Université de Lille - Faculté des sciences et des technologies

Ecole Doctorale Sciences Pour l'Ingénieur

Learning to Interact, Interacting to Learn Action-centric Reinforcement Learning

Apprendre à Interagir, Interagir pour Apprendre, Apprentissage par Renforcement Centré sur les Actions

Thèse de Doctorat en **Informatique**
soutenue publiquement à Villeneuve d'Asq le **28/09/2021** par

Mathieu Seurin

Président du Jury	Fabrice Lefevre	Professeur, Université d'Avignon
Directeur de thèse	Olivier Pietquin	Professeur, Google Research: Brain Team
Co-directeur de thèse	Philippe Preux	Professeur, Univ. Lille/CRIStAL/Inria
Rapporteur	Olivier Sigaud	Professeur, Sorbonne Université
Rapporteur	Ludovic Denoyer	Professeur, Facebook / Sorbonne Université
Examinatrice	Sao Mai Nguyen	Maîtresse de conférence, IMT Atlantique
Invité	Florian Strub	PhD, Deepmind

À tous ceux qui verront leurs erreurs à temp...

Remerciements

Je tiens tout d'abord à remercier mes deux directeurs de thèse, Olivier et Philippe. Olivier, l'expression "se tenir sur les épaules d'un géant" a pris tout son sens avec toi. Scientifiquement, tu as su être quelqu'un sur qui je pouvais compter dans les (nombreux) égarements de cette thèse. Tes idées, ta rigueur m'ont permis de devenir le bébé chercheur que je suis et je te remercie de la chance que tu m'as offerte. En dehors du laboratoire, je te considère véritablement comme un ami et c'est avec plaisir que je continuerai de discuter cinéma, autour d'une côte de bœuf et d'un whisky. Merci beaucoup.

Philippe, ton recul, ta bonne humeur et tes encouragements, même pour les idées les plus farfelues, me manqueront énormément après mon départ. Je te remercie également de la chance que tu m'as donnée et espère que de nombreux doctorants continueront de profiter de cette chance. L'équipe SequeL (et maintenant Scool) possède une atmosphère incroyable et c'est en partie grâce à toi.

La troisième personne à remercier est évidemment Florian, qui m'a suivi tout le long de cette thèse et avec qui j'ai eu la chance de collaborer à plusieurs reprises. Sans ta motivation et ta folie, bien des articles de cette thèse n'auraient pas vu le jour, je ne pourrai jamais assez te remercier. Je ne compte plus les excellentes soirées passées en ta compagnie, et j'espère que cela continuera. La bière à la betterave, c'est toujours non par contre !

Je remercie également mes rapporteurs Olivier Sigaud et Ludovic Denoyer d'avoir accepté de lire cette thèse et de me permettre de la défendre fièrement, un grand merci à vous deux de prendre ce temps.

Mes remerciements vont à mon examinatrice Sao Mai Nguyen ainsi que le président de jury Fabrice Lefevre d'avoir accepté mon invitation à participer au jury, à lire cette thèse et venir sur Lille pour la défense.

Camarade depuis la première heure, mais chercheur plus efficace que moi, je remercie également Édouard qui a été lui aussi un exemple à suivre dans ma thèse. Nos nombreuses discussions, mini-club lecture et ta rigueur m'ont permis de mieux cerner notre immense domaine et d'avancer scientifiquement. C'est également grâce à ces discussions qu'a pu renaître le ciquel (ciné-sequel). Accompagnés de Xuedong, également camarade du premier jour, nous avons parcouru les montagnes de Stellenbosch et profité du Hussard Grill, merci pour ces moments passés en votre compagnie.

Toujours au sein de l'équipe, je remercie particulièrement les autres membres du A06. Dorian, tu as été un ami (ou ennemi, au sens Nietzschéen du terme) d'une très grande importance. Nos débats, soirées ciné et jam resteront des souvenirs forts, et sans lesquels cette dernière année aurait été déprimante. Nathan, Sarah, Antoine et plus récemment

Marc, votre amitié a également beaucoup compté dans cette thèse et j'espère que l'on continuera de se voir ! Yohan également, tu es un vrai A06 !

Je remercie également les autres doctorants présents ou passés, Guillaume, Mahsa, Lilian, Ronan, Merwan, Nicolas, Ralph, Reda, Omar, Yannis, Léonard, Johan, Jules, Nino et les anciens stagiaires Raphaël et Geoffrey.

Merci également aux post-docs, Matteo sans qui EWRL n'aurait jamais vu le jour, Pierre qui a également permis la renaissance de cinéquel, Édouard Oyallon et Ralph.

Merci également à Emilie, Odalric grâce à qui j'ai découvert Buster Keaton, ainsi que Marie-Bénédicte sans qui EWRL et RLSS auraient été des fiascos !

Merci aussi à Alexis Ducarouge, qui, choisissant un autre chemin, m'a permis de contacter Olivier. Merci aussi à David Filliat, mon directeur de stage qui a toujours su trouver le temps de discuter avec moi et m'a convaincu de continuer en recherche.

Remontons un peu plus dans le temps, j'adresse mes sincères remerciements à Marie-Dominique Grihon, professeure de mathématique en terminal S Lycée Sud Médoc, qui m'a poussé à continuer dans les sciences et grâce à qui j'ai pu prendre le chemin que j'ai pris et me plonger dans l'informatique.

Il n'y aurait jamais d'embryon d'idée de thèse sans deux de mes meilleurs amis, Paul et Elias, avec qui notre passion pour l'informatique et les bières à 8° ont permis l'émergence de cette idée folle. Je remercie également Damien, Louis, Pierre et Leslie (bonne chance pour la tienne également !) dont l'amitié me fait encore avancer jour après jour.

À Nantes, je remercie ma seconde famille Vincent, Graziella, Tristan (la science, ça marche!).

Enfin, je remercie mon père Patrick, mon frère Clément et ma mère Laurence pour leur soutien pendant ses nombreuses années, j'espère vous rendre fiers avec ce manuscrit.

Résumé de la thèse

Dans cette thèse de doctorat, nous étudions l'apprentissage séquentiel (dit "par renforcement") en intelligence artificielle, plus particulièrement les notions d'actions et d'interactivité. En apprentissage par renforcement, un agent reçoit des informations sur son environnement et agit en conséquence. Le but étant de maximiser une quantité appelée récompense. La planification (quelles sont les conséquences à long terme des actions effectuées et quelle quantité de récompenses peux-t-on en tirer) et l'exploration (comment récupérer un maximum d'informations en un minimum de temps) sont au cœur de cette discipline.

L'intelligence artificielle et les sciences cognitives ont grandi de concert depuis les années 50. L'étude de la cognition a nourri les pionniers de l'intelligence artificielle et le courant cognitiviste s'inspirait de la modularité d'un ordinateur pour expliquer le fonctionnement du cerveau. Depuis, le courant cognitiviste a laissé place à la cognition incarnée. Au lieu de considérer l'intelligence comme un ensemble de fonctions abstraites, les représentations mentales sont construites et guidées par les interactions avec le monde. Nous souhaitons analyser les algorithmes d'apprentissage par renforcement avec ce même regard, en replaçant les intéractions au centre de notre analyse. Les actions définissent l'interface entre l'agent apprennant et l'environnement: pour le contrôle d'un robot, les actions correspondent aux forces exercées par les moteurs. Dans un jeu vidéo, elles correspondent aux différents boutons que l'on peut presser. On appelle ces différentes formes d'actions possibles: espace d'actions.

Dans un premier temps, nous proposons une taxonomie des différents espaces d'actions et les problèmes qu'ils posent. Par exemple : "Que se passe-t-il lorsqu'un agent doit choisir parmi plusieurs milliers d'actions ?" ou "Comment ignorer des actions inutiles ou généraliser à des actions jamais vues ?"

Dans un deuxième temps, nous montrons qu'en intégrant des connaissances sur les actions, on peut améliorer la vitesse d'apprentissage. Lorsque l'environnement nous empêche de faire certaines actions par sécurité, la prise en compte de cette information permet d'apprendre plus vite. Le deuxième cas porte sur l'exploration dans un environnement contenant multitudes d'objets à utiliser pour résoudre des problèmes type labyrinthe. Nous montrerons que pousser un agent à chercher les actions clefs qui intéragissent avec les objets permet une meilleure exploration que les autres méthodes de l'état de l'art.

La troisième partie de ce manuscrit porte sur l'apprentissage multi-but, c'est à dire apprendre une multitude de séquences d'actions, chacune accomplissant une tâche différente. Nous nous focalisons sur l'apprentissage d'instructions en langage naturel. Le langage simplifie la définition d'une multitude de sous-tâches en décrivant simplement ce que l'agent doit accomplir. Nous proposons un algorithme permettant de réduire la complexité d'apprentissage lorsqu'un grand nombre de buts doit être accomplis.

Enfin, la dernière partie porte sur la transformation de tâche non-interactive (supervisée) en tâche interactive. Rendre l'agent actif dans son apprentissage permet d'élargir les possibilités de l'apprentissage supervisé en lui permettant de choisir lui-même les informations intéressantes. Nous montrons qu'en changeant la définition d'une tâche de reconnaissance de locuteur, on réduit le temps d'apprentissage et le nombre de mots nécessaire à la reconnaissance.

Thesis Abstract

In this Ph.D. thesis, we study sequential decision making (a.k.a Reinforcement Learning or RL) in artificial intelligence, focusing on the notion of *action* and *interactivity*. In reinforcement learning, an agent receives information from its environment and acts. The goal is to maximize a constraint called "the reward". Planning (Anticipating long-term consequences and higher rewards) and Exploration (How to gather as much information as possible in a minimal time) are central to this topic.

The term "Reinforcement Learning" comes from psychology, and ever since, Artificial Intelligence and Cognitive Science have borrowed from each other. Cognitive study inspired early AI pioneers, and computers' modularity influenced cognitivism. Then, Embodied Cognition took over, putting interaction with the world at the center of mental developments. We study reinforcement learning with a similar stance, putting actions at the heart of this thesis. Actions define the interface between the agent and its environment. In robotic control, the actions are the motor's forces. In a video game, actions are the controller's button. We call those differents forms *action space*

Firstly, we propose an action space taxonomy and analyze challenges posed by each type. For example "How reinforcement learning algorithm stands when dealing with thousands of action ?" or "Is it easy to detect and ignore useless actions or generalize to unseen ones ?"

We then study how we can modify current algorithms to take into account action knowledge. The first setting considered is safe RL, where an agent acts under safety constraints. We show that when the environment prevents the agent from doing specific actions, taking into account this signal is essential to learn faster. Secondly, we propose an exploration algorithm nudging the agent to interact as much as possible with the environment.

The third part of the manuscript tackles action abstractions (sequences of interactions representing more general goals). For example, Natural Language can convey multiple sub-task by describing what the agent must accomplish. We propose an algorithm reducing sample complexity when dealing with a high number of instructions in natural language.

The last chapter is more general and formalizes how we can turn supervised setup into interactive ones. By reframing a speaker recognition task into a multi-turn game, we can increase the sample efficiency and reduces the number of words needed.

Contents

Introduction

The autonomy of the living is understood [...] both in regards to its actions and to the way it shapes a world into significance.

(Bourgine et al., 1991)

Every Ph.D. student has to face at some point the most challenging question of all: "What is your Ph.D. about?"
Explaining to the layman may result in an indigestible storm of jargon about your subject, in this instance, Machine Learning. "The field of Artificial Intelligence (**AI**) that gives computers the ability to learn without being explicitly programmed" (Samuel, 1959). It lies at the intersection of probability, optimization, and computer sciences."

At the end of your conference, the layman will rudely ask: "But ... Are you working with robots?". Presumably, in the general audience's mind, intelligence must be embodied. Automations and machines came long before computers, so acting in the real world feels more natural than working with abstract domains. However, the field's history is different.

In the 1950s, McCarthy, Minsky, Rochester, and Shannon proposed to tackle computerized intelligence during 1955' Summer (McCarthy et al., 1955). This very ambitious research project deals with manipulating abstract concepts, calculation, creativity, and reasoning. However, none of the seven internship proposals dealt with embodiment, interactivity, or related concepts.

Concurrently, psychology studied the human brain by examining learning, memory, problem-solving skills, defining intelligence as a set of abstract tasks. This movement is known as cognitivism, the study of mental processes (Mandler, 2002).

It was not until the late 1970s that embodied cognition followed cognitivism, grounding mental processes as part of the body's interactions with the world (Wilson, 2002). In this perspective, intelligence is no longer an abstract and autonomous process, disconnected from the body and its environment. To put it differently, human cognition must be rooted in sensorimotor processing. Those ideas poured into the machine learning world and slowly saw the birth of Reinforcement Learning (**RL**).

It is the subfield of Artificial Intelligence that addresses Sequential Decision Making. *Decision Making* because the agent (or program) receives information through sensors, and in turn, affects the environment through actions. The *Sequential* term comes from repeated interaction with the environment. In fine, the agent is optimized to maximize rewards or as (Sutton et al., 2018) puts it: "Learning how to get something from the environment".

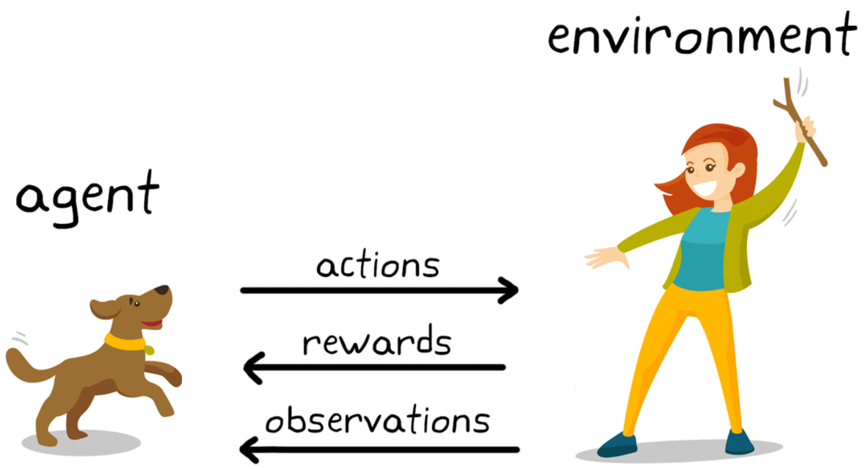


Figure 1: Reinforcement Learning Illustration. The agent (dog) receives observations from the environment (seeing its pet parent holding a stick) and maximizes rewards (food). At first, it does not know what to do; it barks, rolls on the floor, etc. However, by trial and error, the dog eventually learns to perform series of actions (running, fetching the stick) to gather food. (Image Source: https://fr.mathworks.com/discovery/reinforcement-learning.html)

Reinforcement learning is a general paradigm, allowing a diverse set of applications such as the game of Go, where an agent organizes stones to surround more territory than the opponent (Silver et al., 2016) or automatic stratospheric balloons navigation (Bellemare et al., 2020). Those two problems do not look remarkably similar, and strategies to solve one or the other seem unrelated. Go is a board game with long-term planning and strategy, whereas ballons driving is a navigation task in an unpredictable environment. However, both can define a set of actions (putting a stone on the board, burning fuel to go higher) and a reward (winning the game, reaching a destination) and requires trying different strategies to find out which one dominates the others. Discovering strategies (or policies) that gather maximum rewards is central to the reinforcement learning paradigm. In Go, maximizing the reward corresponds to winning every game, whatever strategy the opponent is using. For ballons navigation, it would be to reach its destination every time and quickly. Finding an optimal policy is not an easy task as the agent must cope with environment unpredictability, sensors noises, long-term planning, and exploration. Thus, reinforcement learning is a unifying view on sequential decision-making problems, designing general algorithms that could (theoretically) tackle Go, ballons navigation, and many other problems.

In the last decade, we witnessed great leaps in this direction, Deep Q-Networks played Atari games using raw pixels (Mnih et al., 2015), AlphaGo (Silver et al., 2017) became virtually unbeatable and even mastered Chess and Chogi (Silver et al., 2018). Two reinforcement learning agents became respectively GrandMaster in Starcraft (Vinyals et al., 2019) and Dota (Berner et al., 2019). Each algorithm found an excellent strategy through long training, crunching millions of data points. However, are they efficient? To reach its final performance, AlphaGo played around five million games. A game usually lasting an hour, the algorithm spent about five hundred seventy years playing Go. As a comparison, its opponent, pro-player Lee Sedol (aged 33 at the time of the match), could not spend as much time playing. He is said to be more *data-efficient*: he played less, but extracted more information from each game.[1]. Thus, final performance matters, but the number of interactions to achieve optimality is also central. Chapter 1 introduces more formally classical and recent reinforcement learning methods and discusses how algorithms traditionally reduce their data consumption. The whole thesis is aligned with this principle: increasing the data-efficiency of **RL** algorithms.

Zooming on the actions in Reinforcement Learning

As we discussed, intelligence must be studied through its interactions with the environment. We advocate that reinforcement learning agents should be analyzed with the same considerations. **RL** paradigm already takes into account sensory mechanisms as agents must act according to the environment state. Moreover, numerous examples highlight how changing the world representation affects learning, either by changing the sensors or slightly modifying the inputs (Zhang et al., 2018; Hussenot et al., 2020b).

However, we believe **RL** is a partial answer to the questions raised by the embodied cognition movement. Reinforcement Learning researchers neglected how changing the actions affects the learning dynamics (Kanervisto et al., 2020). The set of actions available in an environment is called an action space. For example, on the Atari2600 (a real console), the joystick selects an orientation (including diagonals) and could be combined with the button (triggering different effects, depending on the game. Thus, the action space is the combination of all possible directions with the button being pressed and released. The Arcade Learning Environment (Bellemare et al., 2013) is a suite of fifty-five games, coming from the Atari2600, aiming to design a single algorithm tackling all the games. However, the action space is not standard across all games. **RL** researchers adapted the action space for each game, simplifying the learning process but losing research opportunities to design algorithms that cope with this constraint. The same goes for every environment; those spaces are heavily designed and modified by hand, for example, by removing action that seems unnecessary or too similar. Designing general agents implies reducing the action space engineering to the minimum and the ultimate goal is to tackle those problems with data-driven approaches.

Several milestones lie before achieving this objective, and a better understanding of the diversity of actions spaces is necessary to propose a unifying view. We review in chapter 2 questions such as: “What type of actions spaces exist and what algorithms can be used on each type ?” Also, the more general we want an agent to be, the more actions it requires; thus we also review how current reinforcement learning algorithms handle large actions spaces. The survey leads us to this thesis’s contributions.

To illustrate our point, we take as an example behavioral data reported by Tucker et al., 1998, showing how human visual processing informs actions controllers. Subjects are

[1]AlphaGo consumes about 170kW/h of energy to play an hour-long game (1202CPU x 100W + 176GPU x 300W for an hour), whereas Lee Sedol consumes about 20W/h, energy efficiency could also be taken into account.

tasked to indicate whether common objects (e.g., a teapot, a frying pan) are upright or inverted by pressing buttons on a remote controller. Authors measure response times when the left hand or the right hand manipulates the controller. The hand dealing with the remote controller is called the response hand. Experiments show that response times are fastest when the response hand is the same as the hand that would be used to grasp the depicted object (*e.g.*, the left hand if the teapot's handle is on the left)." We can conclude that *context gives hints about what is possible.* Translated coarsely using **RL** vocabulary "states should inform which actions can be performed". For example, a wall in front of the agent indicates that going forward is impossible, or a danger sign prevents from doing something foolish. We coined the term "Contextual Ineffectiveness" to describe actions that are ineffective in certain contexts (for safety reasons, for example). In a subsequent section (2.3), we show that reinforcement learning agents are struggling to detect such signal, even when it can be extracted from the state; thus, the agent loses time by trying actions that are ineffective. Chapter 3 stems from the same principle but dive into exploration methods. Current strategies focus on state novelty and unpredictability but fail to push the agent in areas where some specific actions are required. For example, we show that learning to use the action "push" in front of a button or "open" in front of a door is hard for current exploration algorithms and propose a method that detects and rewards rare interactions.

A final disclaimer: Much too often, spurious brain analogies are used to justify some **ML** techniques, which brought confusion to a general audience. The approach taken in the present thesis was to use embodiment theory as a starting point for our journey. However, results presented in this thesis can only apply to reinforcement learning agents and algorithms, and we do not conclude about human cognition.

Thesis Outline

Deep Learning and Reinforcement Learning Background are introduced in Chapter 1. It describes tools developed after 2014/2015 in Deep Reinforcement Learning and gives entry to a reader willing to catch up with the latest literature.

Chapter 2 introduces a general taxonomy of actions spaces, it delves into the diversity, structure, and methods to switch between each type. It also covers state-of-the-art tools to learn an action space structure and tackle *large* action spaces. The next sections motivate safe, and constraint reinforcement learning as an application and details the first contribution of the thesis. By giving an external signal on the action being executed, an adaptation of a state-of-the-art deep reinforcement learning algorithm can greatly reduce the sample complexity and learns to avoid hazardous actions.

Chapter 3 is a summary of exploration and intrinsic motivation methods in reinforcement learning, focusing on large state space, where function approximation is necessary. The second contribution highlights that embodied domain's actions are very different by nature. Some actions are contextual, such as triggering a button or opening a door. The method proposed uses actions with different consequences as an exploration signal.

The next chapter 4 elaborates on the notion of goal in reinforcement learning. Goal-based methods aim to build multi-objective policy, accomplishing a variety of tasks. Those higher-level objectives can be seen as more complex actions, especially in open-ended domains where many goals are available. The background section emphasizes instruction-following using natural language as it offers a natural structure of the environment. The third contribution aims to improve instruction understanding and execution. By learning a language generator that complements the data collection, the policy learns more instructions faster.

Chapter 5 gives a general perspective between Supervised Learning and Reinforcement Learning, thinking about how to convert supervised tasks to interactive ones. Sequential setup can reduce the data consumption when training supervised models, such as Active Learning. The final contribution shows that converting a speaker recognition task to a reinforcement learning setup can reduce the number of samples needed when identifying speakers.

The final chapter discusses future directions in action-centric reinforcement learning.

Thesis Contributions

Papers presented in the thesis

- Mathieu Seurin, Philippe Preux, and Olivier Pietquin (2020a). “"I'm Sorry Dave, I'm Afraid I Can't Do That" Deep Q-Learning from Forbidden Actions”. In: **Proceedings of the International Joint Conference on Neural Networks, (IJCNN)**
- Goeffrey Cideron*, Mathieu Seurin*, Florian Strub, and Olivier Pietquin (2020). “HIGhER: Improving instruction following with Hindsight Generation for Experience Replay”. In: **Proceedings of the IEEE Symposium Series on Computational Intelligence, (SSCI)**
- Mathieu Seurin, Florian Strub, Philippe Preux, and Olivier Pietquin (2020b). “A Machine of Few Words: Interactive Speaker Recognition with Reinforcement Learning”. In: **Proceedings of the IEEE International Conference of the Speech Communication Association, (INTERSPEECH)**
- Mathieu Seurin, Florian Strub, Philippe Preux, and Olivier Pietquin (2021). “Don't Do What Doesn't Matter: Intrinsic Motivation with Action Usefulness”. In: **Proceedings of the Internationnal Joint Conference on Artificial Intelligence, (IJCAI)**

Other contributions

- Florian Strub, Mathieu Seurin, Ethan Perez, Harm De Vries, Jérémie Mary, Philippe Preux, Aaron Courville, and Olivier Pietquin (2018). “Visual reasoning with multi-hop feature modulation”. In: **Proceedings of IEEE European Conference on Computer Vision, (ECCV)**
- Timothée Lesort*, Mathieu Seurin*, Xinrui Li, Natalia Díaz-Rodríguez, and David Filliat (2019). “Deep unsupervised state representation learning with robotic priors: a robustness analysis”. In: **Proceedings of the International Joint Conference on Neural Networks, (IJCNN)**

Event Organization

- Pirotta Matteo, Ronan Fruit, Florian Strub, and Mathieu Seurin (2018). “European Workshop on Reinforcement Learning, (EWRL)”. in: **Scientific Event Organization**
- Nicolas Carrara, Omar Darwiche Domingues, Yannis Flet-Berliac, Emilie Kaufmann, Edouard Leurent, Odalric-Ambrym Maillard, Pierre Ménard, Philippe Preux, Mathieu Seurin, Xuedong Shang, Julien Seznec, Florian Strub, and Mohammad Sadegh Talebi (2019a). “Reinforcement Learning Summer School, (RLSS)”. in: **Scientific Event Organization**

Chapter 1

Deep Reinforcement Learning Cookbook

~~Cooking~~ Deep RL requires confident guesswork and improvisation — experimentation and substitution, dealing with failure and uncertainty in a creative way. —

Paul Theroux *(American Novelist)*

In this chapter, we lay the technical foundations for this manuscript. Mathematical notations, definitions, and recipes lie ahead, but this formalism will reward the reader greatly by broadening its Machine Learning horizons and unlocking access for the following chapters.

Deep Learning and Reinforcement Learning Chefs can skip to Chapter 2.

1.1 Machine Learning and Deep Learning Background

1.1.1 Machine Learning Ingredients

To get started with machine learning, few utensils are necessary. First, select an area of interest (see Section 1.1.2) and collect some information that will form a *dataset*. Secondly, define how to approach the problem. Machine Learning can be split into three mains areas (Bishop, 2006): Supervised Learning, Unsupervised Learning, and Reinforcement Learning. Thirdly, to learn this dataset, select a *model* and its associated *optimizer* in your kitchen cabinet. The model can be anything, ranging from Decision Tree, Bayesian Linear Regression, or Neural Network (This thesis will focus on the latter, but many algorithms presented here can be used with other models). The optimizer depends heavily on the model, and it describes how to adapt the model to *fit* the data.

Among the most successful approaches is Supervised Learning. The dataset is composed of examples $x \in X$, where each example is associated with a label $y \in Y$. The goal is to predict y from x or more formally: given a dataset of $\{(x^{(i)}, y^{(i)})\}_{i=0}^{N}$, learn a model such as $\hat{p}(y|x)$ that generalizes to new instances of x.

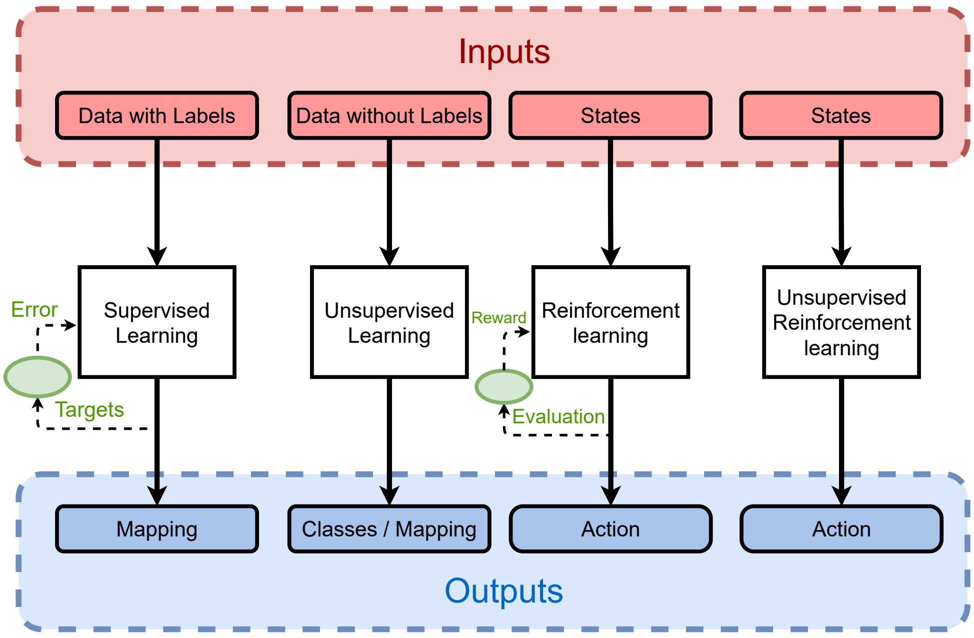


Figure 1.1: Illustration of different learning paradigms.

The second approach, called Unsupervised Learning, has a broader definition. Given a dataset $\{(x^{(i)})\}_{i=0}^{N}$, learn a model of $p(x)$ (or a distribution over X). Unsupervised Learning covers a variety of tasks such as clustering (Rai et al., 2010; Xu et al., 2015), dimensionality reduction (Cunningham et al., 2015) or self-supervised learning (Jing et al., 2020; Doersch et al., 2015)).

Reinforcement Learning (**RL**) (Sutton et al., 2018) is a general paradigm to tackle sequential decision making, or learning to act under uncertainty. Reinforcement learning problems involve learning what to do and mapping observations to actions to maximize a numerical reward signal. The model is not told which actions to take (as in supervised learning), but instead must discover which actions yield the most reward by trying them out. The final goal is to discover the best sequence of actions (called strategy or policy). A whole section is dedicated to introducing **RL** (see Section 1.3). A variation called Unsupervised Reinforcement Learning or Learning without rewards (Lim et al., 2012; Jin et al., 2020), combines ideas from Unsupervised Learning and Reinforcement Learning and will not be explored deeply in this manuscript. However, exploration methods covered in Chapter 3 can tackle partially the no-reward setting.

1.1.2 Exemple of applications

This non-exhaustive list highlights the diversity of topics tackled by machine learning: Climate Change (Rolnick et al., 2019), Material Design (Mirhoseini et al., 2020), Physics (Charpagne et al., 2019), Medicine (Rajkomar et al., 2019; Cireşan et al., 2013; Ronneberger et al., 2015; Hashimoto et al., 2018), Economic forecasting (**stockprediction**; Patel et al., 2015; Asadi et al., 2012), Education (Vie et al., 2017), Biology (Senior et al., 2020; Zhou et al., 2017), Maths (Lample et al., 2019), Ressource Management (Mao et al., 2016), Ballons navigation (Bellemare et al., 2020), Autonomous Driving (Leurent, 2020), Cooking (Xin Wang et al., 2015), Automatic Whisky Brewering (oops, not yet.)

1.2 Deep Learning

Deep Learning (**DL**) (LeCun et al., 2015; Goodfellow et al., 2016) is a sub-field of machine-learning that emerged from the meeting between Neural Network (**NN**) and Graphics Processing Unit Hardware (**GPU**) in the 2010's (Cireşan et al., 2011; Cireşan et al., 2012). Using **GPU** (and later dedicated hardware such as Tensor Processing Unit **TPU** Jouppi et al., 2017) decreased by many orders of magnitude the time to train models and thus increase the capacity to tackle larger datasets.

1.2.1 What tools are necessary to follow this thesis?

Deep Learning architectures are not central to this thesis, but few tools are necessary. General vision architecture such as Convolutional Neural Network (**CNN** LeCun et al., 1995) and a basic understanding of Recurrent Neural Network (**RNN**) such as Long-Short-Term-Memory (**LSTM** Hochreiter et al., 1997). When necessary, we will discuss the models in greater details.

1.2.2 A brief history of Deep Learning

In about ten years, working with gigabytes (**GB**) of data became easier, and it translated into an increase in performance in many tasks. *"The more data you can collect, the greater your results will be,"* says the deep learning practitioner, which can be problematic. We will take two examples to illustrate this dramatic increase in data consumption.

The dataset that launched the deep learning trend in image classification was ImageNet, specially the Image Large Scale Visual Recognition Challenge (ILSVRC Russakovsky et al., 2015), composed of 1.3 million images. Less than 10 years later, Xie et al., 2020 uses a dataset made out of 3.5 billion images to train their model.

Another example taken from Natural Language Processing (**NLP**) where datasets went from 348 millions tokens (around 500**MB** of data) (Bahdanau et al., 2015) to rougly 500 billions tokens (570**GB**) (Brown et al., 2020), and models went from around 60 millions parameters to 175 millions.

By massively parallelizing gradient estimates and backpropagation (Linnainmaa, 1976; Rumelhart et al., 1986), the field went from training a network in six days (Krizhevsky et al., 2012) to a few minutes (You et al., 2018), reducing training cost per parameter (but increasing cost globally). Faster training and bigger datasets translated into an increase in performance, but the story does not end here.

A second benefit comes from the generality of backpropagation. Once basic differentiable blocks are defined, even with crazy recombination and chaining, gradients can be computed. It means that models are much more flexible. They can share information, be reused and fine-tuned (at least in theory).

For example, merging streams from different models, treating more than two modalities is called multi-modal learning (Ngiam et al., 2011). Splitting a stream into multiple sub-components can tackle multi-task learning (Ruder, 2017; Schmidhuber, 2018). Those two fields are, by no means, new, but the simplicity of combining models and using them for different tasks spurred great interest in the machine learning community.

The next section introduces the reinforcement learning paradigm and how it was combined with deep learning in recent years.

1.3 Reinforcement Learning

Drawing its inspiration from Dynamic Programming (**DP**) (Bellman, 1957), Reinforcement Learning[1] (**RL**) aims to solve *sequential decision-making in uncertain environments* or Learning by trial and error in non-deterministic worlds (Sutton et al., 2018). The ultimate goal is to discover the best strategy in each state, called the *policy*. Repeated interaction is mandatory as doing an action in the same configuration can lead to different outcomes due to the environment *stochasticity*. The *best* strategy is the one that gathers the maximum amount of reward along its trajectories, following sections describe more formally how to numerically assess a policy's quality and compare policies.

RL differs from **DP** by the fact that the environment dynamic is unknown. Approximate Dynamic Programming methods learn a model of the environment and apply **DP** techniques, whereas reinforcement learning directly learns to navigate by trial and error.

In the following section, we will cover all the notations, objectives, and some classical algorithms used in **RL**. Notations follow Sigaud et al., 2008 and a list of symbols can also be found at the end of the thesis.

1.3.1 Learning to interact with the environment: Markov Decision Process formalism

Modeling the environment is done using the Markov Decision Process (**MDP**) paradigm (Puterman, 2014). At each timestep t, the agent receives a state $s \in \mathbb{S}$ and chooses an action $a \in \mathbb{A}$ according to a policy $\pi \in \Pi$. The environement computes a new state s', that depends on a transition kernel P and returns a reward r, from the reward function $R : \mathbb{S}^2 \times \mathbb{A} \to [0, 1]$ (some environments consider higher rewards, but $[0, 1]$ is standard).

Ultimately, reinforcement learning algorithms seek to find π^*, the optimal policy. To compare agents and assess what *optimal* means, one must consider a numerical criterion, the higher the value is, the better. The criterion considered in this manuscript is the *discounted cumulative reward*, the objective function can be written as:

$$E^{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right], \tag{1.1}$$

$\gamma \in [0, 1]$ being the discount factor and E^{π} the expection over trajectories when following policy π. The discounted cumulative reward allows tackling *infinite-horizon* MDP and to tune the horizon using the parameter γ. Large γs (close to 1) lead to very long horizons and small γs generate short-sighted agents.

Thus the MDP is defined as the tuple $(\mathbb{S}, \mathbb{A}, R, P, \gamma)$.

[1]sometimes called *Neuro-Dynamic Programming* (Bertsekas et al., 1995), highlighting the **DP** legacy.

To measure the policy's quality, we define the value function $V : \mathbb{S} \rightarrow \mathbb{R}$ and the state-action value function $Q : \mathbb{S} \times \mathbb{A} \rightarrow \mathbb{R}$. They assess the average quantity of reward received when following the policy π and are essential to the reinforcement learning paradigm.

$$\begin{aligned} \forall s \in \mathbb{S} \quad V^{\pi}_{\gamma}(s) &= E^{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s\right], \\ \forall s \in \mathbb{S}, \forall a \in \mathbb{A} \quad Q^{\pi}_{\gamma}(s,a) &= E^{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a\right]. \end{aligned} \tag{1.2}$$

Note that, when writing V_γ or Q_γ, γ will be ignored, becoming V or Q. The optimal value function V^* (resp. state-value function Q^*) is defined as :

$$\begin{aligned} \forall s \in \mathbb{S} \quad V^{*}_{\gamma}(s) &= \max_{\pi} V^{\pi}(s), \\ \forall s \in \mathbb{S}, \forall a \in \mathbb{A} \quad Q^{*}_{\gamma}(s,a) &= \max_{\pi} Q^{\pi}(s,a). \end{aligned} \tag{1.3}$$

And both value-functions are linked with the following relationship:

$$\forall s \in \mathbb{S} \quad V^*(s) = \max_a Q^*(s,a). \tag{1.4}$$

Thus, the optimal policy π^* is defined as :

$$\forall s \in \mathbb{S} \quad \pi^*(s) = \arg\max_a Q^*(s,a). \tag{1.5}$$

In this instance, π^* is *greedy*, it takes the best action indicated by Q. Eq. (1.4) and Eq. (1.5) highlights that finding Q^* imply obtaining V^* and π^*. Thus, **RL** practioners may either: directly learn π^* or find ways to compute Q^* to solve an environment.

Efficiency If two algorithms achieve the maximum discounted cumulative reward, one must consider the number of samples they consume. Thus, a second criterion to consider is the number of samples needed to discover π^*. A method requiring a low amount of data is said to have a high *sample-efficiency* or a low *sample-complexity*. On the contrary, an inferior method is said to have a low sample-efficiency or a high sample-complexity.

1.3.2 How to reinforcement learn?

Reinforcement Learning methods can be classified in at least two categories, *model-free* and *model-based* methods. Model-free algorithms learn a value-function or a policy, while never considering explicitely the overall environment dynamics (thus the name, model-*free*). On the other hand, model-based approaches imply learning a transition model, either by learning the kernel P or other forms of dynamics estimation (forward/inverse models).

Recent works do not assume anything about the environment dynamics Ha et al., 2018; Hafner et al., 2020; Schrittwieser et al., 2020 but model-free methods still outperform their model-based counterpart in discrete actions domains (Hasselt et al., 2019). Thus, this thesis focuses on model-free methods, however, later sections (Section 2.1.3.3 and Section 3.1.2.2) describe how some form of dynamics modelling reduces the sample complexity or refine the exploration.

Model-free methods can be refined in three categories, *pure-critics* algorithms, *pure-actors* and *actor-critic* algorithms.

1.3.2.1 Learning a Value Function

Pure-critics algorithms estimate the optimal value-function V^* (or Q^*) using either Monte-Carlo estimation or by bootstrapping using temporal differences. The optimal policy π^* is then derived from the optimal value function by acting greedily w.r.t V^* (or Q^*).

Temporal Differences (**TD**) methods update the current state's value $V(s_t)$ by using estimates of future states ($V(s_{t+1}), V(s_{t+2})$, etc..). The technique is known as *boostrapping*. The simplest form of value learning is *TD(0)* where $V(s_t)$ is updated using $V(s_{t+1})$

$$V(s_t) \leftarrow V(s_t) + \alpha(\overbrace{r_t + \gamma V(s_{t+1})}^{\text{TD(0) target}} - V(s_t)), \tag{1.6}$$

where α is the learning rate, defining the update's magnitude. Methods such as SARSA (Rummery et al., 1994) or Q-learning (Watkins et al., 1992) falls under this category. Q-learning update rule is defined as follow:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \max_{a'}[Q(s_{t+1}, a')] - Q(s_t, a_t)). \tag{1.7}$$

At the other end of the spectrum lies Monte-Carlo's (**MC**) estimation. **MC** methods rely on whole trajectories to estimate the value function instead of using value estimates.

$$\begin{aligned} V(s_t) &\leftarrow V(s_t) + \alpha(G_t^n - V(s_t)), \\ \text{where } G_t^n &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^n r_{t+n}. \end{aligned} \tag{1.8}$$

MC estimates are unbiased but tend to have a higher variance than bootstrapping; thus, they tend to be less sample-efficient (see Fig. 1.2 for an illustration of **MC** and bootstrap methods)

Complementary approaches aims to combine **TD** and **MC** methods such as TD(λ) (Sutton et al., 2018; Dayan et al., 1994; Tsitsiklis, 1994), SARSA(λ) (Sutton et al., 2018) or Q(λ) (Harutyunyan et al., 2016).

TD(λ) can be understood as one particular way of averaging n-step return G_n. This average contains all the n-step return, each weighted proportionally to λ^{n-1}, where $\lambda \in [0, 1]$:

$$\begin{aligned} V(s_t) &\leftarrow V(s_t) + \alpha(G_t^\lambda - V(s_t)), \\ \text{where } G_t^\lambda =& (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_t^n, \\ \text{and } G_t^n &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^n r_{t+n}. \end{aligned} \tag{1.9}$$

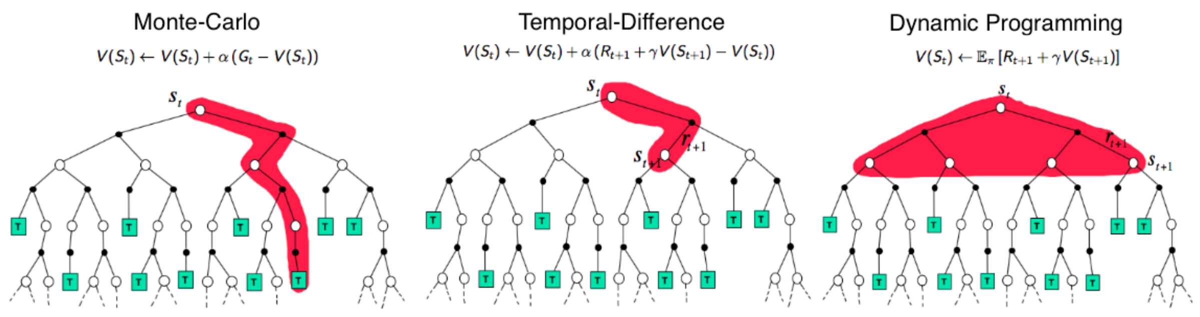

Monte-Carlo
$V(S_t) \leftarrow V(S_t) + \alpha (G_t - V(S_t))$
Temporal-Difference
$V(S_t) \leftarrow V(S_t) + \alpha (R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$
Dynamic Programming
$V(S_t) \leftarrow \mathbb{E}_\pi [R_{t+1} + \gamma V(S_{t+1})]$
S_t
S_{t+1}
r_{t+1}
T

Figure 1.2: Illustration of Monte-Carlo vs Boostrap vs **DP** (Image Source: https://www.davidsilver.uk/wp-content/uploads/2020/03/MC-TD.pdf)

Function Approximation Storing and iterating over every state is impractical when the number of states is large or even infinite (in continuous domains). Thus, approximating the value function (or policy) becomes necessary to tackle larger problems. Linear approximation version of classical dynamic programming were proposed for example in the game of checkers (Samuel, 1988) and later in reinforcement learning : LSTD (Bradtke et al., 1996), LSPI (Lagoudakis et al., 2003), TD(λ) (Tsitsiklis et al., 1997). Non-linear approximation, especially neural network, was popularized by Tesauro, 1995 with TD-Gammon and later (**coulom2002thesis**; Riedmiller, 2005; Ernst et al., 2005) proposed to adapt reinforcement algorithm with neural networks or regression-trees. When using function approximation, policies and value functions are parametrized by θ, a vector of parameters, thus we adopt the following notation : Q_θ, V_θ for value-functions and π_θ for policies.

1.3.2.2 Learning a Policy

Policy-Learning algorithms directly search the policy space. Policy gradient (**PG**) methods (Williams, 1992; Sutton et al., 1999a) differentiate an objective function surrogate (Eq. (1.1)) and update the policy according to its gradient. The general update rule is:

$$\theta \leftarrow \theta + \frac{\partial}{\partial \theta} J(\theta). \tag{1.10}$$

The central question is how to compute $\frac{\partial}{\partial \theta} J(\theta)$. The simplest method rely on computing gradient's Monte-Carlo estimates :

$$\nabla_\theta J(\theta) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta (a_t \mid s_t) G_t, \tag{1.11}$$

which in plain english gives : "Sample a trajectory, if the return is positive, increase the probability of what the agent did, if the return is negative, decrease the probability."

Stemming from this update rule, different methods refines gradient computation by using second-order methods such as Natural Policy Gradient (Kakade, 2001). For a complete overview of first-order and second-order methods, see (Pierrot et al., 2018). On the other hand, Evolutionary methods (Moriarty et al., 1999) are gradient-free methods, relying on hill-climbing techniques to optimize the policy.

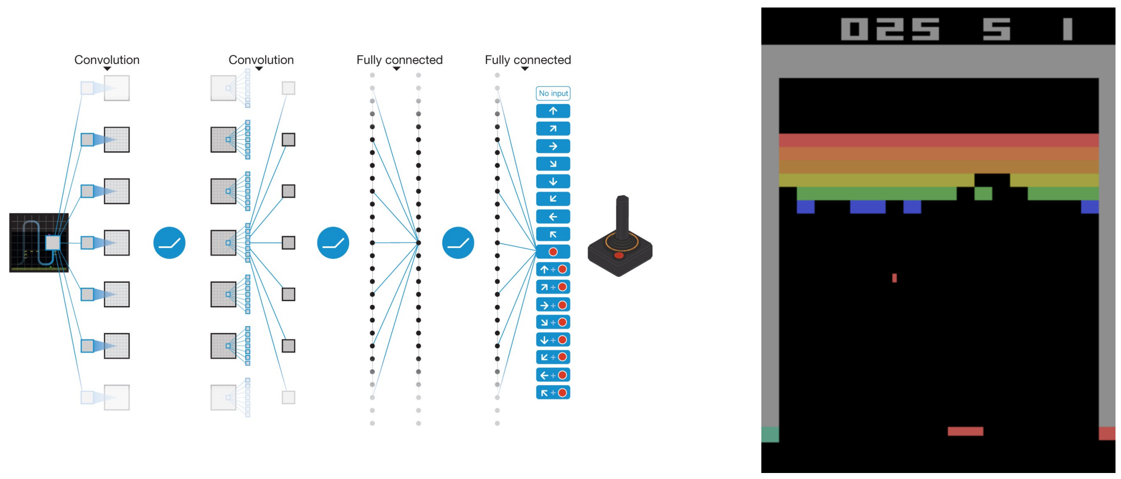


Figure 1.3: (Left) DQN model illustration and its actions space. (Right) Illustration of the input receives by the model, an instance of the Atari Games Environment (Bellemare et al., 2013). (Image Source: https://www.nature.com/articles/nature14236)

Actor-critic algorithms Last but not least, this category of methods intertwine value estimation and policy optimization (Konda et al., 2000; Kimura et al., 2000). Since policy gradient methods rely on monte-carlo estimates, they face a high variance. To reduce it and increase the sample-efficiency, actor-critic methods learn a value function alongside the policy. Instead of waiting until the end of a trajectory to recover Gt, V estimates are used, thus giving the following update rule:

$$\nabla_\theta J(\theta) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta \left(a_t \mid s_t\right) V(s_t). \tag{1.12}$$

Concurrently, $V(s_t)$ is learned using either Monte-Carlo or Temporal Differences.

1.4 Deep Reinforcement Learning (DRL)

DRL refers to the subfield of **RL** that uses neural networks to approximate the value function, the policy, or both. The term emerged after Mnih et al., 2015 achieved strong performances on video games using visual inputs. They combine **CNN**, Q-learning and a couple of tricks (described in Section 1.4.1.2). Working in visual domains was especially challenging because of the high dimensionality of the state space.

In Section 1.4.1, Q-learning is adapted to fit larger neural networks, and Section 1.4.3 describes how to scale such algorithms.

1.4.1 Deep Q-Learning (DQN)

1.4.1.1 Base model : Framing RL as a Supervised Problem

DQN is conceptually simple, since Neural Networks are good at minimizing well defined loss, DQN reframes the **RL** problem as a series of regression tasks. Since Q functions can be learnt by minimizing the TD error, the problem is written:

$$\min_{\theta}\left(\underbrace{r_t+\gamma\max_{a_{t+1}}(Q_\theta(s_{t+1},a_{t+1}))}_{\text{target}}-\underbrace{Q_\theta(s_t,a_t)}_{\text{estimation}}\right)^2. \tag{1.13}$$

For neural networks to learn correctly and converge, it requires batches of uncorrelated data (Keskar et al., 2017), an hypothesis which does not hold in **RL** as trajectories are sampled sequentially. To avoid wasting samples and update the network with independent datapoints, transitions are stored in a replay buffer (Lin, 1992). When updating the Q-function, a batch of randomly selected transitions is sampled from this replay buffer. To benefit even further from the buffer, one can wait for enough transitions to be collected before updating Q and superseding old transitions with fresher ones.

A second problem arises when using non-linear function approximation and boostrap estimates, a problem part of the deadly triad (Sutton et al., 2018; Van Hasselt et al., 2018). In Eq. (1.13) since the target $r_t+\gamma\max_a(Q_\theta(s_{t+1},a))$ and the estimate $Q_\theta(s_t,a)$ are computed with Q_θ it can quickly spiral out of control. To stabilize the training, a second network called target network θ' is used to estimate $Q(s_{t+1},a_{t+1})$ and is updated less frequently than the main network θ, resulting in the following objective:

$$\min_{\theta}\left(r_t+\gamma\max_{a_{t+1}}(Q_{\theta'}(s_{t+1},a_{t+1}))-Q_\theta(s_t,a_t)\right)^2. \tag{1.14}$$

1.4.1.2 Spicing up your DQN

Over the years, additionnal seasoning were added to the main **DQN** recipe. **DQN** uses the same values both to select (greedy selection) and to evaluate an action (max operator in Q-learning). This makes it more likely to select overestimated values, resulting in overoptimistic value estimates. *Double* Q-learning (Van Hasselt et al., 2016) breaks this vicious circle by decorrelating the action selection from the target computation. *Dueling* (Wang et al., 2016) separates the Q architecture in two streams : Baseline and Advantage estimation. A third trick, instead of sampling uniformly in the replay buffer, Prioritized Experience Replay (Schaul et al., 2016) samples transitions proportionnally to the TD-error (Eq. (1.14)). Intuitively, the higher the error is on a specific transition, the more it is replayed. Two more tricks : *NoisyNet* (Fortunato et al., 2018) introduces noise in the parameters θ to improve the exploration and Bellemare et al., 2017 learns a distribution over the return instead of learning the mean. Finally, instead of boostrapping the next state value $Q(s_{t+1},a_{t+1})$, Q can be boostrapped from the value in n timesteps called *n-step* estimation, resulting in a faster training. Combining all these refinements results in Rainbow (Hessel et al., 2018; Hasselt et al., 2019).

1.4.2 Dealing with Partial Observability

The **MDP** formalism and thus all preceding definitions rely on the Markov property. The agent receives a state s that contains all the information necessary to act and remembering past states is unnecessary. However, many environments do not meet this hypothesis, for example, first-person games (Fig. 1.4). Even in classical problems such as CartPole, the

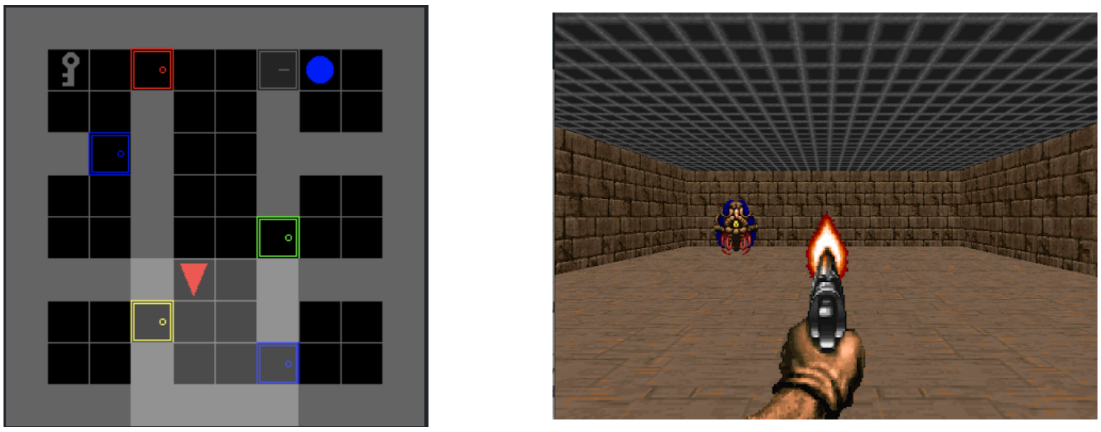


Figure 1.4: Two instances where the observation is partial, on the left, the agent can not see behind doors and on the right, the view is limited to what is in front of the character. Thus, a monster in the agent's back will result in the same observation (nothing in sight), but a very different state (potential death incoming)!

velocity must be given as input; otherwise, multiple states are necessary to estimate the cart speed.

Partially-Observable MDPs (**POMDPs**) (Åström, 1965; Kaelbling et al., 1998) formalize this problem: environment states s are separated from the agent observations o. Multiple states can correspond to the same observation; thus, remembering past observations is necessary to build a *belief states*. If the resulting belief state is Markovian, any reinforcement learning algorithm can be used to tackle a **POMDP**.

Two techniques are standard to deal with partial observability. The first and simplest one, used by Mnih et al., 2015 is to stack the last n observations (usually 4) as input instead of using only the last one. It circumvents simple problems (such as estimating the direction of moving objects). However, longer memorization is doomed to fail as stacking too many observations will drastically increase state size, deteriorating sample efficiency and generalization.

A second strategy employed by Hausknecht et al., 2015 uses a Recurrent Neural Network (**RNN**) on top of the Convolution Network (**CNN**), giving the agent a working memory over longer horizons. Similar to *information state* in the **POMDP** setting (Cassandra, 1998), the **RNN** aggregates past observations. This improvement requires modifying the replay buffer to store sequences of transitions and replay them entirely. Replaying transitions sequentially breaks the sample independence hypothesis, but training remains stable if enough sequences are used (Kapturowski et al., 2018). The resulting algorithm is called *Recurrent DQN* (**RDQN**).

1.4.3 Scaling up

DRL enables to train models on very high-dimensional domains. However, more complex domains also imply longer training times. The original **DQN** data consumption is equivalent to 38 days of human gameplay. Finding ways to speed up algorithms is essential to use those methods on more complicated and real-world problems.

DRL methods benefit differently from the scaling observed in **DL**. The trend where bigger

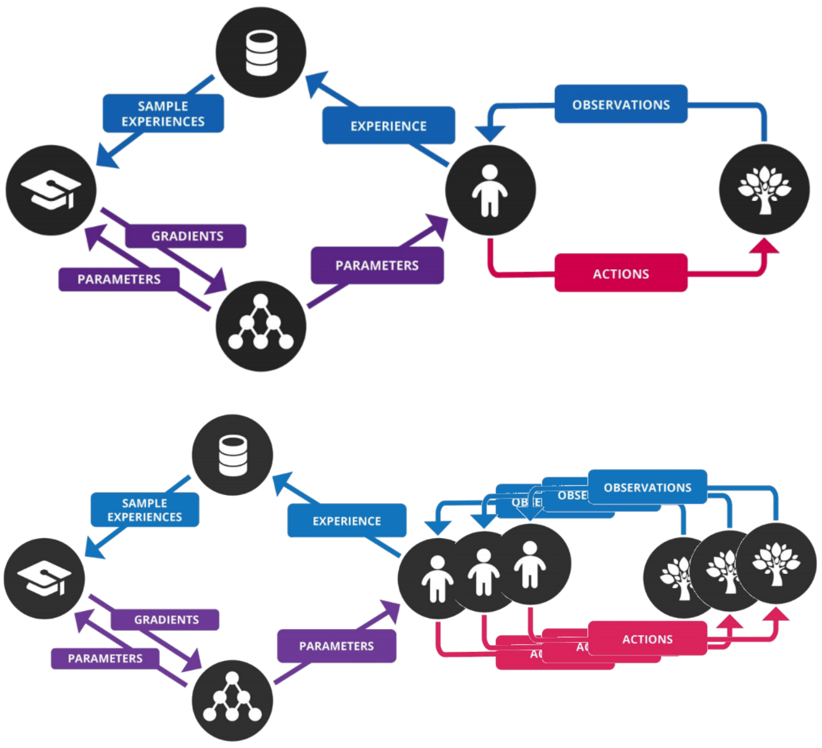


Figure 1.5: Illustration of DQN and Distributed-DQN, the overall architecture stays the same, only the number of actor collecting trajectories changes. (Image Source: https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/)

architectures endlessly improve is more subtle in **RL**. For example, even recent papers on the Atari benchmark are still using Mnih et al., 2015's **CNN** architecture (Kapturowski et al., 2018).

We can point out a few architectural designs that improve performances over simple architecture (Chaplot et al., 2018; Parisotto et al., 2020; Ammanabrolu et al., 2019) but many improvements in the supervised setting do not carry over to **RL**. Moreover, we are far from seeing gigantic models that are flourishing in vision and language (Brown et al., 2020).

The main improvement came from the data collection itself. One of the significant successes behind AlphaGoZero (Silver et al., 2017) is the massive self-play trajectories collection. **DL** in supervised setup requires huge amount of annotated data, **DRL** requires trajectories to evaluate the policy or update the value function. In the following section, we describe *Distributed* algorithms, a way to increase the number of interactions, in the same amount of time.

1.4.3.1 Distributed DQN

Gorila-DQN (Nair et al., 2015) defines a general distributed framework for **RL** and Ape-X **DQN** (Horgan et al., 2018) implements this idea in a distributed version of prioritize replay. Instead of having a single actor[2] collecting trajectories in the environment, about

[2]The term "multi-actor" is not to be confused with *Multi-Agent* learning (Shoham et al., 2008; Nowé et al., 2012)).

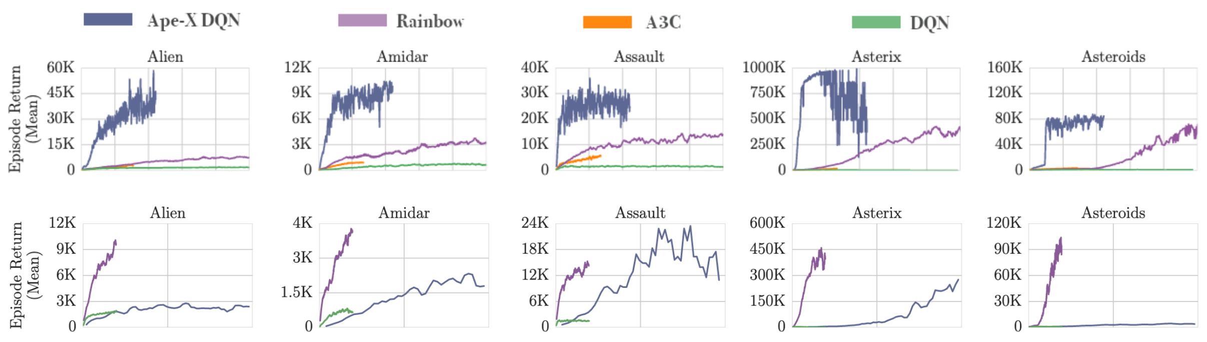


Figure 1.6: Comparison in term of time-efficiency (top) and sample-efficiency (bottom) between : Rainbow vs DQN vs Ape-X vs A3C. (Image Source: (Horgan et al., 2018) https://openreview.net/pdf?id=H1Dy—0Z)

a hundred actors (to thousands Espeholt et al., 2019) interact in parallel to fill the replay buffer and the main learner samples from this quickly updated buffer (see Fig. 1.5). It slightly reduces the *sample-efficiency* compared to Rainbow but greatly increases the sample throughput and *time efficiency* (see Fig. 1.6). R2D2 (Kapturowski et al., 2018) generalizes the idea to Recurrent DQN (**RDQN**) by having multiple actors collecting trajectories in parallel. The replay buffer is also modified to handle trajectories instead of single transitions. Finally, Espeholt et al., 2019 proposes an improvement over Gorila architecture to increase the communication speed between actors and the learner called SEED. In Gorila, each actor stores and computes its policy. Since actors are located on **CPU**, the inference time (computing the action) can be slow, especially for bigger models. SEED improves computational speed by moving everything related to the model on **GPU** while still maintaining a very high number of actors (see Fig. 1.8).

1.4.3.2 Policy Gradient and Actor Critic

Distributed data collection for policy gradient methods was first introduced with Asynchronous Advantage Actor-Critic (**A3C** (Mnih et al., 2016) or **A2C** for its synchronous version). To evaluate the current policy π multiple actors collect trajectories, combined with advantage estimation to reduce the gradient variance.

$$\begin{aligned}\nabla_\theta J(\theta) &\sim \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta\left(a_t \mid s_t\right)\left(r_{t+1}+\gamma V_v\left(s_{t+1}\right)-V_v\left(s_t\right)\right), \\ &= \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta\left(a_t \mid s_t\right) A\left(s_t, a_t\right).\end{aligned} \tag{1.15}$$

To accurately evaluate a policy π, trajectories should be sampled according to π; otherwise, a bias is introduced (Munos et al., 2016). To ensure this property, **A2C** waits for all the actors to complete their trajectories before updating the policy. Then, as soon as the update finishes, the learner sends a copy of the updated policy to the actors. Queuing circumvents *Policy Lag*, a discrepancy between the policy generating the data and the policy being updated. Nevertheless, this careful management brings a lot of computation overhead. Waiting for the actors to complete their tasks wastes time that could be used to update the policy.

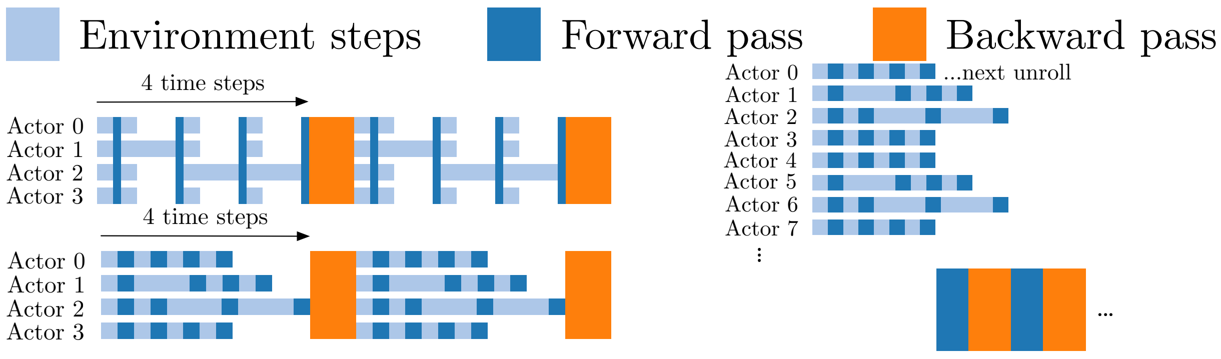


Figure 1.7: (Left) Two A2C versions, either synchronizing at every environment step and updating the learner (top) or waiting for every worker to finish their steps.
(Right) Actors/Workers send their trajectories continuously to a queue. Babaeizadeh et al., 2017 directly uses the trajectory and IMPALA (Espeholt et al., 2018) uses importance sampling to compensate for the discrepency between workers' and learner's policy. (Image Source: (Espeholt et al., 2018) https://arxiv.org/pdf/1802.01561.pdf

Instead of waiting for each actor, Babaeizadeh et al., 2017 creates a trajectories queue. Actors push trajectory as soon as an episode ends, and the learner continuously updates the policy, allowing for more updates and faster training. Authors increase the batch size to compensate for policy lag, however, they still observe training instability and high variance.

A more principled solution is to compensate for the discrepancy between policies using importance sampling (Rubinstein et al., 1981). Espeholt et al., 2018 effectively combines distributed architecture and importance sampling to reduce the bias introduced by policy lag (see Fig. 1.7).

1.4.3.3 Trust Region Update

Another way to increase the sample efficiency of distributed training is to use collected trajectories multiple times. As discussed in Section 1.4.4 and Section 1.4.3.2, off-policyness brings instability and leads to wrong updates. Updating a policy multiple times using the same trajectory means that after the first update, the policy changed, so the trajectory becomes off-policy. To avoid catastrophic updates, TRPO and PPO (Schulman et al., 2015; Schulman et al., 2017) constrain the policy to stay within a region of confidence called *trust-region*. TRPO uses a hard constraint, solved by a linear program, while PPO uses a clipping objective.

1.4.4 Exploration in Reinforcement Learning

So far, we discussed techniques to update the value/policy when given a trajectory and a stream of rewards. However, based on the TD(0) update rule 1.6 or policy gradient, in the absence of rewards, the policy can not be updated. Even when few rewards can be collected easily, should the agent stick to its strategy, *exploiting* its current knowledge? Or should it sacrifice immediate rewards and *explore* more substantially? This problem is known as the Exploration/Exploitation dilemma, faced by every decision-making agent. Classical techniques perturb the policy slightly to explore around the current knowledge boundary such Boltzmann Exploration, ϵ-greedy (Sutton et al., 2018)

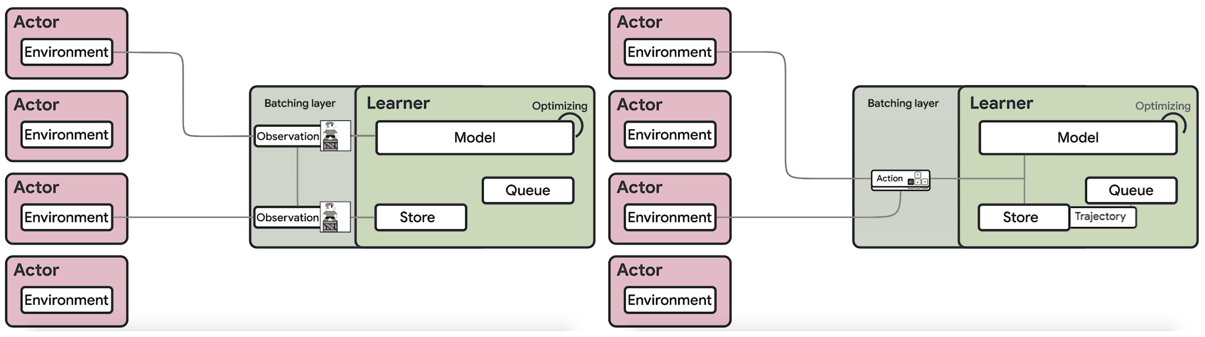


Figure 1.8: SEED Architecture. On the left, the observations are sent to the batching layer and the **GPU** learner computes the corresponding actions (right). Everything is stored in the replay buffer that continuously sends data for the optimization process (updating the policy). (Image Source: https://github.com/google-research/seed_rl)

or Entropy-Regularization (Haarnoja et al., 2017). A subsequent chapter dives deeper into exploration strategies (see Chapter 3).

On-Policy/Off-Policy Exploration requires deviating from the current policy to try other actions, either for a single step or a longer horizon. Certain methods handle this deviation naturally and can be updated nonetheless, such as Q-Learning. Those types of methods are called *off-policy* because the central policy can be updated even when the exploratory policy is completely different. On the other hand of the spectrum lies *on-policy* methods. They require unicity between the main policy and the exploratory one; otherwise, a bias is introduced (Munos et al., 2016). Methods such as SARSA and REINFORCE (Williams, 1992) refine the current policy, thus are on-policy. However, the degree of "on-policyness" varies from method to method. As described in Section 1.4.3.2, A2C-GPU (Babaeizadeh et al., 2017) is more on-policy than IMPALA (Espeholt et al., 2018) because importance sampling compensates for small differences between policies. DQN and its extensions is not completely off-policy (Fedus et al., 2020), as the authors point out "Reducing the oldest policy in the replay buffer improves performance", suggesting that learning from more on-policy data improves performance. To assess off-policyness, Batch **RL** or Offline **RL** (Wiering et al., 2012; Gulcehre et al., 2020; Fu et al., 2020) focuses on training agents with logged data in an offline fashion with no further environment interactions. This setup requires maximum off-policyness as policies can not be evaluated in the environment.

1.5 Chapter Conclusion

Deep Reinforcement Learning techniques allow training models on more complex tasks, bigger state space, and longer horizons but still face inherent difficulties. Final policies are brittle when dealing with new situations (Witty et al., 2018; Cobbe et al., 2019), and overspecialization makes them very sensitive to adversarial attacks (Gleave et al., 2019; Hussenot et al., 2020b). Distributed Reinforcement Learning pipeline is becoming more intricate for the sake of data collection. It increases algorithms' speed by orders of magnitude but deteriorates the sample efficiency (see Fig. 1.6). However, as Sutton, 2019 pointed out, methods that scale with the amount of computing stand the test of time. Increasing sample-efficiency while maintaining scalability should be at the heart of reinforcement learning methods. Changing model, adapting exploration strategies, changing how the transitions are dealt with, each component plays an important role in increasing the sample-efficiency. The following chapter will tackle this problem by focusing on how action spaces are handled and what could be improved to reduce the data-consumption.

Chapter 2

Of Actions And Constraints.

This chapter is the first stopover of our action-centered reinforcement learning journey. In the **RL** paradigm, action spaces lie at the lowest level of abstraction. It is the atomic level of interactivity. The following sections serve four purposes. First, to propose a taxonomy of action spaces for both continuous and discrete domains. Secondly, to define the notion of minimality and contextual ineffectiveness as tools to analyze large action spaces. Thirdly, to present embeddings methods, increasing the learning speed by transferring knowledge between similar actions. Fourthly, more exotic action spaces are covered, such as hybrid and stochastic action spaces.

2.1 Action Space Zoo

2.1.1 Domains example

Reinforcement Learning is a unifying view on the sequential decision-making problem. Algorithms are designed with generality in mind, and tools such as Rllib (Liang et al., 2018), ACME (Hoffman et al., 2020), Coach (Caspi et al., 2017) or Rlberry (Domingues et al., 2021) facilitate reproducibility by providing state-of-the-art algorithms. Uniformization on the environment side was also a requirement, and the gym library (Brockman et al., 2016) was a significant step in this direction. However, generality and standardization come at a certain cost and may hide a great deal of diversity. This section studies action spaces, the interface between the agent and its environment. We take a look at some classical environments and extract a general taxonomy.

Grid Worlds Starting point of almost any **RL** researcher, grid worlds are discrete domains where an agent is tasked to reach a specific destination (exit). State space being discrete, actions are also discrete and finite. Performable actions are moving directions, usually, NORTH, SOUTH, WEST, EAST, which can be executed one at a time. Recent instances such as Minigrid (Chevalier-Boisvert et al., 2019) complexify the state space by adding partial-observation, object encoding, and a variety of tasks such as instruction following and exploration challenges. The action space is also adapted. Since a state is a partial view, directions are relative to the agent angle. Thus, the action space becomes TURN LEFT/RIGHT, GO FORWARD. Additional actions related to objects are also added, such as PICK UP, TRIGGER, DROP.

Mujoco (Todorov et al., 2012) A second very popular benchmark is Mujoco, a physical simulator. Most tasks are based on moving rigid bodies (sticks, legs, ants, humanoid) with different joints. Actions are forces applied to joints at every timestep. Thus, the agent sends a scalar within a finite interval (usually $[-1, 1]$) for each joint.

Atari (Bellemare et al., 2013) The Arcade Learning Environment (ALE) was popularized with **DQN**. Since the Atari2600 is a real console, the interface and actions were standard across all games (55 different ones). The joystick selects an orientation (including diagonals) and could be combined with the button (triggering different effects, depending on the game). However, **RL** researchers adapted the action space for each game, simplifying the learning process but losing research opportunities to design algorithms that cope with this constraint.

Vizdoom/DMLab (Kempka et al., 2016; Beattie et al., 2016) Both environments are sets of first-person problems. Shooting opponents, discovering hidden maze parts, partial observability, and long-horizon are at the core of those benchmarks. Like Minigrid, the agent can move left, right, forward, diagonally (discrete actions) and combine them with some special actions, such as TOGGLE, OPEN, or SHOOT.

TextWorld (Côté et al., 2018) This environment is a set of textual games reminiscent of the 70s and 80s computer terminal game. Every interaction is textual; the agent receives a string describing the surroundings and inventory. For the action space, the agent selects a verb and an object. The number of actions thus depends on the vocabulary size, which can become quite significant in the most challenging instances. Nethack (Küttler et al., 2020) is a procedural dungeon, similar to TextWorld, where actions are a combination of a verb and an object.

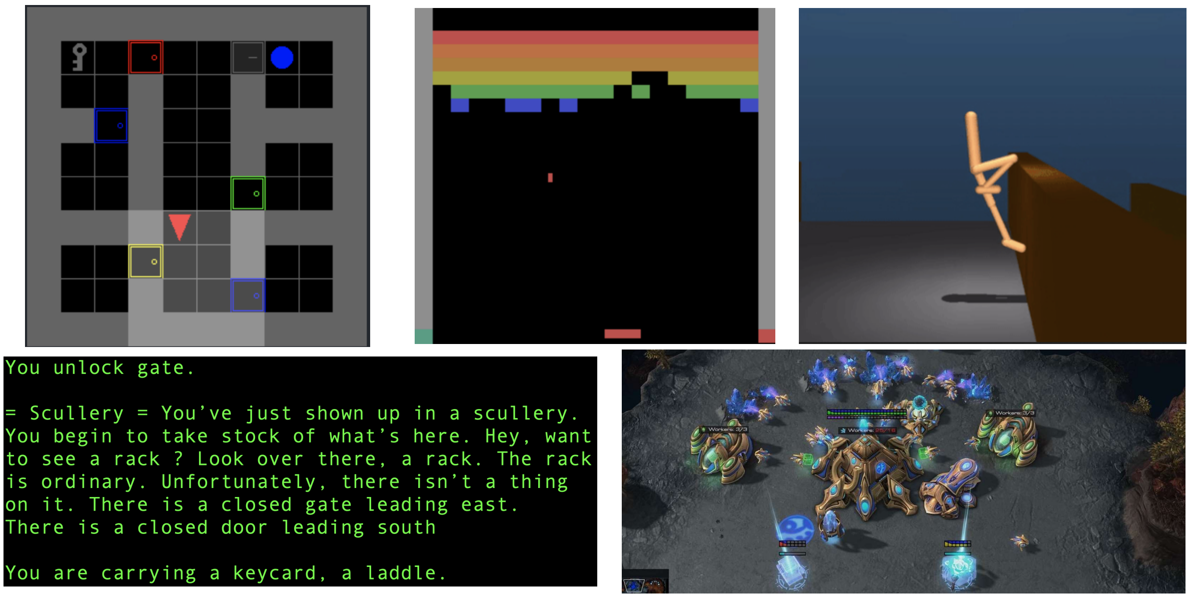
Starcraft/Dota Vinyals et al., 2017; Berner et al., 2019 Both environments are recent computer games where players confront each other, combining strategy and real-time skills. The action space is very diverse, where a player can move characters, build, buy items, cast spells, teleport, etc. The combinations are huge, creating an action space close to 10^{26} actions in Starcraft (Vinyals et al., 2019).

Environments are illustrated in Fig. 2.1. From the different benchmarks, we can draw some regularity. There exist spaces where actions can be enumerated, counted and spaces where actions lie on a continuous spectrum. Spaces where a single action is sent and spaces where actions are compound. The following sections address a more systematic taxonomy and analyze the algorithmic differences to solve each type of environment.

2.1.2 Discrete vs Continuous Domains

Action spaces are split in two general categories : Discrete Finite Action Spaces (**DAS**), and Continuous Bounded Action Spaces (**CAS**).

Discrete Finite Action Spaces (DAS) take two forms. The simplest one is an **Index Form**, where each action is an integer between 0 and $k - 1$ where k is the total number of actions. At each timestep, an agent chooses one action among this set. In simple grid



```
You unlock gate.

= Scullery = You've just shown up in a scullery.
You begin to take stock of what's here. Hey, want
to see a rack ? Look over there, a rack. The rack
is ordinary. Unfortunately, there isn't a thing
on it. There is a closed gate leading east.
There is a closed door leading south

You are carrying a keycard, a laddle.
```

Figure 2.1: (Left to Right, Top to Bottom) Minigrid (Chevalier-Boisvert et al., 2019), Atari game (Breakout) (Bellemare et al., 2013), Mujoco (Todorov et al., 2012), TextWorld (Côté et al., 2018), Starcraft (Vinyals et al., 2017)

worlds, four actions are available (up, down, left, right); the Arcade Learning Environment (Bellemare et al., 2013) uses between 4 and 18 discrete actions (game dependant), Dialog systems choose among a fixed number of utterance or word (Chandramohan et al., 2010).

The second type of action space is **Vectorial**. At each timestep, the agent acts by sending a vector of integers. In many applications, each dimension represents an independent component. For example, one dimension steers the horizontal axis, and the second dimension the vertical axis.

Traditionally, practitionners convert the vectorial form to a index one by taking the product of each dimension (Mnih et al., 2015; Kempka et al., 2016). However, the original structure is lost, and the combinatorial size might create issues (see Section 2.1.3). The other way around (index $\rightarrow$ vectorial) is not straightforward, Sharma et al., 2017; Phelps, 2020 group actions based on expert knowledge. For example, Sharma et al., 2017 transforms the action domain in Atari to have three independent axes (horizontal movement, vertical movement, fire).

Continuous Bounded Action Spaces (CAS) Similarly to **DAS**, **CAS** can have a single dimension (scalar continuous action) or be multi-dimensionnal (an action is a vector). Continuous space is usually associated with robotic setups (Todorov et al., 2012; Levine et al., 2016). Thus, **CAS** are bounded: they can only take on values within a finite interval due to physical constraints (Chou et al., 2017). All Mujoco (Todorov et al., 2012) environments falls under this category.

Different Action Spaces, Different Methods **DAS** can be tackled by any type of model-free methods : Value-based learning algorithms such as Q-Learning (Watkins et al., 1992; Mnih et al., 2015; Hessel et al., 2018), policy-gradient (Williams, 1992) or

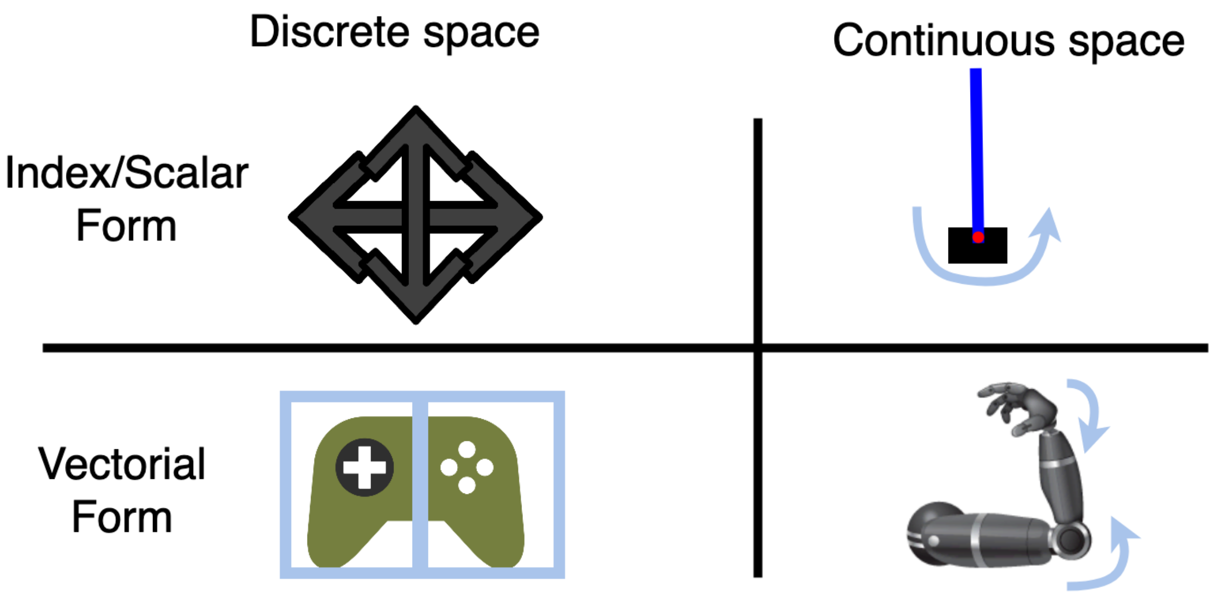


Figure 2.2: Visual Representation of **CAS** and **DAS** both in discrete and vectorial form (illustrated with two dimensions).

actor-critic methods (Mnih et al., 2016; Schulman et al., 2015; Schulman et al., 2017). On the other hand, **CAS** can only rely on policy-gradient and actor-critic (Lillicrap et al., 2016; Fujimoto et al., 2019). Value Learning algorithms can not tackle continuous domains in their current definition for two reasons. Greedy action selection is done by taking the value function's max. This max operation is not straightforward in very large or continuous domains, which makes greedy selection impossible. In Q-learning's case, boostrapping is also impossible due to the max operation in the update (see Eq. (1.7) and Section 2.1.3.6). And secondly, value-network's policy are implicit. Thus, selecting actions by sampling from the policy is impossible. Actor-critics are designed to solve this problem, the actor converging to an explicit sampler of the critic's distribution (Lazaric et al., 2008).

From Continuous Methods To Discrete Policy gradient methods (on-policy) are less sample-efficient than off-policy (value-based) methods (Ibarz et al., 2021). Value-based techniques benefit from bootstrapping and tend to have less variance, which is more efficient. Off-policy techniques (see Section 1.4.4) can benefit from re-using past trajectories to update the value function, which adds to the efficiency. One last problem regarding continuous domains, they are harder to explore. Discrete actions can easily be counted; thus, adapting the exploration to the time an action was taken is simple. However, in continuous domains, agents need to model an action density and explore less selected regions, which is challenging. Thus, continuous domains are harder to work with and practitioners, when possible, switch to discrete domains by relying on discretization (Jaśkowski et al., 2018; Andrychowicz et al., 2020; Tavakoli et al., 2018). Tang et al., 2020b compared PPO/TRPO with different discretizing-level to their continuous counterpart and show that **DAS** are more robust and achieve better asymptotic performance. However, discretizing requires careful tunings. A coarse scheme brings inaccuracy, and an overrefine increases exponentially the action domain. Recent articles propose an adaptive discretization for unidimensional continuous space but still need to be adapted for higher dimensional domain (Sinclair et al., 2019; Sinclair et al., 2020).

Designing action space Few theoretical analyses explore how shaping the action space impacts learning. An empirical evaluation, assessed how modifications to the action space alleviate the exploration problem (Kanervisto et al., 2020). The broad conclusion is that reducing the action space to the minimum necessary seems to be a viable option. **RL** algorithms struggle to discover contextual actions, actions that work only in certain contexts, as discussed in subsequent section (Section 2.1.3.4) and in a following contribution (Section 2.3). Removing those actions might prevent convergence to the optimal policy but alleviates exploration problems by reducing the search space. For example, capping the force in Mujoco might prevent an agent from running but reduces the problem of learning to stand or to walk. Overall, those reduction techniques are done by hand and set in stone, but they drastically impact learning. Data-driven methods replaced carefully designed feature engineering, and handcrafted action space should be addressed with the same spirit.

2.1.3 Tackling Large Action Spaces

While using function approximation, the bigger the action space, the more accurate the approximation needs to be (Thrun et al., 1993). As a result, model training is unstable and rarely used actions are overestimated (Bahdanau et al., 2016; Zaremba et al., 2016). This problem poses a significant challenge in many cases described earlier: Fine discretization, vectorial space, or environments where the number of actions is enormous such as recommender systems, language models, and industrial plants. We introduce three concepts to analyze the complexity of action spaces: Minimality, action similarity, and entanglement.

2.1.3.1 Space Minimality and Contextual Ineffectivness

A *minimal action* space is defined as a space that restricts its actions to the one used by the optimal policy π^*. Piot et al., 2017 elaborates on a similar notion called *associated set policy*. The associated set-policy of π indicates, for each state, the set of action that might be choosen by π. A minimal action space would be the union set policy of π^* (all the actions required to follow π^*). For example, the pause button will never be used in 99% of games, thus ignoring this action is beneficial. Some actions might be more subtlely useless, such as a steering angle never used in a driving game, but overall this type of action is not overrepresented in the current setup. Thus, we introduce a Local Action Space (**LAS**) $\mathbb{A}_s$ defining the set of available actions in state s. **LAS** can also be non-minimal if π^* is not using certain actions in s. Finding every local action space essentially means discovering the optimal policy, but quickly discovering some **LAS** and generalize to unseen state might alleviate sample-complexity.

LAS allows the definition of *Contextual Ineffectiveness* (**CI**) : actions which are non-effective in certain context. For example, in games where some interactions are triggered only in front of objects ('pick up' or 'open'). Discovering which actions are relevant in which context can drastically increase the sample efficiency by easing exploration. A follow-up section (Section 2.1.3.4) describes in which environment **CI** happens and a contribution (Section 3.2) addresses contextual effectiveness, using it as an exploration signal.

More generally, when considering a class of **MDPs**, an action set can be minimal for certain instances but not for others. For example, multi-task agents might use a set of actions to solve one task (opening doors) and leave aside inventory, while another task does not involve doors and requires picking up objects. Thus, taking into account the goal when considering contextual ineffectiveness should be of interest to future works.

2.1.3.2 Facilitating learning by leveraging action similarity

If many actions trigger similar outcomes, an agent wastes time trying all of them independently. Transferring knowledge between related actions is key to increase sample efficiency. Action similarity and contextual ineffectiveness are not completely orthogonal: If action A and B are similar and A is non-effective in particular contexts, it would be appropriate to transfer this knowledge to B. For example, in text-based games, if the action "pick-up sword" is not working, "pick-up katana" is likely to fail too. The following section on action embeddings addresses this problem.

2.1.3.3 Action Embeddings

Intuition from NLP and Vision Embeddings learning is a subset of representation learning (Bengio et al., 2013), it converts discrete variables to a continuous domain. For example, words in **NLP** are one-hot encoded. However, one-hot representation cannot encode semantic proximity between words. Thus, embeddings have been mostly used in **NLP** to organize words in a space where similarity and composition can be easily computed (Mikolov et al., 2013). Words like "crown" and "hat" should be close in this space, and "atomic" should be quite far. How to build such space from data? A simple word embedding method could take the following form: two words are converted from one-hot encoding to a continuous vector using a **MLP**. Both vectors are brought closer using a mean-square error loss if two words appear in the same sentence. In this example, the space is constrained directly, but Word2Vec (Mikolov et al., 2013) constrained the space using a downstream completion task and backpropagating through the network trains the embeddings.

Computing embeddings for *inputs* is now standard for many NLP tasks (Pennington et al., 2014; Brown et al., 2020) but computer vision methods (Akata et al., 2015) propose to embed *label* (or class) in a classification task. For example, "baboon" and "gibbon" should have a closer pixel distribution than "car," this type of information should alleviate learning and generalization to new classes. **RL** methods could benefit from the same general ideas. For example, in a dialog system (Chandramohan et al., 2010; Gao et al., 2018; Ferreira et al., 2013), each action is a dialog act (or a sentence). Algorithms could benefit from knowing that some actions are similar: *"Greetings!"* and *"Hello"* lead to closer outcomes than *"Calm down, take a pill."*

Using pre-computed embeddings in RL van Hasselt et al., 2009 describes a method to use a continuous domain to tackle a discrete action space. In mountain-car, instead of using a set of discrete forces, they compute the policy in the continuous domain and then discretize the action. It enables generalization between similar actions, which reduces sample complexity. However, the action space consists of a unidimensional steering force, which is already almost continuous and smooth. Thus, 1. finding the closest action is easy 2. The continuous domain already encode similarity. However, many domains require a different approach as action similarity can not be easily computed, and finding the closest discrete action can take time.

Dulac-Arnold et al., 2015 generalizes van Hasselt et al., 2009's method to higher dimensional domains, allowing any pre-defined embeddings. To do so, the policy outputs a continuous action and uses a nearest-neighbor algorithm (Fix et al., 1989) to find the closest discrete action. To pre-compute action embeddings, Tessler et al., 2019 uses predefined word embeddings (such as Word2Vec) and they compose actions using an Orthogonal Matching Pursuit algorithm.

The following section proposes to improve upon pre-computed embeddings or design algorithms that can learn them concurrently with the policy.

Learning Action Embeddings Tennenholtz et al., 2019 proposes Act2Vec, an embedding method using expert demonstrations. Similar to Word2Vec (Mikolov et al., 2013), they encode action with respect to their surroundings. Actions that appear in the same context should have a similar function, thus should be close in the representation space. They showed that action embeddings encode a notion of similarity between actions, and clusters represent high-level semantic concepts. They also propose to go beyond 1-step action embeddings and embed sequences of actions (see Fig. 2.3).

Chandak et al., 2019 learns an inverse model (predicting the action between s_t and s_{t+1})), using supervised learning and few collected trajectories to build an action representation. This constrains the embedding to contain information about the transition. Chen et al., 2019 refines this approach by using a probabilistic transition model. Finally, Pritz et al., 2020 builds an action representation coupled with state representation, learned alongside the policy. They also showed that embeddings trained alongside the policy were able to transfer quickly to new action domains.

We expect to see more work in this direction as action representation increases efficiency, generalization, and adaptation to new actions.

2.1.3.4 Discussion on Contextual Ineffectiveness

We previously described how to handle actions similarity, and now look to reduce such large action space by detecting and discarding useless action as defined in Section 2.1.3.1.

Formally, we defined *Contextual ineffectiveness* (**CI**) as actions that, in some situations, do not modify the state. Thus, learning to ignore this action set quickly is key to increase the sample efficiency of **RL** algorithms. Subsequent Section 2.3 and Section 3.2 are framed under this paradigm.

Availability known before acting Some environments already remove unavailable actions when not required by the context. Thus, at every time step, the action set can vary. For example, some Dialog System, depending on the conversation's stage, propose a limited amount of utterances. To cope with this challenge, He et al., 2016b; He et al., 2016a propose a Q-Learning approach. They compute a state, and action embedding for each action available, then perform a dot product between the two. Since the number of actions is small, action selection and bootstrapping using an argmax are quick.

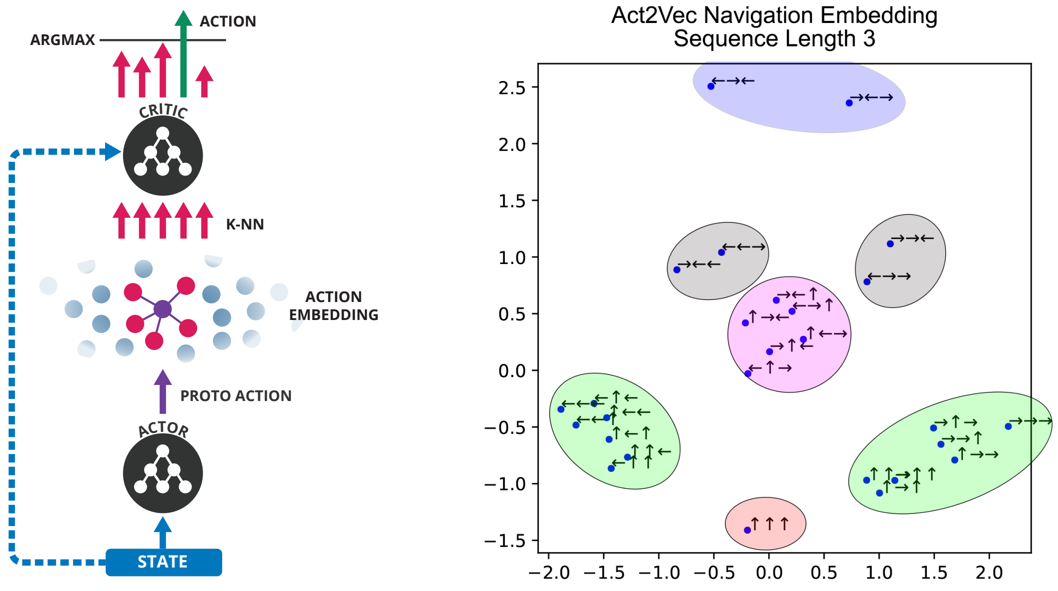


Figure 2.3: (Left) Actor-critic architecture using action embeddings by Dulac-Arnold et al., 2015. (Right) Example of action embeddings using Act2Vec in a simple navigation domain (Tennenholtz et al., 2019)

Availability known after acting Zahavy et al., 2018 eliminates actions based on a signal given by the environment, indicating if a performed action was valid or not. Rapidly discovering which action will not be valid is essential to complete the environment. They learn a *validness model* and use a contextual bandit to assess its certainty. Then, they remove actions from the action set when the confidence is above a certain threshold.

Huang et al., 2020 provides an empirical analysis of the last two setups. They compare two methods: invalid masking and penalty masking. When the availability is known, they mask the invalid actions when querying and updating the policy (invalid masking). They compare it to penalizing the policy using a small negative reward *after* the action is performed (penalty masking). They conclude that actor-critic methods can learn when masking the policy logit and even increases sample efficiency compared to penalizing invalid actions.

Availability not known Alshiekh et al., 2018 considers environments where an external system rejects potentially harmful actions. The agent outputs a set of actions, ordered by preferences, and the simulator picks the best-allowed action.

Previous papers treat actions that are contextually ineffective as harmful, thus try to avoid them quickly. In a subsequent part (Section 3.2), we propose an algorithm that detects contextual ineffectiveness and rewards the agent when it finds out in which context the action is useful. The algorithm reduces ineffective calls and increases relevant action, increasing the sample efficiency and exploration.

2.1.3.5 Independency vs Entanglement

We describe two ways to analyze action space: action similarity and space minimality. A third aspect is dimension independence that we formalize in this section.

For example, Sallans et al., 2004 describes a bit matching task where the goal is to find a binary code called a *key*. Actions are binary vectors, and the reward counts the matching bits. If all bits are correct, the agent receives the maximum reward. If half of the bits are right, it halves the reward, etc. Thus, bits can be selected independently of other bits, and the relation between action and reward is straightforward. On the contrary, it would be much harder to learn an action space where changing a single bit change drastically the reward. This bit matching example proposes a second important notion to analyze action space: the entanglement (or its opposite, the independence).

Factored-MDPs (Dean et al., 1998; Sallans et al., 2004) are a particular case of vectorial spaces (continuous or discrete) and specifically deals with independent actions. The term describes weakly-coupled MDPs (Meuleau et al., 1998), meaning that each action controls an independent part of the environment. Factored-MDPs assume an already disentangled action space (Peshkin et al., 1999; Meuleau et al., 1998; Guestrin et al., 2002; Dulac-Arnold et al., 2012; Cui et al., 2016; Pierrot et al., 2021) or (Metz et al., 2017) in continuous domain. Other work design neural architecture to tackle this problem (Yoshida, 2015; Tavakoli et al., 2018; Song et al., 2019).

Nevertheless, very few papers try to simplify the action space by finding independent components. Sharma et al., 2017 transforms the action domain in Atari to have three independent axes (horizontal, vertical, fire). The transformation is done by hand, but this structural change alleviates the learning problem. Action embeddings are one way of giving structure, but constraining vectorial spaces to have orthogonal components is an exciting topic that future work should address.

2.1.3.6 Max Operation Over All the Actions is Long

In large action spaces, Q-learning quickly faces scalability issues as mentioned in Section 2.1.2. Indeed, such methods face prohibitive cost of the max operator, both to extract the policy and for bellman backups. The problem occurs both in continuous action settings and in large discrete spaces. Alternative methods try to model the distribution induced by the critic using Gibbs sampling (**kimura2007multi**) or Sequential Monte Carlo (Lazaric et al., 2008). Other works rely on cross-entropy method (Lim et al., 2018), Mixed Integer Programming (Ryu et al., 2019) to perform the maximization step. (Van de Wiele et al., 2020) propose to learn a function that reduces the action set considered instead of iterating over all the actions.

Lastly, Pazis et al., 2011 proposed a different value function to avoid the max operation. Q is an exhaustive action enumeration, and V is the extreme action aggregation (average overall a's). They proposed a H function that aggregates over a subset of actions.

2.1.4 Working with Hybrid Action Spaces

Hybrid spaces are vectorial, but some dimensions are discrete, and others are continuous. Stochastic Actor-Critic and Policy Gradients methods naturally handle such spaces (Neunert et al., 2020), but pure-critics do not because of the continuous dimensions, see Section 2.1.2. However, some methods proposed in Section 2.1.3.6 work nicely in this context (Van de Wiele et al., 2020) as they generalize Q-learning to continuous domain.

Hausknecht et al., 2015; Masson et al., 2016; Xiong et al., 2018 propose methods for a particular instance called *Parametrized*-**MDP** : A discrete number of actions, but each action is parametrized by a real value. All methods combine a discrete algorithm and a continous one. The same strategy is used in the large scale Dota environment (Berner et al., 2019).

2.1.5 More Exotic Action Spaces

To conclude this overview, we present variations of the action spaces that do not fit in categories and methods abovementioned. Sunehag et al., 2015 defines Slate-**MDP**'s, a type of vectorial action space where multiple actions are sent at the same time. The central point of slate-MDPs is that the agent chooses a tuple of actions instead of taking one action. Then the environment chooses which one to execute in the underlying MDP. They take as an example a recommendation system. An agent proposes a set of items, but the user chooses whatever he requires within this set; thus, the action selection is not 100% in the agent's hand.

Boutilier et al., 2018 proposes a type of MDP where the action availability is stochastic. To say it differently: in the same state, actions availability can vary. Suppose one wants to use Q-learning directly. Each state is augmented with all possible subsets of actions. In that case, it creates an exponential blow-up in state space size. Thus, Stochastic MDP's require tailored methods.

2.1.6 Conclusion

In this section, we introduced a taxonomy of discrete and continuous space, both taking a scalar and vectorial form. We studied other forms of action spaces such as hybrid, slate, and stochastic and surveyed the problem of large action spaces. Concepts to study action spaces are sketched, such as space minimality, entanglement, and action similarity. The following contribution builds upon this survey as Section 2.3 and Section 3.2 are placed under the contextual ineffectiveness analysis. In Seurin et al., 2020a, availability is known after acting and modifies DQN to integrate this information. In the next sections, we show how it complements safe and constraint **RL** methods to foster safer and quicker learning.

2.2 Deep Garcia and Fernandez

The more constraints one imposes,
the more one frees one's self.

Igor Stravinsky

In this section, we take a look at *constraint* reinforcement learning. Constraining reinforcement learning might be beneficial for two reasons : first, for safety reasons. by avoiding certain hazardous state or reaching an objective while minimizing the energy consumption, **RL** could be applied to real world system with greater confidence. Secondly, constraining may reduce the exploration space and thus, focus on relevant parts of the environment.

In Section 2.2, we update García et al., 2015 survey by giving recent directions that combine *safe reinforcement* learning and non-linear function approximation. We deem an update necessary as **hard**-constraining and guaranteeing that neural networks will behave correctly in unseen situations is still an open problem (Goodfellow et al., 2015; Pei et al., 2017; Silva et al., 2020).

2.2.1 Worst Case Min Max formulation

In critical applications, such as healthcare and energy management, considering the most harmful situation is essential. The average reward might hide unlikely but highly hazardous events. Let us illustrate with an automatic surgeon application. The reward received by the surgeon is proportional to the speed at which it operates. A careful surgeon A takes between 95 and 105 minutes to intervene on a patient. One of its caffeinated colleague (B) takes 20 minutes to perform the same operation, but 25% of the time, an over-energetic cut causes complication that takes 340 minutes to fix. They both take on average 100 minutes to intervene, but it seems reasonable to prefer A over B. The first category of methods that fall within the safe **RL** class modifies the average reward criterion to consider such variations. For example, Markowitz, 1959 introduces a penalization using the variance (σ) of the return, creating an objective taking the following form : $\max_\pi E^\pi\left[\sum_{t=0}^\infty \gamma^t r_t\right] - \sigma^\pi\left[\sum_{t=0}^\infty \gamma^t r_t\right]$.

Other methods consider *worst-case* scenarii. Anticipating such situations is tedious as designing every edge case is untractable. Instead of carefully designing scenarios for each application, *adversarial rl* methods learn the optimal policy alongside an adversary. The adversary tries to exploit the agent's weaknesses by coming up with the hardest situation (either applying small modification to the state or leading the agent into complicated state).

The adversarial setting can be expressed as a two-player, zero-sum game, called *Markov Game* (Littman, 1994; Perolat et al., 2015) in the **MDP** formalism. Morimoto et al., 2000 use this formulation to learn against a *disturbing* agent. The disturber tries to perform the worst possible state modification. At the same time, the control agent acts stoically with respect to noises and optimally with respect to rewards.

2.2.2 Risk-Sensitive Criterion (CVaR)

Adversarial attacks can be arbitrarily difficult, and being robust to every perturbation is impossible (Gleave et al., 2019; Hussenot et al., 2020b; García et al., 2020). Thus, most methods rely on a relaxed worst-case criterion, the Conditionnal Value at Risk (CVaR Artzner et al., 1999; Rockafellar et al., 2000). Instead of lower bounding the worst possible (but very unlikely) incident, CVaR methods maximize a percentile of the reward distribution. It effectively discards near-impossible outcome, but still deals with more frequent events associated with low rewards (or high cost).

Chow et al., 2015; Chow et al., 2017 surveyed CVaR reinforcement learning and proposed a minmax formulation, analoguous to the worst-case setting (people with a control theory background might want to check (Tamar et al., 2016)). Methods of this class tackle the CVaR objective using policy-gradient updates to fit non-linear function approximation. For example, Pinto et al., 2017 adapts the TRPO objective to stay within a region of safe update, Tang et al., 2020a; Singh et al., 2020 use the distributionnal setting (Bellemare et al., 2017) and Rajeswaran et al., 2016 use an ensemble of model.

2.2.3 Constrained Criterion

A Constrained Markov Decision Process (**CMDP** Altman, 1999) is an **MDP** augmented with constraints that restrict the set of possible policies using additional cost functions, like usual reward R.

Although optimal policies for finite CMDPs with known models can be obtained by linear programming (Simão et al., 2021), methods for high-dimensional control are lacking. Achiam et al., 2017a proposed a constrained policy optimization algorithm. It adapts the TRPO objective to ensure the policy satisfies constraints after each update. Pham et al., 2018; Dalal et al., 2018 project $\pi(s)$ within regions that satisfy constraints. They achieve good performance on robotics tasks and keep the policies within constraints but still need to demonstrate robustness in richer domains.

A special case of **CMDP** called Budgeted-MDP (Boutilier et al., 2016) where actions cost a certain amount of budget, that the policy needs to take into account. Carrara et al., 2019b proposed to tackle this problem by learning concurrently multiple policies for different budget amounts.

2.2.4 Conclusion

We briefly presented different optimization criteria to address safe reinforcement learning. They are a necessary step to tackle real-world applications. However, *"It is impossible to completely avoid undesirable situations in high-risk environments without a certain amount of external knowledge"* - (García et al., 2015). The survey presents ways to incorporate external information such as Imitation, Teacher Advice, or Exploration (the latter, covered in Chapter 3). To complement those approaches, we want to incorporate knowledge on the action space inspired by industrial applications.

2.3 First contribution (IJCNN'19): I'm sorry Dave, I'm Afraid I Can't Do That - Deep Q-learning from Forbidden Actions

Real-world environments (*e.g.* industrial robots or power grids) are generally designed with safety constraints in mind implemented in the shape of valid actions masks (see Section 2.1.3.4) or contingency controllers. For example, over-temperature monitoring regulates servo-motors present in robots, disabling motors when approaching their heat limit. Cleaning robots also automatically u-turn when facing an obstacle, preventing damage to the machine and its environment. Violating constraints thus results in rejected actions or entering in a safe mode driven by an external controller, making RL agents incapable of learning from their mistakes.

Beyond critical systems, many areas could benefit from *forbidden actions*. In Natural Language Generation (Reiter et al., 1997) or Dialogue systems (Chandramohan et al., 2010; Chen et al., 2017), syntax parsers or auto-correct mechanisms can act as an external rejection signal. Indicating which word does not fit a generated sentence or pointing out grammar mistakes could improve language generation by greatly reducing the search space, leveraging language learning and generation. These examples show a potential misalignment between the standard RL frameworks and the potential real-world applications. Of course, designing constraints to avoid critical situations requires expert knowledge about the system to be controlled. However, we argue that many environments already implement such contingency measures (obstacle avoidance, circuit breaker).

Thus, we consider a simple type of external constraint prevalent in many real-world problems. When the agent is about to perform a hazardous action, the system rejects it and prevents it from doing so. The agent then follows the natural dynamics of the environment (Alshiekh et al., 2018). We aim at building an algorithm that learns from these rejected actions.

In the general Markov Decision Process (MDP) framework (Puterman, 2014), a rejected action, from the agent's point of view, would be seen as a transition to the same state. Everything happens as if the action had no effect. In earlier section, we coined the term Contextual Ineffectivness (Section 2.1.3.1). We argue that it misrepresents the potential harmfulness of the action and prevents the agent from learning (See Section 2.4.1 for a detailed explanation). Especially, we want Q-learning algorithms to benefit from those constraints. They would learn faster to avoid hazardous states and alleviate exploration problems.

Learning when the environment takes over Orseau et al., 2016 design agents to *not* take into account feedback from the environment. For example, for an agent operating in real-time, it may be necessary for a human operator to prevent executing a harmful sequence and lead the agent into a safer situation. However, if the learning agent expects to receive rewards from this sequence, it may learn in the long run to avoid such interruptions, for example, by disabling the *off-button*. Under this setup, they showed that Q-learning could be interrupted safely, supporting our hypothesis that Q-learning is not integrating the feedback signal.

In the coming sections, we propose a constrained version of Deep Q-learning (DQN) (Mnih et al., 2015) by adding a classification loss that maintains Q-values of forbidden actions below valid ones. We validate our method empirically on two tasks: A maze navigation using visual features and a text-based game. Experiments show that vanilla DQN struggles at solving tasks with rejected actions. Our algorithm reduces the number of calls to forbidden actions and accelerates convergence to near-optimal policies compared to standard DQN.

2.4 Method

2.4.1 Feedback Signal and MDP-$\mathcal{F}$

In this section, we formalize how to incorporate forbidden actions in the MDP framework. We define a *Feedback Signal*, a Boolean indicating whether an action was accepted by the environment or rejected. A MDP-$\mathcal{F}$ is then defined as a tuple $< \mathbb{S}, \mathbb{A}, P, R, \gamma, \mathcal{F} >$ where $\mathcal{F}$ is a function mapping a state s_t and action a_t to a binary value.

$$\mathcal{F} : \mathbb{S} \times \mathbb{A} \rightarrow \{0, 1\},$$

with 0 meaning the action is valid and 1 meaning unsafe/rejected action.

Vanilla Q-learning struggles to differentiate between actions flagged as forbidden and valid ones. Consider the following example: an agent in a state s takes action a flagged as for-

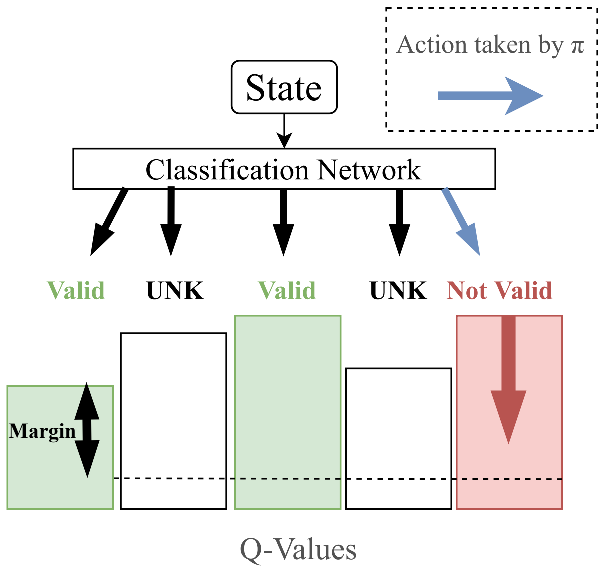


Figure 2.4: Illustration of frontier loss.

bidden ($\mathcal{F}(a, s_t) = 1$). When applying the Q-learning update ($Q(s, a_{feed}) = r(s_t, a_{feed}) + \gamma \max Q(s_{t+1}, a_{t+1})$), since the action was rejected, $r(s_t, a_t) = 0$ and $s_{t+1} = s_t$. Thus the update becomes $Q(s, a_{feed}) = \gamma \max Q(s, a')$. In current Deep Reinforcement Learning setup γ is usually set between 0.99 (Mnih et al., 2015) and 0.999 (Pohlen et al., 2018). DQN-like algorithm will require lots of transitions to make the Q-function of forbidden actions smaller, thus wasting time to explore and collect useful samples. We emphasize that an invalid action indicates an action that could be harmful, so rapidly identifying and avoiding those potentially dangerous situations is crucial.

2.4.2 Frontier loss

We take inspiration from the learning from demonstrations paradigm where one wants to use expert demonstrations to induce the usage of preferred actions in RL agents. For example, Piot et al., 2014; Hester et al., 2018 slightly modify the Q-learning update to nudge expert actions-value above other actions.

The optimal policy π^* (derived from Q^*) will never take a forbidden action as it keeps the agent in the same state. Based on this assumption, we can derive the following rule: for every state encountered during training, the Q-values of all forbidden actions should be below the one of each valid actions, within a certain margin m. This defines a new loss we want to minimize that we call *frontier loss* $J_{\mathcal{F}}$:

$$J_{\mathcal{F}}(Q) = Q(s, a^-) - \min_{a \in \mathcal{V}_s} [Q(s, a) - m], \tag{2.1}$$

$$\begin{aligned} \text{where } \mathcal{V}_s = \quad & \mathbb{A} \cap \{a \text{ s.t. } , \mathcal{F}(s, a) = 0\}, \\ \text{and } \ a^- \in \quad & \mathbb{A} \cap \{a \text{ s.t. } \mathcal{F}(s, a) = 1\}. \end{aligned}$$

The margin m is an hyper-parameter that depends mostly on the Q-values magnitude. In our experiments, since the rewards are bounded between 0 and 1, the margin is small

($m = 0.1$). Zaremba et al., 2016 uses a similar frontier technique but only for binary Q-values (0 or 1).

Frontier loss and classification The main problem regarding this objective function is the need to know which actions are valid for every state. In most tasks, it's unlikely that the agent visits a specific state more than once (*e.g.* visual domains). Thus, function approximation is required to estimate which actions are valid in a given state. To achieve this, we train a neural network to predict, for each state, which action will be **valid**. Along agent's trajectories, for every action taken, we store the corresponding feedback, creating a dataset of $< s, a, F(s,a) >$ tuple. The network, taking the state as input, predicts a binary value for every action (0 for valid, 1 for invalid). For each state, since only one action is labelled, we need to adapt the training regime. We can achieve this by masking the gradients from untaken action, only backpropagating for the action the policy π took. The training procedure is illustrated in Fig. 2.5.

To consider an action as valid and to avoid early mis-classifications, we put a threshold after the *sigmoid* function. The action is considered valid if its activation is above the threshold.

DQN-$\mathcal{F}$ The resulting algorithm is DQN-$\mathcal{F}$, combining the frontier loss and Deep Q-learning. We build a composite loss by using weighting factors η_{DQN} and $\eta_{\mathcal{F}}$ to balance the DQN and the frontier losses:

$$J(Q) = \eta_{\text{DQN}} J_{DQN}(Q) + \eta_{\mathcal{F}} J_{\mathcal{F}}(Q).$$

For all the experiments described below, we use $\eta_{\text{DQN}} = 1$ and $\eta_{\mathcal{F}} = 0.5$.

Algorithm 1: Frontier loss and classification network.

```
Data: minibatch $b$ from replay buffer $\mathcal{R}$, $Q$-network $\mathcal{Q}$, classification network $\mathcal{C}$
Result: Frontier loss
1  loss = 0;
2  for (state $s$, action $a$, feedback $F(s,a)$) in minibatch $b$ do
3      if $F(s,a)$ = 1 then
4          $a^-$ ← a                                         ▷ Renaming for clarity
5          valid_set ← C(s);                                 ▷ Estimated valid action set
6          minQ_valid ← $\min_{a_i \in \text{valid\_set}}$[Q(s, $a_i$)];   ▷ Smallest Q for valid actions
7          if minQ_valid < Q(s, $a^-$) - m then
8              loss = loss + || minQ_valid - m - Q(s, $a^-$) ||$^2$
9                                   ▷ Bring Q(s, $a^-$) below the smallest valid with margin
10 return loss;
```

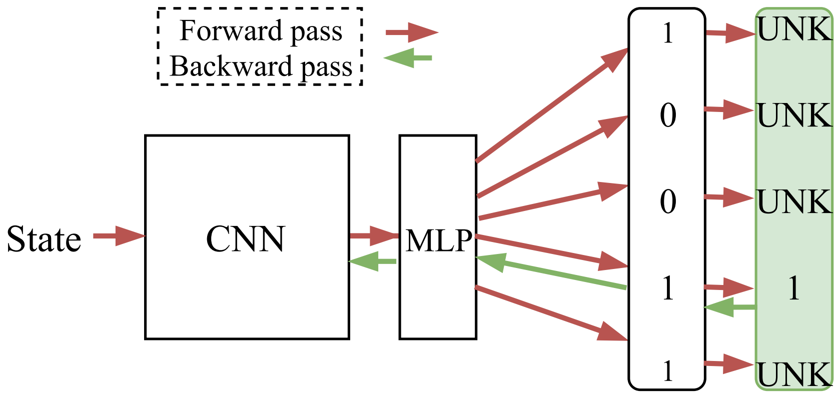


Figure 2.5: Training procedure: the model predicts the validity for each action, and we only backpropagate for the action the agent took.

2.5 Experiments

2.5.1 MiniGrid Enviroment

The first environment is a visual gridworld called Minigrid (Chevalier-Boisvert et al., 2019). The goal is to reach the green zone starting from a random point (see Fig. 2.6). Since we want to study how the agent can integrate feedback about the validity of actions, we increase the action space size. To do so, we create k different room types where the color of the background indicates *which set of actions is valid*. The primary action space is composed of 3 actions (Turning Left, Turning Right, Going Forward) for navigation, but each action is duplicated k times. The action space size becomes $3 \times k$ but only 3 are valid in a given room. For example, in the red room, only actions 11, 12, 13 are valid, and all the others are returning a **not valid** feedback. In our setup, we use $k = 5$ making a total of 15 actions.

The state space is an embedding of the agent's point of view represented as different feature maps allowing the use of convolution layers (more details in Chevalier-Boisvert et al., 2019). Since the environment is partially observable, we stack the last three frames as in Mnih et al., 2015 but we do not use frame-skipping. An episode ends when the agent reaches the green zone or after 200 environment steps (illustrated in Fig. 2.6).

2.5.2 TextWorld Environment

TextWorld (Côté et al., 2018) is a text-based game where the agent interacts with the environment using a verb and an object. States are textual description of the agent's surrounding and the inventory content. Games can be generated, varying in difficulty by changing the number of rooms and number of action before reaching the objective, varying the number of random objects within each rooms etc. We generated a game composed of 3 rooms, 7 objects, and quest length of size 4. An example is shown in Fig. 2.7. In this context, we modified the environment to fit our needs. The action space is composed of all <action> <object> pairs, creating a total of 46 actions. Most of the actions created will be rejected by the simulator since they will not fit the situation the agent is facing. For example, the action "take sword" will be rejected if no sword is available.

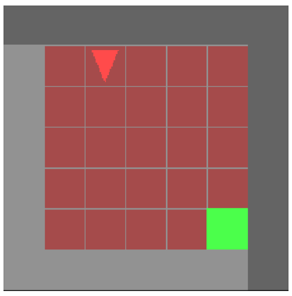
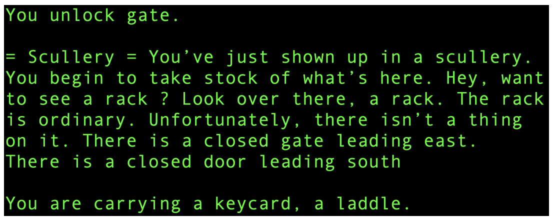


Figure 2.6: An instance of the MiniGrid problem. The state is a partial view of the maze (point of view of the agent) to avoid problem regarding partial observability, we stacked the last 3 frames.



```
You unlock gate.

= Scullery = You've just shown up in a scullery.
You begin to take stock of what's here. Hey, want
to see a rack ? Look over there, a rack. The rack
is ordinary. Unfortunately, there isn't a thing
on it. There is a closed gate leading east.
There is a closed door leading south

You are carrying a keycard, a laddle.
```

Figure 2.7: An example of interaction in TextWorld. The agent has access to: what happened after its latest action ("You open the door, it's very dark in here, [...]"), a room description ("Attic, an empty room, maybe you should head back. You can go North, East") and its inventory content ("Keycard, Mask").

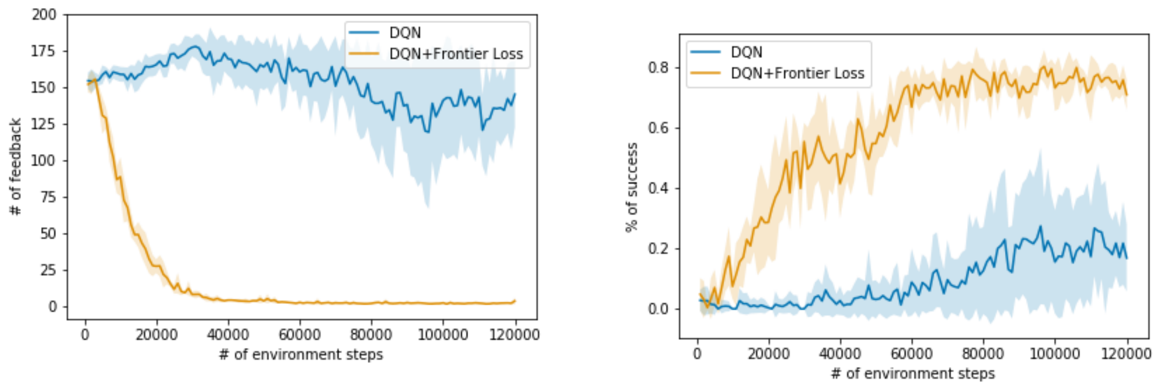


Figure 2.8: DQN-$\mathcal{F}$ (yellow) DQN (blue), the shaded area represents one standard deviation. Results are averaged over 5 random seeds. **Left**: Number of times a forbidden action is taken. **Right**: Percentage of success over time.

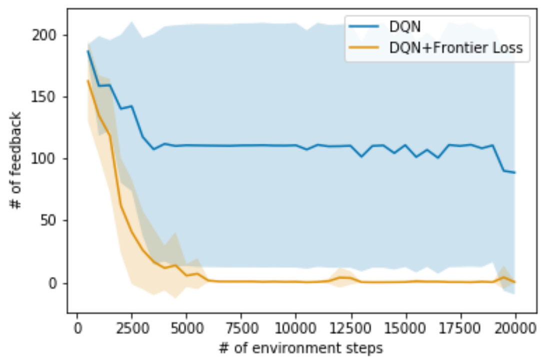
2.5.3 Model and architecture

During all experiments, we use Double Deep Q-Network (DQN) (Hasselt et al., 2016) with uniform Experience Replay and ϵ-greedy exploration. In the Minigrid environment, we use a Convolution Neural Network (LeCun et al., 1995) with a fully-connected layer on top. In TextWorld, inventory, observation, and room descriptions are each encoded by an LSTM (Hochreiter et al., 1997) processed by a fully-connected layer on top.

The classification network matches exactly the architecture used by DQN, *i.e.* ConvNet for Minigrid and LSTM's for TextWorld, the only difference resides in training (explained in Fig. 2.5).

2.6 Results

In Fig. 2.8 and Fig. 2.9, we compare DQN and DQN-$\mathcal{F}$. In the Minigrid domain, DQN struggles to find the optimal policy and reaches the exit only 20% of the time. Most of the time, DQN is able to solve one room but fails to find the set of actions for each room,



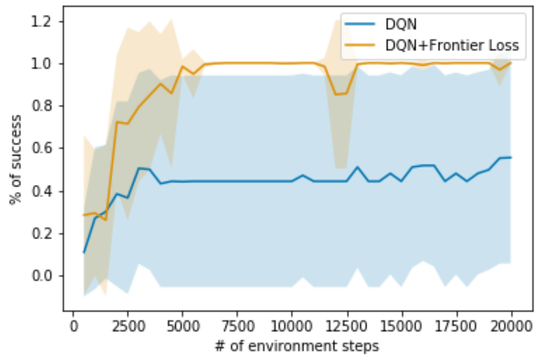


Figure 2.9: DQN-$\mathcal{F}$ (yellow) DQN (blue). Results are averaged over nine random seeds. The shaded area represents one standard deviation. **Left** Number of invalid actions taken by the agent. **Right** Percentage of success over time.

performing forbidden actions over and over. On the contrary, the frontier loss is guiding DQN-$\mathcal{F}$, reducing the number of feedback signals from the environment, and helping to find the optimal policy. Those results are echoed in the TextWorld experiment. DQN solves the game half of the time, and the other half does not encounter the reward and as a result, can not solve the game. This could be mitigated by having a better exploration strategy, but it shows that shaping Q-values with the frontier loss is enough to reduce the sample complexity and guide exploration. We want to emphasize that in early stages of training, the classification network performs poorly due to low quantity of samples but it does not hurt the performances of DQN-$\mathcal{F}$, it is able to quickly learn to avoid forbidden actions. Visualization of Q-values at different stages of training can be found in Fig. 2.10. They clearly illustrate the benefits of using the frontier loss in those setups. Even in the early training stage, the separation between valid actions and invalid is clear, alleviating the difficulty of finding the optimal policy. Whereas DQN Q-values are really hard to distinguish from each other.

2.7 Conclusion

Critical real-world systems are constrained by safety measures that prevent hazardous actions. We hypothesize that Q-learning, a model-free reinforcement learning method struggles with those constraints. In this chapter, we proposed a frontier loss, combined with a classification network, to help DQN. This algorithm nudges rejected actions Q-values below Q-values of valid actions. We showed empirically that the frontier loss reduces the number of calls to rejected actions and guides early exploration, helping Deep Q-learning achieving higher performances. We demonstrate its effectiveness on two benchmarks, a visual grid world and a TextWorld domain.

Broader scope At the moment, applying the frontier to continuous action space is non-trivial but it would be a key to use this type of algorithm in robotics and more realistic settings. Another future improvement would be to combine the loss with action's embedding could allow generalization to unseen actions. For example, learning that "Take sword" is rejected "Grab sword" shouldn't be considered by the algorithm.

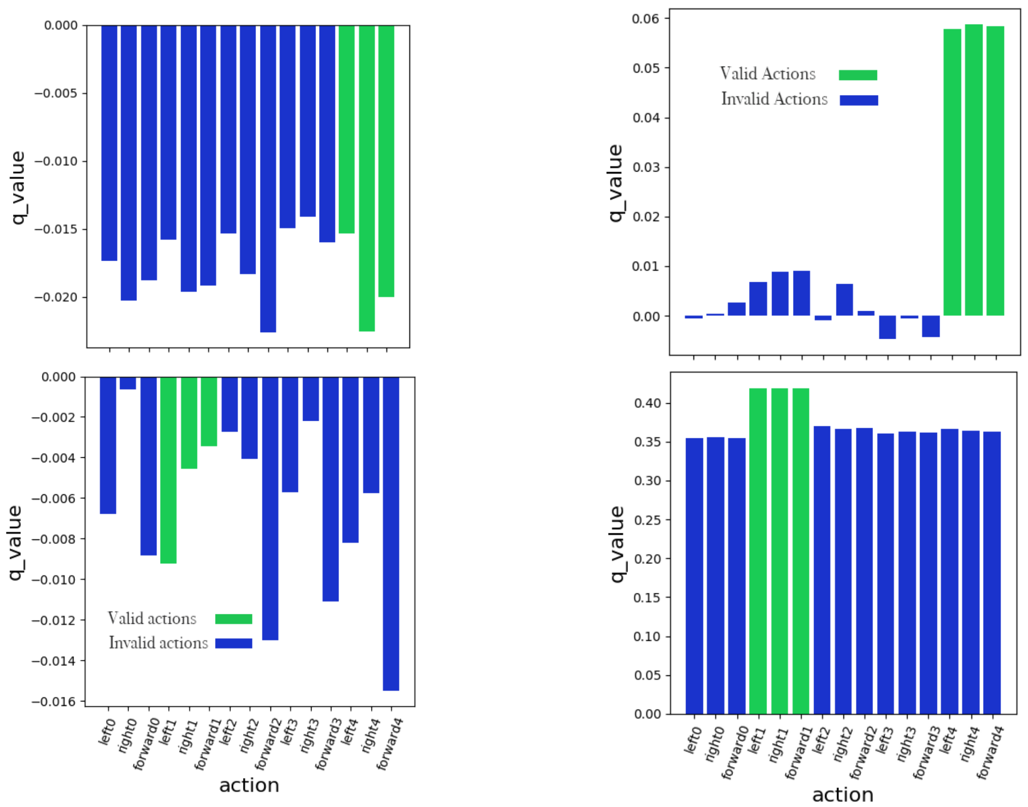


Figure 2.10: **Top:** Q-values, after 150 episodes of training (30 000 environment steps) for a handpicked state. **Bottom:** Q-values, after 100 000 training steps on another state. **Left:** DQN; **Right:** DQN-$\mathcal{F}$.
We can clearly see how the frontier-loss shapes Q-values, separating forbidden action (blue) from good ones (green). This cherry-picked example illustrates well how the Q-distribution is modified, even at that early training stage.

Chapter 3

Exploring using Action Relevance

Lost in the forest, an agent must find ways to survive. A small house is near him, with few apple trees (dense reward, easily accessible). The agent can exploit its ressources until the end or try to explore the forest more deeply. Deeper exploration is more dangerous, as many paths lead to dead trees and hostile animals, but potentially lead to better ressources such as meat, bigger fruits etc. (sparser rewards).

Exploration has thus been one of the longest-running problems of RL (Mozer et al., 1990; Sato et al., 1988; Schmidhuber, 1991; Barto et al., 1991). This chapter sketches a taxonomy of current explorations methods and link them to traditional reinforcement learning strategies (TL;DR in Fig. 3.1). Secondly, by focusing on the actions and their consequences, we propose a method that exploits knowledge about the action space, called **Don't Do What Doesn't Matter**.

3.1 Press E to Explore

Instead of chronologically enumerating methods, we will rely on the classification first proposed by Thrun, 1992 and refine each category as we go. The broader taxonomy proposes two categories: **directed** and **undirected** exploration. *Undirected* methods do not use any domain knowledge and ensure exploration by introducing stochasticity in the agent's policy. In many simple environments, dithering allows enough deviation from the current policy to collect rewards spread in the environment. On the other hand, *Directed* methods try to explore the space more methodically, relying on policy and environment uncertainty, state count, etc.

3.1.1 Undirected Methods in Exploration

The two most common approaches are **Epsilon-greedy**, forcing random actions with probability ϵ and taking the optimal action with probability 1-ϵ, and **Boltzmann exploration** where an agent draws actions proportionally to the Q-values magnitude. The agent will pick up actions associated with higher Q-valuesf more often.

Although they enable learning the optimal policy in the tabular setting, they require a number of steps that grow exponentially with the state space (Whitehead, 1991; Kakade et al., 2003; Strehl et al., 2008) and careful scheduling (Ghavamzadeh et al., 2020; Jin

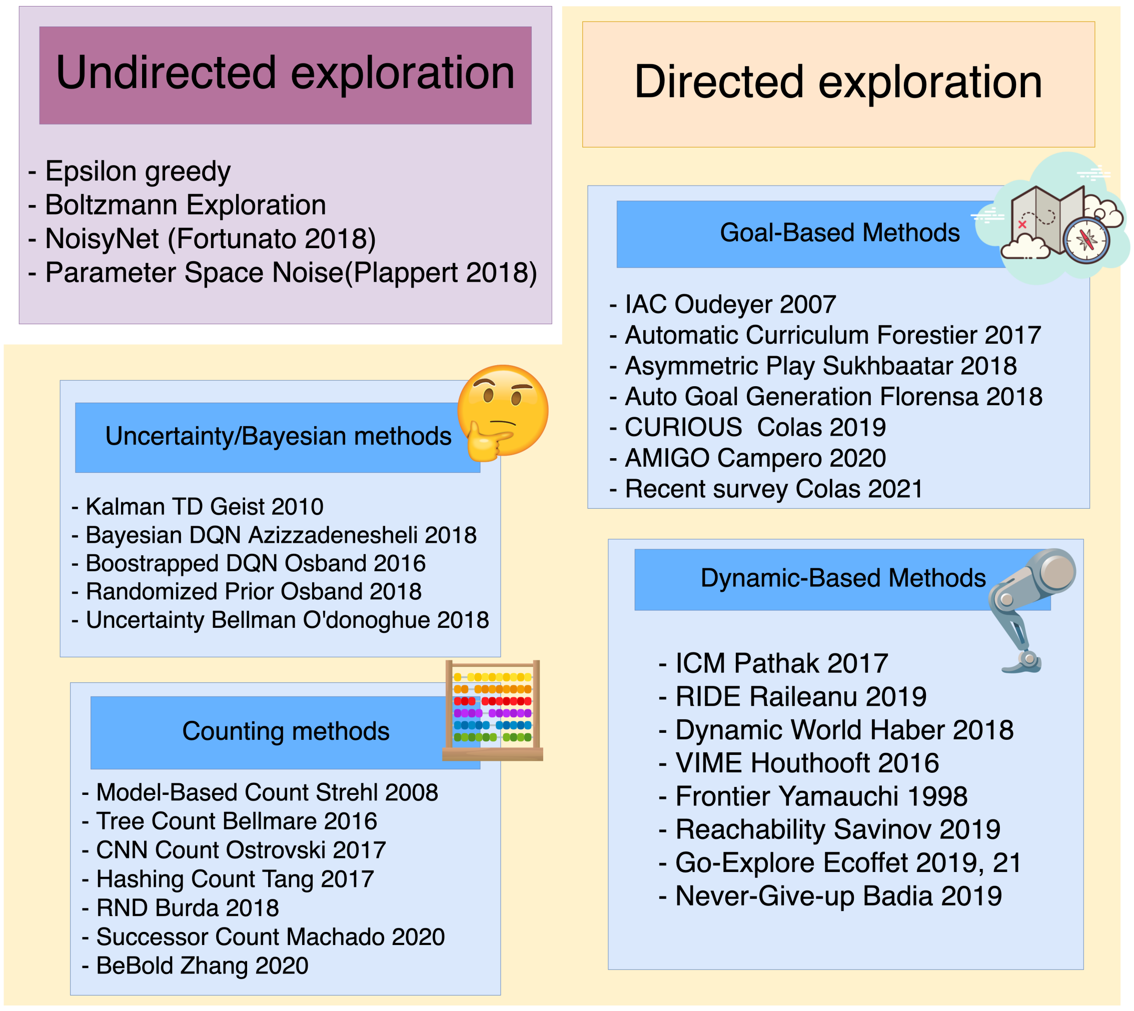

Undirected exploration
- Epsilon greedy
- Boltzmann Exploration
- NoisyNet (Fortunato 2018)
- Parameter Space Noise(Plappert 2018)
Directed exploration
Goal-Based Methods
- IAC Oudeyer 2007
- Automatic Curriculum Forestier 2017
- Asymmetric Play Sukhbaatar 2018
- Auto Goal Generation Florensa 2018
- CURIOUS Colas 2019
- AMIGO Campero 2020
- Recent survey Colas 2021
Uncertainty/Bayesian methods
- Kalman TD Geist 2010
- Bayesian DQN Azizzadenesheli 2018
- Boostrapped DQN Osband 2016
- Randomized Prior Osband 2018
- Uncertainty Bellman O'donoghue 2018
Dynamic-Based Methods
- ICM Pathak 2017
- RIDE Raileanu 2019
- Dynamic World Haber 2018
- VIME Houthooft 2016
- Frontier Yamauchi 1998
- Reachability Savinov 2019
- Go-Explore Ecoffet 2019, 21
- Never-Give-up Badia 2019
Counting methods
- Model-Based Count Strehl 2008
- Tree Count Bellmare 2016
- CNN Count Ostrovski 2017
- Hashing Count Tang 2017
- RND Burda 2018
- Successor Count Machado 2020
- BeBold Zhang 2020

Figure 3.1: Chapter TL;DR : Exploration methods

et al., 2018). Despite this inherent lack of sample efficiency, they remain valuable task-agnostic exploration strategies in large-scale problems with dense rewards (Mnih et al., 2015).

Besides, recent undirected exploration methods have been developed to fit deep neural architectures, such as injecting random noise in the network parameter space (Fortunato et al., 2018; Plappert et al., 2018). Yet, undirected methods still struggle with *sparse reward* signals (Plappert et al., 2018), or any task requiring deep exploration (Osband et al., 2016; Kearns et al., 2002).

The sparse reward problem occurs when rewards are not a dense stream. Since rewards guide policies, its absence leave the agent wandering aimlessly. Thus, pushing the agent to navigate and explore the environment, even in the absence of reward is essential in this setting.

3.1.2 Directed Methods in Exploration

3.1.2.1 Modeling the uncertainty

Bayesian Machine Learning is designed to model *uncertainty* and uses it as an exploration signal. High uncertainty defines points of interest, thus, agents should explore more deeply when unsure about the environment or the policy's behavior.

Bayesian Deep Q-Learning (Azizzadenesheli et al., 2018) directly applies this idea to Q-network by replacing the last layer of linear regression with Bayesian linear regression, using the variance of the posterior as an exploration signal. Osband et al., 2018; Osband et al., 2016 use a similar idea by using an ensemble of Q-functions. Among others, uncertainty has been used to guide exploration towards ill-estimated state-action pairs by relying on the Bellman equation (Geist et al., 2010; O'Donoghue et al., 2018).

Despite being theoretically sound, these methods face scaling difficulties and still struggle with the sparse reward problem.

3.1.2.2 Optimism and Intrinsic Motivation

In the following section, we study another directed exploration approach based on reward bonuses to densify the reward signal. In this setting, the environment reward, namely **extrinsic** rewards, is augmented with an exploration guidance reward signal, namely **intrinsic** rewards (Singh et al., 2004; Simsek et al., 2006). This intrinsic reward spurs exploration by tipping the agent to take a specific course of actions. Furthermore, it makes undirected exploration mechanisms applicable again by spreading milestone rewards during training. Inspired by cognitive science, this intrinsic reward often encodes a degree of "novelty," "surprise," ,"curiosity" (Oudeyer et al., 2007; Berlyne, 1965; Schmidhuber, 1991), "learning progress" (Lopes et al., 2012) or "boredom" (Schmidhuber, 1991; Oudeyer et al., 2008). These common intrinsic motivation mechanisms are broadly categorized into three families: count-based, dynamic-prediction-based, and goal-based methods.

Count-based exploration aims to catalog visited states (or action-states pairs) along episodes to detect unseen states, and drive the agent towards them. It has first been proposed as an exploration heuristic in the early days of RL (Thrun, 1992; Sato et al., 1988;

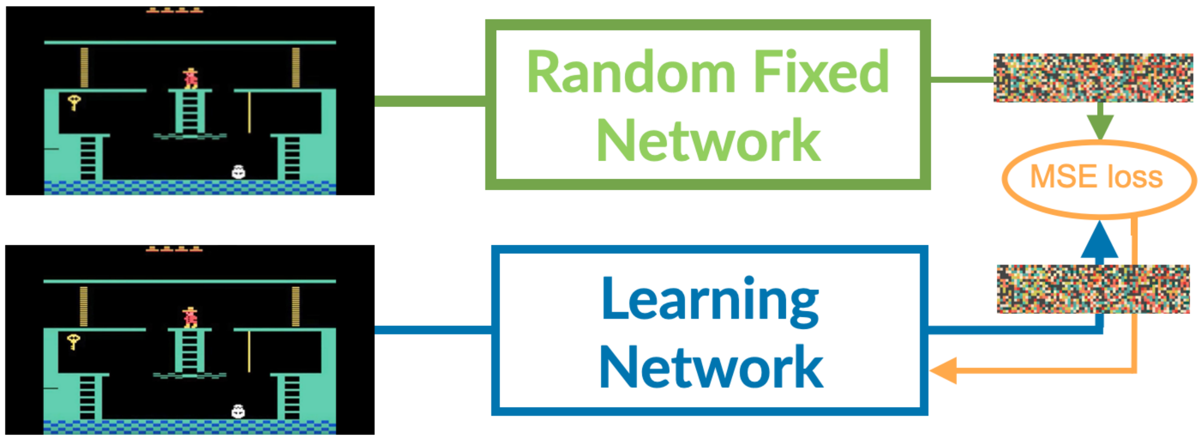


Figure 3.2: Random Network Distillation : A learning network tries to predict the output of a random network with fixed weights. The error is used as an exploration signal, high prediction error indicates novelty. The random network acts as a locally robust hash function (changing one pixel does not completely change the output, thus the learning network should be able to accurately predict states visited many times by the agent.

Barto et al., 1991), before being framed as an intrinsic exploration reward mechanisms in the tabular case (Strehl et al., 2008; Kolter et al., 2009)[1].

Pseudo-counts were then introduced to approximate the state counts (Lopes et al., 2012), where pseudo-counts were estimated through different density models to produce intrinsic rewards. Density models range from contextual trees (Bellemare et al., 2016), generative neural models, e.g. PixelCNN (Ostrovski et al., 2017), or autoencoders combined with a local hashing function (Tang et al., 2017). Differently, Random Network Distillation (**RND**) Burda et al., 2019b uses the prediction error between a randomly initialized network and a trained network as a state-count proxy. The random network acts as a pseudo-count proxy by modeling a locally preserving hashing function. At the same time, the regression error diminishes with the visit count (illustrated in Fig. 3.2). Badia et al., 2020b uses this technique and compose it with multiple intrinsic bonuses by getting both inter and intra-episodic reward mechanisms.

Finally, other methods incorporate back the reward in the pseudo-count state representation using value-state representations (Martin et al., 2017) or the distance between two successor features (Machado et al., 2020).

Yet, count-based methods may explore the immediate surrounding and heavily depend on the state representation quality.

Dynamics prediction exploration aims to encourage the agent to uncover the environment dynamics rather than cataloging states. Such agents learn a world model predicting the consequences of their actions; and they take an interest in challenging and refining it (Haber et al., 2018; Oudeyer et al., 2007; Oudeyer, 2018; Erraqabi et al., 2021). In RL, this intuition is transposed by taking the current state and action to predict the next state representation; the resulting prediction error is then turned into the intrinsic reward signal. Approaches mostly differ in learning the state representation: Stadie et

[1]Count-based were also used as an incertitude metric, which later lead to other exploration strategy based on optimism under uncertainty (Strehl et al., 2008; Auer et al., 2008; Jaksch et al., 2010)

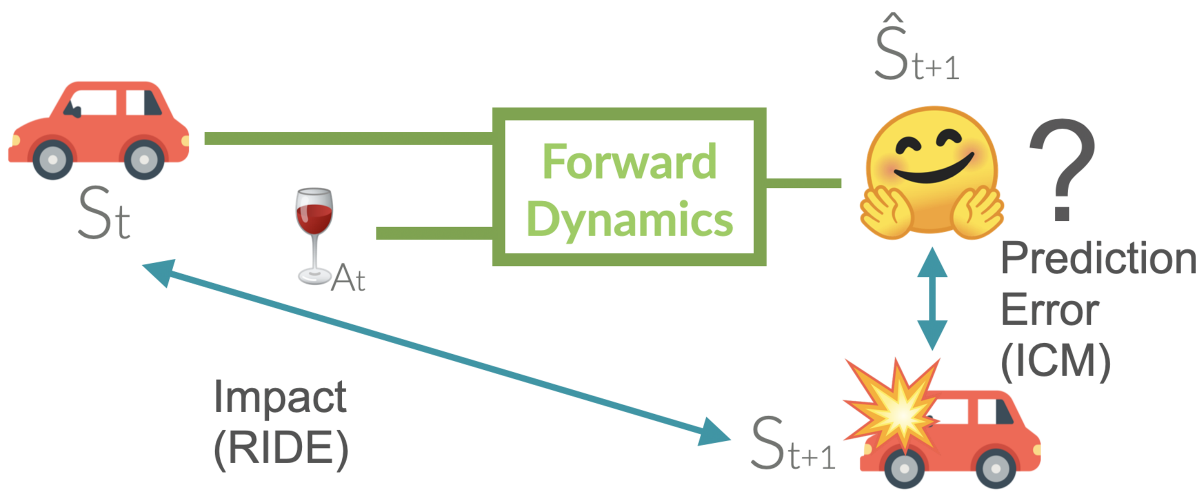


Figure 3.3: Illustration of a dynamic model used to compute intrinsic reward. A **forward** model (predicting s_{t+1} from s_t and a_t) is used to compute the intrinsic reward (In practice, more complicated dynamics can be used, such as using an **inverse** model). Intrinsic Curiosity Module (Pathak et al., 2017) uses prediction error as a signal to explore. RIDE (Raileanu et al., 2020) rewards a great change between $\phi(s_t)$ and $\phi(s_{t+1})$.

al., 2015 compresses raw observations with autoencoders, RND (Burda et al., 2019a) uses random projections, VIME (Houthooft et al., 2016) captures the environment stochasticity by maximizing mutual information with Bayesian Networks. In parallel, Pathak et al., 2017 argues that the state representation should mainly encode features altered by the agent. They thus introduce an inverse model that predicts the action given two consequent states as a training signal. Achiam et al., 2017b; Azar et al., 2019 compute the intrinsic reward across multiple timesteps predictions to better estimate information gain. Other forms of dynamics modeling have been explored with empowerment (Mohamed et al., 2015; Gregor et al., 2016) or auxiliary-task prediction (Kamienny et al., 2020).

Yet, those intrinsic rewards based on prediction errors may attract the agent into irrelevant yet unpredictable transitions. Another drawback is reward evanescence: the intrinsic reward slowly vanishes as the model is getting better. Schmidhuber, 1991; Oudeyer et al., 2007 originally proposed to measure the mean error evolution rather than immediate errors to account for the agent progress. Differently, Raileanu et al., 2020 replace the error prediction by the difference between consecutive representation states, removing the need to compute a vanishing prediction error (Model illustrated in Fig. 3.3).

Finally, environment dynamics can be used to compute a *knowledge frontier*, states that are reachable, but which lead to unseen states (Topiwala et al., 2018; Yamauchi, 1998). Frontier states can be rewarded with a reachability model (Savinov et al., 2019), directly resetting to frontier states (Ecoffet et al., 2019) or storing them in memory (Ecoffet et al., 2021) and finally computing them using count-methods (Zhang et al., 2020).

Goal-based methods provide identifiable and intermediate goals to reward the agent upon completion. Such approaches perform an explicit curriculum by slowly increasing the exploration depth through goal difficulties. They often build on top of the UVFA framework to condition the agent policy (Schaul et al., 2015). Goal-based methods may

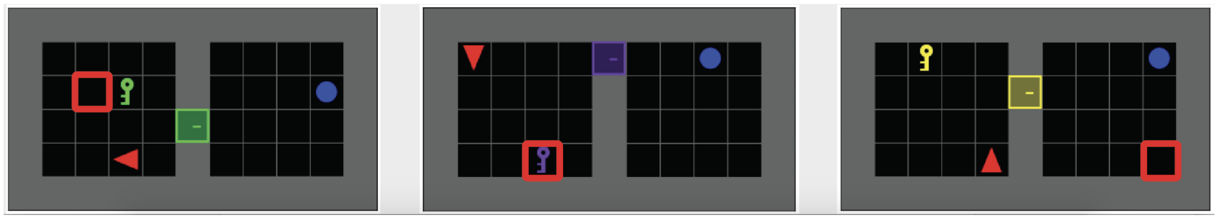


Figure 3.4: An illustration of AMIGO (Campero et al., 2020). The policy model controls the red triangle and must reach the blue ball. A goal-setter proposes a task, illustrated by the red square. Goals should be of increasing complexity to alleviate policy's learning. (Image Source: https://arxiv.org/pdf/2006.12122.pdf)

take several forms ranging from hindsight experience replay (Andrychowicz et al., 2017), curriculum (Nguyen et al., 2021), goal-generation (Forestier et al., 2017; Campero et al., 2020; Sukhbaatar et al., 2018; Florensa et al., 2018; Colas et al., 2019) and hand-crafted goals (Hermann et al., 2017). Yet, they may face unstable training, complex goal definition (Cideron et al., 2020), or require fully observable environments (Campero et al., 2020). For a more detailed taxonomy, see (Colas et al., 2020c)

Intrinsic motivation has also been explored in hierarchical reinforcement learning (Barto et al., 2004; Kulkarni et al., 2016; Duminy et al., 2021), but it goes beyond the scope of this chapter.

It is worth mentioning imitation learning strategies such as (Hussenot et al., 2020a) which retrieve intrinsic motivation signals from human trajectories or methods that selects which demonstrations to choose from (Nguyen et al., 2012; Nguyen et al., 2021).

3.2 Second Contribution (IJCAI'21): Don't Do What Doesn't Matter, Intrinsic Motivation from Action Usefulness

So far, exploration strategies rarely mention action structure or action usage, mainly focusing on state structure. Action are only used to build a transition model.

We therefore aim to shift the emphasis from state novelty distributions towards novel action distributions to develop new intrinsic motivation signals, and consequently, change the exploration behavior. More precisely, we aim at encouraging the agent to visit states that allow rare and relevant actions, i.e. actions that can only be performed in rare occasions.

Imagine that an infant discovers that pushing a button triggers a light; s/he is likely to push everywhere to switch on new lights. By repeating his/her action, the infant may eventually uncover new buttons, and start associating the action *push* to the relevant state features of *buttons*. A similar observation can be made within virtual environments and embodied agents. We expect the agent to first detect rare actions to learn while being nudged towards the states that allow performing such actions.

In this spirit, we propose a new approach we name Don't Do What Doesn't Matter (**DoWhaM**). Instead of uniformly seeking for novel states, **DoWhaM** encourages exploring states allowing actions that are rarely useful; those rarely relevant actions are generally hard to retrieve by random exploration. In other words, the agent is intrinsically rewarded when successfully performing an action that is usually ineffective. We observe that this simple mechanism induces a remarkably different exploration behavior differing from the common state-count and curiosity-based patterns.

Formally, **DoWhaM** keeps track of two quantities for each action: the number of times the action has been used and the number of times the action led to a state change. The resulting intrinsic reward is inversely proportional to the number of times the action has led to a state change. Noticeably, **DoWhaM** primarily keeps count of actions, and can thus be defined as an action count-based method. Besides, tracking actions (as opposed to states) naturally scales in RL: in the discrete case, there is generally less than a few thousand actions, allowing for an exact count. In the continuous case, actions may easily be discretized without using complex density models (Tang et al., 2020b).

We study this approach in the MiniGrid procedurally generated environment (Chevalier-Boisvert et al., 2019). Despite their apparent simplicity, these tasks contain intermediate decisive actions, e.g. picking keys, which have kept in check advanced exploration methods (Raileanu et al., 2020; Campero et al., 2020). We empirically show that **DoWhaM** reduces the sample complexity by a factor of 2 to 10 in a diverse set of environments while resolving the hardest tasks. We then study how **DoWhaM** amends the agent's behavior and compare it to other methods. Finally, we also analyze whether **DoWhaM** may lead to unwanted agent behaviors when facing environments with multiple interactions, which we refer as the *BallPit-problem*.

3.3 Notation Adjustment

In this setting, the reward function is decomposed into an extrinsic reward returned by the environment $r^e(s_t, a_t)$ and a new intrinsic reward $r^i(s_t, a_t, s_{t+1})$. Therefore, the new reward function is defined as : $R(s_t, a_t, s_{t+1}) = r^e(s_t, a_t, s_{t+1}) + \beta r^i(s_t, a_t, s_{t+1})$ where β is an hyperparameter to balance the two return signals. In practice, the extrinsic reward is often a sparse task-specific signal while the intrinsic reward is usually a dense training signal that fosters exploration.

3.4 Don't Do What Doesn't Matter!

3.4.1 Intuition

While most actions consistently move the agent to a new state, some actions do not affect specific states, i.e., the agent remains in the same state after performing it. We hence define an *effective action* if the new state of the environment is different from what it would have been if no action were to be taken. For instance, in tasks involving embodied interaction, such state-action pairs include moving forward while facing a wall or grabbing non-existent objects. Although one may update the MDP only to keep effective actions, such an operation may not always be feasible or desirable in practice. It is thus up to the agent to learn the correct actionable states through exploration. Noticeably, those rare state-actions are often landmarks in the environment dynamics, e.g., triggering buttons or opening doors. One idea is thus to bias the agent to visit states that effectively allow rare actions. **DoWhaM** encapsulates this exploration pattern by (1) detecting rare but effective actions, (2) rewarding the agent when performing these rare actions in context where they are effective. In short, rare and effective actions are the ones that matter.

3.4.2 Method

For every action a_i, the agent tracks two quantities. The number U of times an action has been used during past trajectories, and the number I of times the action impacted, i.e. changed the state $s_t \neq s_{t+1}$[2]. Formally, given the whole history of transitions across episodes $\mathcal{H} = (s_h, a_h, s_{h+1})_{h=0}^{H}$:

$$U^{\mathcal{H}}(a) = \sum_{h=0}^{H} \mathbf{1}_{\{a_h=a\}}, \tag{3.1}$$

$$I^{\mathcal{H}}(a) = \sum_{h=0}^{H} (\mathbf{1}_{\{a_h=a\}} \times \mathbf{1}_{\{s_h \neq s_{h+1}\}}), \tag{3.2}$$

where $\mathbf{1}$ is the indicator function and $\times$ the product operator.

Intuitively, the ratio $I^{\mathcal{H}}(a)/U^{\mathcal{H}}(a)$ indicates how often the action a has been effective along the history $\mathcal{H}$. For instance, actions that move an agent would update the state most of the time, therefore $U(a_i) \approx I(a_i)$. On the other hand, grabbing objects only changes the state in rare occurrences, and $U(a_i) \geq I(a_i)$. We then define the bonus as:

$$B(a_t) = \frac{\xi^{1-\frac{I^{\mathcal{H}}(a_t)}{U^{\mathcal{H}}(a_t)}} - 1}{\xi - 1}, \tag{3.3}$$

where η is a hyperparameter. This function is a continuous approximation of an exponential decay $\exp^{-\xi I^{\mathcal{H}}(a_t)/U^{\mathcal{H}}(a_t)}$. It ranges from 1 when $I^{\mathcal{H}} = 0$ and goes to 0 when $I^{\mathcal{H}} = U^{\mathcal{H}}$. Small ξ leads to a uniform bonus on all actions whereas large values favor rare and efficient actions (see Section 3.4.3 for an illustration).

An intrinsic reward mechanism often requires to discount the intrinsic bonus within an episode. Hence, it prevents the agent from overexploiting, and being stuck in local exploration minima. Inspired by theoretically sound count-based methods (Strehl et al., 2008), we thus divide the previous ratio by an episodic state-count.

Finally, we want to reward actions only in context where they are effective, thus the agent is rewarded only when $st \neq s_{t+1}$, defining the final **DoWhaM** intrinsic reward:

$$r^{i}_{RAM}(s_t, a_t, s_{t+1}) = \begin{cases} \frac{B(a_t)}{\sqrt{N^{\tau}(s_{t+1})}} & \text{if } s_t \neq s_{t+1} \\ 0 & \text{otherwise} \end{cases}, \tag{3.4}$$

where $\tau = (s_t, a_t, r_t)_{t=0}^{l}$ is an episodic trajectory of length l and $N^{\tau}(s) = \sum_{h=0}^{l} \mathbf{1}_{\{s=s_h\}}$ is an episodic state count which is reset at the beginning of each episode. In high-dimensional state space, the episodic state count can be replaced by a pseudo-count (Bellemare et al., 2016) or an episodic novelty mechanism (Badia et al., 2020a).

[2]In noisy or dynamic environment, it is possible to relax or learn this as mentioned in Section 3.6.3.

Action-based Counter As counting methods may sound anachronistic, we emphasize again that actions are ascertainable in RL, i.e. they can be easily enumerated. As opposed to state-counting which requires complex density models (Ostrovski et al., 2017), discrete actions suffer less from the curse of dimensionality, and can easily be binned together in the case of a large action set (Dulac-Arnold et al., 2015). Besides, although **DoWhaM** relies on an episodic state count, a raw approximation is sufficient as it encodes a reward decay.

3.4.3 Decay illustration

The **DoWhaM** reward is a function of the action ratio: $I^{\mathcal{H}}(a)$ the number of times an action impacted the state over $U^{\mathcal{H}}(a)$ the number of usage. The function B Eq. (3.3) acts as an exponential decay starting from 1 when the ratio is 0 (the action never impacted the environment so we want to reward it highly when it is actually modifying the state) and giving 0 reward when the ratio is 1. The action a modifies the state all the time; thus, a is of low interest.

In Fig. 3.5, we illustrate how the parameter ξ shapes the intrinsic reward.

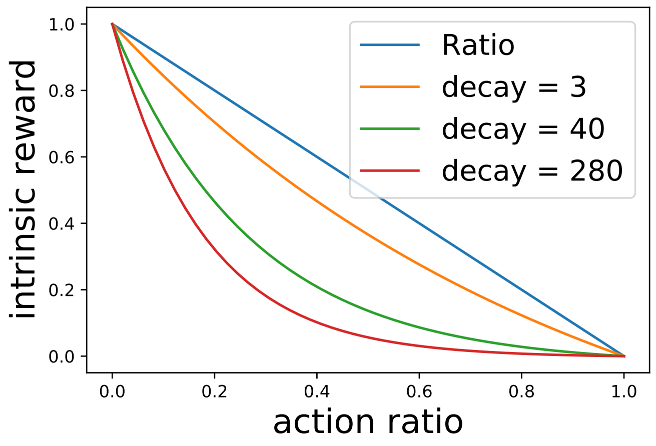


Figure 3.5: Ratio decay function illustrated for different values of ξ

3.5 Experimental settings

We evaluate **DoWhaM** in the procedurally-generated environments MiniGrid (Chevalier-Boisvert et al., 2019). MiniGrid is a partially observable 2D gridworld with a diverse set of tasks. The RL agent needs to collect items and open locked doors before reaching a final destination. Despite its apparent simplicity, several MiniGrid environments require the agent to perform exploration with few specific interactions, and state-of-the-art procedures still struggle to solve them (Raileanu et al., 2020). For each experiment, we report the rolling mean (over 40k timesteps) and standard deviation over 5 seeds.

3.5.1 MiniGrid Environment

Each new MiniGrid world contains a combination of rooms that are populated with objects (balls, boxes or keys), and are linked together through (locked/unlocked) doors. Balls and

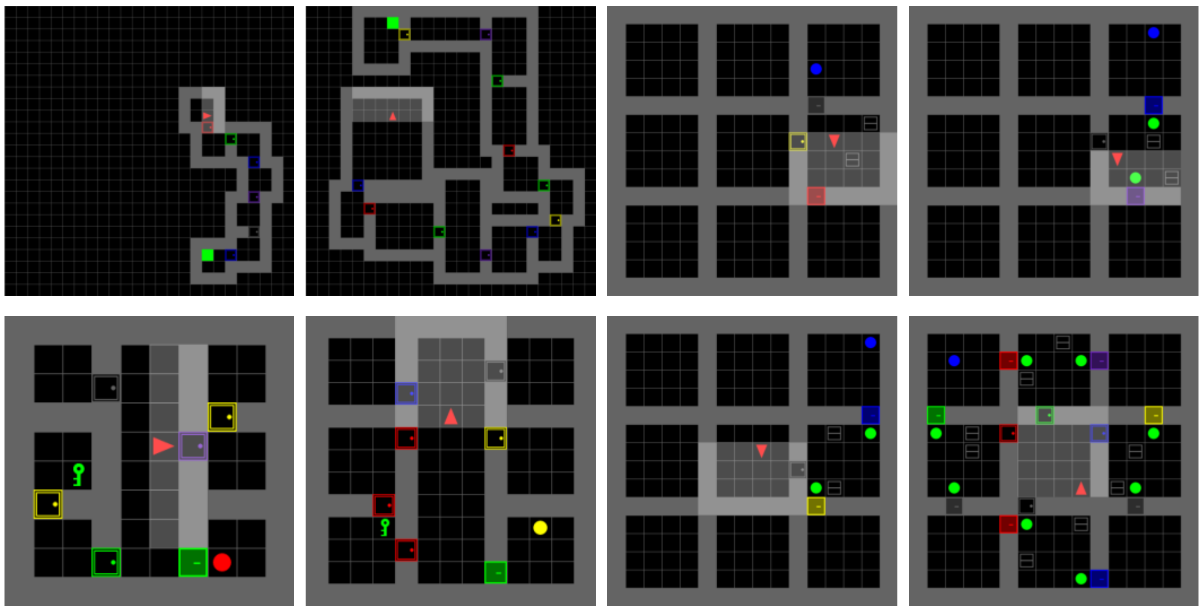


Figure 3.6: Environments used to assess all methods. From top left to down right : MultiRoom (N7S4, N12S10), ObstructedMaze (2Dlh, 2Dlhb), KeyCorridor (S4R3, S5R3), ObstructedMaze(1Q, Full)

keys can be picked up or dropped and the agent may only carry one of them at a time. Boxes can be opened to discover a hidden colored key. Doors can be unlocked with keys matching their color. The agent is rewarded after reaching the goal tile. At each step, the agent observes a 7x7 representation of its field of view and the item it carries if any. The agent may perform one out of seven actions: move forward, turn right, turn left, pick-up object, drop object, toggle. Noticeably, some actions are ineffective in specific states, e.g. moving forward in front of a wall, picking-up/dropping/toggling objects when none is available. Following (Raileanu et al., 2020; Campero et al., 2020), we focus on three hard exploration tasks, which are illustrated in Fig. 3.7.

MultiRoom(N-S): The agent must navigate through a sequence of empty rooms connected by doors of different colors. A map contains N rooms, whose indoor width and height are sampled within 2 and $S-2$ tiles. MultiRoom maps entail limited interaction as the agent only has to toggle doors and no object manipulation is required. Yet, this bare-bone environment constitutes a good preliminary trial.

KeyCorridor(S-R): The agent must explore multiple adjacent unlocked rooms to retrieve a key, open the remaining locked room, and collect the green ball. A map contains a large main corridor connected to $2 \times R$ square rooms of fixed indoor dimension $S-2$. Solving a KeyCorridor map requires the agent to perform a specific sequence of interactions, which makes the task more difficult than MultiRoom.

ObstructedMaze: The agent must explore a grid of rooms that are randomly connected to each others in order to collect a blue ball. Some of the doors are locked and the agent has to either directly collect the keys or toggle boxes to reveal them. Besides, distractor balls are added to block door access. ObstructedMaze can quickly become a hard maze with false leads and complex interactions.

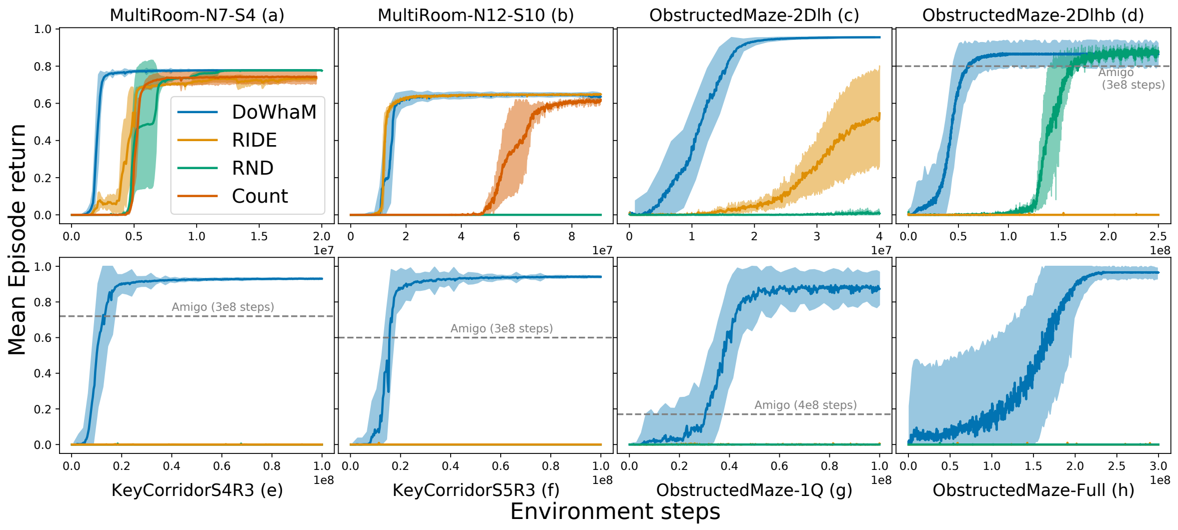


Figure 3.7: Comparison between intrinsically motivated methods on multiple MiniGrid tasks.

3.5.2 Experimental Setting

3.5.2.1 Training

We follow the training protocol defined in (Raileanu et al., 2020; Campero et al., 2020). We use 3 convolution layers with a kernel size of 3, followed by 2 fully-connected layers of size 1024, and an LSTM of hidden size 1024. Finally, we use two separate fully-connected layers of size 1024 for the actor's and critic's head. We train our model with the distributed actor-critic algorithm IMPALA (Espeholt et al., 2018) TorchBeast implementation (Küttler et al., 2019).

3.5.2.2 Baselines

We here cover three families of intrinsically motivated reward mechanisms: counting method (COUNT, RND), dynamic-based (RIDE) and goal-based (AMIGO).

COUNT (Strehl et al., 2008) is a counting method that baits the agent to explore less visited states. In this setting, we use a tabular-count to catalog the state-action pairs.

RND (Burda et al., 2019b) acts as a states' pseudo-count method. The states are first projected with a random network, while a second network is trained to predict this state representation. The normalized prediction error is then used as an intrinsic reward (Illustrated in Fig. 3.2).

RIDE (Raileanu et al., 2020) is a dynamic-based model that builds upon (Pathak et al., 2017). It computes the difference between two consecutive states, encouraging the agent to perform actions that lead it to a maximally different states (illustrated in Fig. 3.3).

AMIGO (Campero et al., 2020) is a hierarchical goal-based method, splitting the agent into two components: an adversarial goal-setter and a goal-condition learner that adversarially creates goals (illustrated in Fig. 3.4).

3.6 Experimental Results

3.6.1 Base environment

Fig. 3.7 displays the results on 8 MiniGrid tasks. Noticeably, **DoWhaM** outperforms all the baselines in sample complexity, and even solves among the most complex worlds. In MultiRoom, we observe that **DoWhaM** outperforms RIDE, RND, and COUNT in the simple setup (N7S4), and matches RIDE's sample complexity performance on the challenging setup (N12S10). Note that **DoWhaM** does not seem to be penalized by the small amount of possible interactions. In KeyCorridor and ObstructedMaze, RIDE, RND, and AMIGO learn in the easiest instances but they struggle as the difficulty, i.e. exploration depth, increases as first observed in (Campero et al., 2020). On the other hand, **DoWhaM** consistently solves all the environments, even the challenging ObstructedMaze-Full.

We derive two hypotheses from those results: (1) State-count rewards exhaustively explore the state space, reducing the overall exploration coverage (2) Curiosity-based rewards do not emphasize enough salient interactions and then explore new but irrelevant state-action pairs. Although such approaches were successful in many environments, those exploration behaviors may fail as soon as specific interactions must be regularly performed in the exploration process. In the following, we thus try to assess those hypotheses.

3.6.2 Intrinsic exploration behavior

We first conduct a series of experiments without external reward to study what *type of exploration* each bonus creates. In other words, what are the inductive exploration biases that arise from the different intrinsic reward mechanisms. To do so, we rely on two metrics: the state visit (plotted as heatmaps) and the action distribution (plotted as bar plots).

3.6.2.1 Rewardless Playground

Playground is designed to visually assess how exploration strategies behave when lots of object are present. Playground is a 14x14 grid, objects are always spawned at the same location but the color changes from episode to episode. An episode lasts 200 steps, no external reward is given and the agent is always spawned in the center, facing a random direction. This sandbox environment has no specific goal, akin to a kindergarten. This environment contains multiple keys, balls, and boxes located in the corners and spawns the agent facing a random direction. Fig. 3.8 shows the agent state visits for during 10^6 training timesteps when only using the intrinsic reward signal.

We observe that RND and **DoWhaM** are both attracted by the objects and explore the space thoroughly, whereas RIDE and COUNT remain close to the center and seldomly reach the objects. This observation backs our results in ObstructedMaze2Dlh, where RND and **DoWhaM** are the only methods exploring thoughtfully the environment. It also confirms our hypothesis that standard state-based approaches, e.g., COUNT, may not be pushed enough to perform in-depth exploration. Surprisingly, the curiosity-based method RIDE has not been strongly incentivized to interact with remote objects, suggesting that it may suffer from its dependency on the state representation. However, these experiments

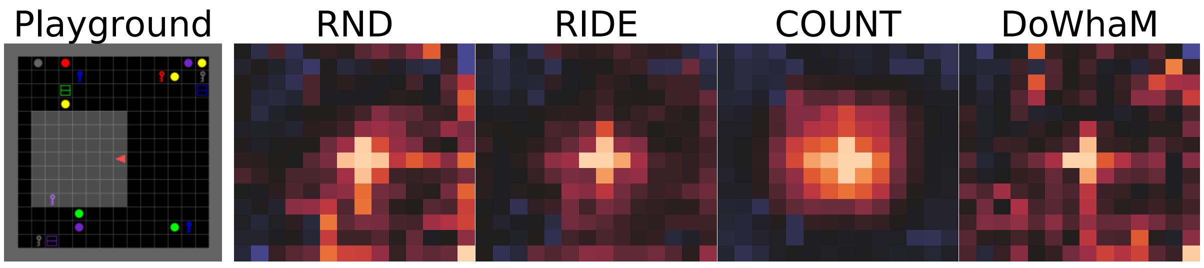


Figure 3.8: States visitation in Playground environment. Bright orange means more visits, **darker and blue** means **less** visits

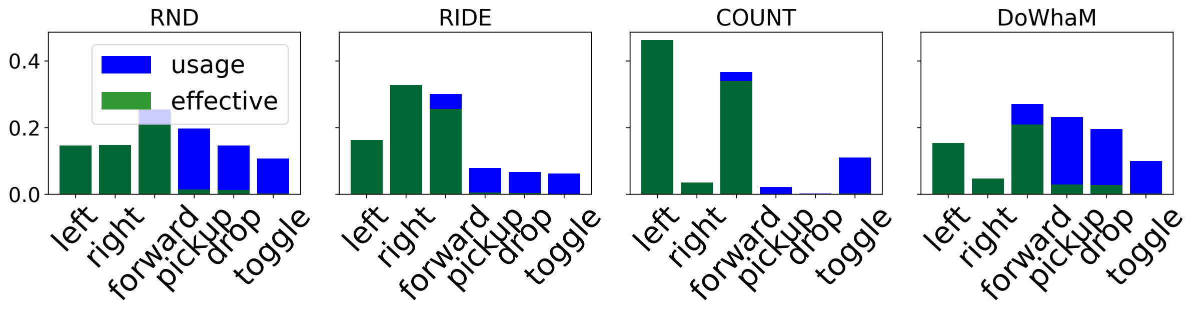


Figure 3.9: Playground Action Distribution for all baselines. The same pattern can be observed, baseline algorithms scarcely use *pickup, drop and toggle* and when they do, it does not impact the environment. **DoWhaM** on the other hand discovers how to use pickup and drop.

do not explain the performance difference between RND and **DoWhaM** on the most challenging setups. Thus, looking at the action distribution is necessary.

Qualitative Analysis After training the agents in Playground, we wanted to assess visually the difference between RIDE and **DoWhaM**. In Fig. 3.10 are displayed two scenarios where we force paths and only inspect the intrinsic bonus given. We see that **DoWhaM** rewards are sparser and pushes the agent to interact with environment's key elements where as RIDE's reward are harder to dissect and less relevant to the objects.

Rewardless KeyCorridorS4R3 We then study the behavior that is solely intrinsically motivated in the KeyCorridor environment to better grasp the **DoWhaM** performance in this setting. Similarly, we trained the agents on KeyCorridorS4R3 for 10^7 timesteps with only the intrinsic reward signal, and results are displayed in Fig. 3.11.

All the baselines – RIDE, RND, and COUNT – remain mostly stuck in the central corridor, where **DoWhaM** explores rooms more uniformly. More impressively, the **DoWhaM** agent naturally picks the key, enters the locked room, and grabs the ball 7% of the times without any extrinsic reward. COUNT, RIDE, and RND all have a success ratio below 0.6%, which may explain why **DoWhaM** manages to solve this task. Further details can be found in Fig. 3.10 and Table 3.1.

We also observe a large discrepancy in the action distribution between the different methods. First, we observe that RND and **DoWhaM** action distributions remain approximately uniform while RIDE and COUNT favor moving actions, reducing the opportunity for interactions. Second, and crucially, the impact distributions $E^{\mathcal{H}}(a)$ differs drastically

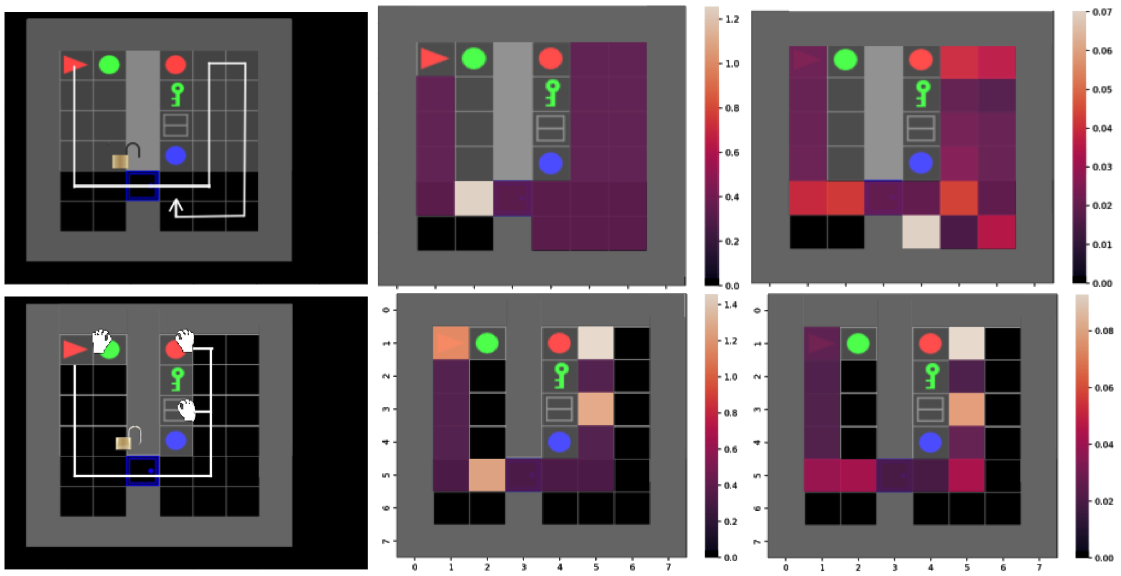


Figure 3.10: Forced Scenario. We enforce a trajectory and observe how the different intrinsic bonuses are given.
(Left) Forced path : Interaction scenario (Middle) **DoWhaM** (Right) RIDE

Algorithm	**RND**	**COUNT**	**RIDE**	**DoWhaM**
% of extrinsic reward collection	0%	0.4%	0.6%	**7%**

Table 3.1: Rewardless KeyCorridor : Percentage of times the agent collects extrinsic rewards while only receiving intrinsic rewards (average over 420 episodes)

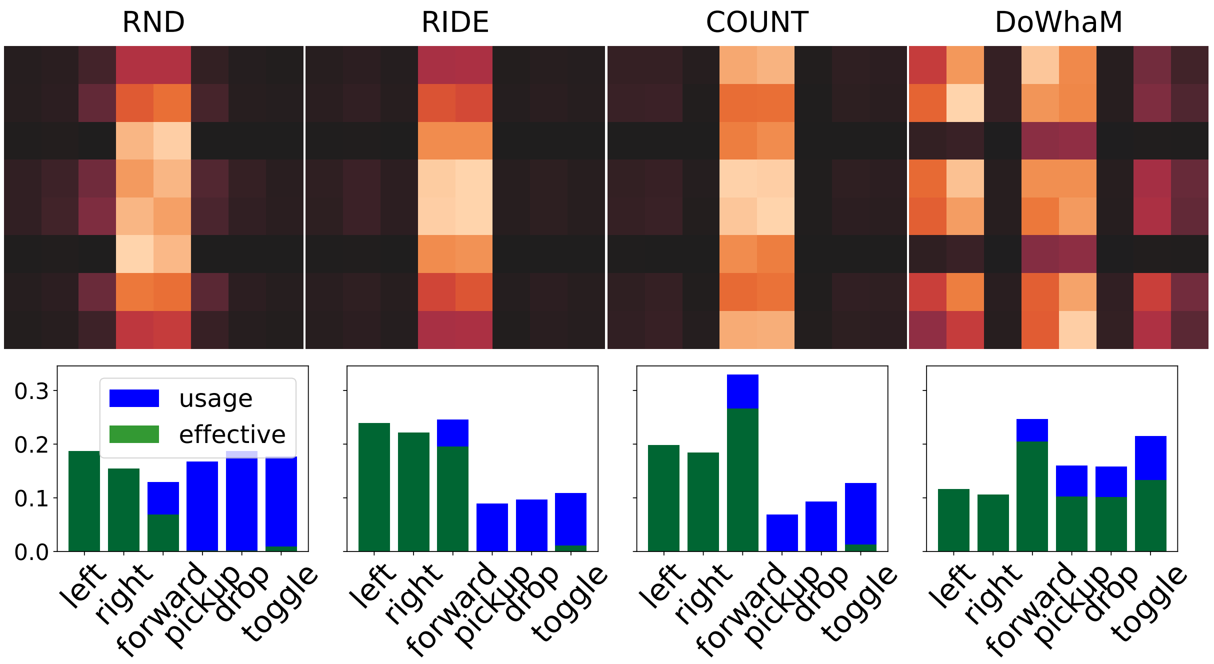


Figure 3.11: State and action distributions in **rewardless** KeyCorridor (S4R3). $U^{\mathcal{H}}(a)$ and $I^{\mathcal{H}}(a)$ action-count are in blue and green. Only **DoWhaM** correctly uses pickup/-drop/toggle during exploration.

between **DoWhaM** and other methods. All agents are trying actions such as *pick, toggle* or *drop*, but those actions are rarely changing the agent's state. These actions are not used in the appropriate context, i.e., in front of an object. It means that rewarding state novelty might not be enough to discover effective actions, thus wasting samples. Although **DoWhaM** and RND had similar state-visitation and action distribution patterns, only **DoWhaM** correctly apprehend rarely effective actions, and correctly use them to explore its environment.

3.6.3 Intrinsic Motivation Pitfalls

The Ball Pit Problem As **DoWhaM** biases the state visit distribution towards performing rare actions, it may introduce a poor exploration pattern when facing too many of such states. We refer to this potential issue as the *Ballpit problem*: the agent remains in rooms with plenty of balls to interact with. We build it on top of MultiRoomN4S6, a simple environment solved by all baselines. **No Ball** is the baseline environment. In **Small** instances, 1 random object is spawned, **More** contains 3 random objects and **Max** rooms are completely filled without blocking the agent path. We emphasize that the agent does not need to interact with anything (with the exception of door) to solve the environment and objects are never in the way of the agent (illustrated on Fig. 3.12).

As the number of objects grows, the performance of all algorithms deteriorates. RND, COUNT are mostly affected by this problem, as the number of states is growing exponentially; thus, counting state occurrence is challenging. RIDE is less affected by the BallPit problem, but most surprisingly, **DoWhaM** is the only one to reach the exit consistently in the most challenging setup. The $I^{\mathcal{H}}(a)/U^{\mathcal{H}}(a)$ ratio correctly balances the exploration bonus, and does not take over the final extrinsic reward (See Fig. 3.12).

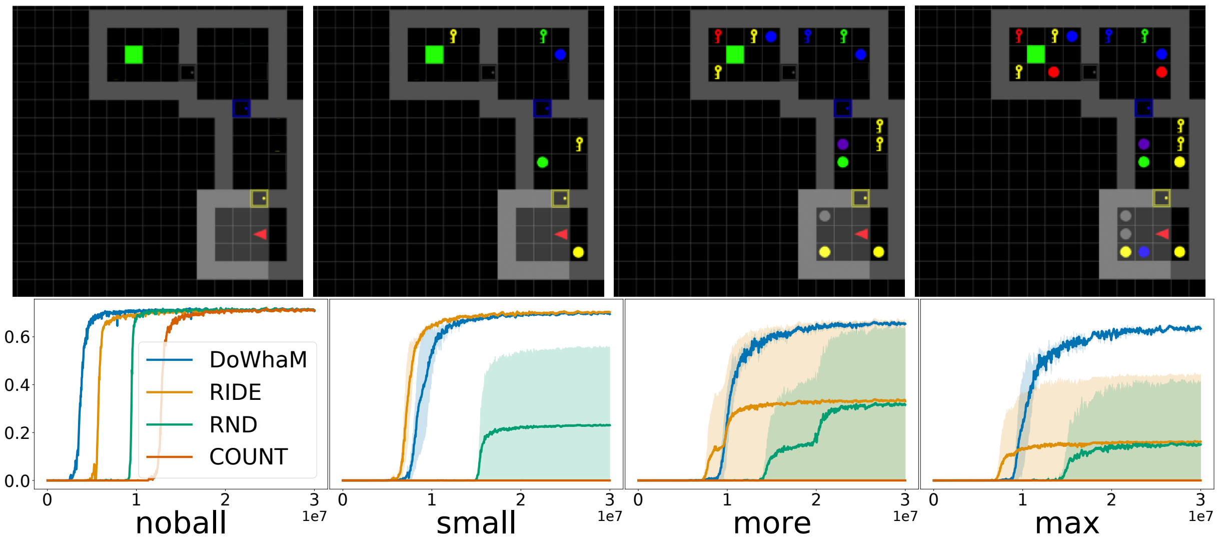


Figure 3.12: As distractors are added (from left to right), we observe a drop in performance for all methods.

The Noisy-TV problem State-count based agent are attracted to state-action pairs with random noise. In its current definition, **DoWhaM** is also affected while computing $I^{\mathcal{H}}$. Similar to (Burda et al., 2019b), this effect can be circumvented by using an inverse model, and we leave it for future work.

ColorMaze In Fig. 3.13, we design a map with a sequence of open rooms, colored floor changing every episode, two boxes with one hidden key, and a locked door leading to the reward.

ColorMaze is a variation of ObstructedMaze2Dlh. Before accessing the room containing keys within boxes, the agent must cross 4 colored rooms. A distractor room is placed at the beginning, leading to nothing. The colors are picked randomly at the beginning of an episode and remains the same until the episode's end. The agent must catch the blue ball within a 576 steps limit to receive the reward. One main difference between MultiRoom and ColorMaze is the absence of doors between colored room.

All baselines remain in the first part of the maze while **DoWhaM** quickly reaches the objects and solves the task. This experiment highlights again how shifting the emphasis from exhaustive state-visit to relevant state-visit can be beneficial, and change the exploration pattern.

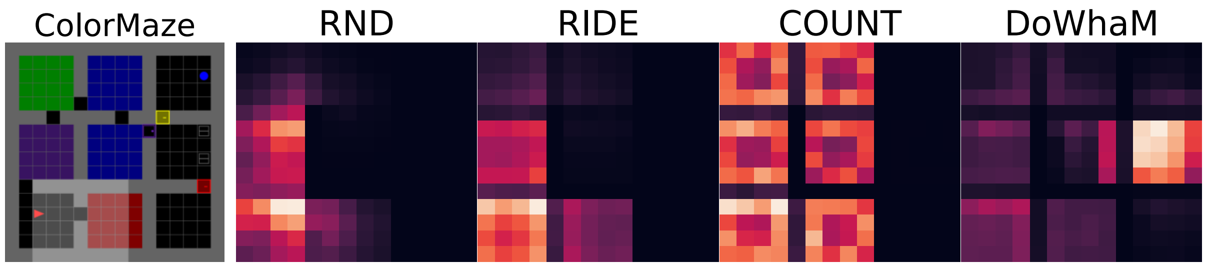


Figure 3.13: RND, RIDE and COUNT remain within the colored region whereas **DoWhaM** learns to go straight to the boxes and keys.

3.7 Conclusion

We introduced Don't Do What Doesn't Matter (**DoWhaM**), a new action-based intrinsic exploration algorithm. As opposed to count-based and curiosity-driven methods, **DoWhaM** shifts the emphasis from novel state to state with relevant actions, rewarding actions that are rarely effective in the environment. Combined with a simple episodic count, **DoWhaM** outperforms recent exploration methods on a variety of hard exploratory tasks in a Minigrid environment. This proof of concept illustrates that action-based exploration is a promising approach as it induces surprisingly different exploration patterns. We also pointed out a new category of problems called *BallPit*, which deteriorate performance of many intrinsically motivated reward approaches.

Chapter 4

Abstraction and Goals

4.1 Introduction

The first part of this thesis dealt with the lowest level of interaction, the action space. In this chapter, we want to go HIGhER (Cideron et al., 2020) by working with more general tasks and goals. When tying up actions together, we create more meaningful policies and focus on dunes instead of looking at individual grains of sand. Going beyond step by step planning, an agent could reason using higher-level objectives. Thus, working with *hierarchy* gives the ability to reason over longer horizons by letting go of low-level details. Once an agent mastered some skills, it opens the door to transfer and new recombination. Then, nothing prevents further aggregation, composing dunes together to form a desert. However, from low-level interaction to concepts and objects lie thousands of years of evolution. From grasping to forging and assembling complex objects, many simple steps must be understood and mastered. Thus, building a self-organizing hierarchy in reinforcement learning (Fikes et al., 1972; Dayan et al., 1993; Sutton et al., 1999b; Barto et al., 2003; Flet-Berliac, 2019) is a daunting task (an oasis, if we want to keep going with this analogy).

Fortunately, as humans, we developed multiple tools to avoid learning from scratch over and over; one of them is natural language. Language has slowly evolved to communicate intents, state objectives, describe complex situations (Kirby et al., 2015). It conveys information compactly by relying on composition and highlighting salient facts. As language can express a vast diversity of goals and situations, it may help to condition the training of interactive agents over heterogeneous and composite tasks (Luketina et al., 2019) and help transfer (Narasimhan et al., 2018). Language is also a reasonning tool, its structure and its alignment with the world create a rich abstraction (Vygotsky, 1934; Colas et al., 2021). Thus, instead of presenting the Hierarchical Reinforcement Learning setting exhaustively, we will focus on the subfield called Language-Conditioned Reinforcement Learning (Schaul et al., 2015; Colas et al., 2020c), where goals are specified by natural language instructions.

In this setting, the agent is given a text description of its goal (e.g. "pick the red ball") and is rewarded when achieving it (Tellex et al., 2011; Chen et al., 2011; Artzi et al., 2013; Luketina et al., 2019; Hermann et al., 2020). The agent has thus to ground the language, i.e., linking and disentangling visual attributes (**shape**, **color**) from language description ("ball", "red") by using rewards to condition its policy toward task completion. The language compositionality allows for a high number of goals and offers generalization opportunities. Unfortunately, conditioning a policy on language also entails a supplementary difficulty as the agent needs to understand linguistic cues to alter its behavior. The agent thus needs to ground its language understanding by relating the words to its observations, actions, and rewards before being able to leverage the language structure (Kiela et al., 2016; Hermann et al., 2017). Once the linguistic symbols are grounded, the agent may then take advantage of language compositionality to condition its policy on new goals (Colas et al., 2020b).

Instruction following have recently drawn much attention following the emergence of several 2D and 3D environments (Chevalier-Boisvert et al., 2019; Brodeur et al., 2017; Anderson et al., 2018). This section first provides an overview of the different approaches, i.e., fully-supervised agent, before exploring approaches focusing on reinforcement learning such as reward shaping, auxiliary losses, and hindsight approaches.

4.1.1 Vision and Language Navigation

Learning to follow natural language instruction is sometimes coined as *Vision and Language Navigation* tasks in computer vision (Anderson et al., 2018; Wang et al., 2019). Most strategies are based on imitation learning, relying on expert demonstrations and knowledge from the environment. For example, Zang et al., 2018 relates instructions to an environment graph, requiring both demonstrations and high-level navigation information. Misra et al., 2018 learns to map a sequence of instructions to landmarks for the low-level controller to follow. Fried et al., 2018 also learns a navigation model and an instruction generator, but the latter is used to generate additional training data for the agent. The setup is hence fully supervised, and requires human demonstrations. These policies are sometimes finetuned to improve navigation abilities in unknown environments. Noticeably, Wang et al., 2019 optimizes an agent to find the shortest path by leveraging language information. The agent learns an instruction generator, and they derive an intrinsic reward by aligning the generator predictions over the ground truth instructions. Those approaches complete long sequences of instructions in visually rich environments but they require a substantial amount of annotated data. In Section 4.3, we intend to discard human supervision to explore learning synergies.

4.2 Goal-Conditionned Reinforcement Learning

4.2.1 Background And Notation

To tackle instruction following using **RL**, few adjustements are required to condition the policy on the goal. We augment our environment with a goal space $\mathbb{G}$ which defines a new reward function $r : \mathbb{S} \times \mathbb{A} \times \mathbb{G} \rightarrow \mathbb{R}$ and policy $\pi : \mathbb{S} \times \mathbb{G} \rightarrow \mathbb{A}$ by conditioning them on a goal descriptor $g \in \mathbb{G}$. Similarly, the Q-function is also conditioned on the goal, and it is referred to as Universal Value Function Approximator (UVFA) Schaul et al., 2015. This approach allows learning holistic policies that generalize over goals in addition to states at the expense of complexifying the training process. We emphasize that agents do not need to balance between different objectives (as opposed to multi-objective-**RL** Moffaert et al., 2014).

Using this formalism does not constrain how the goal is formulated. They can take various form : state to reach (Andrychowicz et al., 2017), images (Sahni et al., 2019), logic abstraction formulas (Li et al., 2018), binary code, and natural language instruction (Luketina et al., 2019).

4.2.2 Hindsight Experience Replay

HER (Andrychowicz et al., 2017) is designed to increase the sample efficiency of off-policy RL algorithms such as DQN in the goal-conditioning setting. It reduces the sparse reward problem by taking advantage of failed trajectories, relabelling them with new goals. An expert then assigns the goal that was achieved by the agent when performing its trajectory, before updating the agent memory replay buffer with an additional positive trajectory.

Formally, HER assumes the existence of a predicate $v : \mathbb{S} \times \mathbb{G} \to \{0, 1\}$ which encodes whether the agent in a state s satisfies the goal $v(s, g) = 1$, and defines the reward function $r(s_t, a, g) = v(s_{t+1}, g)$. At the beginning of an episode, a goal g is drawn from the space $\mathbb{G}$ of goals. At each time step t, the transition $(s_t, a_t, r_t, s_{t+1}, g)$ is stored in the DQN replay buffer, and at the end of an unsuccessful episode, an expert provides an additional goal g' that matches the trajectory. New transitions $(s_t, a_t, r'_t, s_{t+1}, g')$ are thus added to the replay buffer for each time step t, where $r' = r(s_t, a_t, s_{t+1}, g')$. The DQN update rule remains identical to Mnih et al., 2015, transitions are sampled from the replay buffer, and the network is updated using one step td-error minimization.

4.2.3 HER variants

HER has been extended to multiple settings since the original paper. These extensions deal with automatic curriculum learning (Liu et al., 2019a), dynamic goals (Fang et al., 2019), or they adapt goal relabelling to policy gradient methods (Rauber et al., 2019). Closer to our work, Sahni et al., 2019 trains a generative adversarial network to hallucinate visual near-goals state over failed trajectories. However, their method requires heavy engineering as visual goals are extremely complex to generate, and they lack the compact generalization opportunities inherent to language. Chan et al., 2018 also studies HER in the language setting, but the authors only consider the context where a language expert is available.

HER assumes that a mapping m between states s and goals g is given. In the original paper, this requirement is not restrictive as the goal space is a subset of the state space. Thus, the mapping m is straightforward since any state along the trajectory can be used as a substitution goal. In the general case, the goal space differs from the state space, and the mapping function is generally unknown. In the instruction following setting, there is no obvious mapping from visual states to linguistic instructions. It thus requires expert intervention to provide a new language goal given the trajectory, which drastically reduces the interest of HER. Therefore, we here explore how to learn this mapping without any form of expert knowledge nor supervision.

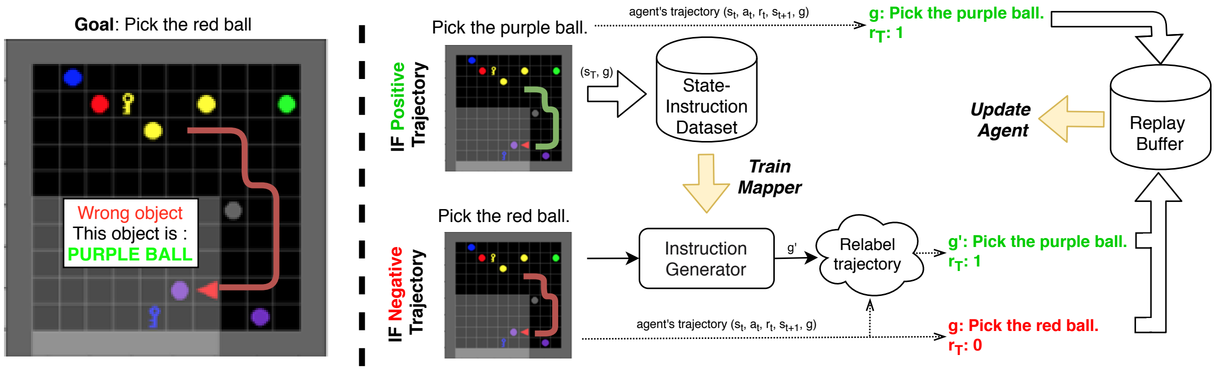


Figure 4.1: Upon positive trajectory, the agent trajectory is added to the RL replay buffer and the goal mapper dataset. Upon failed trajectory, the goal mapper is used to relabel the episode, and both trajectories are appended to the replay buffer. In the original HER paper, the mapping function is bypassed since they are dealing with spatial goals, and therefore, vanilla HER cannot be applied without external expert.

4.3 Third Contribution (SCCI'20): HIGhER : Improving instruction following with Hindsight Generation for Experience Replay

In the following sections, we take advantage of language compositionality to compensate for the lack of reward signals. To do so, we extend Hindsight Experience Replay (HER) to language goals (Andrychowicz et al., 2017). HER originally deals with the sparse reward problems in spatial scenario; it relabels unsuccessful trajectories into successful ones by redefining the policy goal **a posteriori**. As a result, HER creates additional episodes with positive rewards and a more diverse set of goals. Unfortunately, this approach cannot be directly applied when dealing with linguistic goals. As HER requires a mapping between the agent trajectory and the goal to substitute, it requires expert supervision to describe failed episodes with words. Hence, this mapping should either be handcrafted with synthetic bots (Chan et al., 2018), or be learned from human demonstrations, which would both limit HER generality. More generally, language adds a level of semantics, which allows generating textual objective that could not be encoded by simple spatial observations as in regular HER, e.g., "fetch a ball that is not blue" or "pick any red object".

In this work, we introduce Hindsight Generation for Experience Replay (HIGhER), a training procedure where the agent jointly learns the language-goal mapping and the navigation policy by solely interacting with the environment illustrated in Fig. 4.1. HIGhER leverages positive trajectories to learn a mapping function, and tackles the sparse reward problem by relabeling language goals upon negative trajectories in a HER fashion. We evaluate our method on the BabyAI world (Chevalier-Boisvert et al., 2019), showing a clear improvement over RL baselines while highlighting the robustness of HIGhER to noise.

4.4 Hindsight Generation for Experience Replay

Hindsight Generation for Experience Replay (HIGhER) aims to learn a mapping from past experiences that relates a trajectory to a goal in order to apply HER, even when no expert is available. The mapping function relabels unsuccessful trajectories by predicting a substitute goal $\hat{g}$ as an expert would do. The transitions are then appended to the replay buffer. This mapping learning is performed alongside agent policy training.

Besides, we wish to discard any form of expert supervision to learn this mapping as it would reduce the practicability of the approach. Therefore, the core idea is to use environment signals to retrieve training mapping pairs. Instinctively, in the sparse reward setting, trajectories with positive rewards encode ground-truth mapping pairs, while trajectories with negative rewards are mismatched pairs. These cues are thus collected to train the mapping function for HIGhER in a supervised fashion. We emphasize that such signals are inherent to the environment, and an external expert does not provide them. In the following, we only keep positive pairs in order to train a discriminative mapping model.

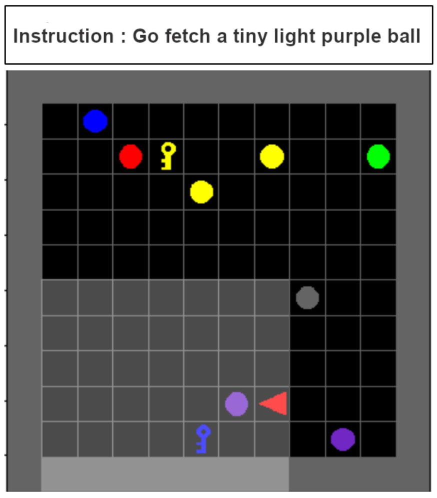
Formally, HIGhER is composed of a dataset $\mathbb{D}$ of $\langle s, g \rangle$ pairs, a replay buffer $\mathbb{B}$ and a parametrized mapping model m_{θ_m}. For each episode, a goal g is picked, and the agent generates transitions $(s_t, a_t, r_t, s_{t+1}, g)$ that are appended to the replay buffer $\mathbb{B}$. The Q-function parameters are updated with an off-policy algorithm by sampling minibatches from $\mathbb{D}$. Upon episode termination, if the goal is achieved, i.e. $f(s_T, g) = 1$, the $\langle s_T, g \rangle$ pair is appended to the dataset $\mathbb{D}$. If the goal is not achieved, a substitute goal is sampled from the mapping model[1] $m_{\theta_m}(s_T) = \hat{g}'$ and the additional transitions $\{(s_t, a_t, r_t, s_{t+1}, \hat{g}')\}_{t=0}^{T}$ are added to the replay buffer. At regular intervals, the mapping model m_{θ_m} is optimized to predict the goal g given the trajectory τ by sampling mini-batches from $\mathbb{D}$. Noticeably, HIGhER can be extended to partially observable environments by replacing the predicate function $v(s, g)$ by $v(\tau, g)$, i.e., the completion of a goal depends on the full trajectory rather than one state. Although we assess HIGhER in the instruction following setting, the proposed procedure can be extended to any other goal modalities.

4.5 Related Methods

4.5.1 Conditioned Language Policy

There have been other attempts to leverage language instruction to improve the agent policy. For instance, (Jiang et al., 2019) computes a high-level language policy to give textual instruction to a low-level policy, enforcing a hierarchical learning training. The authors manage to resolve complicated manipulating task by decomposing the action with language operation. The language mapper performs instruction retrieval into a predefined set of textual goals and yet, the low-level policy benefits from language compositionality and is able to generalize to unseen instructions, as mentioned by the authors. Co-Reyes et al., 2019 trains an agent to refine its policy by collecting language corrections over multiple trajectories on the same task. While the authors focus their effort on integrating language cues, it could be promising to learn the correction function in a HIGhER fashion.

[1]The mapping model can be utilized with an accuracy criterion over a validation set to avoid random goal sampling.



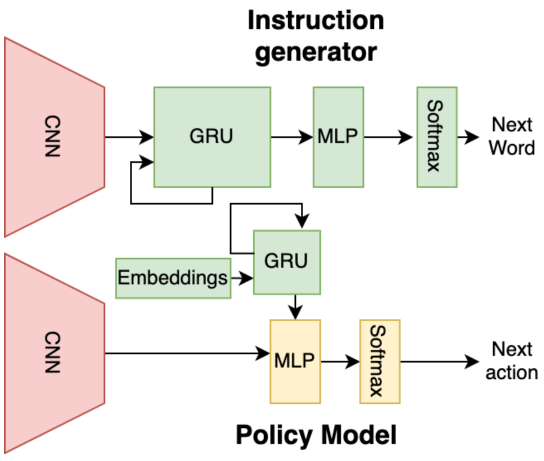


Figure 4.2: **Left:** Illustration of the experimental setup, the agent receives state, encoding its encoded field of view as input and an instruction. Each object has four attributes (only two are visible on the rendering, but the agent can see all four attributes)
Right: Illustration of the architecture used. *Top:* Instruction Generator Model, composed of three layers of convolutionnal neural network, followed by a gated recurrent unit and a two-layer fully connected with dropout using a softmax. *Bottom:* Policy model, composed of three layers of convolutionnal layers to treat the state and a word embedding followed by a gated recurrent unit to encode the instruction. Resulting hidden states are concatenated and fed to a two-layer perceptron followed by a softmax.

4.5.2 IRL for instruction following

Bahdanau et al., 2019a learns a mapping from <instruction, state> to a reward function. The method's aim is to substitute the environment's reward function when instructions can be satisfied by a great diversity of states, making hand-designing reward function tedious. Similarly, Fu et al., 2019 directly learns a reward function and assess its transferability to new environments. Those methods are complementary to ours as they seek to transfer reward function to new environment and we are interested in reducing sample complexity.

4.6 Experiments

4.6.1 Experimental Setting

Environment We experiment our approach on a visual domain called Minigrid (Chevalier-Boisvert et al., 2019). This environment offers a variety of instruction-following tasks using a synthetic language for grounded language learning. We use a 10x10 grid with 10 objects randomly located in the room. Each object has 4 attributes (shade, size, color, and type) inducing a total of 300 different objects (240 objects are used for training, 60 for testing). To the best of our knowledge, the number of different objects and its diversity is greater than concurrent works ((Chaplot et al., 2018) uses 55 train instructions and 15 test instructions and (Hill et al., 2017) has a total of 40 different objects). The agent has four actions {forward, left, right, pick}, and it can only see the 7x7 grid in front of itself. For each episode, one object's attribute is randomly picked as a goal, and the text

	Generator	**Policy**
Batchsize	128	128
Learning Rate	3e-4	1e-5
Word embedding size	32	32
Hidden MLP Size	256	128
Conv channel	[32,64,128]	[32,64,128]
Conv size	[2,2,2]	[2,2,2]
Conv stride	[1,1,1]	[1,1,1]
GRU size	256	128
Dropout	0.5	X
Buffer Size	X	20000
Gamma	X	0.99

Table 4.1: Generator and Policy Hyperparameters.

generator translates it in a synthetic language, e.g., "Fetch a tiny light blue ball." The agent is rewarded when picking one object matching the goal description, which ends the episode; otherwise, the episode stops after 40 steps or after taking an incorrect object (see Fig. 4.2).

Task Complexity It is important to underline the complexity of this task. To get rewards over multiple episodes, the agent must learn to navigate and inspect objects in the room while learning the meaning of each word and how they relate to visual characteristics. The burden comes from reward sparsity as the replay buffer is filled with unsuccessful trajectories RL fails to learn. Alleviating this problem is essential and minigrid is an excellent testbed to assess algorithmic performances as an agent deals with partial observability, visual representation learning, and language grounding only from sparse rewards signal.

Models In this experiment, HIGhER is composed of two separate models (see Fig. 4.2 for an illustration and details). The instruction generator is a neural network outputting a sequence of words given the final state of a trajectory. It is trained by gradient descent using a cross-entropy loss, adam optimizer (learning rate of 3e4, batchsize of 128) on the dataset D collected as described in Section 4.4. We train a DQN network following Mnih et al., 2015 with a dueling head (Wang et al., 2016), double Q-learning (Hasselt et al., 2016), and a prioritized replay buffer (Schaul et al., 2016) over trajectories. The network receives a tuple $< s, g >$ as input and outputs an action corresponding to the argmax over states-actions values $Q(s, a, g)$. We use ϵ-greedy exploration with decaying ϵ.

4.6.2 Instruction Generator Analysis

This section examines the feasibility of HIGhER by analysing two potential issues. We first show that HER is robust to a noisy mapping function (or partially incorrect goals), we then estimate the accuracy and generalisation performance of the instruction generator.

Algorithm 2: Hindsight Generation for Experience Replay (HIGhER)

Given:

- an off-policy RL algorithm (e.g. DQN) A and its associated behavioral policy π
- a reward function $r : \mathbb{S} \times \mathbb{A} \times \mathbb{G} \to \mathbb{R}$.
- a language score (e.g. parser accuracy, BLEU etc.)

```
1  Initialize $A$, replay buffer $\mathbb{B}$, dataset $\mathbb{D}_{train}$ and $\mathbb{D}_{val}$ of $\langle instruction, state \rangle$, Instruction
     Generator $m_{\theta_m}$;
2  for episode=1,M do
3      Sample a goal $g$ and an initial state $s_0$;
4      $t = 0$;
5      repeat
6          Execute an action $a_t$ chosen from the behavioral policy, $a_t \leftarrow \pi(s_t||g)$;
7          Observe a reward $r_t = r(s_t, a_t, g)$ and a new state $s_{t+1}$;
8          Store the transition $(s_t, a_t, r_t, s_{t+1}, g)$ in $\mathbb{B}$;
9          Update $Q$-network parameters using sampled minibatches from $\mathbb{B}$;
10         $t = t + 1$;
11     until episode ends;
12     if $v(s_t, g) = 1$ then
13         Store the pair $\langle s_t, g \rangle$ in $\mathbb{D}_{train}$ or $\mathbb{D}_{val}$;
14         Update $m_{\theta_m}$ parameters by sampling minibatches from $\mathbb{D}_{train}$;
15     end
16     else
17         if $m_{\theta_m}$ language validation score is high enough and $\mathbb{D}_{val}$ is big enough
             then
18             Sample $\hat{g}' = m_{\theta_m}(s_t)$;
19             Replace $g$ by $\hat{g}'$ in the transitions of the last episode and set $\hat{r} = r(s_t, a_t, \hat{g}')$.
20         end
21     end
22 end
```

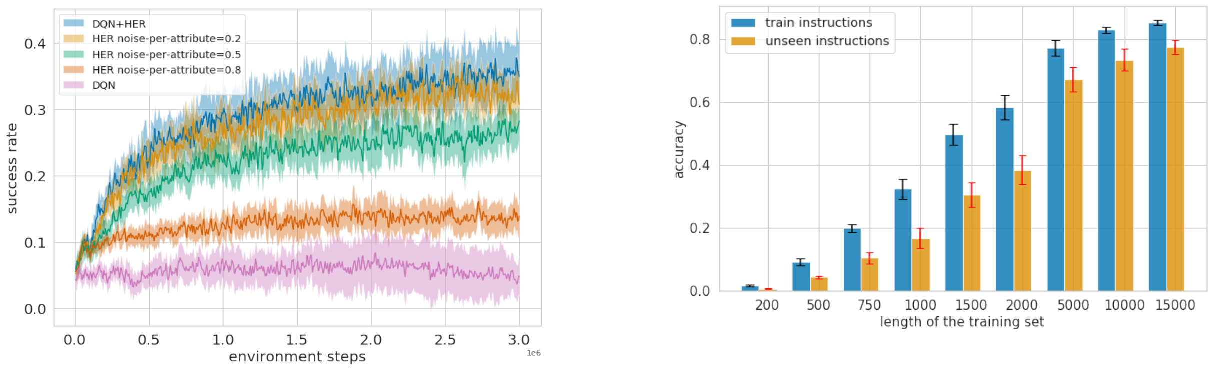


Figure 4.3: **Top**: Agent performance with noisy mapping function. **Bottom**: Instruction generator accuracy over 5k pairs. Figures are averaged over 5 seeds and error bars shows one standard deviation.

4.6.2.1 Noisy instruction generator and HER

We investigate how a noisy mapping m affects performance compared to a perfect mapping. As the learned instruction generator is likely to be imperfect, it is crucial to assess how a noisy mapping may alter the training of the agent. To do so, we train an agent with HER and a synthetic bot to relabel unsuccessful trajectories. We then inject noise in our mapping where each attribute has a fixed probability p to be swapped, e.g. color *blue* may be changed to *green*. For example, when $p = 0.2$, the probability of having the whole instruction correct is $0.8^4 \approx 0.4$. The resulting agent performance is depicted in Fig. 4.3 (left).

The agent performs 80% as well as an agent with perfect expert feedback even when the mapping function has a 50% noise-ratio per attribute. Surprisingly, even highly noisy mappers, with a 80% noise-ratio, still provides an improvement over vanilla DQN-agents. Hence, HER can be applied even when relabelling trajectories with partially correct goals.

We also examine whether this robustness may be induced by the environment properties (e.g. attribute redundancy) rather than HER. We thus compute the number of discriminative features required to pick the correct object. On average, an object can be discriminated with 1.7 features in our setting - which facilitates training, but any object shares at least one property with any other object 70% of the time - which tangles training. Besides, the agent does not know which features are noisy or important. Thus, the agent still has to disentangle the instructions across trajectories in the replay buffer, and this process is still relatively robust to noise.

4.6.2.2 Learning an Instruction Generator

We briefly analyze the sample complexity and generalization properties of the instruction generator. If training the mapping function is more straightforward than learning the agent policy, then we can thus use it to speed up the navigation training. We artificially generate datasets to train the instruction generator on and assess its generalization.

We first split the set of instructions G into two disjoint sets G_{train} and G_{test}. Although all object features are present in both sets, they contain dissimilar combinations of target objects. For instance, *blue*, *dark*, *key*, and *large* are individually present in instructions

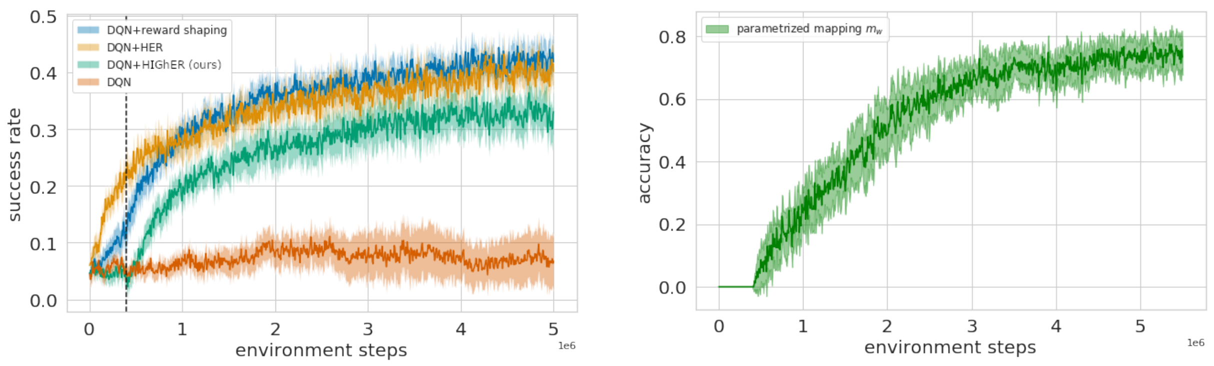


Figure 4.4: **Left**: learning curves for DQN, DQN+HER, DQN+HIGhER in a 10x10 gridworld with 10 objects with 4 attributes. The instruction generator is used after the vertical bar. **Right**: the mapping accuracy for the prediction of instructions. $m_{\boldsymbol{w}}$ starts being trained after collecting 1000 positive trajectories. Results are averaged over 5 seeds with one standard deviation.

of G_{train} and G_{test} but the instruction to get a *large dark blue key* is only in G_{test}. We therefore assess whether a basic compositionality is learned. In the following, we use train/split ratio of 80/20, i.e., 240 vs 60 goals.

We here observe than 1000 positive episodes are necessary to reach around 20% accuracy with our model, and 5000 pairs are enough to reach 70% accuracy. The instruction generator also correctly predicts unseen instructions even with fewer than 1000 samples and the accuracy gap between seen and unseen instructions slowly decrease during training, showing basic compositionality acquisition. As further discussed in Section 4.5, we here use a vanilla mapping architecture to assess the generality of our HIGhER, and more advanced architectures may drastically improve sample complexity (Bahdanau et al., 2019b).

4.6.3 HIGhER for Instruction Following

In the previous section, we observe that: (1) HER is robust to noisy relabeled goals, (2) an instructor generator requires few positive samples to learn basic language compositionality. We thus here combine those two properties to execute HIGhER, i.e. jointly learning the agent policy and language prediction in a online fashion for instruction following.

4.6.3.1 Baselines

We want to assess if the agent benefits from learning an instruction generator and using it to substitute goals as done in HER. We denote this approach DQN+HIGhER. We compare our approach to DQN without goal substitution (called DQN) and DQN with goal substitution from a perfect mapping provided by an external expert (called DQN+HER) available in the BabyAI environment. We emphasize again that it is impossible to have an external expert to apply HER in the general case. Therefore, DQN is a lower bound that we expect to outperform, whereas DQN+HER is the upper bound as the learned mapping can not outperform the expert. Note that we only start using the parametrized mapping function after collecting 1000 positive trajectories, which is around 18% validation accuracy. Finally, we compute an additional DQN baseline denoted DQN+reward: we reward the agent with 0.25 for each matching properties when picking a object given an instruction. It enforces a hand-crafted curriculum and dramatically reduces the reward sparsity, which gives a different perspective on the current task difficulty.

4.6.3.2 Results

In Fig. 4.4 (left), we show the success rate of the benchmarked algorithms per environment steps. We first observe that DQN does not manage to learn a good policy, and pick a random object every episode (with 10 objects, picking at random gives a success rate of 0.1). DQN+HER and DQN+reward quickly manage to pick the correct object 40% of the time. Finally, DQN+HIGhER's success rates increases as soon as we use the mapping function, to rapidly perform nearly as well as DQN+HER. Fig. 4.4 (right) shows the performance accuracy of the mapping generator by environment steps. We observe a steady improvement of the accuracy during training before reaching 78% accuracy after 5M steps. In the end, DQN+HIGhER outperforms DQN by using the exact same amount of information, and even matches the conceptual upper bond computed by DQN+HER. Besides, HIGhER does not alter the optimal policy which can occur when reshaping the reward (Ng et al., 1999). As stated in Section 4.6.3.1, a gap remains between DQN+HER and HIGhER as the latter is building an approximate model of the instruction, thus sometimes failing to relabel correctly as pointed out in Fig. 4.3

4.6.4 Discussion

Improvements over DQN As observed in the previous noisy-HER experiment, the policy success rate starts increasing even when the mapping accuracy is 20%, and DQN+ HIGhER becomes nearly as good as DQN+HER despite having a maximum mapping accuracy of 78%. It demonstrates that DQN+HIGhER manages to trigger the policy learning by better leveraging environment signals compared to DQN. As the instruction generator focuses solely on grounding language, it quickly provides additional training signal to the agent, initiating the navigation learning process.

Generator Analysis We observe that the number of positive trajectories needed to learn a non-random mapping m_{θ_m} is lower than the number of positive trajectories needed to obtain a valid policy with DQN (even after 5M environment steps the policy has 10% success rate). Noticeably, we artificially generate a dataset in Section 4.6.2.2 to train the instruction generator, whereas we follow the agent policy to collect the dataset, which is a more realistic setting. For instance, as the instructor generator is trained on a moving dataset, it could overfit to the first positive samples, but in practice it escapes from local minima and obtains a high final accuracy.

Different factors may also explain the learning speed discrepancy: supervised learning has less variance than reinforcement learning as it has no long-term dependency. The agent instructor generator can also rely on simpler neural architectures than the agent. HIGhER takes advantage of those training facilities to reward the agent ultimately.

Robustness Finally, we observe a virtuous circle that arises. As soon as the mapping is correct, the agent success rate increases, initiating the synergy. The agent then provides additional ground-truth mapping pairs, which increases the mapping accuracy, which improves the quality of substitute goals, which further increases the agent success rate. As a result, there is a natural synergy that occurs between language grounding and navigation policy as each module iteratively provides better training samples to the other model. If we ignore time-out trajectories, around 90% of the trajectories are negative at the beginning of the training. As soon as we start using the instruction generator, 40% the transitions are relabelled by the instructor generator, and 10% of the transitions belong to positive trajectories. As training goes, this ratio is slowly inverted, and after 5M steps, there is only 15% relabelled trajectories left while 60% are actual positive trajectories.

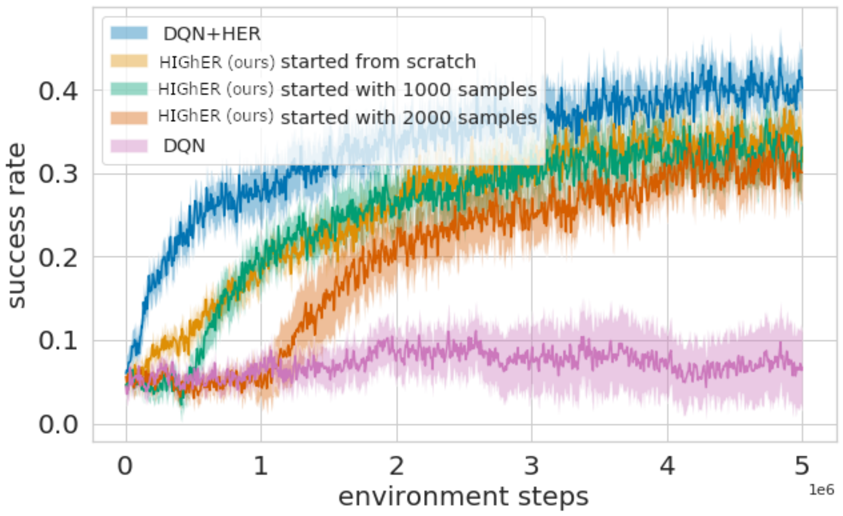


Figure 4.5: The instruction generator is triggered after collecting 0, 1000 and 2000 positive trajectories (i.e, approximately 0%, 20%, 50% accuracy). Even when the instruction generator is not accurate, the policy still makes steady progress and the final success rate is not impacted. Delaying the generator instructor does not provide additional benefit

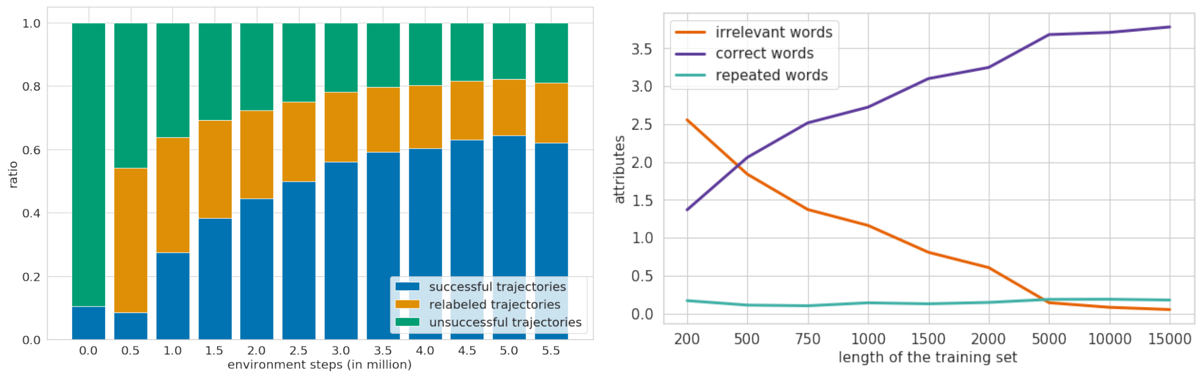


Figure 4.6: Transition distributions in the replay buffer between successful, unsuccessful and relabeled trajectories. We remove time-out trajectories for clarity, which accounts for 54% of the transition in average (±3% over training). **Right**: Evaluating the language learned by the instruction generator on unseen instructions. Over time, the number of correct attributes (purple) is increasing, as the number of irrelevant words (orange) is decreasing. The number of repeated attributes (green) stays low. The beginning clause is ignored as it doesn't provide information regarding the objective.

4.6.5 Language Learned by the Instruction Generator

We here analyze further the language quality of the instruction generator. To do so, we rely on three metrics to assess the generated language quality. The first metric, called *attribute fidelity*, assesses whether every target attribute is present in the generated sentence. For example, for the objective *a large dark blue key*, the generated sentence "Fetch me a large key" only containing two attributes and receives a score of two. *Language precision* counter this effect by counting how many words are not relevant to describe the target object. This second metric is related to precision (or positive predictive value), as generated instructions only contain relevant attributes. Finally, we count *repeated attributes* as language models are known to stutter during early training.

In Fig. 4.6, we compute the three metrics over unseen goal states, examining the compositionality properties of the instruction generator. We observe that generated instructions get more accurate, contain less irrelevant attributes, thus providing the agent with valid goals, even in unseen scenarios. As the instruction generator is trained until convergence as new <state, instruction> pairs are collected, it naturally preserve the overall language structure, and correctly ground symbols: *repeated attributes* score remains low and generated sentences start with the verb and end with the noun while randomly shuffling the attributes as shown in Table 4.2.

#Samples \ Instruction	get a small very_light green key	go fetch a dark grey giant ball
200	get a neutral very_light tiny ball	go get a grey giant neutral giant neutral grey
1000	get a very_light green small ball	you must fetch a grey dark giant ball
10000	get a very_light green small key	go fetch a grey dark giant ball

Table 4.2: Examples of language errors during the training

4.6.6 Complementary Experiment

An n-gram is a sequence of words, e.g. 2-gram corresponds to a two-word sequence. For example the sentence *Get a red ball.* is composed of three 2-gram: *Get a*, *a red*, *red ball* and two 3-gram: *get a red* and *a red ball.* The n-gram measure assesses the language model accuracy by counting how many n-grams in the original sentence is present in the generated one. This measure is close to BLEU score used in machine translation (Papineni et al., 2002).

In our experiments, the language used is synthetic, and attributes *order* is random. Therefore, the attributes' presence is more important than the position of each word. To assess the learned language accuracy, we compare generated sequences to what we call *randomized ground truth*. Comparing generated instruction to instructions generated by the environment is not relevant as the beginning clause (i.e., *Get a* or *Fetch a*, etc.) and attributes order are random. Therefore, for a given ground truth instruction, object attributes are shuffled in the sentence, and the beginning clause is sampled from all possible clauses. Since the beginning clause is random, even randomized ground truth cannot reach an accuracy of 1. The lower bound called *Random Attributes* corresponds to sampling a clause and each attribute randomly.

Fig. 4.7 shows that the language learned by the instruction generator is close to the upper bound *randomized ground truth*. These results correlate with Fig. 4.6 (right),

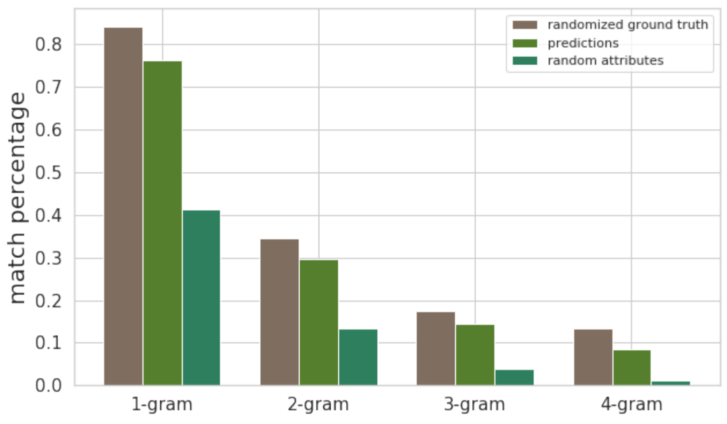


Figure 4.7: Quality of the language learned by the instruction generator with 10 000 samples. For *randomized ground truth* the sentences are the same as the ones from the ground truth but the order of the attributes is shuffled and the clause is changed. For *random attributes* beginning clause and object attributes are picked randomly.

indicating that the instruction generator can produce instruction containing correct object attributes. The fast decrease in accuracy when n grows can be explained by the attributes order randomness.

4.6.7 Limitations

Albeit generic, HIGhER also faces some inherent limitations. From a linguistic perspective, it cannot transcribe negative instructions (**Do not pick the red ball**), or alternatives (**Pick the red ball or the blue key**) in its current form. However, this problem could be alleviated by batching several trajectories with the same goal. Therefore, the model would potentially learn to factorize trajectories into a single language objective. On the policy side, HIGhER still requires a few trajectories to work, and it thus relies on the navigation policy. In other words, historical HER could be applied in the absence of reward signals, while HIGhER only alleviate the sparse reward problem by better leveraging successful trajectories. A natural improvement would be to couple HIGhER with other exploration methods, e.g, intrinsic motivation (Bellemare et al., 2016) or DQN with human demonstration (Hester et al., 2018). Finally, under-trained goal generators might hurt the training in some environments although we did not observe it in our setting as shown in Fig. 4.5. However, a simple validation accuracy allows to circumvent this risk while activating the goal mapper (More details in Algorithm 2). We emphasize again that the instruction generator can be triggered anytime to kick-start the learning as it is independent from the agent.

4.7 Conclusion

We introduced Hindsight Generation for Experience Replay (HIGhER) as an extension to HER for language. We defined a protocol to learn a mapping function to relabel unsuccessful trajectories with predicted consistent language instructions. We showed that HIGhER nearly matches HER performances despite only relying on signals from the environment. We provide empirical evidence that HIGhER manages to alleviate the instruction-following task by jointly learning language grounding and navigation policy with training synergies. HIGhER has mild underlying assumptions, making it valuable to complement to other instruction following methods. More generally, HIGhER can be though as a generative goal mechanism. Automatic goal generation is central to building autonomous agents that learn in open environment and many recent work build upon goal self-generation (Forestier et al., 2017; Colas et al., 2019; Colas et al., 2020a; Kurita et al., 2020; Akakzia et al., 2021), and we expect to see more work in this direction.

Chapter 5

Turning Supervised Learning Into Multi-Turn Interactive Tasks

In this chapter, we will take a step aside and think more broadly about Reinforcement Learning and how its interactivity may help Supervised Learning (**SL**). Supervised learning is a fixed framework, the model receives an input and outputs a prediction. The process is, by design, non-sequential. However, by changing the way the problem is framed, by turning the problem into a multi-turn interactive task, we may reduce the number of samples required to solve such tasks.

The term "interactive" in the following context means that the model can query information during the training phase. Asking to annotate additionnal examples falls under the umbrella of active learning. A section is dedicated to the usage of reinforcement learning in this context (Section 5.1). Another interactive setup, under which our work falls (Section 5.3 and Seurin et al., 2020b) is to query information on the input (Section 5.2), a setup we call Sequential Interactive Learning.

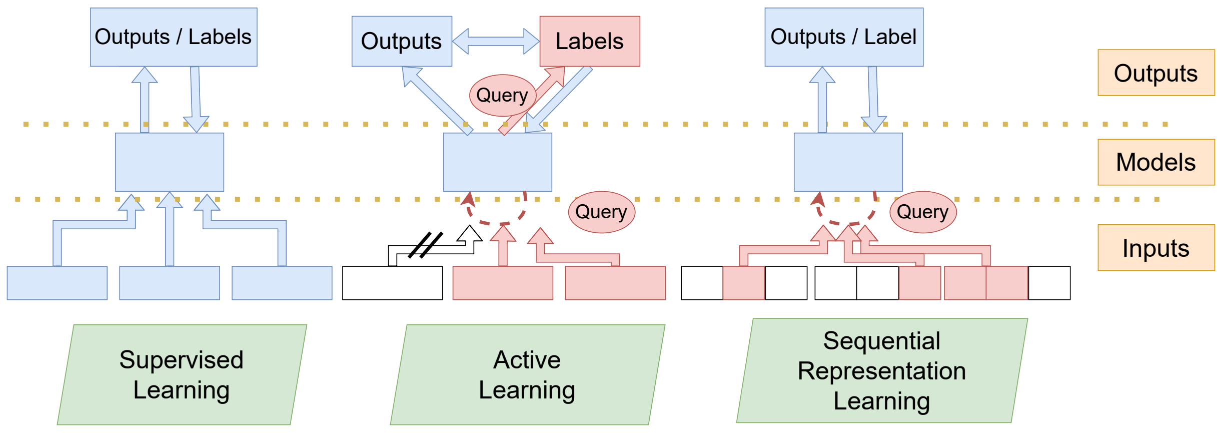


Figure 5.1: Three setups described in this chapter : Supervised Learning, Active Learning and Sequential Representation Learning. Reinforcement Learning and its interactivity can help during the querying in Active Learning and Sequential Representation Learning

5.1 Active Learning and Reinforcement Learning

Active learning (Tong, 2001) is a supervised learning paradigm where the dataset is not labeled. The model selects some samples and query an oracle (e.g human annotator) for labels. The goal is to reduce the burden of annotating the whole dataset beforehand while still maintaining (or even increasing) accuracy. The field of active learning mostly relies on uncertainty estimation (Ren et al., 2020) of each sample. The problem can be framed as single turn game (Contardo et al., 2017), all samples are selected for annotation at the same time. However, we lack a multi-turn approach. Thus, the problem becomes sequential, starting with an untrained model and refining iteratively, alternating training and querying phases. Uncertainty methods are "greedy", picking the most uncertain samples at each step. Since there exists a great diversity of uncertainty measures, none of which stands out as the "best" for active learning, the problem can be seen as an exploration/exploitation problem (Collet et al., 2015; Haussmann et al., 2019). Reinforcement learning can be seen as a nice complementary approach to balance between different measures and adjust the querying mechanism (Liu et al., 2019b). Fang et al., 2017 uses a similar strategy in low-ressource language.

5.2 Sequential Representation Learning

In the following section we consider a second form of interactive learning, where part of the **input** is queried sequentially. We call this approach Sequential Representation Learning (**SRL**). The supervised framework cannot account for this type of granularity where the system must query information before classifiying a sample, instead of processing the input entirely. Sequential Representation Learning can be seen as a form of Structured Prediction solved using the **MDP** formalism Maes et al., 2009; Maes, 2009

Computer vision application blossomed (Minut et al., 2001; Mnih et al., 2014), as such techniques reduce model size, by focusing on small object detector (König et al., 2019) instead of using large global models. **SRL** was also used in object tracking (Jiang et al., 2018) and face tracking in video (Rao et al., 2017). It was also used in the multi-modal dialog, Guesswhat?! (Strub et al., 2017) to locate an object secretly picked up by another player.

In subsequent sections, we focus on Automatic Speaker Recognition (**ASR** Saquib et al., 2010), where a system must identify speakers using few samples of raw audio. Speaker Recognition systems are trained to extract speaker-specific features from speech signals, and during evaluation, test speaker utterances are compared with the already existing utterances. However, dozens of tests recordings are necessary, limiting usage when interacting with humans. When identifying a speaker, only some key features might be necessary, such as certain inflexions or speech mannerisms. However, those discrimative features vary from speaker to speaker. Some pronunciation might be typical of certain speakers. For example, the phoneme 'r' might be pronounced differently depending on your accent. Thus starting with general phoneme and refining based on the received utterances could result in better recognition systems and reduce the amount of phonemes needed.

5.3 Fourth Contribution (INTERSPEECH'20): Machine of Few Words - Interactive Speaker Recognition with Reinforcement Learning

"Good words are worth much and cost little." - George Herbert
In the following section, we build a speaker recognition system that can identify a speaker by using a limited and personalized number of words. Instead of relying on full test utterances across all individuals, we interact with the speakers to iteratively select the most discriminative words.

More generally, a desirable feature of speaker recognition is to adapt its strategy to the current speaker as important features vary from person to person.

Here we propose to envision the problem of building a representation of the speaker as a sequential decision-making problem. The system we want to develop will select words that a speaker must utter so that it can be recognized as fast as possible. We adapt a standard **RL** algorithm to interact with a speaker to maximize the identification accuracy given as little data as possible. After introducing an Interactive Speaker Recognition (**ISR**) game based on the TIMIT dataset to simulate the speaker ASR interaction, we show that the **RL** agent builds an iterative strategy that achieves better recognition performance while querying only a few words. **RL** has been used in speech-based applications such as dialog (Chandramohan et al., 2010; Chandramohan et al., 2012; Khouzaimi et al., 2015) but not to the problem of speaker identification (note that Pietquin et al., 2005 combines **RL** and phonemes similarity).

The contribution of the following sections are thus:

1. To introduce the Interactive Speaker Recognition as an interactive game between the SR module and a human (Sec. 5.4);

2. To formalize ISR as a Markov Decision Process (Puterman, 2014) so as to solve the problem with RL (Sec. 5.5);

3. To introduce a practical Deep RL **ISR** model, and train it on actual data (Sec. 5.6).

Finally, we test our method on the TIMIT dataset and show that **ISR** model successfully personalized the words it requests toward improving speaker identification, outperforming two non-interactive baselines (Sec. 5.7).

5.4 Interactive Speaker Recognition Game

We aim to design an Interactive Speaker Recognition (**ISR**) module that identifies a speaker from a list of speakers only by requesting to utter a few user-specific words. To do so, we first formalize the **ISR** task as an interactive game involving the speaker and the **ISR** module. We then define the notation used to formally describe the game before detailing how we designed the **ISR** module.

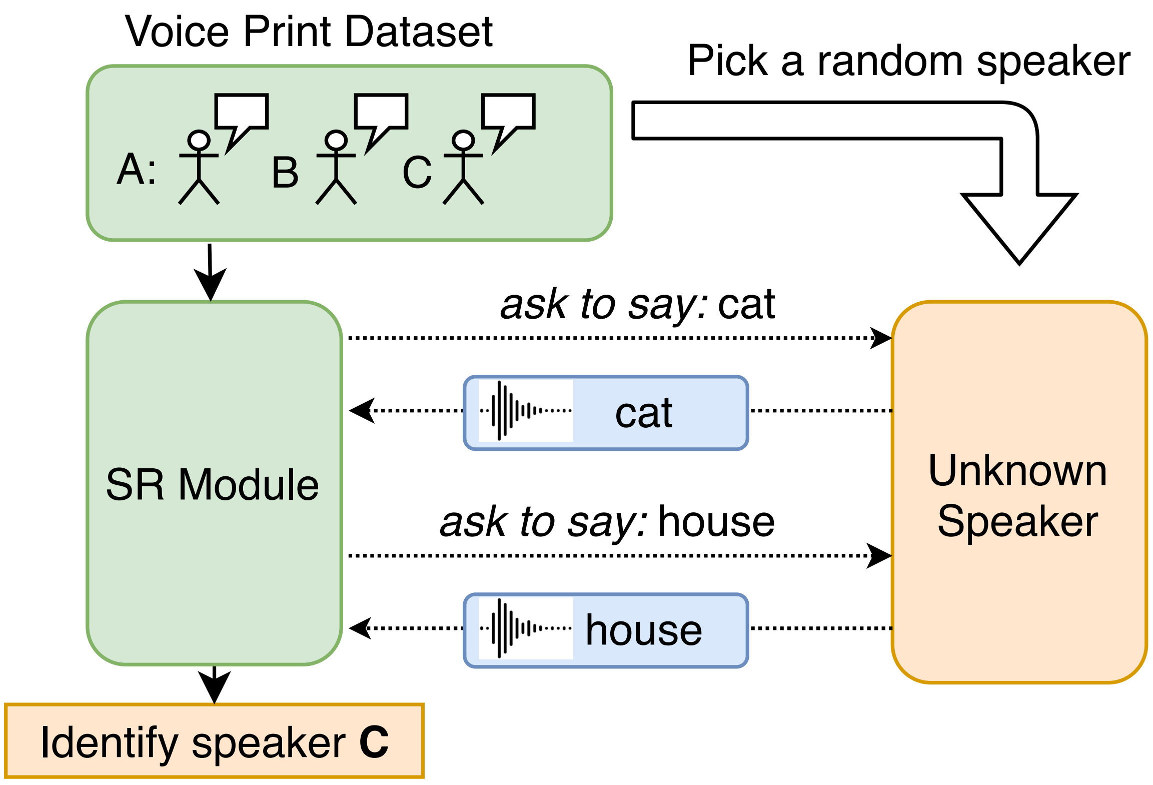


Figure 5.2: Interactive Speaker Recognition game overview

5.4.1 Game Rules

To instantiate the **ISR** game, we first build a list of random individuals, or *guests*. Each guest is characterized by a few spoken sentences (enrolment phase), which act as their signature that we call *voice print*. In a second step, we label one of the guests as the target *speaker* that we aim to identify. Hence a game is defined by d guests characterized with d voice prints, and one of these guests is labeled as the speaker.

As the game starts, the M voice prints are provided to the **ISR** module, and it needs to identify the speaker among the guests. To do so, the **ISR** engine may interact with the speaker, but it can only request the speaker to utter T words within a predefined vocabulary list. At each turn of the game, the **ISR** module asks the speaker to say a word, the speaker pronounces it, and the **ISR** engine updates its internal speaker representation, as detailed in Section 5.6.3, before asking the next word. Again, the **ISR** module may only request T words. Thus, it needs to carefully choose them to correctly identify the speaker.

5.4.2 Game notation

A game is composed of a list of d guests characterized by their voice print $\boldsymbol{m} = [m^u]_{u=1}^d$ where $\boldsymbol{m}$ is a subset from a larger group of registered guests $\mathbb{M}$ of size M, and a predefined vocabulary $\mathbb{C}$ of size c. The **ISR** module aims at building a list of words $\boldsymbol{w} = [w_t]_{t=1}^T \in \mathbb{C}$ to be uttered by the speaker. The uttered version of $\boldsymbol{w}$ is $\boldsymbol{z} = \{z_t\}_{t=1}^T$, where z_t is the representation of word w_t pronounced by the speaker. Note that, for a given $\boldsymbol{w}$, $\boldsymbol{z}$ differs from one speaker to another.

5.4.3 Modelling the Speaker Recognition Module

From a machine learning perspective, we aim to design an **ISR** module that actively builds an internal speaker representation to perform voice print classification. As further discussed in Section 5.6.2, this setting differs from standard SR methods that rely on generic but often long utterances (Snyder et al., 2018). In practice, we can split this task into two sub-modules: 1) an interactive module that queries the speaker to build the representation, and 2) a module that performs the voice print classification. In the following, we refer to these modules as *enquirer* and *guesser*.

Formally, the guesser must retrieve the speaker in a list of d guests characterized by their voice print $m \in \mathbb{M}$ and a sequence of words $\boldsymbol{z}$ uttered by the speaker $m^* \in \boldsymbol{m}$. Thus, the guesser has to link the speaker's uttered words to the speaker's voice print. The enquirer must select the next word w_{t+1} in vocabulary $\mathbb{C}$ that should be pronounced by the speaker given a list of d guests and the sequence of t previously spoken words $[z_{t'}]_{t'=1}^{t}$. Thus, the enquirer's goal is to pick the word that maximizes the guesser's success rate. Therefore, the **ISR** module first queries the speaker with the enquirer. Once the T words are collected, they are forwarded to the guesser to perform the speaker retrieval. In practice, this artificial split allows training the guesser with vanilla supervised learning, i.e., by randomly sampling words to retrieve speakers. The enquirer can hence be trained through reinforcement learning, as explained in the next section.

5.5 Speaker Recognition as an RL Problem

We aim at maximizing the guesser success ratio by allowing the enquirer to interact with the speaker, which makes RL a natural fit to solve the task. In this section, we thus provide the necessary RL terminology before relating the enquirer to the RL setting and defining the optimization protocol.

5.5.1 Markov Decision Process

The enquirer is a parametric policy $\pi_{\boldsymbol{\theta}}$ where $\boldsymbol{\theta}$ is a vector of neural network weights that will be learnt with RL. At the beginning of an episode, the initial state corresponds to the list of guests: $s_0 = \{\boldsymbol{m}\}$. At each time step t, the enquirer picks the action a_t by selecting the next word to utter w_t, where $w_t \sim \pi_{\boldsymbol{\theta}}(s_t)$. a_t is choosen among a fixed vocabulary that the speaker can pronounce. The speaker then pronounces the word w_t, which is processed to obtain z_t before being appended to the state $s_{t+1} = s_t \cup \{z_t\}$. After T words, the state $s_T = \{\boldsymbol{g}, \boldsymbol{z}\}$ is provided to the guesser. The enquirer is rewarded whenever the guesser identifies the speaker, i.e. $r(s_t, a_t) = 0$ if $t < T$ and $r(s_T, a_T) = \mathbf{1}_{[\arg\max_k p(m_k|s_T) = m^*]}$ where $\mathbf{1}$ is the indicator function and $p(m_k|s_T)$ is the output probability given by the guesser.

5.5.2 Enquirer optimization Process

In RL, policy search aims at learning the policy $\pi_{\boldsymbol{\theta}^*}$ that maximizes the expected return by directly optimizing the policy parameters $\boldsymbol{\theta}$. More precisely, we search to maximize the mean value defined as $J(\boldsymbol{\theta}) = E_{\pi_{\boldsymbol{\theta}}}\big[\sum_{t=1}^{T} \gamma^{t-1} r(\boldsymbol{s}_t, a_t)\big]$. To do so, the policy parameters are updated in the direction of the gradient of $J(\boldsymbol{\theta})$. In practice, direct approximation of

$\nabla J(\boldsymbol{\theta})$ may lead to destructively large policy updates, may converge to a poor deterministic policy at early training and it has a high variance. We thus use the recent Proximal Policy Optimization approach (PPO) (Schulman et al., 2017). PPO clips the gradient estimate to have smooth policy updates, adds an entropy term to soften the policy distribution (Geist et al., 2019), and introduce a parametric baseline to reduce the gradient variance (Mnih et al., 2016; Schulman et al., 2017).

5.6 Experimental Protocol

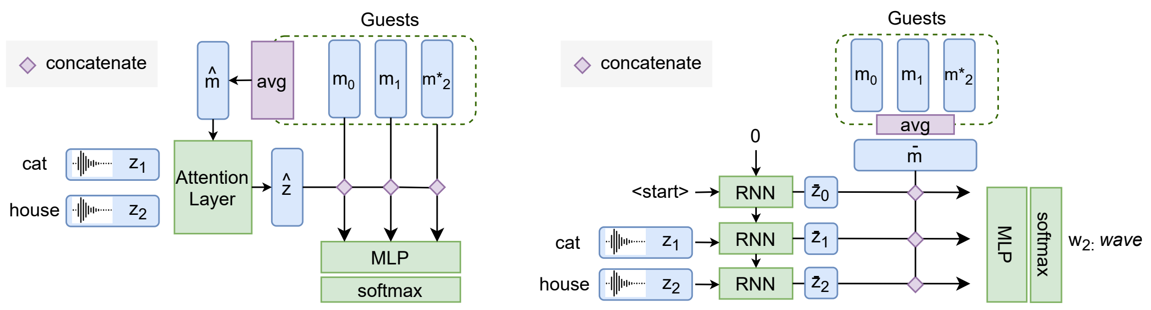


Figure 5.3: (Left) The guesser must retrieve the speaker in the list of guests. (Right) The enquirer must select the next word to utter.

We first detail the data we used to create the **ISR** game before describing the speech processing phase. Finally, we present the neural training procedure.

5.6.1 Dataset

We build the **ISR** game using the TIMIT corpus (Garofolo et al., 1992). This dataset contains the voice recordings of 630 speakers with eight different accents. Each speaker uttered ten sentences, where two sentences are shared among all the speakers, and the eight others differ. Sentences are encoded as 16-bit, 16kHz waveforms. First, we define the **ISR** vocabulary by extracting the words of the two shared sentences, so the enquirer module may always request these words whatever the target speaker. In total, we obtained twenty distinct words such as **dark**, **year**, **carry** while dropping the uninformative specifier **a**. Second, we use the eight remaining sentences to build the speakers' voice print.

5.6.2 Audio Processing

Following Snyder et al., 2018; Snyder et al., 2017; Snyder et al., 2016, we first downsample the waveform to 8kHz before extracting the Mel Frequency Cepstral Coefficient (MFCC). We use MFCCs of dimension 20 with a frame-length of 25ms, mean-normalized over a sliding window of three seconds. We then process the MFCCs features through a pretrained X-Vector network to obtain a high quality voice embedding of fixed dimension 128, where the X-Vector network is trained on augmented Switchboard (Godfrey et al., 1992), Mixer 6 (Chodroff et al., 2016), and NIST SREs (Doddington et al., 2000)[1]. To get the spoken word representation (word that the enquirer will query), we split the two shared sentences into individual words by following the TIMIT word timestamps. We then

[1]available in kaldi library (Povey et al., 2011) at http://www.kaldi-asr.org/models/m3

extract the X-Vector of each word w_t of every speaker u to obtain z_t^u. We compute the voice print by extracting the X-Vector of the eight remaining sentences before averaging them into a final vector of size 128 for each guest m^u.

5.6.3 Speaker Recognition Neural Modules

We here describe the **ISR** training details and illustrate the neural architectures in Fig. 5.3.

Guesser. To train the guesser, we first model the speaker by averaging the voice print into a single vector $\hat{m} = \frac{1}{d}\sum m$. We then pool the X-Vectors with an attention layer conditioned on $\hat{g}$ to get the guesser embedding $\hat{z}$ (Bahdanau et al., 2015):

$$e_t = MLP([z_t, \hat{m}]) \ ; \ \alpha = softmax(\boldsymbol{e}) \ ; \ \hat{z} = \sum_t \alpha_t z_t,$$

where [.,.] is the concatenation operator and MLP is a multilayer perceptron with one hidden layer of size 256. We concatenate the guesser embedding with the speaker voice print before projecting them through a MLP of size 512. Finally, we use a softmax to estimate the probability p^u of each guest to be the speaker, *i.e.* $p(m^u = m^* | \boldsymbol{z}, \boldsymbol{m}) = softmax\big(MLP([m^u, \hat{z}])\big)$. Both MLP have ReLU activations (Nair et al., 2010) with a dropout ratio of 0.5% (Srivastava et al., 2014). The guesser is trained by minimizing the cross-entropy with ADAM (Kingma et al., 2015), a batch size of 1024 and an initial learning rate of 3.10^{-4} over 45k games with five random guests.

Enquirer. To model the enquirer, we first represent the pseudo-sequence of words by feeding the X-Vectors into a bidirectional LSTM (Hochreiter et al., 1997) to get the word hidden state $\bar{x}_t$ of dimension 2*128. Note that we use a start token for the first iteration. In parallel, we average the voice print into a single vector $\bar{m} = \frac{1}{d}\sum m^u$ to get the guests context. We then concatenate the word hidden state $\bar{z}$ and the guest contexts $\bar{m}$ before processing them through a one-hidden-layer MLP of size 256 with ReLU. Finally, a softmax activation estimates the probability of requesting the speaker to utter w_{t+1} as the next word: $p(w_{t+1} | z_t, \cdots, z_1, \boldsymbol{m}) = softmax\big(MLP([\bar{z}_t, \bar{m}])\big)$. The enquirer is trained by maximizing the reward encoded as the the guesser success ratio with PPO (Schulman et al., 2017). We use the ADAM optimizer (Kingma et al., 2015) with a learning rate of 5e-3 and gradient clipping of 1 (Pascanu et al., 2013). We perform 80k episodes of length $T = 3$ steps and $d = 5$ random guests. When applying PPO, we use an entropy coefficient of 0.01, a PPO clipping of 0.2, a discount factor of 0.9, an advantage coefficient of 0.95, and we apply four training batches of size 512 every 1024 transitions.

5.7 Experiments

We run all experiments over five seeds, and report the mean and one-standard deviation when not specified otherwise.

5.7.1 Guesser Evaluation

In this section, we evaluate the guesser accuracy in different settings. As mentioned, we opt to request $T = 3$ words to identify the speaker among $d = 5$ guests. In this default setting, a random policy has a success ratio of 20%, whereas the neural model reaches

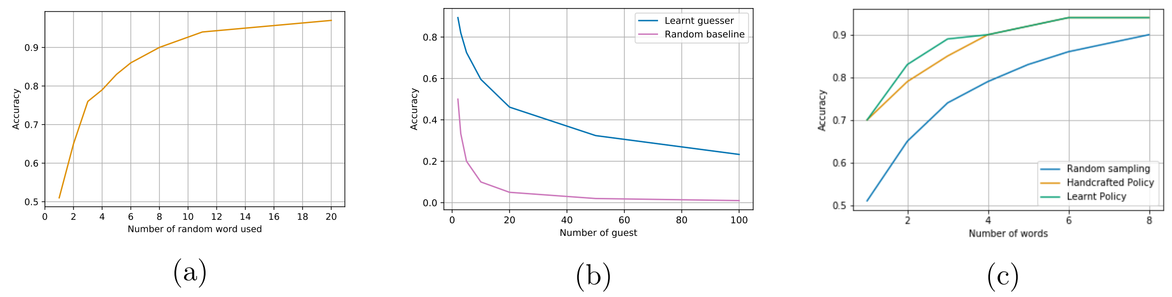


Figure 5.4: (a-b) Guesser test accuracy, respectively varying the number of words (resp. guests) being used (c) Enquirer test accuracy varying the number of queried words. The RL enquirer outperforms the heuristic baseline when selecting a low number of words.

74.1%±0.2 on the test set. As the guesser is trained on random words, these scores may be seen as an **ISR** lower-bound for the enquirer, which would later refine the word selection toward improving the guesser success ratio. Thus, this setting shows an excellent ratio between task difficulty and guesser initial success, allowing to train the enquirer with a relatively dense reward signal.

Word Sweep. We assess the guesser quality to successfully perform speaker recognition when increasing the number of words T in Fig. 5.4a. We observe that a single word only gives 50% speaker retrieval, but the accuracy keeps improving when requesting more words. Noticeably, collecting the full vocabulary only scores up to 97% accuracy.

Guest Sweep. We report the impact of the number of guests d in Fig. 5.4b. The guesser accuracy quickly collapses when increasing the number of guests with $d = 50$ having a 46% success ratio. As the number of words remains small, the guesser experiences increasing difficulty in discriminating the guests. One way to address this problem would be to use a Probabilistic Linear Discriminant Analysis (PLDA Ioffe, 2006) to enforce a discriminative space and explicitly separate the guests based on their class.

5.7.2 Enquirer Evaluation

Model. As previously mentioned, the enquirer aims to find the best sequence of words $\boldsymbol{w}$ that maximizes the guesser accuracy by interacting with the speaker. At each time step, we thus select the word with the highest probability $p(w_{t+1}|z_t, \cdots, z_1, \boldsymbol{m})$ according to the policy without replacement, i.e., the model never requests the same word twice.

Baseline. We compare our approach to two baselines: a random policy, and a heuristic policy. As the name suggests, the random baseline picks T random words without replacement. To obtain a strong baseline, we pre-select words by taking advantage of the guesser model, where we value a sequence of words by computing the guesser accuracy over $\rho = 20000$ games. Optimally, we want to iterate over every tuple of words to retrieve the optimal set; yet, it is computationally intractable as it requires $\rho * \binom{c}{T}$ estimations, where c is vocabulary's size. Therefore, we opt for a heuristic sampling mechanisms. We curated a list of the most discriminant words (words that increase globally the recognition scores) and sample among those instead of the whole list.

Results. In our default setting, the random baseline reaches 74.1% ± 0.2 speaker identification, and the heuristic baseline scores up to 85.1%. The RL enquirer obtains up to

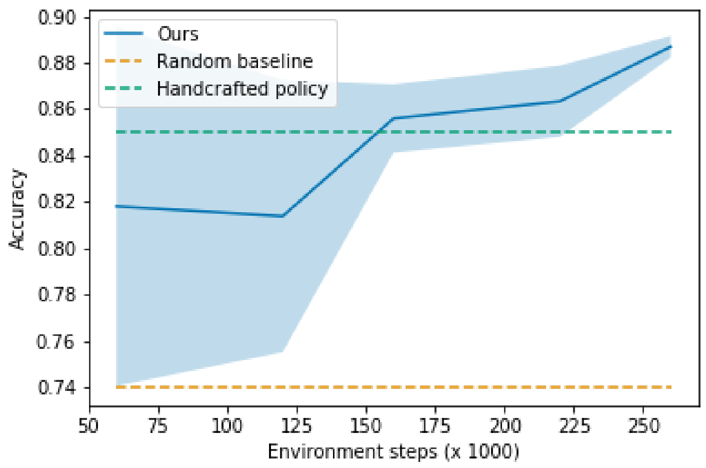


Figure 5.5: Enquirer test accuracy averaged over 3 random seeds

88.6% $\pm$ 0.5, showing that it successfully leverages the guests' voice prints to refine its policy. We show the RL training in Fig. 5.5. At early training, we observe that the **ISR** module still has high variance, and may behave randomly. However, RL enquirer steadily improves upon training, and it consistently outperforms the heuristic baseline.

Word Diversity. To verify whether the enquirer adapts its policy to the guests, we generate a game for every speaker in the test set, and collect the requested words. We then compute the overlap Ω between the tuple of words by estimating the averaged Jaccard-index Jaccard, 1901 of every pair of speakers as follow:

$$\Omega = \frac{1}{\sum_n^{N-1} n} \sum_{i=1}^{N-1} \sum_{j=i}^{N} J(\boldsymbol{w}^i, \boldsymbol{w}^j) \ ; \text{ where } \mathrm{J}(A, B) = \frac{A \cap B}{A \cup B},$$

where N is the number of speakers in the test set and $\boldsymbol{w}^i$ is the word tuple of game i. Intuitively, the lower this number, the more diverse the policy, e.g, the deterministic policy has a Jaccard-index of 1. In the default setting, the random policy has an index of 0.14 while the RL agent has an index of 0.65. Thus, the requested words are indeed diverse.

Requesting Additional Words We here study the impact of increasing the number of words T requested by the enquirer (see Fig. 5.4c for results). First, we observe that the **ISR** module manages to outperform the heuristic policy when requesting two to four words, showing that the interaction with the speaker is beneficial in the low data regime. This effect unsurprisingly diminishes when increasing the number of words. However, we noticed that the enquirer always outputs the same words when $t = 1$. It suggests that the model faces some difficulties contextualizing the guests' voice print before listening to the first speaker utterance. We assume that more advanced multimodal architecture, e.g., multimodal transformers (Lu et al., 2019; Tan et al., 2019), may ease representation learning, further improving the **ISR** agent.

5.8 Conclusions and Future Directions

In the last sections, we introduced the Interactive Speaker Recognition paradigm as an interactive game to improve speaker recognition accuracy while querying only a few words. We formalized it as a Markov Decision Process and trained a neural model using Reinforcement Learning. We showed empirically that the **ISR** model successfully personalizes the words it requests to improve speaker identification, outperforming two non-interactive baselines. Our protocol may go beyond speaker recognition. The model can be adapted to select speech segments in the context of Text-To-Speech training. Interactive querying may also prevent malicious voice generator usage by asking complex words to the generator in a speaker verification setting.

Ideas from preceding sections could be included to this setup. Action are words, thus considering the proximity between words (in the speech domain) could help to simplify policy's work.

Conclusion and Future Directions

I'm part of the second generation of deep reinforcement learning researchers. The first generation witnessed **DQN** (Mnih et al., 2015) and AlphaGo (Silver et al., 2016) while my generation saw the birth of AlphaStar (Vinyals et al., 2019). Reproducing and competing with those recents advances seems to be a lost cause. Algorithms such as IMPALA (Espeholt et al., 2018), R2D2 (Kapturowski et al., 2018) or SEED (Espeholt et al., 2019) require thousands of **CPU** and large **GPU**s or even **TPU**s. Exciting and eye-catching benchmarks were proposed (Beattie et al., 2016; Vinyals et al., 2017; Berner et al., 2019), but the amount of compute will always gives an edge over potential rivals. It is easy to get attracted by those appealing subjects and I plead guilty for trying to compete on those grounds. On the other hand plenty of subjects are still underexplored, and interesting topics are the ones that are still under the radar.

Thesis Summary

Chapter 5 starts with the idea that interactivity in machine learning is beneficial, even when facing supervised setups. By framing the problem of speaker identification as an interactive game, we reduced the amount of utterances required and increase recognition scores. Adaptivity to the speaker being queried is what gives the method an edge. Building upon this idea that interactivity is what set reinforcement learning appart from other methods, we focused our analysis on the actions. Chapter 2 surveys the different action spaces and highlights problem such as the curse of large spaces. We propose contextual ineffectiveness, actions that in certain context do not work, to explain why some environments are hard to learn. We showed experimentally that **DQN** struggles to detect contextual ineffectiveness and proposed an auxiliary loss to shape the Q-values. By using a signal given by the environment, this auxiliary loss is enough to make DQN efficient in two environements. A recent paper (Huang et al., 2020) provides an empirical analysis comparing invalid action masking and penalty masking (analogous to our method). They conclude that actor-critic methods can learn when masking the policy logit and even increases sample efficiency compared to penalizing invalid actions. Thus spending more time on modifying directly the action space to remove invalid actions might be a better strategy than shaping the Q-values (Zahavy et al., 2018). In 3, we played out with the idea that current exploration methods fail to use actions that rarely affects the environment. Thus, we designed **DoWhaM** an exploration strategy that exploits the action consequence as a signal to explore. Finally, in chapter 4 we developped the idea that goals are high-level actions and that learning and generalizing across goal is essential in building autonomous agent. To this end, we developped HIGhER, a generalization of Hindsight Experience Replay (**HER**) for language goal. This side-stepping was suppose to open new possibilities for goal-generation, by using the generator as a self-rewarding mechanism but sometimes, it does not work according to the plan.

Future Directions

In this thesis, we focused our attention on the actions, more specifically how informations on the actions spaces can increase the data-efficiency of reinforcement learning algorithms. The ambition layed in the introduction still remains, trying to build an agent that is able to adapt to the actions, without engineering. The contributions proposed a tiny step in this direction but generality is still far ahead. The following paragraph either develop promising directions treated in the thesis or propose new paths not explored in the preceding chapters.

Exploration using action knowledge Exploring actions with different consequences was a key idea behind **DoWhaM**. Detecting contextual ineffectiveness and focusing on actions that rarely trigger changes in the environment was one of many possibilites. Some actions can trigger a set of diverse consequences, for example shooting at an explosive barrel, an enemy or a wall impacts the environment differently. Finding states where actions trigger rare (or unseen) consequences and go beyond the binary approach in **DoWhaM** (effective or not effective) could be a next intesting step. One could build a cluster of consequences for each action and try to reach state that will fall in the smallest cluster, to explore under visited areas.

Continuous action spaces are harder to explore than discrete, potentially creating a conflict with action embeddings. Action embeddings reduce sample complexity but combining it with other exploration strategy is still an open problem. Also, developing better exploration strategy in continuous domains would benefit as a whole the community as many problems use continuous actions.

Adaptation and Transfer to new action spaces Quick adaptation to new domains and tasks is another potential application. Chen et al., 2019 used action embeddings to transfert between fighters in an arcade game and we expect to see more work in this direction. At the moment, standard benchmarks in this domain are missing, thus designing environments to assess adaptation capability will be key to advance the field. For example a set of standard games where actions concepts are similar but games looks different such as Cobbe et al., 2019. Concepts such as action removal at testing time, to assess robustness or adding new actions during training should be at the heart of those benchmarks. Another potential direction for transfer is to use language description to give quick insights on what the action are doing. For example, quick explanation such as "Action A jumps, B interact with object", similar to Zhong et al., 2019.

Strenghening of our understanding of actions spaces. Action embeddings has only been used for discrete domains, assessing how well this method adapts to continuous domain seems like a natural enhancement. In Chapter 2, we developped the idea of some action being *independent* such as in Factored-MDPs or space being *entangled*. Some spaces are harder to learn than others, thus it should inspire to learn some transformation that reduces the complexity of vectorial spaces, by finding orthogonal component for example. This idea could also be used to study how well action embeddings are disentangled.

Coming up with algorithms that are able to tackle hybrid domains will be essential as agents get more and more general, and tackle more complex setups. Games like Dota already have hybrid spaces and we expect to see more environments using those.

Goal-based method and language When considering a class of **MDP**'s, an action set can be minimal for certain instances but not for others. For example, multi-goal agents might use a set of actions to solve one task (opening doors) and leave aside inventory while another task do not involve doors and requires picking up objects. Thus, generalizing **DoWhaM** type of methods to multi-goal setups in another interesting venue.

Questions raised with action spaces (in Section 2.1) are relevant to goal spaces and studying more systematically those questions could leverage both domains:

1. "How many goals current methods can handle ?"
2. "How to measure goal space complexity ?"
3. "How can we leverage transfer between goals ?"
4. "How goal representation alleviates policy learning ?"

More generally, goal-based methods and finding ways for agents to generate their goals is a very interesting research avenue. HIGhER was a first step, using a goal-generation mechanisms, however, we expect to see more work in this direction in combination of language methods.

List of Acronyms

A

A2C Advantage Actor-Critic

A3C Asynchronous Advantage Actor-Critic

AI Artificial Intelligence

ASR Automatic Speaker Recognition

C

CAS Continuous Bounded Action Space

CMDP Constraint Markov Decision Process

CNN Convolutionnal Neural Network

D

DAS Discrete Finite Action Space

DDQN Dueling Double Q-Learning

DL Deep Learning

DoWhaM Do What Matters!

DP Dynamic Programming

DQN Deep Q-Learning

DRL Deep Reinforcement Learning

G

GB Giga-Bytes

GPU Graphics Processing Unit

H

HER Hinsight Experience Replay

I

ISR Interactive Speaker Recognition

L

LSTM Long Short-Term Memory : Recurrent Neural Architecture

M

MB Mega-Bytes

MDP Markov Decision Process

ML Machine Learning

MLP Multi-Layer Perceptron

N

NLP Natural Language Processing

NN Neural Network

P

PG Policy Gradient

POMDP Partially-Observable Markov Decision Process

R

RDQN Recurrent Deep Q-Network, DQN adaptation for POMDP

RL Reinforcement Learning

S

SL Supervised Learning

SRL Sequential Representation Learning

T

TB Tera-Bytes

TPU Tensor Processing Unit, hardware specialized in tensor processing, used in Deep Learning cluster

TTS Text-To-Speech

List of Symbols

Mathematical notations

E expectation over a distribution

x sample to be classified

$\mathbb{N}$ set of integers

y sample's label

θ weights of a neural net

$\mathbb{R}$ set of real numbers

t time index, discrete value greater than 0

Algorithm Constant

β Intrinsic reward weighting parameter

k number of discrete actions

γ discount factor in $[0, 1]$

ϵ ϵ-greedy exploration parameter

η Weighting parameter for a defined loss

ξ Decaying parameter, in ratio function B

Markov Decision Processes

Q state-action value function (* for optimal value, $^\pi$ for policy value)

V state value function (* for optimal value, $^\pi$ for policy value)

$\mathbb{A}$ set of actions $a \in \mathcal{A}$

$\mathbb{B}$ replay buffer

I^H Impact function $E^H : \mathbb{A} \to \mathbb{N}$

F feedback function, $F : \mathbb{S} \times \mathbb{A} \to \{0, 1\}$

$\mathbb{D}$ goal dataset

$\mathbb{G}$ set of goals $g \in \mathbb{G}$

m_{θ_m} mapping function, parametrized by θ_m, $m_{\theta_m} : \mathbb{S} \to \mathbb{G}$

$\pi^\star$ optimal policy

π_θ parametrized policy

π policy

B ratio function $B : \mathbb{A} \to [0, 1]$ for **DoWhaM** intrinsic reward

$R(s_t, a, s_{t+1})$ reward function $R : \mathbb{S} \times \mathbb{A} \times \mathbb{S} \to [0, 1]$

N^τ episodic state count within trajectory τ

$\mathcal{S}$ set of states $s \in \mathcal{S}$

τ episodic trajectory of length l $[(s_t, a_t, r_t)]_{t=0}^{l}$

$P(s' \mid s, a)$ stochastic transition fonction

U^H usage function $U^H : \mathbb{A} \to \mathbb{N}$

v goal validness function $v : \mathbb{S} \times \mathbb{G} \to \{0, 1\}$

Interactive Speaker Recognition

$\mathbf{m}$ list of voice print for the current game

d number of guest within an episode

$\mathbb{C}$ vocabulary composed of c words w

$\mathbb{M}$ set of all guests of size M

z word representation (X-vector)

Bibliography

Achiam, Joshua, David Held, Aviv Tamar, and Pieter Abbeel (2017a). “Constrained policy optimization”. In: **Proceedings of the International Conference on Machine Learning, (ICML)** (page 36).

Achiam, Joshua and Shankar Sastry (2017b). “Surprise-based intrinsic motivation for deep reinforcement learning”. In: **arXiv:1703.01732** (page 49).

Akakzia, Ahmed, Cédric Colas, Pierre-Yves Oudeyer, Mohamed Chetouani, and Olivier Sigaud (2021). “Grounding Language to Autonomously-Acquired Skills via Goal Generation”. In: **Proceedings of the International Conference on Learning Representations, (ICLR)** (page 77).

Akata, Zeynep, Florent Perronnin, Zaid Harchaoui, and Cordelia Schmid (2015). “Label-embedding for image classification”. In: **IEEE transactions on pattern analysis and machine intelligence** 38.7, pp. 1425–1438 (page 30).

Alshiekh, Mohammed, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu (2018). “Safe Reinforcement Learning via Shielding”. In: **Proceedings of the Internationnal Conference on Artificial Intelligence, (AAAI)** (pages 32, 37).

Altman, Eitan (1999). **Constrained Markov decision processes**. Vol. 7. CRC Press (page 36).

Ammanabrolu, Prithviraj and Mark Riedl (2019). “Playing Text-Adventure Games with Graph-Based Deep Reinforcement Learning”. In: **Proceedings of the Annual Meeting of the Association for Computational Linguistics, (ACL)** (page 19).

Anderson, Peter, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian D. Reid, Stephen Gould, and Anton van den Hengel (2018). “Vision-and-Language Navigation: Interpreting Visually-Grounded Navigation Instructions in Real Environments”. In: **Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, (CVPR)** (page 64).

Andrychowicz, Marcin, Dwight Crow, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba (2017). “Hindsight Experience Replay”. In: **Proceedings of the Neural Information Processing Systems Conference, (NIPS)** (pages 50, 65, 66).

Andrychowicz, OpenAI: Marcin, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. (2020). “Learning dexterous in-hand manipulation”. In: **The International Journal of Robotics Research** 39.1, pp. 3–20 (page 28).

Artzi, Yoav and Luke Zettlemoyer (2013). “Weakly Supervised Learning of Semantic Parsers for Mapping Instructions to Actions”. In: **Transactions of the Association for Computational Linguistics** 1, pp. 49–62. DOI: 10.1162/tacl_a_00209. URL: https://www.aclweb.org/anthology/Q13-1005 (page 63).

Artzner, Philippe, Freddy Delbaen, Jean-Marc Eber, and David Heath (1999). “Coherent measures of risk”. In: **Mathematical finance** 9.3, pp. 203–228 (page 35).

Asadi, Shahrokh, Esmaeil Hadavandi, Farhad Mehmanpazir, and Mohammad Masoud Nakhostin (2012). “Hybridization of evolutionary Levenberg–Marquardt neural networks and data pre-processing for stock market prediction”. In: **Knowledge-Based Systems** 35, pp. 245–258 (page 10).

Åström, Karl (1965). “Optimal control of Markov processes with incomplete state information I”. In: **Journal of Mathematical Analysis and Applications** 10, pp. 174–205 (page 18).

Auer, Peter, Thomas Jaksch, and Ronald Ortner (2008). “Near-optimal Regret Bounds for Reinforcement Learning”. In: **Proceedings of the Neural Information Processing Systems Conference, (NIPS)** (page 48).

Azar, Mohammad Gheshlaghi, Bilal Piot, Bernardo Avila Pires, Jean-Bastien Grill, Florent Altché, and Rémi Munos (2019). “World discovery models”. In: **arXiv:1902.07685** (page 49).

Azizzadenesheli, Kamyar, Emma Brunskill, and Animashree Anandkumar (2018). “Efficient exploration through bayesian deep q-networks”. In: **2018 Information Theory and Applications Workshop (ITA)** (page 47).

Babaeizadeh, M, I Frosio, S Tyree, J Clemons, and J Kautz (2017). “Reinforcement Learning through Asynchronous Advantage Actor-Critic on a GPU”. In: **Proceedings of the International Conference on Learning Representations, (ICLR)** (pages 21, 22).

Badia, Adrià Puigdomènech, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Zhaohan Daniel Guo, and Charles Blundell (2020a). “Agent57: Outperforming the Atari Human Benchmark”. In: **Proceedings of the International Conference on Machine Learning, (ICML)** (page 52).

Badia, Adrià Puigdomènech, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, Bilal Piot, Steven Kapturowski, Olivier Tieleman, Martıén Arjovsky, Alexander Pritzel, Andrew Bolt, and Charles Blundell (2020b). “Never Give Up: Learning Directed Exploration Strategies”. In: **Proceedings of the International Conference on Learning Representations, (ICLR)** (page 48).

Bahdanau, Dzmitry, Philemon Brakel, Kelvin Xu, Anirudh Goyal, Ryan Lowe, Joelle Pineau, Aaron Courville, and Yoshua Bengio (2016). “An Actor-Critic Algorithm for Sequence Prediction”. In: **Proceedings of the International Conference on Learning Representations, (ICLR)** (page 29).

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2015). “Neural Machine Translation by Jointly Learning to Align and Translate”. In: **Proceedings of the International Conference on Learning Representations, (ICLR)** (pages 11, 85).

Bahdanau, Dzmitry, Felix Hill, Jan Leike, Edward Hughes, Seyed Arian Hosseini, Pushmeet Kohli, and Edward Grefenstette (2019a). “Learning to Understand Goal Specifications by Modelling Reward”. In: **Proceedings of the International Conference on Learning Representations, (ICLR)** (page 68).

Bahdanau, Dzmitry, Shikhar Murty, Michael Noukhovitch, Thien Huu Nguyen, Harm de Vries, and Aaron C. Courville (2019b). “Systematic Generalization: What Is Required and Can It Be Learned?” In: **Proceedings of the International Conference on Learning Representations, (ICLR)** (page 72).

Barto, Andrew G and Sridhar Mahadevan (2003). “Recent advances in hierarchical reinforcement learning”. In: **Discrete event dynamic systems** 13.1-2, pp. 41–77 (page 63).

Barto, Andrew G and Satinder Pal Singh (1991). “On the computational economics of reinforcement learning”. In: **Connectionist Models**. Elsevier, pp. 35–44 (pages 45, 48).

Barto, Andrew G, Satinder Singh, and Nuttapong Chentanez (2004). “Intrinsically motivated learning of hierarchical collections of skills”. In: **Proceedings of the IEEE International Conference on Development and Learning, (ICDL)** (page 50).

Beattie, Charles, Joel Z. Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, Julian Schrittwieser, Keith Anderson, Sarah York, Max Cant, Adam Cain, Adrian Bolton, Stephen Gaffney, Helen King, Demis Hassabis, Shane Legg, and Stig Petersen (2016). **DeepMind Lab**. arXiv: 1612.03801 [cs.AI] (pages 26, 89).

Bellemare, Marc, Salvatore Candido, Pablo Samuel Castro, Jun Gong, Marlos C Machado, Subhodeep Moitra, Sameera S Ponda, and Ziyu Wang (2020). “Autonomous navigation of stratospheric balloons using reinforcement learning”. In: **Nature** 588.7836, pp. 77–82 (pages 2, 10).

Bellemare, Marc, Will Dabney, and Rémi Munos (2017). “A Distributional Perspective on Reinforcement Learning”. In: **Proceedings of the International Conference on Machine Learning, (ICML)** (pages 17, 35).

Bellemare, Marc, Yavar Naddaf, Joel Veness, and Michael Bowling (2013). “The arcade learning environment: An evaluation platform for general agents”. In: **Journal of Artificial Intelligence Research** 47, pp. 253–279 (pages 3, 16, 26, 27).

Bellemare, Marc, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Rémi Munos (2016). “Unifying Count-Based Exploration and Intrinsic Motivation”. In: **Proceedings of the Neural Information Processing Systems Conference, (NIPS)** (pages 48, 52, 76).

Bellman, Richard (1957). **Dynamic Programming**. Dover Publications. ISBN: 9780486428093 (page 12).

Bengio, Yoshua, Aaron Courville, and Pascal Vincent (2013). "Representation learning: A review and new perspectives". In: **IEEE transactions on pattern analysis and machine intelligence** 35.8, pp. 1798–1828 (page 30).

Berlyne, Daniel (1965). **Structure and direction in thinking.** John Wiley (page 47).

Berner, Christopher, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. (2019). "Dota 2 with large scale deep reinforcement learning". In: **arXiv preprint arXiv:1912.06680** (pages 2, 26, 34, 89).

Bertsekas, Dimitri P and John N Tsitsiklis (1995). "Neuro-dynamic programming: an overview". In: **Proceedings of the IEEE Conference on Decision and Control** (page 12).

Bishop, Christopher (2006). **Pattern recognition and machine learning**. springer (page 9).

Bourgine, Paul and FJ Varela (1991). "Toward a practice of autonomous systems". In: **Proceedings of the European Conference on Artificial Life, (ECAL)** (page 1).

Boutilier, Craig, Alon Cohen, Avinatan Hassidim, Yishay Mansour, Ofer Meshi, Martin Mladenov, and Dale Schuurmans (2018). "Planning and learning with stochastic action sets". In: **Proceedings of the Internationnal Joint Conference on Artificial Intelligence, (IJCAI)** (page 34).

Boutilier, Craig and Tyler Lu (2016). "Budget allocation using weakly coupled, constrained Markov decision processes". In: **Proceedings of the Conference on Uncertainty in Artificial Intelligence, (UAI)** (page 36).

Bradtke, Steven J and Andrew G Barto (1996). "Linear least-squares algorithms for temporal difference learning". In: **Machine learning** 22.1, pp. 33–57 (page 15).

Brockman, Greg, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba (2016). "Openai gym". In: **arXiv preprint arXiv:1606.01540** (page 25).

Brodeur, Simon, Ethan Perez, Ankesh Anand, Florian Golemo, Luca Celotti, Florian Strub, Jean Rouat, Hugo Larochelle, and Aaron Courville (2017). "HoME: a Household Multimodal Environment". In: **ViGIL Workshop** (page 64).

Brown, Tom, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei (2020). "Language Models are Few-Shot Learners". In: **Proceedings of the Neural Information Processing Systems Conference, (NIPS)** (pages 11, 19, 30).

Burda, Yuri, Harrison Edwards, Deepak Pathak, Amos J. Storkey, Trevor Darrell, and Alexei A. Efros (2019a). "Large-Scale Study of Curiosity-Driven Learning". In: **Proceedings of the International Conference on Learning Representations, (ICLR)** (page 49).

Burda, Yuri, Harrison Edwards, Amos J. Storkey, and Oleg Klimov (2019b). “Exploration by random network distillation”. In: **Proceedings of the International Conference on Learning Representations, (ICLR)** (pages 48, 55, 60).

Campero, Andres, Roberta Raileanu, Heinrich Küttler, Joshua B Tenenbaum, Tim Rocktäschel, and Edward Grefenstette (2020). “Learning with AMIGo: Adversarially Motivated Intrinsic Goals”. In: **Proceedings of the International Conference on Learning Representations, (ICLR)** (pages 50, 51, 54–56).

Carrara, Nicolas, Omar Darwiche Domingues, Yannis Flet-Berliac, Emilie Kaufmann, Edouard Leurent, Odalric-Ambrym Maillard, Pierre Ménard, Philippe Preux, Mathieu Seurin, Xuedong Shang, Julien Seznec, Florian Strub, and Mohammad Sadegh Talebi (2019a). “Reinforcement Learning Summer School, (RLSS)”. In: **Scientific Event Organization** (page 7).

Carrara, Nicolas, Edouard Leurent, Romain Laroche, Tanguy Urvoy, Odalric-Ambrym Maillard, and Olivier Pietquin (2019b). “Budgeted Reinforcement Learning in Continuous State Space”. In: **Advances in Neural Information Processing Systems** 32, pp. 9299–9309 (page 36).

Caspi, Itai, Gal Leibovich, Gal Novik, and Shadi Endrawis (Dec. 2017). **Reinforcement Learning Coach**. DOI: `10.5281/zenodo.1134899`. URL: `https://doi.org/10.5281/zenodo.1134899` (page 25).

Cassandra, Anthony (1998). “Exact and approximate algorithms for partially observable Markov decision processes”. In: (page 18).

Chan, Harris, Yuhuai Wu, Jamie Kiros, Sanja Fidler, and Jimmy Ba (2018). “ACTRCE: Augmenting Experience via Teacher’s Advice For Multi-Goal Reinforcement Learning”. In: **Goal Specifications for RL workshop** (pages 65, 66).

Chandak, Yash, Georgios Theocharous, James Kostas, Scott M. Jordan, and Philip S. Thomas (2019). “Learning Action Representations for Reinforcement Learning”. In: **Proceedings of the International Conference on Machine Learning, (ICML)** (page 31).

Chandramohan, Senthilkumar, Matthieu Geist, Fabrice Lefevre, and Olivier Pietquin (2012). “Behavior Specific User Simulation in Spoken Dialogue Systems”. In: **Speech Communication; 10. ITG Symposium** (page 81).

Chandramohan, Senthilkumar, Matthieu Geist, and Olivier Pietquin (2010). “Optimizing spoken dialogue management with fitted value iteration”. In: **Proceedings of the IEEE International Conference of the Speech Communication Association, (INTERSPEECH)** (pages 27, 30, 36, 81).

Chaplot, Devendra Singh, Kanthashree Mysore Sathyendra, Rama Kumar Pasumarthi, Dheeraj Rajagopal, and Ruslan Salakhutdinov (2018). “Gated-Attention Architectures for Task-Oriented Language Grounding”. In: **Proceedings of the Internationnal Conference on Artificial Intelligence, (AAAI)** (pages 19, 68).

Charpagne, Marie-Agathe, Florian Strub, and Tresa M Pollock (2019). “Accurate reconstruction of EBSD datasets by a multimodal data approach using an evolutionary algorithm”. In: **Materials Characterization** 150, pp. 184–198 (page 10).

Chen, David L. and Raymond J. Mooney (2011). “Learning to Interpret Natural Language Navigation Instructions from Observations”. In: **Proceedings of the Internationnal Conference on Artificial Intelligence, (AAAI)** (page 63).

Chen, Hongshen, Xiaorui Liu, Dawei Yin, and Jiliang Tang (2017). “A survey on dialogue systems: Recent advances and new frontiers”. In: **Acm Sigkdd Explorations Newsletter** 19.2, pp. 25–35 (page 36).

Chen, Yu, Yingfeng Chen, Yu Yang, Ying Li, Jianwei Yin, and Changjie Fan (2019). “Learning Action-Transferable Policy with Action Embedding”. In: **Proceedings of the Internationnal Conference on Artificial Intelligence, (AAAI)** (pages 31, 90).

Chevalier-Boisvert, Maxime, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio (2019). “BabyAI: A Platform to Study the Sample Efficiency of Grounded Language Learning”. In: **Proceedings of the International Conference on Learning Representations, (ICLR)** (pages 25, 27, 40, 51, 53, 64, 66, 68).

Chodroff, Eleanor, Matthew Maciejewski, Jan Trmal, Sanjeev Khudanpur, and John Godfrey (2016). “New release of Mixer-6: Improved validity for phonetic study of speaker variation and identification”. In: **Proceedings of the International Conference on Language Resources and Evaluation, (LREC)** (page 84).

Chou, Po-Wei, Daniel Maturana, and Sebastian Scherer (2017). “Improving Stochastic Policy Gradients in Continuous Control with Deep Reinforcement Learning using the Beta Distribution”. In: **Proceedings of the International Conference on Machine Learning, (ICML)** (page 27).

Chow, Yinlam, Mohammad Ghavamzadeh, Lucas Janson, and Marco Pavone (2017). “Risk-constrained reinforcement learning with percentile risk criteria”. In: **The Journal of Machine Learning Research** 18.1, pp. 6070–6120 (page 35).

Chow, Yinlam, Aviv Tamar, Shie Mannor, and Marco Pavone (2015). “Risk-sensitive and robust decision-making: a CVaR optimization approach”. In: **Proceedings of the Neural Information Processing Systems Conference, (NIPS)** (page 35).

Cideron, Goeffrey, Mathieu Seurin, Florian Strub, and Olivier Pietquin (2020). “HIGhER: Improving instruction following with Hindsight Generation for Experience Replay”. In: **Proceedings of the IEEE Symposium Series on Computational Intelligence, (SSCI)** (pages 50, 63).

Cideron*, Goeffrey, Mathieu Seurin*, Florian Strub, and Olivier Pietquin (2020). “HIGhER: Improving instruction following with Hindsight Generation for Experience Replay”. In: **Proceedings of the IEEE Symposium Series on Computational Intelligence, (SSCI)** (page 7).

Cireşan, Dan, Alessandro Giusti, Luca M Gambardella, and Jürgen Schmidhuber (2013). “Mitosis detection in breast cancer histology images with deep neural networks”. In: **Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention** (page 10).

Cireşan, Dan, Ueli Meier, Jonathan Masci, Luca Maria Gambardella, and Jürgen Schmidhuber (2011). “Flexible, high performance convolutional neural networks for image classification". In: **Proceedings of the Internationnal Joint Conference on Artificial Intelligence, (IJCAI)** (page 11).

Cireşan, Dan, Ueli Meier, and Jürgen Schmidhuber (2012). "Multi-column deep neural networks for image classification". In: **Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, (CVPR)** (page 11).

Co-Reyes, John D., Abhishek Gupta, Suvansh Sanjeev, Nick Altieri, Jacob Andreas, John DeNero, Pieter Abbeel, and Sergey Levine (2019). "Guiding Policies with Language via Meta-Learning". In: **Proceedings of the International Conference on Learning Representations, (ICLR)** (page 67).

Cobbe, Karl, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman (2019). "Quantifying generalization in reinforcement learning". In: **Proceedings of the International Conference on Machine Learning, (ICML)** (pages 22, 90).

Colas, Cédric, Ahmed Akakzia, Pierre-Yves Oudeyer, Mohamed Chetouani, and Olivier Sigaud (2020a). "Language-Conditioned Goal Generation: a New Approach to Language Grounding for RL". In: **arXiv preprint arXiv:2006.07043** (page 77).

Colas, Cédric, Pierre Fournier, Mohamed Chetouani, Olivier Sigaud, and Pierre-Yves Oudeyer (2019). "CURIOUS: intrinsically motivated modular multi-goal reinforcement learning". In: **Proceedings of the International Conference on Machine Learning, (ICML)** (pages 50, 77).

Colas, Cédric, Tristan Karch, Nicolas Lair, Jean-Michel Dussoux, Clément Moulin-Frier, Peter Dominey, and Pierre-Yves Oudeyer (2020b). "Language as a Cognitive Tool to Imagine Goals in Curiosity Driven Exploration". In: **Advances in Neural Information Processing Systems** 33 (page 64).

Colas, Cédric, Tristan Karch, Clément Moulin-Frier, and Pierre-Yves Oudeyer (2021). **Language as a cognitive tool**. URL: https://hal.archives-ouvertes.fr/hal-03159786 (page 63).

Colas, Cédric, Tristan Karch, Olivier Sigaud, and Pierre-Yves Oudeyer (2020c). "Intrinsically Motivated Goal-Conditioned Reinforcement Learning: a Short Survey". In: **arXiv preprint arXiv:2012.09830** (pages 50, 63).

Collet, Timothé and Olivier Pietquin (2015). "Optimism in Active Learning". In: **Computational Intelligence and Neuroscience** 2015, pp. 1–17 (page 80).

Contardo, Gabriella, Ludovic Denoyer, and Thierry Artières (2017). "A Meta-Learning Approach to One-Step Active-Learning". In: **International Workshop on Automatic Selection, Configuration and Composition of Machine Learning Algorithms** (page 80).

Côté, Marc-Alexandre, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, Wendy Tay, and Adam Trischler (2018). "TextWorld: A Learning Environment for Text-based Games". In: **CoRR** abs/1806.11532 (pages 26, 27, 40).

Cui, Hao and Roni Khardon (2016). "Online symbolic gradient-based optimization for factored action MDPs". In: **Proceedings of the Internationnal Joint Conference on Artificial Intelligence, (IJCAI)** (page 33).

Cunningham, John P and Zoubin Ghahramani (2015). “Linear dimensionality reduction: Survey, insights, and generalizations”. In: **The Journal of Machine Learning Research** 16.1, pp. 2859–2900 (page 10).

Dalal, Gal, Krishnamurthy Dvijotham, Matej Vecerik, Todd Hester, Cosmin Paduraru, and Yuval Tassa (2018). “Safe exploration in continuous action spaces”. In: **arXiv preprint arXiv:1801.08757** (page 36).

Dayan, Peter and Geoffrey E Hinton (1993). “Feudal Reinforcement Learning”. In: **Proceedings of the Neural Information Processing Systems Conference, (NIPS)** (page 63).

Dayan, Peter and Terrence J Sejnowski (1994). “TD (λ) converges with probability 1”. In: **Machine Learning** 14.3, pp. 295–301 (page 14).

Dean, Thomas L, Robert Givan, and Kee-Eung Kim (1998). “Solving Stochastic Planning Problems with Large State and Action Spaces.” In: **Proceedings of the Internationnal Conference on Artificial Intelligence Planning Systems, (AIPS)** (page 33).

Doddington, George R, Mark A Przybocki, Alvin F Martin, and Douglas A Reynolds (2000). “The NIST speaker recognition evaluation–overview, methodology, systems, results, perspective”. In: **Speech communication** 31.2-3, pp. 225–254 (page 84).

Doersch, Carl, Abhinav Gupta, and Alexei A Efros (2015). “Unsupervised visual representation learning by context prediction”. In: **Proceedings of IEEE Internationnal Conference on Computer Vision, (ICCV)** (page 10).

Domingues, Omar Darwiche, Yannis Flet-Berliac, Edouard Leurent, Pierre Ménard, Xuedong Shang, and Michal Valko (2021). **rlberry - A Reinforcement Learning Library for Research and Education**. https://github.com/rlberry-py/rlberry (page 25).

Dulac-Arnold, Gabriel, Ludovic Denoyer, Philippe Preux, and Patrick Gallinari (2012). “Fast reinforcement learning with large action sets using error-correcting output codes for MDP factorization”. In: **Proceedings of the European Conference On Machine Learning and Knowledge Discovery in Databases, (ECML-PKDD)** (page 33).

Dulac-Arnold, Gabriel, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin (2015). “Deep reinforcement learning in large discrete action spaces”. In: **arXiv preprint arXiv:1512.07679** (pages 31, 32, 53).

Duminy, Nicolas, Sao Mai Nguyen, Junshuai Zhu, Dominique Duhaut, and Jerome Kerdreux (2021). “Intrinsically motivated open-ended multi-task learning using transfer learning to discover task hierarchy”. In: **Applied Sciences** 11.3, p. 975 (page 50).

Ecoffet, Adrien, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune (2019). “Go-explore: a new approach for hard-exploration problems”. In: **arXiv:1901.10995** (page 49).

– (2021). “First return, then explore”. In: **Nature** 590.7847, pp. 580–586 (page 49).

Ernst, Damien, Pierre Geurts, and Louis Wehenkel (2005). “Tree-based batch mode reinforcement learning”. In: **Journal of Machine Learning Research** 6, pp. 503–556 (page 15).

Erraqabi, Akram, Mingde Zhao, Marlos C. Machado, Yoshua Bengio, Sainbayar Sukhbaatar, Ludovic Denoyer, and Alessandro Lazaric (2021). “Exploration-Driven Representation Learning in Reinforcement Learning”. In: **ICML 2021 Workshop on Unsupervised Reinforcement Learning** (page 48).

Espeholt, Lasse, Raphaël Marinier, Piotr Stanczyk, Ke Wang, and Marcin Michalski (2019). “SEED RL: Scalable and Efficient Deep-RL with Accelerated Central Inference”. In: **Proceedings of the International Conference on Learning Representations, (ICLR)** (pages 20, 89).

Espeholt, Lasse, Hubert Soyer, Rémi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu (2018). “IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures”. In: **Proceedings of the International Conference on Machine Learning, (ICML)** (pages 21, 22, 55, 89).

Fang, Meng, Yuan Li, and Trevor Cohn (2017). “Learning how to Active Learn: A Deep Reinforcement Learning Approach”. In: **Proceedings of the Conference on Empirical Methods in Natural Language Processing, (EMNLP)** (page 80).

Fang, Meng, Cheng Zhou, Bei Shi, Boqing Gong, Jia Xu, and Tong Zhang (2019). “DHER: Hindsight Experience Replay for Dynamic Goals”. In: **Proceedings of the International Conference on Learning Representations, (ICLR)** (page 65).

Fedus, William, Prajit Ramachandran, Rishabh Agarwal, Yoshua Bengio, Hugo Larochelle, Mark Rowland, and Will Dabney (2020). “Revisiting fundamentals of experience replay”. In: **Proceedings of the International Conference on Machine Learning, (ICML)** (page 22).

Ferreira, Emmanuel and Fabrice Lefèvre (2013). “Expert-based reward shaping and exploration scheme for boosting policy learning of dialogue management”. In: **2013 IEEE Workshop on Automatic Speech Recognition and Understanding** (page 30).

Fikes, Richard E, Peter E Hart, and Nils J Nilsson (1972). “Learning and executing generalized robot plans”. In: **Artificial Intelligence** 3, pp. 251–288. ISSN: 0004-3702. DOI: https://doi.org/10.1016/0004-3702(72)90051-3. URL: https://www.sciencedirect.com/science/article/pii/0004370272900513 (page 63).

Fix, Evelyn and Joseph Lawson Hodges (1989). “Discriminatory analysis. Nonparametric discrimination: Consistency properties”. In: **International Statistical Review/Revue Internationale de Statistique** 57.3, pp. 238–247 (page 31).

Flet-Berliac, Yannis (2019). “The Promise of Hierarchical Reinforcement Learning”. In: **The Gradient** (page 63).

Florensa, Carlos, David Held, Xinyang Geng, and Pieter Abbeel (2018). “Automatic Goal Generation for Reinforcement Learning Agents”. In: **Proceedings of the International Conference on Machine Learning, (ICML)** (page 50).

Forestier, Sébastien, Rémy Portelas, Yoan Mollard, and Pierre-Yves Oudeyer (2017). "Intrinsically motivated goal exploration processes with automatic curriculum learning". In: **arXiv:1708.02190** (pages 50, 77).

Fortunato, Meire, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Matteo Hessel, Ian Osband, Alex Graves, Volodymyr Mnih, Rémi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg (2018). "Noisy Networks For Exploration". In: **Proceedings of the International Conference on Learning Representations, (ICLR)** (pages 17, 47).

Fried, Daniel, Ronghang Hu, Volkan Cirik, Anna Rohrbach, Jacob Andreas, Louis-Philippe Morency, Taylor Berg-Kirkpatrick, Kate Saenko, Dan Klein, and Trevor Darrell (2018). "Speaker-Follower Models for Vision-and-Language Navigation". In: **Proceedings of the Neural Information Processing Systems Conference, (NeurIPS)** (page 64).

Fu, Justin, Anoop Korattikara, Sergey Levine, and Sergio Guadarrama (2019). "From Language to Goals: Inverse Reinforcement Learning for Vision-Based Instruction Following". In: **Proceedings of the International Conference on Learning Representations, (ICLR)** (page 68).

Fu, Justin, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine (2020). **D4RL: Datasets for Deep Data-Driven Reinforcement Learning**. arXiv: `2004.07219 [cs.LG]` (page 22).

Fujimoto, Scott, David Meger, and Doina Precup (2019). "Off-policy deep reinforcement learning without exploration". In: **Proceedings of the International Conference on Machine Learning, (ICML)** (page 28).

Gao, Jianfeng, Michel Galley, and Lihong Li (2018). "Neural Approaches to Conversational AI". In: **Proceedings of the International Conference on Research Development in Information Retrieval, (SIGIR)** (page 30).

García, Javier and Fernando Fernándex (2015). "A comprehensive survey on safe reinforcement learning". In: **Journal of Machine Learning Research** 16.4, pp. 1437–1480. ISSN: 1532-4435. URL: `http://www.jmlr.org/papers/volume16/garcia15a/garcia15a.pdf%7B%5C%%7D0Ahttp://dblp.uni-trier.de/db/journals/jmlr/jmlr16.html%7B%5C#%7DGarciaF15` (pages 34, 36).

García, Javier, Rubén Majadas, and Fernando Fernández (2020). "Learning adversarial attack policies through multi-objective reinforcement learning". In: **Engineering Applications of Artificial Intelligence** 96, p. 104021 (page 35).

Garofolo, J., Lori Lamel, W. Fisher, Jonathan Fiscus, D. Pallett, N. Dahlgren, and V. Zue (Nov. 1992). "TIMIT Acoustic-phonetic Continuous Speech Corpus". In: **Linguistic Data Consortium** (page 84).

Geist, Matthieu and Olivier Pietquin (2010). "Kalman temporal differences". In: **Journal of artificial intelligence research** 39, pp. 483–532 (page 47).

Geist, Matthieu, Bruno Scherrer, and Olivier Pietquin (2019). "A Theory of Regularized Markov Decision Processes". In: **Proceedings of the International Conference on Machine Learning, (ICML)** (page 84).

Ghavamzadeh, Mohammad, Alessandro Lazaric, and Matteo Pirotta (2020). **Exploration in Reinforcement Learning**. Tutorial at AAAI'20. URL: https://rlgammazero.github.io/ (page 45).

Gleave, Adam, Michael Dennis, Cody Wild, Neel Kant, Sergey Levine, and Stuart Russell (2019). "Adversarial Policies: Attacking Deep Reinforcement Learning". In: **Proceedings of the International Conference on Learning Representations, (ICLR)** (pages 22, 35).

Godfrey, John J, Edward C Holliman, and Jane McDaniel (1992). "SWITCHBOARD: Telephone speech corpus for research and development". In: **Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, (ICASSP)** (page 84).

Goodfellow, Ian, Yoshua Bengio, Aaron Courville, and Yoshua Bengio (2016). **Deep learning**. Vol. 1. 2. MIT press Cambridge (page 11).

Goodfellow, Ian, Jonathon Shlens, and Christian Szegedy (2015). "Explaining and Harnessing Adversarial Examples". In: **Proceedings of the International Conference on Learning Representations, (ICLR)** (page 35).

Gregor, Karol, Danilo Jimenez Rezende, and Daan Wierstra (2016). "Variational intrinsic control". In: (page 49).

Guestrin, Carlos, Michail Lagoudakis, and Ronald Parr (2002). "Coordinated reinforcement learning". In: **Proceedings of the International Conference on Machine Learning, (ICML)** (page 33).

Gulcehre, Caglar, Ziyu Wang, Alexander Novikov, Tom Le Paine, Sergio Gómez Colmenarejo, Konrad Zolna, Rishabh Agarwal, Josh Merel, Daniel Mankowitz, Cosmin Paduraru, Gabriel Dulac-Arnold, Jerry Li, Mohammad Norouzi, Matt Hoffman, Ofir Nachum, George Tucker, Nicolas Heess, and Nando deFreitas (2020). **RL Unplugged: Benchmarks for Offline Reinforcement Learning**. arXiv: 2006.13888 [cs.LG] (page 22).

Ha, David and Jürgen Schmidhuber (2018). "Recurrent World Models Facilitate Policy Evolution". In: **Proceedings of the Neural Information Processing Systems Conference, (NIPS)** (page 13).

Haarnoja, Tuomas, Haoran Tang, Pieter Abbeel, and Sergey Levine (2017). "Reinforcement learning with deep energy-based policies". In: **Proceedings of the International Conference on Machine Learning, (ICML)** (page 22).

Haber, Nick, Damian Mrowca, Stephanie Wang, Fei-Fei Li, and Daniel L. Yamins (2018). "Learning to Play With Intrinsically-Motivated, Self-Aware Agents". In: **Proceedings of the Neural Information Processing Systems Conference, (NeurIPS)** (page 48).

Hafner, Danijar, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi (2020). "Dream to Control: Learning Behaviors by Latent Imagination". In: **Proceedings of the International Conference on Learning Representations, (ICLR)** (page 13).

Harutyunyan, Anna, Marc Bellemare, Tom Stepleton, and Rémi Munos (2016). “Q(λ) with Off-Policy Corrections”. In: **Proceedings of International Conference on Algorithmic Learning Theory, (ALT)** (page 14).

Hashimoto, Daniel A, Guy Rosman, Daniela Rus, and Ozanan R Meireles (2018). “Artificial intelligence in surgery: promises and perils”. In: **Annals of surgery** 268.1, p. 70 (page 10).

Hasselt, Hado van, Arthur Guez, and David Silver (2016). “Deep Reinforcement Learning with Double Q-Learning”. In: **Proceedings of the Internationnal Conference on Artificial Intelligence, (AAAI)** (pages 41, 69).

Hasselt, Hado van, Matteo Hessel, and John Aslanides (2019). “When to use parametric models in reinforcement learning?” In: **arXiv preprint arXiv:1906.05243** (pages 13, 17).

Hausknecht, Matthew and Peter Stone (2015). “Deep recurrent q-learning for partially observable mdps”. In: **arXiv preprint arXiv:1507.06527** (pages 18, 33).

Haussmann, Manuel, Fred Hamprecht, and Melih Kandemir (2019). “Deep Active Learning with Adaptive Acquisition”. In: **Proceedings of the Internationnal Joint Conference on Artificial Intelligence, (IJCAI)** (page 80).

He, Ji, Jianshu Chen, Xiaodong He, Jianfeng Gao, Lihong Li, Li Deng, and Mari Ostendorf (2016a). “Deep Reinforcement Learning with a Natural Language Action Space”. In: **Proceedings of the Annual Meeting of the Association for Computational Linguistics, (ACL)** (page 31).

He, Ji, Mari Ostendorf, Xiaodong He, Jianshu Chen, Jianfeng Gao, Lihong Li, and Li Deng (2016b). “Deep Reinforcement Learning with a Combinatorial Action Space for Predicting Popular Reddit Threads”. In: **Proceedings of the Conference on Empirical Methods in Natural Language Processing, (EMNLP)** (page 31).

Hermann, Karl Moritz, Felix Hill, Simon Green, Fumin Wang, Ryan Faulkner, Hubert Soyer, David Szepesvari, Wojciech Marian Czarnecki, Max Jaderberg, Denis Teplyashin, et al. (2017). “Grounded language learning in a simulated 3d world”. In: **arXiv:1706.06551** (pages 50, 64).

Hermann, Karl Moritz, Mateusz Malinowski, Piotr Mirowski, Andras Banki-Horvath, Keith Anderson, and Raia Hadsell (2020). “Learning to Follow Directions in Street View”. In: **Proceedings of the Internationnal Conference on Artificial Intelligence, (AAAI)** (page 63).

Hessel, Matteo, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Gheshlaghi Azar, and David Silver (2018). “Rainbow: Combining Improvements in Deep Reinforcement Learning”. In: **Proceedings of the Internationnal Conference on Artificial Intelligence, (AAAI)** (pages 17, 27).

Hester, Todd, Matej Veceriék, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, Gabriel Dulac-Arnold, John P. Agapiou, Joel Z. Leibo, and Audrunas Gruslys (2018). “Deep Q-learning From Demonstrations”. In: **Proceedings of the Internationnal Conference on Artificial Intelligence, (AAAI)** (pages 38, 76).

Hill, Felix, Karl Moritz Hermann, Phil Blunsom, and Stephen Clark (2017). "Understanding grounded language learning agents". In: **arXiv preprint arXiv:1710.09867** (page 68).

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long short-term memory". In: **Neural computation** 9.8, pp. 1735–1780 (pages 11, 41, 85).

Hoffman, Matt, Bobak Shahriari, John Aslanides, Gabriel Barth-Maron, Feryal Behbahani, Tamara Norman, Abbas Abdolmaleki, Albin Cassirer, Fan Yang, Kate Baumli, Sarah Henderson, Alex Novikov, Sergio Gómez Colmenarejo, Serkan Cabi, Caglar Gulcehre, Tom Le Paine, Andrew Cowie, Ziyu Wang, Bilal Piot, and Nando de Freitas (2020). "Acme: A Research Framework for Distributed Reinforcement Learning". In: **arXiv preprint arXiv:2006.00979**. URL: https://arxiv.org/abs/2006.00979 (page 25).

Horgan, Dan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado van Hasselt, and David Silver (2018). "Distributed Prioritized Experience Replay". In: **Proceedings of the International Conference on Learning Representations, (ICLR)** (pages 19, 20).

Houthooft, Rein, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel (2016). "VIME: Variational Information Maximizing Exploration". In: **Proceedings of the Neural Information Processing Systems Conference, (NIPS)** (page 49).

Huang, Shengyi and Santiago Ontanon (2020). "A closer look at invalid action masking in policy gradient algorithms". In: **arXiv preprint arXiv:2006.14171** (pages 32, 89).

Hussenot, Léonard, Robert Dadashi, Matthieu Geist, and Olivier Pietquin (2020a). "Show me the Way: Intrinsic Motivation from Demonstrations". In: **Proceedings of the International Conference on Autonomous Agents and MultiAgent Systems, (AAMAS)** (page 50).

Hussenot, Léonard, Matthieu Geist, and Olivier Pietquin (2020b). "CopyCAT: Taking Control of Neural Policies with Constant Attacks". In: **Proceedings of the International Conference on Autonomous Agents and MultiAgent Systems, (AAMAS)** (pages 3, 22, 35).

Ibarz, Julian, Jie Tan, Chelsea Finn, Mrinal Kalakrishnan, Peter Pastor, and Sergey Levine (2021). "How to train your robot with deep reinforcement learning: lessons we have learned". In: **The International Journal of Robotics Research** 40.4-5, pp. 698–721 (page 28).

Ioffe, Sergey (2006). "Probabilistic linear discriminant analysis". In: **Proceedings of IEEE European Conference on Computer Vision, (ECCV)** (page 86).

Jaccard, Paul (1901). "Distribution de la flore alpine dans le bassin des Dranses et dans quelques regions voisines". In: **Bull Soc Vaudoise Sci Nat** 37, pp. 241–272 (page 87).

Jaksch, Thomas, Ronald Ortner, and Peter Auer (2010). "Near-optimal Regret Bounds for Reinforcement Learning." In: **Journal of Machine Learning Research** 11.4 (page 48).

Jaśkowski, Wojciech, Odd Rune Lykkebø, Nihat Engin Toklu, Florian Trifterer, Zdeněk Buk, Jan Koutnıék, and Faustino Gomez (2018). "Reinforcement Learning to Run…

Fast". In: **The NIPS'17 Competition: Building Intelligent Systems**. Springer, pp. 155–167 (page 28).

Jiang, Yiding, Shixiang Gu, Kevin Murphy, and Chelsea Finn (2019). "Language as an Abstraction for Hierarchical Deep Reinforcement Learning". In: **Proceedings of the Neural Information Processing Systems Conference, (NeurIPS)** (page 67).

Jiang, Yifan, Hyunhak Shin, and Hanseok Ko (2018). "Precise Regression for Bounding Box Correction for Improved Tracking Based on Deep Reinforcement Learning". In: **Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, (ICASSP)** (page 80).

Jin, Chi, Zeyuan Allen-Zhu, Sebastien Bubeck, and Michael I Jordan (2018). "Is Q-learning provably efficient?" In: **Proceedings of the Neural Information Processing Systems Conference, (NIPS)** (page 45).

Jin, Chi, Akshay Krishnamurthy, Max Simchowitz, and Tiancheng Yu (2020). "Reward-free exploration for reinforcement learning". In: **Proceedings of the International Conference on Machine Learning, (ICML)** (page 10).

Jing, Longlong and Yingli Tian (2020). "Self-supervised visual feature learning with deep neural networks: A survey". In: **IEEE Transactions on Pattern Analysis and Machine Intelligence** (page 10).

Jouppi, Norman P, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. (2017). "In-datacenter performance analysis of a tensor processing unit". In: **Proceedings of the International Symposium on Computer Architecture** (page 11).

Kaelbling, Leslie Pack, Michael L Littman, and Anthony R Cassandra (1998). "Planning and acting in partially observable stochastic domains". In: **Artificial intelligence** 101.1-2, pp. 99–134 (page 18).

Kakade, Sham (2001). "A natural policy gradient". In: **Advances in neural information processing systems** 14 (page 15).

Kakade, Sham Machandranath et al. (2003). "On the sample complexity of reinforcement learning". PhD thesis. University of London (page 45).

Kamienny, Pierre-Alexandre, Matteo Pirotta, Alessandro Lazaric, Thibault Lavril, Nicolas Usunier, and Ludovic Denoyer (2020). "Learning adaptive exploration strategies in dynamic environments through informed policy regularization". In: **arXiv preprint arXiv:2005.02934** (page 49).

Kanervisto, Anssi, Christian Scheller, and Ville Hautamäki (2020). "Action space shaping in deep reinforcement learning". In: **Proceedings of the IEEE Conference on Games (CoG)** (pages 3, 29).

Kapturowski, Steven, Georg Ostrovski, John Quan, Remi Munos, and Will Dabney (2018). "Recurrent Experience Replay in Distributed Reinforcement Learning". In: **Proceedings of the International Conference on Learning Representations, (ICLR)** (pages 18–20, 89).

Kearns, Michael and Satinder Singh (2002). "Near-optimal reinforcement learning in polynomial time". In: **Machine learning** 49.2-3, pp. 209–232 (page 47).

Kempka, Michał, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski (2016). "ViZDoom: A Doom-based AI Research Platform for Visual Reinforcement Learning". In: **Proceedings of the IEEE Conference on Computational Intelligence and Games**. The best paper award (pages 26, 27).

Keskar, Nitish Shirish, Jorge Nocedal, Ping Tak Peter Tang, Dheevatsa Mudigere, and Mikhail Smelyanskiy (2017). "On large-batch training for deep learning: Generalization gap and sharp minima". In: **Proceedings of the International Conference on Learning Representations, (ICLR)** (page 17).

Khouzaimi, Hatim, Romain Laroche, and Fabrice Lefèvre (2015). "Optimising Turn-Taking Strategies With Reinforcement Learning". In: **Proceedings of the 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue** (page 81).

Kiela, Douwe, Luana Bulat, Anita L Vero, and Stephen Clark (2016). "Virtual embodiment: A scalable long-term strategy for artificial intelligence research". In: **arXiv preprint arXiv:1610.07432** (page 64).

Kimura, Hajime, Shigenobu Kobayashi, et al. (2000). "An analysis of actor-critic algorithms using eligibility traces: reinforcement learning with imperfect value functions". In: **Journal of Japanese Society for Artificial Intelligence** 15.2, pp. 267–275 (page 16).

Kingma, Diederik P. and Jimmy Ba (2015). "Adam: A Method for Stochastic Optimization". In: **Proceedings of the International Conference on Learning Representations, (ICLR)** (page 85).

Kirby, Simon, Monica Tamariz, Hannah Cornish, and Kenny Smith (2015). "Compression and communication in the cultural evolution of linguistic structure". In: **Cognition** 141, pp. 87–102 (page 63).

Kolter, J. Zico and Andrew Y. Ng (2009). "Near-Bayesian exploration in polynomial time". In: **Proceedings of the International Conference on Machine Learning, (ICML)** (page 48).

Konda, Vijay R and John N Tsitsiklis (2000). "Actor-critic algorithms". In: **Proceedings of the Neural Information Processing Systems Conference, (NIPS)** (page 16).

König, Jonas, Simon Malberg, Martin Martens, Sebastian Niehaus, Artus Krohn-Grimberghe, and Arunselvan Ramaswamy (2019). "Multi-stage reinforcement learning for object detection". In: **Science and Information Conference** (page 80).

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "ImageNet Classification with Deep Convolutional Neural Networks". In: **Proceedings of the Neural Information Processing Systems Conference, (NIPS)** (page 11).

Kulkarni, Tejas D., Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum (2016). "Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation". In: **Proceedings of the Neural Information Processing Systems Conference, (NIPS)** (page 50).

Kurita, Shuhei and Kyunghyun Cho (2020). "Generative Language-Grounded Policy in Vision-and-Language Navigation with Bayes' Rule". In: **Proceedings of the International Conference on Learning Representations, (ICLR)** (page 77).

Küttler, Heinrich, Nantas Nardelli, Thibaut Lavril, et al. (2019). "TorchBeast: A PyTorch Platform for Distributed RL". In: **arXiv:1910.03552**. URL: https://github.com/facebookresearch/torchbeast (page 55).

Küttler, Heinrich, Nantas Nardelli, Alexander H. Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel (2020). "The NetHack Learning Environment". In: **Proceedings of the Neural Information Processing Systems Conference, (NeurIPS)** (page 26).

Lagoudakis, Michail G and Ronald Parr (2003). "Least-squares policy iteration". In: **The Journal of Machine Learning Research** 4, pp. 1107–1149 (page 15).

Lample, Guillaume and François Charton (2019). "Deep Learning For Symbolic Mathematics". In: **Proceedings of the International Conference on Learning Representations, (ICLR)** (page 10).

Lazaric, Alessandro, Marcello Restelli, and Andrea Bonarini (2008). "Reinforcement Learning in Continuous Action Spaces through Sequential Monte Carlo Methods". In: **Proceedings of the Neural Information Processing Systems Conference, (NIPS)** (pages 28, 33).

LeCun, Yann, Yoshua Bengio, et al. (1995). "Convolutional networks for images, speech, and time series". In: **The handbook of brain theory and neural networks** (pages 11, 41).

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). "Deep learning". In: **nature** 521.7553, pp. 436–444 (page 11).

Lesort*, Timothée, Mathieu Seurin*, Xinrui Li, Natalia Díaz-Rodríguez, and David Filliat (2019). "Deep unsupervised state representation learning with robotic priors: a robustness analysis". In: **Proceedings of the International Joint Conference on Neural Networks, (IJCNN)** (page 7).

Leurent, Edouard (2020). "Safe and Efficient Reinforcement Learning for Behavioural Planning in Autonomous Driving". Theses. Université de Lille. URL: https://hal.inria.fr/tel-03035705 (page 10).

Levine, Sergey, Chelsea Finn, Trevor Darrell, and Pieter Abbeel (2016). "End-to-end training of deep visuomotor policies". In: **The Journal of Machine Learning Research** 17.1, pp. 1334–1373 (page 27).

Li, Xiao, Yao Ma, and Calin Belta (2018). "A policy search method for temporal logic specified reinforcement learning tasks". In: **Proceedings of the IEEE Annual American Control Conference (ACC)** (page 65).

Liang, Eric, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica (2018). "RLlib: Abstractions for distributed reinforcement learning". In: **Proceedings of the International Conference on Machine Learning, (ICML)** (page 25).

Lillicrap, Timothy P., Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra (2016). "Continuous control with deep reinforcement learning". In: **Proceedings of the International Conference on Learning Representations, (ICLR)** (page 28).

Lim, Shiau and Peter Auer (2012). "Autonomous exploration for navigating in mdps". In: **Conference on Learning Theory** (page 10).

Lim, Sungsu, Ajin Joseph, Lei Le, Yangchen Pan, and Martha White (2018). "Actor-Expert: A Framework for using Q-learning in Continuous Action Spaces". In: **arXiv preprint arXiv:1810.09103** (page 33).

Lin, Long-Ji (1992). "Self-improving reactive agents based on reinforcement learning, planning and teaching". In: **Machine learning** 8.3-4, pp. 293–321 (page 17).

Linnainmaa, Seppo (1976). "Taylor expansion of the accumulated rounding error". In: **BIT Numerical Mathematics** 16.2, pp. 146–160 (page 11).

Littman, Michael (1994). "Markov games as a framework for multi-agent reinforcement learning". In: **Machine learning proceedings 1994**. Elsevier, pp. 157–163 (page 35).

Liu, Hao, Alexander Trott, Richard Socher, and Caiming Xiong (2019a). "Competitive experience replay". In: **Proceedings of the International Conference on Learning Representations, (ICLR)** (page 65).

Liu, Zimo, Jingya Wang, Shaogang Gong, Huchuan Lu, and Dacheng Tao (2019b). "Deep reinforcement active learning for human-in-the-loop person re-identification". In: **Proceedings of IEEE Internationnal Conference on Computer Vision, (ICCV)** (page 80).

Lopes, Manuel, Tobias Lang, Marc Toussaint, and Pierre-Yves Oudeyer (2012). "Exploration in Model-based Reinforcement Learning by Empirically Estimating Learning Progress". In: **Proceedings of the Neural Information Processing Systems Conference, (NIPS)** (pages 47, 48).

Lu, Jiasen, Dhruv Batra, Devi Parikh, and Stefan Lee (2019). "ViLBERT: Pretraining Task-Agnostic Visiolinguistic Representations for Vision-and-Language Tasks". In: **Proceedings of the Neural Information Processing Systems Conference, (NeurIPS)** (page 87).

Luketina, Jelena, Nantas Nardelli, Gregory Farquhar, Jakob N. Foerster, Jacob Andreas, Edward Grefenstette, Shimon Whiteson, and Tim Rocktäschel (2019). "A Survey of Reinforcement Learning Informed by Natural Language". In: **Proceedings of the Internationnal Joint Conference on Artificial Intelligence, (IJCAI)** (pages 63, 65).

Machado, Marlos C., Marc G. Bellemare, and Michael Bowling (2020). "Count-Based Exploration with the Successor Representation". In: **Proceedings of the Internationnal Conference on Artificial Intelligence, (AAAI)** (page 48).

Maes, Francis (2009). "Learning in Markov decision processes for structured prediction: applications to sequence labeling, tree transformation and learning for search". PhD thesis. Paris 6 (page 80).

Maes, Francis, Ludovic Denoyer, and Patrick Gallinari (2009). “Structured prediction with reinforcement learning”. In: **Machine learning** 77.2, pp. 271–301 (page 80).

Mandler, George (2002). “Origins of the cognitive (r) evolution”. In: **Journal of the History of the Behavioral Sciences** 38.4, pp. 339–353 (page 1).

Mao, Hongzi, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula (2016). “Resource management with deep reinforcement learning”. In: **Proc. of ACM Workshop on Hot Topics in Networks** (page 10).

Markowitz, Harry (1959). **Portfolio selection** (page 35).

Martin, Jarryd, Suraj Narayanan Sasikumar, Tom Everitt, and Marcus Hutter (2017). “Count-Based Exploration in Feature Space for Reinforcement Learning”. In: **Proceedings of the Internationnal Joint Conference on Artificial Intelligence, (IJCAI)** (page 48).

Masson, Warwick, Pravesh Ranchod, and George Konidaris (2016). “Reinforcement learning with parameterized actions”. In: **Proceedings of the Internationnal Conference on Artificial Intelligence, (AAAI)**. 1 (page 33).

Matteo, Pirotta, Ronan Fruit, Florian Strub, and Mathieu Seurin (2018). “European Workshop on Reinforcement Learning, (EWRL)”. In: **Scientific Event Organization** (page 7).

McCarthy, John, Marvin L Minsky, Nathaniel Rochester, and Claude E Shannon (1955). “A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955”. In: **AI magazine** 27.4, pp. 12–12 (page 1).

Metz, Luke, Julian Ibarz, Navdeep Jaitly, and James Davidson (2017). “Discrete sequential prediction of continuous actions for deep rl”. In: **arXiv preprint arXiv:1705.05035** (page 33).

Meuleau, Nicolas, Milos Hauskrecht, Kee-Eung Kim, Leonid Peshkin, Leslie Pack Kaelbling, Thomas L Dean, and Craig Boutilier (1998). “Solving very large weakly coupled Markov decision processes”. In: **Proceedings of the Internationnal Conference on Artificial Intelligence, (AAAI)** (page 33).

Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean (2013). “Efficient estimation of word representations in vector space”. In: **arXiv preprint arXiv:1301.3781** (pages 30, 31).

Minut, Silviu and Sridhar Mahadevan (2001). “A reinforcement learning model of selective visual attention”. In: **Proceedings of the International Conference on Autonomous Agents and MultiAgent Systems, (AAMAS)** (page 80).

Mirhoseini, Azalia, Anna Goldie, Mustafa Yazgan, Joe Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Sungmin Bae, et al. (2020). “Chip placement with deep reinforcement learning”. In: **arXiv preprint arXiv:2004.10746** (page 10).

Misra, Dipendra, Andrew Bennett, Valts Blukis, Eyvind Niklasson, Max Shatkhin, and Yoav Artzi (2018). “Mapping Instructions to Actions in 3D Environments with Visual Goal Prediction”. In: **Proceedings of the Conference on Empirical Methods in Natural Language Processing, (EMNLP)** (page 64).

Mnih, Volodymyr, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu (2016). "Asynchronous Methods for Deep Reinforcement Learning". In: **Proceedings of the International Conference on Machine Learning, (ICML)** (pages 20, 28, 84).

Mnih, Volodymyr, Nicolas Heess, Alex Graves, and Koray Kavukcuoglu (2014). "Recurrent models of visual attention". In: **Proceedings of the Neural Information Processing Systems Conference, (NIPS)** (page 80).

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. (2015). "Human-level control through deep reinforcement learning". In: **Nature** 518.7540, p. 529 (pages 2, 16, 18, 19, 27, 37, 38, 40, 47, 65, 69, 89).

Moffaert, Kristof Van and Ann Nowé (2014). "Multi-Objective Reinforcement Learning using Sets of Pareto Dominating Policies". In: **Journal of Machine Learning Research** 15.107, pp. 3663–3692. URL: http://jmlr.org/papers/v15/vanmoffaert14a.html (page 64).

Mohamed, Shakir and Danilo Jimenez Rezende (2015). "Variational Information Maximisation for Intrinsically Motivated Reinforcement Learning". In: **Proceedings of the Neural Information Processing Systems Conference, (NIPS)** (page 49).

Moriarty, David, Alan C Schultz, and John J Grefenstette (1999). "Evolutionary algorithms for reinforcement learning". In: **Journal of Artificial Intelligence Research** 11, pp. 241–276 (page 15).

Morimoto, Jun and Kenji Doya (2000). "Robust reinforcement learning". In: **Proceedings of the Neural Information Processing Systems Conference, (NIPS)** (page 35).

Mozer, Michael C and Jonathan Bachrach (1990). "Discovering the structure of a reactive environment by exploration". In: **Proceedings of the Neural Information Processing Systems Conference, (NIPS)** (page 45).

Munos, Rémi, Thomas Stepleton, Anna Harutyunyan, and Marc Bellemare (2016). "Safe and efficient off-policy reinforcement learning". In: **Proceedings of the Neural Information Processing Systems Conference, (NIPS)** (pages 20, 22).

Nair, Arun, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, et al. (2015). "Massively parallel methods for deep reinforcement learning". In: **arXiv preprint arXiv:1507.04296** (page 19).

Nair, Vinod and Geoffrey E. Hinton (2010). "Rectified Linear Units Improve Restricted Boltzmann Machines". In: **Proceedings of the International Conference on Machine Learning, (ICML)** (page 85).

Narasimhan, Karthik, Regina Barzilay, and Tommi Jaakkola (2018). "Grounding language for transfer in deep reinforcement learning". In: **JAIR** 63, pp. 849–874 (page 63).

Neunert, Michael, Abbas Abdolmaleki, Markus Wulfmeier, Thomas Lampe, Tobias Springenberg, Roland Hafner, Francesco Romano, Jonas Buchli, Nicolas Heess, and Martin Riedmiller (2020). "Continuous-discrete reinforcement learning for hybrid control in robotics". In: **Proceedings of the Conference on Robot Learning, (CORL)** (page 33).

Ng, Andrew Y, Daishi Harada, and Stuart Russell (1999). "Policy invariance under reward transformations: Theory and application to reward shaping". In: **Proceedings of the International Conference on Machine Learning, (ICML)** (page 73).

Ngiam, Jiquan, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Y Ng (2011). "Multimodal deep learning". In: **Proceedings of the International Conference on Machine Learning, (ICML)** (page 11).

Nguyen, Sao Mai, Nicolas Duminy, Alexandre Manoury, Dominique Duhaut, and Cedric Buche (2021). "Robots Learn Increasingly Complex Tasks with Intrinsic Motivation and Automatic Curriculum Learning". In: **KI-Künstliche Intelligenz** 35.1, pp. 81–90 (page 50).

Nguyen, Sao Mai and Pierre-Yves Oudeyer (2012). "Active choice of teachers, learning strategies and goals for a socially guided intrinsic motivation learner". In: **Paladyn** 3.3, pp. 136–146 (page 50).

Nowé, Ann, Peter Vrancx, and Yann-Michaël De Hauwere (2012). "Game theory and multi-agent reinforcement learning". In: **Reinforcement Learning**. Springer, pp. 441–470 (page 19).

O'Donoghue, Brendan, Ian Osband, Rémi Munos, and Volodymyr Mnih (2018). "The Uncertainty Bellman Equation and Exploration". In: **Proceedings of the International Conference on Machine Learning, (ICML)** (page 47).

Orseau, Laurent and Stuart Armstrong (2016). "Safely Interruptible Agents". In: **Proceedings of the Conference on Uncertainty in Artificial Intelligence, (UAI)** (page 37).

Osband, Ian, John Aslanides, and Albin Cassirer (2018). "Randomized Prior Functions for Deep Reinforcement Learning". In: **Proceedings of the Neural Information Processing Systems Conference, (NeurIPS)** (page 47).

Osband, Ian, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy (2016). "Deep Exploration via Bootstrapped DQN". In: **Proceedings of the Neural Information Processing Systems Conference, (NIPS)** (page 47).

Ostrovski, Georg, Marc Bellemare, Aäron van den Oord, and Rémi Munos (2017). "Count-Based Exploration with Neural Density Models". In: **Proceedings of the International Conference on Machine Learning, (ICML)** (pages 48, 53).

Oudeyer, Pierre-Yves (2018). "Computational theories of curiosity-driven learning". In: **arXiv:1802.10546** (page 48).

Oudeyer, Pierre-Yves, Frdric Kaplan, and Verena V Hafner (2007). "Intrinsic motivation systems for autonomous mental development". In: **IEEE on Evolutionary Computation** 11.2, pp. 265–286 (pages 47–49).

Oudeyer, Pierre-Yves, Frederic Kaplan, et al. (2008). "How can we define intrinsic motivation". In: **Proceedings of the International Conference on Epigenetic Robotics (EpiRob)** (page 47).

Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu (2002). “Bleu: a Method for Automatic Evaluation of Machine Translation”. In: **Proceedings of the Annual Meeting of the Association for Computational Linguistics, (ACL)** (page 75).

Parisotto, Emilio, Francis Song, Jack Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant Jayakumar, Max Jaderberg, Raphael Lopez Kaufman, Aidan Clark, Seb Noury, et al. (2020). “Stabilizing transformers for reinforcement learning”. In: **Proceedings of the International Conference on Machine Learning, (ICML)** (page 19).

Pascanu, Razvan, Tomás Mikolov, and Yoshua Bengio (2013). “On the difficulty of training recurrent neural networks”. In: **Proceedings of the International Conference on Machine Learning, (ICML)** (page 85).

Patel, Jigar, Sahil Shah, Priyank Thakkar, and Ketan Kotecha (2015). “Predicting stock market index using fusion of machine learning techniques”. In: **Expert Systems with Applications** 42.4, pp. 2162–2172 (page 10).

Pathak, Deepak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell (2017). “Curiosity-driven Exploration by Self-supervised Prediction”. In: **Proceedings of the International Conference on Machine Learning, (ICML)** (pages 49, 55).

Pazis, Jason and Ronald Parr (2011). “Generalized value functions for large action sets”. In: **Proceedings of the International Conference on Machine Learning, (ICML)** (page 33).

Pei, Kexin, Yinzhi Cao, Junfeng Yang, and Suman Jana (2017). “Deepxplore: Automated whitebox testing of deep learning systems”. In: **Proceedings of the Symposium on Operating Systems Principles (SOSP)** (page 35).

Pennington, Jeffrey, Richard Socher, and Christopher D. Manning (2014). “GloVe: Global Vectors for Word Representation”. In: **Proceedings of the Conference on Empirical Methods in Natural Language Processing, (EMNLP)** (page 30).

Perolat, Julien, Bruno Scherrer, Bilal Piot, and Olivier Pietquin (2015). “Approximate Dynamic Programming for Two-Player Zero-Sum Markov Games”. In: **Proceedings of the International Conference on Machine Learning, (ICML)** (page 35).

Peshkin, Leonid, Nicolas Meuleau, and Leslie Pack Kaelbling (1999). “Learning Policies with External Memory”. In: **Proceedings of the International Conference on Machine Learning, (ICML)** (page 33).

Pham, Tu-Hoa, Giovanni De Magistris, and Ryuki Tachibana (2018). “Optlayer-practical constrained optimization for deep reinforcement learning in the real world”. In: **Proceedings of the Conference on Robotics and Automation, (ICRA)** (page 36).

Phelps, Nathan (2020). “Reinforcement learning in large, structured action spaces: A simulation study of decision support for spinal cord injury rehabilitation”. In: (page 27).

Pierrot, Thomas, Valentin Macé, Jean-Baptiste Sevestre, Louis Monier, Alexandre Laterre, Nicolas Perrin, Karim Beguir, and Olivier Sigaud (2021). **Factored Action Spaces in Deep Reinforcement Learning**. URL: `https://openreview.net/forum?id=naSAkn2Xo46` (page 33).

Pierrot, Thomas, Nicolas Perrin, and Olivier Sigaud (2018). “First-order and second-order variants of the gradient descent in a unified framework”. In: **arXiv preprint arXiv:1810.08102** (page 15).

Pietquin, Olivier and Richard Beaufort (2005). “Comparing ASR modeling methods for spoken dialogue simulation and optimal strategy learning”. In: **Proceedings of the IEEE International Conference of the Speech Communication Association, (INTERSPEECH)** (page 81).

Pinto, Lerrel, James Davidson, Rahul Sukthankar, and Abhinav Gupta (2017). “Robust adversarial reinforcement learning”. In: **Proceedings of the International Conference on Machine Learning, (ICML)** (page 35).

Piot, Bilal, Matthieu Geist, and Olivier Pietquin (2014). “Boosted Bellman residual minimization handling expert demonstrations”. In: **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)** 8725 LNAI.PART 2, pp. 549–564. ISSN: 16113349. DOI: 10.1007/978-3-662-44851-9_35. URL: http://www.cristal.univ-lille.fr/%7B~%7Dpietquin/pdf/ECML%7B%5C_%7D2014%7B%5C_%7DOPMGBP.pdf (page 38).

– (2017). “Bridging the Gap Between Imitation Learning and Inverse Reinforcement Learning”. In: **IEEE Transactions on Neural Networks and Learning Systems** 28.8, pp. 1814–1826. DOI: 10.1109/TNNLS.2016.2543000. URL: https://hal.archives-ouvertes.fr/hal-01629654 (page 29).

Plappert, Matthias, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y. Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz (2018). “Parameter Space Noise for Exploration”. In: **Proceedings of the International Conference on Learning Representations, (ICLR)** (page 47).

Pohlen, Tobias, Bilal Piot, Todd Hester, Mohammad Gheshlaghi Azar, Dan Horgan, David Budden, Gabriel Barth-Maron, Hado Van Hasselt, John Quan, Mel Večerıék, et al. (2018). “Observe and look further: Achieving consistent performance on atari”. In: **arXiv preprint arXiv:1805.11593** (page 38).

Povey, Daniel, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, Jan Silovsky, Georg Stemmer, and Karel Vesely (2011). “The Kaldi Speech Recognition Toolkit”. In: **IEEE 2011 Workshop on Automatic Speech Recognition and Understanding (ASRU)** (page 84).

Pritz, Paul J, Liang Ma, and Kin K Leung (2020). “Joint State-Action Embedding for Efficient Reinforcement Learning”. In: **arXiv preprint arXiv:2010.04444** (page 31).

Puterman, Martin (2014). **Markov Decision Processes.: Discrete Stochastic Dynamic Programming**. John Wiley & Sons (pages 12, 37, 81).

Rai, Pradeep and Shubha Singh (2010). “A survey of clustering techniques”. In: **International Journal of Computer Applications** 7.12, pp. 1–5 (page 10).

Raileanu, Roberta and Tim Rocktäschel (2020). “RIDE: Rewarding Impact-Driven Exploration for Procedurally-Generated Environments”. In: **Proceedings of the International Conference on Learning Representations, (ICLR)** (pages 49, 51, 53–55).

Rajeswaran, Aravind, Sarvjeet Ghotra, Balaraman Ravindran, and Sergey Levine (2016). “Epopt: Learning robust neural network policies using model ensembles”. In: **Proceedings of the International Conference on Learning Representations, (ICLR)** (page 35).

Rajkomar, Alvin, Jeffrey Dean, and Isaac Kohane (2019). “Machine learning in medicine”. In: **New England Journal of Medicine** 380.14, pp. 1347–1358 (page 10).

Rao, Yongming, Jiwen Lu, and Jie Zhou (2017). “Attention-aware deep reinforcement learning for video face recognition”. In: **Proceedings of IEEE Internationnal Conference on Computer Vision, (ICCV)** (page 80).

Rauber, Paulo, Avinash Ummadisingu, Filipe Mutz, and Jürgen Schmidhuber (2019). “Hindsight policy gradients”. In: **Proceedings of the International Conference on Learning Representations, (ICLR)** (page 65).

Reiter, Ehud and Robert Dale (1997). “Building applied natural language generation systems”. In: **Natural Language Engineering** 3.1, pp. 57–87 (page 36).

Ren, Pengzhen, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang (2020). “A Survey of Deep Active Learning”. In: **arXiv preprint arXiv:2009.00236** (page 80).

Riedmiller, Martin (2005). “Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method”. In: **Proceedings of the European Conference On Machine Learning, (ECML)** (page 15).

Rockafellar, R Tyrrell, Stanislav Uryasev, et al. (2000). “Optimization of conditional value-at-risk”. In: **Journal of risk** 2, pp. 21–42 (page 35).

Rolnick, David, Priya L Donti, Lynn H Kaack, Kelly Kochanski, Alexandre Lacoste, Kris Sankaran, Andrew Slavin Ross, Nikola Milojevic-Dupont, Natasha Jaques, Anna Waldman-Brown, et al. (2019). “Tackling climate change with machine learning”. In: **arXiv preprint arXiv:1906.05433** (page 10).

Ronneberger, Olaf, Philipp Fischer, and Thomas Brox (2015). “U-net: Convolutional networks for biomedical image segmentation”. In: **Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention** (page 10).

Rubinstein, Reuven Y and Dirk P Kroese (1981). **Simulation and the Monte Carlo method**. Vol. 10. John Wiley & Sons (page 21).

Ruder, Sebastian (2017). “An overview of multi-task learning in deep neural networks”. In: **arXiv preprint arXiv:1706.05098** (page 11).

Rumelhart, David, Geoffrey E Hinton, and Ronald J Williams (1986). “Learning representations by back-propagating errors”. In: **nature** 323.6088, pp. 533–536 (page 11).

Rummery, Gavin A and Mahesan Niranjan (1994). **On-line Q-learning using connectionist systems**. Vol. 37. University of Cambridge, Department of Engineering Cambridge, UK (page 14).

Russakovsky, Olga, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. (2015). “Imagenet large scale visual recognition challenge”. In: **International journal of computer vision** 115.3, pp. 211–252 (page 11).

Ryu, Moonkyung, Yinlam Chow, Ross Anderson, Christian Tjandraatmadja, and Craig Boutilier (2019). “CAQL: Continuous Action Q-Learning”. In: **Proceedings of the International Conference on Learning Representations, (ICLR)** (page 33).

Sahni, Himanshu, Toby Buckley, Pieter Abbeel, and Ilya Kuzovkin (2019). “Visual Hindsight Experience Replay”. In: **Proceedings of the Neural Information Processing Systems Conference, (NeurIPS)** (page 65).

Sallans, Brian and Geoffrey E Hinton (2004). “Reinforcement learning with factored states and actions”. In: **The Journal of Machine Learning Research** 5, pp. 1063–1088 (page 33).

Samuel, Arthur (1959). “Some studies in machine learning using the game of checkers”. In: **IBM Journal of research and development** 3.3, pp. 210–229 (page 1).

– (1988). “Some studies in machine learning using the game of checkers. II—recent progress”. In: **Computer Games I**, pp. 366–400 (page 15).

Saquib, Zia, Nirmala Salam, Rekha P Nair, Nipun Pandey, and Akanksha Joshi (2010). “A survey on automatic speaker recognition systems”. In: **Signal Processing and Multimedia**, pp. 134–145 (page 80).

Sato, Mitsuo, Kenichi Abe, and Hiroshi Takeda (1988). “Learning control of finite Markov chains with an explicit trade-off between estimation and control”. In: **IEEE transactions on systems, man, and cybernetics** 18.5, pp. 677–684 (pages 45, 47).

Savinov, Nikolay, Anton Raichuk, Damien Vincent, Raphaël Marinier, Marc Pollefeys, Timothy P. Lillicrap, and Sylvain Gelly (2019). “Episodic Curiosity through Reachability”. In: **Proceedings of the International Conference on Learning Representations, (ICLR)** (page 49).

Schaul, Tom, Daniel Horgan, Karol Gregor, and David Silver (2015). “Universal Value Function Approximators”. In: **Proceedings of the International Conference on Machine Learning, (ICML)** (pages 49, 63, 64).

Schaul, Tom, John Quan, Ioannis Antonoglou, and David Silver (2016). “Prioritized Experience Replay”. In: **Proceedings of the International Conference on Learning Representations, (ICLR)** (pages 17, 69).

Schmidhuber, Jürgen (1991). “A possibility for implementing curiosity and boredom in model-building neural controllers”. In: **Proceedings of the International Conference on Simulation of Adaptive Aehavior: From Animals to Animats** (pages 45, 47, 49).

– (2018). “One big net for everything”. In: **arXiv preprint arXiv:1802.08864** (page 11).

Schrittwieser, Julian, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver (2020). “Mastering Atari, Go, chess and shogi by planning with a learned model”. In: **Nature** 7839, pp. 604–609 (page 13).

Schulman, John, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz (2015). “Trust region policy optimization”. In: **Proceedings of the International Conference on Machine Learning, (ICML)** (pages 21, 28).

Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov (2017). “Proximal policy optimization algorithms”. In: **arXiv preprint arXiv:1707.06347** (pages 21, 28, 84, 85).

Senior, Andrew W, Richard Evans, John Jumper, James Kirkpatrick, Laurent Sifre, Tim Green, Chongli Qin, Augustin Žídek, Alexander W R Nelson, Alex Bridgland, Hugo Penedones, Stig Petersen, Karen Simonyan, Steve Crossan, Pushmeet Kohli, David T Jones, David Silver, Koray Kavukcuoglu, and Demis Hassabis (2020). “Improved protein structure prediction using potentials from deep learning”. In: **Nature** 577.7792, pp. 706–710. ISSN: 1476-4687. DOI: 10.1038/s41586-019-1923-7. URL: https://doi.org/10.1038/s41586-019-1923-7 (page 10).

Seurin, Mathieu, Philippe Preux, and Olivier Pietquin (2020a). “"I'm Sorry Dave, I'm Afraid I Can't Do That" Deep Q-Learning from Forbidden Actions”. In: **Proceedings of the International Joint Conference on Neural Networks, (IJCNN)** (pages 7, 34).

Seurin, Mathieu, Florian Strub, Philippe Preux, and Olivier Pietquin (2020b). “A Machine of Few Words: Interactive Speaker Recognition with Reinforcement Learning”. In: **Proceedings of the IEEE International Conference of the Speech Communication Association, (INTERSPEECH)** (pages 7, 79).

– (2021). “Don't Do What Doesn't Matter: Intrinsic Motivation with Action Usefulness”. In: **Proceedings of the Internationnal Joint Conference on Artificial Intelligence, (IJCAI)** (page 7).

Sharma, Sahil, Aravind Suresh, Rahul Ramesh, and Balaraman Ravindran (2017). “Learning to factor policies and action-value functions: Factored action space representations for deep reinforcement learning”. In: **arXiv preprint arXiv:1705.07269** (pages 27, 33).

Shoham, Yoav and Kevin Leyton-Brown (2008). **Multiagent systems: Algorithmic, game-theoretic, and logical foundations**. Cambridge University Press (page 19).

Sigaud, Olivier and Olivier Buffet (2008). **Processus décisionnels de Markov en intelligence artificielle** (page 12).

Silva, Samuel Henrique and Peyman Najafirad (2020). “Opportunities and challenges in deep learning adversarial robustness: A survey”. In: **arXiv preprint arXiv:2007.00753** (page 35).

Silver, David, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. (2016). “Mastering the game of Go with deep neural networks and tree search”. In: **nature** 529.7587, pp. 484–489 (pages 2, 89).

Silver, David, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. (2018). “A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play”. In: **Science** 362.6419, pp. 1140–1144 (page 2).

Silver, David, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. (2017). "Mastering the game of go without human knowledge". In: **nature** 550.7676, pp. 354–359 (pages 2, 19).

Simão, Thiago D, Nils Jansen, and Matthijs TJ Spaan (2021). "AlwaysSafe: Reinforcement learning without safety constraint violations during training". In: **Proceedings of the International Conference on Autonomous Agents and MultiAgent Systems, (AAMAS)** (page 36).

Simsek, Özgür and Andrew G. Barto (2006). "An intrinsic reward mechanism for efficient exploration". In: **Proceedings of the International Conference on Machine Learning, (ICML)** (page 47).

Sinclair, Sean R, Siddhartha Banerjee, and Christina Lee Yu (2019). "Adaptive discretization for episodic reinforcement learning in metric spaces". In: **Proceedings of the ACM on Measurement and Analysis of Computing Systems** 3.3, pp. 1–44 (page 29).

Sinclair, Sean, Tianyu Wang, Gauri Jain, Siddhartha Banerjee, and Christina Yu (2020). "Adaptive Discretization for Model-Based Reinforcement Learning". In: **Advances in Neural Information Processing Systems** 33 (page 29).

Singh, Rahul, Qinsheng Zhang, and Yongxin Chen (2020). "Improving Robustness via Risk Averse Distributional Reinforcement Learning". In: **Proceedings of the Conference on Learning for Dynamics and Control, (L4DC)** (page 35).

Singh, Satinder, Andrew Barto, and Nuttapong Chentanez (2004). "Intrinsically Motivated Reinforcement Learning". In: **Proceedings of the Neural Information Processing Systems Conference, (NIPS)** (page 47).

Snyder, David, Daniel Garcia-Romero, Daniel Povey, and Sanjeev Khudanpur (2017). "Deep Neural Network Embeddings for Text-Independent Speaker Verification." In: **Proceedings of the IEEE International Conference of the Speech Communication Association, (INTERSPEECH)** (page 84).

Snyder, David, Daniel Garcia-Romero, Gregory Sell, Daniel Povey, and Sanjeev Khudanpur (2018). "X-Vectors: Robust DNN Embeddings for Speaker Recognition". In: **Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, (ICASSP)** (pages 83, 84).

Snyder, David, Pegah Ghahremani, Daniel Povey, Daniel Garcia-Romero, Yishay Carmiel, and Sanjeev Khudanpur (2016). "Deep neural network-based speaker embeddings for end-to-end speaker verification". In: **Prof. of IEEE Spoken Language Technology Workshop (SLT)** (page 84).

Song, Hyungseok, Hyeryung Jang, Hai H Tran, Se-eun Yoon, Kyunghwan Son, Donggyu Yun, Hyoju Chung, and Yung Yi (2019). "Solving Continual Combinatorial Selection via Deep Reinforcement Learning". In: **Proceedings of the Internationnal Joint Conference on Artificial Intelligence, (IJCAI)** (page 33).

Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov (2014). "Dropout: a simple way to prevent neural networks from overfitting". In: **The journal of machine learning research** 15.1, pp. 1929–1958 (page 85).

Stadie, Bradly C, Sergey Levine, and Pieter Abbeel (2015). "Incentivizing exploration in reinforcement learning with deep predictive models". In: **Workshop on Deep Reinforcement Learning (NIPS)** (page 48).

Strehl, Alexander L and Michael L Littman (2008). "An analysis of model-based interval estimation for Markov decision processes". In: **Journal of Computer and System Sciences** 74.8, pp. 1309–1331 (pages 45, 48, 52, 55).

Strub, Florian, Mathieu Seurin, Ethan Perez, Harm De Vries, Jérémie Mary, Philippe Preux, Aaron Courville, and Olivier Pietquin (2018). "Visual reasoning with multi-hop feature modulation". In: **Proceedings of IEEE European Conference on Computer Vision, (ECCV)** (page 7).

Strub, Florian, Harm de Vries, Jérémie Mary, Bilal Piot, Aaron C. Courville, and Olivier Pietquin (2017). "End-to-end optimization of goal-driven and visually grounded dialogue systems". In: **Proceedings of the Internationnal Joint Conference on Artificial Intelligence, (IJCAI)** (page 80).

Sukhbaatar, Sainbayar, Zeming Lin, Ilya Kostrikov, Gabriel Synnaeve, Arthur Szlam, and Rob Fergus (2018). "Intrinsic Motivation and Automatic Curricula via Asymmetric Self-Play". In: **Proceedings of the International Conference on Learning Representations, (ICLR)** (page 50).

Sunehag, Peter, Richard Evans, Gabriel Dulac-Arnold, Yori Zwols, Daniel Visentin, and Ben Coppin (2015). "Deep reinforcement learning with attention for slate markov decision processes with high-dimensional states and actions". In: **arXiv preprint arXiv:1512.01124** (page 34).

Sutton, R., D. McAllester, S. Singh, Y. Mansour, et al. (1999a). "Policy gradient methods for reinforcement learning with function approximation." In: **Proceedings of the Neural Information Processing Systems Conference, (NIPS)** (page 15).

Sutton, Richard (2019). **The Bitter Lesson**. http://www.incompleteideas.net/IncIdeas/BitterLesson.html (page 22).

Sutton, Richard and Andrew Barto (2018). **Reinforcement learning: An introduction** (pages 2, 10, 12, 14, 17, 21).

Sutton, Richard, Doina Precup, and Satinder Singh (1999b). "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning". In: **Artificial intelligence** 112.1-2, pp. 181–211 (page 63).

Tamar, Aviv, Yinlam Chow, Mohammad Ghavamzadeh, and Shie Mannor (2016). "Sequential decision making with coherent risk". In: **IEEE Transactions on Automatic Control** 62.7, pp. 3323–3338 (page 35).

Tan, Hao and Mohit Bansal (2019). "LXMERT: Learning Cross-Modality Encoder Representations from Transformers". In: **Proceedings of the Conference on Empirical Methods in Natural Language Processing, (EMNLP)** (page 87).

Tang, Haoran, Rein Houthooft, Davis Foote, Adam Stooke, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel (2017). "#Exploration: A Study of Count-Based Exploration for Deep Reinforcement Learning". In: **Proceedings of the Neural Information Processing Systems Conference, (NIPS)** (page 48).

Tang, Yichuan Charlie, Jian Zhang, and Ruslan Salakhutdinov (2020a). “Worst Cases Policy Gradients”. In: **Proceedings of the Conference on Robot Learning, (CORL)** (page 35).

Tang, Yunhao and Shipra Agrawal (2020b). “Discretizing Continuous Action Space for On-Policy Optimization”. In: **Proceedings of the Internationnal Conference on Artificial Intelligence, (AAAI)**. 04 (pages 28, 51).

Tavakoli, Arash, Fabio Pardo, and Petar Kormushev (2018). “Action branching architectures for deep reinforcement learning”. In: **Proceedings of the Internationnal Conference on Artificial Intelligence, (AAAI)**. 1 (pages 28, 33).

Tellex, Stefanie, Thomas Kollar, Steven Dickerson, Matthew R. Walter, Ashis Gopal Banerjee, Seth J. Teller, and Nicholas Roy (2011). “Understanding Natural Language Commands for Robotic Navigation and Mobile Manipulation”. In: **Proceedings of the Internationnal Conference on Artificial Intelligence, (AAAI)** (page 63).

Tennenholtz, Guy and Shie Mannor (2019). “The Natural Language of Actions”. In: **Proceedings of the International Conference on Machine Learning, (ICML)** (pages 31, 32).

Tesauro, G (1995). “Temporal difference learning and TD-Gammon”. In: **Communications of the ACM** 38.3, pp. 58–68 (page 15).

Tessler, Chen, Tom Zahavy, Deborah Cohen, Daniel J Mankowitz, and Shie Mannor (2019). “Action assembly: Sparse imitation learning for text based games with combinatorial action spaces”. In: **arXiv preprint arXiv:1905.09700** (page 31).

Thrun, Sebastian (1992). **Efficient exploration in reinforcement learning**. CMU-CS-92-102 (pages 45, 47).

Thrun, Sebastian and Anton Schwartz (1993). “Issues in using function approximation for reinforcement learning”. In: **Proceedings of the Fourth Connectionist Models Summer School** (page 29).

Todorov, Emanuel, Tom Erez, and Yuval Tassa (2012). “Mujoco: A physics engine for model-based control”. In: **Proceedings of the International Conference on Intelligent Robots and Systems, (IROS)** (pages 26, 27).

Tong, Simon (2001). **Active learning: theory and applications**. Vol. 1. Stanford University USA (page 80).

Topiwala, Anirudh, Pranav Inani, and Abhishek Kathpal (2018). “Frontier based exploration for autonomous robot”. In: **arXiv preprint arXiv:1806.03581** (page 49).

Tsitsiklis, John (1994). “Asynchronous stochastic approximation and Q-learning”. In: **Machine learning** 16.3, pp. 185–202 (page 14).

Tsitsiklis, John N and Benjamin Van Roy (1997). “An analysis of temporal-difference learning with function approximation”. In: **IEEE transactions on automatic control** 42.5, pp. 674–690 (page 15).

Tucker, M. and R. Ellis (1998). “On the relations between seen objects and components of potential actions.” In: **Journal of experimental psychology. Human perception and performance** 24 3, pp. 830–46 (page 3).

Van de Wiele, Tom, David Warde-Farley, Andriy Mnih, and Volodymyr Mnih (2020). “Q-learning in enormous action spaces via amortized approximate maximization”. In: **arXiv preprint arXiv:2001.08116** (page 33).

van Hasselt, H. and M. A. Wiering (2009). “Using continuous action spaces to solve discrete problems”. In: **Proceedings of the International Joint Conference on Neural Networks, (IJCNN)** (pages 30, 31).

Van Hasselt, Hado, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil (2018). “Deep reinforcement learning and the deadly triad”. In: **arXiv preprint arXiv:1812.02648** (page 17).

Van Hasselt, Hado, Arthur Guez, and David Silver (2016). “Deep reinforcement learning with double q-learning”. In: **Proceedings of the Internationnal Conference on Artificial Intelligence, (AAAI)**. 1 (page 17).

Vie, Jill-Jênn, Fabrice Popineau, Éric Bruillard, and Yolaine Bourda (2017). “A review of recent advances in adaptive assessment”. In: **Learning analytics: fundaments, applications, and trends**, pp. 113–142 (page 10).

Vinyals, Oriol, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. (2019). “Grandmaster level in StarCraft II using multi-agent reinforcement learning”. In: **Nature** 575.7782, pp. 350–354 (pages 2, 26, 89).

Vinyals, Oriol, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, et al. (2017). “Starcraft ii: A new challenge for reinforcement learning”. In: **arXiv preprint arXiv:1708.04782** (pages 26, 27, 89).

Vygotsky, Lev (1934). **Thought and language**. MIT press (page 63).

Wang, Xin, Qiuyuan Huang, Asli Çelikyilmaz, Jianfeng Gao, Dinghan Shen, Yuan-Fang Wang, William Yang Wang, and Lei Zhang (2019). “Reinforced Cross-Modal Matching and Self-Supervised Imitation Learning for Vision-Language Navigation”. In: **Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, (CVPR)** (page 64).

Wang, Ziyu, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas (2016). “Dueling Network Architectures for Deep Reinforcement Learning”. In: **Proceedings of the International Conference on Machine Learning, (ICML)** (pages 17, 69).

Watkins, Christopher JCH and Peter Dayan (1992). “Q-learning”. In: **Machine learning** 8.3-4, pp. 279–292 (pages 14, 27).

Whitehead, Steven (1991). “Complexity and cooperation in Q-learning”. In: **Proceedings of the Internationnal Conference on Artificial Intelligence, (AAAI)** (page 45).

Wiering, Marco A and Martijn Van Otterlo (2012). “Reinforcement learning”. In: **Adaptation, learning, and optimization** 12.3 (page 22).

Williams, Ronald (1992). “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. In: **Machine learning** 8.3-4, pp. 229–256 (pages 15, 22, 27).

Wilson, Margaret (2002). “Six views of embodied cognition”. In: **Psychonomic bulletin & review** 9.4, pp. 625–636 (page 1).

Witty, Sam, Jun Ki Lee, Emma Tosch, Akanksha Atrey, Michael Littman, and David Jensen (2018). “Measuring and characterizing generalization in deep reinforcement learning”. In: **arXiv preprint arXiv:1812.02868** (page 22).

Xie, Qizhe, Minh-Thang Luong, Eduard Hovy, and Quoc V Le (2020). “Self-training with noisy student improves imagenet classification”. In: **Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, (CVPR)** (page 11).

Xin Wang, D. Kumar, N. Thome, M. Cord, and F. Precioso (2015). “Recipe recognition with large multimodal food dataset”. In: **2015 IEEE International Conference on Multimedia Expo Workshops (ICMEW)** (page 10).

Xiong, Jiechao, Qing Wang, Zhuoran Yang, Peng Sun, Lei Han, Yang Zheng, Haobo Fu, Tong Zhang, Ji Liu, and Han Liu (2018). “Parametrized deep q-networks learning: Reinforcement learning with discrete-continuous hybrid action space”. In: **arXiv preprint arXiv:1810.06394** (page 33).

Xu, Dongkuan and Yingjie Tian (2015). “A comprehensive survey of clustering algorithms”. In: **Annals of Data Science** 2.2, pp. 165–193 (page 10).

Yamauchi, Brian (1998). “Frontier-based exploration using multiple robots”. In: **Proceedings of the International Conference on Autonomous Agents and MultiAgent Systems, (AAMAS)** (page 49).

Yoshida, Naoto (2015). “Q-networks for binary vector actions”. In: **arXiv preprint arXiv:1512.01332** (page 33).

You, Yang, Zhao Zhang, Cho-Jui Hsieh, James Demmel, and Kurt Keutzer (2018). “Imagenet training in minutes”. In: **Proceedings of the International Conference on Parallel Processing, (ICPP)** (page 11).

Zahavy, Tom, Matan Haroush, Nadav Merlis, Daniel J. Mankowitz, and Shie Mannor (2018). “Learn What Not to Learn: Action Elimination with Deep Reinforcement Learning”. In: **Proceedings of the Neural Information Processing Systems Conference, (NeurIPS)** (pages 32, 89).

Zang, Xiaoxue, Ashwini Pokle, Marynel Vázquez, Kevin Chen, Juan Carlos Niebles, Alvaro Soto, and Silvio Savarese (2018). “Translating Navigation Instructions in Natural Language to a High-Level Plan for Behavioral Robot Navigation”. In: **Proceedings of the Conference on Empirical Methods in Natural Language Processing, (EMNLP)** (page 64).

Zaremba, Wojciech, Tomas Mikolov, Armand Joulin, and Rob Fergus (2016). “Learning simple algorithms from examples”. In: **Proceedings of the International Conference on Machine Learning, (ICML)** (pages 29, 39).

Zhang, Amy, Yuxin Wu, and Joelle Pineau (2018). “Natural environment benchmarks for reinforcement learning”. In: **arXiv preprint arXiv:1811.06032** (page 3).

Zhang, Tianjun, Huazhe Xu, Xiaolong Wang, Yi Wu, Kurt Keutzer, Joseph E Gonzalez, and Yuandong Tian (2020). “BeBold: Exploration Beyond the Boundary of Explored Regions”. In: **arXiv preprint arXiv:2012.08621** (page 49).

Zhong, Victor, Tim Rocktäschel, and Edward Grefenstette (2019). “RTFM: Generalising to New Environment Dynamics via Reading”. In: **Proceedings of the International Conference on Learning Representations, (ICLR)** (page 90).

Zhou, Zhenpeng, Xiaocheng Li, and Richard N Zare (2017). “Optimizing chemical reactions with deep reinforcement learning”. In: **ACS central science** 3.12, pp. 1337–1344 (page 10).