.7	Acquisition functions	66
3.3.8	Toward multi-fidelity Bayesian optimization for HPO	70
3.4	Parallel HPO	76
3.4.1	Parallel Grid Search and Random Search	77
3.4.2	Parallel Simulated Annealing	77
3.4.3	Parallel Genetic Algorithm	79
3.4.4	Parallel Gaussian Process-based Bayesian Optimization	79
3.4.5	Parallel multi-fidelity HPO	80
4	Partition-based global optimization	81
4.1	Fractal-based decomposition algorithms	81
4.2	Preliminaries	82
4.3	Background and related works	85
4.3.1	Piyavskii and Shubert algorithm	85
4.3.2	DIRECT: Dividing Rectangles	86
4.3.3	DOO, SOO, NMSO: Optimistic Optimization	88
4.3.4	FRACTOP, FDA and PolyFRAC: FBD metaheuristics	89
4.3.5	Frameworks for partition-based algorithms	90
4.4	Zellij: An algorithmic framework for FBD algorithms	91
4.4.1	Geometrical fractal object	92
4.4.2	Tree search	101
4.4.3	Scoring fractals	105
4.4.4	Exploration and exploitation strategies	106
4.5	Instantiating algorithms within Zellij	108
4.5.1	FDA	108
4.5.2	SOO and NMSO	109
4.5.3	DIRECT	111
4.6	Experimental setup	114
4.7	Results analysis and discussion	116
4.7.1	Sensitivity to the dimensionality	116

4.7.2 Sensitivity to the tree search	116
4.7.3 Sensitivity to function properties	119
4.8 Fractal decomposition based on Latin Hypercubes	123
4.8.1 Latin Hypercube Sampling	123
4.8.2 Nested Latin Hypercube Partition	123
4.8.3 Other components	126
4.8.4 Experimentation	127
4.9 Conclusion	132
5 Silent networks: a vicious trap for Hyperparameter Optimization	135
5.1 Preliminary experiments	135
5.2 Infeasible solutions, early stopping and black-box constraints	138
5.2.1 Silent networks and infeasible solutions	138
5.2.2 Early stopping	139
5.2.3 Black-box constraints	141
5.3 Experimental setup	142
5.3.1 High dimensional and constrained Bayesian Optimization	142
5.3.2 Search spaces definition	142
5.3.3 Simulators and datasets	144
5.3.4 Hardware and software specifications	145
5.3.5 Algorithmic details	145
5.4 Computational results on large-scale experiments	146
5.4.1 Analysis of the HPO process	146
5.4.2 Analysis of the best solutions found	149
5.4.3 Analysis of the hyperparameters	151
5.5 Ablation studies, sensitivity analysis and discussions	156
5.5.1 Ablation studies	156
5.6 Conclusion	162
6 Accelerating Hyperparameter Optimization with Multi-Fidelity	163
6.1 Multi-fidelity and spiking neural networks	164
6.1.1 Improved early stopping and constraints	164
6.1.2 Cost aware Bayesian optimization	165
6.1.3 Cost aware Thompson sampling	167
6.2 Experimental setup	168
6.3 Results and discussion	171
6.3.1 Analysis of the best solutions	171
6.3.2 Analysis of CASCBO	173
6.3.3 Comparison with Random Search	179
6.4 Conclusion	184
Main conclusion	185
Thesis contributions	185
Future works	188
Acknowledgments	189
Code availability	189
Bibliography	191

A Trust Region and Scalable Constrained Bayesian Optimization	217
A.1 Trust Region Bayesian Optimization	217
A.2 Scalable Constrained Bayesian Optimization	218
A.3 SCBO with asynchronous trust regions	219
B Sensitivity analysis on the stopping criterion	223
C Analysis of the asynchronous parallelization.	225
D Search spaces of chapter 5	231
E Search spaces of chapter 6	239
F Description of all hyperparameters	253
G Computer programs	255
Contents	267

Abstract

Artificial Neural Networks (ANNs) are a machine learning technique that has become indispensable. By learning from data, ANNs make it possible to solve certain complex cognitive tasks. Over the last three decades, ANNs have seen numerous major advances. These advances have enabled the development of image recognition, large language models, or text-to-image conversion. Undeniably, ANNs have become an invaluable tool for many applications, and this growing interest led in 2020 to the boom of generative models. However, several new barriers could put the brakes on the interest in these models. The first brake is the end of Moore's Law, due to the physical limits reached by transistors. But also, while research has long focused on the predictive performances of ANNs, other aspects have been neglected. These include energy efficiency, robustness, security, interpretability, transparency and so on^a. This is why we need to go beyond von Neumann architectures for reducing the energy footprint, and the neuromorphic approach is a serious breakthrough candidate through biomimicry of the human brain via Spiking Neural Networks (SNNs).

Unfortunately, SNNs are currently struggling to outperform conventional methods. As they are more recent and therefore less studied, a better approach to their design could make it possible to combine performance and low-energy cost. That is why the automatic design of SNNs is studied within this thesis, with a focus on HyperParameter Optimization (HPO). The aim is to improve the HPO algorithms and to better understand the behavior of SNNs regarding their hyperparameters.

Keywords: spiking neural networks, hyperparameter optimization, global optimization, parallel computing, decomposition-based optimization, bayesian optimization

OPTIMISATION PARALLÈLE DES HYPERPARAMÈTRES DES RÉSEAUX IMPULSIONNELS

Résumé

Les Réseaux de Neurones Artificiels (RNAs) sont une technique d'apprentissage machine devenue aujourd'hui incontournable, permettant de résoudre certaines tâches cognitives complexes par un apprentissage automatique. Depuis ces trois dernières décennies, les RNAs ont connu de nombreuses avancées majeures. Ces avancées ont permis le développement de la reconnaissance d'images, des modèles de langage géants ou de la conversion texte-image. Indéniablement, les RNAs sont devenus un outil précieux ayant mené, depuis 2020, au boom des modèles générationnels. Cependant, certaines barrières pourraient freiner l'intérêt pour ces modèles. Notamment, la fin de la loi de Moore, due aux limites physiques atteintes par les transistors. Mais, tandis que la recherche s'est longtemps concentrée sur les performances prédictives des RNAs, d'autres aspects ont été négligés. C'est le cas de l'efficacité énergétique, mais également de la robustesse, de la sécurité, de l'interprétabilité, de la transparence, etc^a. Il faut donc aller au-delà des architectures de von Neumann afin de réduire l'empreinte énergétique, et l'approche neuromorphique est un candidat de rupture sérieux utilisant le biomimétisme du cerveau via des Réseaux de Neurones à Impulsions (RNIs).

Toutefois, les RNIs peinent à surpasser les performances des RNAs. Les RNIs étant plus récents, et donc moins étudiés, une meilleure approche de leur conception pourrait permettre d'allier performances et faible coût énergétique. C'est pourquoi la conception automatique des RNIs est étudiée dans cette thèse. L'intérêt est notamment porté sur l'Optimisation des HyperParamètres (OHP). Ainsi, nous étudions l'impact de l'OHP sur les RNIs, et l'impact des RNIs sur l'OHP. Le but étant d'améliorer les algorithmes utilisés et de mieux comprendre le comportement des RNIs au regard de leurs hyperparamètres.

^a<https://futureoflife.org/open-letter/pause-giant-ai-experiments/>

Mots clés : réseaux de neurones à impulsions, optimisation des hyperparamètres, optimisation globale, calculs parallèles, optimisation par décomposition, optimisation bayésienne

CRISAL

UMR 9189 – Université de Lille - Campus scientifique – Bâtiment ESPRIT – Avenue Henri Poincaré – 59655 Villeneuve d'Ascq – France