An Introduction to Deep Reinforcement Learning

Vincent François-Lavet
McGill University
vincent.francois-lavet@mcgill.ca

Peter Henderson
McGill University
peter.henderson@mail.mcgill.ca

Riashat Islam
McGill University
riashat.islam@mail.mcgill.ca

Marc G. Bellemare
Google Brain
bellemare@google.com

Joelle Pineau
Facebook, McGill University
jpineau@cs.mcgill.ca

Contents

An Introduction to Deep Reinforcement Learning

Vincent François-Lavet[1], Peter Henderson[2], Riashat Islam[3], Marc G. Bellemare[4] and Joelle Pineau[5]

[1]*McGill University; vincent.francois-lavet@mcgill.ca*
[2]*McGill University; peter.henderson@mail.mcgill.ca*
[3]*McGill University; riashat.islam@mail.mcgill.ca*
[4]*Google Brain; bellemare@google.com*
[5]*Facebook, McGill University; jpineau@cs.mcgill.ca*

ABSTRACT

Deep reinforcement learning is the combination of reinforcement learning (RL) and deep learning. This field of research has been able to solve a wide range of complex decision-making tasks that were previously out of reach for a machine. Thus, deep RL opens up many new applications in domains such as healthcare, robotics, smart grids, finance, and many more. This manuscript provides an introduction to deep reinforcement learning models, algorithms and techniques. Particular focus is on the aspects related to generalization and how deep RL can be used for practical applications. We assume the reader is familiar with basic machine learning concepts.

1

Introduction

1.1 Motivation

A core topic in machine learning is that of sequential decision-making. This is the task of deciding, from experience, the sequence of actions to perform in an uncertain environment in order to achieve some goals. Sequential decision-making tasks cover a wide range of possible applications with the potential to impact many domains, such as robotics, healthcare, smart grids, finance, self-driving cars, and many more.

Inspired by behavioral psychology (see e.g., Sutton, 1984), reinforcement learning (RL) proposes a formal framework to this problem. The main idea is that an artificial agent may learn by interacting with its environment, similarly to a biological agent. Using the experience gathered, the artificial agent should be able to optimize some objectives given in the form of cumulative rewards. This approach applies in principle to any type of sequential decision-making problem relying on past experience. The environment may be stochastic, the agent may only observe partial information about the current state, the observations may be high-dimensional (e.g., frames and time series), the agent may freely gather experience in the environment or, on the contrary, the data

may be may be constrained (e.g., not access to an accurate simulator or limited data).

Over the past few years, RL has become increasingly popular due to its success in addressing challenging sequential decision-making problems. Several of these achievements are due to the combination of RL with deep learning techniques (LeCun *et al.*, 2015; Schmidhuber, 2015; Goodfellow *et al.*, 2016). This combination, called deep RL, is most useful in problems with high dimensional state-space. Previous RL approaches had a difficult design issue in the choice of features (Munos and Moore, 2002; Bellemare *et al.*, 2013). However, deep RL has been successful in complicated tasks with lower prior knowledge thanks to its ability to learn different levels of abstractions from data. For instance, a deep RL agent can successfully learn from visual perceptual inputs made up of thousands of pixels (Mnih *et al.*, 2015). This opens up the possibility to mimic some human problem solving capabilities, even in high-dimensional space — which, only a few years ago, was difficult to conceive.

Several notable works using deep RL in games have stood out for attaining super-human level in playing Atari games from the pixels (Mnih *et al.*, 2015), mastering Go (Silver *et al.*, 2016a) or beating the world's top professionals at the game of Poker (Brown and Sandholm, 2017; Moravčik *et al.*, 2017). Deep RL also has potential for real-world applications such as robotics (Levine *et al.*, 2016; Gandhi *et al.*, 2017; Pinto *et al.*, 2017), self-driving cars (You *et al.*, 2017), finance (Deng *et al.*, 2017) and smart grids (François-Lavet, 2017), to name a few. Nonetheless, several challenges arise in applying deep RL algorithms. Among others, exploring the environment efficiently or being able to generalize a good behavior in a slightly different context are not straightforward. Thus, a large array of algorithms have been proposed for the deep RL framework, depending on a variety of settings of the sequential decision-making tasks.

1.2   Outline

The goal of this introduction to deep RL is to guide the reader towards effective use and understanding of core methods, as well as provide

references for further reading. After reading this introduction, the reader should be able to understand the key different deep RL approaches and algorithms and should be able to apply them. The reader should also have enough background to investigate the scientific literature further and pursue research on deep RL.

In Chapter 2, we introduce the field of machine learning and the deep learning approach. The goal is to provide the general technical context and explain briefly where deep learning is situated in the broader field of machine learning. We assume the reader is familiar with basic notions of supervised and unsupervised learning; however, we briefly review the essentials.

In Chapter 3, we provide the general RL framework along with the case of a Markov Decision Process (MDP). In that context, we examine the different methodologies that can be used to train a deep RL agent. On the one hand, learning a value function (Chapter 4) and/or a direct representation of the policy (Chapter 5) belong to the so-called model-free approaches. On the other hand, planning algorithms that can make use of a learned model of the environment belong to the so-called model-based approaches (Chapter 6).

We dedicate Chapter 7 to the notion of generalization in RL. Within either a model-based or a model-free approach, we discuss the importance of different elements: (i) feature selection, (ii) function approximator selection, (iii) modifying the objective function and (iv) hierarchical learning. In Chapter 8, we present the main challenges of using RL in the online setting. In particular, we discuss the exploration-exploitation dilemma and the use of a replay memory.

In Chapter 9, we provide an overview of different existing benchmarks for evaluation of RL algorithms. Furthermore, we present a set of best practices to ensure consistency and reproducibility of the results obtained on the different benchmarks.

In Chapter 10, we discuss more general settings than MDPs: (i) the Partially Observable Markov Decision Process (POMDP), (ii) the distribution of MDPs (instead of a given MDP) along with the notion of transfer learning, (iii) learning without explicit reward function and (iv) multi-agent systems. We provide descriptions of how deep RL can be used in these settings.

In Chapter 11, we present broader perspectives on deep RL. This includes a discussion on applications of deep RL in various domains, along with the successes achieved and remaining challenges (e.g. robotics, self driving cars, smart grids, healthcare, etc.). This also includes a brief discussion on the relationship between deep RL and neuroscience.

Finally, we provide a conclusion in Chapter 12 with an outlook on the future development of deep RL techniques, their future applications, as well as the societal impact of deep RL and artificial intelligence.

2

Machine learning and deep learning

Machine learning provides automated methods that can detect patterns in data and use them to achieve some tasks (Christopher, 2006; Murphy, 2012). Three types of machine learning tasks can be considered:

- *Supervised learning* is the task of inferring a classification or regression from labeled training data.

- *Unsupervised learning* is the task of drawing inferences from datasets consisting of input data without labeled responses.

- *Reinforcement learning* (RL) is the task of learning how agents ought to take sequences of actions in an environment in order to maximize cumulative rewards.

To solve these machine learning tasks, the idea of function approximators is at the heart of machine learning. There exist many different types of function approximators: linear models (Anderson *et al.*, 1958), SVMs (Cortes and Vapnik, 1995), decisions tree (Liaw, Wiener, *et al.*, 2002; Geurts *et al.*, 2006), Gaussian processes (Rasmussen, 2004), deep learning (LeCun *et al.*, 2015; Schmidhuber, 2015; Goodfellow *et al.*, 2016), etc.

1958
SVMs 1995
2002/2006
2004
2015~

6

In recent years, mainly due to recent developments in deep learning, machine learning has undergone dramatic improvements when learning from high-dimensional data such as time series, images and videos. These improvements can be linked to the following aspects: (i) an exponential increase of computational power with the use of GPUs and distributed computing (Krizhevsky *et al.*, 2012), (ii) methodological breakthroughs in deep learning (Srivastava *et al.*, 2014; Ioffe and Szegedy, 2015; He *et al.*, 2016; Szegedy *et al.*, 2016; Klambauer *et al.*, 2017), (iii) a growing eco-system of softwares such as Tensorflow (Abadi *et al.*, 2016) and datasets such as ImageNet (Russakovsky *et al.*, 2015). All these aspects are complementary and, in the last few years, they have lead to a virtuous circle for the development of deep learning.

In this chapter, we discuss the supervised learning setting along with the key concepts of bias and overfitting. We briefly discuss the unsupervised setting with tasks such as data compression and generative models. We also introduce the deep learning approach that has become key to the whole field of machine learning. Using the concepts introduced in this chapter, we cover the reinforcement learning setting in later chapters.

2.1 Supervised learning and the concepts of bias and overfitting

In its most abstract form, supervised learning consists in finding a function $f : \mathcal{X} \to \mathcal{Y}$ that takes as input $x \in \mathcal{X}$ and gives as output $y \in \mathcal{Y}$ ($\mathcal{X}$ and $\mathcal{Y}$ depend on the application):

$$y = f(x). \tag{2.1}$$

A supervised learning algorithm can be viewed as a function that maps a dataset D_{LS} of learning samples $(x, y) \overset{\text{i.i.d.}}{\sim} (X, Y)$ into a model. The prediction of such a model at a point $x \in \mathcal{X}$ of the input space is denoted by $f(x \mid D_{LS})$. Assuming a random sampling scheme, s.t. $D_{LS} \sim \mathcal{D}_{LS}$, $f(x \mid D_{LS})$ is a random variable, and so is its average error over the input space. The expected value of this quantity is given by:

$$I[f] = \mathbb{E}_X \mathbb{E}_{D_{LS}} \mathbb{E}_{Y|X} L\left(Y, f(X \mid D_{LS})\right), \tag{2.2}$$

where $L(\cdot, \cdot)$ is the loss function. If $L(y, \hat{y}) = (y - \hat{y})^2$, the error decomposes naturally into a sum of a bias term and a variance term[1]. This bias-variance decomposition can be useful because it highlights a tradeoff between an error due to erroneous assumptions in the model selection/learning algorithm (the bias) and an error due to the fact that only a finite set of data is available to learn that model (the parametric variance). Note that the parametric variance is also called the overfitting error[2]. Even though there is no such direct decomposition for other loss underfitting. functions (James, 2003), there is always a tradeoff between a sufficiently rich model (to reduce the model bias, which is present even when the amount of data would be unlimited) and a model not too complex (so as to avoid overfitting to the limited amount of data). Figure 2.1 provides an illustration.

Without knowing the joint probability distribution, it is impossible to compute $I[f]$. Instead, we can compute the empirical error on a sample of data. Given n data points (x_i, y_i), the empirical error is

$$I_S[f] = \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i)).$$

The *generalization error* is the difference between the error on a sample set (used for training) and the error on the underlying joint probability distribution. It is defined as

$$G = I[f] - I_S[f].$$

---

[1] The bias-variance decomposition (Geman *et al.*, 1992) is given by:

$$\mathop{\mathbb{E}}_{D_{LS}} \mathop{\mathbb{E}}_{Y|X} (Y - f(X \mid D_{LS}))^2 = \sigma^2(x) + \text{bias}^2(x), \qquad (2.3)$$

where

$$\text{bias}^2(x) \triangleq \left(\mathbb{E}_{Y|x}(Y) - \mathbb{E}_{D_{LS}} f(x \mid D_{LS}) \right)^2,$$

$$\sigma^2(x) \triangleq \underbrace{\mathbb{E}_{Y|x} \left(Y - \mathbb{E}_{Y|x}(Y) \right)^2}_{\text{Internal variance}} + \underbrace{\mathbb{E}_{D_{LS}} \left(f(x \mid D_{LS}) - \mathbb{E}_{D_{LS}} f(x \mid D_{LS}) \right)^2}_{\text{Parametric variance}}, \qquad (2.4)$$

[2] For any given model, the parametric variance goes to zero with an arbitrary large dataset by considering the strong law of convergence.
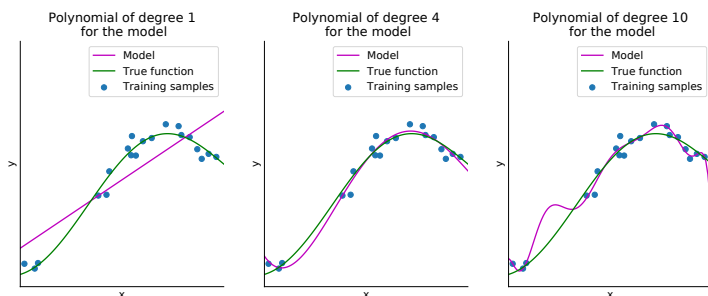
Figure 2.1: Illustration of overfitting and underfitting for a simple 1D regression task in supervised learning (based on one example from the library scikit-learn (Pedregosa *et al.*, 2011)). In this illustration, the data points (x, y) are noisy samples from a true function represented in green. In the left figure, the degree 1 approximation is underfitting, which means that it is not a good model, even for the training samples; on the right, the degree 10 approximation is a very good model for the training samples but is overly complex and fails to provide a good generalization.

In machine learning, the complexity of the function approximator provides upper bounds on the generalization error. The generalization error can be bounded by making use of complexity measures, such as the Rademacher complexity (Bartlett and Mendelson, 2002), or the VC-dimension (Vapnik, 1998). However, even though it lacks strong theoretical foundations, it has become clear in practice that the strength of deep neural networks is their generalization capability, even with a high number of parameters (hence a potentially high complexity) (Zhang *et al.*, 2016).

2.2 Unsupervised learning

Unsupervised learning is a branch of machine learning that learns from data that do not have any label. It relates to using and identifying patterns in the data for tasks such as data compression or generative models.

Data compression or dimensionality reduction involve encoding information using a smaller representation (e.g., fewer bits) than the original representation. For instance, an auto-encoder consists of an encoder and a decoder. The encoder maps the original image $x_i \in \mathbb{R}^M$

onto a low-dimensional representation $z_i = e(x_i; \theta_e) \in \mathbb{R}^m$, where $m << M$; the decoder maps these features back to a high-dimensional representation $d(z_i; \theta_d) \approx e^{-1}(z_i; \theta_e)$. Auto-encoders can be trained by optimizing for the reconstruction of the input through supervised learning objectives.

Generative models aim at approximating the true data distribution of a training set so as to generate new data points from the distribution. Generative adversarial networks (Goodfellow *et al.*, 2014) use an adversarial process, in which two models are trained simulatenously: a generative model G captures the data distribution, while a discriminative model D estimates whether a sample comes from the training data rather than G. The training procedure corresponds to a minimax two-player game.

2.3   The deep learning approach

Deep learning relies on a function $f : \mathcal{X} \to \mathcal{Y}$ parameterized with $\theta \in \mathbb{R}^{n_\theta}$ ($n_\theta \in \mathbb{N}$):

$$y = f(x; \theta). \tag{2.5}$$

A deep neural network is characterized by a succession of multiple processing layers. Each layer consists in a non-linear transformation and the sequence of these transformations leads to learning different levels of abstraction (Erhan *et al.*, 2009; Olah *et al.*, 2017).

First, let us describe a very simple neural network with one fully-connected hidden layer (see Fig 2.2). The first layer is given the input values (i.e., the input features) x in the form of a column vector of size n_x ($n_x \in \mathbb{N}$). The values of the next hidden layer are a transformation of these values by a non-linear parametric function, which is a matrix multiplication by W_1 of size $n_h \times n_x$ ($n_h \in \mathbb{N}$), plus a bias term b_1 of size n_h, followed by a non-linear transformation:

$$h = A(W_1 \cdot x + b_1), \tag{2.6}$$

where A is the *activation function*. This non-linear activation function is what makes the transformation at each layer non-linear, which ultimately provides the expressivity of the neural network. The hidden layer h of

size n_h can in turn be transformed to other sets of values up to the last transformation that provides the output values y. In this case:

$$y = (W_2 \cdot h + b_2), \tag{2.7}$$

where W_2 is of size $n_y \times n_h$ and b_2 is of size n_y $(n_y \in \mathbb{N})$.
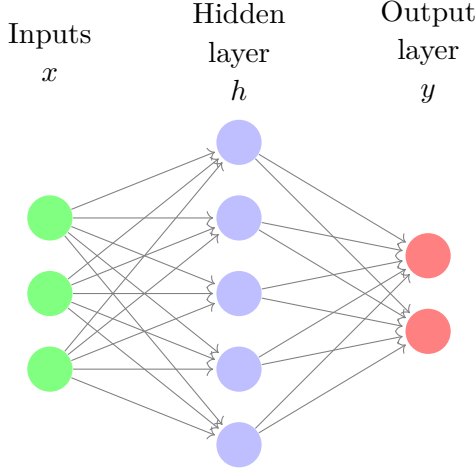
Figure 2.2: Example of a neural network with one hidden layer.

All these layers are trained to minimize the empirical error $I_S[f]$. The most common method for optimizing the parameters of a neural network is based on gradient descent via the backpropagation algorithm (Rumelhart *et al.*, 1988). In the simplest case, at every iteration, the algorithm changes its internal parameters θ so as to fit the desired function:

$$\theta \leftarrow \theta - \alpha \nabla_\theta I_S[f], \tag{2.8}$$

where α is the learning rate.

In current applications, many different types of neural network layers have appeared beyond the simple feedforward networks just introduced. Each variation provides specific advantages, depending on the application (e.g., good tradeoff between bias and overfitting in a supervised learning setting). In addition, within one given neural network, an arbitrarily large number of layers is possible, and the trend

in the last few years is to have an ever-growing number of layers, with more than 100 in some supervised learning tasks (Szegedy *et al.*, 2017). We merely describe here two types of layers that are of particular interest in deep RL (and in many other tasks).

Convolutional layers (LeCun, Bengio, *et al.*, 1995) are particularly well suited for images and sequential data (see Fig 2.3), mainly due to their translation invariance property. The layer's parameters consist of a set of learnable filters (or kernels), which have a small receptive field and which apply a convolution operation to the input, passing the result to the next layer. As a result, the network learns filters that activate when it detects some specific features. In image classification, the first layers learn how to detect edges, textures and patterns; then the following layers are able to detect parts of objects and whole objects (Erhan *et al.*, 2009; Olah *et al.*, 2017). In fact, a convolutional layer is a particular kind of feedforward layer, with the specificity that many weights are set to 0 (not learnable) and that other weights are shared.
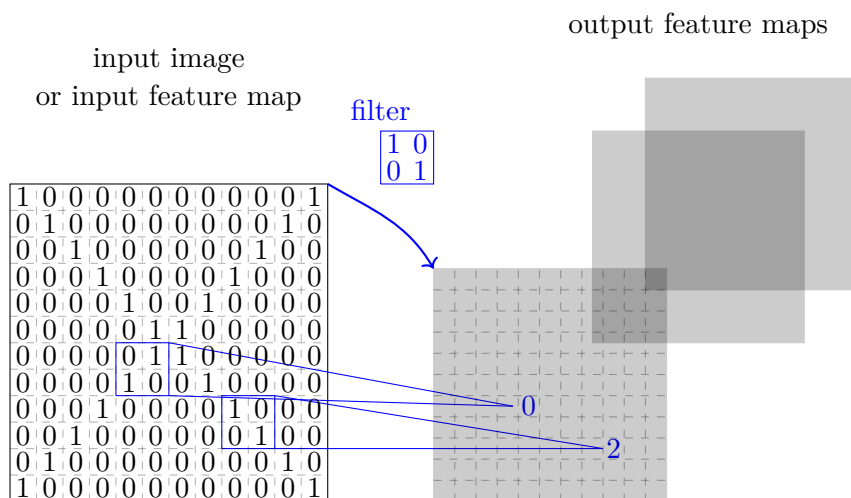
Figure 2.3: Illustration of a convolutional layer with one input feature map that is convolved by different filters to yield the output feature maps. The parameters that are learned for this type of layer are those of the filters. For illustration purposes, some results are displayed for one of the output feature maps with a given filter (in practice, that operation is followed by a non-linear activation function).
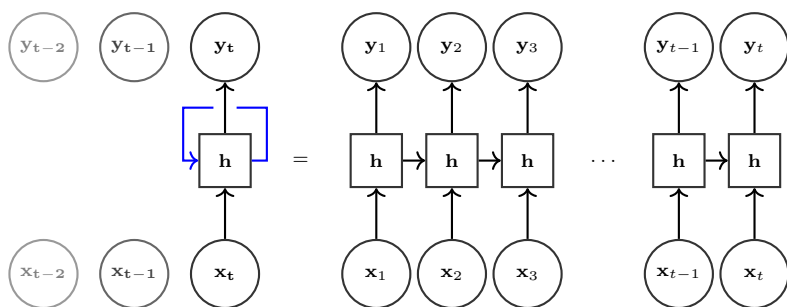
Figure 2.4: Illustration of a simple recurrent neural network. The layer denoted by "h" may represent any non linear function that takes two inputs and provides two outputs. On the left is the simplified view of a recurrent neural network that is applied recursively to (x_t, y_t) for increasing values of t and where the blue line presents a delay of one time step. On the right, the neural network is unfolded with the implicit requirement of presenting all inputs and outputs simultaneously.

Recurrent layers are particularly well suited for sequential data (see Fig 2.4). Several different variants provide particular benefits in different settings. One such example is the long short-term memory network (LSTM) (Hochreiter and Schmidhuber, 1997), which is able to encode information from long sequences, unlike a basic recurrent neural network. Neural Turing Machines (NTMs) (Graves *et al.*, 2014) are another such example. In such systems, a differentiable "external memory" is used for inferring even longer-term dependencies than LSTMs with low degradation.

Several other specific neural network architectures have also been studied to improve generalization in deep learning. For instance, it is possible to design an architecture in such a way that it automatically focuses on only some parts of the inputs with a mechanism called attention (Xu *et al.*, 2015; Vaswani *et al.*, 2017). Other approaches aim to work with symbolic rules by learning to create programs (Reed and De Freitas, 2015; Neelakantan *et al.*, 2015; Johnson *et al.*, 2017; Chen *et al.*, 2017).

To be able to actually apply the deep RL methods described in the later chapters, the reader should have practical knowledge of applying deep learning methods in simple supervised learning settings (e.g., MNIST classification). For information on topics such as the importance

of input normalizations, weight initialization techniques, regularization techniques and the different variants of gradient descent techniques, the reader can refer to several reviews on the subject (LeCun *et al.*, 2015; Schmidhuber, 2015; Goodfellow *et al.*, 2016) as well as references therein.

In the following chapters, the focus is on reinforcement learning, in particular on methods that scale to deep neural network function approximators. These methods allows for learning a wide variety of challenging sequential decision-making tasks directly from rich high-dimensional inputs.

3

Introduction to reinforcement learning

Reinforcement learning (RL) is the area of machine learning that deals
with sequential decision-making. In this chapter, we describe how the
RL problem can be formalized as an agent that has to make decisions
in an environment to optimize a given notion of cumulative rewards.
It will become clear that this formalization applies to a wide variety
of tasks and captures many essential features of artificial intelligence
such as a sense of cause and effect as well as a sense of uncertainty and
nondeterminism. This chapter also introduces the different approaches
to learning sequential decision-making tasks and how deep RL can be
useful.

A key aspect of RL is that an agent *learns* a good behavior. This
means that it modifies or acquires new behaviors and skills incrementally.
Another important aspect of RL is that it uses trial-and-error *experience*
(as opposed to e.g., dynamic programming that assumes full knowledge
of the environment a priori). Thus, the RL agent does not require
complete knowledge or control of the environment; it only needs to be
able to interact with the environment and collect information. In an
offline setting, the experience is acquired a priori, then it is used as a
batch for learning (hence the offline setting is also called batch RL).

This is in contrast to the *online* setting where data becomes available in a sequential order and is used to progressively update the behavior of the agent. In both cases, the core learning algorithms are essentially the same but the main difference is that in an online setting, the agent can influence how it gathers experience so that it is the most useful for learning. This is an additional challenge mainly because the agent has to deal with the *exploration/exploitation* dilemma while learning (see §8.1 for a detailed discussion). But learning in the online setting can also be an advantage since the agent is able to gather information specifically on the most interesting part of the environment. For that reason, even when the environment is fully known, RL approaches may provide the most computationally efficient approach in practice as compared to some dynamic programming methods that would be inefficient due to this lack of specificity.

3.1   Formal framework

The reinforcement learning setting

The general RL problem is formalized as a discrete time stochastic control process where an agent interacts with its environment in the following way: the agent starts, in a given state within its environment $s_0 \in \mathcal{S}$, by gathering an initial observation $\omega_0 \in \Omega$. At each time step t, the agent has to take an action $a_t \in \mathcal{A}$. As illustrated in Figure 3.1, it follows three consequences: (i) the agent obtains a reward $r_t \in \mathcal{R}$, (ii) the state transitions to $s_{t+1} \in \mathcal{S}$, and (iii) the agent obtains an observation $\omega_{t+1} \in \Omega$. This control setting was first proposed by Bellman, 1957b and later extended to learning by Barto *et al.*, 1983. Comprehensive treatment of RL fundamentals are provided by Sutton and Barto, 2017. Here, we review the main elements of RL before delving into deep RL in the following chapters.

The Markov property

For the sake of simplicity, let us consider first the case of Markovian stochastic control processes (Norris, 1998).
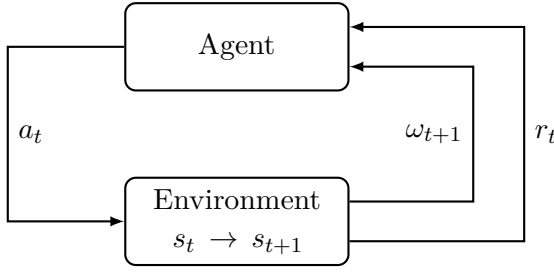
Figure 3.1: Agent-environment interaction in RL.

Definition 3.1. A discrete time stochastic control process is Markovian (i.e., it has the Markov property) if

- $\mathbb{P}(\omega_{t+1} \mid \omega_t, a_t) = \mathbb{P}(\omega_{t+1} \mid \omega_t, a_t, \ldots, , \omega_0, a_0)$, and

- $\mathbb{P}(r_t \mid \omega_t, a_t) = \mathbb{P}(r_t \mid \omega_t, a_t, \ldots, , \omega_0, a_0)$.

The Markov property means that the future of the process only depends on the current observation, and the agent has no interest in looking at the full history.

A Markov Decision Process (MDP) (Bellman, 1957a) is a discrete time stochastic control process defined as follows:

Definition 3.2. An MDP is a 5-tuple $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$ where:

- $\mathcal{S}$ is the state space,

- $\mathcal{A}$ is the action space,

- $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition function (set of conditional transition probabilities between states),

- $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathcal{R}$ is the reward function, where $\mathcal{R}$ is a continuous set of possible rewards in a range $R_{\max} \in \mathbb{R}^+$ (e.g., $[0, R_{\max}]$),

- $\gamma \in [0, 1)$ is the discount factor.

The system is fully observable in an MDP, which means that the observation is the same as the state of the environment: $\omega_t = s_t$. At each time step t, the probability of moving to s_{t+1} is given by the state

transition function $T(s_t, a_t, s_{t+1})$ and the reward is given by a bounded reward function $R(s_t, a_t, s_{t+1}) \in \mathcal{R}$. This is illustrated in Figure 3.2. Note that more general cases than MDPs are introduced in Chapter 10.
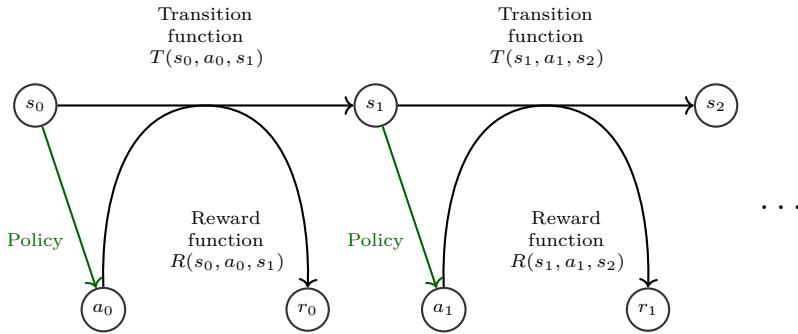
Figure 3.2: Illustration of a MDP. At each step, the agent takes an action that changes its state in the environment and provides a reward.

Different categories of policies

A policy defines how an agent selects actions. Policies can be categorized under the criterion of being either stationary or non-stationary. A non-stationary policy depends on the time-step and is useful for the finite-horizon context where the cumulative rewards that the agent seeks to optimize are limited to a finite number of future time steps (Bertsekas *et al.*, 1995). In this introduction to deep RL, infinite horizons are considered and the policies are stationary[1].

Policies can also be categorized under a second criterion of being either deterministic or stochastic:

- In the deterministic case, the policy is described by $\pi(s) : \mathcal{S} \to \mathcal{A}$.

- In the stochastic case, the policy is described by $\pi(s, a) : \mathcal{S} \times \mathcal{A} \to [0, 1]$ where $\pi(s, a)$ denotes the probability that action a may be chosen in state s.

---

[1]The formalism can be directly extended to the finite horizon context. In that case, the policy and the cumulative expected returns should be time-dependent.

The expected return

Throughout this survey, we consider the case of an RL agent whose goal is to find a policy $\pi(s, a) \in \Pi$, so as to optimize an expected return $V^\pi(s) : \mathcal{S} \to \mathbb{R}$ (also called V-value function) such that

$$V^\pi(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s, \pi\right], \tag{3.1}$$

where:

- $r_t = \underset{a \sim \pi(s_t, \cdot)}{\mathbb{E}} R(s_t, a, s_{t+1})$,

- $\mathbb{P}(s_{t+1} | s_t, a_t) = T(s_t, a_t, s_{t+1})$ with $a_t \sim \pi(s_t, \cdot)$,

From the definition of the expected return, the optimal expected return can be defined as:

$$V^*(s) = \max_{\pi \in \Pi} V^\pi(s). \tag{3.2}$$

In addition to the V-value function, a few other functions of interest can be introduced. The Q-value function $Q^\pi(s, a) : \mathcal{S} \times A \to \mathbb{R}$ is defined as follows:

$$Q^\pi(s, a) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s, a_t = a, \pi\right]. \tag{3.3}$$

This equation can be rewritten recursively in the case of an MDP using Bellman's equation:

$$Q^\pi(s, a) = \sum_{s' \in S} T(s, a, s') \left(R(s, a, s') + \gamma Q^\pi(s', a = \pi(s'))\right). \tag{3.4}$$

Similarly to the V-value function, the optimal Q-value function $Q^*(s, a)$ can also be defined as

$$Q^*(s, a) = \max_{\pi \in \Pi} Q^\pi(s, a). \tag{3.5}$$

The particularity of the Q-value function as compared to the V-value function is that the optimal policy can be obtained directly from $Q^*(s, a)$:

$$\pi^*(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q^*(s, a). \tag{3.6}$$

The optimal V-value function $V^*(s)$ is the expected discounted reward when in a given state s while following the policy π^* thereafter. The

optimal Q-value $Q^*(s, a)$ is the expected discounted return when in a given state s and for a given action a while following the policy π^* thereafter.

It is also possible to define the advantage function

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s). \tag{3.7}$$

This quantity describes how good the action a is, as compared to the expected return when following directly policy π.

Note that one straightforward way to obtain estimates of either $V^\pi(s)$, $Q^\pi(s, a)$ or $A^\pi(s, a)$ is to use Monte Carlo methods, i.e. defining an estimate by performing several simulations from s while following policy π. In practice, we will see that this may not be possible in the case of limited data. In addition, even when it is possible, we will see that other methods should usually be preferred for computational efficiency.

3.2   Different components to learn a policy

An RL agent includes one or more of the following components:

- a representation of a *value function* that provides a prediction of how good each state or each state/action pair is,

- a direct representation of the *policy* $\pi(s)$ or $\pi(s, a)$, or

- a *model* of the environment (the estimated transition function and the estimated reward function) in conjunction with a planning algorithm.

The first two components are related to what is called *model-free* RL and are discussed in Chapters 4, 5. When the latter component is used, the algorithm is referred to as *model-based* RL, which is discussed in Chapter 6. A combination of both and why using the combination can be useful is discussed in §6.2. A schema with all possible approaches is provided in Figure 3.3.

For most problems approaching real-world complexity, the state space is high-dimensional (and possibly continuous). In order to learn an estimate of the model, the value function or the policy, there are two main advantages for RL algorithms to rely on deep learning:
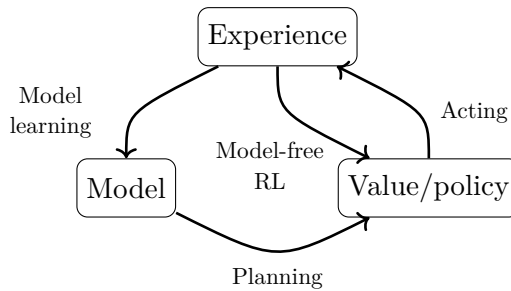
Figure 3.3: General schema of the different methods for RL. The direct approach uses a representation of either a value function or a policy to act in the environment. The indirect approach makes use of a model of the environment.

- Neural networks are well suited for dealing with high-dimensional sensory inputs (such as times series, frames, etc.) and, in practice, they do not require an exponential increase of data when adding extra dimensions to the state or action space (see Chapter 2).

- In addition, they can be trained incrementally and make use of additional samples obtained as learning happens.

3.3 Different settings to learn a policy from data

We now describe key settings that can be tackled with RL.

3.3.1 Offline and online learning

Learning a sequential decision-making task appears in two cases: (i) in the offline learning case where only limited data on a given environment is available and (ii) in an online learning case where, in parallel to learning, the agent gradually gathers experience in the environment. In both cases, the core learning algorithms introduced in Chapters 4 to 6 are essentially the same. The specificity of the batch setting is that the agent has to learn from limited data without the possibility of interacting further with the environment. In that case, the idea of generalization introduced in Chapter 7 is the main focus. In the online setting, the learning problem is more intricate and learning without requiring a large amount of data (sample efficiency) is not only influenced by the

capability of the learning algorithm to generalize well from the limited experience. Indeed, the agent has the possibility to gather experience via an *exploration/exploitation strategy*. In addition, it can use a *replay memory* to store its experience so that it can be reprocessed at a later time. Both the exploration and the replay memory will be discussed in Chapter 8). In both the batch and the online settings, a supplementary consideration is also the computational efficiency, which, among other things, depends on the efficiency of a given gradient descent step. All these elements will be introduced with more details in the following chapters. A general schema of the different elements that can be found in most deep RL algorithms is provided in Figure 3.4.
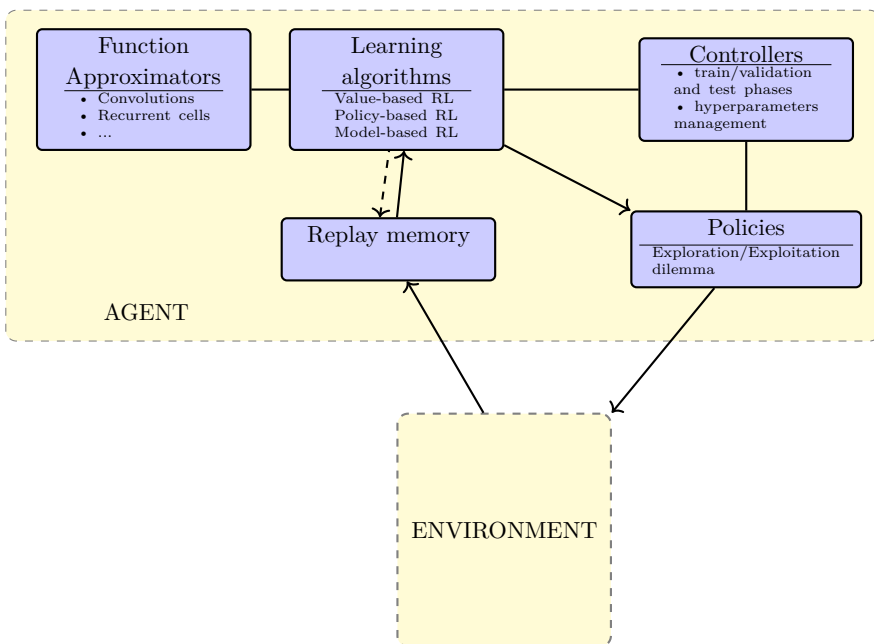
Figure 3.4: General schema of deep RL methods.

3.3.2   Off-policy and on-policy learning

According to Sutton and Barto, 2017, « on-policy methods attempt to evaluate or improve the policy that is used to make decisions, whereas

off-policy methods evaluate or improve a policy different from that used to generate the data ». In off-policy based methods, learning is straightforward when using trajectories that are not necessarily obtained under the current policy, but from a different behavior policy $\beta(s,a)$. In those cases, experience replay allows reusing samples from a different behavior policy. On the contrary, on-policy based methods usually introduce a bias when used with a replay buffer as the trajectories are usually not obtained solely under the current policy π. As will be discussed in the following chapters, this makes off-policy methods sample efficient as they are able to make use of any experience; in contrast, on-policy methods would, if specific care is not taken, introduce a bias when using off-policy trajectories.

4

Value-based methods for deep RL

The value-based class of algorithms aims to build a value function, which subsequently lets us define a policy. We discuss hereafter one of the simplest and most popular value-based algorithms, the Q-learning algorithm (Watkins, 1989) and its variant, the fitted Q-learning, that uses parameterized function approximators (Gordon, 1996). We also specifically discuss the main elements of the deep Q-network (DQN) algorithm (Mnih *et al.*, 2015) which has achieved superhuman-level control when playing ATARI games from the pixels by using neural networks as function approximators. We then review various improvements of the DQN algorithm and provide resources for further details. At the end of this chapter and in the next chapter, we discuss the intimate link between value-based methods and policy-based methods.

4.1 Q-learning

The basic version of Q-learning keeps a lookup table of values $Q(s, a)$ (Equation 3.3) with one entry for every state-action pair. In order to learn the optimal Q-value function, the Q-learning algorithm makes use

of the Bellman equation for the Q-value function (Bellman and Dreyfus, 1962) whose unique solution is $Q^*(s, a)$:

$$Q^*(s, a) = (\mathcal{B}Q^*)(s, a), \tag{4.1}$$

where $\mathcal{B}$ is the Bellman operator mapping any function $K : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ into another function $\mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and is defined as follows:

$$(\mathcal{B}K)(s, a) = \sum_{s' \in S} T(s, a, s') \left(R(s, a, s') + \gamma \max_{a' \in \mathcal{A}} K(s', a') \right). \tag{4.2}$$

By Banach's theorem, the fixed point of the Bellman operator $\mathcal{B}$ exists since it is a contraction mapping[1]. In practice, one general proof of convergence to the optimal value function is available (Watkins and Dayan, 1992) under the conditions that:

- the state-action pairs are represented discretely, and

- all actions are repeatedly sampled in all states (which ensures sufficient exploration, hence not requiring access to the transition model).

This simple setting is often inapplicable due to the high-dimensional (possibly continuous) state-action space. In that context, a parameterized value function $Q(s, a; \theta)$ is needed, where θ refers to some parameters that define the Q-values.

4.2 Fitted Q-learning

Experiences are gathered in a given dataset D in the form of tuples $< s, a, r, s' >$ where the state at the next time-step s' is drawn from $T(s, a, \cdot)$ and the reward r is given by $R(s, a, s')$. In fitted Q-learning (Gordon, 1996), the algorithm starts with some random initialization of the Q-values $Q(s, a; \theta_0)$ where θ_0 refers to the initial parameters (usually

---

[1]The Bellman operator is a contraction mapping because it can be shown that for any pair of bounded functions $K, K' : S \times A \to \mathbb{R}$, the following condition is respected:

$$\|TK - TK'\|_\infty \leq \gamma \|K - K'\|_\infty.$$

such that the initial Q-values should be relatively close to 0 so as to avoid slow learning). Then, an approximation of the Q-values at the kth iteration $Q(s, a; \theta_k)$ is updated towards the target value

$$Y_k^Q = r + \gamma \max_{a' \in \mathcal{A}} Q(s', a'; \theta_k), \tag{4.3}$$

where θ_k refers to some parameters that define the Q-values at the kth iteration.

In neural fitted Q-learning (NFQ) (Riedmiller, 2005), the state can be provided as an input to the Q-network and a different output is given for each of the possible actions. This provides an efficient structure that has the advantage of obtaining the computation of $\max_{a' \in A} Q(s', a'; \theta_k)$ in a single forward pass in the neural network for a given s'. The Q-values are parameterized with a neural network $Q(s, a; \theta_k)$ where the parameters θ_k are updated by stochastic gradient descent (or a variant) by minimizing the square loss:

$$\mathrm{L}_{DQN} = \left(Q(s, a; \theta_k) - Y_k^Q \right)^2. \tag{4.4}$$

Thus, the Q-learning update amounts in updating the parameters:

$$\theta_{k+1} = \theta_k + \alpha \left(Y_k^Q - Q(s, a; \theta_k) \right) \nabla_{\theta_k} Q(s, a; \theta_k), \tag{4.5}$$

where α is a scalar step size called the learning rate. Note that using the square loss is not arbitrary. Indeed, it ensures that $Q(s, a; \theta_k)$ should tend without bias to the expected value of the random variable Y_k^Q [2]. Hence, it ensures that $Q(s, a; \theta_k)$ should tend to $Q^*(s, a)$ after many iterations in the hypothesis that the neural network is well-suited for the task and that the experience gathered in the dataset D is sufficient (more details will be given in Chapter 7).

When updating the weights, one also changes the target. Due to the generalization and extrapolation abilities of neural networks, this approach can build large errors at different places in the state-action space[3]. Therefore, the contraction mapping property of the Bellman

---

[2]The minimum of $\mathbb{E}[(Z - c)^2]$ occurs when the constant c equals the expected value of the random variable Z.

[3]Note that even fitted value iteration with linear regression can diverge (Boyan and Moore, 1995). However, this drawback does not happen when using linear

operator in Equation 4.2 is not enough to guarantee convergence. It is verified experimentally that these errors may propagate with this update rule and, as a consequence, convergence may be slow or even unstable (Baird, 1995; Tsitsiklis and Van Roy, 1997; Gordon, 1999; Riedmiller, 2005). Another related damaging side-effect of using function approximators is the fact that Q-values tend to be overestimated due to the max operator (Van Hasselt *et al.*, 2016). Because of the instabilities and the risk of overestimation, specific care has be taken to ensure proper learning.

4.3 Deep Q-networks

Leveraging ideas from NFQ, the deep Q-network (DQN) algorithm introduced by Mnih *et al.* (2015) is able to obtain strong performance in an online setting for a variety of ATARI games, directly by learning from the pixels. It uses two heuristics to limit the instabilities:

- The target Q-network in Equation 4.3 is replaced by $Q(s', a'; \theta_k^-)$ where its parameters θ_k^- are updated only every $C \in \mathbb{N}$ iterations with the following assignment: $\theta_k^- = \theta_k$. This prevents the instabilities to propagate quickly and it reduces the risk of divergence as the target values Y_k^Q are kept fixed for C iterations. The idea of target networks can be seen as an instantiation of fitted Q-learning, where each period between target network updates corresponds to a single fitted Q-iteration.

- In an online setting, the replay memory (Lin, 1992) keeps all information for the last $N_{\text{replay}} \in \mathbb{N}$ time steps, where the experience is collected by following an ϵ-greedy policy[4]. The updates are then made on a set of tuples $< s, a, r, s' >$ (called mini-batch) selected randomly within the replay memory. This

---

function approximators that only have interpolation abilities such as kernel-based regressors (k-nearest neighbors, linear and multilinear interpolation, etc.) (Gordon, 1999) or tree-based ensemble methods (Ernst *et al.*, 2005). However, these methods have not proved able to handle successfully high-dimensional inputs.

[4]It takes a random action with probability ϵ and follows the policy given by $\text{argmax}_{a \in \mathcal{A}} Q(s, a; \theta_k)$ with probability $1 - \epsilon$.

technique allows for updates that cover a wide range of the state-action space. In addition, one mini-batch update has less variance compared to a single tuple update. Consequently, it provides the possibility to make a larger update of the parameters, while having an efficient parallelization of the algorithm.

A sketch of the algorithm is given in Figure 4.1.

In addition to the target Q-network and the replay memory, DQN uses other important heuristics. To keep the target values in a reasonable scale and to ensure proper learning in practice, rewards are clipped between -1 and +1. Clipping the rewards limits the scale of the error derivatives and makes it easier to use the same learning rate across multiple games (however, it introduces a bias). In games where the player has multiple lives, one trick is also to associate a terminal state to the loss of a life such that the agent avoids these terminal states (in a terminal state the discount factor is set to 0).

In DQN, many deep learning specific techniques are also used. In particular, a preprocessing step of the inputs is used to reduce the input dimensionality, to normalize inputs (it scales pixels value into [-1,1]) and to deal with some specificities of the task. In addition, convolutional layers are used for the first layers of the neural network function approximator and the optimization is performed using a variant of stochastic gradient descent called RMSprop (Tieleman, 2012).

4.4 Double DQN

The max operation in Q-learning (Equations 4.2, 4.3) uses the same values both to select and to evaluate an action. This makes it more likely to select overestimated values in case of inaccuracies or noise, resulting in overoptimistic value estimates. Therefore, the DQN algorithm induces an upward bias. The double estimator method uses two estimates for each variable, which allows for the selection of an estimator and its value to be uncoupled (Hasselt, 2010). Thus, regardless of whether errors in the estimated Q-values are due to stochasticity in the environment, function approximation, non-stationarity, or any other source, this allows for the removal of the positive bias in estimating the action
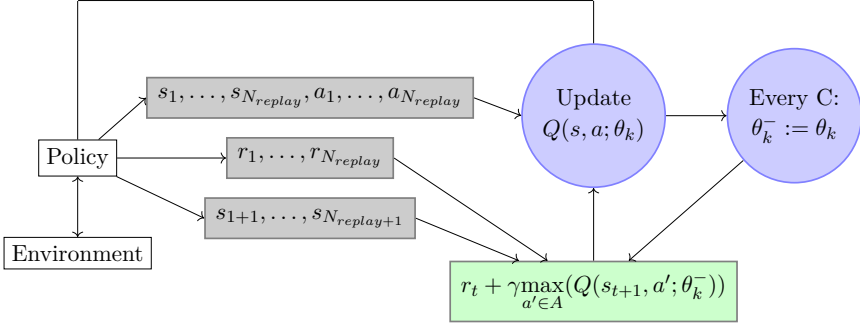
Figure 4.1: Sketch of the DQN algorithm. $Q(s, a; \theta_k)$ is initialized to random values (close to 0) everywhere in its domain and the replay memory is initially empty; the target Q-network parameters θ_k^- are only updated every C iterations with the Q-network parameters θ_k and are held fixed between updates; the update uses a mini-batch (e.g., 32 elements) of tuples $< s, a >$ taken randomly in the replay memory along with the corresponding mini-batch of target values for the tuples.

values. In Double DQN, or DDQN (Van Hasselt *et al.*, 2016), the target value Y_k^Q is replaced by

$$Y_k^{DDQN} = r + \gamma Q(s', \underset{a \in \mathcal{A}}{\operatorname{argmax}} Q(s', a; \theta_k); \theta_k^-), \qquad (4.6)$$

which leads to less overestimation of the Q-learning values, as well as improved stability, hence improved performance. As compared to DQN, the target network with weights θ_t^- are used for the evaluation of the current greedy action. Note that the policy is still chosen according to the values obtained by the current weights θ.

4.5 Dueling network architecture

In (Wang *et al.*, 2015), the neural network architecture decouples the value and advantage function $A^\pi(s, a)$ (Equation 3.7), which leads to improved performance. The Q-value function is given by

$$\begin{aligned} Q(s, a; \theta^{(1)}, \theta^{(2)}, \theta^{(3)}) = {} & V\left(s; \theta^{(1)}, \theta^{(3)}\right) \\ & + \left(A\left(s, a; \theta^{(1)}, \theta^{(2)}\right) - \max_{a' \in \mathcal{A}} A\left(s, a'; \theta^{(1)}, \theta^{(2)}\right)\right). \end{aligned} \qquad (4.7)$$

Now, for $a^* = \operatorname{argmax}_{a' \in \mathcal{A}} Q(s, a'; \theta^{(1)}, \theta^{(2)}, \theta^{(3)})$, we obtain $Q(s, a^*; \theta^{(1)}, \theta^{(2)}, \theta^{(3)}) = V(s; \theta^{(1)}, \theta^{(3)})$. As illustrated in Figure 4.2,

the stream $V(s; \theta^{(1)}, \theta^{(3)})$ provides an estimate of the value function, while the other stream produces an estimate of the advantage function. The learning update is done as in DQN and it is only the structure of the neural network that is modified.
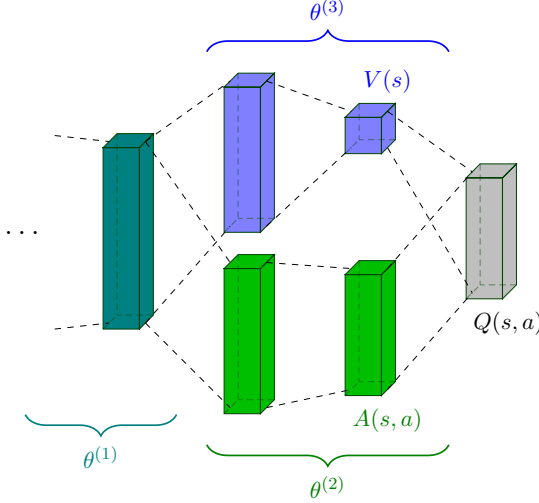
Figure 4.2: Illustration of the dueling network architecture with the two streams that separately estimate the value $V(s)$ and the advantages $A(s, a)$. The boxes represent layers of a neural network and the grey output implements equation 4.7 to combine $V(s)$ and $A(s, a)$.

In fact, even though it loses the original semantics of V and A, a slightly different approach is preferred in practice because it increases the stability of the optimization:

$$
\begin{aligned}
Q(s, a; \theta^{(1)}, \theta^{(2)}, \theta^{(3)}) = &\ V\left(s; \theta^{(1)}, \theta^{(3)}\right) \\
&+ \left(A\left(s, a; \theta^{(1)}, \theta^{(2)}\right) - \frac{1}{|\mathcal{A}|} \sum_{a' \in \mathcal{A}} A\left(s, a'; \theta^{(1)}, \theta^{(2)}\right) \right).
\end{aligned}
\tag{4.8}
$$

In that case, the advantages only need to change as fast as the mean, which appears to work better in practice (Wang *et al.*, 2015).

4.6 Distributional DQN

The approaches described so far in this chapter all directly approximate
the expected return in a value function. Another approach is to aim for
a richer representation through a value distribution, i.e. the distribution
of possible cumulative returns (Jaquette *et al.*, 1973; Morimura *et al.*,
2010). This value distribution provides more complete information of
the intrinsic randomness of the rewards and transitions of the agent
within its environment (note that it is not a measure of the agent's
uncertainty about the environment).

The value distribution Z^π is a mapping from state-action pairs to
distributions of returns when following policy π. It has an expectation
equal to Q^π:

$$Q^\pi(s, a) = \mathbb{E} Z^\pi(s, a).$$

This random return is also described by a recursive equation, but one
of a distributional nature:

$$Z^\pi(s, a) = R(s, a, S') + \gamma Z^\pi(S', A'), \qquad (4.9)$$

where we use capital letters to emphasize the random nature of the
next state-action pair (S', A') and $A' \sim \pi(\cdot|S')$. The distributional
Bellman equation states that the distribution of Z is characterized by
the interaction of three random variables: the reward $R(s, a, S')$, the
next state-action (S', A'), and its random return $Z^\pi(S', A')$.

It has been shown that such a distributional Bellman equation can
be used in practice, with deep learning as the function approximator
(Bellemare *et al.*, 2017; Dabney *et al.*, 2017; Rowland *et al.*, 2018). This
approach has the following advantages:

- It is possible to implement risk-aware behavior (see e.g., Morimura
  et al., 2010).

- It leads to more performant learning in practice. This may appear
  surprising since both DQN and the distributional DQN aim to
  maximize the expected return (as illustrated in Figure 4.3). One of
  the main elements is that the distributional perspective naturally
  provides a richer set of training signals than a scalar value function

$Q(s, a)$. These training signals that are not a priori necessary for optimizing the expected return are known as *auxiliary tasks* (Jaderberg *et al.*, 2016) and lead to an improved learning (this is discussed in §7.2.1).

(a) Example MDP.

(b) Sketch (in an idealized version) of the estimate of resulting value distribution $\hat{Z}^{\pi_1}$ and $\hat{Z}^{\pi_2}$ as well as the estimate of the Q-values $\hat{Q}^{\pi_1}, \hat{Q}^{\pi_2}$.
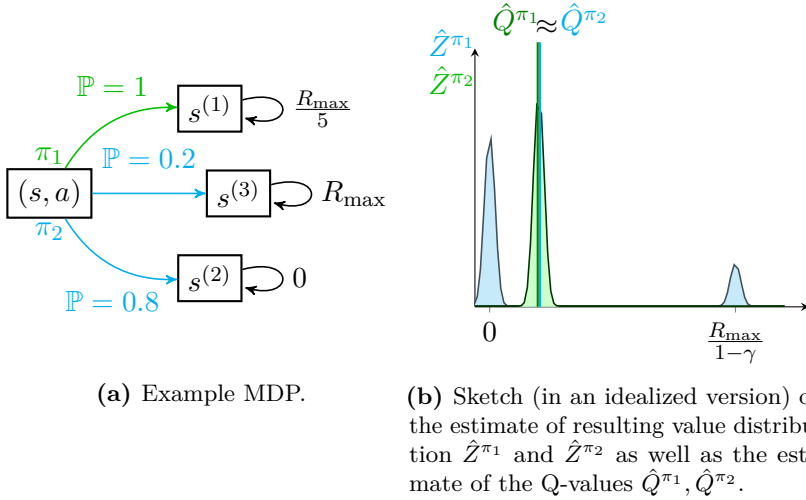
Figure 4.3: For two policies illustrated on Fig (a), the illustration on Fig (b) gives the value distribution $Z^{(\pi)}(s, a)$ as compared to the expected value $Q^\pi(s, a)$. On the left figure, one can see that π_1 moves with certainty to an absorbing state with reward at every step $\frac{R_{\max}}{5}$, while π_2 moves with probability 0.2 and 0.8 to absorbing states with respectively rewards at every step $R_{\max}$ and 0. From the pair (s, a), the policies π_1 and π_2 have the same expected return but different value distributions.

4.7   Multi-step learning

In DQN, the target value used to update the Q-network parameters (given in Equation 4.3) is estimated as the sum of the immediate reward and a contribution of the following steps in the return. That contribution is estimated based on its own value estimate at the next time-step. For that reason, the learning algorithm is said to *bootstrap* as it recursively uses its own value estimates (Sutton, 1988).

This method of estimating a target value is not the only possibility. Non-bootstrapping methods learn directly from returns (Monte Carlo)

and an intermediate solution is to use a multi-step target (Sutton, 1988; Watkins, 1989; Peng and Williams, 1994; Singh and Sutton, 1996). Such a variant in the case of DQN can be obtained by using the n-step target value given by:

$$Y_k^{Q,n} = \sum_{t=0}^{n-1} \gamma^t r_t + \gamma^n \max_{a' \in A} Q(s_n, a'; \theta_k) \qquad (4.10)$$

where $(s_0, a_0, r_0, \cdots, s_{n-1}, a_{n-1}, r_{n-1}, s_n)$ is any trajectory of $n+1$ time steps with $s = s_0$ and $a = a_0$. A combination of different multi-steps targets can also be used:

$$Y_k^{Q,n} = \sum_{i=0}^{n-1} \lambda_i \left(\sum_{t=0}^{i} \gamma^t r_t + \gamma^{i+1} \max_{a' \in A} Q\left(s_{i+1}, a'; \theta_k\right) \right) \qquad (4.11)$$

with $\sum_{i=0}^{n-1} \lambda_i = 1$. In the method called $TD(\lambda)$ (Sutton, 1988), $n \to \infty$ and λ_i follow a geometric law: $\lambda_i \propto \lambda^i$ where $0 \le \lambda \le 1$.

To bootstrap or not to bootstrap? Bootstrapping has both advantages and disadvantages. On the negative side, using pure bootstrapping methods (such as in DQN) are prone to instabilities when combined with function approximation because they make recursive use of their own value estimate at the next time-step. On the contrary, methods such as n-step Q-learning rely less on their own value estimate because the estimate used is decayed by γ^n for the nth step backup. In addition, methods that rely less on bootstrapping can propagate information more quickly from delayed rewards as they learn directly from returns (Sutton, 1996). Hence they might be more computationally efficient.

Bootstrapping also has advantages. The main advantage is that using value bootstrap allows learning from off-policy samples. Indeed, methods that do not use pure bootstrapping, such as n-step Q-learning with $n > 1$ or $TD(\lambda)$, are in principle on-policy based methods that would introduce a bias when used with trajectories that are not obtained solely under the behavior policy μ (e.g., stored in a replay buffer).

The conditions required to learn efficiently and safely with eligibility traces from off-policy experience are provided by Munos *et al.*, 2016; Harutyunyan *et al.*, 2016. In the control setting, the retrace operator (Munos *et al.*, 2016) considers a sequence of target policies π that depend

on the sequence of Q-functions (such as ϵ-greedy policies), and seek to approximate Q^* (if π is greedy or becomes increasingly greedy w.r.t. the Q estimates). It leads to the following target:

$$Y = Q(s,a) + \left[\sum_{t \geq 0} \gamma^t \left(\prod_{c_s=1}^{t} c_s \right) (r_t + \gamma \mathbb{E}_\pi Q(s_{t+1}, a') - Q(s_t, a_t)) \right] \quad (4.12)$$

where $c_s = \lambda \min \left(1, \frac{\pi(s,a)}{\mu(s,a)} \right)$ with $0 \leq \lambda \leq 1$ and μ is the behavior policy (estimated from observed samples). This way of updating the Q-network has guaranteed convergence, does not suffer from a high variance and it does not cut the traces unnecessarily when π and μ are close. Nonetheless, one can note that estimating the target is more expansive to compute as compared to the one-step target (such as in DQN) because the Q-value function has to be estimated on more states.

4.8   Combination of all DQN improvements and variants of DQN

The original DQN algorithm can combine the different variants discussed in §4.4 to §4.7 (as well as some discussed in Chapter 8.1) and that has been studied by Hessel *et al.*, 2017. Their experiments show that the combination of all the previously mentioned extensions to DQN provides state-of-the-art performance on the Atari 2600 benchmarks, both in terms of sample efficiency and final performance. Overall, a large majority of Atari games can be solved such that the deep RL agents surpass the human level performance.

Some limitations remain with DQN-based approaches. Among others, these types of algorithms are not well-suited to deal with large and/or continuous action spaces. In addition, they cannot explicitly learn stochastic policies. Modifications that address these limitations will be discussed in the following Chapter 5, where we discuss policy-based approaches. Actually, the next section will also show that value-based and policy-based approaches can be seen as two facets of the same model-free approach. Therefore, the limitations of discrete action spaces and deterministic policies are only related to DQN.

One can also note that value-based or policy-based approaches do not make use of any model of the environment, which limits their sample

efficiency. Ways to combine model-free and model-based approaches will be discussed in Chapter 6.

5

Policy gradient methods for deep RL

This section focuses on a particular family of reinforcement learning algorithms that use policy gradient methods. These methods optimize a performance objective (typically the expected cumulative reward) by finding a good policy (e.g a neural network parameterized policy) thanks to variants of stochastic gradient ascent with respect to the policy parameters. Note that policy gradient methods belong to a broader class of policy-based methods that includes, among others, evolution strategies. These methods use a learning signal derived from sampling instantiations of policy parameters and the set of policies is developed towards policies that achieve better returns (e.g., Salimans *et al.*, 2017).

In this chapter, we introduce the stochastic and deterministic gradient theorems that provide gradients on the policy parameters in order to optimize the performance objective. Then, we present different RL algorithms that make use of these theorems.

5.1 Stochastic Policy Gradient

The expected return of a stochastic policy π starting from a given state s_0 from Equation 3.1 can be written as (Sutton *et al.*, 2000):

$$V^\pi(s_0) = \int_\mathcal{S} \rho^\pi(s) \int_\mathcal{A} \pi(s,a) R'(s,a) da ds, \qquad (5.1)$$

where $R'(s,a) = \int_{s' \in \mathcal{S}} T(s,a,s') R(s,a,s')$ and $\rho^\pi(s)$ is the discounted state distribution defined as

$$\rho^\pi(s) = \sum_{t=0}^\infty \gamma^t Pr\{s_t = s | s_0, \pi\}.$$

For a differentiable policy π_w, the fundamental result underlying these algorithms is the policy gradient theorem (Sutton *et al.*, 2000):

$$\nabla_w V^{\pi_w}(s_0) = \int_\mathcal{S} \rho^{\pi_w}(s) \int_\mathcal{A} \nabla_w \pi_w(s,a) Q^{\pi_w}(s,a) da ds. \qquad (5.2)$$

This result allows us to adapt the policy parameters w: $\Delta w \propto \nabla_w V^{\pi_w}(s_0)$ from experience. This result is particularly interesting since the policy gradient does not depend on the gradient of the state distribution (even though one might have expected it to). The simplest way to derive the policy gradient estimator (i.e., estimating $\nabla_w V^{\pi_w}(s_0)$ from experience) is to use a *score function gradient estimator*, commonly known as the REINFORCE algorithm (Williams, 1992). The likelihood ratio trick can be exploited as follows to derive a general method of estimating gradients from expectations:

$$\begin{aligned} \nabla_w \pi_w(s,a) &= \pi_w(s,a) \frac{\nabla_w \pi_w(s,a)}{\pi_w(s,a)} \\ &= \pi_w(s,a) \nabla_w \log(\pi_w(s,a)). \end{aligned} \qquad (5.3)$$

Considering Equation 5.3, it follows that

$$\nabla_w V^{\pi_w}(s_0) = \mathbb{E}_{s \sim \rho^{\pi_w}, a \sim \pi_w} \left[\nabla_w \left(\log \pi_w(s,a) \right) Q^{\pi_w}(s,a) \right]. \qquad (5.4)$$

Note that, in practice, most policy gradient methods effectively use undiscounted state distributions, without hurting their performance (Thomas, 2014).

So far, we have shown that policy gradient methods should include a policy evaluation followed by a policy improvement. On the one hand, the policy evaluation estimates Q^{π_w}. On the other hand, the policy improvement takes a gradient step to optimize the policy $\pi_w(s, a)$ with respect to the value function estimation. Intuitively, the policy improvement step increases the probability of the actions proportionally to their expected return.

The question that remains is how the agent can perform the policy evaluation step, i.e., how to obtain an estimate of $Q^{\pi_w}(s, a)$. The simplest approach to estimating gradients is to replace the Q function estimator with a cumulative return from entire trajectories. In the Monte-Carlo policy gradient, we estimate the $Q^{\pi_w}(s, a)$ from rollouts on the environment while following policy π_w. The Monte-Carlo estimator is an unbiased well-behaved estimate when used in conjunction with the back-propagation of a neural network policy, as it estimates returns until the end of the trajectories (without instabilities induced by bootstrapping). However, the main drawback is that the estimate requires on-policy rollouts and can exhibit high variance. Several rollouts are typically needed to obtain a good estimate of the return. A more efficient approach is to instead use an estimate of the return given by a value-based approach, as in actor-critic methods discussed in §5.3.

We make two additional remarks. First, to prevent the policy from becoming deterministic, it is common to add an entropy regularizer to the gradient. With this regularizer, the learnt policy can remain stochastic. This ensures that the policy keeps exploring.

Second, instead of using the value function Q^{π_w} in Eq. 5.4, an advantage value function A^{π_w} can also be used. While $Q^{\pi_w}(s, a)$ summarizes the performance of each action for a given state under policy π_w, the advantage function $A^{\pi_w}(s, a)$ provides a measure of comparison for each action to the expected return at the state s, given by $V^{\pi_w}(s)$. Using $A^{\pi_w}(s, a) = Q^{\pi_w}(s, a) - V^{\pi_w}(s)$ has usually lower magnitudes than $Q^{\pi_w}(s, a)$. This helps reduce the variance of the gradient estimator $\nabla_w V^{\pi_w}(s_0)$ in the policy improvement step, while not modifying the

expectation[1]. In other words, the value function $V^{\pi_w}(s)$ can be seen as a *baseline* or *control variate* for the gradient estimator. When updating the neural network that fits the policy, using such a baseline allows for improved numerical efficiency – i.e. reaching a given performance with fewer updates – because the learning rate can be bigger.

5.2 Deterministic Policy Gradient

The policy gradient methods may be extended to deterministic policies. The Neural Fitted Q Iteration with Continuous Actions (NFQCA) (Hafner and Riedmiller, 2011) and the Deep Deterministic Policy Gradient (DDPG) (Silver *et al.*, 2014; Lillicrap *et al.*, 2015) algorithms introduce the direct representation of a policy in such a way that it can extend the NFQ and DQN algorithms to overcome the restriction of discrete actions.

Let us denote by $\pi(s)$ the deterministic policy: $\pi(s) : \mathcal{S} \rightarrow \mathcal{A}$. In discrete action spaces, a direct approach is to build the policy iteratively with:

$$\pi_{k+1}(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \ Q^{\pi_k}(s, a), \tag{5.5}$$

where π_k is the policy at the kth iteration. In continuous action spaces, a greedy policy improvement becomes problematic, requiring a global maximisation at every step. Instead, let us denote by $\pi_w(s)$ a differentiable deterministic policy. In that case, a simple and computationally attractive alternative is to move the policy in the direction of the gradient of Q, which leads to the Deep Deterministic Policy Gradient (DDPG) algorithm (Lillicrap *et al.*, 2015):

$$\nabla_w V^{\pi_w}(s_0) = \mathbb{E}_{s \sim \rho^{\pi_w}} \left[\nabla_w (\pi_w) \nabla_a (Q^{\pi_w}(s, a)) |_{a = \pi_w(s)} \right]. \tag{5.6}$$

This equation implies relying on $\nabla_a (Q^{\pi_w}(s, a))$ (in addition to $\nabla_w \pi_w$), which usually requires using actor-critic methods (see §5.3).

---

[1]Indeed, subtracting a baseline that only depends on s to $Q^{\pi_w}(s, a)$ in Eq. 5.2 does not change the gradient estimator because $\forall s, \int_{\mathcal{A}} \nabla_w \pi_w(s, a) da = 0$.

5.3   Actor-Critic Methods

As we have seen in §5.1 and §5.2, a policy represented by a neural
network can be updated by gradient ascent for both the deterministic
and the stochastic case. In both cases, the policy gradient typically
requires an estimate of a value function for the current policy. One
common approach is to use an actor-critic architecture that consists of
two parts: an actor and a critic (Konda and Tsitsiklis, 2000). The actor
refers to the policy and the critic to the estimate of a value function (e.g.,
the Q-value function). In deep RL, both the actor and the critic can be
represented by non-linear neural network function approximators (Mnih
et al., 2016). The actor uses gradients derived from the policy gradient
theorem and adjusts the policy parameters w. The critic, parameterized
by θ, estimates the approximate value function for the current policy π:
$Q(s, a; \theta) \approx Q^{\pi}(s, a)$.

The critic

From a (set of) tuples $< s, a, r, s' >$, possibly taken from a replay
memory, the simplest off-policy approach to estimating the critic is to
use a pure bootstrapping algorithm $TD(0)$ where, at every iteration,
the current value $Q(s, a; \theta)$ is updated towards a target value:

$$Y_k^Q = r + \gamma Q(s', a = \pi(s'); \theta) \qquad (5.7)$$

This approach has the advantage of being simple, yet it is not
computationally efficient as it uses a pure bootstrapping technique that
is prone to instabilities and has a slow reward propagation backwards
in time (Sutton, 1996). This is similar to the elements discussed in the
value-based methods in §4.7.

   The ideal is to have an architecture that is

- sample-efficient such that it should be able to make use of both
  off-policy and and on-policy trajectories (i.e., it should be able to
  use a replay memory), and

- computationally efficient: it should be able to profit from the
  stability and the fast reward propagation of on-policy methods
  for samples collected from near on-policy behavior policies.

There are many methods that combine on- and off-policy data for policy evaluation (Precup, 2000). The algorithm *Retrace*(λ) (Munos *et al.*, 2016) has the advantages that (i) it can make use of samples collected from any behavior policy without introducing a bias and (ii) it is efficient as it makes the best use of samples collected from near on-policy behavior policies. That approach was used in actor-critic architectures described by Wang *et al.* (2016b) and Gruslys *et al.* (2017). These architectures are sample-efficient thanks to the use of a replay memory, and computationally efficient since they use multi-step returns which improves the stability of learning and increases the speed of reward propagation backwards in time.

The actor

From Equation 5.4, the off-policy gradient in the policy improvement phase for the stochastic case is given as:

$$\nabla_w V^{\pi_w}(s_0) = \mathbb{E}_{s \sim \rho^{\pi_\beta}, a \sim \pi_\beta} \left[\nabla_\theta \left(\log \ \pi_w(s,a) \right) Q^{\pi_w}(s,a) \right]. \qquad (5.8)$$

where β is a behavior policy generally different than π, which makes the gradient generally biased. This approach usually behaves properly in practice but the use of a biased policy gradient estimator makes difficult the analysis of its convergence without the GLIE assumption (Munos *et al.*, 2016; Gruslys *et al.*, 2017)[2].

In the case of actor-critic methods, an approach to perform the policy gradient on-policy without experience replay has been investigated with the use of asynchronous methods, where multiple agents are executed in parallel and the actor-learners are trained asynchronously (Mnih *et al.*, 2016). The parallelization of agents also ensures that each agent experiences different parts of the environment at a given time step. In that case, n-step returns can be used without introducing a bias. This simple idea can be applied to any learning algorithm that requires

---

[2]Greedy in the Limit with Infinite Exploration (GLIE) means that the behavior policies are required to become greedy (no exploration) in the limit of an online learning setting where the agent has gathered an infinite amount of experience. It is required that « (i) each action is executed infinitely often in every state that is visited infinitely often, and (ii) in the limit, the learning policy is greedy with respect to the Q-value function with probability 1 »(Singh *et al.*, 2000).

on-policy data and it removes the need to maintain a replay buffer. However, this asynchronous trick is not sample efficient.

An alternative is to combine off-policy and on-policy samples to trade-off both the sample efficiency of off-policy methods and the stability of on-policy gradient estimates. For instance, Q-Prop (Gu *et al.*, 2017b) uses a Monte Carlo on-policy gradient estimator, while reducing the variance of the gradient estimator by using an off-policy critic as a control variate. One limitation of Q-Prop is that it requires using on-policy samples for estimating the policy gradient.

5.4   Natural Policy Gradients

Natural policy gradients are inspired by the idea of natural gradients for the updates of the policy. Natural gradients can be traced back to the work of Amari, 1998 and has been later adapted to reinforcement learning (Kakade, 2001).

Natural policy gradient methods use the steepest direction given by the Fisher information metric, which uses the manifold of the objective function. In the simplest form of steepest ascent for an objective function $J(w)$, the update is of the form $\triangle w \propto \nabla_w J(w)$. In other words, the update follows the direction that maximizes $(J(w) - J(w + \triangle w))$ under a constraint on $|| \triangle w ||_2$. In the hypothesis that the constraint on $\triangle w$ is defined with another metric than L_2, the first-order solution to the constrained optimization problem typically has the form $\triangle w \propto B^{-1} \nabla_w J(w)$ where B is an $n_w \times n_w$ matrix. In natural gradients, the norm uses the Fisher information metric, given by a local quadratic approximation to the KL divergence $D_{KL}(\pi^w || \pi^{w + \triangle w})$. The natural gradient ascent for improving the policy π_w is given by

$$\triangle w \propto F_w^{-1} \nabla_w V^{\pi_w}(\cdot), \tag{5.9}$$

where F_w is the Fisher information matrix given by

$$F_w = \mathbb{E}_{\pi_w}[\nabla_w \log \pi_w(s, \cdot)(\nabla_w \log \pi_w(s, \cdot))^T]. \tag{5.10}$$

Policy gradients following $\nabla_w V^{\pi_w}(\cdot)$ are often slow because they are prone to getting stuck in local plateaus. Natural gradients, however, do not follow the usual steepest direction in the parameter space, but the

steepest direction with respect to the Fisher metric. Note that, as the angle between natural and ordinary gradient is never larger than ninety degrees, convergence is also guaranteed when using natural gradients.

The caveat with natural gradients is that, in the case of neural networks and their large number of parameters, it is usually impractical to compute, invert, and store the Fisher information matrix (Schulman *et al.*, 2015). This is the reason why natural policy gradients are usually not used in practice for deep RL; however alternatives inspired by this idea have been found and they are discussed in the following section.

5.5 Trust Region Optimization

As a modification to the natural gradient method, policy optimization methods based on a *trust region* aim at improving the policy while changing it in a controlled way. These constraint-based policy optimization methods focus on restricting the changes in a policy using the KL divergence between the action distributions. By bounding the size of the policy update, trust region methods also bound the changes in state distributions guaranteeing improvements in policy.

TRPO (Schulman *et al.*, 2015) uses constrained updates and advantage function estimation to perform the update, resulting in the reformulated optimization given by

$$\max_{\triangle w} \mathbb{E}_{s\sim\rho^{\pi_w},a\sim\pi_w} \left[\frac{\pi_{w+\triangle w}(s,a)}{\pi_w(s,a)} A^{\pi_w}(s,a) \right] \qquad (5.11)$$

subject to $\mathbb{E}\, D_{\mathrm{KL}}\left(\pi_w(s,\cdot)||\pi_{w+\triangle w}(s,\cdot)\right) \leq \delta$, where $\delta \in \mathbb{R}$ is a hyperparameter. From empirical data, TRPO uses a conjugate gradient with KL constraint to optimize the objective function.

Proximal Policy Optimization (PPO) (Schulman *et al.*, 2017b) is a variant of the TRPO algorithm, which formulates the constraint as a penalty or a clipping objective, instead of using the KL constraint. Unlike TRPO, PPO considers modifying the objective function to penalize changes to the policy that move $r_t(w) = \frac{\pi_{w+\triangle w}(s,a)}{\pi_w(s,a)}$ away from 1. The clipping objective that PPO maximizes is given by

$$\mathbb{E}_{s\sim\rho^{\pi_w},a\sim\pi_w} \left[\min\left(r_t(w)A^{\pi_w}(s,a), \mathrm{clip}(r_t(w), 1-\epsilon, 1+\epsilon)A^{\pi_w}(s,a)\right) \right]$$
$$(5.12)$$

where $\epsilon \in \mathbb{R}$ is a hyperparameter. This objective function clips the probability ratio to constrain the changes of r_t in the interval $[1 - \epsilon, 1 + \epsilon]$.

5.6   Combining policy gradient and Q-learning

Policy gradient is an efficient technique for improving a policy in a reinforcement learning setting. As we have seen, this typically requires an estimate of a value function for the current policy and a sample efficient approach is to use an actor-critic architecture that can work with off-policy data.

These algorithms have the following properties unlike the methods based on DQN discussed in Chapter 4:

- They are able to work with continuous action spaces. This is particularly interesting in applications such as robotics, where forces and torques can take a continuum of values.

- They can represent stochastic policies, which is useful for building policies that can explicitly explore. This is also useful in settings where the optimal policy is a stochastic policy (e.g., in a multi-agent setting where the Nash equilibrium is a stochastic policy).

However, another approach is to combine policy gradient methods directly with off-policy Q-learning (O'Donoghue *et al.*, 2016). In some specific settings, depending on the loss function and the entropy regularization used, value-based methods and policy-based methods are equivalent (Fox *et al.*, 2015; O'Donoghue *et al.*, 2016; Haarnoja *et al.*, 2017; Schulman *et al.*, 2017a). For instance, when adding an entropy regularization, Eq. 5.4 can be written as

$$\nabla_w V^{\pi_w}(s_0) = \mathbb{E}_{s,a} \left[\nabla_w \left(\log \ \pi_w(s,a) \right) Q^{\pi_w}(s,a) \right] + \alpha \mathbb{E}_s \nabla_w H^{\pi_w}(s). \tag{5.13}$$

where $H^\pi(s) = -\sum_a \pi(s,a) \log \pi(s,a)$. From this, one can note that an optimum is satisfied by the following policy: $\pi_w(s,a) = exp(A^{\pi_w}(s,a)/\alpha - H^{\pi_w}(s))$. Therefore, we can use the policy to derive an estimate of the advantage function: $\tilde{A}^{\pi_w}(s,a) = \alpha(\log \pi_w(s,a) + H^\pi(s))$.

We can thus think of all model-free methods as different facets of the same approach.

One remaining limitation is that both value-based and policy-based methods are model-free and they do not make use of any model of the environment. The next chapter describes algorithms with a model-based approach.

6

Model-based methods for deep RL

In Chapters 4 and 5, we have discussed the model-free approach that rely either on a value-based or a policy-based method. In this chapter, we introduce the model-based approach that relies on a model of the environment (dynamics and reward function) in conjunction with a planning algorithm. In §6.2, the respective strengths of the model-based versus the model-free approaches are discussed, along with how the two approaches can be integrated.

6.1 Pure model-based methods

A model of the environment is either explicitly given (e.g., in the game of Go for which all the rules are known a priori) or learned from experience. To learn the model, yet again function approximators bring significant advantages in high-dimensional (possibly partially observable) environments (Oh *et al.*, 2015; Mathieu *et al.*, 2015; Finn *et al.*, 2016a; Kalchbrenner *et al.*, 2016; Duchesne *et al.*, 2017; Nagabandi *et al.*, 2018). The model can then act as a proxy for the actual environment.

When a model of the environment is available, planning consists in interacting with the model to recommend an action. In the case of discrete actions, lookahead search is usually done by generating

potential trajectories. In the case of a continuous action space, trajectory optimization with a variety of controllers can be used.

6.1.1 Lookahead search

A lookahead search in an MDP iteratively builds a decision tree where the current state is the root node. It stores the obtained returns in the nodes and focuses attention on promising potential trajectories. The main difficulty in sampling trajectories is to balance *exploration* and *exploitation*. On the one hand, the purpose of exploration is to gather more information on the part of the search tree where few simulations have been performed (i.e., where the expected value has a high variance). On the other hand, the purpose of exploitation is to refine the expected value of the most promising moves.

Monte-Carlo tree search (MCTS) techniques (Browne *et al.*, 2012) are popular approaches to lookahead search. Among others, they have gained popularity thanks to prolific achievements in the challenging task of computer Go (Brügmann, 1993; Gelly *et al.*, 2006; Silver *et al.*, 2016a). The idea is to sample multiple trajectories from the current state until a terminal condition is reached (e.g., a given maximum depth) (see Figure 6.1 for an illustration). From those simulation steps, the MCTS algorithm then recommends an action to take.

Recent works have developed strategies to directly learn end-to-end the model, along with how to make the best use of it, without relying on explicit tree search techniques (Pascanu *et al.*, 2017). These approaches show improved sample efficiency, performance, and robustness to model misspecification compared to the separated approach (simply learning the model and then relying on it during planning).

6.1.2 Trajectory optimization

Lookahead search techniques are limited to discrete actions, and alternative techniques have to be used for the case of continuous actions. If the model is differentiable, one can directly compute an analytic policy gradient by backpropagation of rewards along trajectories (Nguyen and Widrow, 1990). For instance, PILCO (Deisenroth and Rasmussen, 2011) uses Gaussian processes to learn a probabilistic model of the
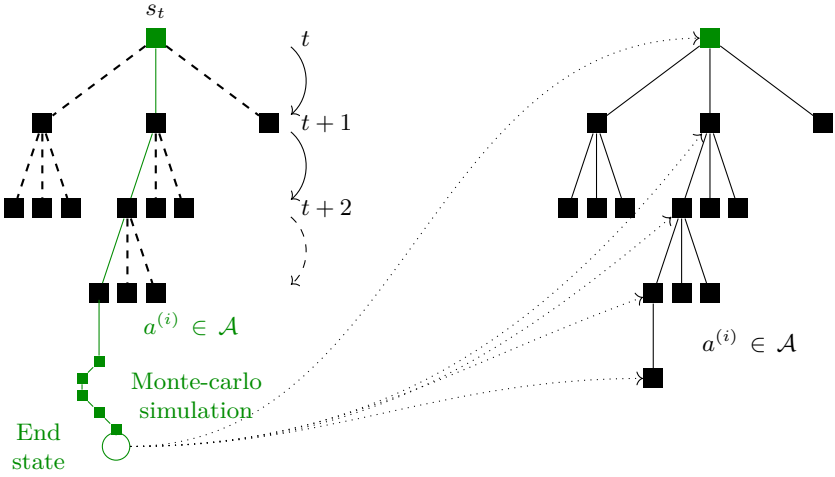
Figure 6.1: Illustration of how a MCTS algorithm performs a Monte-Carlo simulation and builds a tree by updating the statistics of the different nodes. Based on the statistics gathered for the current node s_t, the MCTS algorithm chooses an action to perform on the actual environment.

dynamics. It can then explicitly use the uncertainty for planning and policy evaluation in order to achieve a good sample efficiency. However, the gaussian processes have not been able to scale reliably to high-dimensional problems.

One approach to scale planning to higher dimensions is to aim at leveraging the generalization capabilities of deep learning. For instance, Wahlström *et al.* (2015) uses a deep learning model of the dynamics (with an auto-encoder) along with a model in a latent state space. Model-predictive control (Morari and Lee, 1999) can then be used to find the policy by repeatedly solving a finite-horizon optimal control problem in the latent space. It is also possible to build a probabilistic generative model in a latent space with the objective that it possesses a locally linear dynamics, which allows control to be performed more efficiently (Watter *et al.*, 2015). Another approach is to use the trajectory optimizer as a teacher rather than a demonstrator: guided policy search (Levine and Koltun, 2013) takes a few sequences of actions suggested by another controller. iIt then learns to adjust the policy from these sequences. Methods that leverage trajectory optimization have

demonstrated many capabilities, for instance in the case of simulated 3D bipeds and quadrupeds (e.g., Mordatch *et al.*, 2015).

6.2 Integrating model-free and model-based methods

The respective strengths of the model-free versus model-based approaches depend on different factors. First, the best suited approach depends on whether the agent has access to a model of the environment. If that's not the case, the learned model usually has some inaccuracies that should be taken into account. Note that learning the model can share the hidden-state representations with a value-based approach by sharing neural network parameters (Li *et al.*, 2015).

Second, a model-based approach requires working in conjunction with a planning algorithm (or controller), which is often computationally demanding. The time constraints for computing the policy $\pi(s)$ via planning must therefore be taken into account (e.g., for applications with real-time decision-making or simply due to resource limitations).

Third, for some tasks, the structure of the policy (or value function) is the easiest one to learn, but for other tasks, the model of the environment may be learned more efficiently due to the particular structure of the task (less complex or with more regularity). Thus, the most performant approach depends on the structure of the model, policy, and value function (see the coming Chapter 7 for more details on generalization). Let us consider two examples to better understand this key consideration. In a labyrinth where the agent has full observability, it is clear how actions affect the next state and the dynamics of the model may easily be generalized by the agent from only a few tuples (for instance, the agent is blocked when trying to cross a wall of the labyrinth). Once the model is known, a planning algorithm can then be used with high performance. Let us now discuss another example where, on the contrary, planning is more difficult: an agent has to cross a road with random events happening everywhere on the road. Let us suppose that the best policy is simply to move forward except when an object has just appeared in front of the agent. In that case, the optimal policy may easily be captured by a model-free approach, while a model-based approach would be more difficult (mainly due to the stochasticity of the model which

leads to many different possible situations, even for one given sequence of actions).
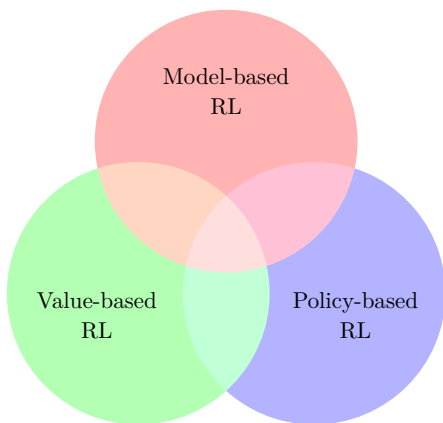
Figure 6.2: Venn diagram in the space of possible RL algorithms.

We now describe how it is possible to obtain advantages from both worlds by integrating learning and planning into one end-to-end training procedure so as to obtain an efficient algorithm both in performance (sample efficient) and in computation time. A Venn diagram of the different combinations is given in Figure 6.2.

When the model is available, one direct approach is to use tree search techniques that make use of both value and policy networks (e.g., Silver *et al.*, 2016a). When the model is not available and under the assumption that the agent has only access to a limited number of trajectories, the key property is to have an algorithm that generalizes well (see Chapter 7 for a discussion on generalization). One possibility is to build a model that is used to generate additional samples for a model-free reinforcement learning algorithm (Gu *et al.*, 2016b). Another possibility is to use a model-based approach along with a controller such as MPC to perform basic tasks and use model-free fine-tuning in order to achieve task success (Nagabandi *et al.*, 2017).

Other approaches build neural network architectures that combine both model-free and model-based elements. For instance, it is possible to combine a value function with steps of back-propagation through a model (Heess *et al.*, 2015). The VIN architecture (Tamar *et al.*, 2016)

is a fully differentiable neural network with a planning module that learns to plan from model-free objectives (given by a value function). It works well for tasks that involve planning-based reasoning (navigation tasks) from one initial position to one goal position and it demonstrates strong generalization in a few different domains.

In the same spirit, the predictron (Silver *et al.*, 2016b) is aimed at developing a more generally applicable algorithm that is effective in the context of planning. It works by implicitly learning an internal model in an abstract state space, which is used for policy evaluation. The predictron is trained end-to-end to learn, from the abstract state space, (i) the immediate reward and (ii) value functions over multiple planning depths. The predictron architecture is limited to policy evaluation, but the idea was extended to an algorithm that can learn an optimal policy in an architecture called VPN (Oh *et al.*, 2017). Since VPN relies on n-step Q-learning, it requires however on-policy data.

Other works have proposed architectures that combine model-based and model-free approaches. Schema Networks (Kansky *et al.*, 2017) learn the dynamics of an environment directly from data by enforcing some relational structure. The idea is to use a richly structured architecture such that it provides robust generalization thanks to an object-oriented approach for the model.

I2As (Weber *et al.*, 2017) does not use the model to directly perform planning but it uses the predictions as additional context in deep policy networks. The proposed idea is that I2As could learn to interpret predictions from the learned model to construct implicit plans.

TreeQN (Farquhar *et al.*, 2017) constructs a tree by recursively applying an implicit transition model in an implicitly learned abstract state space, built by estimating Q-values. Farquhar *et al.* (2017) also propose ATreeC, which is an actor-critic variant that augments TreeQN with a softmax layer to form a stochastic policy network.

The CRAR agent explicitly learns both a value function and a model via a shared low-dimensional learned encoding of the environment, which is meant to capture summarized abstractions and allow for efficient planning (François-Lavet *et al.*, 2018). By forcing an expressive representation, the CRAR approach creates an interpretable low-

dimensional representation of the environment, even far temporally from any rewards or in the absence of model-free objectives.

Improving the combination of model-free and model-based ideas is one key area of research for the future development of deep RL algorithms. We therefore expect to see smarter and richer structures in that domain.

7

The concept of generalization

Generalization is a central concept in the field of machine learning, and reinforcement learning is no exception. In an RL algorithm (model-free or model-based), generalization refers to either

- the capacity to achieve good performance in an environment where limited data has been gathered, or

- the capacity to obtain good performance in a related environment.

In the former case, the agent must learn how to behave in a test environment that is identical to the one it has been trained on. In that case, the idea of generalization is directly related to the notion of *sample efficiency* (e.g., when the state-action space is too large to be fully visited). In the latter case, the test environment has common patterns with the training environment but can differ in the dynamics and the rewards. For instance, the underlying dynamics may be the same but a transformation on the observations may have happened (e.g., noise, shift in the features, etc.). That case is related to the idea of transfer learning (discussed in §10.2) and meta-learning (discussed in §10.1.2).

Note that, in the online setting, one mini-batch gradient update is usually done at every step. In that case, the community has also used the term sample efficiency to refer to how fast the algorithm learns, which is measured in terms of performance for a given number of steps (number of learning steps=number of transitions observed). However, in that context, the result depends on many different elements. It depends on the learning algorithm and it is, for instance, influenced by the possible variance of the target in a model-free setting. It also depends on the exploration/exploitation, which will be discussed in §8.1 (e.g, instabilities may be good). Finally, it depends on the actual generalization capabilities.

In this chapter, the goal is to study specifically the aspect of generalization. We are not interested in the number of mini-batch gradient descent steps that are required but rather in the performance that a deep RL algorithm can have in the offline case where the agent has to learn from limited data. Let us consider the case of a finite dataset D obtained on the exact same task as the test environment. Formally, a dataset available to the agent $D \sim \mathcal{D}$ can be defined as a set of four-tuples $< s, a, r, s' > \in \mathcal{S} \times \mathcal{A} \times \mathcal{R} \times \mathcal{S}$ gathered by sampling independently and identically (i.i.d.)[1]

- a given number of state-action pairs (s, a) from some fixed distribution with $\mathbb{P}(s, a) > 0$, $\forall (s, a) \in \mathcal{S} \times \mathcal{A}$,

- a next state $s' \sim T(s, a, \cdot)$, and

- a reward $r = R(s, a, s')$.

We denote by D_∞ the particular case of a dataset D where the number of tuples tends to infinity.

A learning algorithm can be seen as a mapping of a dataset D into a policy π_D (independently of whether the learning algorithm has a

---

[1]That i.i.d. assumption can, for instance, be obtained from a given distribution of initial states by following a stochastic sampling policy that ensures a non-zero probability of taking any action in any given state. That sampling policy should be followed during at least H time steps with the assumption that all states of the MDP can be reached in a number of steps smaller than H from the given distribution of initial states.

model-based or a model-free approach). In that case, we can decompose the suboptimality of the expected return as follows:

$$
\begin{aligned}
\mathop{\mathbb{E}}_{D\sim\mathcal{D}}\left[V^{\pi^*}(s) - V^{\pi_D}(s)\right] &= \mathop{\mathbb{E}}_{D\sim\mathcal{D}}\left[V^{\pi^*}(s) - V^{\pi_{D\infty}}(s) + V^{\pi_{D\infty}}(s) - V^{\pi_D}(s)\right] \\
&= \underbrace{(V^{\pi^*}(s) - V^{\pi_{D\infty}}(s))}_{\text{asymptotic bias}} \\
&\quad + \underbrace{\mathop{\mathbb{E}}_{D\sim\mathcal{D}}\left[V^{\pi_{D\infty}}(s) - V^{\pi_D}(s)\right]}_{\text{error due to finite size of the dataset } D} \quad .
\end{aligned}
$$

$$(7.1)$$

This decomposition highlights two different terms: (i) an asymptotic bias which is independent of the quantity of data and (ii) an overfitting term directly related to the fact that the amount of data is limited. The goal of building a policy π_D from a dataset D is to obtain the lowest overall suboptimality. To do so, the RL algorithm should be well adapted to the task (or the set of tasks).

In the previous section, two different types of approaches (model-based and model-free) have been discussed, as well as how to combine them. We have discussed the algorithms that can be used for different approaches but we have in fact left out many important elements that have an influence on the bias-overfitting tradeoff (e.g., Zhang *et al.*, 2018c; Zhang *et al.*, 2018a for illustrations of overfitting in deep RL).

As illustrated in Figure 7.1, improving generalization can be seen as a tradeoff between (i) an error due to the fact that the algorithm trusts completely the frequentist assumption (i.e., discards any uncertainty on the limited data distribution) and (ii) an error due to the bias introduced to reduce the risk of overfitting. For instance, the function approximator can be seen as a form of structure introduced to force some generalization, at the risk of introducing a bias. When the quality of the dataset is low, the learning algorithm should favor more robust policies (i.e., consider a smaller class of policies with stronger generalization capabilities). When the quality of the dataset increases, the risk of overfitting is lower and the learning algorithm can trust the data more, hence reducing the asymptotic bias.

Figure 7.1: Schematic representation of the bias-overfitting tradeoff.

As we will see, for many algorithmic choices, there is in practice a tradeoff to be made between asymptotic bias and overfitting that we simply call "bias-overfitting tradeoff". In this section, we discuss the following key elements that are at stake when one wants to improve generalization in deep RL:

- the state representation,

- the learning algorithm (type of function approximator and model-free vs model-based),

- the objective function (e.g., reward shaping, tuning the training discount factor), and

- using hierarchical learning.

Throughout those discussions, a simple example is considered. This example is, by no means, representative of the complexity of real-world problems but it is enlightening to simply illustrate the concepts that will be discussed. Let us consider an MDP with $N_\mathcal{S}$ states, $N_\mathcal{S} = 11$ and $N_\mathcal{A}$ actions, $N_\mathcal{A} = 4$. Let us suppose that the main part of the environment is a square 3×3 grid world (each represented by a tuple (x, y) with $x = \{0, 1, 2\}, y = \{0, 1, 2\}$), such as illustrated in Figure 7.2. The agent starts in the central state $(1, 1)$. In every state, it selects one of the 4 actions corresponding to 4 cardinal directions (up, down, left and right), which leads the agent to transition deterministically in a state immediately next to it, except when it tries to move out of the domain. On the upper part and lower part of the domain, the agent is

stuck in the same state if it tries to move out of the domain. On the left, the agent transitions deterministically to a given state, which will provide a reward of 0.6 for any action at the next time step. On the right side of the square, the agent transitions with a probability 25% to another state that will provide, at the next time step, a reward of 1 for any action (the rewards are 0 for all other states). When a reward is obtained, the agent transitions back to the central state.
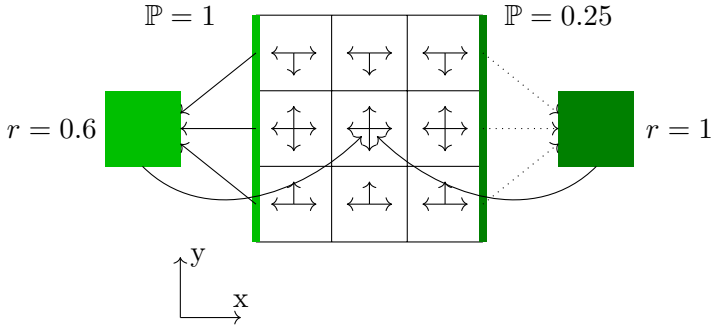
Figure 7.2: Representation of a simple MDP that illustrates the need of generalization.

In this example, if the agent has perfect knowledge of its environment, the best expected cumulative reward (for a discount factor close to 1) would be to always go to the left direction and repeatedly gather a reward of 0.6 every 3 steps (as compared to gathering a reward of 1 every 6 steps on average). Let us now suppose that only limited information has been obtained on the MDP with only one tuple of experience $< s, a, r, s' >$ for each couple $< s, a >$. According to the limited data in the frequentist assumption, there is a rather high probability ($\sim 58\%$) that at least one transition from the right side seems to provide a deterministic access to $r = 1$. In those cases and for either a model-based or a model-free approach, if the learning algorithm comes up with the optimal policy in an empirical MDP built from the frequentist statistics, it would actually suffer from poor generalization as it would choose to try to obtain the reward $r = 1$.

We discuss hereafter the different aspects that can be used to avoid overfitting to limited data; we show that it is done by favoring robust

policies within the policy class, usually at the expense of introducing some bias. At the end, we also discuss how the bias-overfitting tradeoff can be used in practice to obtain the best performance from limited data.

7.1   Feature selection

The idea of selecting the right features for the task at hand is key in the whole field of machine learning and also highly prevalent in reinforcement learning (see e.g., Munos and Moore, 2002; Ravindran and Barto, 2004; Leffler *et al.*, 2007; Kroon and Whiteson, 2009; Dinculescu and Precup, 2010; Li *et al.*, 2011; Ortner *et al.*, 2014; Mandel *et al.*, 2014; Jiang *et al.*, 2015a; Guo and Brunskill, 2017; François-Lavet *et al.*, 2017). The appropriate level of abstraction plays a key role in the bias-overfitting tradeoff and one of the key advantages of using a small but rich abstract representation is to allow for improved generalization.

Overfitting   When considering many features on which to base the policy (in the example the y-coordinate of the state as illustrated in Figure 7.3), an RL algorithm may take into consideration spurious correlations, which leads to overfitting (in the example, the agent may infer that the y-coordinate changes something to the expected return because of the limited data).

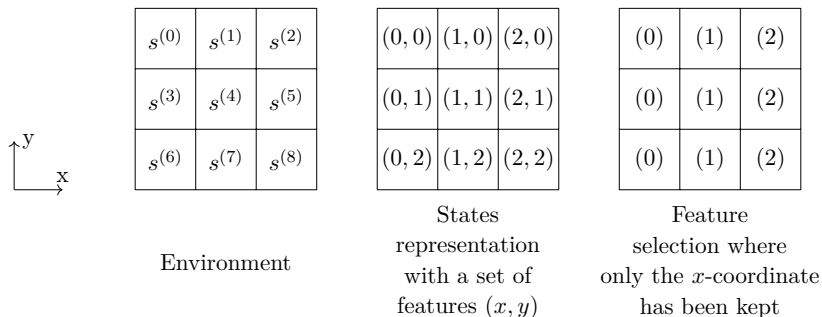

Figure 7.3: Illustration of the state representation and feature selection process. In this case, after the feature selection process, all states with the same x-coordinate are considered as indistinguishable.

Asymptotic bias   Removing features that discriminate states with a very different role in the dynamics introduces an asymptotic bias. Indeed, the same policy would be enforced on undistinguishable states, hence leading to a sub-optimal policy.

In deep RL, one approach is to first infer a factorized set of generative factors from the observations. This can be done for instance with an encoder-decoder architecture variant (Higgins *et al.*, 2017; Zhang *et al.*, 2018b). These features can then be used as inputs to a reinforcement learning algorithm. The learned representation can, in some contexts, greatly help for generalization as it provides a more succinct representation that is less prone to overfitting. However, an auto-encoder is often too strong of a constraint. On the one hand, some features may be kept in the abstract representation because they are important for the reconstruction of the observations, though they are otherwise irrelevant for the task at hand (e.g., the color of the cars in a self-driving car context). On the other hand, crucial information about the scene may also be discarded in the latent representation, particularly if that information takes up a small proportion of the observations x in pixel space (Higgins *et al.*, 2017). Note that in the deep RL setting, the abstraction representation is intertwined with the use of deep learning. This is discussed in detail in the following section.

7.2 Choice of the learning algorithm and function approximator selection

The function approximator in deep learning characterizes how the features will be treated into higher levels of abstraction (a fortiori it can thus give more or less weight to some features). If there is, for instance, an attention mechanism in the first layers of a deep neural network, the mapping made up of those first layers can be seen as a feature selection mechanism.

On the one hand, if the function approximator used for the value function and/or the policy and/or the model is too simple, an asymptotic bias may appear. On the other hand, when the function approximator has poor generalization, there will be a large error due to the finite size

of the dataset (overfitting). In the example above, a particularly good choice of a model-based or model-free approach associated with a good choice of a function approximator could infer that the y-coordinate of the state is less important than the x-coordinate, and generalize that to the policy.

Depending on the task, finding a performant function approximator is easier in either a model-free or a model-based approach. The choice of relying more on one or the other approach is thus also a crucial element to improve generalization, as discussed in §6.2.

One approach to mitigate non-informative features is to force the agent to acquire a set of symbolic rules adapted to the task and to reason on a more abstract level. This abstract level reasoning and the improved generalization have the potential to induce high-level cognitive functions such as transfer learning and analogical reasoning (Garnelo *et al.*, 2016). For instance, the function approximator may embed a relational learning structure (Santoro *et al.*, 2017) and thus build on the idea of relational reinforcement learning (Džeroski *et al.*, 2001).

7.2.1   Auxiliary tasks

In the context of deep reinforcement learning, it was shown by Jaderberg *et al.* (2016) that augmenting a deep reinforcement learning agent with auxiliary tasks within a jointly learned representation can drastically improve sample efficiency in learning. This is done by maximizing simultaneously many pseudo-reward functions such as immediate reward prediction ($\gamma = 0$), predicting pixel changes in the next observation, or predicting activation of some hidden unit of the agent's neural network. The argument is that learning related tasks introduces an inductive bias that causes a model to build features in the neural network that are useful for the range of tasks (Ruder, 2017). Hence, this formation of more significant features leads to less overfitting.

In deep RL, it is possible to build an abstract state such that it provides sufficient information for simultaneously fitting an internal meaningful dynamics as well as the estimation of the expected value of an optimal policy. By explicitly learning both the model-free and model-based components through the state representation, along with an

approximate entropy maximization penalty, the CRAR agent (François-Lavet *et al.*, 2018) shows how it is possible to learn a low-dimensional representation of the task. In addition, this approach can directly make use of a combination of model-free and model-based, with planning happening in a smaller latent state space.

7.3 Modifying the objective function

In order to improve the policy learned by a deep RL algorithm, one can optimize an objective function that diverts from the actual objective. By doing so, a bias is usually introduced but this can in some cases help with generalization. The main approaches to modify the objective function are either (i) to modify the reward of the task to ease learning (reward shaping), or (ii) tune the discount factor at training time.

7.3.1 Reward shaping

Reward shaping is a heuristic for faster learning. In practice, reward shaping uses prior knowledge by giving intermediate rewards for actions that lead to desired outcome. It is usually formalized as a function $F(s, a, s')$ added to the original reward function $R(s, a, s')$ of the original MDP (Ng *et al.*, 1999). This technique is often used in deep reinforcement learning to improve the learning process in settings with sparse and delayed rewards (e.g., Lample and Chaplot, 2017).

7.3.2 Discount factor

When the model available to the agent is estimated from data, the policy found using a shorter planning horizon can actually be better than a policy learned with the true horizon (Petrik and Scherrer, 2009; Jiang *et al.*, 2015b). On the one hand, artificially reducing the planning horizon leads to a bias since the objective function is modified. On the other hand, if a long planning horizon is targeted (the discount factor γ is close to 1), there is a higher risk of overfitting. This overfitting can intuitively be understood as linked to the accumulation of the errors in the transitions and rewards estimated from data as compared to the actual transition and reward probabilities. In the example above

(Figure 7.2), in the case where the upper right or lower right states would seem to lead deterministically to $r = 1$ from the limited data, one may take into account that it requires more steps and thus more uncertainty on the transitions (and rewards). In that context, a low training discount factor would reduce the impact of rewards that are temporally distant. In the example, a discount factor close to 0 would discount the estimated rewards at three time steps much more strongly than the rewards two time steps away, hence practically discarding the potential rewards that can be obtained by going through the corners as compared to the ones that only require moving along the x-axis.

In addition to the bias-overfitting tradeoff, a high discount factor also requires specific care in value iteration algorithms as it can lead to instabilities in convergence. This effect is due to the mappings used in the value iteration algorithms with bootstrapping (e.g., Equation 4.2 for the Q-learning algorithm) that propagate errors more strongly with a high discount factor. This issue is discussed by Gordon (1999) with the notion of *non-expansion/expansion mappings*. When bootstrapping is used in a deep RL value iteration algorithm, the risk of instabilities and overestimation of the value function is empirically stronger for a discount factor close to one (François-Lavet *et al.*, 2015).

7.4   Hierarchical learning

The possibility of learning temporally extended actions (as opposed to atomic actions that last for one time-step) has been formalized under the name of options (Sutton *et al.*, 1999). Similar ideas have also been denoted in the literature as macro-actions (McGovern *et al.*, 1997) or abstract actions (Hauskrecht *et al.*, 1998). The usage of options is an important challenge in RL because it is essential when the task at hand requires working on long time scales while developing generalization capabilities and easier transfer learning between the strategies. A few recent works have brought interesting results in the context of fully differentiable (hence learnable in the context of deep RL) options discovery. In the work of Bacon *et al.*, 2016, an *option-critic* architecture is presented with the capability of learning simultaneously the internal policies and the termination conditions of

options, as well as the policy over options. In the work of Vezhnevets *et al.*, 2016, the deep recurrent neural network is made up of two main elements. The first module generates an action-plan (stochastic plan of future actions) while the second module maintains a commitment-plan which determines when the action-plan has to be updated or terminated. Many variations of these approaches are also of interest (e.g., Kulkarni *et al.*, 2016; Mankowitz *et al.*, 2016). Overall, building a learning algorithm that is able to do hierarchical learning can be a good way of constraining/favoring some policies that have interesting properties and thus improving generalization.

7.5 How to obtain the best bias-overfitting tradeoff

From the previous sections, it is clear that there is a large variety of algorithmic choices and parameters that have an influence on the bias-overfitting tradeoff (including the choice of approach between model-based and model-free). An overall combination of all these elements provides a low overall sub-optimality.

For a given algorithmic parameter setting and keeping all other things equal, the right level of complexity is the one at which the increase in bias is equivalent to the reduction of overfitting (or the increase in overfitting is equivalent to the reduction of bias). However, in practice, there is usually not an analytical way to find the right tradeoffs to be made between all the algorithmic choices and parameters. Still, there are a variety of practical strategies that can be used. We now discuss them for the batch setting case and the online setting case.

7.5.1 Batch setting

In the batch setting case, the selection of the policy parameters to effectively balance the bias-overfitting tradeoff can be done similarly to that in supervised learning (e.g., cross-validation) as long as the performance criterion can be estimated from a subset of the trajectories from the dataset D not used during training (i.e., a validation set).

One approach is to fit an MDP model to the data via regression (or simply use the frequentist statistics for finite state and action space).

The empirical MDP can then be used to evaluate the policy. This purely model-based estimator has alternatives that do not require fitting a model. One possibility is to use a policy evaluation step obtained by generating artificial trajectories from the data, without explicitly referring to a model, thus designing a Model-free Monte Carlo-like (MFMC) estimator (Fonteneau *et al.*, 2013). Another approach is to use the idea of *importance sampling* that lets us obtain an estimate of $V^\pi(s)$ from trajectories that come from a behavior policy $\beta \neq \pi$, where β is assumed to be known (Precup, 2000). That approach is unbiased but the variance usually grows exponentially in horizon, which renders the method unsuitable when the amount of data is low. A mix of the regression-based approach and the importance sampling approach is also possible (Jiang and Li, 2016; Thomas and Brunskill, 2016), and the idea is to use a *doubly-robust estimator* that is both unbiased and with a lower variance than the importance sampling estimators.

Note that there exists a particular case where the environment's dynamics are known to the agent, but contain a dependence on an exogenous time series (e.g., trading in energy markets, weather-dependent dynamics) for which the agent only has finite data. In that case, the exogenous signal can be broken down in training time series and validation time series (François-Lavet *et al.*, 2016b). This allows training on the environment with the training time series and this allows estimating any policy on the environment with the validation time series.

7.5.2   Online setting

In the online setting, the agent continuously gathers new experience. The bias-overfitting tradeoff still plays a key role at each stage of the learning process in order to achieve good sampling efficiency. Indeed, a performant policy from given data is part of the solution to an efficient exploration/exploitation tradeoff. For that reason, progressively fitting a function approximator as more data becomes available can in fact be understood as a way to obtain a good bias-overfitting tradeoff throughout learning. With the same logic, progressively increasing the discount factor allows optimizing the bias-overfitting tradeoff through

learning (François-Lavet *et al.*, 2015). Besides, optimizing the bias-overfitting tradeoff also suggests the possibility to dynamically adapt the feature space and/or the function approximator. For example, this can be done through ad hoc regularization, or by adapting the neural network architecture, using for instance the NET2NET transformation (Chen *et al.*, 2015).

8

Particular challenges in the online setting

As discussed in the introduction, reinforcement learning can be used in two main settings: (i) the batch setting (also called offline setting), and (ii) the online setting. In a batch setting, the whole set of transitions (s, a, r, s') to learn the task is fixed. This is in contrast to the *online* setting where the agent can gather new experience gradually. In the online setting, two specific elements have not yet been discussed in depth. First, the agent can influence how to gather experience so that it is the most useful for learning. This is the *exploration/exploitation* dilemma that we discuss in Section 8.1. Second, the agent has the possibility to use a replay memory (Lin, 1992) that allows for a good data-efficiency. We discuss in Section 8.2 what experience to store and how to reprocess that experience.

8.1 Exploration/Exploitation dilemma

The exploration-exploitation dilemma is a well-studied tradeoff in RL (e.g., Thrun, 1992). Exploration is about obtaining information about the environment (transition model and reward function) while exploitation is about maximizing the expected return given the current knowledge. As an agent starts accumulating knowledge about its environment, it

has to make a tradeoff between learning more about its environment (exploration) or pursuing what seems to be the most promising strategy with the experience gathered so far (exploitation).

8.1.1   Different settings in the exploration/exploitation dilemma

There exist mainly two different settings. In the first setting, the agent is expected to perform well without a separate training phase. Thus, an explicit tradeoff between exploration versus exploitation appears so that the agent should explore only when the learning opportunities are valuable enough for the future to compensate what direct exploitation can provide. The sub-optimality $\mathbb{E}_{s_0} V^*(s_0) - V^\pi(s_0)$ of an algorithm obtained in this context is known as the *cumulative regret* [1]. The deep RL community is usually not focused on this case, except when explicitly stated such as in the works of Wang *et al.* (2016a) and Duan *et al.* (2016b).

In the more common setting, the agent is allowed to follow a *training policy* during a first phase of interactions with the environment so as to accumulate training data and hence learn a *test policy*. In the training phase, exploration is only constrained by the interactions it can make with the environment (e.g., a given number of interactions). The test policy should then be able to maximize a cumulative sum of rewards in a separate phase of interaction. The sub-optimality $\mathbb{E}_{s_0} V^*(s_0) - V^\pi(s_0)$ obtained in this case of setting is known as the *simple regret*. Note that an implicit exploration/exploitation is still important. On the one hand, the agent has to ensure that the lesser-known parts of the environment are not promising (exploration). On the other hand, the agent is interested in gathering experience in the most promising parts of the environment (which relates to exploitation) to refine the knowledge of the dynamics. For instance, in the bandit task provided in Figure 8.1, it should be clear with only a few samples that the option on the right is less promising and the agent should gather experience mainly on the two most promising arms to be able to discriminate the best one.

---

[1]This term is mainly used in the bandit community where the agent is in only one state and where a distribution of rewards is associated to each action; see e.g., Bubeck *et al.*, 2011.
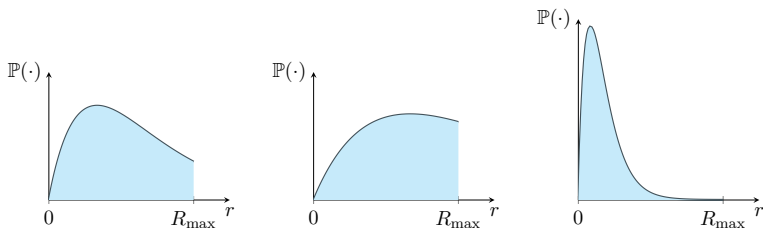
Figure 8.1: Illustration of the reward probabilities of 3 arms in a multi-armed bandit problem.

8.1.2 Different approaches to exploration

The exploration techniques are split into two main categories: (i) directed exploration and (ii) undirected exploration (Thrun, 1992).

In the undirected exploration techniques, the agent does not rely on any exploration specific knowledge of the environment (Thrun, 1992). For instance, the technique called ϵ-greedy takes a random action with probability ϵ and follows the policy that is believed to be optimal with probability $1-\epsilon$. Other variants such as softmax exploration (also called Boltzmann exploration) takes an action with a probability that depends on the associated expected return.

Contrary to the undirected exploration, directed exploration techniques make use of a memory of the past interactions with the environment. For MDPs, directed exploration can scale polynomially with the size of the state space while undirected exploration scales in general exponentially with the size of the state space (e.g., E^3 by Kearns and Singh, 2002; R-max by Brafman and Tennenholtz, 2003; ...). Inspired by the Bayesian setting, directed exploration can be done via heuristics of exploration bonus (Kolter and Ng, 2009) or by maximizing Shannon information gains (e.g., Sun *et al.*, 2011).

Directed exploration is, however, not trivially applicable in high-dimensional state spaces (e.g., Kakade *et al.*, 2003). With the development of the generalization capabilities of deep learning, some possibilities have been investigated. The key challenge is to handle, for high-dimensional spaces, the exploration/exploitation tradeoff in a principled way – with the idea to encourage the exploration of the environment

where the uncertainty due to limited data is the highest. When rewards are not sparse, a measure of the uncertainty on the value function can be used to drive the exploration (Dearden *et al.*, 1998; Dearden *et al.*, 1999). When rewards are sparse, this is even more challenging and exploration should in addition be driven by some novelty measures on the observations (or states in a Markov setting).

Before discussing the different techniques that have been proposed in the deep RL setting, one can note that the success of the first deep RL algorithms such as DQN also come from the exploration that arises naturally. Indeed, following a simple ϵ-greedy scheme online often proves to be already relatively efficient thanks to the natural instability of the Q-network that drives exploration (see Chapter 4 for why there are instabilities when using bootstrapping in a fitted Q-learning algorithm with neural networks).

Different improvements are directly built on that observation. For instance, the method of "Bootstrapped DQN" (Osband *et al.*, 2016) makes an explicit use of randomized value functions. Along similar lines, efficient exploration has been obtained by the induced stochasticity of uncertainty estimates given by a dropout Q-network (Gal and Ghahramani, 2016) or parametric noise added to its weights (Lipton *et al.*, 2016; Plappert *et al.*, 2017; Fortunato *et al.*, 2017). One specificity of the work done by Fortunato *et al.*, 2017 is that, similarly to Bayesian deep learning, the variance parameters are learned by gradient descent from the reinforcement learning loss function.

Another common approach is to have a directed scheme thanks to *exploration rewards* given to the agent via heuristics that estimate novelty (Schmidhuber, 2010; Stadie *et al.*, 2015; Houthooft *et al.*, 2016). In (Bellemare *et al.*, 2016; Ostrovski *et al.*, 2017), an algorithm provides the notion of novelty through a pseudo-count from an arbitrary density model that provides an estimate of how many times an action has been taken in similar states. This has shown good results on one of the most difficult Atari 2600 games, Montezuma's Revenge.

In (Florensa *et al.*, 2017), useful skills are learned in pre-training environments, which can then be utilized in the actual environment to improve exploration and train a high-level policy over these skills. Similarly, an agent that learns a set of auxiliary tasks may use them to

efficiently explore its environment (Riedmiller *et al.*, 2018). These ideas are also related to the creation of options studied in (Machado *et al.*, 2017a), where it is suggested that exploration may be tackled by learning options that lead to specific modifications in the state representation derived from proto-value functions.

Exploration strategies can also make use of a model of the environment along with planning. In that case, a strategy investigated in (Salge *et al.*, 2014; Mohamed and Rezende, 2015; Gregor *et al.*, 2016; Chiappa *et al.*, 2017) is to have the agent choose a sequence of actions by planning that leads to a representation of state as different as possible to the current state. In (Pathak *et al.*, 2017; Haber *et al.*, 2018), the agent optimizes both a model of its environment and a separate model that predicts the error/uncertainty of its own model. The agent can thus seek to take actions that adversarially challenge its knowledge of the environment (Savinov *et al.*, 2018).

By providing rewards on unfamiliar states, it is also possible to explore efficiently the environments. To determine the bonus, the current observation can be compared with the observations in memory. One approach is to define the rewards based on how many environment steps it takes to reach the current observation from those in memory (Savinov *et al.*, 2018). Another approach is to use a bonus positively correlated to the error of predicting features from the observations (e.g., features given by a fixed randomly initialized neural network) (Burda *et al.*, 2018).

Other approaches require either demonstrations or guidance from human demonstrators. One line of work suggests using natural language to guide the agent by providing exploration bonuses when an instruction is correctly executed (Kaplan *et al.*, 2017). In the case where demonstrations from expert agents are available, another strategy for guiding exploration in these domains is to imitate good trajectories. In some cases, it is possible to use demonstrations from experts even when they are given in an environment setup that is not exactly the same (Aytar *et al.*, 2018).

8.2 Managing experience replay

In online learning, the agent has the possibility to use a replay memory (Lin, 1992) that allows for data-efficiency by storing the past experience of the agent in order to have the opportunity to reprocess it later. In addition, a replay memory also ensures that the mini-batch updates are done from a reasonably stable data distribution kept in the replay memory (for N_{replay} sufficiently large) which helps for convergence/stability. This approach is particularly well-suited in the case of off-policy learning as using experience from past (i.e. different) policies does not introduce any bias (usually it is even good for exploration). In that context, methods based for instance on a DQN learning algorithm or model-based learning can safely and efficiently make use of a replay memory. In an online setting, the replay memory keeps all information for the last $N_{\text{replay}} \in \mathbb{N}$ time steps, where N_{replay} is constrained by the amount of memory available.

While a replay memory allows processing the transitions in a different order than they are experienced, there is also the possibility to use prioritized replay. This allows for consideration of the transitions with a different frequency than they are experienced depending on their significance (that could be which experience to store and which ones to replay). In (Schaul *et al.*, 2015b), the prioritization increases with the magnitude of the transitions' TD error, with the aim that the "unexpected" transitions are replayed more often.

A disadvantage of prioritized replay is that, in general, it also introduces a bias; indeed, by modifying the apparent probabilities of transitions and rewards, the expected return gets biased. This can readily be understood by considering the simple example illustrated in Figure 8.2 where an agent tries to estimate the expected return for a given tuple $< s, a >$. In that example, a cumulative return of 0 is obtained with probability $1 - \epsilon$ (from next state $s^{(1)}$) while a cumulative return of $C > 0$ is obtained with probability ϵ (from next state $s^{(2)}$). In that case, using prioritized experience replay will bias the expected return towards a value higher than ϵC since any transition leading to $s^{(2)}$ will be replayed with a probability higher than ϵ.

$$T(s, a, s^{(1)}) = 1 - \epsilon$$

$$\boxed{s^{(1)}} \quad V^\pi(s^{(1)}) = 0$$

$$\boxed{s}$$

$$T(s, a, s^{(2)}) = \epsilon$$

$$\boxed{s^{(2)}} \quad V^\pi(s^{(2)}) = C > 0$$
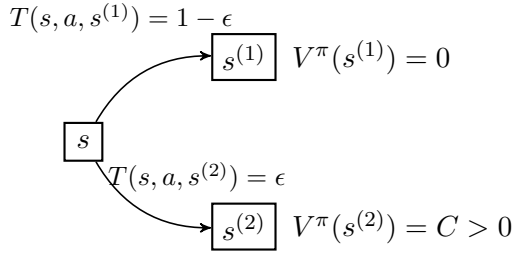
Figure 8.2: Illustration of a state s where for a given action a, the value of $Q^\pi(s, a; \theta)$ would be biased if prioritized experience replay is used ($\epsilon << 1$).

Note that this bias can be partly or completely corrected using weighted importance sampling, and this correction is important near convergence at the end of training (Schaul *et al.*, 2015b).

9

Benchmarking Deep RL

Comparing deep learning algorithms is a challenging problem due to the stochastic nature of the learning process and the narrow scope of the datasets examined during algorithm comparisons. This problem is exacerbated in deep reinforcement learning. Indeed, deep RL involves both stochasticity in the environment and stochasticity inherent to model learning, which makes ensuring fair comparisons and reproducibility especially difficult. To this end, simulations of many sequential decision-making tasks have been created to serve as benchmarks. In this section, we present several such benchmarks. Next, we give key elements to ensure consistency and reproducibility of experimental results. Finally, we also discuss some open-source implementations for deep RL algorithms.

9.1 Benchmark Environments

9.1.1 Classic control problems

Several classic control problems have long been used to evaluate reinforcement learning algorithms. These include balancing a pole on a cart (Cartpole) (Barto *et al.*, 1983), trying to get a car

up a mountain using momentum (Mountain Car) (Moore, 1990), swinging a pole up using momentum and subsequently balancing it (Acrobot) (Sutton and Barto, 1998). These problems have been commonly used as benchmarks for tabular RL and RL algorithms using linear function approximators (Whiteson *et al.*, 2011). Nonetheless, these simple environments are still sometimes used to benchmark deep RL algorithms (Ho and Ermon, 2016; Duan *et al.*, 2016a; Lillicrap *et al.*, 2015).

9.1.2 Games

Board-games have also been used for evaluating artificial intelligence methods for decades (Shannon, 1950; Turing, 1953; Samuel, 1959; Sutton, 1988; Littman, 1994; Schraudolph *et al.*, 1994; Tesauro, 1995; Campbell *et al.*, 2002). In recent years, several notable works have stood out in using deep RL for mastering Go (Silver *et al.*, 2016a) or Poker (Brown and Sandholm, 2017; Moravčik *et al.*, 2017).

In parallel to the achievements in board games, video games have also been used to further investigate reinforcement learning algorithms. In particular,

- many of these games have large observation space and/or large action space;

- they are often non-Markovian, which require specific care (see §10.1);

- they also usually require very long planning horizons (e.g., due to sparse rewards).

Several platforms based on video games have been popularized. The Arcade Learning Environment (ALE) (Bellemare *et al.*, 2013) was developed to test reinforcement algorithms across a wide range of different tasks. The system encompasses a suite of iconic Atari games, including Pong, Asteroids, Montezuma's Revenge, etc. Figure 9.1 shows sample frames from some of these games. On most of the Atari games, deep RL algorithms have reached super-human level (Mnih *et al.*, 2015). Due to the similarity in state and action spaces between different Atari

games or different variants of the same game, they are also a good test-bed for evaluating generalization of reinforcement learning algorithms (Machado *et al.*, 2017b), multi-task learning (Parisotto *et al.*, 2015) and for transfer learning (Rusu *et al.*, 2015).
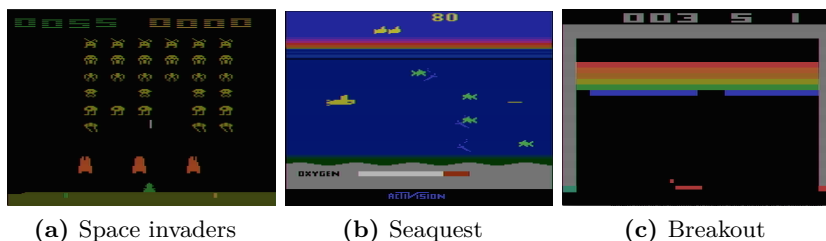

(a) Space invaders   **(b)** Seaquest   **(c)** Breakout

Figure 9.1: Illustration of three Atari games.

The General Video Game AI (GVGAI) competition framework (Perez-Liebana *et al.*, 2016) was created and released with the purpose of providing researchers a platform for testing and comparing their algorithms on a large variety of games and under different constraints. The agents are required to either play multiple unknown games with or without access to game simulations, or to design new game levels or rules.

VizDoom (Kempka *et al.*, 2016) implements the Doom video game as a simulated environment for reinforcement learning. VizDoom has been used as a platform for investigations of reward shaping (Lample and Chaplot, 2017), curriculum learning (Wu and Tian, 2016), predictive planning (Dosovitskiy and Koltun, 2016), and meta-reinforcement learning (Duan *et al.*, 2016b).

The open-world nature of Minecraft also provides a convenient platform for exploring reinforcement learning and artificial intelligence. Project Malmo (Johnson *et al.*, 2016) is a framework that provides easy access to the Minecraft video game. The environment and framework provide layers of abstraction that facilitate tasks ranging from simple navigation to collaborative problem solving. Due to the nature of the simulation, several works have also investigated lifelong-learning, curriculum learning, and hierarchical planning using Minecraft as a

platform (Tessler *et al.*, 2017; Matiisen *et al.*, 2017; Branavan *et al.*, 2012; Oh *et al.*, 2016).

Similarly, Deepmind Lab (Beattie *et al.*, 2016) provides a 3D platform adapted from the Quake video game. The Labyrinth maze environments provided with the framework have been used in work on hierarchical, lifelong and curriculum learning (Jaderberg *et al.*, 2016; Mirowski *et al.*, 2016; Teh *et al.*, 2017).

Finally, "StarCraft II" (Vinyals *et al.*, 2017) and "Starcraft: Broodwar" (Wender and Watson, 2012; Synnaeve *et al.*, 2016) provide similar benefits in exploring lifelong-learning, curriculum learning, and other related hierarchical approaches. In addition, real-time strategy (RTS) games – as with the Starcraft series – are also an ideal testbed for multi-agent systems. Consequently, several works have investigated these aspects in the Starcraft framework (Foerster *et al.*, 2017b; Peng *et al.*, 2017a; Brys *et al.*, 2014).

9.1.3   Continuous control systems and robotics domains

While games provide a convenient platform for reinforcement learning, the majority of those environments investigate discrete action decisions. In many real-world systems, as in robotics, it is necessary to provide frameworks for continuous control.

In that setting, the MuJoCo (Todorov *et al.*, 2012) simulation framework is used to provide several locomotion benchmark tasks. These tasks typically involve learning a gait to move a simulated robotic agent as fast as possible. The action space is the amount of torque to apply to motors on the agents' joints, while the observations provided are typically the joint angles and positions in the 3D space. Several frameworks have built on top of these locomotion tasks to provide hierarchical task environments (Duan *et al.*, 2016a) and multi-task learning platforms (Henderson *et al.*, 2017a).

Because the MuJoCo simulator is closed-source and requires a license, an open-source initiative called Roboschool (Schulman *et al.*, 2017b) provides the same locomotion tasks along with more complex tasks involving humanoid robot simulations (such as learning to run and chase a moving flag while being hit by obstacles impeding progress).
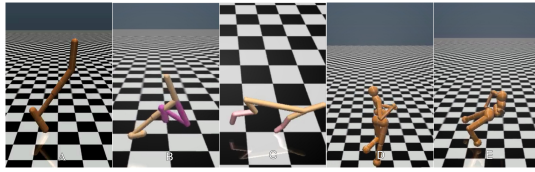
Figure 9.2: Screenshots from MuJoCo locomotion benchmark environments provided by OpenAI Gym.

These tasks allow for evaluation of complex planning in reinforcement learning algorithms.

Physics engines have also been used to investigate transfer learning to real-world applications. For instance, the Bullet physics engine (Coumans, Bai, *et al.*, 2016) has been used to learn locomotion skills in simulation, for character animation in games (Peng *et al.*, 2017b) or for being transferred to real robots (Tan *et al.*, 2018). This also includes manipulation tasks (Rusu *et al.*, 2016; Duan *et al.*, 2017) where a robotic arm stacks cubes in a given order. Several works integrate Robot Operating System (ROS) with physics engines (such as ODE, or Bullet) to provide RL-compatible access to near real-world robotic simulations (Zamora *et al.*, 2016; Ueno *et al.*, 2017). Most of them can also be run on real robotic systems using the same software.

There exists also a toolkit that leverages the Unity platform for creating simulation environments (Juliani *et al.*, 2018). This toolkit enables the development of learning environments that are rich in sensory and physical complexity and supports the multi-agent setting.

9.1.4 Frameworks

Most of the previously cited benchmarks have open-source code available. There also exists easy-to-use wrappers for accessing many different benchmarks. One such example is OpenAI Gym (Brockman *et al.*, 2016). This wrapper provides ready access to environments such as algorithmic, Atari, board games, Box2d games, classical control problems, MuJoCo robotics simulations, toy text problems, and others. Gym Retro[1] is a wrapper similar to OpenAI Gym and it provides over 1,000 games

---

[1]https://github.com/openai/retro

across a variety of backing emulators. The goal is to study the ability of deep RL agents to generalize between games that have similar concepts but different appearances. Other frameworks such as μniverse[2] and SerpentAI[3] also provide wrappers for specific games or simulations.

9.2   Best practices to benchmark deep RL

Ensuring best practices in scientific experiments is crucial to continued scientific progress. Across various fields, investigations in reproducibility have found problems in numerous publications, resulting in several works providing experimental guidelines in proper scientific practices (Sandve *et al.*, 2013; Baker, 2016; Halsey *et al.*, 2015; Casadevall and Fang, 2010). To this end, several works have investigated proper metrics and experimental practices when comparing deep RL algorithms (Henderson *et al.*, 2017b; Islam *et al.*, 2017; Machado *et al.*, 2017b; Whiteson *et al.*, 2011).

Number of Trials, Random Seeds and Significance Testing

Stochasticity plays a large role in deep RL, both from randomness within initializations of neural networks and stochasticity in environments. Results may vary significantly simply by changing the random seed. When comparing the performance of algorithms, it is therefore important to run many trials across different random seeds.

In deep RL, it has become common to simply test an algorithm's effectiveness with an average across a few learning trials. While this is a reasonable benchmark strategy, techniques derived from significance testing (Demšar, 2006; Bouckaert and Frank, 2004; Bouckaert, 2003; Dietterich, 1998) have the advantage of providing statistically grounded arguments in favor of a given hypothesis. In practice for deep RL, significance testing can be used to take into account the standard deviation across several trials with different random seeds and environment conditions. For instance, a simple 2-sample *t*-test can give an idea of whether performance gains are significantly due to the algorithm performance

---

[2]https://github.com/unixpickle/muniverse
[3]https://github.com/SerpentAI/SerpentAI

or to noisy results in highly stochastic settings. In particular, while several works have used the top-K trials and simply presented those as performance gains, this has been argued to be inadequate for fair comparisons (Machado *et al.*, 2017b; Henderson *et al.*, 2017b).

In addition, one should be careful not to over-interpret the results. It is possible that a hypothesis can be shown to hold for one or several given environments and under one or several given set of hyperparameters, but fail in other settings.

Hyperparameter Tuning and Ablation Comparisons

Another important consideration is ensuring a fair comparison between learning algorithms. In this case, an ablation analysis compares alternate configurations across several trials with different random seeds. It is especially important to tune hyperparameters to the greatest extent possible for baseline algorithms. Poorly chosen hyperparameters can lead to an unfair comparison between a novel and a baseline algorithm. In particular, network architecture, learning rate, reward scale, training discount factor, and many other parameters can affect results significantly. Ensuring that a novel algorithm is indeed performing much better requires proper scientific procedure when choosing such hyperparameters (Henderson *et al.*, 2017b).

Reporting Results, Benchmark Environments, and Metrics

Average returns (or cumulative reward) across evaluation trajectories are often reported as a comparison metric. While some literature (Gu *et al.*, 2016a; Gu *et al.*, 2017c) has also used metrics such as average maximum return or maximum return within Z samples, these may be biased to make results for highly unstable algorithms appear more significant. For example, if an algorithm reaches a high maximum return quickly, but then diverges, such metrics would ensure this algorithm appears successful. When choosing metrics to report, it is important to select those that provide a fair comparison. If the algorithm performs better in average maximum return, but worse by using an average return metric, it is important to highlight both results and describe the benefits and shortcomings of such an algorithm (Henderson *et al.*, 2017b).

This is also applicable to the selection of which benchmark environments to report during evaluation. Ideally, empirical results should cover a large mixture of environments to determine in which settings an algorithm performs well and in which settings it does not. This is vital for determining real-world performance applications and capabilities.

9.3 Open-source software for Deep RL

A deep RL agent is composed of a learning algorithm (model-based or model-free) along with specific structure(s) of function approximator(s). In the online setting (more details are given in Chapter 8), the agent follows a specific exploration/exploitation strategy and typically uses a memory of its previous experience for sample efficiency.

While many papers release implementations of various deep RL algorithms, there also exist some frameworks built to facilitate the development of new deep RL algorithms or to apply existing algorithms to a variety of environments. We provide a list of some of the existing frameworks in Appendix A.

10

Deep reinforcement learning beyond MDPs

We have so far mainly discussed how an agent is able to learn how to behave in a given Markovian environment where all the interesting information (the state $s_t \in \mathcal{S}$) is obtained at every time step t. In this chapter, we discuss more general settings with (i) non-Markovian environments, (ii) transfer learning and (iii) multi-agent systems.

10.1 Partial observability and the distribution of (related) MDPs

In domains where the Markov hypothesis holds, it is straightforward to show that the policy need not depend on what happened at previous time steps to recommend an action (by definition of the Markov hypothesis). This section describes two different cases that complicate the Markov setting: the partially observable environments and the distribution of (related) environments.

Those two settings are at first sight quite different conceptually. However, in both settings, at each step in the sequential decision process, the agent may benefit from taking into account its whole observable history up to the current time step t when deciding what action to perform. In other words, a history of observations can be used as a pseudo-state (pseudo-state because that refers to a different and abstract

stochastic control process). Any missing information in the history of observations (potentially long before time t) can introduce a bias in the RL algorithm (as described in Chapter 7 when some features are discarded).

10.1.1 The partially observable scenario

In this setting, the agent only receives, at each time step, an observation of its environment that does not allow it to identify the state with certainty. A Partially Observable Markov Decision Process (POMDP) (Sondik, 1978; Kaelbling *et al.*, 1998) is a discrete time stochastic control process defined as follows:

Definition 10.1. A POMDP is a 7-tuple $(\mathcal{S}, \mathcal{A}, T, R, \Omega, O, \gamma)$ where:

- $\mathcal{S}$ is a finite set of states $\{1, \ldots, N_{\mathcal{S}}\}$,

- $\mathcal{A}$ is a finite set of actions $\{1, \ldots, N_{\mathcal{A}}\}$,

- $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition function (set of conditional transition probabilities between states),

- $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathcal{R}$ is the reward function, where $\mathcal{R}$ is a continuous set of possible rewards in a range $R_{\max} \in \mathbb{R}^+$ (e.g., $[0, R_{\max}]$ without loss of generality),

- Ω is a finite set of observations $\{1, \ldots, N_{\Omega}\}$,

- $O : \mathcal{S} \times \Omega \to [0, 1]$ is a set of conditional observation probabilities, and

- $\gamma \in [0, 1)$ is the discount factor.

The environment starts in a distribution of initial states $b(s_0)$. At each time step $t \in \mathbb{N}_0$, the environment is in a state $s_t \in \mathcal{S}$. At the same time, the agent receives an observation $\omega_t \in \Omega$ that depends on the state of the environment with probability $O(s_t, \omega_t)$, after which the agent chooses an action $a_t \in \mathcal{A}$. Then, the environment transitions to state $s_{t+1} \in \mathcal{S}$ with probability $T(s_t, a_t, s_{t+1})$ and the agent receives a reward $r_t \in \mathcal{R}$ equal to $R(s_t, a_t, s_{t+1})$.

When the full model (T, R and O) are known, methods such as Point-Based Value Iteration (PBVI) algorithm (Pineau *et al.*, 2003) for POMDP planning can be used to solve the problem. If the full POMDP model is not available, other reinforcement learning techniques have to be used.

A naive approach to building a space of candidate policies is to consider the set of mappings taking only the very last observation(s) as input. However, in a POMDP setting, this leads to candidate policies that are typically not rich enough to capture the system dynamics, thus suboptimal. In that case, the best achievable policy is stochastic (Singh *et al.*, 1994), and it can be obtained using policy gradient. The alternative is to use a history of previously observed features to better estimate the hidden state dynamics. We denote by $\mathcal{H}_t = \Omega \times (\mathcal{A} \times \mathcal{R} \times \Omega)^t$ the set of histories observed up to time t for $t \in \mathbb{N}_0$ (see Fig. 10.1), and by $\mathcal{H} = \bigcup\limits_{t=0}^{\infty} \mathcal{H}_t$ the space of all possible observable histories.
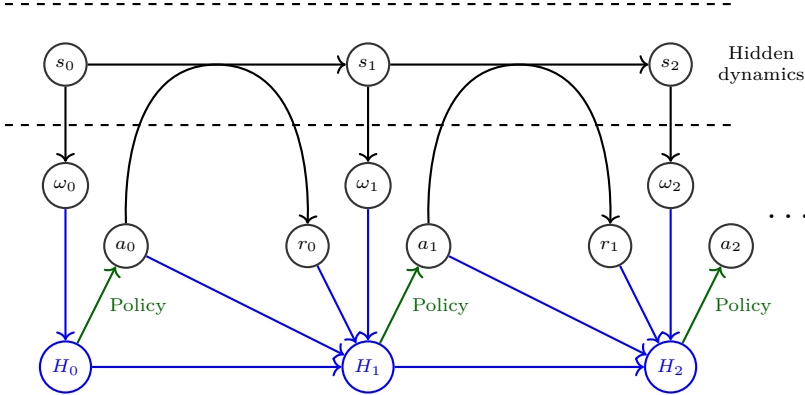
Figure 10.1: Illustration of a POMDP. The actual dynamics of the POMDP is depicted in dark while the information that the agent can use to select the action at each step is the whole history H_t depicted in blue.

A straightforward approach is to take the whole history $H_t \in \mathcal{H}$ as input (Braziunas, 2003). However, increasing the size of the set of candidate optimal policies generally implies: (i) more computation to search within this set (Singh *et al.*, 1994; McCallum, 1996) and, (ii) an increased risk of including candidate policies suffering from overfitting

due to lack of sufficient data, which thus leads to a bias-overfitting tradeoff when learning policies from data (François-Lavet *et al.*, 2017).

In the case of deep RL, the architectures used usually have a smaller number of parameters and layers than in supervised learning due to the more complex RL setting, but the trend of using ever smarter and complex architectures in deep RL happens similarly to supervised learning tasks. Architectures such as convolutional layers or recurrency are particularly well-suited to deal with a large input space because they offer interesting generalization properties. A few empirical successes on large scale POMDPs make use of convolutional layers (Mnih *et al.*, 2015) and/or recurrent layers (Hausknecht and Stone, 2015), such as LSTMs (Hochreiter and Schmidhuber, 1997).

10.1.2 The distribution of (related) environments

In this setting, the environment of the agent is a distribution of different (yet related) tasks that differ for instance in the reward function or in the probabilities of transitions from one state to another. Each task $T_i \sim \mathcal{T}$ can be defined by the observations $\omega_t \in \Omega$ (which are equal to s_t if the environments are Markov), the rewards $r_t \in \mathcal{R}$, as well as the effect of the actions $a_t \in \mathcal{A}$ taken at each step. Similarly to the partially observable context, we denote the history of observations by H_t, where $H_t \in \mathcal{H}_t = \Omega \times (\mathcal{A} \times \mathcal{R} \times \Omega)^t$. The agent aims at finding a policy $\pi(a_t | H_t; \theta)$ with the objective of maximizing its expected return, defined (in the discounted setting) as

$$\mathop{\mathbb{E}}_{T_i \sim \mathcal{T}} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid H_t, \pi \right].$$

An illustration of the general setting of meta learning on non-Markov environments is given in Figure 10.2.

Different approaches have been investigated in the literature. The Bayesian approach aims at explicitly modeling the distribution of the different environments, if a prior is available (Ghavamzadeh *et al.*, 2015). However, it is often intractable to compute the Bayesian-optimal strategy and one has to rely on more practical approaches that do not require an explicit model of the distribution. The concept of *meta-learning* or *learning to learn* aims at discovering, from experience, how
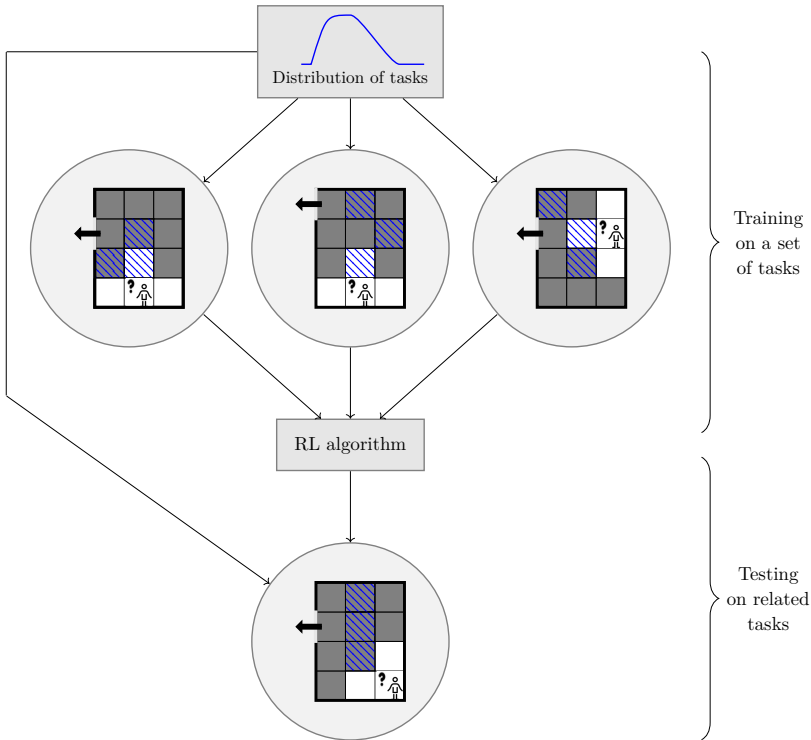
Figure 10.2: Illustration of the general setting of meta learning on POMDPs for a set of labyrinth tasks. In this illustration, it is supposed that the agent only sees the nature of the environment just one time step away from him.

to behave in a range of tasks and how to negotiate the exploration-exploitation tradeoff (Hochreiter *et al.*, 2001). In that case, deep RL techniques have been investigated by, e.g., Wang *et al.*, 2016a; Duan *et al.*, 2016b with the idea of using recurrent networks trained on a set of environments drawn i.i.d. from the distribution.

Some other approaches have also been investigated. One possibility is to train a neural network to imitate the behavior of known optimal policies on MDPs drawn from the distribution (Castronovo *et al.*, 2017). The parameters of the model can also be explicitly trained such that a small number of gradient steps in a new task from the distribution will produce fast learning on that task (Finn *et al.*, 2017).

There exists different denominations for this setting with a distribution of environments. For instance, the denomination of "multi-task setting" is usually used in settings where a short history of observations is sufficient to clearly distinguish the tasks. As an example of a multi-task setting, a deep RL agent can exceed median human performance on the set of 57 Atari games with a single set of weights (Hessel *et al.*, 2018). Other related denominations are the concepts of "contextual" policies (Da Silva *et al.*, 2012) and "universal"/"goal conditioned" value functions (Schaul *et al.*, 2015a) that refer to learning policies or value function within the same dynamics for multiple goals (multiple reward functions).

10.2 Transfer learning

Transfer learning is the task of efficiently using previous knowledge from a source environment to achieve a good performance in a target environment. In a transfer learning setting, the target environment should not be in the distribution of the source tasks. However, in practice, the concept of transfer learning is sometimes closely related to meta learning, as we discuss hereafter.

10.2.1 Zero-shot learning

The idea of zero-shot learning is that an agent should be able to act appropriately in a new task directly from experience acquired on other similar tasks. For instance, one use case is to learn a policy in a simulation environment and then use it in a real-world context where gathering experience is not possible or severely constrained (see §11.2). To achieve this, the agent must either (i) develop generalization capacities described in Chapter 7 or (ii) use specific transfer strategies that explicitly retrain or replace some of its components to adjust to new tasks.

To develop generalization capacities, one approach is to use an idea similar to data augmentation in supervised learning so as to make sense of variations that were not encountered in the training data. Exactly as in the meta-learning setting (§10.1.2), the actual (unseen) task may

appear to the agent as just another variation if there is enough data augmentation on the training data. For instance, the agent can be trained with deep RL techniques on different tasks simultaneously, and it is shown by Parisotto *et al.*, 2015 that it can generalize to new related domains where the exact state representation has never been observed. Similarly, the agent can be trained in a simulated environment while being provided with different renderings of the observations. In that case, the learned policy can transfer well to real images (Sadeghi and Levine, 2016; Tobin *et al.*, 2017). The underlying reason for these successes is the ability of the deep learning architecture to generalize between states that have similar high-level representations and should therefore have the same value function/policy in different domains. Rather than manually tuning the randomization of simulations, one can also adapt the simulation parameters by matching the policy behavior in simulation to the real world (Chebotar *et al.*, 2018). Another approach to zero-shot transfer is to use algorithms that enforce states that relate to the same underlying task but have different renderings to be mapped into an abstract state that is close (Tzeng *et al.*, 2015; François-Lavet *et al.*, 2018).

10.2.2   Lifelong learning or continual learning

A specific way of achieving transfer learning is to aim at *lifelong learning* or *continual learning.* According to Silver *et al.*, 2013, lifelong machine learning relates to the capability of a system to learn many tasks over a lifetime from one or more domains.

In general, deep learning architectures can generalize knowledge across multiple tasks by sharing network parameters. A direct approach is thus to train function approximators (e.g. policy, value function, model, etc.) sequentially in different environments. The difficulty of this approach is to find methods that enable the agent to retain knowledge in order to more efficiently learn new tasks. The problem of retaining knowledge in deep reinforcement learning is complicated by the phenomenon of catastrophic forgetting, where generalization to previously seen data is lost at later stages of learning.

The straightforward approach is to either (i) use experience replay from all previous experience (as discussed in §8.2), or (ii) retrain occasionally on previous tasks similar to the meta-learning setting (as discussed in §10.1.2).

When these two options are not available, or as a complement to the two previous approaches, one can use deep learning techniques that are robust to forgetting, such as progressive networks (Rusu *et al.*, 2016). The idea is to leverage prior knowledge by adding, for each new task, lateral connections to previously learned features (that are kept fixed). Other approaches to limiting catastrophic forgetting include slowing down learning on the weights important for previous tasks (Kirkpatrick *et al.*, 2016) and decomposing learning into skill hierarchies (Stone and Veloso, 2000; Tessler *et al.*, 2017).
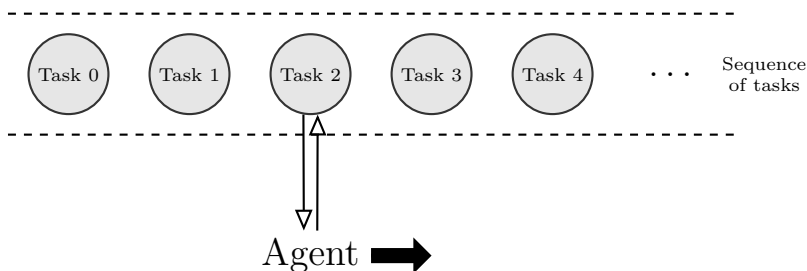
Figure 10.3: Illustration of the continual learning setting where an agent has to interact sequentially with related (but different) tasks.

10.2.3 Curriculum learning

A particular setting of continual learning is curriculum learning. Here, the goal is to explicitly design a sequence of source tasks for an agent to train on such that the final performance or learning speed is improved on a target task. The idea is to start by learning small and easy aspects of the target task and then to gradually increase the difficulty level (Bengio *et al.*, 2009; Narvekar *et al.*, 2016). For instance, Florensa *et al.* (2018) use generative adversarial training to automatically generate goals for a contextual policy such that they are always at the appropriate level of difficulty. As the difficulty and number of tasks increase, one

possibility to satisfy the bias-overfitting tradeoff is to consider network transformations through learning.

10.3 Learning without explicit reward function

In reinforcement learning, the reward function defines the goals to be achieved by the agent (for a given environment and a given discount factor). Due to the complexity of environments in practical applications, defining a reward function can turn out to be rather complicated. There are two other possibilities: (i) given demonstrations of the desired task, we can use imitation learning or extract a reward function using inverse reinforcement learning; (ii) a human may provide feedback on the agent's behavior in order to define the task.

10.3.1 Learning from demonstrations

In some circumstances, the agent is only provided with trajectories of an expert agent (also called the teacher), without rewards. Given an observed behavior, the goal is to have the agent perform similarly. Two approaches are possible:

- Imitation learning uses supervised learning to map states to actions from the observations of the expert's behavior (e.g., Giusti *et al.*, 2016). Among other applications, this approach has been used for self-driving cars to map raw pixels directly to steering commands thanks to a deep neural network (Bojarski *et al.*, 2016).

- Inverse reinforcement learning (IRL) determines a possible reward function given observations of optimal behavior. When the system dynamics is known (except the reward function), this is an appealing approach particularly when the reward function provides the most generalizable definition of the task (Ng, Russell, *et al.*, 2000; Abbeel and Ng, 2004). For example, let us consider a large MDP for which the expert always ends up transitioning to the same state. In that context, one may be able to easily infer, from only a few trajectories, what the probable goal of the task is (a reward function that explains the behavior of the teacher),

as opposed to directly learning the policy via imitation learning, which is much less efficient. Note that recent works bypass the requirement of the knowledge of the system dynamics (Boularias *et al.*, 2011; Kalakrishnan *et al.*, 2013; Finn *et al.*, 2016b) by using techniques based on the principle of maximum causal entropy (Ziebart, 2010).

A combination of the two approaches has also been investigated by Neu and Szepesvári, 2012; Ho and Ermon, 2016. In particular, Ho and Ermon, 2016 use adversarial methods to learn a discriminator (i.e., a reward function) such that the policy matches the distribution of demonstrative samples.

It is important to note that in many real-world applications, the teacher is not exactly in the same context as the agent. Thus, transfer learning may also be of crucial importance (Schulman *et al.*, 2016; Liu *et al.*, 2017).

Another setting requires the agent to learn directly from a sequence of observations without corresponding actions (and possibly in a slightly different context). This may be done in a meta-learning setting by providing positive reward to the agent when it performs as it is expected based on the demonstration of the teacher. The agent can then act based on new unseen trajectories of the teacher, with the objective that is can generalize sufficiently well to perform new tasks (Paine *et al.*, 2018).

10.3.2 Learning from direct feedback

Learning from feedback investigates how an agent can interactively learn behaviors from a human teacher who provides positive and negative feedback signals. In order to learn complex behavior, human trainer feedbacks has the potential to be more performant than a reward function defined a priori (MacGlashan *et al.*, 2017; Warnell *et al.*, 2017). This setting can be related to the idea of curriculum learning discussed in §10.2.3.

In the work of Hadfield-Menell *et al.*, 2016, the cooperative inverse reinforcement learning framework considers a two-player game between a human and a robot interacting with an environment with the purpose

of maximizing the human's reward function. In the work of Christiano *et al.*, 2017, it is shown how learning a separate reward model using supervised learning lets us significantly reduce the amount of feedback required from a human teacher. They also present the first practical applications of using human feedback in the context of deep RL to solve tasks with a high dimensional observation space.

10.4 Multi-agent systems

A multi-agent system is composed of multiple interacting agents within an environment (Littman, 1994).

Definition 10.2. A multi-agent POMDP with N agents is a tuple $(\mathcal{S}, \mathcal{A}_N, \ldots, \mathcal{A}_N, T, R_1, \ldots, R_N, \Omega, O_1, \ldots, O_N, \gamma)$ where:

- $\mathcal{S}$ is a finite set of states $\{1, \ldots, N_{\mathcal{S}}\}$ (describing the possible configurations of all agents),

- $\mathcal{A} = \mathcal{A}_1 \times \ldots \times \mathcal{A}_n$ is a finite set of actions $\{1, \ldots, N_{\mathcal{A}}\}$,

- $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition function (set of conditional transition probabilities between states),

- $\forall i, R_i : \mathcal{S} \times \mathcal{A}_i \times \mathcal{S} \to \mathcal{R}$ is the reward function for agent i, where $\mathcal{R}$ is a continuous set of possible rewards in a range $R_{\max} \in \mathbb{R}^+$ (e.g., $[0, R_{\max}]$ without loss of generality),

- Ω is a finite set of observations $\{1, \ldots, N_{\Omega}\}$,

- $\forall i, O_i : \mathcal{S} \times \Omega \to [0, 1]$ is a set of conditional observation probabilities, and

- $\gamma \in [0, 1)$ is the discount factor.

For this type of system, many different settings can be considered.

- Collaborative versus non-collaborative setting. In a pure collaborative setting, agents have a shared reward measurement ($R_i = R_j, \forall\, i, j \in [1, \ldots, N]$). In a mixed or non-collaborative (possibly adversarial) setting each agent obtains different rewards. In both

cases, each agent i aims to maximize a discounted sum of its rewards $\sum_{t=0}^{H} \gamma^t r_t^{(i)}$.

- Decentralized versus centralized setting. In a decentralized setting, each agent selects its own action conditioned only on its local information. When collaboration is beneficial, this decentralized setting can lead to the emergence of communication between agents in order to share information (e.g., Sukhbaatar *et al.*, 2016).

  In a centralized setting, the RL algorithm has access to all observations $w^{(i)}$ and all rewards $r^{(i)}$. The problem can be reduced to a single-agent RL problem on the condition that a single objective can be defined (in a purely collaborative setting, the unique objective is straightforward). Note that even when a centralized approach can be considered (depending on the problem), an architecture that does not make use of the multi-agent structure usually leads to sub-optimal learning (e.g., Sunehag *et al.*, 2017).

In general, multi-agent systems are challenging because agents are independently updating their policies as learning progresses, and therefore the environment appears non-stationary to any particular agent. For training one particular agent, one approach is to select randomly the policies of all other agents from a pool of previously learned policies. This can stabilize training of the agent that is learning and prevent overfitting to the current policy of the other agents (Silver *et al.*, 2016a).

In addition, from the perspective of a given agent, the environment is usually strongly stochastic even with a known, fixed policy for all other agents. Indeed, any given agent does not know how the other agents will act and consequently, it doesn't know how its own actions contribute to the rewards it obtains. This can be partly explained due to partial observability, and partly due to the intrinsic stochasticity of the policies followed by other agents (e.g., when there is a high level of exploration). For these reasons, a high variance of the expected global return is observed, which makes learning challenging (particularly when used in conjunction with bootstrapping). In the context of the collaborative

setting, a common approach is to use an actor-critic architecture with a centralized critic during learning and decentralized actor (the agents can be deployed independently). These topics have already been investigated in works by Foerster *et al.*, 2017a; Sunehag *et al.*, 2017; Lowe *et al.*, 2017 as well as in the related work discussed in these papers. Other works have shown how it is possible to take into account a term that either considers the learning of the other agent (Foerster *et al.*, 2018) or an internal model that predicts the actions of other agents (Jaques *et al.*, 2018).

Deep RL agents are able to achieve human-level performance in 3D multi-player first-person video games such as Quake III Arena Capture the Flag (Jaderberg *et al.*, 2018). Thus, techniques from deep RL have a large potential for many real-world tasks that require multiple agents to cooperate in domains such as robotics, self-driving cars, etc.

11

Perspectives on deep reinforcement learning

In this section, we first mention some of the main successes of deep RL. Then, we describe some of the main challenges for tackling an even wider range of real-world problems. Finally, we discuss some parallels that can be found between deep RL and neuroscience.

11.1   Successes of deep reinforcement learning

Deep RL techniques have demonstrated their ability to tackle a wide range of problems that were previously unsolved. Some of the most renowned achievements are

- beating previous computer programs in the game of backgammon (Tesauro, 1995),

- attaining superhuman-level performance in playing Atari games from the pixels (Mnih *et al.*, 2015),

- mastering the game of Go (Silver *et al.*, 2016a), as well as

- beating professional poker players in the game of heads up no-limit Texas hold'em: Libratus (Brown and Sandholm, 2017) and Deepstack (Moravčik *et al.*, 2017).

These achievements in popular games are important because they show the potential of deep RL in a variety of complex and diverse tasks that require working with high-dimensional inputs. Deep RL has also shown lots of potential for real-world applications such as robotics (Kalashnikov *et al.*, 2018), self-driving cars (You *et al.*, 2017), finance (Deng *et al.*, 2017), smart grids (François-Lavet *et al.*, 2016b), dialogue systems (Fazel-Zarandi *et al.*, 2017), etc. In fact, Deep RL systems are already in production environments. For example, Gauci *et al.* (2018) describe how Facebook uses Deep RL such as for pushing notifications and for faster video loading with smart prefetching.

RL is also applicable to fields where one could think that supervised learning alone is sufficient, such as sequence prediction (Ranzato *et al.*, 2015; Bahdanau *et al.*, 2016). Designing the right neural architecture for supervised learning tasks has also been cast as an RL problem (Zoph and Le, 2016). Note that those types of tasks can also be tackled with evolutionary strategies (Miikkulainen *et al.*, 2017; Real *et al.*, 2017).

Finally, it should be mentioned that deep RL has applications in classic and fundamental algorithmic problems in the field of computer science, such as the travelling salesman problem (Bello *et al.*, 2016). This is an NP-complete problem and the possibility to tackle it with deep RL shows the potential impact that it could have on several other NP-complete problems, on the condition that the structure of these problems can be exploited.

11.2 Challenges of applying reinforcement learning to real-world problems

The algorithms discussed in this introduction to deep RL can, in principle, be used to solve many different types of real-world problems. In practice, even in the case where the task is well defined (explicit reward function), there is one fundamental difficulty: it is often not possible to let an agent interact freely and sufficiently in the actual environment (or set of environments), due to either safety, cost or time constraints. We can distinguish two main cases in real-world applications:

1. The agent may not be able to interact with the true environment but only with an inaccurate simulation of it. This scenario occurs

for instance in robotics (Zhu *et al.*, 2016; Gu *et al.*, 2017a). When first learning in a simulation, the difference with the real-world domain is known as the *reality gap* (see e.g. Jakobi *et al.*, 1995).

2. The acquisition of new observations may not be possible anymore (e.g., the batch setting). This scenario happens for instance in medical trials, in tasks with dependence on weather conditions or in trading markets (e.g., energy markets and stock markets).

Note that a combination of the two scenarios is also possible in the case where the dynamics of the environment may be simulated but where there is a dependence on an exogenous time series that is only accessible via limited data (François-Lavet *et al.*, 2016b).

In order to deal with these limitations, different elements are important:

- One can aim to develop a simulator that is as accurate as possible.

- One can design the learning algorithm so as to improve generalization and/or use transfer learning methods (see Chapter 7).

11.3 Relations between deep RL and neuroscience

One of the interesting aspects of deep RL is its relations to neuroscience. During the development of algorithms able to solve challenging sequential decision-making tasks, biological plausibility was not a requirement from an engineering standpoint. However, biological intelligence has been a key inspiration for many of the most successful algorithms. Indeed, even the ideas of reinforcement and deep learning have strong links with neuroscience and biological intelligence.

Reinforcement   In general, RL has had a rich conceptual relationship to neuroscience. RL has used neuroscience as an inspiration and it has also been a tool to explain neuroscience phenomena (Niv, 2009). RL models have also been used as a tool in the related field of neuroeconomics (Camerer *et al.*, 2005), which uses models of human decision-making to inform economic analyses.

The idea of reinforcement (or at least the term) can be traced back to the work of Pavlov (1927) in the context of animal behavior. In the Pavlovian conditioning model, reinforcement is described as the strengthening/weakening effect of a behavior whenever that behavior is preceded by a specific stimulus. The Pavlovian conditioning model led to the development of the Rescorla-Wagner Theory (Rescorla, Wagner, *et al.*, 1972), which assumed that learning is driven by the error between predicted and received reward, among other prediction models. In computational RL, those concepts have been at the heart of many different algorithms, such as in the development of temporal-difference (TD) methods (Sutton, 1984; Schultz *et al.*, 1997; Russek *et al.*, 2017). These connections were further strengthened when it was found that the dopamine neurons in the brain act in a similar manner to TD-like updates to direct learning in the brain (Schultz *et al.*, 1997).

Driven by such connections, many aspects of reinforcement learning have also been investigated directly to explain certain phenomena in the brain. For instance, computational models have been an inspiration to explain cognitive phenomena such as exploration (Cohen *et al.*, 2007) and temporal discounting of rewards (Story *et al.*, 2014). In cognitive science, Kahneman (2011) has also described that there is a dichotomy between two modes of thoughts: a "System 1" that is fast and instinctive and a "System 2" that is slower and more logical. In deep reinforcement, a similar dichotomy can be observed when we consider the model-free and the model-based approaches. As another example, the idea of having a meaningful abstract representation in deep RL can also be related to how animals (including humans) think. Indeed, a conscious thought at a particular time instant can be seen as a low-dimensional combination of a few concepts in order to take decisions (Bengio, 2017).

There is a dense and rich literature about the connections between RL and neuroscience and, as such, the reader is referred to the work of Sutton and Barto (2017), Niv (2009), Lee *et al.* (2012), Holroyd and Coles (2002), Dayan and Niv (2008), Dayan and Daw (2008), Montague (2013), and Niv and Montague (2009) for an in-depth history of the development of reinforcement learning and its relations to neuroscience.

Deep learning   Deep learning also finds its origin in models of neural processing in the brain of biological entities. However, subsequent developments are such that deep learning has become partly incompatible with current knowledge of neurobiology (Bengio *et al.*, 2015). There exists nonetheless many parallels. One such example is the convolutional structure used in deep learning that is inspired by the organization of the animal visual cortex (Fukushima and Miyake, 1982; LeCun *et al.*, 1998).

Much work is still needed to bridge the gap between machine learning and general intelligence of humans (or even animals). Looking back at all the achievements obtained by taking inspiration from neuroscience, it is natural to believe that further understanding of biological brains could play a vital role in building more powerful algorithms and conversely. In particular, we refer the reader to the survey by Hassabis *et al.*, 2017 where the bidirectional influence between deep RL and neuroscience is discussed.

12

Conclusion

Sequential decision-making remains an active field of research with many theoretical, methodological and experimental challenges still open. The important developments in the field of deep learning have contributed to many new avenues where RL methods and deep learning are combined. In particular, deep learning has brought important generalization capabilities, which opens new possibilities to work with large, high-dimensional state and/or action spaces. There is every reason to think that this development will continue in the coming years with more efficient algorithms and many new applications.

12.1 Future development of deep RL

In deep RL, we have emphasized in this manuscript that one of the central questions is the concept of generalization. To this end, the new developments in the field of deep RL will surely develop the current trends of taking explicit algorithms and making them differentiable so that they can be embedded in a specific form of neural network and can be trained end-to-end. This can bring algorithms with richer and smarter structures that would be better suited for reasoning on a more abstract level, which would allow to tackle an even wider range of

applications than they currently do today. Smart architectures could also be used for hierarchical learning where much progress is still needed in the domain of temporal abstraction.

We also expect to see deep RL algorithms going in the direction of meta-learning and lifelong learning where previous knowledge (e.g., in the form of pre-trained networks) can be embedded so as to increase performance and training time. Another key challenge is to improve current transfer learning abilities between simulations and real-world cases. This would allow learning complex decision-making problems in simulations (with the possibility to gather samples in a flexible way), and then use the learned skills in real-world environments, with applications in robotics, self-driving cars, etc.

Finally, we expect deep RL techniques to develop improved curiosity driven abilities to be able to better discover by themselves their environment.

12.2   Applications and societal impact of deep RL and artificial intelligence in general

In terms of applications, many areas are likely to be impacted by the possibilities brought by deep RL. It is always difficult to predict the timelines for the different developments, but the current interest in deep RL could be the beginning of profound transformations in information and communication technologies, with applications in clinical decision support, marketing, finance, resource management, autonomous driving, robotics, smart grids, and more.

Current developments in artificial intelligence (both for deep RL or in general for machine learning) follows the development of many tools brought by information and communications technologies. As with all new technologies, this comes with different potential opportunities and challenges for our society.

On the positive side, algorithms based on (deep) reinforcement learning promise great value to people and society. They have the potential to enhance the quality of life by automating tedious and exhausting tasks with robots (Levine *et al.*, 2016; Gandhi *et al.*, 2017; Pinto *et al.*, 2017). They may improve education by providing adaptive

content and keeping students engaged (Mandel *et al.*, 2014). They can improve public health with, for instance, intelligent clinical decision-making (Fonteneau *et al.*, 2008; Bennett and Hauser, 2013). They may provide robust solutions to some of the self-driving cars challenges (Bojarski *et al.*, 2016; You *et al.*, 2017). They also have the possibility to help managing ecological resources (Dieterich, 2009) or reducing greenhouse gas emissions by, e.g., optimizing traffic (Li *et al.*, 2016). They have applications in computer graphics, such as for character animation (Peng *et al.*, 2017b). They also have applications in finance (Deng *et al.*, 2017), smart grids (François-Lavet, 2017), etc.

However, we need to be careful that deep RL algorithms are safe, reliable and predictable (Amodei *et al.*, 2016; Bostrom, 2017). As a simple example, to capture what we want an agent to do in deep RL, we frequently end up, in practice, designing the reward function, somewhat arbitrarily. Often this works well, but sometimes it produces unexpected, and potentially catastrophic behaviors. For instance, to remove a certain invasive species from an environment, one may design an agent that obtains a reward every time it removes one of these organisms. However, it is likely that to obtain the maximum cumulative rewards, the agent will learn to let that invasive species develop and only then would eliminate many of the invasive organisms, which is of course not the intended behavior. All aspects related to safe exploration are also potential concerns in the hypothesis that deep RL algorithms are deployed in real-life settings.

In addition, as with all powerful tools, deep RL algorithms also bring societal and ethical challenges (Brundage *et al.*, 2018), raising the question of how they can be used for the benefit of all. Even tough different interpretations can come into play when one discusses human sciences, we mention in this conclusion some of the potential issues that may need further investigation.

The ethical use of artificial intelligence is a broad concern. The specificity of RL as compared to supervised learning techniques is that it can naturally deal with sequences of interactions, which is ideal for chatbots, smart assistants, etc. As it is the case with most technologies, regulation should, at some point, ensure a positive impact of its usage.

In addition, machine learning and deep RL algorithms will likely yield to further automation and robotisation than it is currently possible. This is clearly a concern in the context of autonomous weapons, for instance (Walsh, 2017). Automation also influences the economy, the job market and our society as a whole. A key challenge for humanity is to make sure that the future technological developments in artificial intelligence do not create an ecological crisis (Harari, 2014) or deepen the inequalities in our society with potential social and economic instabilities (Piketty, 2013).

We are still at the very first steps of deep RL and artificial intelligence in general. The future is hard to predict; however, it is key that the potential issues related to the use of these algorithms are progressively taken into consideration in public policies. If that is the case, these new algorithms can have a positive impact on our society.

Appendices

A Deep RL frameworks

Here is a list of some well-known frameworks used for deep RL:

- DeeR (François-Lavet *et al.*, 2016a) is focused on being (i) easily accessible and (ii) modular for researchers.

- Dopamine (Bellemare *et al.*, 2018) provides standard algorithms along with baselines for the ATARI games.

- ELF (Tian *et al.*, 2017) is a research platform for deep RL, aimed mainly to real-time strategy games.

- OpenAI baselines (Dhariwal *et al.*, 2017) is a set of popular deep RL algorithms, including DDPG, TRPO, PPO, ACKTR. The focus of this framework is to provide implementations of baselines.

- PyBrain (Schaul *et al.*, 2010) is a machine learning library with some RL support.

- rllab (Duan *et al.*, 2016a) provides a set of benchmarked implementations of deep RL algorithms.

- TensorForce (Schaarschmidt *et al.*, 2017) is a framework for deep RL built around Tensorflow with several algorithm implementations. It aims at moving reinforcement computations into the Tensorflow graph for performance gains and efficiency. As such, it is heavily tied to the Tensorflow deep learning library. It provides many algorithm implementations including TRPO, DQN, PPO, and A3C.

Even though, they are not tailored specifically for deep RL, we can also cite the two following frameworks for reinforcement learning:

- RL-Glue (Tanner and White, 2009) provides a standard interface that allows to connect RL agents, environments, and experiment programs together.

- RLPy (Geramifard *et al.*, 2015) is a framework focused on value-based RL using linear function approximators with discrete actions.

Table 1 provides a summary of some properties of the aforementioned libraries.

Framework	Deep RL	Python interface	Automatic GPU support
DeeR	yes	yes	yes
Dopamine	yes	yes	yes
ELF	yes	no	yes
OpenAI baselines	yes	yes	yes
PyBrain	yes	yes	no
RL-Glue	no	yes	no
RLPy	no	yes	no
rllab	yes	yes	yes
TensorForce	yes	yes	yes

Table 1: Summary of some characteristics for a few existing RL frameworks.

References

Abadi, M., A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, *et al.* 2016. "Tensor-Flow: Large-Scale Machine Learning on Heterogeneous Distributed Systems". *arXiv preprint arXiv:1603.04467*.

Abbeel, P. and A. Y. Ng. 2004. "Apprenticeship learning via inverse reinforcement learning". In: *Proceedings of the twenty-first international conference on Machine learning*. ACM. 1.

Amari, S. 1998. "Natural Gradient Works Efficiently in Learning". *Neural Computation*. 10(2): 251–276.

Amodei, D., C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané. 2016. "Concrete problems in AI safety". *arXiv preprint arXiv:1606.06565*.

Anderson, T. W., T. W. Anderson, T. W. Anderson, T. W. Anderson, and E.-U. Mathématicien. 1958. *An introduction to multivariate statistical analysis*. Vol. 2. Wiley New York.

Aytar, Y., T. Pfaff, D. Budden, T. L. Paine, Z. Wang, and N. de Freitas. 2018. "Playing hard exploration games by watching YouTube". *arXiv preprint arXiv:1805.11592*.

Bacon, P.-L., J. Harb, and D. Precup. 2016. "The option-critic architecture". *arXiv preprint arXiv:1609.05140*.

Bahdanau, D., P. Brakel, K. Xu, A. Goyal, R. Lowe, J. Pineau, A. Courville, and Y. Bengio. 2016. "An actor-critic algorithm for sequence prediction". *arXiv preprint arXiv:1607.07086.*

Baird, L. 1995. "Residual algorithms: Reinforcement learning with function approximation". In: *ICML.* 30–37.

Baker, M. 2016. "1,500 scientists lift the lid on reproducibility". *Nature News.* 533(7604): 452.

Bartlett, P. L. and S. Mendelson. 2002. "Rademacher and Gaussian complexities: Risk bounds and structural results". *Journal of Machine Learning Research.* 3(Nov): 463–482.

Barto, A. G., R. S. Sutton, and C. W. Anderson. 1983. "Neuronlike adaptive elements that can solve difficult learning control problems". *IEEE transactions on systems, man, and cybernetics.* (5): 834–846.

Beattie, C., J. Z. Leibo, D. Teplyashin, T. Ward, M. Wainwright, H. Küttler, A. Lefrancq, S. Green, V. Valdés, A. Sadik, *et al.* 2016. "DeepMind Lab". *arXiv preprint arXiv:1612.03801.*

Bellemare, M. G., P. S. Castro, C. Gelada, K. Saurabh, and S. Moitra. 2018. "Dopamine". https://github.com/google/dopamine.

Bellemare, M. G., W. Dabney, and R. Munos. 2017. "A distributional perspective on reinforcement learning". *arXiv preprint arXiv:1707.06887.*

Bellemare, M. G., Y. Naddaf, J. Veness, and M. Bowling. 2013. "The Arcade Learning Environment: An evaluation platform for general agents." *Journal of Artificial Intelligence Research.* 47: 253–279.

Bellemare, M. G., S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos. 2016. "Unifying Count-Based Exploration and Intrinsic Motivation". *arXiv preprint arXiv:1606.01868.*

Bellman, R. 1957a. "A Markovian decision process". *Journal of Mathematics and Mechanics*: 679–684.

Bellman, R. 1957b. "Dynamic Programming".

Bellman, R. E. and S. E. Dreyfus. 1962. "Applied dynamic programming".

Bello, I., H. Pham, Q. V. Le, M. Norouzi, and S. Bengio. 2016. "Neural Combinatorial Optimization with Reinforcement Learning". *arXiv preprint arXiv:1611.09940.*

Bengio, Y. 2017. "The Consciousness Prior". *arXiv preprint arXiv:1709.08568*.

Bengio, Y., D.-H. Lee, J. Bornschein, T. Mesnard, and Z. Lin. 2015. "Towards biologically plausible deep learning". *arXiv preprint arXiv:1502.04156*.

Bengio, Y., J. Louradour, R. Collobert, and J. Weston. 2009. "Curriculum learning". In: *Proceedings of the 26th annual international conference on machine learning*. ACM. 41–48.

Bennett, C. C. and K. Hauser. 2013. "Artificial intelligence framework for simulating clinical decision-making: A Markov decision process approach". *Artificial intelligence in medicine*. 57(1): 9–19.

Bertsekas, D. P., D. P. Bertsekas, D. P. Bertsekas, and D. P. Bertsekas. 1995. *Dynamic programming and optimal control*. Vol. 1. No. 2. Athena scientific Belmont, MA.

Bojarski, M., D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, *et al.* 2016. "End to end learning for self-driving cars". *arXiv preprint arXiv:1604.07316*.

Bostrom, N. 2017. *Superintelligence*. Dunod.

Bouckaert, R. R. 2003. "Choosing between two learning algorithms based on calibrated tests". In: *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*. 51–58.

Bouckaert, R. R. and E. Frank. 2004. "Evaluating the replicability of significance tests for comparing learning algorithms". In: *PAKDD*. Springer. 3–12.

Boularias, A., J. Kober, and J. Peters. 2011. "Relative Entropy Inverse Reinforcement Learning." In: *AISTATS*. 182–189.

Boyan, J. A. and A. W. Moore. 1995. "Generalization in reinforcement learning: Safely approximating the value function". In: *Advances in neural information processing systems*. 369–376.

Brafman, R. I. and M. Tennenholtz. 2003. "R-max-a general polynomial time algorithm for near-optimal reinforcement learning". *The Journal of Machine Learning Research*. 3: 213–231.

Branavan, S., N. Kushman, T. Lei, and R. Barzilay. 2012. "Learning high-level planning from text". In: *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*. Association for Computational Linguistics. 126–135.

Braziunas, D. 2003. "POMDP solution methods". *University of Toronto, Tech. Rep.*

Brockman, G., V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. 2016. "OpenAI Gym".

Brown, N. and T. Sandholm. 2017. "Libratus: The Superhuman AI for No-Limit Poker". *International Joint Conference on Artificial Intelligence (IJCAI-17)*.

Browne, C. B., E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. 2012. "A survey of monte carlo tree search methods". *IEEE Transactions on Computational Intelligence and AI in games*. 4(1): 1–43.

Brügmann, B. 1993. "Monte carlo go". *Tech. rep.* Citeseer.

Brundage, M., S. Avin, J. Clark, H. Toner, P. Eckersley, B. Garfinkel, A. Dafoe, P. Scharre, T. Zeitzoff, B. Filar, *et al.* 2018. "The Malicious Use of Artificial Intelligence: Forecasting, Prevention, and Mitigation". *arXiv preprint arXiv:1802.07228*.

Brys, T., A. Harutyunyan, P. Vrancx, M. E. Taylor, D. Kudenko, and A. Nowé. 2014. "Multi-objectivization of reinforcement learning problems by reward shaping". In: *Neural Networks (IJCNN), 2014 International Joint Conference on*. IEEE. 2315–2322.

Bubeck, S., R. Munos, and G. Stoltz. 2011. "Pure exploration in finitely-armed and continuous-armed bandits". *Theoretical Computer Science*. 412(19): 1832–1852.

Burda, Y., H. Edwards, A. Storkey, and O. Klimov. 2018. "Exploration by Random Network Distillation". *arXiv preprint arXiv:1810.12894*.

Camerer, C., G. Loewenstein, and D. Prelec. 2005. "Neuroeconomics: How neuroscience can inform economics". *Journal of economic Literature*. 43(1): 9–64.

Campbell, M., A. J. Hoane, and F.-h. Hsu. 2002. "Deep blue". *Artificial intelligence*. 134(1-2): 57–83.

Casadevall, A. and F. C. Fang. 2010. "Reproducible science".

Castronovo, M., V. François-Lavet, R. Fonteneau, D. Ernst, and A. Couëtoux. 2017. "Approximate Bayes Optimal Policy Search using Neural Networks". In: *9th International Conference on Agents and Artificial Intelligence (ICAART 2017)*.

Chebotar, Y., A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox. 2018. "Closing the Sim-to-Real Loop: Adapting Simulation Randomization with Real World Experience". *arXiv preprint arXiv:1810.05687*.

Chen, T., I. Goodfellow, and J. Shlens. 2015. "Net2net: Accelerating learning via knowledge transfer". *arXiv preprint arXiv:1511.05641*.

Chen, X., C. Liu, and D. Song. 2017. "Learning Neural Programs To Parse Programs". *arXiv preprint arXiv:1706.01284*.

Chiappa, S., S. Racaniere, D. Wierstra, and S. Mohamed. 2017. "Recurrent Environment Simulators". *arXiv preprint arXiv:1704.02254*.

Christiano, P., J. Leike, T. B. Brown, M. Martic, S. Legg, and D. Amodei. 2017. "Deep reinforcement learning from human preferences". *arXiv preprint arXiv:1706.03741*.

Christopher, M. B. 2006. *Pattern recognition and machine learning.* Springer.

Cohen, J. D., S. M. McClure, and J. Y. Angela. 2007. "Should I stay or should I go? How the human brain manages the trade-off between exploitation and exploration". *Philosophical Transactions of the Royal Society of London B: Biological Sciences.* 362(1481): 933–942.

Cortes, C. and V. Vapnik. 1995. "Support-vector networks". *Machine learning.* 20(3): 273–297.

Coumans, E., Y. Bai, *et al.* 2016. "Bullet". http://pybullet.org/.

Da Silva, B., G. Konidaris, and A. Barto. 2012. "Learning parameterized skills". *arXiv preprint arXiv:1206.6398*.

Dabney, W., M. Rowland, M. G. Bellemare, and R. Munos. 2017. "Distributional Reinforcement Learning with Quantile Regression". *arXiv preprint arXiv:1710.10044*.

Dayan, P. and N. D. Daw. 2008. "Decision theory, reinforcement learning, and the brain". *Cognitive, Affective, & Behavioral Neuroscience.* 8(4): 429–453.

Dayan, P. and Y. Niv. 2008. "Reinforcement learning: the good, the bad and the ugly". *Current opinion in neurobiology.* 18(2): 185–196.

Dearden, R., N. Friedman, and D. Andre. 1999. "Model based Bayesian exploration". In: *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence.* Morgan Kaufmann Publishers Inc. 150–159.

Dearden, R., N. Friedman, and S. Russell. 1998. "Bayesian Q-learning".

Deisenroth, M. and C. E. Rasmussen. 2011. "PILCO: A model-based and data-efficient approach to policy search". In: *Proceedings of the 28th International Conference on machine learning (ICML-11).* 465–472.

Demšar, J. 2006. "Statistical comparisons of classifiers over multiple data sets". *Journal of Machine learning research.* 7(Jan): 1–30.

Deng, Y., F. Bao, Y. Kong, Z. Ren, and Q. Dai. 2017. "Deep direct reinforcement learning for financial signal representation and trading". *IEEE transactions on neural networks and learning systems.* 28(3): 653–664.

Dhariwal, P., C. Hesse, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu. 2017. "OpenAI Baselines".

Dietterich, T. G. 1998. "Approximate statistical tests for comparing supervised classification learning algorithms". *Neural computation.* 10(7): 1895–1923.

Dietterich, T. G. 2009. "Machine learning and ecosystem informatics: challenges and opportunities". In: *Asian Conference on Machine Learning.* Springer. 1–5.

Dinculescu, M. and D. Precup. 2010. "Approximate predictive representations of partially observable systems". In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10).* 895–902.

Dosovitskiy, A. and V. Koltun. 2016. "Learning to act by predicting the future". *arXiv preprint arXiv:1611.01779.*

Duan, Y., M. Andrychowicz, B. Stadie, J. Ho, J. Schneider, I. Sutskever, P. Abbeel, and W. Zaremba. 2017. "One-Shot Imitation Learning". *arXiv preprint arXiv:1703.07326.*

Duan, Y., X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. 2016a. "Benchmarking deep reinforcement learning for continuous control". In: *International Conference on Machine Learning.* 1329–1338.

Duan, Y., J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel. 2016b. "RL$\hat{2}$: Fast Reinforcement Learning via Slow Reinforcement Learning". *arXiv preprint arXiv:1611.02779.*

Duchesne, L., E. Karangelos, and L. Wehenkel. 2017. "Machine learning of real-time power systems reliability management response". *PowerTech Manchester 2017 Proceedings.*

Džeroski, S., L. De Raedt, and K. Driessens. 2001. "Relational reinforcement learning". *Machine learning.* 43(1-2): 7–52.

Erhan, D., Y. Bengio, A. Courville, and P. Vincent. 2009. "Visualizing higher-layer features of a deep network". *University of Montreal.* 1341(3): 1.

Ernst, D., P. Geurts, and L. Wehenkel. 2005. "Tree-based batch mode reinforcement learning". In: *Journal of Machine Learning Research.* 503–556.

Farquhar, G., T. Rocktäschel, M. Igl, and S. Whiteson. 2017. "TreeQN and ATreeC: Differentiable Tree Planning for Deep Reinforcement Learning". *arXiv preprint arXiv:1710.11417.*

Fazel-Zarandi, M., S.-W. Li, J. Cao, J. Casale, P. Henderson, D. Whitney, and A. Geramifard. 2017. "Learning Robust Dialog Policies in Noisy Environments". *arXiv preprint arXiv:1712.04034.*

Finn, C., P. Abbeel, and S. Levine. 2017. "Model-agnostic meta-learning for fast adaptation of deep networks". *arXiv preprint arXiv:1703.03400.*

Finn, C., I. Goodfellow, and S. Levine. 2016a. "Unsupervised learning for physical interaction through video prediction". In: *Advances In Neural Information Processing Systems.* 64–72.

Finn, C., S. Levine, and P. Abbeel. 2016b. "Guided cost learning: Deep inverse optimal control via policy optimization". In: *Proceedings of the 33rd International Conference on Machine Learning.* Vol. 48.

Florensa, C., Y. Duan, and P. Abbeel. 2017. "Stochastic neural networks for hierarchical reinforcement learning". *arXiv preprint arXiv:1704.03012.*

Florensa, C., D. Held, X. Geng, and P. Abbeel. 2018. "Automatic goal generation for reinforcement learning agents". In: *International Conference on Machine Learning.* 1514–1523.

Foerster, J., R. Y. Chen, M. Al-Shedivat, S. Whiteson, P. Abbeel, and I. Mordatch. 2018. "Learning with opponent-learning awareness". In: *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems. 122–130.

Foerster, J., G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson. 2017a. "Counterfactual Multi-Agent Policy Gradients". *arXiv preprint arXiv:1705.08926*.

Foerster, J., N. Nardelli, G. Farquhar, P. Torr, P. Kohli, S. Whiteson, *et al.* 2017b. "Stabilising experience replay for deep multi-agent reinforcement learning". *arXiv preprint arXiv:1702.08887*.

Fonteneau, R., S. A. Murphy, L. Wehenkel, and D. Ernst. 2013. "Batch mode reinforcement learning based on the synthesis of artificial trajectories". *Annals of operations research*. 208(1): 383–416.

Fonteneau, R., L. Wehenkel, and D. Ernst. 2008. "Variable selection for dynamic treatment regimes: a reinforcement learning approach".

Fortunato, M., M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, *et al.* 2017. "Noisy networks for exploration". *arXiv preprint arXiv:1706.10295*.

Fox, R., A. Pakman, and N. Tishby. 2015. "Taming the noise in reinforcement learning via soft updates". *arXiv preprint arXiv:1512.08562*.

François-Lavet, V. *et al.* 2016a. "DeeR". https://deer.readthedocs.io/.

François-Lavet, V. 2017. "Contributions to deep reinforcement learning and its applications in smartgrids". *PhD thesis*. University of Liege, Belgium.

François-Lavet, V., Y. Bengio, D. Precup, and J. Pineau. 2018. "Combined Reinforcement Learning via Abstract Representations". *arXiv preprint arXiv:1809.04506*.

François-Lavet, V., D. Ernst, and F. Raphael. 2017. "On overfitting and asymptotic bias in batch reinforcement learning with partial observability". *arXiv preprint arXiv:1709.07796*.

François-Lavet, V., R. Fonteneau, and D. Ernst. 2015. "How to Discount Deep Reinforcement Learning: Towards New Dynamic Strategies". *arXiv preprint arXiv:1512.02011*.

François-Lavet, V., D. Taralla, D. Ernst, and R. Fonteneau. 2016b. "Deep Reinforcement Learning Solutions for Energy Microgrids Management". In: *European Workshop on Reinforcement Learning.*

Fukushima, K. and S. Miyake. 1982. "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition". In: *Competition and cooperation in neural nets.* Springer. 267–285.

Gal, Y. and Z. Ghahramani. 2016. "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning". In: *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016.* 1050–1059.

Gandhi, D., L. Pinto, and A. Gupta. 2017. "Learning to Fly by Crashing". *arXiv preprint arXiv:1704.05588.*

Garnelo, M., K. Arulkumaran, and M. Shanahan. 2016. "Towards Deep Symbolic Reinforcement Learning". *arXiv preprint arXiv:1609.05518.*

Gauci, J., E. Conti, Y. Liang, K. Virochsiri, Y. He, Z. Kaden, V. Narayanan, and X. Ye. 2018. "Horizon: Facebook's Open Source Applied Reinforcement Learning Platform". *arXiv preprint arXiv:1811.00260.*

Gelly, S., Y. Wang, R. Munos, and O. Teytaud. 2006. "Modification of UCT with patterns in Monte-Carlo Go".

Geman, S., E. Bienenstock, and R. Doursat. 1992. "Neural networks and the bias/variance dilemma". *Neural computation.* 4(1): 1–58.

Geramifard, A., C. Dann, R. H. Klein, W. Dabney, and J. P. How. 2015. "RLPy: A Value-Function-Based Reinforcement Learning Framework for Education and Research". *Journal of Machine Learning Research.* 16: 1573–1578.

Geurts, P., D. Ernst, and L. Wehenkel. 2006. "Extremely randomized trees". *Machine learning.* 63(1): 3–42.

Ghavamzadeh, M., S. Mannor, J. Pineau, A. Tamar, *et al.* 2015. "Bayesian reinforcement learning: A survey". *Foundations and Trends® in Machine Learning.* 8(5-6): 359–483.

Giusti, A., J. Guzzi, D. C. Cireşan, F.-L. He, J. P. Rodriguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. Di Caro, *et al.* 2016. "A machine learning approach to visual perception of forest trails for mobile robots". *IEEE Robotics and Automation Letters.* 1(2): 661–667.

Goodfellow, I., Y. Bengio, and A. Courville. 2016. *Deep learning.* MIT Press.

Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. 2014. "Generative adversarial nets". In: *Advances in neural information processing systems.* 2672–2680.

Gordon, G. J. 1996. "Stable fitted reinforcement learning". In: *Advances in neural information processing systems.* 1052–1058.

Gordon, G. J. 1999. "Approximate solutions to Markov decision processes". *Robotics Institute*: 228.

Graves, A., G. Wayne, and I. Danihelka. 2014. "Neural turing machines". *arXiv preprint arXiv:1410.5401.*

Gregor, K., D. J. Rezende, and D. Wierstra. 2016. "Variational Intrinsic Control". *arXiv preprint arXiv:1611.07507.*

Gruslys, A., M. G. Azar, M. G. Bellemare, and R. Munos. 2017. "The Reactor: A Sample-Efficient Actor-Critic Architecture". *arXiv preprint arXiv:1704.04651.*

Gu, S., E. Holly, T. Lillicrap, and S. Levine. 2017a. "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates". In: *Robotics and Automation (ICRA), 2017 IEEE International Conference on.* IEEE. 3389–3396.

Gu, S., T. Lillicrap, Z. Ghahramani, R. E. Turner, and S. Levine. 2016a. "Q-prop: Sample-efficient policy gradient with an off-policy critic". *arXiv preprint arXiv:1611.02247.*

Gu, S., T. Lillicrap, Z. Ghahramani, R. E. Turner, and S. Levine. 2017b. "Q-Prop: Sample-Efficient Policy Gradient with An Off-Policy Critic". In: *5th International Conference on Learning Representations (ICLR 2017).*

Gu, S., T. Lillicrap, Z. Ghahramani, R. E. Turner, B. Schölkopf, and S. Levine. 2017c. "Interpolated Policy Gradient: Merging On-Policy and Off-Policy Gradient Estimation for Deep Reinforcement Learning". *arXiv preprint arXiv:1706.00387.*

Gu, S., T. Lillicrap, I. Sutskever, and S. Levine. 2016b. "Continuous Deep Q-Learning with Model-based Acceleration". *arXiv preprint arXiv:1603.00748.*

Guo, Z. D. and E. Brunskill. 2017. "Sample efficient feature selection for factored mdps". *arXiv preprint arXiv:1703.03454.*

Haarnoja, T., H. Tang, P. Abbeel, and S. Levine. 2017. "Reinforcement learning with deep energy-based policies". *arXiv preprint arXiv:1702.08165.*

Haber, N., D. Mrowca, L. Fei-Fei, and D. L. Yamins. 2018. "Learning to Play with Intrinsically-Motivated Self-Aware Agents". *arXiv preprint arXiv:1802.07442.*

Hadfield-Menell, D., S. J. Russell, P. Abbeel, and A. Dragan. 2016. "Cooperative inverse reinforcement learning". In: *Advances in neural information processing systems.* 3909–3917.

Hafner, R. and M. Riedmiller. 2011. "Reinforcement learning in feedback control". *Machine learning.* 84(1-2): 137–169.

Halsey, L. G., D. Curran-Everett, S. L. Vowler, and G. B. Drummond. 2015. "The fickle P value generates irreproducible results". *Nature methods.* 12(3): 179–185.

Harari, Y. N. 2014. *Sapiens: A brief history of humankind.*

Harutyunyan, A., M. G. Bellemare, T. Stepleton, and R. Munos. 2016. "Q (\lambda) with Off-Policy Corrections". In: *International Conference on Algorithmic Learning Theory.* Springer. 305–320.

Hassabis, D., D. Kumaran, C. Summerfield, and M. Botvinick. 2017. "Neuroscience-inspired artificial intelligence". *Neuron.* 95(2): 245–258.

Hasselt, H. V. 2010. "Double Q-learning". In: *Advances in Neural Information Processing Systems.* 2613–2621.

Hausknecht, M. and P. Stone. 2015. "Deep recurrent Q-learning for partially observable MDPs". *arXiv preprint arXiv:1507.06527.*

Hauskrecht, M., N. Meuleau, L. P. Kaelbling, T. Dean, and C. Boutilier. 1998. "Hierarchical solution of Markov decision processes using macro-actions". In: *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence.* Morgan Kaufmann Publishers Inc. 220–229.

He, K., X. Zhang, S. Ren, and J. Sun. 2016. "Deep residual learning for image recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 770–778.

Heess, N., G. Wayne, D. Silver, T. Lillicrap, T. Erez, and Y. Tassa. 2015. "Learning continuous control policies by stochastic value gradients". In: *Advances in Neural Information Processing Systems.* 2944–2952.

Henderson, P., W.-D. Chang, F. Shkurti, J. Hansen, D. Meger, and G. Dudek. 2017a. "Benchmark Environments for Multitask Learning in Continuous Domains". *ICML Lifelong Learning: A Reinforcement Learning Approach Workshop.*

Henderson, P., R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. 2017b. "Deep Reinforcement Learning that Matters". *arXiv preprint arXiv:1709.06560.*

Hessel, M., J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver. 2017. "Rainbow: Combining Improvements in Deep Reinforcement Learning". *arXiv preprint arXiv:1710.02298.*

Hessel, M., H. Soyer, L. Espeholt, W. Czarnecki, S. Schmitt, and H. van Hasselt. 2018. "Multi-task Deep Reinforcement Learning with PopArt". *arXiv preprint arXiv:1809.04474.*

Higgins, I., A. Pal, A. A. Rusu, L. Matthey, C. P. Burgess, A. Pritzel, M. Botvinick, C. Blundell, and A. Lerchner. 2017. "Darla: Improving zero-shot transfer in reinforcement learning". *arXiv preprint arXiv:1707.08475.*

Ho, J. and S. Ermon. 2016. "Generative adversarial imitation learning". In: *Advances in Neural Information Processing Systems.* 4565–4573.

Hochreiter, S. and J. Schmidhuber. 1997. "Long short-term memory". *Neural computation.* 9(8): 1735–1780.

Hochreiter, S., A. S. Younger, and P. R. Conwell. 2001. "Learning to learn using gradient descent". In: *International Conference on Artificial Neural Networks.* Springer. 87–94.

Holroyd, C. B. and M. G. Coles. 2002. "The neural basis of human error processing: reinforcement learning, dopamine, and the error-related negativity." *Psychological review*. 109(4): 679.

Houthooft, R., X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel. 2016. "Vime: Variational information maximizing exploration". In: *Advances in Neural Information Processing Systems*. 1109–1117.

Ioffe, S. and C. Szegedy. 2015. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". *arXiv preprint arXiv:1502.03167*.

Islam, R., P. Henderson, M. Gomrokchi, and D. Precup. 2017. "Reproducibility of Benchmarked Deep Reinforcement Learning Tasks for Continuous Control". *ICML Reproducibility in Machine Learning Workshop*.

Jaderberg, M., V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu. 2016. "Reinforcement learning with unsupervised auxiliary tasks". *arXiv preprint arXiv:1611.05397*.

Jaderberg, M., W. M. Czarnecki, I. Dunning, L. Marris, G. Lever, A. G. Castaneda, C. Beattie, N. C. Rabinowitz, A. S. Morcos, A. Ruderman, *et al.* 2018. "Human-level performance in first-person multiplayer games with population-based deep reinforcement learning". *arXiv preprint arXiv:1807.01281*.

Jakobi, N., P. Husbands, and I. Harvey. 1995. "Noise and the reality gap: The use of simulation in evolutionary robotics". In: *European Conference on Artificial Life*. Springer. 704–720.

James, G. M. 2003. "Variance and bias for general loss functions". *Machine Learning*. 51(2): 115–135.

Jaques, N., A. Lazaridou, E. Hughes, C. Gulcehre, P. A. Ortega, D. Strouse, J. Z. Leibo, and N. de Freitas. 2018. "Intrinsic Social Motivation via Causal Influence in Multi-Agent RL". *arXiv preprint arXiv:1810.08647*.

Jaquette, S. C. *et al.* 1973. "Markov decision processes with a new optimality criterion: Discrete time". *The Annals of Statistics*. 1(3): 496–505.

Jiang, N., A. Kulesza, and S. Singh. 2015a. "Abstraction selection in model-based reinforcement learning". In: *Proceedings of the 32nd International Conference on Machine Learning (ICML-15).* 179–188.

Jiang, N., A. Kulesza, S. Singh, and R. Lewis. 2015b. "The Dependence of Effective Planning Horizon on Model Accuracy". In: *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems.* International Foundation for Autonomous Agents and Multiagent Systems. 1181–1189.

Jiang, N. and L. Li. 2016. "Doubly robust off-policy value evaluation for reinforcement learning". In: *Proceedings of The 33rd International Conference on Machine Learning.* 652–661.

Johnson, J., B. Hariharan, L. van der Maaten, J. Hoffman, L. Fei-Fei, C. L. Zitnick, and R. Girshick. 2017. "Inferring and Executing Programs for Visual Reasoning". *arXiv preprint arXiv:1705.03633.*

Johnson, M., K. Hofmann, T. Hutton, and D. Bignell. 2016. "The Malmo Platform for Artificial Intelligence Experimentation." In: *IJCAI.* 4246–4247.

Juliani, A., V.-P. Berges, E. Vckay, Y. Gao, H. Henry, M. Mattar, and D. Lange. 2018. "Unity: A General Platform for Intelligent Agents". *arXiv preprint arXiv:1809.02627.*

Kaelbling, L. P., M. L. Littman, and A. R. Cassandra. 1998. "Planning and acting in partially observable stochastic domains". *Artificial intelligence.* 101(1): 99–134.

Kahneman, D. 2011. *Thinking, fast and slow.* Macmillan.

Kakade, S. 2001. "A Natural Policy Gradient". In: *Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada].* 1531–1538.

Kakade, S., M. Kearns, and J. Langford. 2003. "Exploration in metric state spaces". In: *ICML.* Vol. 3. 306–312.

Kalakrishnan, M., P. Pastor, L. Righetti, and S. Schaal. 2013. "Learning objective functions for manipulation". In: *Robotics and Automation (ICRA), 2013 IEEE International Conference on.* IEEE. 1331–1336.

Kalashnikov, D., A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine. 2018. "Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation". *arXiv preprint arXiv:1806.10293.*

Kalchbrenner, N., A. v. d. Oord, K. Simonyan, I. Danihelka, O. Vinyals, A. Graves, and K. Kavukcuoglu. 2016. "Video pixel networks". *arXiv preprint arXiv:1610.00527.*

Kansky, K., T. Silver, D. A. Mély, M. Eldawy, M. Lázaro-Gredilla, X. Lou, N. Dorfman, S. Sidor, S. Phoenix, and D. George. 2017. "Schema Networks: Zero-shot Transfer with a Generative Causal Model of Intuitive Physics". *arXiv preprint arXiv:1706.04317.*

Kaplan, R., C. Sauer, and A. Sosa. 2017. "Beating Atari with Natural Language Guided Reinforcement Learning". *arXiv preprint arXiv:1704.05539.*

Kearns, M. and S. Singh. 2002. "Near-optimal reinforcement learning in polynomial time". *Machine Learning.* 49(2-3): 209–232.

Kempka, M., M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski. 2016. "Vizdoom: A doom-based ai research platform for visual reinforcement learning". In: *Computational Intelligence and Games (CIG), 2016 IEEE Conference on.* IEEE. 1–8.

Kirkpatrick, J., R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, *et al.* 2016. "Overcoming catastrophic forgetting in neural networks". *arXiv preprint arXiv:1612.00796.*

Klambauer, G., T. Unterthiner, A. Mayr, and S. Hochreiter. 2017. "Self-Normalizing Neural Networks". *arXiv preprint arXiv:1706.02515.*

Kolter, J. Z. and A. Y. Ng. 2009. "Near-Bayesian exploration in polynomial time". In: *Proceedings of the 26th Annual International Conference on Machine Learning.* ACM. 513–520.

Konda, V. R. and J. N. Tsitsiklis. 2000. "Actor-critic algorithms". In: *Advances in neural information processing systems.* 1008–1014.

Krizhevsky, A., I. Sutskever, and G. E. Hinton. 2012. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems.* 1097–1105.

Kroon, M. and S. Whiteson. 2009. "Automatic feature selection for model-based reinforcement learning in factored MDPs". In: *Machine Learning and Applications, 2009. ICMLA'09. International Conference on.* IEEE. 324–330.

Kulkarni, T. D., K. Narasimhan, A. Saeedi, and J. Tenenbaum. 2016. "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation". In: *Advances in Neural Information Processing Systems.* 3675–3683.

Lample, G. and D. S. Chaplot. 2017. "Playing FPS Games with Deep Reinforcement Learning." In: *AAAI.* 2140–2146.

LeCun, Y., Y. Bengio, and G. Hinton. 2015. "Deep learning". *Nature.* 521(7553): 436–444.

LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner. 1998. "Gradient-based learning applied to document recognition". *Proceedings of the IEEE.* 86(11): 2278–2324.

LeCun, Y., Y. Bengio, *et al.* 1995. "Convolutional networks for images, speech, and time series". *The handbook of brain theory and neural networks.* 3361(10): 1995.

Lee, D., H. Seo, and M. W. Jung. 2012. "Neural basis of reinforcement learning and decision making". *Annual review of neuroscience.* 35: 287–308.

Leffler, B. R., M. L. Littman, and T. Edmunds. 2007. "Efficient reinforcement learning with relocatable action models". In: *AAAI.* Vol. 7. 572–577.

Levine, S., C. Finn, T. Darrell, and P. Abbeel. 2016. "End-to-end training of deep visuomotor policies". *Journal of Machine Learning Research.* 17(39): 1–40.

Levine, S. and V. Koltun. 2013. "Guided policy search". In: *International Conference on Machine Learning.* 1–9.

Li, L., Y. Lv, and F.-Y. Wang. 2016. "Traffic signal timing via deep reinforcement learning". *IEEE/CAA Journal of Automatica Sinica.* 3(3): 247–254.

Li, L., W. Chu, J. Langford, and X. Wang. 2011. "Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms". In: *Proceedings of the fourth ACM international conference on Web search and data mining.* ACM. 297–306.

Li, X., L. Li, J. Gao, X. He, J. Chen, L. Deng, and J. He. 2015. "Recurrent reinforcement learning: a hybrid approach". *arXiv preprint arXiv:1509.03044.*

Liaw, A., M. Wiener, *et al.* 2002. "Classification and regression by randomForest". *R news.* 2(3): 18–22.

Lillicrap, T. P., J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. 2015. "Continuous control with deep reinforcement learning". *arXiv preprint arXiv:1509.02971.*

Lin, L.-J. 1992. "Self-improving reactive agents based on reinforcement learning, planning and teaching". *Machine learning.* 8(3-4): 293–321.

Lipton, Z. C., J. Gao, L. Li, X. Li, F. Ahmed, and L. Deng. 2016. "Efficient exploration for dialogue policy learning with BBQ networks & replay buffer spiking". *arXiv preprint arXiv:1608.05081.*

Littman, M. L. 1994. "Markov games as a framework for multi-agent reinforcement learning". In: *Proceedings of the eleventh international conference on machine learning.* Vol. 157. 157–163.

Liu, Y., A. Gupta, P. Abbeel, and S. Levine. 2017. "Imitation from Observation: Learning to Imitate Behaviors from Raw Video via Context Translation". *arXiv preprint arXiv:1707.03374.*

Lowe, R., Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch. 2017. "Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments". *arXiv preprint arXiv:1706.02275.*

MacGlashan, J., M. K. Ho, R. Loftin, B. Peng, D. Roberts, M. E. Taylor, and M. L. Littman. 2017. "Interactive Learning from Policy-Dependent Human Feedback". *arXiv preprint arXiv:1701.06049.*

Machado, M. C., M. G. Bellemare, and M. Bowling. 2017a. "A Laplacian Framework for Option Discovery in Reinforcement Learning". *arXiv preprint arXiv:1703.00956.*

Machado, M. C., M. G. Bellemare, E. Talvitie, J. Veness, M. Hausknecht, and M. Bowling. 2017b. "Revisiting the Arcade Learning Environment: Evaluation Protocols and Open Problems for General Agents". *arXiv preprint arXiv:1709.06009.*

Mandel, T., Y.-E. Liu, S. Levine, E. Brunskill, and Z. Popovic. 2014. "Offline policy evaluation across representations with applications to educational games". In: *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*. International Foundation for Autonomous Agents and Multiagent Systems. 1077–1084.

Mankowitz, D. J., T. A. Mann, and S. Mannor. 2016. "Adaptive Skills Adaptive Partitions (ASAP)". In: *Advances in Neural Information Processing Systems*. 1588–1596.

Mathieu, M., C. Couprie, and Y. LeCun. 2015. "Deep multi-scale video prediction beyond mean square error". *arXiv preprint arXiv:1511.05440*.

Matiisen, T., A. Oliver, T. Cohen, and J. Schulman. 2017. "Teacher-Student Curriculum Learning". *arXiv preprint arXiv:1707.00183*.

McCallum, A. K. 1996. "Reinforcement learning with selective perception and hidden state". *PhD thesis*. University of Rochester.

McGovern, A., R. S. Sutton, and A. H. Fagg. 1997. "Roles of macro-actions in accelerating reinforcement learning". In: *Grace Hopper celebration of women in computing*. Vol. 1317.

Miikkulainen, R., J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, A. Navruzyan, N. Duffy, and B. Hodjat. 2017. "Evolving Deep Neural Networks". *arXiv preprint arXiv:1703.00548*.

Mirowski, P., R. Pascanu, F. Viola, H. Soyer, A. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, *et al.* 2016. "Learning to navigate in complex environments". *arXiv preprint arXiv:1611.03673*.

Mnih, V., A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. 2016. "Asynchronous methods for deep reinforcement learning". In: *International Conference on Machine Learning*.

Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.* 2015. "Human-level control through deep reinforcement learning". *Nature*. 518(7540): 529–533.

Mohamed, S. and D. J. Rezende. 2015. "Variational information maximisation for intrinsically motivated reinforcement learning". In: *Advances in neural information processing systems*. 2125–2133.

Montague, P. R. 2013. "Reinforcement Learning Models Then-and-Now: From Single Cells to Modern Neuroimaging". In: *20 Years of Computational Neuroscience*. Springer. 271–277.

Moore, A. W. 1990. "Efficient memory-based learning for robot control".

Morari, M. and J. H. Lee. 1999. "Model predictive control: past, present and future". *Computers & Chemical Engineering*. 23(4-5): 667–682.

Moravčik, M., M. Schmid, N. Burch, V. Lisy, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, and M. Bowling. 2017. "DeepStack: Expert-level artificial intelligence in heads-up no-limit poker". *Science*. 356(6337): 508–513.

Mordatch, I., K. Lowrey, G. Andrew, Z. Popovic, and E. V. Todorov. 2015. "Interactive control of diverse complex characters with neural networks". In: *Advances in Neural Information Processing Systems*. 3132–3140.

Morimura, T., M. Sugiyama, H. Kashima, H. Hachiya, and T. Tanaka. 2010. "Nonparametric return distribution approximation for reinforcement learning". In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*. 799–806.

Munos, R. and A. Moore. 2002. "Variable resolution discretization in optimal control". *Machine learning*. 49(2): 291–323.

Munos, R., T. Stepleton, A. Harutyunyan, and M. Bellemare. 2016. "Safe and efficient off-policy reinforcement learning". In: *Advances in Neural Information Processing Systems*. 1046–1054.

Murphy, K. P. 2012. "Machine Learning: A Probabilistic Perspective."

Nagabandi, A., G. Kahn, R. S. Fearing, and S. Levine. 2017. "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning". *arXiv preprint arXiv:1708.02596*.

Nagabandi, A., G. Kahn, R. S. Fearing, and S. Levine. 2018. "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 7559–7566.

Narvekar, S., J. Sinapov, M. Leonetti, and P. Stone. 2016. "Source task creation for curriculum learning". In: *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems. 566–574.

Neelakantan, A., Q. V. Le, and I. Sutskever. 2015. "Neural programmer: Inducing latent programs with gradient descent". *arXiv preprint arXiv:1511.04834*.

Neu, G. and C. Szepesvári. 2012. "Apprenticeship learning using inverse reinforcement learning and gradient methods". *arXiv preprint arXiv:1206.5264*.

Ng, A. Y., D. Harada, and S. Russell. 1999. "Policy invariance under reward transformations: Theory and application to reward shaping". In: *ICML*. Vol. 99. 278–287.

Ng, A. Y., S. J. Russell, *et al.* 2000. "Algorithms for inverse reinforcement learning." In: *Icml*. 663–670.

Nguyen, D. H. and B. Widrow. 1990. "Neural networks for self-learning control systems". *IEEE Control systems magazine*. 10(3): 18–23.

Niv, Y. 2009. "Reinforcement learning in the brain". *Journal of Mathematical Psychology*. 53(3): 139–154.

Niv, Y. and P. R. Montague. 2009. "Theoretical and empirical studies of learning". In: *Neuroeconomics*. Elsevier. 331–351.

Norris, J. R. 1998. *Markov chains*. No. 2. Cambridge university press.

O'Donoghue, B., R. Munos, K. Kavukcuoglu, and V. Mnih. 2016. "PGQ: Combining policy gradient and Q-learning". *arXiv preprint arXiv:1611.01626*.

Oh, J., V. Chockalingam, S. Singh, and H. Lee. 2016. "Control of Memory, Active Perception, and Action in Minecraft". *arXiv preprint arXiv:1605.09128*.

Oh, J., X. Guo, H. Lee, R. L. Lewis, and S. Singh. 2015. "Action-conditional video prediction using deep networks in atari games". In: *Advances in Neural Information Processing Systems*. 2863–2871.

Oh, J., S. Singh, and H. Lee. 2017. "Value Prediction Network". *arXiv preprint arXiv:1707.03497*.

Olah, C., A. Mordvintsev, and L. Schubert. 2017. "Feature Visualization". *Distill*. https://distill.pub/2017/feature-visualization.

Ortner, R., O.-A. Maillard, and D. Ryabko. 2014. "Selecting near-optimal approximate state representations in reinforcement learning". In: *International Conference on Algorithmic Learning Theory*. Springer. 140–154.

Osband, I., C. Blundell, A. Pritzel, and B. Van Roy. 2016. "Deep Exploration via Bootstrapped DQN". *arXiv preprint arXiv:1602.04621*.

Ostrovski, G., M. G. Bellemare, A. v. d. Oord, and R. Munos. 2017. "Count-based exploration with neural density models". *arXiv preprint arXiv:1703.01310*.

Paine, T. L., S. G. Colmenarejo, Z. Wang, S. Reed, Y. Aytar, T. Pfaff, M. W. Hoffman, G. Barth-Maron, S. Cabi, D. Budden, *et al.* 2018. "One-Shot High-Fidelity Imitation: Training Large-Scale Deep Nets with RL". *arXiv preprint arXiv:1810.05017*.

Parisotto, E., J. L. Ba, and R. Salakhutdinov. 2015. "Actor-mimic: Deep multitask and transfer reinforcement learning". *arXiv preprint arXiv:1511.06342*.

Pascanu, R., Y. Li, O. Vinyals, N. Heess, L. Buesing, S. Racanière, D. Reichert, T. Weber, D. Wierstra, and P. Battaglia. 2017. "Learning model-based planning from scratch". *arXiv preprint arXiv:1707.06170*.

Pathak, D., P. Agrawal, A. A. Efros, and T. Darrell. 2017. "Curiosity-driven exploration by self-supervised prediction". In: *International Conference on Machine Learning (ICML)*. Vol. 2017.

Pavlov, I. P. 1927. *Conditioned reflexes*. Oxford University Press.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.* 2011. "Scikit-learn: Machine learning in Python". *Journal of Machine Learning Research*. 12(Oct): 2825–2830.

Peng, J. and R. J. Williams. 1994. "Incremental multi-step Q-learning". In: *Machine Learning Proceedings 1994*. Elsevier. 226–232.

Peng, P., Q. Yuan, Y. Wen, Y. Yang, Z. Tang, H. Long, and J. Wang. 2017a. "Multiagent Bidirectionally-Coordinated Nets for Learning to Play StarCraft Combat Games". *arXiv preprint arXiv:1703.10069*.

Peng, X. B., G. Berseth, K. Yin, and M. van de Panne. 2017b. "DeepLoco: Dynamic Locomotion Skills Using Hierarchical Deep Reinforcement Learning". *ACM Transactions on Graphics (Proc. SIGGRAPH 2017)*. 36(4).

Perez-Liebana, D., S. Samothrakis, J. Togelius, T. Schaul, S. M. Lucas, A. Couëtoux, J. Lee, C.-U. Lim, and T. Thompson. 2016. "The 2014 general video game playing competition". *IEEE Transactions on Computational Intelligence and AI in Games*. 8(3): 229–243.

Petrik, M. and B. Scherrer. 2009. "Biasing approximate dynamic programming with a lower discount factor". In: *Advances in neural information processing systems*. 1265–1272.

Piketty, T. 2013. "Capital in the Twenty-First Century".

Pineau, J., G. Gordon, S. Thrun, *et al.* 2003. "Point-based value iteration: An anytime algorithm for POMDPs". In: *IJCAI*. Vol. 3. 1025–1032.

Pinto, L., M. Andrychowicz, P. Welinder, W. Zaremba, and P. Abbeel. 2017. "Asymmetric Actor Critic for Image-Based Robot Learning". *arXiv preprint arXiv:1710.06542*.

Plappert, M., R. Houthooft, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel, and M. Andrychowicz. 2017. "Parameter Space Noise for Exploration". *arXiv preprint arXiv:1706.01905*.

Precup, D. 2000. "Eligibility traces for off-policy policy evaluation". *Computer Science Department Faculty Publication Series*: 80.

Ranzato, M., S. Chopra, M. Auli, and W. Zaremba. 2015. "Sequence level training with recurrent neural networks". *arXiv preprint arXiv:1511.06732*.

Rasmussen, C. E. 2004. "Gaussian processes in machine learning". In: *Advanced lectures on machine learning*. Springer. 63–71.

Ravindran, B. and A. G. Barto. 2004. "An algebraic approach to abstraction in reinforcement learning". *PhD thesis*. University of Massachusetts at Amherst.

Real, E., S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, Q. Le, and A. Kurakin. 2017. "Large-Scale Evolution of Image Classifiers". *arXiv preprint arXiv:1703.01041*.

Reed, S. and N. De Freitas. 2015. "Neural programmer-interpreters". *arXiv preprint arXiv:1511.06279*.

Rescorla, R. A., A. R. Wagner, *et al.* 1972. "A theory of Pavlovian conditioning: Variations in the effectiveness of reinforcement and nonreinforcement". *Classical conditioning II: Current research and theory.* 2: 64–99.

Riedmiller, M. 2005. "Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method". In: *Machine Learning: ECML 2005.* Springer. 317–328.

Riedmiller, M., R. Hafner, T. Lampe, M. Neunert, J. Degrave, T. Van de Wiele, V. Mnih, N. Heess, and J. T. Springenberg. 2018. "Learning by Playing - Solving Sparse Reward Tasks from Scratch". *arXiv preprint arXiv:1802.10567.*

Rowland, M., M. G. Bellemare, W. Dabney, R. Munos, and Y. W. Teh. 2018. "An Analysis of Categorical Distributional Reinforcement Learning". *arXiv preprint arXiv:1802.08163.*

Ruder, S. 2017. "An overview of multi-task learning in deep neural networks". *arXiv preprint arXiv:1706.05098.*

Rumelhart, D. E., G. E. Hinton, and R. J. Williams. 1988. "Learning representations by back-propagating errors". *Cognitive modeling.* 5(3): 1.

Russakovsky, O., J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.* 2015. "Imagenet large scale visual recognition challenge". *International Journal of Computer Vision.* 115(3): 211–252.

Russek, E. M., I. Momennejad, M. M. Botvinick, S. J. Gershman, and N. D. Daw. 2017. "Predictive representations can link model-based reinforcement learning to model-free mechanisms". *bioRxiv*: 083857.

Rusu, A. A., S. G. Colmenarejo, C. Gulcehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell. 2015. "Policy distillation". *arXiv preprint arXiv:1511.06295.*

Rusu, A. A., M. Vecerik, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell. 2016. "Sim-to-real robot learning from pixels with progressive nets". *arXiv preprint arXiv:1610.04286.*

Sadeghi, F. and S. Levine. 2016. "CAD2RL: Real single-image flight without a single real image". *arXiv preprint arXiv:1611.04201.*

Salge, C., C. Glackin, and D. Polani. 2014. "Changing the environment based on empowerment as intrinsic motivation". *Entropy.* 16(5): 2789–2819.

Salimans, T., J. Ho, X. Chen, and I. Sutskever. 2017. "Evolution Strategies as a Scalable Alternative to Reinforcement Learning". *arXiv preprint arXiv:1703.03864.*

Samuel, A. L. 1959. "Some studies in machine learning using the game of checkers". *IBM Journal of research and development.* 3(3): 210–229.

Sandve, G. K., A. Nekrutenko, J. Taylor, and E. Hovig. 2013. "Ten simple rules for reproducible computational research". *PLoS computational biology.* 9(10): e1003285.

Santoro, A., D. Raposo, D. G. Barrett, M. Malinowski, R. Pascanu, P. Battaglia, and T. Lillicrap. 2017. "A simple neural network module for relational reasoning". *arXiv preprint arXiv:1706.01427.*

Savinov, N., A. Raichuk, R. Marinier, D. Vincent, M. Pollefeys, T. Lillicrap, and S. Gelly. 2018. "Episodic Curiosity through Reachability". *arXiv preprint arXiv:1810.02274.*

Schaarschmidt, M., A. Kuhnle, and K. Fricke. 2017. "TensorForce: A TensorFlow library for applied reinforcement learning".

Schaul, T., J. Bayer, D. Wierstra, Y. Sun, M. Felder, F. Sehnke, T. Rückstieß, and J. Schmidhuber. 2010. "PyBrain". *The Journal of Machine Learning Research.* 11: 743–746.

Schaul, T., D. Horgan, K. Gregor, and D. Silver. 2015a. "Universal value function approximators". In: *Proceedings of the 32nd International Conference on Machine Learning (ICML-15).* 1312–1320.

Schaul, T., J. Quan, I. Antonoglou, and D. Silver. 2015b. "Prioritized Experience Replay". *arXiv preprint arXiv:1511.05952.*

Schmidhuber, J. 2010. "Formal theory of creativity, fun, and intrinsic motivation (1990–2010)". *IEEE Transactions on Autonomous Mental Development.* 2(3): 230–247.

Schmidhuber, J. 2015. "Deep learning in neural networks: An overview". *Neural Networks.* 61: 85–117.

Schraudolph, N. N., P. Dayan, and T. J. Sejnowski. 1994. "Temporal difference learning of position evaluation in the game of Go". In: *Advances in Neural Information Processing Systems.* 817–824.

Schulman, J., P. Abbeel, and X. Chen. 2017a. "Equivalence Between Policy Gradients and Soft Q-Learning". *arXiv preprint arXiv:1704.06440*.

Schulman, J., J. Ho, C. Lee, and P. Abbeel. 2016. "Learning from demonstrations through the use of non-rigid registration". In: *Robotics Research*. Springer. 339–354.

Schulman, J., S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz. 2015. "Trust Region Policy Optimization". In: *ICML*. 1889–1897.

Schulman, J., F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. 2017b. "Proximal policy optimization algorithms". *arXiv preprint arXiv:1707.06347*.

Schultz, W., P. Dayan, and P. R. Montague. 1997. "A neural substrate of prediction and reward". *Science*. 275(5306): 1593–1599.

Shannon, C. 1950. "Programming a Computer for Playing Chess". *Philosophical Magazine*. 41(314).

Silver, D. L., Q. Yang, and L. Li. 2013. "Lifelong Machine Learning Systems: Beyond Learning Algorithms." In: *AAAI Spring Symposium: Lifelong Machine Learning*. Vol. 13. 05.

Silver, D., G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. 2014. "Deterministic Policy Gradient Algorithms". In: *ICML*.

Silver, D., A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.* 2016a. "Mastering the game of Go with deep neural networks and tree search". *Nature*. 529(7587): 484–489.

Silver, D., H. van Hasselt, M. Hessel, T. Schaul, A. Guez, T. Harley, G. Dulac-Arnold, D. Reichert, N. Rabinowitz, A. Barreto, *et al.* 2016b. "The predictron: End-to-end learning and planning". *arXiv preprint arXiv:1612.08810*.

Singh, S. P., T. S. Jaakkola, and M. I. Jordan. 1994. "Learning Without State-Estimation in Partially Observable Markovian Decision Processes." In: *ICML*. 284–292.

Singh, S. P. and R. S. Sutton. 1996. "Reinforcement learning with replacing eligibility traces". *Machine learning*. 22(1-3): 123–158.

Singh, S., T. Jaakkola, M. L. Littman, and C. Szepesvári. 2000. "Convergence results for single-step on-policy reinforcement-learning algorithms". *Machine learning*. 38(3): 287–308.

Sondik, E. J. 1978. "The optimal control of partially observable Markov processes over the infinite horizon: Discounted costs". *Operations research*. 26(2): 282–304.

Srivastava, N., G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. 2014. "Dropout: a simple way to prevent neural networks from overfitting." *Journal of Machine Learning Research*. 15(1): 1929–1958.

Stadie, B. C., S. Levine, and P. Abbeel. 2015. "Incentivizing Exploration In Reinforcement Learning With Deep Predictive Models". *arXiv preprint arXiv:1507.00814*.

Stone, P. and M. Veloso. 2000. "Layered learning". *Machine Learning: ECML 2000*: 369–381.

Story, G., I. Vlaev, B. Seymour, A. Darzi, and R. Dolan. 2014. "Does temporal discounting explain unhealthy behavior? A systematic review and reinforcement learning perspective". *Frontiers in behavioral neuroscience*. 8: 76.

Sukhbaatar, S., A. Szlam, and R. Fergus. 2016. "Learning multiagent communication with backpropagation". In: *Advances in Neural Information Processing Systems*. 2244–2252.

Sun, Y., F. Gomez, and J. Schmidhuber. 2011. "Planning to be surprised: Optimal bayesian exploration in dynamic environments". In: *Artificial General Intelligence*. Springer. 41–51.

Sunehag, P., G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, *et al.* 2017. "Value-Decomposition Networks For Cooperative Multi-Agent Learning". *arXiv preprint arXiv:1706.05296*.

Sutton, R. S. 1988. "Learning to predict by the methods of temporal differences". *Machine learning*. 3(1): 9–44.

Sutton, R. S. 1996. "Generalization in reinforcement learning: Successful examples using sparse coarse coding". *Advances in neural information processing systems*: 1038–1044.

Sutton, R. S. and A. G. Barto. 1998. *Reinforcement learning: An introduction*. Vol. 1. No. 1. MIT press Cambridge.

Sutton, R. S. and A. G. Barto. 2017. *Reinforcement Learning: An Introduction (2nd Edition, in progress)*. MIT Press.

Sutton, R. S., D. A. McAllester, S. P. Singh, and Y. Mansour. 2000. "Policy gradient methods for reinforcement learning with function approximation". In: *Advances in neural information processing systems*. 1057–1063.

Sutton, R. S., D. Precup, and S. Singh. 1999. "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning". *Artificial intelligence*. 112(1-2): 181–211.

Sutton, R. S. 1984. "Temporal credit assignment in reinforcement learning".

Synnaeve, G., N. Nardelli, A. Auvolat, S. Chintala, T. Lacroix, Z. Lin, F. Richoux, and N. Usunier. 2016. "TorchCraft: a Library for Machine Learning Research on Real-Time Strategy Games". *arXiv preprint arXiv:1611.00625*.

Szegedy, C., S. Ioffe, V. Vanhoucke, and A. Alemi. 2016. "Inception-v4, inception-resnet and the impact of residual connections on learning". *arXiv preprint arXiv:1602.07261*.

Szegedy, C., S. Ioffe, V. Vanhoucke, and A. A. Alemi. 2017. "Inception-v4, inception-resnet and the impact of residual connections on learning." In: *AAAI*. Vol. 4. 12.

Tamar, A., S. Levine, P. Abbeel, Y. WU, and G. Thomas. 2016. "Value iteration networks". In: *Advances in Neural Information Processing Systems*. 2146–2154.

Tan, J., T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke. 2018. "Sim-to-Real: Learning Agile Locomotion For Quadruped Robots". *arXiv preprint arXiv:1804.10332*.

Tanner, B. and A. White. 2009. "RL-Glue: Language-independent software for reinforcement-learning experiments". *The Journal of Machine Learning Research*. 10: 2133–2136.

Teh, Y. W., V. Bapst, W. M. Czarnecki, J. Quan, J. Kirkpatrick, R. Hadsell, N. Heess, and R. Pascanu. 2017. "Distral: Robust Multitask Reinforcement Learning". *arXiv preprint arXiv:1707.04175*.

Tesauro, G. 1995. "Temporal difference learning and TD-Gammon". *Communications of the ACM*. 38(3): 58–68.

Tessler, C., S. Givony, T. Zahavy, D. J. Mankowitz, and S. Mannor. 2017. "A Deep Hierarchical Approach to Lifelong Learning in Minecraft." In: *AAAI*. 1553–1561.

Thomas, P. 2014. "Bias in natural actor-critic algorithms". In: *International Conference on Machine Learning*. 441–448.

Thomas, P. S. and E. Brunskill. 2016. "Data-efficient off-policy policy evaluation for reinforcement learning". In: *International Conference on Machine Learning*.

Thrun, S. B. 1992. "Efficient exploration in reinforcement learning".

Tian, Y., Q. Gong, W. Shang, Y. Wu, and C. L. Zitnick. 2017. "ELF: An Extensive, Lightweight and Flexible Research Platform for Real-time Strategy Games". *Advances in Neural Information Processing Systems (NIPS)*.

Tieleman, H. 2012. "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude". *COURSERA: Neural Networks for Machine Learning*.

Tobin, J., R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. 2017. "Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World". *arXiv preprint arXiv:1703.06907*.

Todorov, E., T. Erez, and Y. Tassa. 2012. "MuJoCo: A physics engine for model-based control". In: *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE. 5026–5033.

Tsitsiklis, J. N. and B. Van Roy. 1997. "An analysis of temporal-difference learning with function approximation". *Automatic Control, IEEE Transactions on*. 42(5): 674–690.

Turing, A. M. 1953. "Digital computers applied to games". *Faster than thought*.

Tzeng, E., C. Devin, J. Hoffman, C. Finn, P. Abbeel, S. Levine, K. Saenko, and T. Darrell. 2015. "Adapting deep visuomotor representations with weak pairwise constraints". *arXiv preprint arXiv:1511.07111*.

Ueno, S., M. Osawa, M. Imai, T. Kato, and H. Yamakawa. 2017. ""Re: ROS": Prototyping of Reinforcement Learning Environment for Asynchronous Cognitive Architecture". In: *First International Early Research Career Enhancement School on Biologically Inspired Cognitive Architectures*. Springer. 198–203.

Van Hasselt, H., A. Guez, and D. Silver. 2016. "Deep Reinforcement Learning with Double Q-Learning." In: *AAAI*. 2094–2100.

Vapnik, V. N. 1998. "Statistical learning theory. Adaptive and learning systems for signal processing, communications, and control".

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. 2017. "Attention Is All You Need". *arXiv preprint arXiv:1706.03762.*

Vezhnevets, A., V. Mnih, S. Osindero, A. Graves, O. Vinyals, J. Agapiou, *et al.* 2016. "Strategic attentive writer for learning macro-actions". In: *Advances in Neural Information Processing Systems.* 3486–3494.

Vinyals, O., T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, *et al.* 2017. "StarCraft II: A New Challenge for Reinforcement Learning". *arXiv preprint arXiv:1708.04782.*

Wahlström, N., T. B. Schön, and M. P. Deisenroth. 2015. "From pixels to torques: Policy learning with deep dynamical models". *arXiv preprint arXiv:1502.02251.*

Walsh, T. 2017. *It's Alive!: Artificial Intelligence from the Logic Piano to Killer Robots.* La Trobe University Press.

Wang, J. X., Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick. 2016a. "Learning to reinforcement learn". *arXiv preprint arXiv:1611.05763.*

Wang, Z., V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas. 2016b. "Sample efficient actor-critic with experience replay". *arXiv preprint arXiv:1611.01224.*

Wang, Z., N. de Freitas, and M. Lanctot. 2015. "Dueling network architectures for deep reinforcement learning". *arXiv preprint arXiv:1511.06581.*

Warnell, G., N. Waytowich, V. Lawhern, and P. Stone. 2017. "Deep TAMER: Interactive Agent Shaping in High-Dimensional State Spaces". *arXiv preprint arXiv:1709.10163.*

Watkins, C. J. and P. Dayan. 1992. "Q-learning". *Machine learning.* 8(3-4): 279–292.

Watkins, C. J. C. H. 1989. "Learning from delayed rewards". *PhD thesis.* King's College, Cambridge.

Watter, M., J. Springenberg, J. Boedecker, and M. Riedmiller. 2015. "Embed to control: A locally linear latent dynamics model for control from raw images". In: *Advances in neural information processing systems*. 2746–2754.

Weber, T., S. Racanière, D. P. Reichert, L. Buesing, A. Guez, D. J. Rezende, A. P. Badia, O. Vinyals, N. Heess, Y. Li, *et al.* 2017. "Imagination-Augmented Agents for Deep Reinforcement Learning". *arXiv preprint arXiv:1707.06203*.

Wender, S. and I. Watson. 2012. "Applying reinforcement learning to small scale combat in the real-time strategy game StarCraft: Broodwar". In: *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*. IEEE. 402–408.

Whiteson, S., B. Tanner, M. E. Taylor, and P. Stone. 2011. "Protecting against evaluation overfitting in empirical reinforcement learning". In: *Adaptive Dynamic Programming And Reinforcement Learning (ADPRL), 2011 IEEE Symposium on*. IEEE. 120–127.

Williams, R. J. 1992. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". *Machine learning*. 8(3-4): 229–256.

Wu, Y. and Y. Tian. 2016. "Training agent for first-person shooter game with actor-critic curriculum learning".

Xu, K., J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio. 2015. "Show, attend and tell: Neural image caption generation with visual attention". In: *International Conference on Machine Learning*. 2048–2057.

You, Y., X. Pan, Z. Wang, and C. Lu. 2017. "Virtual to Real Reinforcement Learning for Autonomous Driving". *arXiv preprint arXiv:1704.03952*.

Zamora, I., N. G. Lopez, V. M. Vilches, and A. H. Cordero. 2016. "Extending the OpenAI Gym for robotics: a toolkit for reinforcement learning using ROS and Gazebo". *arXiv preprint arXiv:1608.05742*.

Zhang, A., N. Ballas, and J. Pineau. 2018a. "A Dissection of Overfitting and Generalization in Continuous Reinforcement Learning". *arXiv preprint arXiv:1806.07937*.

Zhang, A., H. Satija, and J. Pineau. 2018b. "Decoupling Dynamics and Reward for Transfer Learning". *arXiv preprint arXiv:1804.10689*.

Zhang, C., O. Vinyals, R. Munos, and S. Bengio. 2018c. "A Study on Overfitting in Deep Reinforcement Learning". *arXiv preprint arXiv:1804.06893*.

Zhang, C., S. Bengio, M. Hardt, B. Recht, and O. Vinyals. 2016. "Understanding deep learning requires rethinking generalization". *arXiv preprint arXiv:1611.03530*.

Zhu, Y., R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi. 2016. "Target-driven visual navigation in indoor scenes using deep reinforcement learning". *arXiv preprint arXiv:1609.05143*.

Ziebart, B. D. 2010. *Modeling purposeful adaptive behavior with the principle of maximum causal entropy*. Carnegie Mellon University.

Zoph, B. and Q. V. Le. 2016. "Neural architecture search with reinforcement learning". *arXiv preprint arXiv:1611.01578*.