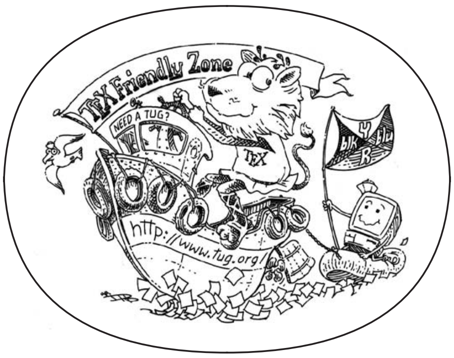
CONTRIBUTIONS TO TRANSFER LEARNING: FROM SUPERVISED TO DEEP REINFORCEMENT LEARNING

MATTHIA SABATELLI



September 2015 – version 4.2

Ohana means family.
Family means nobody gets left behind, or forgotten.

— Lilo & Stitch

Dedicated to the loving memory of Rudolf Miede.

1939 – 2005

ABSTRACT

Short summary of the contents... a great guide by Kent Beck how to write good abstracts can be found here:

https://plg.uwaterloo.ca/~migod/research/beckOOPSLA.html

PUBLICATIONS

Some ideas and figures have appeared previously in the following publications:

Put your publications from the thesis here. The packages `multibib` or `bibtopic` etc. can be used to handle multiple different bibliographies in your document.

You're never gonna grow if you don't grow now
You're never gonna know if you don't find out
You're never going back never turning around
You're never gonna go if you don't go now

ACKNOWLEDGEMENTS

Put your acknowledgements here.

Many thanks to everybody who already sent me a postcard!

Regarding the typography and other help, many thanks go to Marco Kuhlmann, Philipp Lehman, Lothar Schlesier, Jim Young, Lorenzo Pantieri and Enrico Gregorio[1], Jörg Sommer, Joachim Köstler, Daniel Gottschlag, Denis Aydin, Paride Legovini, Steffen Prochnow, Nicolas Repp, Hinrich Harms, Roland Winkler, and the whole LaTeX-community for support, ideas and some great software.

Regarding LyX: The LyX port was initially done by *Nicholas Mariette* in March 2009 and continued by *Ivo Pletikosić* in 2011. Thank you very much for your work and the contributions to the original style.

1 Members of GuIT (Gruppo Italiano Utilizzatori di TeX e LaTeX)

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

DRY Don't Repeat Yourself

API Application Programming Interface

UML Unified Modeling Language

Part I

PRELIMINARIES

1 SUPERVISED LEARNING AND DEEP NEURAL NETWORKS

Outline

structure ...

1.1 INTRODUCTION

2 REINFORCEMENT LEARNING AND DEEP NEURAL NETWORKS

Outline

This chapter introduces the research field of Reinforcement Learning (RL) and presents how its algorithms can successfully be combined with neural networks. The successful marriage between RL and deep neural networks comes with the name of Deep Reinforcement Learning (DRL) and has gained tremendous attention from the machine learning community. Despite its recently gained popularity, however, we will also see that many of the concepts underlying today's most popular DRL breakthroughs date back to a time when training neural networks was not the common practice it is nowadays. We start by providing a general introduction to the field of RL in Sec. 2.1 where we describe the main objectives of this machine learning paradigm and see how it differs from the supervised learning setting that we have described in the previous chapter. We then present the mathematical framework that underpins the development of RL algorithms in Sec. 2.2,2.3 and 2.4. In Sec. 2.5 and Sec. 2.6 we describe how one can create RL algorithms and why it is desirable to integrate the resulting algorithms with neural networks. In Sec. 2.7 we describe the field of DRL and introduce some of the most popular techniques that have been proposed over the years. This chapter ends with Sec. 2.8 where we discuss a few of the main challenges that currently characterize DRL and that have served as inspiration for the research that will be presented in Chapters 7, 8 and 9 of this dissertation.

2.1 INTRODUCTION

In the previous chapter, we have described Supervised Learning (SL), a machine learning framework that aims at constructing models which can answer statistical questions about data coming in the form of input-output pairs. When these models are built successfully, it is possible to use them to make predictions about the behavior of new unseen data. Training SL models is a process which from some perspective is very static. Datasets are divided into training, validation, and testing sets, and besides providing a model with a large set of samples drawn from these datasets, there is no real interaction between the learning algorithm and the data that drives the learning process. In Reinforcement Learning (RL), this drastically changes. The goal is not anymore to learn a mapping between a set of fixed input samples and their respective targets, but to train an algorithm that learns how to interact with an environment. RL is, therefore, a much more dynamic learning paradigm, where the concept of time is omnipresent and is critical for the development of algorithms that not only need to solve a specific problem but additionally also have to be able to adapt themselves while training progresses. What makes RL so challenging is that these algorithms have to learn how to interact with the environment without any form of supervision and therefore can not rely on e.g., examples of successful interactions.

In RL, a learning algorithm is usually denoted as the agent, and it can come in numerous forms: it can range from being a self-driving car that needs to learn how to drive to a recommendation system whose goal is to propose products to users navigating the web. More generally, we define an RL agent as any system that, given a specific situation, has to choose which action to perform. However, there is one more additional component that makes RL the challenging machine learning setting it is. It is not enough for an agent to just learn how to interact with the environment, it is even more desirable for it to learn an interaction which can be defined ‘intelligent". Going back to the self-driving car example, an ‘intelligent" agent would not only be a car that can drive autonomously but a car that is also able to do this while complying with the driving code. Because of this concept of learning how to make (intelligent) decisions while interacting, the problems tackled by RL algorithms are also reffered to as optimal decision making problems, which are also the target of research fields other than machine learning as, control theory. Interestingly both worlds try to solve the same set of problems, one by tackling them through algorithms that are denoted as ‘intelligent", while the other through the development of algorithms that are ‘optimal." Throughout this dissertation, we will not make a clear distinction between these two worlds and will assume that algorithms resulting in intelligent behaviors will correspond to behaviors that are also optimal. Nevertheless, we encourage the reader that has finished reading this chapter to assess whether acting optimally necessarily coincides with acting intelligently.

2.2 MARKOV DECISION PROCESSES

Before starting to develop RL algorithms for sequential decision making problems, we need to formulate the problem within a specific mathematical framework: in RL this framework is that of Markov Decision Processes (MDPs) [99, 100]. Throughout this dissertation, we will characterize MDPs, and the resulting RL concepts, by using the mathematical notation that was used by Sutton and Barto [128] in their seminal book about RL, although it is worth noting that within the literature, different formulations can be found for expressing the same kind of concepts [12–14, 18].

We start by introducing the following elements:

- A set of possible states $\mathcal{S}$, that can be visited by an agent while it is interacting with the MDP, where $s_t \in \mathcal{S}$ denotes the state being visited at time-step t.

- A set of possible actions $\mathcal{A}$ that are available to the agent when it is in a certain state, where $a_t \in \mathcal{A}(s_t)$ denotes the action that is performed by the agent in state s at time-step t.

- A transition function $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \Rightarrow [0, 1]$ that defines the probability for an agent to visit state s_{t+1}, based on its current state and the action which will be performed thereafter.

- A reward function $\Re : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \Rightarrow \mathbb{R}$ which returns a reward signal r_{t+1} when an agent performs action a_t in state s_t and transits to s_{t+1}.

- A discount factor denoted as $\gamma \in [0, 1]$.

Based on these concepts a MDP is defined by the following tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma \rangle$ and is also commonly denoted in the RL literature as the environment. The way the agent interacts with this environment is given by its policy π, defined as a probability distribution over $a \in \mathcal{A}(s)$ for each $s \in \mathcal{S}$:

$$\pi(a|s) = \Pr\{a_t = a | s_t = s\}, \text{ for all } s \in \mathcal{S} \text{ and } a \in \mathcal{A}. \tag{1}$$

A policy can be deterministic if $\forall s : \pi(s, a) = 1$ for exactly one $a \in \mathcal{A}(s)$ and $\pi(s, b) = 0$ for all other $b \in \mathcal{A}(s)$, while it is stationary if it does not change over time. In both cases we can define it as follows:

Definition 1 *A policy is a mapping from states to actions $\pi : \mathcal{S} \Rightarrow \mathcal{A}$.*

The elements of the MDP allow us to properly model the dynamics of an agent interacting with its environment, an interaction which can be summarized as follows: at each time-step t the environment provides the agent with a certain state s_t, the agent then performs action a_t which results into the reward signal r_{t+1}. After performing such action the agent will enter into a new state s_{t+1}. This continuous interaction with the environment is also known as the Reinforcement Learning loop, and can technically be infinite. However, this is never the case in practice, since an agent will eventually visit a state that only transits to itself (denoted as terminal), which will therefore stop the agent-environment interaction. We visually represent the Reinforcement Learning loop in Fig. 1.

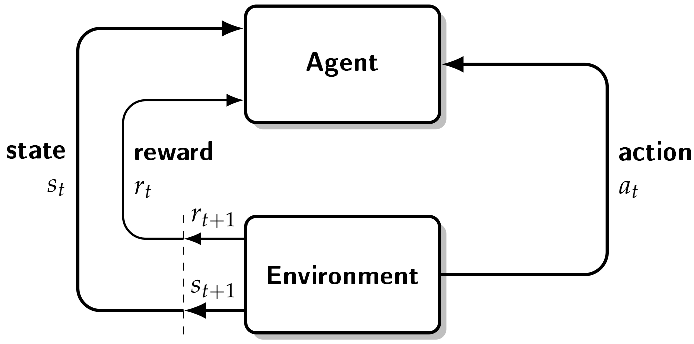


Figure 1: A visual representation of how an agent interacts with an environment as modeled by a Markov decision process. Figure inspired from page 48 of the Sutton and Barto [128] textbook.

Each interaction of the agent with the environment is defined as an episode, which consists of one, or several trajectories τ, that come in the form of the following sequence:

$$\langle(s_t, a_t, r_t, s_{t+1})\rangle, t = 0, \ldots, T-1 \tag{2}$$

where T is a random variable representing the length of the episode.

A key property of the environment is that it fulfills the Markov property which is defined as follows:

Definition 2 *A discrete stochastic process is Markovian if the conditional distribution of the next state of the process only depends from the current state of the process.*

This implies that the only information that is necessary for predicting to which state an agent will step next are s_t and a_t, a concept which can be expressed formally as:

$$p(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \ldots) = p(s_{t+1}|s_t, a_t). \tag{3}$$

Interestingly, the same property also holds for the reward that the agent will get, meaning that the reward that an agent obtains is only determined by its previous action, and not by the history of all previously taken actions, as defined by:

$$(r_t|s_t, a_t, \ldots, s_1, a_1) = p(r_t|s_t, a_t). \tag{4}$$

2.3 GOALS AND RETURNS

So far, we have defined all the elements that model an agent's interaction with an environment while introducing some of its fundamental properties. However, we do not yet know what the purpose of this interaction is. In RL, an agent's goal is defined with respect to the reward signal r_t that is returned by the reward function $\Re$ and is very straightforward: maximizing the total amount of reward it receives while interacting with the environment. In the simplest case, we can define this as:

$$G_t = r_t, r_{t+1}, r_{t+2}, \dots, r_T. \tag{5}$$

While simple and intuitive, this formulation has one major drawback: it treats each reward signal equally since it does not distinguish rewards that are obtained in the near future, r_t, from the ones that will be obtained in the more distant future, r_{T-1}. To deal with this issue, we need an additional concept known as discounting, and that is governed by the discount rate parameter γ, also known as the discount factor. γ allows us to weight the different reward signals based on how close or distant in the future these rewards are received by the agent. By introducing γ in Eq. 5 we can now define the expected discounted return as:

$$\begin{aligned} G_t &= r_t + \gamma r_{t+1}, \gamma^2 r_{t+2} + ... && (6)\\ &= \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}. && (7)\end{aligned}$$

The role of γ can be interpreted as follows: a reward obtained k time steps in the future is only worth γ^{k-1} times what it would be worth if received immediately. It is easy to see how different γ values can result in different agent's behaviors. If $\gamma = 0$ an agent will only take into account immediate rewards, therefore aiming to maximize r_{t+1} only and resulting into having a "myopic" behavior. If γ approaches 1 the agent will become more "far-sighted", it will take future rewards into account more strongly and will therefore increase its chances of accessing rewards that will result into a higher cumulative return. Please note that by defining $\gamma \leq 1$, we can make the infinite sum presented in Eq. 7 finite as long as the sequence of rewards r_k is bounded.

While the role of γ is often taken for granted within the RL literature, it is worth noting that as mentioned by Hessel et al. [51] and Schmidhuber [114], γ is an artificial concept that is not present in fields such as traditional control theory or engineering. This is because γ corresponds to a concept that does not exist in the real world and that in practice distorts the actual value of r_t in an exponentially shrinking fashion. Even if it is considered standard practice to include a discount factor in the development of RL algorithms, it is worth noting that making γ part of the RL framework corresponds to including a form of "inductive bias" within the resulting algorithms. It is common knowledge that low discount factors result in poor performance and that it is therefore as beneficial as possible to set γ as close to 1, yet choosing an appropriate γ parameter can be more challenging than expected, especially when RL algorithms are combined with function approximators. Wiering and Van Hasselt [156] show that different algorithms prefer different discount factors, while François-Lavet, Fonteneau, and Ernst [33] show the benefits of initially starting with a low discount factor which gradually gets increased while training progresses. Finally, Van Seijen, Fatemi, and Tavakoli [145] introduce a method that allows the use of low discount factors for approximate RL algorithms while at the same time highlighting that the common perception of the role of γ might need revision from the RL community.

2.4 VALUE FUNCTIONS

We are now ready to introduce the arguably most important concept underlying many RL algorithms: the concept of value. We can define the value of a state s, as well as the value of a specific policy π or of a particular action a, anyhow, independently from what we are considering, the notion of value is always directly linked to the concept of expected discounted return defined in Eq. 7. Given an MDP and a policy π, we can determine the value of a state s as a function that measures the expected return that the agent will receive when starting in s and following π thereafter.

$$V^{\pi}(s) = \mathbb{E}\bigg[\sum_{k=0}^{\infty}\gamma^k r_{t+k}\bigg| s_t = s, \pi\bigg]. \tag{8}$$

$V^{\pi}(s)$ is also known as the state-value function and intuitively tells us how good or how bad it is for an agent to be in a certain state. While this function is only conditioned on the state that is being visited by the agent, we can also condition it on the actions that the agent takes. By doing so we will quantify how good or bad it is for the agent to take a certain action a in a certain state. This function comes with the name of state-action value function and is defined as follows:

$$Q^{\pi}(s,a) = \mathbb{E}\bigg[\sum_{k=0}^{\infty}\gamma^k r_{t+k}\bigg| s_t = s, a_t = a, \pi\bigg]. \tag{9}$$

Both value functions are very powerful since they allow us to characterize an agent's behavior by quantitatively assessing its interaction with the environment. They can be seen as the agent's knowledge and represent its desirability of being in a specific state. As we will see in the coming sections, accurately modeling these value functions is one of RL's major goals.

A key property of $V^{\pi}(s)$ and $Q^{\pi}(s,a)$ is that both value functions satisfy a consistency condition that allows us to define both functions recursively. For example let us consider the state-value function $V^{\pi}(s)$ presented in Eq. 8, we can rewrite it as:

$$V^{\pi}(s) = \mathbb{E}\big[\sum_{k=0}^{\infty}\gamma^k r_{t+k}\big|s_t = s, \pi\big] \tag{10}$$

$$= \mathbb{E}\big[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \big|s_t = s, \pi\big] \tag{11}$$

$$= \mathbb{E}\big[r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + \ldots)\big|s_t = s, \pi\big] \tag{12}$$

$$= \mathbb{E}\big[r_{t+1} + \gamma V^{\pi}(s_{t+1})\big|s_t = s, \pi\big] \tag{13}$$

$$= \sum_a \pi(s,a) \sum_{s+1} p(s_{t+1}|s,a)\big[\Re(s_t, a, s_{t+1}) + \gamma V^{\pi}(s_{t+1})\big]. \tag{14}$$

Similar steps can be followed when considering $Q^{\pi}(s,a)$ which can then be recursively defined as:

$$Q^{\pi}(s,a) = \sum_{s_{t+1}} p(s_{t+1}|s,a)\big(\Re(s_t,a,s_{t+1}) + \gamma \sum_{a_{t+1}} \pi(s_{t+1},a_{t+1})Q^{\pi}(s_{t+1},a_{t+1})\big). \tag{15}$$

When it comes to sequential decision making, we are interested in maximizing each state's value or each state-action pair, since by doing so, we will be finding a policy π that is optimal. The optimal policy π^* is a policy that realizes the optimal expected return defined as:

$$V^*(s) = \max_{\pi} V^{\pi}(s), \text{ for all } s \in \mathcal{S} \tag{16}$$

and the optimal Q value function:

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \text{ for all } s \in \mathcal{S} \text{ and } a \in \mathcal{A}. \tag{17}$$

When we recursively define both optimal value functions as we did for Eq. 8 we obtain:

$$V^*(s_t) = \max_a \sum_{s_{t+1}} p(s_{t+1}|s_t,a)\Big[\Re(s_t,a,s_{t+1}) + \gamma V^*(s_{t+1})\Big] \tag{18}$$

for the optimal state-value function, and

$$Q^*(s_t,a_t) = \sum_{s_{t+1}} p(s_{t+1}|s_t,a_t)\Big[\Re(s_t,a_t,s_{t+1}) + \gamma \max_a Q^*(s_{t+1},a)\Big], \tag{19}$$

for the optimal state-action value function. Both value functions are well known to correspond to the Bellman optimality equations [10].

If the optimal Q function is learned it becomes a straightforward task to derive an optimal policy since one only needs to select the action which has the highest value in each state as defined by:

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} Q^*(s,a) \text{ for all } s \in \mathcal{S}. \tag{20}$$

It is also worth noting that the Q function and the V function satisfy the following equality

$$V^*(s) = \max_{a \in \mathcal{A}} Q^*(s, a) \text{ for all } s \in \mathcal{S} \tag{21}$$

which comes from the fact $V^*(s) \leq \max_{a \in \mathcal{A}}$ for all $s \in \mathcal{S}$. As we will later see throughout this thesis this equality is particularly important for the development of many RL algorithms.

2.5 LEARNING VALUE FUNCTIONS

The V function and the Q function play a crucial role in the development of optimal decision making algorithms, and over the years, several methods have been introduced to learn them. While all these algorithms' ultimate goal is to yield an optimal policy, there exist cases for which learning these value functions is easier than others. The complexity of learning a value function depends on how many MDP components are known to the agent. If the agent has access to all five of the components of the MDP that we introduced in Sec. 2.2, these algorithms are part of a collection of methods that comes with the name of Dynamic Programming (DP). DP algorithms such as *value-iteration*, *policy-iteration* and variants [11, 151] learn an optimal value function, or optimal policy, by exploiting the fact that the transition function $\mathcal{P}$, and the reward function $\Re$ of the MDP are known. While DP methods can be considered as the progenitors of many RL algorithms, we will not discuss them here since throughout this thesis, we will be interested in scenarios for which $\mathcal{P}$ and $\Re$ are unknown. Specifically, we will introduce novel methods that aim to learn an optimal value function without requiring to learn an approximation of the transition and reward functions ($\widehat{\mathcal{P}}$ and $\widehat{\Re}$) neither, therefore placing all contributions of this dissertation within the model-free RL literature.

2.5.1 *Monte Carlo Methods*

The first family of methods that can learn optimal value functions when no complete knowledge of the environment is available comes with the name of Monte Carlo (MC) methods. MC algorithms only require RL trajectories to discover an optimal policy and achieve this by sampling and averaging the rewards obtained while the agent is interacting with the environment. While MC methods can be used both for learning $V^*(s)$ and for learning $Q^*(s, a)$, in this section, we only present how one can learn the state-value function. MC algorithms' key idea relies on computing the actual sum of discounted rewards that an agent obtains once an episode finishes. This corresponds to computing the quantity defined in Eq. 5. Once this value is computed, it can be used for updating the current value of each state with the following update rule:

$$V(s_t) := V(s_t) + \alpha\big[G_t - V(s_t)\big] \quad (22)$$

where $\alpha \in [0, 1]$ is the learning rate controlling how much we want to change the value estimate of a state based on G_t. As a practical example let us consider the MDP represented in Fig. 2. Let us assume that the starting state of the environment is s_0 while the terminal state of the environment is s_2, and that the agent follows a policy π that results into the following state visits: s_1, s_0, s_2. The rewards associated to each visited state are therefore $-1, +2$ and $+3$ respectively. If we set the discount factor to 0.99 we know that the real discounted return that is obtained at the end of the agent-environment interaction when starting in state s_0 is $\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} = -1 + \gamma 2 + \gamma^2 3 \approx 3.92$. If we now assume that the value of s_0 has never been updated before and that is therefore 0, and that we set $\alpha = 0.5$, the result of one MC update for s_0 will be ≈ 1.96.

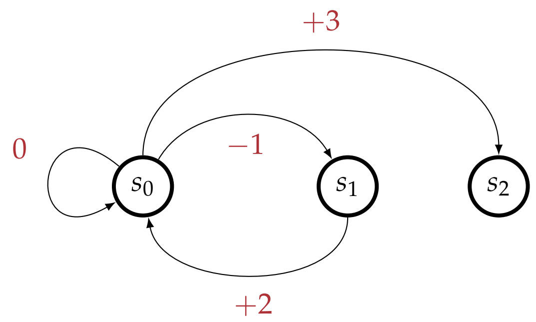


Figure 2: A visual representation of a simple MDP. For each state transition the corresponding reward is presented in red.

When dealing with MC learning, it can be possible that a certain state is visited more than once before a terminal state is reached. In the MDP represented in Fig. 2 this is the case for s_0. If that happens, one must decide when to update $V(s)$ and which value to use as G_t, since different state visits result in different G_t values. There are two typical ways to deal with this: either update $V(s)$ only once, or one can update $V(s)$ each time the state is visited by simply using as G_t the average of all the different discounted returns. The first update strategy comes with the name of , while the latter is denoted as . While several successful applications of MC methods exist [54, 65, 75], a well known issue from this family of algorithms is that they suffer from highly biased updates. In fact, one needs to compute the sum presented in Eq. 5 over all visited states, resulting in returns with considerable variance. It is easy to see how this can become an issue, especially when the length of the episodes increases since the larger the episode's length, the more significant the variance of the updates. Furthermore, an additional drawback of MC methods is that one must wait until the agent visits a terminal state before being able to perform an update that is based on Eq. 22, the latter drawback can result in slow learning and is addressed by the methods are presented hereafter.

2.5.2 *Temporal Difference Learning*

Temporal Difference (TD) Learning [126, 130] is a learning paradigm that allows overcoming the issues mentioned above that characterize MC learning based methods. The key idea of TD-Learning is to update the value of each state with respect to a single MC update, therefore overcoming the hurdle of having to wait for the end of an episode before being able to update the value of a state. Just as MC methods TD-Learning algorithms also learn an optimal value function based on the experience that the agent collects. However, these algorithms base their updates only on the value of a single, consecutive state rather than on the real discounted return that is dependent on the entire sequence of visited states. Updating the value of a state with respect to the value of its successor state only is a technique that comes with the name of bootstrapping, and is a very effective design choice that reduces the variance in the updates. Bootstrapping can be used to learn the V function and the Q function and is at the core of the most popular model-free RL algorithms. The first and simplest form of TD-Learning was introduced by Sutton [126] for learning the state-value function with an algorithm that updates the value of a state based on the following learning rule:

$$V(s_t) := V(s_t) + \alpha\big[r_t + \gamma V(s_{t+1}) - V(s_t)\big]. \tag{23}$$

We can now clearly see that differently from what happens in the MC update presented in Eq. 22 the update of a state now only depends on the reward and the value of the next state. This quantity is denoted as the TD-error δ_t and is defined as:

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t). \tag{24}$$

where $r_t + \gamma V(s_{t+1})$ is also known as the TD-target. If we again consider the simple MDP represented in Fig. 2 and assume that the value of each state of the process is set to 0 while the discount factor γ is again set to 0.5, a TD update for $V(s_0)$ based on action a_3 will result into the new value estimate of 1.5. TD-Learning is a very effective strategy for building algorithms that can learn in an online, fully incremental fashion since one only needs to wait for a single time-step before updating the considered value function. Due to its striking simplicity, TD-Learning has been widely adopted by RL practitioners developing algorithms for learning the Q function. We will present some of the most important algorithms hereafter.

Q-LEARNING: introduced by Watkins and Dayan [150] is arguably the most popular model-free RL algorithm. It works by keeping track of an estimate of the state-action value function $Q : \mathcal{S} \times \mathcal{A} \rightarrow \Re$ and updates each visited state-action pair with the following update rule:

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha \big[r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a_t) - Q(s_t, a_t) \big]. \quad (25)$$

The key component of Q-Learning's update rule is the max operator, which characterizes its TD-error and that is necessary for constructing the TD-target. Since there are as many Q values as there are actions available to the agent, one must choose which Q value to use as a reference when updating the value of the state-action pair that the agent is currently visiting. The max operator simply chooses the state-action pair with the largest Q value, a simple design choice that has the appealing property of making Q-Learning converge to $Q^*(s, a)$ with probability 1 as long as all state-action pairs are visited infinitely often. Interestingly this guarantee holds even if the agent follows a random policy. The max operator also defines Q-Learning as an off-policy learning algorithm, since the Q values chosen for the construction of the TD-target might not correspond to the ones that are associated with the state that the agent will visit after having updated its Q function.

SARSA: also known as 'online Q-Learning" [105] can be seen as the most straightforward extension of the TD-Learning method presented in Eq. 23, and similarly to Q-Learning is an algorithm that aims at learning the state-action value function Q. The key idea of SARSA is to update a state-action value with respect to the Q value that is associated to the state that the agent will visit after a certain action is performed. Therefore, SARSA does not use the max operator within its TD-error and constructs TD-targets that represent the policy that the agent is following, a characteristic that defines SARSA as an on-policy RL algorithm. The way SARSA learns the Q function is given by the following update rule

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha \big[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \big], \quad (26)$$

where we can clearly see how the algorithm uses all the elements of the quintuple of events $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$ a property that gives rise to the name *sarsa*. Not using the max operator in Eq. 26 results in an algorithm that, differently from Q-Learning, does not directly learn the optimal Q function anymore, but rather learns to estimate $Q^{\pi}(s, a)$. This has the drawback of not guaranteeing convergence to $Q^*(s, a)$ for any random policy anymore. To overcome this, SARSA needs an exploration policy that is greedy in the limit of infinite exploration [119]. This can be achieved with the popular $\epsilon -$ greedy selection policy which defines the action that the agent takes as:

$$a_t = \begin{cases} \arg\max_{a \in \mathcal{A}} Q(s_t, a) & \text{with probability } 1 - \epsilon \\ a \sim \mathcal{U}(\mathcal{A}) & \text{with probability } \epsilon \end{cases} \tag{27}$$

where ϵ is a hyperparameter that changes while training progresses. During early training iterations, its value is close to 1, while it approaches 0 by the end of training. This allows the agent to take actions that are representative of a large set of policies when the learned Q function does not yet correspond to $Q^*(s, a)$, while it will favor greedy actions at the end of training. This is a simple yet effective strategy to deal with the exploration-exploitation dilemma, and it is worth noting that its use is not limited to on-policy RL algorithms only. Furthermore, the method presented in Eq. 27 represents only one possible way of balancing exploration and exploitation, and although it is arguably the most popular of such methods, it is not the only existing one. We refer the reader to chapter 5 of [153] for a thorough analysis of different exploration algorithms.

DOUBLE Q-LEARNING: in some environments, Q-Learning is known to perform poorly. This poor performance stems from the fact that the algorithm largely overestimates some state-action values due to the max operator in its TD-error [135]. The max operator serves for constructing an approximation of the maximum expected action-value of a state, which, as discussed by Van Hasselt [142], is a technique that results in positively biased estimates [120, 122]. In some RL problems, this can significantly influence the learning procedure, which has led the RL community to develop a set of solutions that try to mitigate this bias [68, 69, 97, 167]. Among the different solutions, Double Q-Learning [142] is probably the most popular one. Its main idea is to keep track of two different state-action value functions, Q_1 and Q_2, which get alternatively used for selecting which action to perform. When one of the two Q functions determines the action that maximizes the state-action value of the next state, the remaining value function is used for evaluating this estimate. This can be achieved with the following rule:

$$Q_1(s_t, a_t) := Q_1(s_t, a_t) + \alpha\big[r_t + \gamma Q_2(s_{t+1}, a^*) - Q_1(s_t, a_t)\big], \tag{28}$$

where $a^* = \arg\max_{a \in \mathcal{A}}(Q_1(s_{t+1}))$. Note that at each time step, only one of the two Q functions gets updated. While training progresses, the choice of which Q function to update is determined randomly. In the case it is Q_2, the update rule is identical to the one presented in Eq. 28 with the only difference being that the role of the two Q functions is swapped. Double Q-Learning converges to the optimal state-action value function with probability 1 under the same conditions as

Q-Learning. Van Hasselt [142] shows that using two separate Q functions significantly mitigates the overestimation bias, yet this comes at the price of an algorithm that is twice more expensive in terms of memory requirements. It is also worth noting that although Double Q-Learning does not overestimate the state-action values, it might underestimate them, which in some environments can still yield poor performance.

QV(λ)-LEARNING: first introduced by Wiering [155] and further developed by Wiering and Van Hasselt [156] is an on-policy RL algorithm which differently from the previously introduced methods keeps track of an estimate of the state-value function $V : \mathcal{S} \rightarrow \Re$ alongside the usual estimate of the state-action value function $Q : \mathcal{S} \times \mathcal{A} \rightarrow \Re$. Since the goal is to jointly learn two value functions, QV(λ)-Learning requires two separate update rules. The V function is learned via the same form of TD-Learning that we introduced in Eq. 23, with the only difference being the addition of the eligibility traces $e_t(s)$ at the end of the update rule (an RL technique that we will not discuss in this dissertation). QV(λ)-Learning, therefore, learns the V function with the following update rule:

$$V(s) := V(s) + \alpha\big[r_t + \gamma V(s_{t+1}) - V(s_t)\big]e_t(s). \tag{29}$$

Since as discussed earlier only learning the V function is not sufficient for deriving an optimal policy one needs to learn the Q function as well. In QV(λ)-Learning this is done as follows:

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha\big[r_t + \gamma V(s_{t+1}) - Q(s_t, a_t)\big]. \tag{30}$$

An attractive property of the algorithm is that it uses the same TD-target ($r_t + \gamma V(s_{t+1})$) for defining the two different TD-errors that are required for learning the state-value and the state-action value functions. Among the main insights that motivate learning two value functions over one Wiering [155] mentions the possibility that the V function, since it does not depend on the agent's actions might converge faster than the Q function. As described earlier, the V function only depends on the state space of the MDP, which by definition is smaller than the state-action space. For a more in-depth and formal presentation of the conditions that show the benefits of jointly learning the V function alongside the Q function, we refer the reader to chapter 5 of [44].

2.6 FUNCTION APPROXIMATORS

If it is true that model-free RL algorithms are very powerful methods for learning an optimal policy when parts of the MDP are unknown, it is also true that all the algorithms mentioned above suffer from the curse of dimensionality. Model-free algorithms are typically implemented in a tabular fashion, meaning that the state values, or state-action values, are stored within a table of sizes $|\mathcal{S}|$ and $|\mathcal{S} \times \mathcal{A}|$ respectively. Albeit straightforward and easy to implement, such an approach presents severe limitations. The first major drawback of the tabular representation approach is that it does not scale well with respect to the MDP complexity. If the environment's state and action spaces become very large, storing a table quickly becomes unfeasible in terms of storage space. Furthermore, tabular representations are also unable to deal with continuous states. A natural solution to this problem could consist of discretizing the state space; however, this approach still results in the aforementioned storage space issues when done thoroughly. Therefore, if one wants to use RL techniques, even when the state space of the MDP is large, a better solution is needed. This solution is based on parametrized function approximation. In this context, the goal is not to learn the exact value function anymore but to rather replace its tabular representation with a parameterized function. This function's parameters can then be adjusted based on the RL algorithms that we introduced in Sec. 2.5.2.

2.6.1 *Linear Functions*

The most straightforward type of function approximator one can use is a linear function. Given a state-action tuple that gets represented as a feature vector $\mathbf{x} = [f_1(s,a), f_2(s,a), ..., f_i(s,a)] \in \mathbb{R}^2$, and a function parametrized by a vector of parameters θ, as shown in [154] we can redefine the value of a state-action pair as:

$$Q(s,a) = \sum_i \theta_{i,a}\mathbf{x}_i(s). \tag{31}$$

Given a trajectory $\langle s_t, a_t, r_t, s_{t+1}\rangle$ the Q-Learning algorithm presented in Eq. 58 can now be used for updating the parameters θ for all i with the following update rule:

$$\theta_{i,a_t} := \theta_{i,a_t} + \alpha(r_t + \gamma \max_{a\in\mathcal{A}} Q(s_{t+1}, a_t) - Q(s_t, a_t)\mathbf{x}_i(s_t). \tag{32}$$

We can observe that this update rule is equivalent to updating the parameter vector θ for minimizing the mean squared error loss between a given state-action tuple and Q-Learning's TD-target since

$$\mathcal{L}(\theta) = \frac{1}{2}\big(y_t - Q(s_t, a_t)\big)^2 \tag{33}$$

$$\frac{\partial \mathcal{L}}{\partial \theta_{i,a_t}} = -\big(y_t - Q(s_t, a_t)\big)\mathbf{x}_i(s_t) \tag{34}$$

$$\theta_{i,a_t} := \theta_{i,a_t} + \alpha(r_t + \gamma \max_{a\in\mathcal{A}} Q(s_{t+1}, a_t) - Q(s_t, a_t)\mathbf{x}_i(s_t). \tag{35}$$

Similar steps can be used for adapting all of the RL algorithms that we introduced in the previous section. As a representative example for the on-policy learning case let us consider the SARSA algorithm. One can learn an approximation of the Q function by updating the parameters of a linear function as follows:

$$\theta_{i,a_t} := \theta_{i,a_t} + \alpha(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\mathbf{x}_i(s_t). \tag{36}$$

Among the different linear functions one can use there are CMACs [61], radial basis functions (RBFs) [94], linear neural networks [80] and linear support vector machines (SVMs). Although these techniques can successfully deal with the aforementioned curse of dimensionality problem, non-linear functions are usually preferred since their representational power is much larger than the one of linear methods. Throughout this dissertation, we are interested in non-linear functions that come in the form of deep neural networks. As we have seen in the previous chapter, neural networks such as e.g convolutional networks are able to learn very rich representations from their inputs. However, this also makes these kinds of models particularly challenging to train in an RL context. We will now describe how one can successfully deal with some of the challenges that characterize the use of deep neural networks in RL by presenting some of the most important algorithms that have been introduced over the years.

2.7 DEEP REINFORCEMENT LEARNING

Before looking into how RL algorithms should be integrated within deep neural networks, it is important to mention that RL techniques have been successfully used combined with (less powerful) neural networks for over three decades. In fact, the field known as Connectionist Reinforcement Learning (CRL) resulted in the very first algorithms that managed to outperform human experts on specific tasks. Among the multiple possible examples of this family of techniques, we mention the TD-Gammon program introduced by Tesauro [134]. TD-Gammon successfully learns an approximation of the popular Backgammon boardgame's evaluation function through the same TD-Learning methods that we presented in Sec. 2.5.2. Tesauro's program achieved a level of play thatcomparable to the one of the top human Backgammon players of its time and is even nowadays considered one of the most important RL breakthroughs. For a more detailed presentation about the successful applications of CRL algorithms, we refer the reader to [19].

While certainly successful for a certain set of problems (see for example chapter 1 of [107]), CRL techniques also present severe limitations. Since they only use multi-layer perceptrons as function approximators, these algorithms cannot be used for tackling problems where the state representation of the MDP is highly dimensional. To overcome this, more complicated and powerful networks are required.

Deep Reinforcement Learning (DRL) [6, 34, 70] is a research field that combines RL algorithms with deeper and more complex neural architectures. In value based model-free DRL we are interested in learning an approximation of an optimal value function with a deep neural network that comes with parameters θ

$$V(s;\theta) \approx V^*(s) \qquad (37) \qquad\qquad Q(s,a;\theta) \approx Q^*(s,a) \qquad (38)$$

and that usually comes in the form of a convolutional neural network. We now describe some of the algorithms which have contributed to the development of DRL the most.

DEEP Q-LEARNING (DQN): just like Q-Learning is arguably the most important tabular model-free RL algorithm, so is DQN when it comes to DRL. First introduced by Mnih et al. [83] and then made popular by the work presented in [84] this algorithm can certainly be considered as the very first successful example of a neural network that is able to learn an approximation of the optimal state-action value function just from high sensory inputs (in this case images). As the name suggests, Deep Q-Learning (DQN)[1] is based upon the Q-Learning algorithm and aims at learning an approximation of the optimal state-action value function Q. This is done by reshaping Q-Learning's update rule, presented in Eq. 57, into a differentiable loss function that can be used for training a convolutional network with the following objective function:

$$\mathcal{L}(\theta) = \mathbb{E}_{\langle s_t,a_t,r_t,s_{t+1}\rangle \sim U(D)}\bigg[\Big(r_t + \gamma \max_{a\in\mathcal{A}} Q(s_{t+1},a;\theta^-) - Q(s_t,a_t;\theta)\Big)^2\bigg]. \qquad (39)$$

We can start by observing that the general principles that characterize the algorithm are the same ones that made it possible to generalize Q-Learning to the use of linear function approximators, although it is worth noting that differently from when a linear function is used, the mapping between input and feature spaces is now not preserved anymore. Similarly to what we presented in Sec. 2.6.1, we can see from Eq. 58 that learning $Q(s,a,\theta)$ is again achieved by minimizing the squared error loss between the $Q(s_t,a_t;\theta)$ estimates and the off-policy TD-target

$$y_t^{DQN} = r_t + \gamma \max_{a\in\mathcal{A}} Q(s_{t+1},a;\theta^-). \qquad (40)$$

1 In DQN the 'N" in the acronym stays for 'Network" and replaces the, arguably more intuitive, 'L" of Learning.

Despite this similarity, DQN requires some additional algorithmic design choices, without which it would result impossible to successfully train a neural network with Eq. 58. These additions, which significantly make DQN differ from the algorithm presented in Eq. 32 are the following:

- Experience Replay: a memory buffer, D, represented as a queue which stores RL trajectories of the form $\langle s_t, a_t, r_t, s_{t+1}\rangle$. Once this memory buffer is filled with a large set of these quadruples, DQN uniformly samples batches of trajectories for training its network. This makes it possible to exploit past trajectories multiple times by reusing them while training, which makes the overall algorithm more sample efficient. Furthermore, using a memory buffer also improves the stability of the training procedure. Recall that each trajectory is representative of a certain episode. By repeatedly randomly sampling a different τ from the memory buffer, a resulting mini-batch of trajectories will be representative of different episodes and of different policies. As a consequence, the correlation between trajectories within a mini-batch will be small. Although made popular by the DQN algorithm, using an experience replay buffer for tackling sequential decision making problems was already presented by Lin [73].

- Target Network: We can observe from Eq. 59 that the TD-target used by DQN for bootstrapping is not computed by the Q network that is being optimized (θ), but rather from a second separate network that is parameterized with θ^{-}. This second network has the same structure as the main Q network, but its weights do not change each time RL experiences are sampled from D. On the contrary, its weights are temporally frozen and only periodically get updated with the parameters of the main network θ as defined by an appropriate hyperparameter. Note that this is a design choice that is not motivated by the TD-Learning paradigm that we presented in Sec. 2.5.2, where we have seen that TD-Learning based methods learn in a fully online fashion by updating their value estimates based on their own future estimates. With a target-network, although the θ network still learns via the methods of temporal differences, it now requires an auxiliary, external model if it wants to successfully learn $\approx Q^*(s, a; \theta)$. Several works have studied the target network's role to understand why this design choice appears to be necessary for DRL, yet the DRL community does not fully understand the role of θ^{-}. For more about this topic, we refer the reader to .

With all these concepts in place we can show that given a training iteration i, differentiating this objective function with respect to θ gives the following gradient:

$$\nabla_{\theta_i} y_t^{DQN}(\theta_i) = \mathbb{E}_{\langle s_t, a_t, r_t, s_{t+1}\rangle \sim U(D)} \Big[\big(r_t + \\ \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a; \theta_{i-1}^{-}) - Q(s_t, a_t; \theta_i) \big) \nabla_{\theta_i} Q(s_t, a_t; \theta_i) \Big]. \quad (41)$$

The DQN algorithm showcased its entire potential in [84] where Mnih and colleagues developed a convolutional neural network that trained with Eq. 58 learned how to successfully play most of the `Atari` games that are part of the popular Atari Arcade Learning Environment (ALE) [9], a well-known platform that even nowadays serves as a benchmark when for testing the performance of DRL algorithms. In the ALE, a DRL algorithm has to learn how to play 57 different emulations of `Atari` games which are specifically designed within a simulator (see Fig. 3 for a visualization of some of the games that are part of the ALE suite). Remarkably, DQN learned not only learned how to play most of the games of the platform but also achieved a final performance that, on most games, was superior to the one of human expert players. What is even more remarkable is that this was achieved by providing as inputs to the network the images representing the game only, therefore making the model learn just from its own experience in a pure model-free RL fashion. Since then, DQN has been successfully used for a large variety of applications ranging from healthcare [102, 140], robotics [58] and natural language processing [45, 90] to even particle physics [76, 112]. However, despite all these remarkable applications, the algorithm still comes with some drawbacks, some of which are addressed by the algorithms presented below.

DOUBLE DEEP Q-LEARNING (DDQN): Van Hasselt, Guez, and Silver [143] showed that the DQN algorithm suffers from the same issue that also characterizes the Q-Learning algorithm: the overestimation bias of the Q function. They show that DQN is prone to learn overestimated Q-values because the same values are used both for selecting an action ($\max_{a \in \mathcal{A}}$) and for evaluating it ($Q(s_{t+1}, a; \theta^{-})$). This becomes clearer when re-writing DQN's TD-target presented in Eq. 59 as:

$$y_t^{DQN} = r_t + \gamma\, Q(s_{t+1}, \arg\max_{a \in \mathcal{A}} Q(s_{t+1}, a; \theta); \theta^{-}). \quad (42)$$

As a result, DQN tends to approximate the expected maximum value of a state, instead of its maximum expected value. As presented in Sec. 2.5.2, in the tabular case this can be solved by keeping track of two separate Q functions, and by randomly preferring one Q function over the other when it comes to selecting which action to execute.

Figure 3: A visual representation of some of the `Atari` games that are part of the Arcade Learning Environment (ALE) [9]. From left to right `Breakout`, `Seaquest`, `Qbert`, `Pong`, `MsPacman`, `KungFu-Master`, `Enduro` and `Space Invaders`. Most of these games will be of interest in chapters 7 and 8. Image courtesy of .

DDQN generalizes this idea and untangles the action selection process from its evaluation by taking advantage of the previously introduced target network θ^{-}. DDQN's target stays the same as in DQN with the main difference being that the selection of an action, given by the online Q-network θ, and the evaluation of the resulting policy, given by θ^{-}, can now result into smaller overestimations simply by symmetrically updating the two sets of weights (θ and θ^{-}) which can easily be achieved by regularly switching their roles during training. While not always significantly impacting the performance of DQN (one can still act optimally even if some actions are associated to unrealistically high $Q(s, a)$ estimates), there are also cases for which the overestimation bias of the Q function significantly slows down the training process, and even prevents the DQN algorithm from improving its policy over time at all. We will come back to this issue in Chapter 7.

PRIORITIZED EXPERIENCE REPLAY (PER): we have seen that next to the target network θ^{-} an equally important role within the DQN algorithm is played by the experience replay memory buffer D. Schaul et al. [113] showed that the efficiency of how Deep Q-Networks use this buffer could be improved. Their claim stems from the fact that, as shown in Eq. 58, the RL trajectories that get sampled from the memory buffer when constructing a mini-batch of trajectories are sampled uniformly $\sim U(D)$. This approach's main drawback is that it considers each τ stored in the buffer as equally important and representative for training. However, it is easy to imagine learning situations where some trajectories are more valuable than others. For example, at the beginning of training, most of the trajectories contained within D will be representative of early agent-environment interactions. It is, therefore, safe to assume that the network will learn the $Q(s, a)$ estimates representative of these early dynamics much faster than it will learn the Q values, which are associated with trajectories occurring more rarely. The idea of PER is to use only highly informative trajectories when it comes to building the mini-batches that are used for training the network. The importance of different trajectories is given by their respective TD-error. PER ensures that the probability of sampling trajectories is proportional to their respective TD-errors: the higher the TD-error, the larger the probability for a specific τ to be sampled. In practice, given a trajectory τ, the probability of sampling it is given by the following equation:

$$P(\tau) = \frac{p_\tau^\alpha}{\sum_k p_k^\alpha} \tag{43}$$

where p_τ is $|\delta_\tau + \epsilon|$ with ϵ being a small positive number ensuring that the probability of sampling a trajectory remains positive even in the edge case where the TD-error is 0. Although simple and intuitive implementing a PER buffer is not that straightforward and still presents some algorithmic caveats that need to be taken into account. Yet, if done correctly it dramatically improves the sample efficiency of Deep Q-Networks [90].

DUELING NETWORKS: while the DDQN algorithm directly tackles a fundamental algorithmic bias that characterizes the way DQN learns the Q function, and PER addresses the inefficiency of its memory buffer, the contribution presented by Wang et al. [148] is of slightly different nature. Their work consists of a novel type of neural architecture called the Dueling Network. This is a contribution that resembles more the kind of progress that is made by the supervised learning community, which, as we discussed in the previous chapter, has put a lot of effort into developing novel neural architectures for tackling computer vision tasks. Nevertheless, Wang's work is a perfect example that showcases how in DRL, carefully designing the function approximator is just as important as properly defining its objective function. A Dueling network is a network that, after performing a series of convolutions, instead of directly outputting the state-action values for a specific state as DQN and DDQN do, adds some intermediate computations. The idea is to estimate the value of a state and the advantages for each action before outputting the final Q values. The state values are computed based on Eq. 8, while the advantage function A is simply the difference between the Q function and the V function:

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s). \tag{44}$$

To successfully estimate state values, advantages, and state-action values, the network requires a specific architecture consisting of three separate streams. Each stream is responsible for estimating one of the three value functions and is initialized with its own parameters $\theta^{(\cdot)}$. The final Q function of the model is then obtained by combining what is learned by each stream as follows:

$$Q(s,a;\theta^{(1)},\theta^{(2)},\theta^{(3)}) = V\big(s;\theta^{(1)},\theta^{(3)}\big)+ \\ \big(A(s,a;\theta^{(1)},\theta^{(2)}) - \max_{a_{t+1}\in\mathcal{A}} A(s,a_{t+1};\theta^{(1)},\theta^{(2)})\big). \quad (45)$$

Building a network with different task-specific streams is a design choice that resembles the way models are built when tackling multi-task classification tasks. However, note that the output of each stream that either estimates the state values or the advantage function gets aggregated in a final layer that estimates the state-action values. Minimizing the objective function presented in Eq. 45 is done by following the same principles that DQN and DDQN also use. The idea of taking into account the V function when learning the Q function is a concept which will come back, in a different flavor, in chapters 7 and 8.

POLICY GRADIENT METHODS: so far we have only considered algorithms that are part of the action-value family of methods, which, as explained in Sec. 2.5 are techniques that derive an optimal policy from a learned Q function. Yet, there is a collection of algorithms that is able to learn a policy directly, and that therefore bypasses the requirement of having to consult state-action values when it comes to action selection. These methods, which have contributed to the development of DRL just as much as action-value methods, come with the name of Policy Gradients. They directly parametrize a policy at each time-step as $\pi(a|s;\theta) = \Pr a_t = a, s_t = s\ \theta_t = \theta$ and seek to optimize the parameters θ such that the performance of the policy is maximized. Note that this is drastically different from all the methods which we have seen so far, where the aim, in fact, was to minimize the TD-error through gradient descent optimization. Since policy gradients aim to maximize their performance, they learn via gradient ascent and therefore update their parameters as follows

$$\theta \leftarrow \theta + \alpha\nabla\xi(\theta). \quad (46)$$

Here α is again the learning rate, and ξ is a measure that quantifies the performance of the policy. Similarly to action-value based methods, one can parametrize a policy either with a linear function or with a deep neural network. When the action space is discrete, it is common practice to parametrize π through the same exponential softmax distribution, which we have seen in the previous chapter when presenting neural networks trained for classification tasks. Therefore we have

$$\pi(a|s;\theta) = \frac{e^{h(s,a;\theta)}}{\sum_b e^{h(s,b;\theta)}} \quad (47)$$

where $h(s, a; \theta)$ is any function approximator. By doing so, policy gradient methods naturally deal with the exploration-exploitation trade-off since they simply assign different probabilities to the different actions. This also allows these methods to approach a deterministic policy (which cannot be achieved when using ϵ-greedy action selection) and makes these algorithms arguably easier to train since learning a policy could be easier than learning state-action returns. The fundamental result which allows optimizing any differentiable policy is the policy gradient theorem [129]. Sutton et al. [129] show that the gradient of π does not depend from the gradient of the state distribution and that it can therefore be expressed as follows:

$$\nabla\xi(\theta) = \sum_s \mu(s) \sum_a Q^\pi(s, a)\nabla\pi(a|s; \theta) \tag{48}$$

where $\mu(s)$ is the stationary on-policy distribution under π. By expressing the gradient as such, it is now possible to optimize π even when the state distribution of the environment is unknown (as discussed in Sec. 2.5). Policy gradient methods can be used for learning a policy through the same kind of techniques which in Sec. 2.5 were used for learning value functions. Within the Monte Carlo setting the arguably most important of such algorithms is REINFORCE [158], while Actor-Critic algorithms [38, 42, 71, 85, 115–117, 149] learn a policy with the additional help of a bootstrapped value function (typically V or A).

RAINBOW: from the examples above, it is clear that much fast progress has been achieved by the DRL community in creating algorithms capable of learning faster and better. Explaining all the individual value-based contributions that have made DRL the popular research field it is nowadays [48], is beyond the scope of this chapter, yet, if there is one algorithm that encapsulates most of the progress that the DRL community has achieved over the last years, that is Rainbow [50]. Rainbow is a single, almighty agent that integrates within the same algorithm most of the important breakthroughs that DRL researchers have introduced over the last decade, ranging from the previously mentioned DDQN algorithm and PER system to more recent techniques such as distributional DRL [8], multi-step learning, distributed training [85] and noisy networks [32] that allow for better exploration.

2.8 CHALLENGES AND DESIDERATA

Similar to what we did for the field of supervised-learning we now end this chapter by presenting some of the main limitations that currently characterize the field of DRL. We briefly describe three of the most significant challenges that have resulted in the research questions we have tried to address throughout this dissertation.

DIVERGENCE & 'THE DEADLY TRIAD" The combination of RL with function approximation can, unfortunately, result in unstable training and algorithms prone to diverge while learning. As a representative example of what it means for an RL algorithm to diverge, let us consider the following MDP firstly introduced by Tsitsiklis and Van Roy [141]. The MDP consists of three states s_0, s_1 and the terminal state s_3. Each state is described by a single scalar feature ψ such that $\psi(s_0) = 1$ and $\psi(s_1) = 2$. The estimated state-value of each state is therefore given by $V(s) = \psi \times \omega$, where ω is the single weight we would like to update. The MDP is represented in Fig. Each state transition is associated with a reward of 0, which means that the optimal weight value for having perfect value predictions is $\omega^* = 0$. Let us now assume that we are updating the state-value function based on an on-policy learning scheme as discussed in Sec. 2.5.2. We then know that each time we are updating the state-value function for s_0, the value of s_1 will, in expectation, also be updated multiple times. This however, changes if we are following an off-policy learning scheme since each time we update the state-value function for s_0, we do not necessarily update the value of s_1 anymore. As shown by Van Hasselt et al. [144] if we would now update ω based on Eq. 23 we would have to modify ω as $\Delta\omega \propto r_t + \gamma(V(s_{t+1}) - V(s_t))$. Which results into $0 + \gamma 2\omega - \omega = \gamma 2\omega - \omega = (2\gamma - 1)\omega$. If we then set the discount factor $\gamma > 0.5$ as is common practice, we can see that we have $2\gamma > 1$, which will make any weight $\omega \neq 0$ be updated away from the desired value of 0.

Sutton and Barto [128] show that the cause of this type of divergence occurs when RL algorithms are combined with three concepts which we have already encountered in this chapter. These concepts are:

- Bootstrapping (Sec. 2.5.2)
- Off-Policy Learning (Sec. 2.5.2)
- Function Approximation (Sec. 2.6)

This combination is known as the 'Deadly Triad" of DRL, and it is well known that if all of these three elements are combined within the same algorithm, divergence cannot be avoided. However, if an agent can bypass the need to rely on either one of these elements, then instability can already be prevented. Several work has addressed the 'Deadly Triad" of DRL [31, 49, 144] to answer the natural question *'which element can be given up when developing DRL algorithms?"* So far, the community seems to agree that giving up on function approximators is clearly not possible. As discussed in Sec. 2.6 even linear functions can already play a crucial role in developing algorithms that deal with large and complex MDPs. On the other hand, it is not yet known what to give up between off-policy learning and bootstrapping; what is known however, is that also because of the 'Deadly Triad", successfully training DRL solutions can be computationally very expensive since long training times are required for training agents which constantly try not to diverge. We will introduce a solution to this issue in Chapter 7.

COMPUTATIONAL RESOURCES: while most of the DRL algorithms presented in this chapter naturally follow from the tabular RL literature and are therefore not particularly hard to implement, successfully training and benchmarking these algorithms is not an equivalently easy task. The aforementioned divergence issue reflects itself in DRL algorithms that work well only if combined with a large range of carefully chosen hyperparameters. If a few of these hyperparameters are not set correctly, the algorithms miserably fail to learn even the most simple tasks. Unfortunately, the intuition of even the most expert DRL practitioner is not enough for identifying which learning parameters work best. As an example, let us consider the previously described Rainbow agent [50]. The learning rate that yields successful training is set to 0.0000625, clearly this is a value that could only be identified after performing an exhaustive grid search over a large set of potential learning rates. If on top of this, we also consider that training an agent on the previously mentioned ALE is a process that can last from taking days to even weeks [57], it is clear that the computational resources that are required for successfully evaluating DRL algorithms might not be accessible to all of the DRL community. For an excellent position paper about this topic, we refer the reader to the work of Obando-Ceron and Castro [91]. It is, therefore, desirable to develop algorithms that do not require extraordinary computational costs or that, if they do, can limit this use by learning and converging faster. We address this issue in Chapter 7 and 8.

GENERALIZATION AND TRANSFERABILITY The limitation mentioned above can naturally be addressed by studying whether agents trained for solving specific tasks can generalize to new unseen problems. If this would be the case, this could have significant practical benefits. DRL algorithms would not have to be trained from scratch each time they face a new problem, limiting the aforementioned requirement of high computational costs significantly. Furthermore, as mentioned in the introduction of this dissertation, there is arguably nothing more intelligent than learning how to solve a problem than learning a solution to a problem that is general enough to a larger set of problems. We thoroughly study the degree of transferability of DRL algorithms in Chapter 8 and present how the RL community has already studied the transfer learning properties of RL algorithms in the coming chapter.

3 TRANSFER LEARNING

Outline

We now present Transfer Learning (TL), a machine learning methodology that aims at creating algorithms that are capable of retaining and reusing previously learned knowledge when getting trained on new, unseen problems. Most of the contributions presented within this dissertation are motivated by TL, therefore, we now introduce the reader to this specific learning paradigm with the goal of providing him/her with all the preliminary knowledge that is necessary for fully understanding the research that will be presented in the coming chapters. We start with a gentle introduction to TL in Sec. 3.1 where we present the main concepts underlying TL and explain why it is desirable to have machine learning models that are transferable. We then show in Sec. 3.2 some practical, high level, examples that visually represent the benefits that come from adopting TL strategies in machine learning. We will then provide more rigorous, mathematical definitions of TL in Sec. 3.3 where we will characterize TL both for supervised learning as for reinforcement learning. In Sec. 3.4 we thoroughly review how TL has already been studied by the deep learning community, and we will again focus on supervised learning and on reinforcement learning. We end this chapter with Sec. 3.5 where we summarize the most relevant TL concepts that underpin the contributions that will be presented throughout in this dissertation.

3.1 INTRODUCTION

3.2 RATIONALE OF TRANSFER LEARNING

Before mathematically defining TL we start by building some intuition and visually represent how one can observe the benefits of TL in practice. We assume that we would like to solve a certain task, and that we can choose between two different models: a model that has never dealt with any kind of task before, and a second model, which is identical to the first one with the major difference being that it has already seen a similar task in the past. Because of their different nature, we refer to the first model as a model that will be trained from scratch, and to the second model as a pre-trained model. Ideally, as mentioned in the previous section, we would like the latter model to perform better on the considered task that the first one. Yet, how can we asses if one model is better than the other one? As initially presented by Langley [62] and later generalized by Lazaric [64] we would like the performance of a pre-trained model to result into three possible improvements. If at least one of these improvements is observed while training, we can then consider the pre-trained model to be better than the scratch model. These improvements are the following:

- Learning Speed Improvements: in this scenario the performance between a pre-trained model and a model trained from scratch is identical by the time training is finished, however when this kind of improvement appears we observe that the pre-trained model converges faster than the model trained from scratch. An example of this TL improvement is presented in the first plot of Fig. 4. The goal is to train a model such that by the end of training its performance reaches a value of 200. We can clearly see that both models manage to converge to this desired performance value, but that the pre-trained model manages to converge already after ≈ 50 training iterations, whereas the model trained from scratch requires more than 200 training iterations to perform similarly. Also note that the performance of both models at the beginning of training is identical, with both models starting with an initial performance of ≈ 20.

- Jumpstart Improvements: similarly to the previous case, also in this scenario there are no significant differences between the performance of a pre-trained model and the performance of a random model by the end of training. However this changes when we consider the very first training iteration. If jumpstart improvements appear, we can usually observe that when both models start their training process, the performance of the pre-trained model is much closer to the one that will be obtained by the end of training than the one of the scratch model. We visually report an example of this scenario in the third plot of Fig. 4. In this case the goal is to train a model such that by the end of training its performance will be of ≈ -100. We can clearly see that by the end of training both models are able to successfully achieve this goal, but that at the very first training iteration, the performance of the pre-trained model is significantly closer to the desired final performance (≈ -250) than the one of the scratch model (≈ -450).

- Asymptotic Improvements: when this TL improvement appears, the final performance of a pre-trained model is significantly higher than the one of a model trained from scratch. It is worth noting that similarly to what happens when learning speed improvements are observed, also in this case the performance of both models is identical when training begins, and that this

TL improvement only presents itself after several training iterations. A visualization of this TL improvement can be observed in the last plot of Fig. 4 where we can observe that for the first ≈ 20 training iterations there are no differences in terms of performance between a pre-trained model and a model trained from scratch. However the more training iterations are performed, the more the pre-trained model starts outperforming the model trained from scratch, reaching a final performance that is almost twice as good by the 100th training iteration.

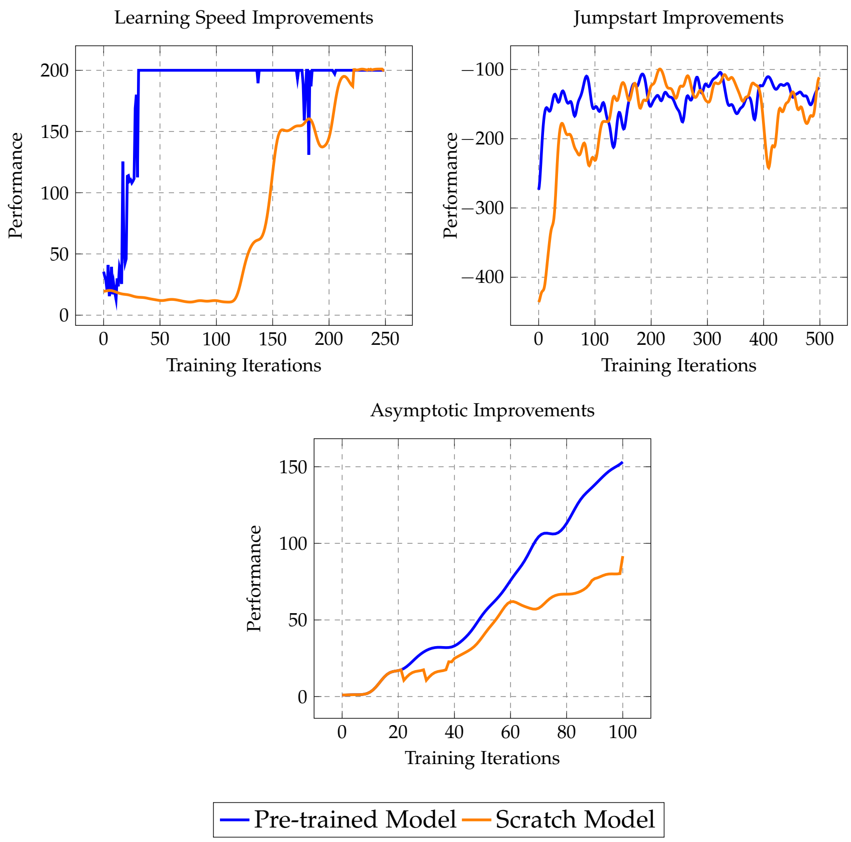


Figure 4: A visualization of the three possible desired outcomes that can come from adopting Transfer Learning strategies as initially defined by Langley [62] and later by Lazaric [64].

While all the improvements presented in Fig. 4 are all highly desirable, it is worth noting that some of them can be more preferable than others. In fact, the potential benefits of TL highly depend from the problem at hand. As an example, let us consider a training situation where the main goal is that of minimizing the overall training time of a model. In this particular case, jumpstart and learning improvements are more desirable than asymptotic improvements, since the latter improvement might not result into a model that converges to a desired solution faster. On the other hand, if the main objective is that of training a model which performs as best as possible, than evidently, asymptotic improvements are preferred. It is also worth noting that the examples presented in Fig. 4 are not mutually exclusive, and that the benefits of TL can present themselves as a combination of improvements, rather than in the form of a single, isolated, improvement.

Now that we have introduced the key ideas behind TL, and presented why adopting TL training strategies is beneficial we move to formally characterizing this machine learning paradigm both from a supervised learning perspective as from a reinforcement learning one.

3.3 MATHEMATICAL DEFINITIONS

As is common throughout this thesis we start by focusing on the supervised learning case.

3.3.1 *Supervised Learning*

The first two definitions that we provide are borrowed from Zhuang et al. [171] and are the ones of domain and task. The first one is defined as follows:

Definition 3 *A domain $\mathcal{D}$ is the combination between an input space $\mathcal{X}$ and a marginal distribution $P(X)$, $\mathcal{D} = \{\mathcal{X}, P(X)\}$ where X denotes an instance set defined as $X = \{\mathbf{x}|\mathbf{x_i} \in \mathcal{X}, i = 1, \cdots, n\}$.*

Examples of domains can be natural images, time-series data, biomedical markers and so on. In supervised learning we know that each domain is associated to its respective task, which is representative of the problem we would like to solve. A task is defined as follows:

Definition 4 *A task $\mathcal{T}$ consists of a label space $\mathcal{Y}$ and a decision function f, i.e., $\mathcal{T} = \{\mathcal{Y}, f\}$. The decision function f is implicit and can only be learned by sampling data from $\mathcal{X}$ which comes in the form of $\{x_i, y_i\}$ pairs where x_i and $y_i \in \mathcal{Y}$.*

Examples of possible decision functions f can be classifiers which categorize natural images in their respective classes, or regressors that are able to predict future values in a time-series. The key concept underlying TL is that, differently from the common supervised learning scenario where the only information that is available to a model are one domain and one task, we now have access to at least two domains. The one that corresponds to the task we would like to solve, called the target-domain $\mathcal{D}_T$, and an additional, possibly related, domain that comes with the name of the source-domain $\mathcal{D}_S$. With all these concepts in place we can now define Transfer Learning as:

Definition 5 *Given one, or more, observations corresponding to* $m^s \in \mathbb{N}^+$ *source domain(s) and tasks(s) (i.e.,* $\{(\mathcal{D}_{Si}, \mathcal{T}_{Si}|i = 1 \cdots, m^s)\}$*) and some additional observation(s) about* $m^T \in \mathbb{N}^+$ *target domain(s) and task(s) (i.e.* $\{(\mathcal{D}_{Tj}, \mathcal{T}_{Tj}|j = 1, \cdots, m^T)\}$*), transfer learning utilizes the knowledge implied in the source domains to improve the performance of the learned decision functions* $f_j^T (j = 1, \cdots, m^T)$ *on the target domain(s).*

As pointed out by Pan and Yang [92] the condition that the source and the target domains might be different $\mathcal{D}_S \neq \mathcal{D}_T$ implies that either their respective input spaces are different as well $\mathcal{X}_S \neq \mathcal{X}_T$, or that their corresponding marginal distributions are different $P_s(X) \neq P_T(X)$. Similarly, if the tasks are different instead $\mathcal{T}_S \neq \mathcal{T}_T$, then either one between the output spaces, or the conditional distributions has to be different ($\mathcal{Y}_S \neq \mathcal{Y}_T$ or $P(Y_S|X_S) \neq P(Y_T|X_T)$).

Based on the consistency between the source and target input spaces $\mathcal{X}$, and the respective output spaces $\mathcal{Y}$, one can categorize TL into three following settings.

INDUCTIVE TRANSFER LEARNING This TL scenario is characterized by the fact that the target task $\mathcal{T}_T$ is different from the source task $\mathcal{T}_S$, while the source domain $\mathcal{D}_S$ and the target domain $\mathcal{D}_T$ might, or might not, be similar. As originally presented in [92] we define inductive transfer learning as:

Definition 6 *Given a source domain* $\mathcal{D}_S$ *and a learning task* $\mathcal{T}_S$*, and a target domain* $\mathcal{D}_T$ *and a learning task* $\mathcal{T}_T$*, inductive transfer learning aims to help improving the target predictive function* $f_T(\cdot)$ *in* $\mathcal{D}_T$ *by using the knowledge in* $\mathcal{D}_S$ *and* $\mathcal{T}_S$*, where* $\mathcal{T}_S \neq \mathcal{T}_T$*.*

A key requirement of this type of TL is that some labeled data in the target domain is necessary for inducing the objective predictive function $f_T(\cdot)$. To build some more intuition about this kind of TL let us assume that we would like to train a model on our target task $\mathcal{T}_T$ which corresponds to recognizing what kind of Japanese letter is depicted in an image. Instead of training a model only on a dataset of letters, we instead start from a model that has already been trained to recognize digits, which will therefore be our source task $\mathcal{T}_S$. Note that in this case the source task and the target tasks are evidently different $\mathcal{T}_S \neq \mathcal{T}_T$ (classifying digits vs classifying letters), but that the input space of the source and target domains is the same $\mathcal{X}_S = \mathcal{X}_T$, since it corresponds to black and white images as represented in Fig. 5. Please note that in this example, since we use a model that is pretrained on images representing digits, we assume that we have had access to some labeled data in the source domain in the past. However the definition of inductive transfer learning does not require labeled data within the source domain to be strictly necessary.

TRANSDUCTIVE TRANSFER LEARNING Originally proposed by Arnold, Nallapati, and Cohen [5], this type of TL is characterized by the fact

Figure 5: A visualization of two datasets that can be used for performing inductive transfer learning. On the left we represent instances of the popular MNIST dataset [163], while on the right instances of the Kuzushiji-MNIST dataset [24]. We see that both datasets share the same domain $\mathcal{D}_S = \mathcal{D}_T$ (black and white images), but are associated to different tasks $\mathcal{T}_S \neq \mathcal{T}_T$ (classification of digits vs classification of Japanese letters).

that the source and target tasks are the same $\mathcal{T}_S = \mathcal{T}_T$, but their respective domains are different $\mathcal{D}_S \neq \mathcal{D}_T$. It is also possible that the feature spaces between domains are the same but, if that is the case, then the marginal probability distributions are different $P(X_S) \neq P(X_T)$. In its original formulation, Arnold, Nallapati, and Cohen [5] assumed that all unlabeled data in the target domain is available at training time, however we hereafter report the definition of Pan and Yang [92] who instead relax this condition and require only a subset of unlabeled target data to be seen at training time.

Definition 7 *Given a source domain $\mathcal{D}_S$ and a learning task $\mathcal{T}_S$, and a target domain $\mathcal{D}_T$ and a learning task $\mathcal{T}_T$, transductive transfer learning aims to help improving the target predictive function $f_T(\cdot)$ in $\mathcal{D}_T$ by using the knowledge in $\mathcal{D}_S$ and $\mathcal{T}_S$, where $\mathcal{D}_S \neq \mathcal{D}_T$ and $\mathcal{T}_S = \mathcal{T}_T$.*

As an example of transductive transfer learning let us assume that we would like to train a model to classify which type of clothing is depicted in an image. This time, differently from the inductive transfer learning case, the images in our target domain $\mathcal{D}_T$ are not black and white images anymore, but rather colored images, therefore they are defined over the RGB domain (see right image of Fig. 6). We now assume that we have access to a pre-trained model which has already been trained to classify the same type of clothes, with the main difference being that the images constituting the source domain $\mathcal{D}_S$ were black and white images (see left image of Fig. 6). If we now train this pre-trained model on our colored dataset we see that our setting fits the transductive transfer learning scenario: our considered domains are different $\mathcal{D}_S \neq \mathcal{D}_T$ (colored vs black and white images), but their

Figure 6: Two datasets that are representative of transductive transfer learning. On the left we show images coming from the Fashion-MNIST dataset [161], while on the right we report instances of the same dataset that are colored. In this case tasks among datasets are shared $\mathcal{T}_S = \mathcal{T}_T$ (classification of clothes), but the respective images come from different domains $\mathcal{D}_S \neq \mathcal{D}_T$ (black and white vs RGB images).

respective tasks are the same $\mathcal{T}_S \neq \mathcal{T}_T$, since a model is always trained to classify types of clothing.

Although this section focuses on supervised learning, we still report for the sake of completeness a definition of unsupervised transfer learning hereafter, and see how this kind of TL is linked to the types of TL that we have analyzed so far

UNSUPERVISED TRANSFER LEARNING Arguably considered to be the most challenging, and the least explored type of TL, unsupervised transfer learning is characterized by the total absence at training time of labeled data in both the source domain and the target domain. As mentioned by Pan and Yang [92] very little research work has so far explored this TL paradigm, with the only existing works exploring typical unsupervised learning topics such as clustering [25, 55, 101] and dimensionality reduction [147, 168, 169]. Unsupervised transfer learning is defined as follows:

Definition 8 *Given a source domain $\mathcal{D}_S$ and a learning task $\mathcal{T}_S$, and a target domain $\mathcal{D}_T$ and a learning task $\mathcal{T}_T$, unsupervised transfer learning aims to help improving the target predictive function $f_T(\cdot)$ in $\mathcal{D}_T$ by using the knowledge in $\mathcal{D}_S$ and $\mathcal{T}_S$, where $\mathcal{T}_S \neq \mathcal{T}_T$ and $\mathcal{Y}_S$ and $\mathcal{Y}_T$ are not observable.*

Based on this definition we can note that unsupervised transfer learning is more similar to inductive transfer learning than to to transductive transfer learning since we again assume that the source and the target tasks are different $\mathcal{T}_S \neq \mathcal{T}_T$.

3.3.2 *Reinforcement Learning*

When it comes to the reinforcement learning setup, the aforementioned definition of TL slightly changes, and becomes arguably less general. Recall from Chapter that in RL, the main goal is that of training an agent such that it becomes able to interact with its environment, a problem that is modeled with Markov Decision Process (MDP). It follows that in the RL context, the previously introduced concept of domain $\mathcal{D}$ (which could come in numerous flavors), now comes in the arguably more strictly defined form of a MDP $\mathcal{M}$. Just like domains, MDPs can either be representative of a source task, $\mathcal{M}_S$, or of a target $\mathcal{M}_T$ task, with the latter case corresponding to the main RL problem we are interested to solve. Also the previously introduced predictive function $f(\cdot)$ is now defined more precisely, since it corresponds to the task of learning an optimal policy π^* for $\mathcal{M}_T$. Based on these concepts we give the following definition of TL for reinforcement learning that is adapted from the one proposed by Zhu, Lin, and Zhou [170].

Definition 9 *Given a source MDP $\mathcal{M}_S$ and a target MDP $\mathcal{M}_D$, transfer learning in reinforcement learning aims to learn an optimal policy π^* for $\mathcal{M}_D$ by exploiting some prior knowledge related to $\mathcal{M}_S$, $\mathcal{K}_S$, together with the knowledge that underlies $\mathcal{M}_T$, $\mathcal{K}_T$ such that:*

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{s\sim\mu_0^t, a\sim\pi}\big[Q^{\pi}_{\mathcal{M}}(s,a)\big], \tag{49}$$

where $\pi = \zeta(\mathcal{K}_S \sim \mathcal{M}_S, \mathcal{K}_T \sim \mathcal{M}_T) : \mathcal{S}^t \rightarrow \mathcal{A}^t$ is a function mapping from the states to actions for $\mathcal{M}_T$ learned thanks to both $\mathcal{K}_S$ and $\mathcal{K}_T$.

Note that differently from the supervised learning case we are now making explicit use of the concept of knowledge $\mathcal{K}$, which is what we would like to retain when moving from a source MDP $\mathcal{M}_S$ to a target MDP $\mathcal{M}_T$. We do this because RL is a machine learning paradigm that is, arguably, more complex than the supervised learning one. A complexity which stems from the fact that in RL there are concepts such as e.g. rewards and policies, which are, by definition, not present in the supervised learning setup. As a result, $\mathcal{K}$ can come in forms that it cannot take in SL, and correctly identifying which kind of knowledge to transfer between $\mathcal{M}_S$ and $\mathcal{M}_T$ is just as important as developing a method that successfully transfers this knowledge in the first place. As mentioned by Lazaric [64] and by Tirinzoni, Sanchez, and Restelli [137], $\mathcal{K}$ can come in the following forms:

TRANSFER OF INSTANCES: in this scenario $\mathcal{K}$ corresponds to RL trajectories coming in the form $\langle s_t, a_t, r_t, s_{t+1}\rangle$ and that have been collected on one, or possibly multiple, source MDPs $\mathcal{M}_S$. Such trajectories can then be used both in a model-based RL setting, as done by

Taylor, Jong, and Stone [133], or for speeding up the process of learning a value function as described in [66] and [63]. Ideally, transferring RL trajectories should result into algorithms that are highly sample efficient, although it is worth noting that this property, albeit desirable, can constrain the source task $\mathcal{M}_S$ and the target task $\mathcal{M}_T$ to have similar transition models and reward functions. This instance of TL is usually used within the batch RL setup, where gathering experience samples for $\mathcal{M}_T$ can be particularly expensive or time consuming, a constraint that can be overcome by using trajectories coming from $\mathcal{M}_S$ instead. The typical challenge in such cases then consists in correctly identifying which samples coming from $\mathcal{M}_S$ are the most informative ones for solving $\mathcal{M}_T$ [138].

TRANSFER OF REPRESENTATIONS:

TRANSFER OF PARAMETERS:

3.4 DEEP TRANSFER LEARNING

3.4.1 *General Framework*

- Off the Shelf Classification
- Fine-Tuning

3.4.2 *Literature Review*

SUPERVISED LEARNING not limited to vision —> mention NLP, BERT, ...

REINFORCEMENT LEARNING

3.5 RELEVANCE FOR THIS DISSERTATION

Part II

TRANSFER LEARNING FOR SUPERVISED LEARNING

4 ON THE TRANSFERABILITY OF CONVOLUTIONAL NETWORKS FOR CLASSIFICATION

Contributions and Outline

This chapter contributes to the field of (Deep) Transfer Learning (TL) by investigating whether popular neural networks that come as pre-trained on datasets containing natural images can perform equally well once they are used on non-natural datasets. Specifically, we explore whether these models can be used for tackling three different art classification problems. Furthermore, assuming this is the case, we also explore whether it is possible to improve on such performance. The chapter is structured as follows: we start by providing the reader with some background information in Sec. 4.1. In Sec. 4.2 we present a brief theoretical reminder of the field of TL, a description of the datasets that we have used and the methodological details about the experiments that we have performed. In Sec. 6.4 we present and discuss our results. A summary of the main contributions of this work ends the chapter in Sec. 6.6.

This chapter is based on the publication Sabatelli et al. [110].

4.1 INTRODUCTION AND RELATED WORK

Over the past decade Deep Convolutional Neural Networks (DCNNs) have become one of the most used and successful algorithms in Computer Vision (CV) [27, 77, 139]. Due to their ability to automatically learn representative features by incrementally down sampling the input via a set of non linear transformations, these kind of neural networks have rapidly established themselves as the state of the art algorithm on a large set of CV problems. Within different CV testbeds large attention has been paid to the ImageNet challenge [26], a CV benchmark that aims to test the performance of different image classifiers on a dataset that contains one million natural images distributed over thousand different classes. The availability of such a large dataset, combined with the possibility of training deep neural networks in parallel over several `GPUs` [60], has lead to the development of a large set of different neural architectures that have continued to outperform each other over the years [22, 47, 52, 118, 131]. A promising research field in which the classification performances of such DCNNs can be exploited is that of *Digital Heritage* [95]. Due to a growing and rapid process of digitization, museums have started to digitize large parts of their cultural heritage collections, leading to the creation of several digital open datasets [4, 82]. The images constituting these datasets are mostly matched with descriptive metadata which, as presented by Mensink and Van Gemert [82], can be used to define a set of challenging machine learning tasks. However, the number of samples in these datasets is far smaller than those in, for instance, the ImageNet challenge and this can become a serious constraint when trying to successfully train deep networks from scratch.

The lack of available training data is a well known issue in the deep learning community and is one of the main reasons that has led to the development of the research field of Transfer Learning (TL). The main idea of TL consists of training a machine learning algorithm on a new task (e.g. a classification problem) while exploiting knowledge that the algorithm has already learned on a previously related task (a different classification problem). This machine learning paradigm has proved to be extremely successful in deep learning, where it has been shown how popular models that were trained on many large datasets [53, 121], were able to achieve very promising results on classification problems from heterogeneous domains, ranging from medical imaging [132] or gender recognition [159] over plant classification [104] to galaxy detection [3].

In this work we explore whether the TL paradigm can be successfully applied to three different art classification problems. We use four neural architectures that have obtained strong results on the ImageNet challenge in recent years and we investigate their performance when it comes to attributing the *authorship* to different artworks, recognizing the *material* which has been used by the artists in their creations, and identifying the *artistic category* these artworks fall into. We do so by comparing two possible approaches that can be used to tackle the different classification tasks. The first one, known as off the shelf classification [103], simply retrieves the features that were learned by the networks on other datasets and uses them as input for a new classifier. In this scenario the weights of the model do not change during the training phase, and the final, top-layer classifier is the only component of the architecture which is actually trained. This changes in our second explored approach, known as fine tuning, where the weights of the original network are "unfrozen" and the neural architectures are trained together with the final classifier.

Kornblith, Shlens, and Le [59] have shown the benefits that this particular pre-training approach has. In particular, DCNNs which have been trained on the ImageNet challenge typically lead to superior results when compared to the same architectures trained from scratch. However, this is not necessarily beneficial and in some cases networks that are randomly initialized are able to achieve the same performance as ImageNet pre-trained models. However, none of the results presented in [59] report experiments on datasets containing heritage objects, it is thus still an open question how such pre-trained DCNNs would perform in such a classification scenario. In the rest of this chapter we report results that extensively study the performance of such neural networks; at the same time we also assess whether better TL performance can be obtained when using neural networks that, in addition to the ImageNet dataset, have additionally been pre-trained on a large artistic collection.

4.2 METHODS

We now present the methods that underpin our research. We start by giving a brief formal reminder of TL. We then introduce the three classification tasks under scrutiny, together with a brief description of the datasets. Finally, we present the neural architectures that we have used for our experiments.

4.2.1 *Transfer Learning*

A supervised learning (SL) problem can be identified by three elements: an input space $\mathcal{X}_t$, an output space $\mathcal{Y}_t$, and a probability distribution $p_t(x, y)$ defined over $\mathcal{X}_t \times \mathcal{Y}_t$ (where t stands for 'target', as this is the main problem we would like to solve). The goal of SL is then to build a function $f : \mathcal{X}_t \rightarrow \mathcal{Y}_t$ that minimizes the expectation over $p_t(x, y)$ of a given loss function ℓ assessing the predictions made by f:

$$E_{(x,y)\sim p_t(x,y)}\{\ell(y, f(x))\}, \tag{50}$$

when the only information available to build this function is a learning sample of input-output pairs $LS_t = \{(x_i, y_i)|i = 1, \ldots, N_t\}$ drawn independently from $p_t(x, y)$. In the general transfer learning setting, one assumes that an additional dataset LS_s, called the source data, is available that corresponds to a different, but related, SL problem. More formally, the source SL problem is assumed to be defined through a triplet $(\mathcal{X}_s, \mathcal{Y}_s, p_s(x, y))$, where at least either $\mathcal{X}_s \neq \mathcal{X}_t$, $\mathcal{Y}_s \neq \mathcal{Y}_t$, or $p_s \neq p_t$. The goal of TL is then to exploit the source data LS_s together with the target data LS_t to potentially find a better model f in terms of the expected loss (50) than when only LS_t is used for training this model. Transfer learning is especially useful when there is a lot of source data, whereas target data is more scarce.

Depending on the availability of labels in the target and source data and on how the source and target problems differ, one can distinguish different TL settings [93]. In what follows, we assume that labels are available in both the source and target data and that the input spaces $\mathcal{X}_t$ and $\mathcal{X}_s$, that both correspond to color images, match. Output spaces and joint distributions will however differ between the source and target problems, as they will typically correspond to different classification problems (ImageNet object recognition versus art classification tasks). Our problem is thus an instance of *inductive transfer learning* [93]. While several inductive transfer learning algorithms exist, hereafter we focus on model transfer techniques, where information between the source and target problems is exchanged in the form of a neural network that comes as pre-trained on the source data. Although potentially suboptimal, this approach has the advantage of being more computationally efficient, as it does not require to train a model using both the source and the target data.

4.2.2 *Datasets and Classification Challenges*

For our experiments we use two datasets which come from two different heritage collections. The first one contains the largest number of samples and comes from the Rijksmuseum in Amsterdam[1]. On the other hand, our second 'Antwerp' dataset is much smaller. This dataset presents a random sample that is available as open data from a larger heritage repository: DAMS (Digital Asset Management System)[2]. This repository can be searched manually via the web-interface or queried via a Linked Open Data API. It aggregates the digital collections of the foremost GLAM institutions (Galleries, Libraries, Archives, Museums) in the city of Antwerp in Belgium. Thus, this dataset presents a varied and representative sample of the sort of heritage data that is nowadays being collected at the level of individual cities across the globe. While it is much smaller, its coverage of cultural production is similar to that of the Rijksmuseum dataset and presents an ideal testing ground for the transfer learning task under scrutiny here.

Both image datasets come with metadata encoded in the Dublin Core metadata standard [152]. We selected three well-understood classification challenges: (1) "material classification" which consists in identifying the material the different heritage objects are made of (e.g paper, gold, porcelain, ...) ; (2) "type classification" in which the neural networks have to classify in which artistic category the samples fall into (e.g. print, sculpture, drawing, ...), and finally (3) "artist classification", where the main goal is to appropriately match each sample of the dataset with its creator (from now on we refer to these classification tasks as challenge 1, 2 and 3 respectively). As reported in Table 1 we can see that the Rijksmuseum collection is the dataset with the largest amount of samples per challenge (N_t) and the highest amount of labels to classify (Q_t). Furthermore it is also worth noting that there was no metadata available when it comes to the first classification challenge for the Antwerp dataset (as marked by the $\times$ symbol),

1 https://staff.fnwi.uva.nl/t.e.j.mensink/uva12/rijks/
2 https://dams.antwerpen.be/

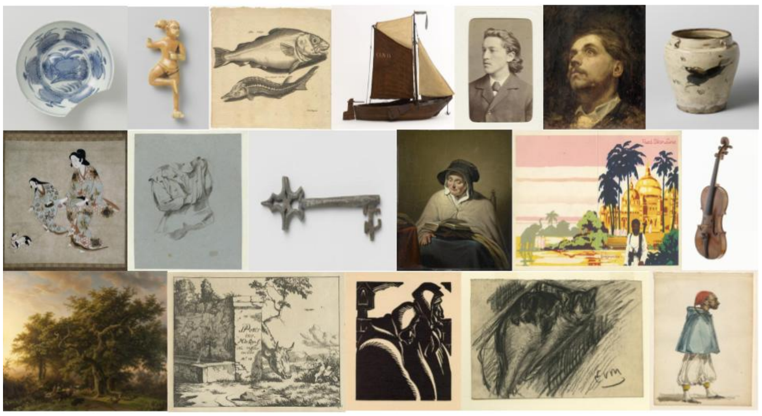


Figure 7: A visualization of the images that are used for our experiments. It is possible to see how the samples range from images representing plates made of porcelain to violins, and from Japanese artworks to a more simple picture of a key.

and that there are some common labels between the two heritage collections when it comes to challenge 2. A visualization reporting some of the images that are present in both datasets is shown in Figure 7.

Table 1: An overview of the two datasets that are used in our experiments. Each color of the table corresponds to a different classification challenge, starting from challenge 1 which is represented in yellow, challenge 2 in blue and finally challenge 3 in red. Furthermore we represent with N_t the amount of samples constituting the datasets and with Q_t the number of labels. Lastly, we also report if there are common labels between the two heritage collections.

Challenge	Dataset	N_t	Q_t	% of overlap
Material	Rijksmuseum	110,668	206	None
	Antwerp	×	×	
Type	Rijksmuseum	112,012	1,054	
	Antwerp	23,797	920	≈ 15%
Artist	Rijksmuseum	82,018	1,196	None
	Antwerp	18,656	903	

We use 80% of the datasets for training while the remaining $2 \times 10\%$ is used for validation and testing respectively. Furthermore, we ensure that only classes which occur at least once in all the splits are used for our experiments. Naturally, in order to keep all comparisons fair between neural architectures and different TL approaches, all experiments have been performed on the exact same data splits which, together with the code used for all our experiments, are publicly released to the CV community [3].

3 https://github.com/paintception/Deep-Transfer-Learning-for-Art-Classification-Problems

4.2.3 *Convolutional Networks and Classification Approaches*

For our experiments we use four pre-trained DCNNs that have all obtained state of the art results on the ImageNet classification challenge. The neural architectures are VGG19 [118], Inception-V3 [131], Xception [22] and ResNet50 [162]. We use the implementations of the networks that are provided by the `Keras` Deep Learning library [23] together with their appropriate `Tensorflow` weights [1] that come from the `Keras` official repository as well. Since all architectures have been built in order to deal with the ImageNet dataset we replace the final classification layer of each network with a new one. This final layer simply consists of a new *softmax* output, with as many neurons as there are classes to classify, which follows a 2D global average pooling operation. We rely on this dimensionality reduction step because we do not add any fully connected layers between the last convolution layer and the *softmax* output. Hence, in this way we are able to obtain a feature vector, X, out of the rectified activation feature maps of the network that can be properly classified. Since all experiments are treated as a multi-class classification problem all networks minimize the *categorical crossentropy* function loss function.

We investigate two possible classification approaches that are based on the previously mentioned pre-trained architectures. The first one, denoted as off the shelf classification, only trains a final *softmax* classifier on X, which is retrieved from the different models after performing one forward pass of the image through the network [4]. This approach is intended to explore whether the features that are learned by the network on the ImageNet dataset are informative enough in order to properly train a machine learning classifier on the previously introduced art classification challenges. If this would be the case, such pre-trained models could be used as appropriate feature extractors without having to rely on expensive `GPU` computations for training. Naturally, they would only require the training of the final classifier without having to compute any backpropagation operations over the entire network. From now on we will refer to these networks with θ^{-}.

Our second approach is generally known as fine tuning and differs from the previous one by the fact that together with the final *softmax* output the entire network is trained as well. This means that unlike the off the shelf approach, the entirety of the neural architecture gets "unfrozen" and is optimized during training. The potential benefit of this approach lies in the fact that the models get independently trained on samples coming from the artistic datasets, and therefore their classification predictions will not be restricted to what the networks previously learned on the ImageNet dataset only. Evidently,

---

4 Please note that instead of a *softmax* layer any kind of machine learning classifier can be used instead. We experimented with both Support Vector Machines (SVMs) and Random Forests but since the results did not significantly differ between classifiers we decided to not include them here.

such an approach is computationally more demanding. We refer to these networks as θ^i, where i stands for the source task these models have been trained on, namely the ImageNet dataset. We visually present a simplified representation of the classification approaches considered in this chapter together with how they differ from not pre-trained model in Fig. 8.

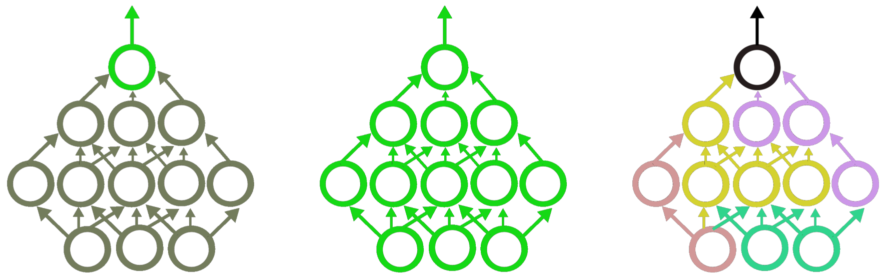


Figure 8: A simplified visual representation of the different training paradigms that are considered in this chapter. The first plot represents a model that comes as pre-trained on a source task but which is not allowed to update most of its weights during training (represented in gray): the only trainable parameters of this network are the ones that parametrize the final layer of the network and that are represented in green. In the second plot we visually represent a model that is parametrized with weights that have been learned on a certain source task, and that when training the model on the target task get "unfrozen", therefore making the model fully trainable. Lastly we visualize a model that does not come as pre-trained on any source task at all but that is initialized with random weights instead (represented by the various colours).

In order to maximize the performance of all models we follow some of the recommendations presented by Masters and Luschi [79] and train the networks with a relatively small batch size of 32 samples. We do not perform any data augmentation operations besides a standard pixel normalization to the $[0, 1]$ range and a re-scaling operation which resizes the images to the input size that is required by the different models. Regarding the stochastic optimization procedures of the different classifiers, we use two different optimizers, that after preliminary experiments, turned out to be the best performing ones. For the off the shelf approach we use the RMSprop optimizer [136] which has been initialized with its default hyperparameters (learning rate = 0.001, a *momentum* value $\rho = 0.9$ and $\epsilon = 1e - 08$). On the other hand, when we fine tune the DCNNs we use the standard (and less greedy) Stochastic Gradient Descent (SGD) algorithm with the same learning rate, 0.001, and a *Nesterov Momentum* value set to 0.9. Training has been controlled by the *Early Stopping* method [21] which interrupted training as soon as the validation loss did not decrease for

7 epochs in a row. The model which is then used on the testing set is the one which obtained the smallest validation loss while training.

To the best of our knowledge, the results presented in this chapter are the first ones that systematically asses to which extent models pre-trained on the ImageNet dataset can be used as valuable architectures when tackling art classification problems. Furthermore, we also present the first results that investigate it is also not known whether the fine tuning approach can yield better results than the off the shelf one, and if using such pre-trained models would yield better performance than training the same architectures from scratch as observed by Kornblith, Shlens, and Le [59].

4.3 RESULTS

Our experimental results are divided in two different sections, depending on which kind of dataset has been used. We first report the results that we have obtained when using architectures that were pre-trained on the ImageNet dataset only, and aimed to tackle the three classification problems of the Rijksmuseum dataset that were presented in Section 4.2.2. We report these results in Section 4.3.1 where we explore the benefits of using the ImageNet dataset as the TL source data, and how well such pre-trained models generalize when it comes to the artistic domain. We then present the results from classifying the Antwerp dataset, using models that are both pre-trained on the ImageNet dataset and on the Rijksmuseum collection in Section 4.3.3. We investigate whether these neural architectures, which have already been trained to tackle art classification problems before, perform better than the ones which have been trained on the ImageNet dataset only.

All results show comparisons between the off the shelf classification approach and the fine tuning scenario. In addition to that, in order to establish the potential benefits that TL from ImageNet has over training a model from scratch, we also report the results that have been obtained when training a network with weights that have been initially sampled from a "He-Uniform" distribution [46]. Since we take advantage of the work presented by Bidoia et al. [15] we use the Inception-V3 architecture. We refer to it in all figures as Scratch-V3 and always visualize it with a solid orange line. Figures 9 and 10 report the performance in terms of accuracy that the models have obtained on the validation sets. While the performances that the neural architectures have obtained on the final testing set are reported in Tables 2 and 3.

4.3.1 *From Natural to Art Images*

The first results that we report have been obtained on the "material" classification challenge. We believe that this can be considered as the easiest classification task within the ones that we have introduced in Section 4.2.2 for two main reasons. First, the number of possible classes the networks have to deal with is more than five times smaller when compared to the other two challenges. Furthermore, we also believe that this classification task is, within the limits, the most similar one when compared to the original ImageNet challenge. Hence, the features that might be useful in order to classify the different natural images on the latter classification testbed might be not too dissimilar from the ones that are needed to properly recognize the material that the different samples of the Rijksmuseum collection are made of. If this would be the case we would expect a very similar performance between the off the shelf classification approach and the fine tuning one. Comparing the learning curves of the two classification strategies in Figure 9, we actually observe that the fine tuning approach leads to significant improvements when compared to the off the shelf one, for three architectures out of the four tested ones. Note however that, in support of our hypothesis, the off the shelf approach can still reach high accuracy values on this problem and is also competitive with the DCNN trained from scratch. This suggests that features extracted from networks pretrained on ImageNet are relevant for material classification.

When comparing the learning curves of the two classification strategies reported in Figure 9, we can observe that the fine tuning approach leads to significant improvements when compared to the off the shelf one, for three architectures out of the four tested ones. Note however that, in support of our hypothesis, the off the shelf approach can still reach high accuracy values on this problem and is also competitive with the network that is trained from scratch. This suggests that features extracted from pretrained ImageNet models are relevant for material classification.

We can also observe that the ResNet50 architecture is the architecture which, when fine tuned, performs overall best when compared to the other three models. This happens despite it being the network that initially performed worse as a simple feature extractor in the off the shelf experiments. As reported in Table 2 we can see that this kind of behavior reflects itself on the separated testing set as well, where it obtained the highest testing set accuracy when fine tuned (92.95%), and the lowest one when the off the shelf approach was used (86.81%). It is worth noting that the performance between the different neural architectures do not strongly differ between each other once they are fine tuned, with all models performing around $\approx$ 92% on the final testing set. Furthermore, special attention needs to be given to the

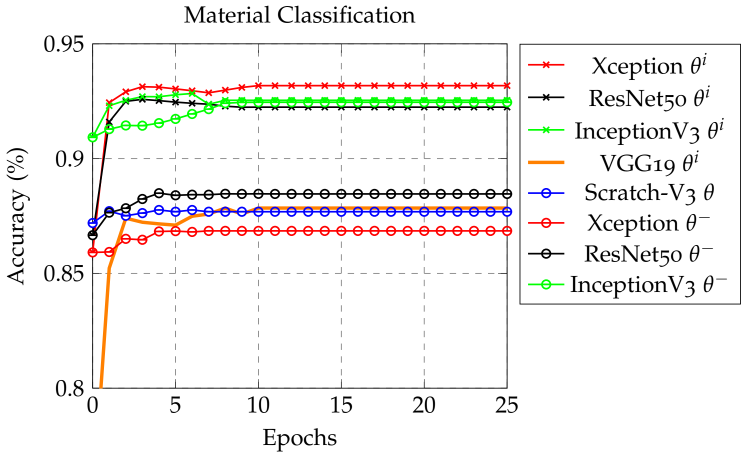


Figure 9: Comparison between the fine tuning approach (θ^i) versus the off the shelf one (θ^-) when classifying the material of the heritage objects of the Rijksmuseum dataset. We can observe that for three out of four neural architectures the first approach leads to significant improvements when compared to the latter one. Furthermore, we can also observe that training a randomly initialized model from scratch (solid orange line) leads to worse results than fine-tuning a network that comes as pre-trained on the ImageNet dataset.

VGG19 architecture, which does not seem to benefit from the fine tuning approach as much as the other architectures do. In fact, its off the shelf performance on the testing set (92.12%) is very similar to its fine tuned one (92.23%). This suggests that this neural architecture is the only one which, in this task, and when pre-trained on ImageNet, can successfully be used as a simple feature extractor without having to rely on complete retraining.

When analyzing the performance of the different neural architectures on the "type" and "artist" classification challenges (respectively the left and right plots reported in Figure 10), we observe that on these problems fine tuning strategy leads to even more significant improvements when compared to what we observed in the previous experiment. The results obtained on the second challenge show again that the ResNet50 architecture is the architecture which leads to the worse results if the off the shelf approach is used (its testing set accuracy is as low as 71.23%) and similarly to what has been observed before, it then becomes the best performing model when fine tuned, with a final accuracy of 91.30%. Differently from what has been observed in the previous experiment, the VGG19 architecture, despite being the network performing best when used as off the shelf feature extractor, this time performs significantly worse than when it is fine tuned, which highlights the benefits of this latter training approach. Similarly to what has been observed before, our results are again not significantly in favor of any fine tuned neural architecture, with all final accuracies being around $\approx 91\%$.

If the classification challenges that we have analyzed so far have highlighted the significant benefits of the fine tuning approach over the off the shelf one, it is also important to note that the latter approach is still able to lead to satisfying results. In fact, a final accuracy of 92.12% has been obtained when using the VGG19 architecture on the first challenge and a classification rate of 77.33% was reached by the same architecture on the second challenge. Despite the latter accuracy being very far in terms of performance from the one obtained when fine tuning the network (90.27%), these results still show that models pre-trained on ImageNet do learn particular features that can also be used for classifying the "material" and the "type" of heritage objects. However, when analyzing the results from the "artist" challenge, we can see that this is partially not the case anymore.

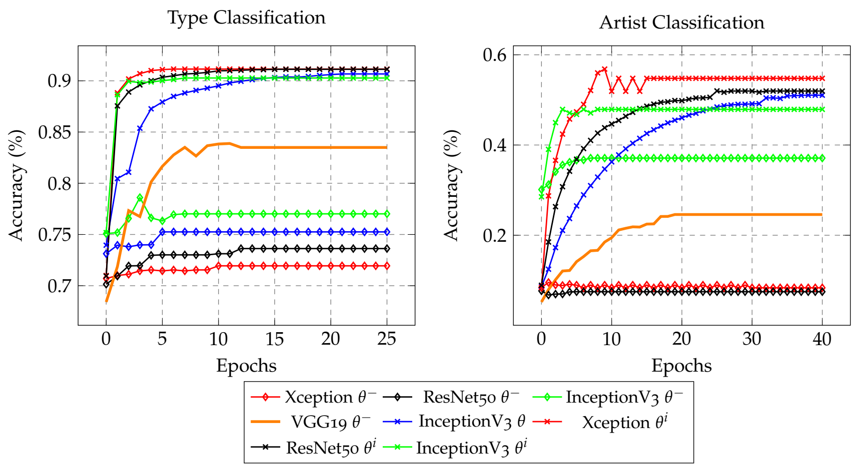


Figure 10: A similar analysis as the one which has been reported in Figure 9 but for the second and third classification challenges (left and right figures respectively). The results show again the significant benefits that fine tuning (reported by the dashed line plots) has when compared to the off the shelf approach (reported by the dash-dotted lines) and how this latter strategy miserably underperforms when it comes to artist classification. Furthermore we again see the benefits that using a pre-trained DCNN has over training the architecture from scratch (solid orange line).

When considering the third classification challenge we can observe that the Xception, ResNet50, and Inception-V3 architectures all perform extremely poorly if not fine tuned, with the latter two models not being able to even reach a 10% classification rate. Better results are obtained when using the VGG19 architecture, which reaches a final accuracy of 38.11%. Most importantly, the performance of each model are again significantly improved when the networks are fine tuned. As already observed in the previous experiments, ResNet50 outperforms the other architectures on the validation set. However, on the test set (see Table 2), the overall best performing network is Inception-V3 (with a final accuracy of 51.73%), which suggests that

ResNet50 suffered from overfitting. It is important to state two major important points about this set of experiments. The first one relates to the final classification accuracy that is obtained by all models, and that at first sight might seem disappointing. While it is true that these classification rates are significantly lower when compared to the ones obtained in the previous two experiments, it is important to highlight how a large set of artists present in the dataset are associated to an extremely limited amount of samples. This reflects a lack of appropriate training data which does not allow the models to learn all the features that are necessary for successfully dealing with this particular classification challenge. In order to do so, we believe that more training data is required. Moreover, it is worth pointing out that despite performing very poorly when used as off the shelf feature extractors, ImageNet pre-trained models do still perform better once they are fine tuned than a model that is trained from scratch. This suggests that these networks do learn potentially representative features when it comes the classification of artists, but in order to properly classify them, the model need to be fine tuned.

Table 2: An overview of the results obtained by the different models on the testing set when classifying the heritage objects of the Rijksmuseum. The overall best performing architecture is reported in a green cell, while the second best performing one is reported in a yellow one. The additional columns "Params" and "X" report the amount of parameters the networks have to learn and the size of the feature vector that is used as input for the softmax classifier.

Challenge	**DCNN**	**off the shelf**	**fine tuning**	**Params**	X
1	Xception	87.69%	92.13%	21K	2048
1	InceptionV3	88.24%	92.10%	22K	2048
1	ResNet50	86.81%	92.95%	24K	2048
1	VGG19	92.12%	92.23%	20K	512
2	Xception	74.80%	90.67%	23K	2048
2	InceptionV3	72.96%	91.03%	24K	2048
2	ResNet50	71.23%	91.30%	25K	2048
2	VGG19	77.33%	90.27%	20K	512
3	Xception	10.92%	51.43%	23K	2048
3	InceptionV3	.07%	51.73%	24K	2048
3	ResNet50	.08%	46.13%	26K	2048
3	VGG19	38.11%	44.98%	20K	512

4.3.2 *Discussion*

In the previous section, we have investigated whether four different architectures pre-trained on the ImageNet dataset can be successfully used to address three art classification problems. We have observed that this is particularly the case when it comes to classifying the material and the type, where in fact, the off the shelf approach already yielded satisfactory results. However, most importantly, we have also shown that the performance of all models can be significantly improved when the networks are fine tuned, and that an ImageNet initialization is beneficial when compared to training a randomly initialized network from scratch. Furthermore, we have discovered that the pre-trained DCNNs fail if used as simple feature extractors when having they are used for attributing the authorship to the different heritage objects. In the next section, we explore the performance of fine tuned models that are trained for tackling two of the already seen classification challenges on a different heritage collection. For this problem, we will again compare the off the shelf approach with the fine tuning one.

4.3.3 *From One Art Collection to Another*

Table 3 compares the results that we obtained on the Antwerp dataset when using ImageNet pre-trained models (θ^i) versus the same architectures that were fine tuned on the Rijksmuseum dataset (θ^r). While looking at the performance of the different neural architectures two interesting results can be highlighted. First, models which have been fine tuned on the Rijksmuseum dataset outperform the ones pre-trained on ImageNet in both classification challenges. This happens to be the case both when the networks are used as simple feature extractors and when they are fine tuned. On the "type" classification challenge, this result is not surprising since, as discussed in Section 4.2.2, the types corresponding to the heritage objects of the two collections partially overlap. This is more surprising on the "artist" classification challenge however, since there is no overlap at all between the artists of the Rijksmuseum and the ones from the Antwerp dataset. A second interesting result, which is consistent with the results presented in the previous section, revolves around the observation that it is always beneficial to fine tune the networks over just using them as off the shelf feature extractors. Once the models get fine tuned on the Antwerp dataset, these DCNNs, which have also been fine tuned on the Rijksmuseum dataset, outperform the architectures that were pre-trained on ImageNet only. This happened to be the case for both classification challenges and for all considered architectures, as reported in Table 3. This demonstrates how beneficial it is for the models to have been trained on a similar source task and how this can lead to significant improvements both when the networks are used as feature extractors and when they are fine tuned.

Table 3: The results obtained on the classification experiments performed on the Antwerp dataset with models that have been initially pre-trained on ImageNet (θ^i) and the same architectures which have been fine tuned on the Rijksmuseum dataset (θ^r). Our results show that the latter pre-trained networks yield better results both if used as off the shelf feature extractors and if fine tuned.

Challenge	DCNN	θ^i + off the shelf	θ^r + off the shelf	θ^i + fine tuning	θ^r + fine tuning
2	Xception	42.01%	62.92%	69.74%	72.03%
2	InceptionV3	43.90%	57.65%	70.58%	71.88%
2	ResNet50	41.59%	64.95%	76.50%	78.15%
2	VGG19	38.36%	60.10%	70.37%	71.21%
3	Xception	48.52%	54.81%	58.15%	58.47%
3	InceptionV3	21.29%	53.41%	56.68%	57.84%
3	ResNet50	22.39%	31.38%	62.57%	69.01%
3	VGG19	49.90%	53.52%	54.90%	60.01%

4.3.4 *Selective Attention*

The benefits of the fine tuning approach over the off the shelf one are clear from our previous experiments. Nevertheless, we do not have any insights yet as to what exactly allows fine tuned models to outperform the architectures which are pre-trained on ImageNet only. In order to provide an answer to that, we investigate which pixels of each input image contribute the most to the final classification predictions of the networks. We do this by using the "VisualBackProp" algorithm presented by [16], which is able to identify which feature maps of the networks are the most informative ones with respect to their final prediction. Once these feature maps are identified, they get backpropagated to the original input image, and visualized as a saliency map according to their weights. The higher the activation of the filters, the brighter the set of pixels covered by these filters are represented.

The results that we have obtained provide interesting insights about how fine tuned models develop novel selective attention mechanisms over the images, which are very different from the ones that characterize the networks that are pre-trained on ImageNet. We report the existence of these mechanisms in Figure 11 where we visualize the different saliency maps between a model pre-trained on ImageNet and the same neural architecture which has been fine tuned on the Rijksmuseum collection. In Figure 11 we visualize which sets of pixels allow the fine tuned model to successfully classify an artist of the Rijksmuseum collection that the same architecture was not able to initially recognize. It is possible to notice that the saliency maps of the latter architecture either correspond to what is more similar to a natural image, as represented by the central image of the first row of plots, or even to what appear to be non informative pixels at all, as shown by the second image in the second row. However, when considering the fine tuned model we clearly observe that these saliency maps change. In this case the network attends towards the set of pixels that represent people in the bottom, suggesting that this is what allows the model to appropriately recognize the artist of the considered artwork.

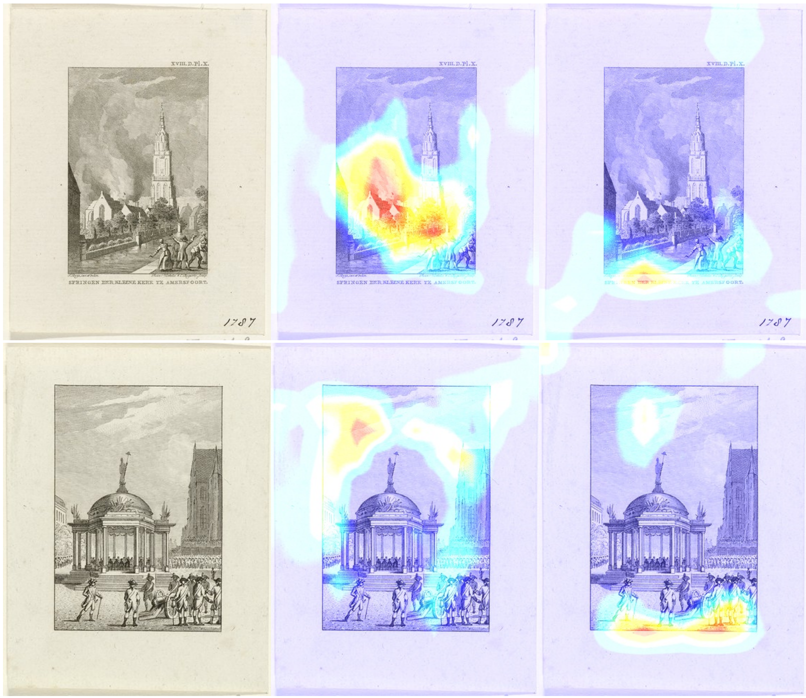


Figure 11: Add caption

These observations can be related to parallel insights in authorship attribution research [29], an established task from Natural Language Processing that is highly similar in nature to artist recognition. In this field, preference is typically given to high-frequency function words (articles, prepositions, particles etc.) over content words (nouns, adjectives, verbs, etc.), because the former are generally considered to be less strongly related to the specific content or topic of a work. As such, function words or stop words lend themselves more easily to attribution across different topics and genres. In art history, strikingly similar views have been expressed by the well-known scholar Giovanni Morelli (1816-1891), who published seminal studies in the field of artist recognition [160]. In Morelli's view too, the attribution of a painting could not happen on the basis of the specific content or composition of a painting, because these items were too strongly influenced by the topic of a painting or the wishes of a patron. Instead, Morelli proposed to base attributions to so-called *Grundformen* or small, seemingly insignificant details that occur frequently in all paintings and typically show clear traces of an artist's individual style, such as ears, hands or feat, a painting's function words, so to speak. The saliency maps above reveal a similar shift in attention when the ImageNet weights are adapted on the Rijksmuseum data: instead of focusing on higher-level content features, the network shifts its attention to lower layers in the network, seemingly focusing on insignificant details, that nevertheless appear crucial to perform artist attribution.

4.4 CONCLUSION

This paper provides insights about the potential that the field of TL has for art classification. We have investigated the behavior of DCNNs which have been originally pre-trained on a very different classification task and shown how their performances can be improved when these networks are fine tuned. Moreover, we have observed how such neural architectures perform better than if they are trained from scratch and develop new saliency maps that can provide insights about what makes these DCNNs outperform the ones that are pre-trained on the ImageNet dataset. Such saliency maps reflect themselves in the development of new features, which can then be successfully used by the DCNNs when classifying heritage objects that come from different heritage collections. It turns out that the fine tuned models are a better alternative to the same kind of architectures which are pre-trained on ImageNet only, and can serve the CV community which will deal with similar machine learning problems.

As future work, we aim to investigate whether the results that we have obtained on the Antwerp dataset will also apply to a larger set of smaller heritage collections. Furthermore, we want to explore the performances of densely connected layers [52] and understand which layers of the currently analyzed networks contribute the most to their final classification performances. This might allow us to combine the best parts of each neural architecture into one single novel DCNN which will be able to tackle all three classification tasks at the same time.

5

A NOVEL DATASET FOR TESTING TRANSFER LEARNING

Outline

5.1 INTRODUCTION

6 ON THE TRANSFERABILITY OF LOTTERY WINNERS

Contributions and Outline

In Chapters 4 and we have always performed Transfer Learning (TL) with extremely large and deep convolutional neural networks. In this chapter however, we take a different approach. We use TL as a tool for, not only exploiting the performance of deep neural networks, but also for characterizing a relatively new deep learning phenomenon that comes with the name of the "Lottery Ticket Hypothesis" (LTH). Specifically we investigate whether lottery winners that are found on datasets of natural images contain inductive biases that are generic enough to allow them to generalize to non-natural image distributions. To do so, we present to the first results that study the transferability of winning initializations in this particular training setting. Furthermore we also show that the LTH offers a novel way for doing TL when the training data is scarce.

The rest of this chapter is structured as follows: Sec. 6.1 introduces the LTH and presents the reasons that have motivated studying this phenomenon from a TL perspective. Sec. 6.2 and Sec. 6.3 present the experimental setup that was used throughout this chapter, while Sec. 6.4 and Sec. 6.4.3 present the main findings of our research. The chapter ends by contextualizing its content with respect to the existing literature in Sec. 6.5 and by identifying some potential avenues for future work in Sec. 6.6.

This chapter is based on the publication Sabatelli, Kestemont, and Geurts [108].

6.1 INTRODUCTION

The "Lottery-Ticket-Hypothesis" (LTH) [35] states that within large randomly initialized neural networks there exist smaller sub-networks which, if trained from their initial weights, can perform just as well as the fully trained unpruned network from which they are extracted. This happens to be possible because the weights of these sub-networks seem to be particularly well initialized before training starts, therefore making these smaller architectures suitable for learning (see Fig 12 for an illustration). These sub-networks, i.e., the pruned structure together with their initial weights, are called winning tickets, as they appear to have won the initialization lottery. Since winning tickets only contain a very limited amount of parameters, they yield faster training, inference, and sometimes even better final performance than their larger over-parametrized counterparts [35, 37]. So far, winning tickets are typically identified by an iterative procedure that cycles through several steps of network training and weight pruning, starting from a randomly initialized unpruned network. While simple and intuitive, the resulting algorithm, has unfortunately a high computational cost. Despite the fact that the resulting sparse networks can be trained efficiently and in isolation from their initial weights, the LTH idea has not yet led to more efficient solutions for training a sparse network, than existing pruning algorithms that all also require to first fully train an unpruned network [28, 43, 72, 86, 172].

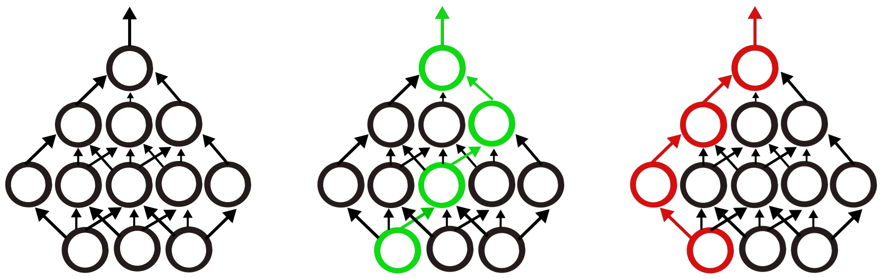


Figure 12: A visual representation of the LTH as introduced by Frankle and Carbin [35]. Let us consider a simplified version of a two hidden layer feedforward neural network as is depicted in the first image on the left. The LTH states that within this neural network there exist multiple smaller networks (represented in green), which can perform just as well as their larger counterpart. Training these sparse models from scratch successfully is only possible as long as their weights are initialized with the same values that were also used when the larger (black) model was initialized. Furthermore, the structure of these sparse models appears to be crucial as well, since it is not possible to randomly extract any subset of weights from an unpruned model and successfully train the resulting sparse network (represented in red in the last figure) from scratch. We visually represent the performance od models that are the winners of the LTH in the two plots reported in Figure 13.

Since the introduction of the idea of the LTH, several research works have focused on understanding what makes some weights so special to be the winners of the initialization lottery. Among the different tested approaches, which will be reviewed in Sec. 6.5, one research direction in particular has looked into how well winning ticket initializations can be transferred among different training settings (datasets and optimizers), an approach which aims at characterizing the winners of the LTH by studying to what extent their inductive

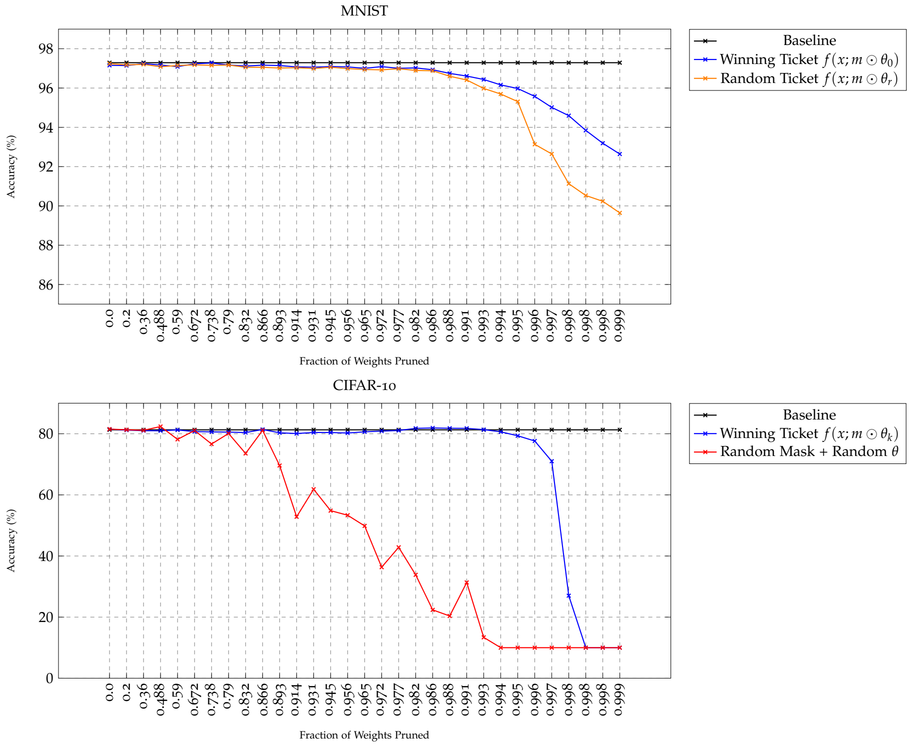


Figure 13: A visual representation of the performance of lottery winners that replicate the findings first presented by Frankle and Carbin [35]. In the first plot we consider a multilayer perceptron that gets trained on the MNIST dataset. After the network gets trained from scratch it obtains a final accuracy of $\approx 97\%$ as reported by the black line. We can observe that winning tickets $f(x; m \odot \theta_0)$ only start performing worse than the network they have been extracted from once a large fraction of their weights gets pruned. We can also observe how crucial it is to re-initialize the weights of the pruned models with the same weights that were used when initializing the unpruned model from scratch (θ_0). If random weights are used instead (θ_r), the pruned masks appear to be less robust to pruning (orange curve). In the second plot we show how important it is for a pruned model to come in the form of $f(x; \odot \theta_k)$ after training a ResNet-50 architecture on the CIFAR-10 dataset, since we show that it is not possible to simply extract any random subset of weights from a deep convolutional network and obtain a performance that is robust to pruning after randomly initializing the parameters of the model (red curve).

biases are generic [87]. The most interesting findings of this study are that winning tickets generalize across datasets, within the natural image domain at least, and that tickets obtained from larger datasets typically generalize better. This opens the door to the transfer of winning tickets between datasets, which makes the high computational cost required to identify them much more acceptable practically, as this cost has to be paid only once and can be shared across datasets.

In this paper, we build on top of this latter work. While Morcos et al. [87] focused on the natural image domain, we investigate the possibility of transferring winning tickets obtained from the natural image domain to datasets in non natural image domains. This question has an important practical interest as datasets in non natural image domains are typically scarcer than datasets in natural image domains. They would therefore potentially benefit more from a successful transfer of sparse networks, since the latter can be expected to require less data for training than large over-parametrized networks. Furthermore, besides studying their generalization capabilities, we also focus on another interesting property that characterizes models that win the LTH, and which so far has received less research attention. As originally presented in [35], pruned models which are the winners of the LTH can yield a final performance which is better than the one obtained by larger over-parametrized networks. In this work we explore whether it is worth seeking for such pruned models when training data is scarce, a scenario that is well known to constraint the training of deep neural networks. To answer these two questions, we carried out experiments on several datasets from two very different non natural image domains: digital pathology and digital heritage.

6.2 DATASETS

We consider seven datasets that come from two different, unrelated sources: histopathology and digital heritage. Each dataset comes with its training, validation and testing splits. Furthermore the datasets change in terms of size, resolution, and amount of labels that need to be classified. We report an overview about the size of these datasets in Table 4 while a visual representation of the samples constituting these datasets in Fig. 14. The Digital-Pathology (DP) data comes from the Cytomine [78] web application, an open-source platform that allows interdisciplinary researchers to work with large-scale images. While Cytomine has collected a large number of datasets over the years, in this work we have limited our analysis to a subset of four datasets that all represent tissues and cells from either human or animal organs: `Human-LBA`, `Lung-Tissues`, and `Mouse-LBA` were originally proposed in [88], while `Bone-Marrow` comes from [56]. All four datasets have been used in [88], that researched whether neural networks pre-trained on natural images could successfully be re-used in the DP domain. In this paper, we explore whether an alternative to the transfer-learning approaches presented in [88] could be based on training pruned networks that are the winners of the LTH. This will allow us to investigate the two research questions introduced in Sec. 6.1: we will explore whether winning initializations that are found on datasets of natural images do generalize to non-natural domains, and whether sparse models winners of the LTH can perform better than larger unpruned models that get trained from scratch. Regarding the field of Digital-Humanities (DH) we have created three novel datasets that all revolve around the classification of artworks. We consider two different classification tasks that have already been thoroughly studied by researchers bridging between the fields of Computer Vision (CV) and DH [82, 110, 123]. The first task consists in identifying the artist of the different artworks, while the second one aims at classifying which kind of artwork is depicted in the different images, a challenge which is usually referred to in the literature as type-classification [82, 110]. When it comes to the artist-classification task we have created two different datasets, which purpose will be better explained in Sec. 6.4.3. All images are publicly available as part of the WikiArt gallery [98] and can also be found within the large popular OmniArt dataset [124]. Albeit as we have seen in Chapter 4 in DH it is actually easier to find large datasets than in histopathology, it is worth mentioning that we have kept the size of these datasets intentionally small in order to fit the research questions introduced in Sec. 6.1. Furthermore, it is also worth noting that there are several additional challenges that need to be overcome when training deep neural networks on artistic collections, which therefore motivate the use of this kind of datasets in this work. The size, texture, and resolution of the images coming from the DH are usually representative of different time periods, artistic movements and might have gone through different digitization processes, which are all reasons that make these datasets largely varied and challenging.

Table 4: A brief overview of the seven different datasets which have been used in this work. As usual throughout this thesis N_t corresponds to the total amount of samples that are present in the dataset, while Q_t represents the number of classes.

Dataset	Training-Set	Validation-Set	Testing-Set	N_t	Q_t
`Human-LBA`	4051	346	1023	5420	9
`Lung-Tissues`	4881	562	888	6331	10
`Mouse-LBA`	1722	716	1846	4284	8
`Bone-Marrow`	522	130	639	1291	8
`Artist-Classification-1`	3103	389	389	3881	20
`Type-Classification`	2868	360	360	3588	20
`Artist-Classification-2`	2827	353	353	3533	19

6.3 EXPERIMENTAL SETUP

We follow an experimental set-up similar to the one that was introduced in [87] (and that has been validated by [41]). Let us define a neural network $f(x;\theta)$ that gets randomly initialized with parameters $\theta_0 \sim \mathcal{D}_\theta$ and then trained for j iterations over an input space $\mathcal{X}$, and

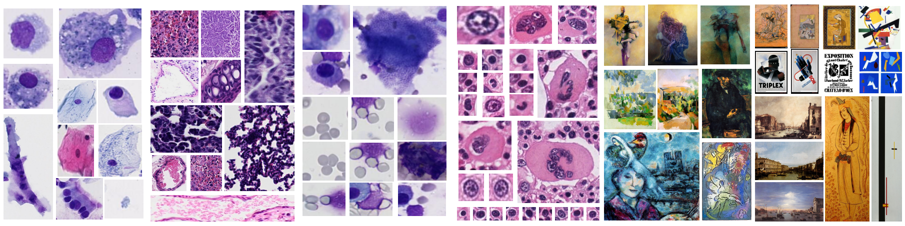


Figure 14: Some image samples that constitute the non-natural image datasets which have been used in this work. From left to right we have the `Human-LBA`, `Lung-Tissues`, `Mouse-LBA` and `Bone-Marrow` datasets, while finally we report some examples that represent artworks which come from the field of digital heritage and that are therefore similar to the images we have used for the experiments reported in Chapter 4.

an output space $\mathcal{Y}$. At the end of training a percentage of the parameters in θ_j gets pruned, a procedure which results in a mask m. The parameters in θ_j which did not get pruned are then reset to the values they had at θ_k, where k represents an early training iteration. A winning ticket corresponds to the combination between the previously obtained mask, and the parameters θ_k, and is defined as $f(x; m \odot \theta_k)$ [1]. Constructing a winning ticket with parameters θ_k, instead of θ_0, is a procedure which is known as late-resetting [37], and is a simple but effective trick that makes it possible to stably find winning initializations in deep convolutional neural networks [37, 87]. In this study $f(x; \theta)$ comes in the form of a ResNet-50 architecture [43] which gets trained on the three popular CV natural image datasets (visually represented in Fig. 15): CIFAR-10/100 and Fashion-MNIST. Following [43, 87], 31 winning tickets $f(x; m \odot \theta_k)$ of increasing sparsity are obtained from each of these three datasets by repeating 31 iterations of network training and magnitude pruning with a pruning rate of 20%. More precisely, at each pruning iteration, the network is trained for several epochs (using early stopping on the validation set as described in the appendix) and then the 20% of weights with the lowest magnitudes are pruned. The parameters θ_k that define each of the 31 tickets are then taken as the weights of the corresponding pruned networks at the kth epoch of the first pruning iteration. Once these pruned networks are found we aim at investigating whether their parameters θ_k contain inductive biases that allow them to generalize to the non-natural image domain. To do so we replace the final fully connected layer of each winning ticket with a randomly initialized layer that has as many output nodes as there are classes to classify. We then fine-tune each of these networks on the non-natural image

1 Note that this formulation generalizes the original version of the LTH [35] that we have represented in Fig. 12, where a winning ticket is obtained after resetting the unpruned parameters of the network to the values they had right after initialization, therefore defining a winning ticket as $f(x; m \odot \theta_0)$.

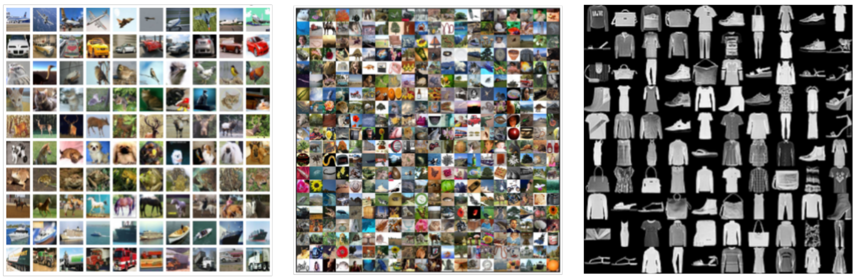


Figure 15: The three natural datasets used in our experiments to find winning tickets. From left to right samples from the CIFAR-10/100 and Fashion-MNIST datasets.

datasets considered in this study. At the end of training, we study the performance of each winning ticket in two different ways. First, we compare the performance of each network to the performance of a fully unpruned network that gets randomly initialized and trained from scratch. Second, we also compare the performance of winning tickets that have been found on a natural image dataset to 31 new sparse models that are the winners of the LTH on the considered target dataset. Since it is not known to which extent pruned networks that contain weights that are the winners of the LTH on a natural image dataset can generalize to target distributions that do not contain natural images, we report the first results that investigate the potential of a novel transfer-learning scheme which has so far only been studied on datasets from the natural image domain. Moreover, testing the performance of sparse networks that contain winning tickets that are specific to a non-natural image target distribution also allows us to investigate whether it is worth pruning large networks with the hope of finding smaller models that might perform better than a large over-parametrized one. As mentioned in Sec. 6.1, pruned networks that are initialized with the winning weights can sometimes perform better than a fully unpruned network. Identifying such sparse networks leads to a very significant reduction of model size, which can be a very effective way of regularization when training data is scarce.

6.4 RESULTS

The results of all our experiments are visually reported in the plots of Fig. 16. Each line plot represents the final performance that is obtained by a pruned model that contains a winning ticket initialization on the final testing-set of our target datasets. This performance is reported on the y-axis of the plots, while on the x-axis we represent the fraction of weights that is pruned from the original ResNet-50 architecture. As explained in the previous section the performance of each winning ticket is compared to the performance that is obtained by an unpruned, over-parametrized architecture that is reported by the black dashed lines. The models that are the winners of the LTH on a natural image dataset are reported by the green, red and purple lines, while the winners of the LTH on a non-natural target dataset are reported by the blue lines. Furthermore, when it comes to the latter lottery tickets, we also report the performance that is obtained by winning tickets that get randomly reinitialized ($f(x; m \odot \theta_0^{'})$ with $\theta_0^{'} \sim \mathcal{D}_\theta$). These results are reported by the orange lines. Shaded areas around all line plots correspond to ± 1 std. that has been obtained after repeating and averaging the results of our experiments over four different random seeds.

6.4.1 *On the Importance of Finding Winning Initializations*

We can start by observing that pruned models which happen to be the winners of the LTH either on a natural dataset, or on a non-natural one, can maintain a good final performance until large pruning rates are reached. This is particularly evident on the first three datasets, where models that keep only $\approx 1\%$ of their original weights barely suffer from any drop in performance. This gets a little bit less evident on the last three datasets, where the performance of winning ticket initializations that are directly found on the considered target dataset starts getting harmed once a fraction of ≈ 97 of original weights are pruned. These results show that an extremely large part of the parameters of a ResNet-50 architecture can be considered as superfluous, therefore confirming the LTH when datasets contain non-natural images. More importantly, we also observe that pruned models winners of the LTH, significantly outperform larger over-parametrized models that get trained from scratch. This can be very clearly seen in all plots where the performance of pruned models is always consistently better than what is reported by the black dashed line. To get a better sense of how much these pruned networks perform better than their larger unpruned counterparts, we report in Table 5 the performance that is obtained by the best performing pruned model, found over all 31 possible pruned models, and compare it to the performance of an unpruned architecture. The exact fraction of weights which is pruned from an original ResNet-50 architecture is reported in Table 6 for each configuration. We can observe that no matter which dataset has been used as a source for finding a winning ticket initialization, all pruned networks reach a final accuracy that is significantly higher than the one that is obtained after training an unpruned model from scratch. While in most cases the difference in terms of performance is of $\approx 10\%$ (see e.g. the `Human-LBA, Lung-Tissues` and the `Type-Classification` datasets), it is worth highlighting that there are other cases in which this difference is even larger. This is the case for the `Mouse-LBA` and `Artist-Classification-1` datasets where a winning ticket coming from the CIFAR-10 dataset performs more than 20% better than a model trained from scratch. These results show that in order to maximize the performance of deep networks it is always worth finding and training pruned models which are the winners of the LTH.

Table 5: The results comparing the performance that is obtained on the testing-set by the best pruned model winner of the LTH, and an unpruned architecture trained from scratch. The overall best performing model is reported in a green cell, while the second best one in a yellow cell. We can observe that pruned models winners of the LTH perform significantly better than a larger over-parametrized architecture that gets trained from scratch. As can be seen by the results obtained on the `Mouse-LBA` and `Artist-Classification-1` datasets the difference in terms of performance can be particularly large ($\approx$ 20%).

Target-Dataset	Scratch-Training	CIFAR-10	CIFAR-100	Fashion-MNIST	Target-Ticket
`Human-LBA`	$71.85_{\pm 1.12}$	$79.17_{\pm 1.85}$	$76.97_{\pm 0.73}$	$77.32_{\pm 1.85}$	$81.72_{\pm 0.39}$
`Lung-Tissues`	$84.75_{\pm 0.81}$	$88.90_{\pm 1.97}$	$87.61_{\pm 0.90}$	$87.61_{\pm 0.11}$	$90.48_{\pm 0.16}$
`Mouse-LBA`	$48.17_{\pm 1.18}$	$74.20_{\pm 2.04}$	$57.42_{\pm 0.48}$	$52.27_{\pm 1.73}$	$68.20_{\pm 3.79}$
`Bone-Marrow`	$64.66_{\pm 1.36}$	$71.75_{\pm 3.36}$	$69.87_{\pm 0.39}$	$68.77_{\pm 0.39}$	$72.55_{\pm 0.46}$
`Artist-Classification-1`	$45.88_{\pm 0.42}$	$66.58_{\pm 1.54}$	$65.55_{\pm 1.79}$	$63.88_{\pm 0.12}$	$58.74_{\pm 1.92}$
`Type-Classification`	$41.36_{\pm 2.31}$	$58.63_{\pm 2.97}$	$60.56_{\pm 0.44}$	$58.92_{\pm 0.59}$	$50.44_{\pm 2.23}$

Table 6: Some additional information about the lottery winners which performance is reported in Table 5. For each winning ticket we report the fraction of weights that is pruned from an original ResNet-50 architecture and that therefore characterizes the level of sparsity of the overall best performing lottery ticket. The results in the Scratch-Training column are not reported since these are unpruned models that are trained from scratch.

Target-Dataset	Scratch-Training	CIFAR-10	CIFAR-100	Fashion-MNIST	Target-Ticket
`Human-LBA`	-	0.945	0.79	0.886	0.832
`Lung-Tissues`	-	0.977	0.977	0.672	0.965
`Mouse-LBA`	-	0.972	0.893	0.738	0.931
`Bone-Marrow`	-	0.866	0.988	0.931	0.914
`Artist-Classification-1`	-	0.972	0.993	0.991	0.931
`Type-Classification`	-	0.991	0.931	0.995	0.963

6.4.2 *On the Generalization Properties of Lottery Winners*

We then investigate whether natural image tickets can generalize to the non-natural setting. Findings differ across datasets. When considering the datasets that come from the field of DP, we can see that, in three out of four cases, winning tickets that are found on a natural image dataset get outperformed by sparse winning networks that come after training a model on the biomedical dataset. This is particularly evident in the results obtained on the `Human-LBA` and `Lung-Tissues` datasets where the highest testing-set accuracy is consistently reached

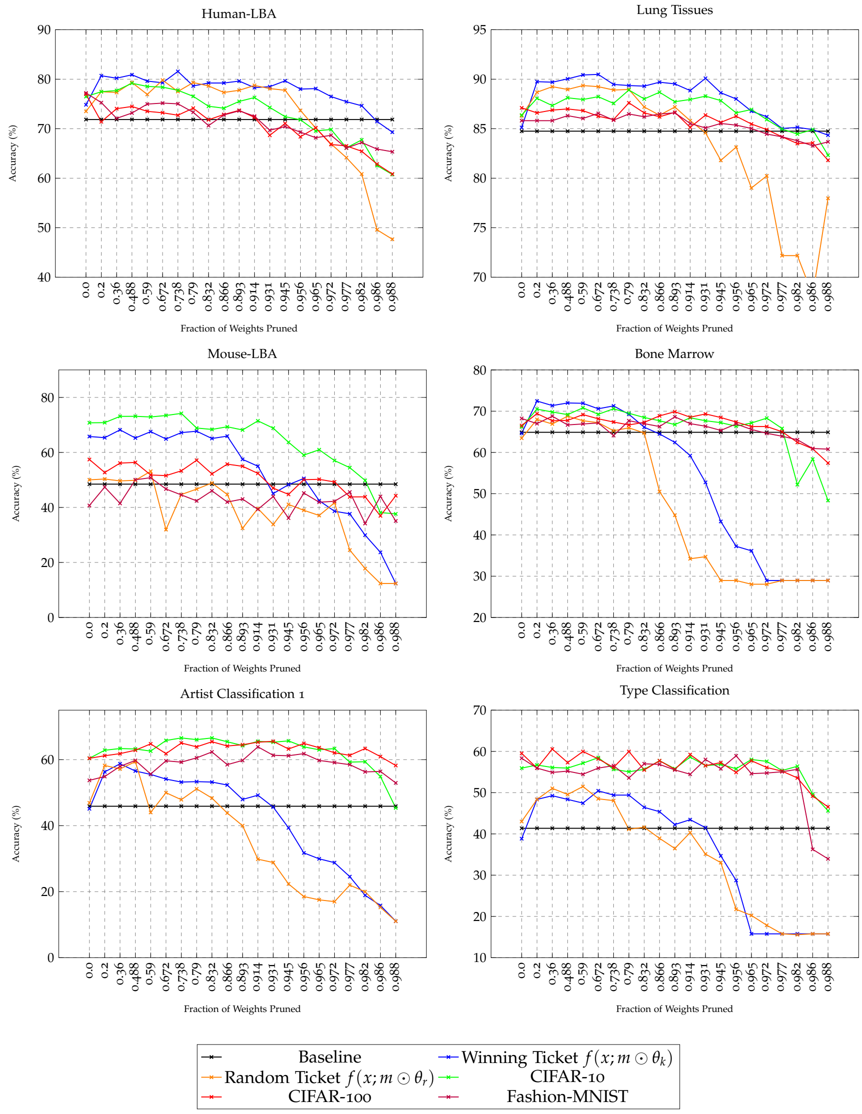


Figure 16: An overview of the results showing that sparse models that are the winners of the LTH (represented by the coloured lines) significantly outperform unpruned networks which get randomly initialized and trained from scratch (dashed black line). This happens to be the case on all tested datasets, no matter whether a winning initialization comes from a natural image source or not. It is however worth mentioning that, especially on the biomedical datasets, natural image tickets get outperformed by sparse networks that that are the winners of the LTH on a biomedical dataset. On the other hand this is not the case when it comes to the classification of arts where natural image tickets outperform the ones which are found within artistic collections.

by the blue line plots. When it comes to the `Bone-Marrow` dataset the difference in terms of performance between the best natural image ticket, in this case coming from the CIFAR-10 dataset, and the one coming from the biomedical dataset, is less evident (see Table 5 for the exact accuracies). Furthermore, it is worth highlighting that on the Bone-Marrow dataset, albeit natural image models seem to get outperformed by the ones found on the biomedical dataset, the performance of the latter ones appears to be less stable once extremely large pruning rates are reached. When it comes to the Mouse-LBA dataset these results slightly differ. In fact, this dataset corresponds to the only case where a natural image source ticket outperforms a non-natural one. As can be seen, by the green line plot, pruned models coming from the CIFAR-10 dataset outperform the ones found on the Mouse-LBA dataset. When focusing our analysis on the classification of arts, we see that the results change greatly from the ones obtained on the biomedical datasets. In this case, all of the natural image lottery winners, no matter the dataset they were originally found on, outperform the same kind of models that were found after training a full network on the artistic collection. We can see from Table 5 that the final testing performance is similar among all of the best natural image tickets. Similarly to what has been noticed on the Bone-Marrow dataset we can again observe that tickets coming from a non-natural data distribution seem to suffer more from large pruning rates.

These results show both the potential and the limitations that natural image winners of the LTH can offer when they are fine-tuned on datasets of non-natural images. The results obtained on the artistic datasets suggest that winning initializations contain inductive biases that are strong enough to get at least successfully transferred to the artistic domain, therefore confirming some of the claims that were made by Morcos et al. [87]. However, it also appears that there are stronger limitations to the transferability of winning initializations which were not observed by Morcos et al. [87]. In fact, our results show that on DP data the best strategy is to find a winning ticket directly on the biomedical dataset, and that winning initializations found on natural image datasets, albeit outperforming a randomly initialized unpruned network, perform worse than pruned models that are the winners of the LTH on a biomedical dataset.

6.4.3 *Additional Studies*

To characterize the transferability of winning initializations even more, while at the same time gaining a deeper understanding of the LTH, we have performed a set of three additional experiments which help us characterize this phenomenon even better.

6.4.3.1 *Lottery Tickets VS fine-tuned pruned models*

So far we have focused our transfer-learning study on lottery tickets that come in the form of $f(x; m \odot \theta_k)$, where, as mentioned in Sec. 6.3, θ_k corresponds to the weights that parametrize a neural network at a very early training iteration. This formalization is however different from more common transfer-learning scenarios like the ones we have explored in Chapter 4 where neural networks get transferred with the weights that are obtained at the end of the training process [88, 110]. We have therefore studied whether there is a difference in terms of performance between transferring and fine-tuning a lottery ticket with parameters θ_k, and the same kind of pruned network which is initialized with the weights that are obtained once the network is fully trained on a source task. We define these kind of models as $f(x; m \odot \theta_i)$ where i stays for the last training iteration. We report some examples of this behaviour in the plots presented in Fig. 17, where we consider $f(x; m \odot \theta_i)$ models which were trained on the CIFAR-10 and CIFAR-100 datasets, and then transferred and fine-tuned on the `Human-LBA` dataset. We found that these models overall perform worse than lottery tickets, while also being less robust to pruning. This also shows that on this dataset, the slightly inferior performance of the natural image tickets with respect to the target tickets is not due to the weight re-initialization.

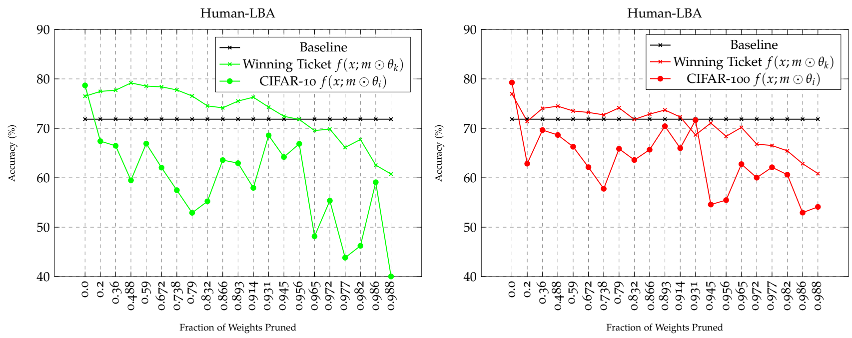


Figure 17: Our results showing the advantages of transferring lottery winners over pruned models that are fully fine-tuned on natural datasets (CIFAR-10/100). We can observe that their performance is overall inferior to the one of lottery tickets and that these models are significantly less robust to pruning. We believe that the reason behind their poor performance revolves around the fact that, once completely trained on a specific source task, and after having gone through the pruning stage, these models lose the necessary flexibility that is required for them to adapt to a new task.

6.4.3.2 *Transferring tickets from similar non-natural domains*

We investigated whether it is beneficial to fine-tune lottery winners that, instead of coming from a natural image distribution, come from a related non-natural dataset. Specifically we tested whether winning tickets generated on the `Human-LBA` dataset generalize to the `Mouse-LBA` one (since both datasets are representative of the field of Live-Blood-Analysis), and whether lottery winners coming from the `Artist-Classification-1` dataset generalized to the `Artist-Classification-2` one. We visually represent these results in Fig. 18. As one might expect, we found that it is beneficial to transfer winning tickets that come from a related source. Specifically, `Human-LBA` tickets can perform just as well as winning tickets that are generated on the `Mouse-LBA` dataset, while at the same time also being more robust to large pruning rates. When it comes to lottery winners found on the `Artist-Classification-1` dataset we have observed that these tickets can even outperform the ones generated on the `Artist-Classification-2` one. Overall these results confirm the claims that we made in Chapter 4 where we already highlighted the benefits that could come from transferring models that were trained on a similar source as the target task. This conclusion now also holds for models that are the winners of the LTH.

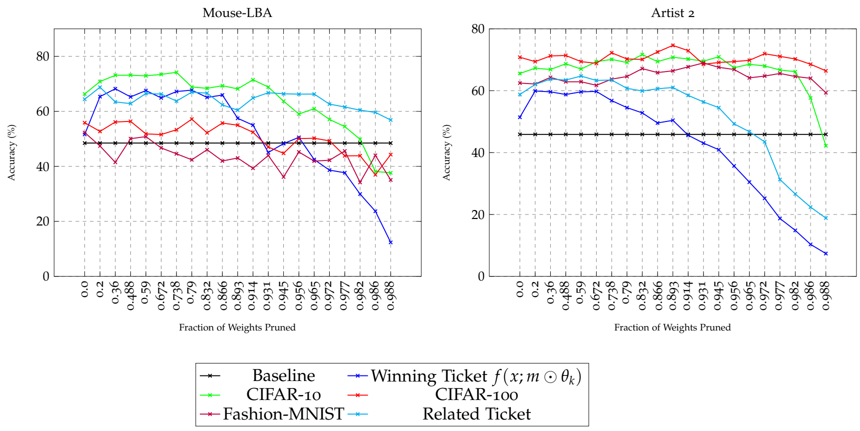


Figure 18: Our results showing the benefits of transferring lottery winners that have been identified on a related source task. We can observe that on the `Mouse-LBA` dataset, winning tickets that were obtained on the `Human-LBA` dataset are the ones that are the most robust ones to pruning, while on the `Artist-Classification-2` dataset we can observe that lottery winners that have been obtained on the `Artist-Classification-1` dataset are both more robust to pruning, while they also yield overall better performance.

6.4.3.3 *On the size of the training set*

We have observed from the blue line plots of Fig. 16 that there are cases in which lottery winners are very robust to extremely large pruning rates (see as an example the first and second plots), while there are other cases in which their performance deteriorates faster with respect to the fraction of weights that get pruned. The most robust performance is obtained by winning tickets that are generated on the `Human-LBA` and `Lung-Tissues` datasets, which are the two target datasets that contain the largest amount of training samples. We have therefore studied whether there is a relationship between the size of the training data that is used for finding lottery winners, and the robustness in terms of performance of the resulting pruned models. We generated lottery winners after incrementally reducing the size of the training data by 75%, 50% and 25%, and then investigated whether we could observe a similar drop in performance as the one we have observed in the last three blue line-plots of Fig. 16 once a large fraction of weights got pruned. Perhaps surprisingly, we have observed that this was not the case, and as can be seen by the plots represented in Fig. 19, the performance of lottery winners that are found when using only 25% of the training set is just as stable as the one of winning tickets which are generated on the entire dataset. It is however worth mentioning that, albeit the performance of such sparse models is robust, their final performance on the testing set is lower than the one that is obtained by winning tickets that have been trained on the full training data distribution.

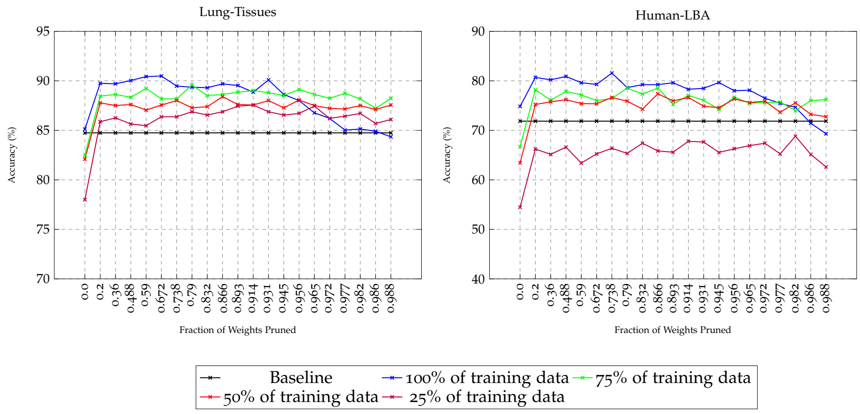


Figure 19: Our study showing that the robustness to large pruning rates of lottery winners does not depend from the size of the training dataset. We can observe that even when lottery tickets after are trained on only 25% of the dataset their performance remains stable with respect to the fraction of pruned weights. These results suggest that the less stable performance of `Bone-Marrow` lottery tickets observed in Fig. 16 does not depend from the small training set.

6.5 RELATED WORK

The research presented in this paper contributes to a better understanding of the LTH by exploring the generalization and transfer-learning properties of lottery tickets. The closest approach to what has been presented in this work is certainly [87], which shows that winning models can generalize across datasets of natural images and across different optimizers. As mentioned in Sec. 6.3, a large part of our experimental setup is based on this work. Besides the work presented in [87], there have been other attempts that aimed to better understand the LTH after studying it from a transfer-learning perspective. However, just as the study presented by Morcos et al. [87], all this research limited its analysis to natural images. Van Soelen and Sheppard [146] transfer winning tickets among different partitions of the CIFAR-10 dataset, while Mehta [81] shows that sparse models can successfully get transferred from the CIFAR-10 dataset to other object recognition tasks. While these results seem to suggest that lottery tickets contain inductive biases which are strong enough to generalize to different domains, it is worth highlighting that their transfer-learning properties were only studied after considering the CIFAR-10 dataset as a possible source for winning ticket initializations, a limitation which we overcome in this work. It is also worth mentioning that the research presented in this paper is strongly connected to the work presented by Frankle et al. [37]. While the first paper that introduced the LTH limited its analysis to relatively simple neural architectures, such as multilayer perceptrons and convolutional networks which were tested on small CV datasets, the presence of winning initializations in larger, more popular convolutional models such as the ones that we used in Chapter 4 trained on large datasets [106] was only first presented by [37]. Since in this work we have used a ResNet-50 architecture [47], we have followed all the recommendations that were introduced in [37], for successfully identifying the winners of the LTH in larger models. More specifically we mention the late-resetting procedure which resets the weights of a pruned model to the weights that are obtained after k training iterations instead of to the values which were used at the beginning of training (as explained in Sec. 6.3), a procedure which has shown to be related to *linear mode connectivity* [36]. While the work presented in this paper has limited its analysis to networks that minimize an objective function that is relevant for classification problems, it is worth noting that more recent approaches have identified lottery winners in different training settings. Yu et al. [164] have shown that winning initializations can be found when neural networks are trained on tasks ranging from natural language processing to reinforcement learning, while Sun et al. [125] successfully identify sparse winning models in a multi-task learning scenario. As future work, we want to study whether lottery tickets can be found on different neural architectures, and also when neural networks are trained on CV tasks other than classification. More specifically we aim at studying whether winners of the LTH, which are found on popular natural image datasets such as [74] and [30] when tackling image localization and segmentation tasks, can generalize to non-natural settings which might include the segmentation of biomedical data, or the localization of objects within artworks.

6.6 CONCLUSION

We have investigated the transfer learning potential of pruned neural networks that are the winners of the LTH from datasets of natural images to datasets containing non-natural images. We have explored this in training conditions where the size of the training data is relatively small. All of the results presented in this work confirm that it is always beneficial to train a sparse model, winner of the LTH, instead of a larger over-parametrized one. Regarding our study on the transferability of winning tickets we have reported the first results which study this phenomenon under non-natural data distributions by using datasets coming from the fields of digital pathology and heritage. While for the case of artistic data it seems that winning tickets from the natural image domain contain inductive biases which are strong enough to generalize to this specific domain, we have also shown that this approach can present stronger limitations when it comes to biomedical data. This probably stems from the fact that DP images are further away from natural images than artistic ones. We have also shown that lottery tickets perform significantly better than fully trained pruned models, that it is beneficial to transfer lottery winners from different, but related, non-natural sources, and that the performance of lottery tickets is not dependant on the size of the training data. To conclude, we provide a better characterization of the LTH while simultaneously showing that when training data is limited, the performance of deep neural networks can get significantly improved by using lottery winners over larger over-parametrized ones.

Part III

TRANSFER LEARNING FOR REINFORCEMENT LEARNING

7 THE DEEP QUALITY-VALUE LEARNING FAMILY OF ALGORITHMS

Contributions and Outline

In the first part of this thesis we have thoroughly studied the level of transferability of deep neural networks that get trained in a supervised learning fashion. From the results of our studies we concluded that significant benefits can come from using pre-trained models over networks that get trained from scratch, and that transfer learning can be a valuable machine learning paradigm for studying the generalization properties of neural networks. So far we have always considered the setting in which, before getting trained on a new task, a model could rely on some **fixed** knowledge it had learned on a separate but related task. In this chapter however, we take a different approach: we develop a novel family of Deep Reinforcement Learning (DRL) algorithms, that instead of learning on top of on what a model has learned in the past, rely on what is **simultaneously** being learned by a different, yet related model. The information that is being learned by one model can be transferred and used by a second one in a very dynamical fashion, which within the DRL context yields faster, more robust and better performance.

The structure of this chapter is the following: in Sec. 7.1 and Sec. 7.2 we provide the reader with some background information about the field of DRL and present the necessary mathematical notation that will be used throughout this chapter. In Sec. 7.4 we introduce the main algorithmic contributions of our research, the performance of which is studied from different perspectives in Sec. 7.5. The chapter ends with Sec. 7.6 and Sec. 7.7 which provide a set of additional studies that characterize the performance of our newly introduced algorithms even further, while providing insights about potential possible future work.

This chapter is based on the following two publications: Sabatelli et al. [109] and Sabatelli et al. [111]

7.1 INTRODUCTION

In value-based Reinforcement Learning (RL) the aim is to construct algorithms which learn *value functions* that are either able to estimate how good or bad it is for an agent to be in a particular state, or how good it is for an agent to perform a particular action in a given state. Such functions are respectively denoted as the state-value function $V(s)$, and the state-action value function $Q(s, a)$ [128]. In Deep Reinforcement Learning (DRL) the aim is to approximate these value functions with e.g. deep convolutional neural networks [67], which can serve as universal function approximators and powerful feature extractors. Classic model-free RL algorithms like Q-Learning [150], Double Q-Learning [142] and SARSA [105] have all led to the development of a "deep" version of themselves in which the original RL update rules are expressed as objective functions that can be minimized by gradient descent [84, 143, 165]. Despite their successful applications [70], the aforementioned algorithms only aim at approximating the Q function, while completely ignoring the V function. This approach, however, is prone to issues that go back to standard RL literature. As shown by Van Hasselt, Guez, and Silver [143] the DQN algorithm [84] is known to overestimate the values of the Q function and requires an additional target network to not diverge (which role, as shown by Achiam, Knight, and Abbeel [2], is not yet fully understood). These overestimations can partially be corrected by the DDQN [143] algorithm, which, despite yielding stability improvements, does not always prevent its Q networks from diverging [**van2018deep**] and sometimes even underestimating the Q function. Furthermore, DRL algorithms are also extremely slow to train. In what follows, we introduce a new family of DRL algorithms based on the key idea of simultaneously learning the V function alongside the Q function with two separate neural networks. Our main insight is that by jointly approximating the V function and the Q function, the task of learning one of these value functions can be sped up if the model that is responsible for learning it, can rely on what is being learned by the model responsible for learning the remaining value function. We show that this simple, yet effective idea yields faster, more robust and better model-free Deep Reinforcement Learning.

7.2 PRELIMINARIES

We formally define the RL setting as a Markov Decision Process (MDP) where the main components are a finite set of states $\mathcal{S} = \{s^1, s^2, ..., s^n\}$, a finite set of actions $\mathcal{A}$ and a time-counter variable t. In each state $s_t \in \mathcal{S}$, the RL agent can perform an action $a_t \in \mathcal{A}(s_t)$ and transit to the next state as defined by a transition probability distribution $p(s_{t+1}|s_t, a_t)$. When moving from s_t to a successor state s_{t+1} the agent receives a reward signal r_t coming from the reward function $\Re(s_t, a_t, s_{t+1})$. The actions of the agent are selected based on its policy

$\pi : \mathcal{S} \to \mathcal{A}$ that maps each state to a particular action. For every state $s \in \mathcal{S}$, under policy π its *value function* V^π is defined as:

$$V^\pi(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \middle| s_t = s, \pi\right], \tag{51}$$

which denotes the expected cumulative discounted reward that the agent will get when starting in state s and by following policy π thereafter. Similarly, we can also define the *state-action* value function Q for denoting the value of taking action a in state s based on policy π as:

$$Q^\pi(s,a) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \middle| s_t = s, a_t = a, \pi\right]. \tag{52}$$

Both functions are computed with respect to the discount factor $\gamma \in [0, 1]$ which controls the trade-off between immediate and long term rewards. The goal of an RL agent is to find a policy π^* that realizes the optimal expected return:

$$V^*(s) = \max_\pi V^\pi(s), \text{ for all } s \in \mathcal{S} \tag{53}$$

and the optimal Q value function:

$$Q^*(s,a) = \max_\pi Q^\pi(s,a) \text{ for all } s \in \mathcal{S} \text{ and } a \in \mathcal{A}. \tag{54}$$

It is well-known that optimal value functions satisfy the Bellman optimality equation as given by

$$V^*(s_t) = \max_a \sum_{s_{t+1}} p(s_{t+1}|s_t, a)\left[\Re(s_t, a, s_{t+1}) + \gamma V^*(s_{t+1})\right] \tag{55}$$

for the state-value function, and by

$$Q^*(s_t, a_t) = \sum_{s_{t+1}} p(s_{t+1}|s_t, a_t)\left[\Re(s_t, a_t, s_{t+1}) + \gamma \max_a Q^*(s_{t+1}, a)\right], \tag{56}$$

for the state-action value function. Both functions can either be learned via Monte Carlo methods or by Temporal-Difference (TD) learning [126], with the latter approach being so far the most popular choice among model-free RL algorithms [105, 142, 150].

7.3 RELATED WORK

While RL algorithms have been successfully combined with shallow neural networks for over two decades [134], the use of these algorithms with deeper architectures is more recent. The contribution which has established the potential of DRL can certainly be identified with the Deep-Q-Network (DQN), the first algorithm which uses a convolutional neural network for successfully learning an approximation of the Q function from high dimensional inputs [84]. This approximation is learned by reshaping the popular Q-Learning algorithm introduced by Watkins and Dayan [150] to an objective function which can be minimized by gradient descent. The original Q-Learning algorithm learns the state-action value function as follows

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha\big[r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a) - Q(s_t, a_t)\big] \quad (57)$$

where α corresponds to the learning rate. The DQN algorithm adapts this update rule to a differentiable loss function which can be used for training a neural network that is parametrized by θ. This objective function comes in the following form:

$$L(\theta) = \mathbb{E}_{\langle s_t, a_t, r_t, s_{t+1}\rangle \sim U(D)} \bigg[\Big(r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a; \theta^-) - Q(s_t, a_t; \theta)\Big)^2\bigg]. \quad (58)$$

Within this loss there are two components which ensure stable training. The first one is the Experience-Replay memory buffer (D), first introduced by Lin [73], a buffer coming in the form of a queue which stores RL experiences $\langle s_t, a_t, r_t, s_{t+1}\rangle$. When it comes to the popular Atari Arcade Learning (ALE) [9] benchmark, the DQN algorithm uniformly samples mini-batches of 32 experiences for minimizing Eq. 58, a procedure which starts as soon as at least 50.000 experiences are stored within the queue. Furthermore, there is a second component which ensures stable training denoted as the target-network. DQN learns an approximation of the Q function via TD-Learning, meaning that the approximated Q-function is regressed towards TD-targets which are computed by the approximated Q function itself. The TD-target, defined as y_t^{DQN}, is expressed as follows:

$$y_t^{DQN} = r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a; \theta^-), \quad (59)$$

and is computed by the target network θ^- instead from the online Q-network θ. The online network, and its target counterpart, have the exact same structure, with the main difference being that the parameters of the latter do not get optimized each time a mini-batch of experiences is sampled from the memory buffer. On the contrary its weights are temporally frozen and are only periodically updated with the θ weights (as defined by an appropriate hyperparameter). Given a training iteration i, differentiating the objective function of Eq. 58 with respect to θ gives the following gradient:

$$\nabla_{\theta_i} y_t^{DQN}(\theta_i) = \mathbb{E}_{\langle s_t, a_t, r_t, s_{t+1}\rangle \sim U(D)} \bigg[\Big(r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a; \theta^-_{i-1}) - Q(s_t, a_t; \theta_i)\Big) \nabla_{\theta_i} Q(s_t, a_t; \theta_i)\bigg]. \quad (60)$$

Despite yielding super-human performance on most games coming from the ALE, DQN has shown to be suffering from the same issues which characterize the Q-Learning algorithm [142]. Among these issues we mention the overestimation bias of the Q-function which has been well characterized in the tabular setting by Van Hasselt [142] and later in the deep learning context by Van Hasselt, Guez, and Silver [143]. In short, DQN is prone to learn overestimated Q-values because the same values are used both for selecting an action ($\max_{a\in\mathcal{A}}$) and for evaluating it ($Q(s_{t+1}, a; \theta^{-})$). As originally presented by Van Hasselt, Guez, and Silver [143] this becomes clearer when re-writing Eq. 59 as:

$$y_t^{DQN} = r_t + \gamma\, Q(s_{t+1}, \arg\max_{a\in\mathcal{A}} Q(s_{t+1}, a; \theta); \theta^{-}). \tag{61}$$

As a result, DQN tends to approximate the expected maximum value of a state, instead of its maximum expected value. To solve this problem the DDQN algorithm untangles the selection of an action from its evaluation by taking advantage of the target network θ^{-}. DDQN's target is the same as DQN's with the main difference being that the selection of an action, given by the online Q-network θ, and the evaluation of the resulting policy, given by θ^{-}, can get unbiased by symmetrically updating the two sets of weights (θ and θ^{-}). This can be achieved by regularly switching their roles during training.

Several extensions of DQN and DDQN have been proposed over the years, to make these algorithms learn faster and more data-efficient. We refer the reader to [70] for a more in-depth review of these contributions. Within this chapter, we are only interested in synchronous DRL algorithms which learn an approximation of a value function. This is achieved by following the same experimental setups that have been used for DQN and DDQN, and that will be reviewed in Sec. 7.5.1 of this chapter. Therefore, in what follows we will aim at comparing our novel DRL algorithms to DQN and DDQN only, while leaving their potential integration within more sophisticated DRL techniques as future work.

7.4 A NOVEL FAMILY OF DEEP REINFORCEMENT LEARNING ALGORITHMS

Just as much as DQN and DDQN are based on two tabular RL algorithms, so are the main contributions presented in this chapter. More specifically we extend two RL algorithms which were first introduced by Wiering [155] and then extended by Wiering and Van Hasselt [156] to the use of deep neural networks that serve as function approximators. Training these algorithms robustly is done by taking advantage of some of the techniques which have been reviewed in the previous section.

7.4.1 *DQV-Learning*

Our first contribution is the Deep Quality-Value (DQV) Learning algorithm, a novel DRL algorithm which aims at jointly approximating the V function alongside the Q function in an *on-policy* learning setting. This algorithm is based on the QV(λ) algorithm [155], a tabular RL algorithm which learns the V function via the simplest form of TD-Learning [126], and uses the estimates that are learned by this value function to update the Q function in a Q-Learning resembling way. Specifically, after a transition $\langle\ s_t,\ a_t,\ r_t,\ s_{t+1}\ \rangle$, QV($\lambda$) uses the TD($\lambda$) learning rule [126] to update the V function for all states:

$$V(s) := V(s) + \alpha\big[r_t + \gamma V(s_{t+1}) - V(s_t)\big]e_t(s), \tag{62}$$

where α stands again for the learning rate and γ is the discount factor, while $e_t(s)$ are the elibility traces [40, 96, 157] that are necessary for keeping track if a particular state has occurred before a certain time-step or not. These are updated for all states as follows:

$$e_t(s) = \gamma\lambda e_{t-1}(s) + \eta_t(s), \tag{63}$$

where $\eta_t(s)$ is an indicator function that returns a value of 1 whether a particular state occurred at time t and 0 otherwise. Before updating the V function, QV(λ) updates the Q function first, and does this via the following update rule:

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha\big[r_t + \gamma V(s_{t+1}) - Q(s_t, a_t)\big]. \tag{64}$$

We take inspiration from this specific learning dynamic and aim at learning an approximation of both the V function, and the Q function, with two neural networks that are respectively parametrized by Φ and θ. To do so, we follow the same principles which have led to the development of the DQN algorithm and that reshape Eq. 57 to Eq. 58. Therefore, starting from Eq.62, and after removing $e_t(s)$ for simplicity, we get the following objective function which is used by DQV for learning the state-value function:

$$L(\Phi) = \mathbb{E}_{\langle s_t, a_t, r_t, s_{t+1}\rangle \sim U(D)}\left[\big(r_t + \gamma V(s_{t+1}; \Phi^-) - V(s_t; \Phi)\big)^2\right], \tag{65}$$

while the following loss is minimized for learning the Q function when starting from Eq. 64:

$$L(\theta) = \mathbb{E}_{\langle s_t, a_t, r_t, s_{t+1}\rangle \sim U(D)}\left[\big(r_t + \gamma V(s_{t+1}; \Phi^-) - Q(s_t, a_t; \theta)\big)^2\right], \tag{66}$$

where D is again the Experience-Replay memory buffer, used for uniformly sampling batches of RL trajectories $\langle s_t, a_t, r_t, s_{t+1}\rangle$, and Φ^- is the target-network used for the construction of the TD-errors. Please note that the role of this target network is different from its role within the DQN algorithm. In DQV this network corresponds to a copy of the network which approximates the state-value function and not the state-action value function. It is also worth noting that both networks learn from the same TD-target which comes in the following form:

$$y_t^{DQV} = r_t + \gamma V(s_{t+1}; \Phi^-). \tag{67}$$

The gradient with respect to both loss functions can be easily expressed similarly as done in Eq. 60 for the DQN algorithm.

7.4.2 *DQV-Learning with Multilayer Perceptrons*

We start by exploring whether this learning dynamic of jointly approximating two value functions simultaneously, and let the Q function bootstrap from the TD-targets that are learned from the V network can yield successful results on a set of preliminary experiments. To do so, we use two classic control problems that are well known in the RL literature: `Acrobot` [127] and `Cartpole` [7] with both environments being provided by the Open-AI Gym package [17]. We approximate the V function and the Q function with a two hidden layer Multilayer Perceptron (MLP) that is activated by a ReLU non linearity ($f(x) = max(0, x)$) and compare the performance of DQV to the one of the DQN and the DDQN algorithms, which use the same MLP but for approximating the Q function only. Given the simplicity of these two control problems we did not integrate DQV with the target network Φ^- yet. Our preliminary results reported in Fig. 20, show the benefits that can come from training two separate networks with the update rules reported in Eq.65 and Eq.66. We can in fact observe that on both control problems DQV-Learning outperforms DQN and DDQN, by converging significantly faster.

7.4.3 *DQV-Max Learning*

Based on the successful results presented in Fig. 20 that highlight the potential benefits that could come from jointly approximating two value functions over one, we now introduce the Deep Quality-Value-Max (DQV-Max) algorithm, a novel DRL algorithm which builds on top of some of the ideas that characterize DQV. Similarly as done for DQV, we still aim at jointly learning an approximation of the V function and the Q function, but in this case, the goal is to do this with an *off-policy* learning scheme. To construct this algorithm we take inspiration from the QV-Max RL algorithm introduced by Wiering and Van Hasselt [156]. The key component of QV-Max is the use of the $\max_{a \in \mathcal{A}} Q(s_{t+1}, a)$ operator, which makes RL algorithms learn *off-*

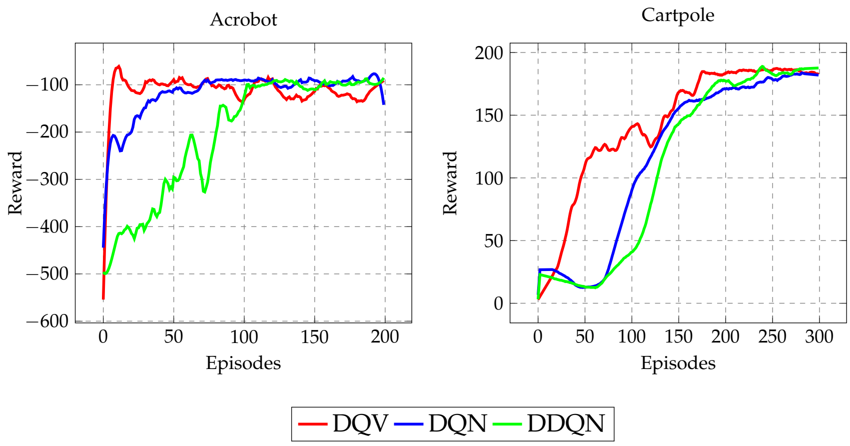


Figure 20: Our preliminary results that show the benefits in terms of convergence time that can come from jointly approximating the V function alongside the Q function. We can observe that the DQV-Learning algorithm yields faster convergence when compared to popular algorithms which only approximate the Q function: DQN and DDQN.

policy. We use this operator when approximating the V function and for computing TD-errors which correspond to the ones that are also used by the DQN algorithm. However, within DQV-Max, these TD-errors are used by the state-value network and not by the state-action value network. This results in the following loss which is used for learning the V function:

$$L(\Phi) = \mathbb{E}_{\langle s_t,a_t,r_t,s_{t+1}\rangle \sim U(D)}\Big[\big(r_t + \gamma \max_{a\in\mathcal{A}} Q(s_{t+1},a;\theta^-) - V(s_t;\Phi)\big)^2\Big]. \tag{68}$$

In this case the target network θ^- corresponds to the same target network that is used by DQN. The TD-error $r_t + \gamma \max_{a\in\mathcal{A}} Q(s_{t+1},a;\theta^-)$ is however only used for learning the V function. When it comes to the Q function we use the same update rule that is presented in Eq. 66 with the only difference being that in this case no Φ^- target network is used. Despite requiring the computation of two different targets for learning, we noticed that DQV-Max did not benefit from using two distinct target networks, therefore its loss function for approximating the Q function is simply:

$$L(\theta) = \mathbb{E}_{\langle s_t,a_t,r_t,s_{t+1}\rangle \sim U(D)}\Big[\big(r_t + \gamma V(s_{t+1};\Phi) - Q(s_t,a_t;\theta)\big)^2\Big]. \tag{69}$$

The pseudocode of both DQV and DQV-Max is presented at the end of this thesis in Algorithm **??** which can be found in Appendix. The pseudocode is an adaptation of a standard DRL training loop which corresponds to what is usually presented within the literature [84]. We just make explicit use of the hyperparameters `total_a` and `c` which ensure that enough actions have been performed by the agent before updating the weights of the target network. We also ensure via the hyperparameter `total_e`, that enough episodes are stored within the memory buffer (which has capacity $\mathcal{N}$) before starting to optimize the neural networks.

7.5 RESULTS

7.5.1 *Global Evaluation*

We evaluate the performance of DQV and DQV-Max on a subset of 15 games coming from the popular `Atari-2600` benchmark [9]. Our newly introduced algorithms are compared against DQN and DDQN. To keep all the comparisons as fair as possible we follow the same experimental setup and evaluation protocol which was used in [84] and [143]. The only difference between DQV and DQV-Max, and DQN and DDQN is the exploration schedule which is used. Differently from the latter two algorithms, which use an epsilon-greedy strategy which has an ϵ starting value of 1.0, DQV and DQV-Max's exploration policy starts with an initial ϵ value of 0.5. All other hyperparameters, ranging from the size of the Experience-Replay memory buffer to the architectures of the neural networks, are kept the same among all algorithms. We refer the reader to the original DQN paper [84] for an in-depth overview of all these hyperparameters. The performance of the algorithms is tested based on the popular `no-op action` evaluation regime. At the end of the training, the learned policies are tested over a series of episodes for a total amount of 5 minutes of emulator time. All testing episodes start by executing a set of partially random actions to test the level of generalization of the learned policies. We present our results in Table 7 where the best performing algorithm is reported in a green cell while the second-best performing algorithm is reported in a yellow cell. As is common within the DRL literature, the table also reports the scores which would be obtained by an expert human player and by a random policy. When the scores over games are equivalent, we report in the green and yellow cells the fastest and second fastest algorithm with respect to its convergence time.

We can start by observing that DQV and DQV-Max successfully master all the environments on which they have been tested, with the only exception being the `Montezuma's Revenge` game. It is well-known that this game requires more sophisticated exploration strategies than the epsilon-greedy one [32], and was also not mastered by DQN and DDQN when these algorithms were introduced. We can also observe that there is no algorithm which performs best on all the tested environments even though, as highlighted by the green and yellow cells, the algorithms of the DQV-family seem to generally perform better than DQN and DDQN, with DQV-Max being the overall best performing algorithm in our set of experiments. When either DQV or DQV-Max are not the best performing algorithm (see for example the `Boxing` and `CrazyClimber` environments), we can still observe that our algorithms managed to converge to a policy which is not significantly worst than the one learned by DQN and DDQN. There is however one exception being the `RoadRunner` environment. In fact, in this game, DDQN significantly outperforms DQV and DQV-Max. It is also worth noting the results on the `BankHeist` and `Enduro` environments. Both DQN and DDQN failed to achieve super-human performance on these games, while DQV and DQV-Max successfully managed to obtain a significantly higher score than the one obtained by a professional human player. On the `BankHeist` environment DQV and DQV-Max obtain ≈ 400 points more than an expert human player, while on the `Enduro` environment their performance is almost three times better than the one obtained by DQN and DDQN.

Table 7: The results obtained by DQV and DQV-Max on a subset of 15 `Atari` games. We can see that our newly introduced algorithms have a comparable, and often even better performance than DQN and DDQN. As highlighted by the green cells the overall best performing algorithm in our set of experiments is DQV-Max while the second-best performing algorithm is DQV (as reported by the yellow cells). Specific attention should be given to the games `BankHeist` and `Enduro` where DQV and DQV-Max are the only algorithms which can master the game with a final super-human performance.

Environment	Random	Human	DQN [84]	DDQN [143]	DQV	DQV-Max
`Asteroids`	719.10	13156.70	1629.33	930.60	1445.40	1846.08
`Bank Heist`	14.20	734.40	429.67	728.30	1236.50	1118.28
`Boxing`	0.10	4.30	71.83	81.70	78.66	80.15
`Crazy Climber`	10780.50	35410.50	114103.33	101874.00	108600.00	1000131.00
`Enduro`	0.00	309.60	301.77	319.50	829.33	875.64
`Fishing Derby`	-91.70	5.50	-0.80	20.30	1.12	20.42
`Frostbite`	65.20	4334.70	328.33	241.50	271.86	281.36
`Gopher`	257.60	2321.00	8520.00	8215.40	8230.30	7940.00
`Ice Hockey`	-11.20	0.90	-1.60	-2.40	-1.88	-1.12
`James Bond`	29.00	406.70	576.67	438.00	372.41	440.80
`Montezuma's Revenge`	0.00	4366.70	0.00	0.00	0.00	0.00
`Ms.Pacman`	307.30	15693.40	2311.00	3210.00	3590.00	3390.00
`Pong`	-20.70	9.30	18.90	21.00	21.00	21.00
`Road Runner`	11.50	7845.00	18256.67	48377.00	39290.00	20700.00
`Zaxxon`	32.50	9173.30	4976.67	10182.00	10950.00	8487.00

7.5.2 *Convergence Time*

While DRL algorithms have certainly obtained impressive results on the `Atari-2600` benchmark, it is also true that the amount of training time which is required by these algorithms can be very long. Over the years, several techniques, ranging from Prioritized Experience Replay (PER) [148] to the Rainbow extensions introduced by Hessel et al. [50], have been proposed to reduce the training time of DRL algorithms. It is therefore natural to investigate whether jointly approximating the V function alongside the Q function can lead to significant benefits in this behalf. Unlike the Q function, the state-value function is not dependent on the set of possible actions that the agent may take, and therefore requires fewer parameters to converge. Since DQV and DQV-Max use the estimates of the V network to train the Q function, it is possible that the Q function could directly benefit from these estimates and as a result converge faster than when regressed towards itself (as happens in DQN).

We use two self-implemented versions of DQN and DDQN for comparing the convergence time that is required during training by all the tested algorithms on three increasingly complex `Atari` games: `Boxing, Pong` and `Enduro`. Our results, reported in Fig. 21, show that DQV and DQV-Max converge significantly faster than DQN and DDQN, therefore confirming the preliminary results which we reported in Fig. **??**, and highlighting once again the benefits of jointly approximating two value functions instead of one when it comes to the overall convergence time that is required by the algorithms. Even though, as presented in Table 7, DQV and DQV-Max do not always significantly outperform DQN and DDQN in terms of the final cumulative reward which is obtained, it is worth noting that these algorithms require significantly less training episodes to converge on all tested games. This benefit makes our two novel algorithms faster alternatives within model-free DRL.

7.5.3 *Quality of the Learned Value Functions*

It is well-known that the combination of RL algorithms with function approximators can yield DRL algorithms that diverge. The popular Q-Learning algorithm is known to result in unstable learning both if linear [141] and non-linear functions are used when approximating the Q function [**van2018deep**]. This divergence according to Sutton and Barto [128] is caused by the interplay of three elements that are known as the *'Deadly Triad'* of DRL. The elements of this triad are:

- *a function approximator*: which is used for learning an approximation of a value function that could not be learned in the tabular RL setting due to a too large state-action space.

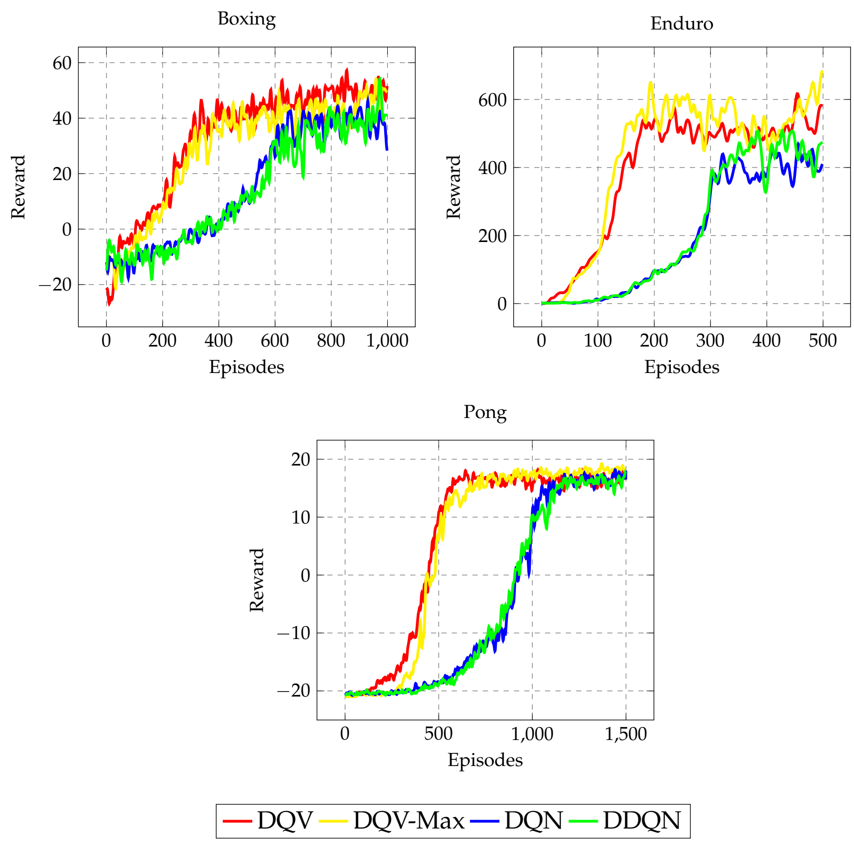


Figure 21: Learning curves obtained during training on three different `Atari` games by DQV and DQV-Max, and DQN and DDQN. We can observe that on these games both DQV and DQV-Max converge significantly faster than DQN and DDQN and that they obtain higher cumulative rewards on the `Enduro` environment.

- *bootstrapping*: when the algorithms use a future estimated value for learning the same kind of estimate.
- *off-policy learning*: when a future estimated value is different from the one which would be computed by the policy the agent is following.

van2018deep have shown that the *'Deadly Triad'* is responsible for enhancing one of the most popular biases that characterize the Q-Learning algorithm: the overestimation bias of the Q function [142]. It is therefore natural to study how DQV and DQV-Max relate to the *'Deadly Triad'* of DRL, and to investigate up to what extent these algorithms suffer from the overestimation bias of the Q function. To do this we monitor the estimates that are given by the network that is responsible for approximating the Q function. More specifically, at training time, we compute the averaged $\max\limits_{a \in \mathcal{A}} Q(s_{t+1}, a)$ over a set (n) of full evaluation episodes as defined by

$$\frac{1}{n}\sum_{t=1}^{n} \max_{a \in \mathcal{A}} Q(s_{t+1}, a; \theta). \tag{70}$$

As suggested by Van Hasselt, Guez, and Silver [143] these estimates can then be compared to the averaged discounted return of all visited states that comes from an agent that has already concluded training. By analyzing whether the Q values which are estimated while training differ from the ones which should be predicted by the end of it, it is possible to quantitatively characterize the level of divergence of DRL algorithms. We report our results in Figs. 22, 23, 24 and 25 where the black full lines correspond to the value estimates that come from each algorithm at training time, while the coloured lines correspond to the actual averaged discounted return that is given by an already trained agent.

We can start by observing that the values denoting the averaged discounted return obtained by each algorithm differ among agents. This is especially the case when it comes to the `Enduro` environment, and is a result which is in line with what has been presented in Table 5: DQV and DQV-Max lead to better final policies than DQN and DDQN. Furthermore, when we compare these baseline values to the value estimates that are obtained during training, we can observe that the ones obtained by the DQN algorithm significantly diverge from the ones which should be predicted by the end of training. This behavior is known to be caused by the overestimation bias of the Q function which can be corrected by the DDQN algorithm. By analyzing the value estimates of DQV and DQV-Max we can observe that both algorithms produce value estimates which are more similar to the ones computed by DDQN than to the ones given by DQN. This is especially the case for DQV, in fact its value estimates nicely correspond to the averaged discounted return baseline, both on the `Pong` environment and on the `Enduro` environment. The estimates coming from DQV-Max, however, seem to diverge more when compared to DQV and DDQN's ones. This is clearer on the `Enduro` environment, where the algorithm does show some divergence. However, we can also observe that this divergence is less strong when compared to DQN's one. The value estimates of the latter algorithm keep growing over time, while DQV-Max's ones get bounded while training progresses. This results in smaller estimated Q values. We believe that there are mainly two reasons why our algorithms suffer less from the overestimation bias of the Q function. When it comes to DQV, we believe that this algorithm suffers less from this bias since it is an *on-policy* learning algorithm. Such algorithms are trained on exploration actions with lower Q values. Because of its *on-policy* learning scheme, DQV also does not present one element of the *'Deadly Triad'*, which might help reducing divergence. When it comes to DQV-Max, we believe that the reason why this algorithm does not diverge as much as DQN can be found in the way it approximates the Q function. One key component of the *'Deadly Triad'*, is that divergence occurs if the Q function is learned by regressing towards itself. As given by Eq. 69

we can see that this does not hold for DQV-Max, since the Q function bootstraps with respect to estimates that come from the V network. We believe that this specific learning dynamic, which also holds for the DQV algorithm, makes our algorithms less prone to estimate large Q values.

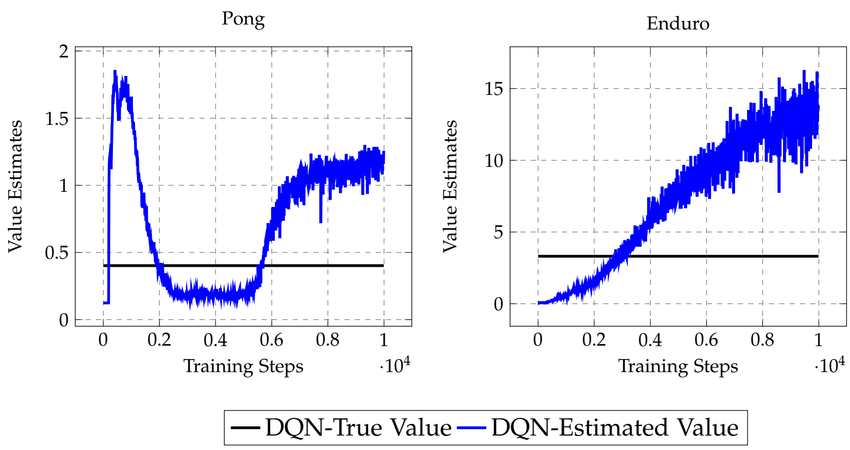


Figure 22: Results investigating the extent to which the DQN algorithm suffers from the overestimation bias of the Q function. We can observe that on the `Pong` environment during the early stages of training, the $\max_{a \in \mathcal{A}} Q(s_{t+1}, a)$ estimates quickly grow, while on the `Enduro` game the values estimated by the Q network keep indefinitely growing, therefore making the algorithm significantly diverge from the real return that is obtained by a trained agent.

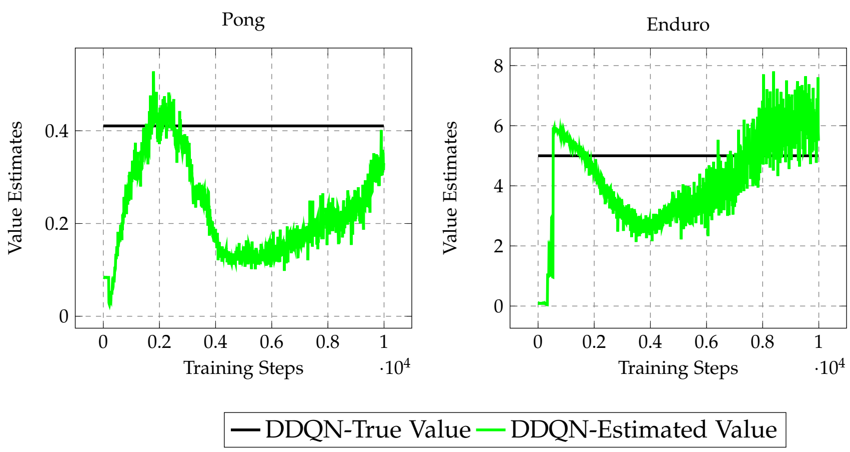


Figure 23: Results investigating the extent to which the DDQN algorithm suffers from the overestimation bias of the Q function. We can observe that compared to the analysis presented in Fig. 22, the DDQN algorithm prevents its Q-Network from diverging since on both `Atari` environments the $\max_{a \in \mathcal{A}} Q(s_{t+1}, a)$ estimates do not diverge from the observed real return of a trained agent. Results that replicate the findings reported by **van2018deep**.

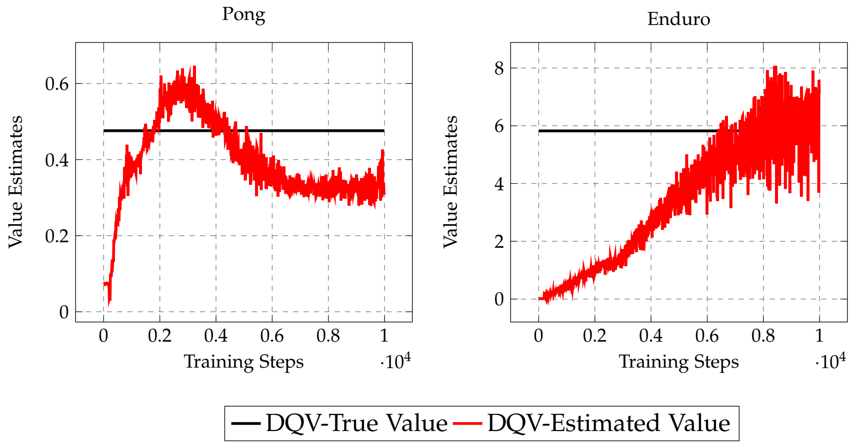


Figure 24: Results investigating the extent to which the DQV algorithm suffers from the overestimation bias of the Q function. We can observe that the performance of the algorithm is similar to the one observed in Fig. 23 for the DDQN algorithm. On both environments the estimated cumulative reward does not diverge from the real return that is obtained by the end of training, therefore suggesting that DQV-Learning does not suffer from the overestimation bias of the Q function. It is also worth noting the difference between the real return obtained by the DQN and the DDQN algorithms on the `Enduro` environment, and the one obtained by DQV. As can be seen by the black line, the real return obtained by a DQV agent is higher than DQN and DDQN's one, a result which shows that DQV converges to a better policy than DQN and DDQN.

7.6 ADDITIONAL STUDIES

As introduced in Sec. 7.4 DQV and DQV-Max use two separate neural networks for approximating the Q function and the V function. To verify whether two different architectures are needed for making both algorithms perform well, we have experimented with a series of variants of the DQV-Learning algorithm. The aim of these experiments is that of reducing the number of trainable parameters that are required by the original version of DQV, and investigate whether its performance could get harmed when reducing the capacity of the algorithm. The studied DQV's extensions are the following:

1. *Hard-DQV*: a version of DQV which uses one single common neural network for approximating both the Q and the V functions. An additional output node, needed for estimating the value of a state, is added next to the output nodes which estimate the different Q values. The parameters of this algorithm are therefore 'hardly-shared' among the agent, and provide the benefit of halving the total amount of trainable parameters of DQV. The different outputs of the network get then alternatively optimized according to Eq. 65 and 66.

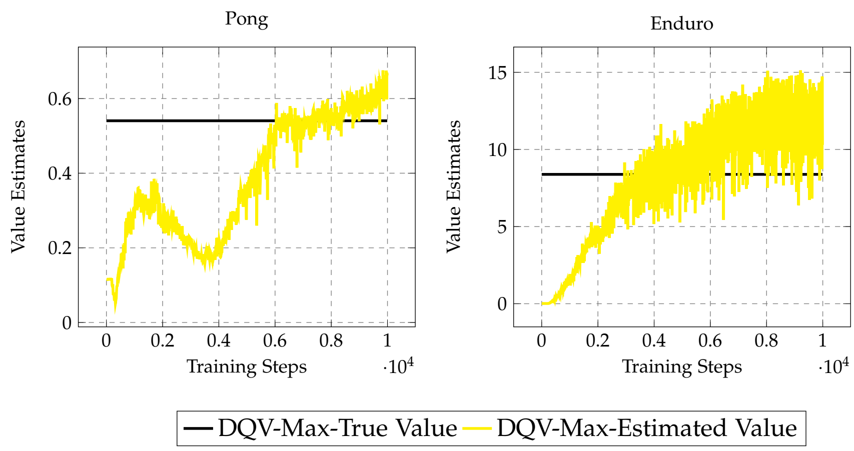


Figure 25: Results investigating the extent to which the DQV-Max algorithm suffers from the overestimation bias of the Q function. We can observe that on the `Pong` environment the value estimates of the algorithm are comparable to the ones of DDQN and DQV, therefore showing the DQV-Max also diverges significantly less than DQN. On the `Enduro` environment we can observe that the algorithm does diverge, although, differently from what is reported in Fig. 22, the $\max_{a \in \mathcal{A}} Q(s_{t+1}, a)$ estimates seem to converge towards an upper bound (≈ 15). Similarly to what is reported in Fig.24 we can again observe that the real return obtained by a trained agent is higher compared to the one obtain by DQN, DDQN and DQV, therefore confirming the results presented in Table 7 which see the DQV-Max algorithm as the best performing algorithm on the `Enduro` game.

2. *Dueling-DQV*: a slightly more complicated version of Hard-DQV which adds one specific hidden layer before the output nodes that estimate the Q and V functions. In this case, the outputs of the neural network which learn one of the two value functions, partly benefit from some specific weights that are not shared within the neural network. This approach is similar to the one used by the 'Dueling-Architecture' presented by Wang et al. [148], therefore the name Dueling-DQV. While it is well established that three convolutional layers are needed [84, 143] for learning the Q function, the same might not be true when it comes to learning the V function. We thus report experiments with three different versions of Dueling-DQV: Dueling-1st, Dueling-2nd, and Dueling-3rd. The difference between these methods is simply the location of the hidden layer which precedes the output that learns the V function. It can be positioned after the first convolutional layer, the second or the third one. Training this architecture is done as for Hard-DQV.

3. *Tiny-DQV*: the neural architectures used by DQV and DQV-Max that approximate the V function and the Q function follow the one which was initially introduced by the DQN algorithm [84].

This corresponds to a three-hidden layer convolutional neural network which is followed by a fully connected layer of 512 hidden units. The first convolutional layer has 32 channels while the last two layers have 64 channels. In Tiny-DQV we reduce the number of trainable parameters of DQV by reducing the number of channels at each convolution operation. Tiny-DQV only uses 8 channels after the first convolutional layer and 16 at the second and third convolutional layers. Furthermore, the size of the final fully connected layer is reduced to only 128 hidden units. The choice of this architecture is motivated by the work presented in [**van2018deep**] which studies the role of the capacity of the DDQN algorithm. Unlike the Hard-DQV and Dueling-DQV extensions, the parameters of Tiny-DQV are not shared at all among the networks that are responsible for approximating the V function and the Q function.

The results obtained by these alternative versions of DQV are presented in Figs. 26, 27 and 28 where we report the learning curves obtained by the tested algorithms on six different `Atari` games. Each DQV extension is directly compared to the original DQV algorithm. We can observe that all the extensions of DQV, which aim at reducing the number of trainable parameters of the algorithm, fail in performing as well as the original DQV algorithm. Starting from Fig.26 we can observe that *Hard-DQV* does not only yield significantly lower rewards (see the results obtained on `Boxing`) but also presents more unstable training (as highlighted by the results obtained on the `Pong` environment). Lower rewards and unstable training also characterize the *Tiny-DQV* algorithm (see results on `BankHeist` and `CrazyClimber` reported in Fig.28). Overall the most promising extensions of DQV are its *Dueling* counterparts, we have observed in particular that the best performing architecture over most of our experiments was the *Dueling-DQV-3rd* one. As can be seen by the results reported in Fig. 27 on the `Pong` environment we can observe that *Dueling-DQV-3rd* has a comparable performance to DQV, even though it converges slower. Unfortunately, *Dueling-DQV-3rd* still shows some limitations, in particular when tested on more complicated environments such as `Enduro`, we can observe that it under-performs DQV with ≈ 200 points. It is also worth mentioning that the idea of approximating the V function before the Q function explored by *Dueling-DQV-1st* and *Dueling-DQV-2nd* yielded negative results.

7.7 DISCUSSION AND CONCLUSION

We have presented two novel model-free DRL algorithms which in addition to learning an approximation of the Q function also aim at learning an approximation of the V function. We have compared DQV and DQV-Max Learning to DRL algorithms which only learn

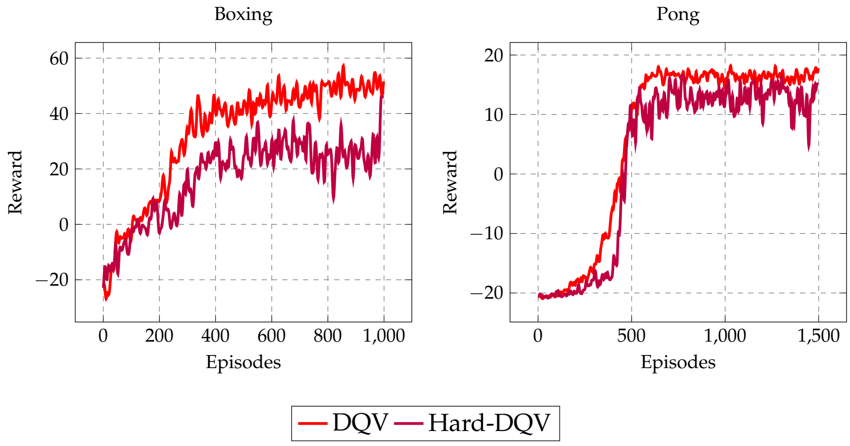


Figure 26: Our results which aim to approximate the V and the Q function with a unique, shared paramterized network, an approach that is heavily inspired by multi-task learning studies that can be found in supervised learning [20, 89, 166]. We can see that this extension of DQV, named Hard-DQV, significantly underperforms the original DQV-Learning algorithm.

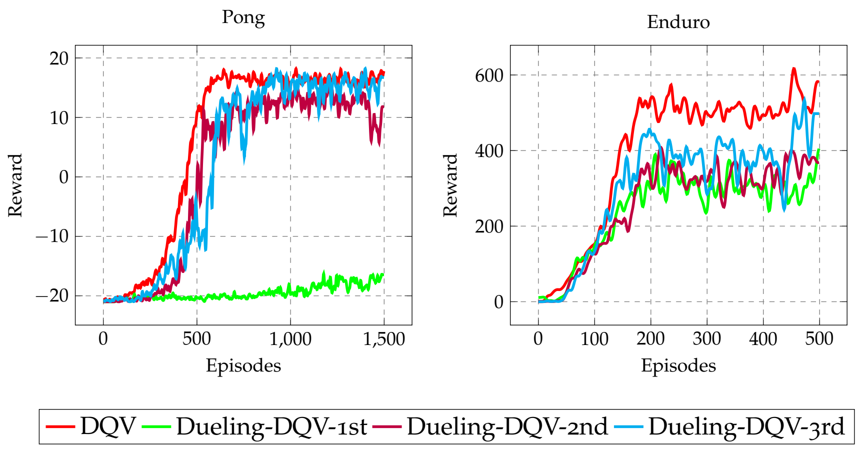


Figure 27: Our extensions of DQV that aim to reduce the amount of trainable parameters of the algorithm by following an approach similar to the one presented by Wang et al. [148] when "Dueling Networks" for DRL have been introduced. We can observe that among all the three Dueling-DQV extensions, only the Dueling-DQV-3rd one yielded good performance, but as highlighted by the results obtained on the `Enduro` environment, its performance is still inferior when compared to the one of the original DQV algorithm.

an approximation of the Q function, and showed the benefits which come from jointly approximating two value functions over one. Our newly introduced algorithms learn significantly faster than DQN and DDQN and show that approximating both the V function and the Q function can yield significant benefits both in an *on-policy* learning setting as in an *off-policy* learning one. This specific training dynamic

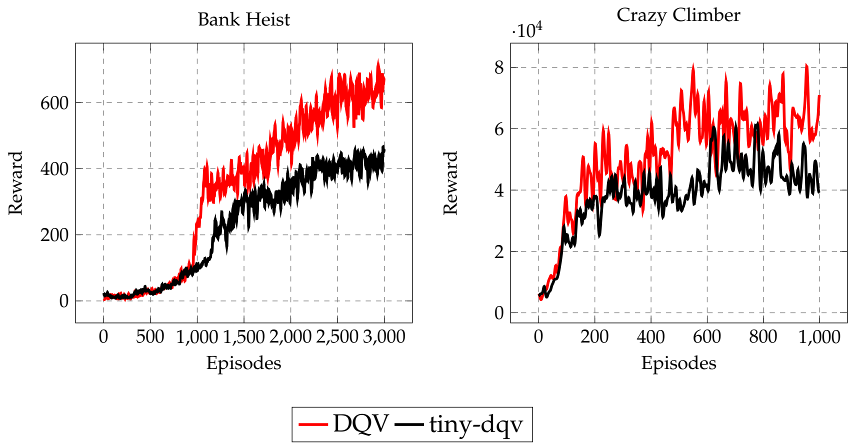


Figure 28: Learning curves obtained when reducing the capacity of the convolutional networks that approximate the V and the Q functions. We can observe that albeit each value function is approximated with its own parametrized network, the tiny-dqv extension still yields worse performance. These results highlight that it is not sufficient to simply have two separate neural networks for DQV to perform well, but that a crucial role in DQV's performance is played by the capacity of the networks that are used as well.

allows for a better learned Q function which makes DQV and DQV-Max less prone to estimate unrealistically large Q values. All these benefits come however at a price: to successfully learn two value functions, two separate neural networks with enough capacity are required.

We identify several directions for further research that focus on the following points: an integration of the algorithms of the DQV-family with all the extensions which have improved the DQN algorithm over the years; an integration of DQV and DQV-Max within an Actor-Critic framework which will allow us to tackle continuous-control problems, and lastly, a study of how the algorithms of the DQV-family will perform in a Batch-DRL setting. It has been shown that DRL algorithms fail when learning from a fixed data set of trajectories instead of dynamically interacting with the environment [39]. This is due to a phenomenon known as extrapolation error. We will investigate to what extent jointly learning two value functions instead of one will cope with this additional DRL bias.

8

ON THE TRANSFERABILITY OF DEEP-Q NETWORKS

9 BRIDGING THE GAP BETWEEN SUPERVISED AND REINFORCEMENT LEARNING

Part IV

APPENDIX

BIBLIOGRAPHY

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. "TensorFlow: A System for Large-Scale Machine Learning." In: *OSDI*. Vol. 16. 2016, pp. 265–283.

[2] Joshua Achiam, Ethan Knight, and Pieter Abbeel. "Towards Characterizing Divergence in Deep Q-Learning." In: *arXiv preprint arXiv:1903.08894* (2019).

[3] Sandro Ackermann, Kevin Schawinksi, Ce Zhang, Anna K Weigel, and M Dennis Turp. "Using transfer learning to detect galaxy mergers." In: *Monthly Notices of the Royal Astronomical Society* (2018).

[4] Nancy Allen. "Collaboration through the Colorado digitization project." In: *First Monday* 5.6 (2000).

[5] Andrew Arnold, Ramesh Nallapati, and William W Cohen. "A comparative study of methods for transductive transfer learning." In: *Seventh IEEE international conference on data mining workshops (ICDMW 2007)*. IEEE. 2007, pp. 77–82.

[6] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. "Deep reinforcement learning: A brief survey." In: *IEEE Signal Processing Magazine* 34.6 (2017), pp. 26–38.

[7] Andrew G Barto, Richard S Sutton, and Charles W Anderson. "Neuronlike adaptive elements that can solve difficult learning control problems." In: *IEEE transactions on systems, man, and cybernetics* 5 (1983), pp. 834–846.

[8] Marc G Bellemare, Will Dabney, and Rémi Munos. "A distributional perspective on reinforcement learning." In: *International Conference on Machine Learning*. PMLR. 2017, pp. 449–458.

[9] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. "The arcade learning environment: An evaluation platform for general agents." In: *Journal of Artificial Intelligence Research* 47 (2013), pp. 253–279.

[10] Richard Bellman. "Dynamic programming." In: *Science* 153.3731 (1966), pp. 34–37.

[11] Dimitri P Bertsekas. "Value and policy iterations in optimal control and adaptive dynamic programming." In: *IEEE transactions on neural networks and learning systems* 28.3 (2015), pp. 500–509.

[12] Dimitri P Bertsekas. *Reinforcement learning and optimal control*. Athena Scientific Belmont, MA, 2019.

[13] Dimitri P Bertsekas and John N Tsitsiklis. "Neuro-dynamic programming: an overview." In: *Proceedings of 1995 34th IEEE conference on decision and control*. Vol. 1. IEEE. 1995, pp. 560–564.

[14] Dimitri P Bertsekas et al. *Dynamic programming and optimal control: Vol. 1*. Athena scientific Belmont, 2000.

[15] Francesco Bidoia, Matthia Sabatelli, Amirhossein Shantia, Marco A. Wiering, and Lambert Schomaker. "A Deep Convolutional Neural Network for Location Recognition and Geometry based Information." In: *Proceedings of the 7th International Conference on Pattern Recognition Applications and Methods, ICPRAM 2018, Funchal, Madeira - Portugal, January 16-18, 2018.* 2018, pp. 27–36.

[16] Mariusz Bojarski, Anna Choromanska, Krzysztof Choromanski, Bernhard Firner, Larry Jackel, Urs Muller, and Karol Zieba. "VisualBackProp: efficient visualization of CNNs." In: *arXiv preprint arXiv:1611.05418* (2016).

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

[18] Lucian Busoniu, Robert Babuska, Bart De Schutter, and Damien Ernst. *Reinforcement learning and dynamic programming using function approximators*. Vol. 39. CRC press, 2010.

[19] Lucian Buşoniu, Damien Ernst, Bart De Schutter, and Robert Babuška. "Approximate reinforcement learning: An overview." In: *2011 IEEE symposium on adaptive dynamic programming and reinforcement learning (ADPRL)*. IEEE. 2011, pp. 1–8.

[20] Rich Caruana. "Multitask learning." In: *Machine learning* 28.1 (1997), pp. 41–75.

[21] Rich Caruana, Steve Lawrence, and C Lee Giles. "Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping." In: *Advances in neural information processing systems*. 2001, pp. 402–408.

[22] François Chollet. "Xception: Deep learning with depthwise separable convolutions." In: *arXiv preprint* (2016).

[23] François Chollet et al. *Keras*. 2015.

[24] Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. "Deep learning for classical japanese literature." In: *arXiv preprint arXiv:1812.01718* (2018).

[25] Wenyuan Dai, Qiang Yang, Gui-Rong Xue, and Yong Yu. "Self-taught clustering." In: *Proceedings of the 25th international conference on Machine learning*. 2008, pp. 200–207.

[26] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "Imagenet: A large-scale hierarchical image database." In: *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE. 2009, pp. 248–255.

[27] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. "Decaf: A deep convolutional activation feature for generic visual recognition." In: *International conference on machine learning*. 2014, pp. 647–655.

[28] Xin Dong, Shangyu Chen, and Sinno Pan. "Learning to prune deep neural networks via layer-wise optimal brain surgeon." In: *Advances in Neural Information Processing Systems*. 2017, pp. 4857–4867.

[29] Stamatatos Efstathios. "A survey of modern authorship attribution methods." In: *Journal of the American Society for Information Science and Technology* 3 (2009), pp. 538–556. DOI: 10.1002/asi.21001.

[30] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. "The pascal visual object classes (voc) challenge." In: *International journal of computer vision* 88.2 (2010), pp. 303–338.

[31] William Fedus, Prajit Ramachandran, Rishabh Agarwal, Yoshua Bengio, Hugo Larochelle, Mark Rowland, and Will Dabney. "Revisiting fundamentals of experience replay." In: *International Conference on Machine Learning*. PMLR. 2020, pp. 3061–3071.

[32] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, et al. "Noisy networks for exploration." In: *arXiv preprint arXiv:1706.10295* (2017).

[33] Vincent François-Lavet, Raphaël Fonteneau, and Damien Ernst. "How to Discount Deep Reinforcement Learning: Towards New Dynamic Strategies." In: *NIPS 2015 Workshop on Deep Reinforcement Learning*. 2015.

[34] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G Bellemare, and Joelle Pineau. "An introduction to deep reinforcement learning." In: *arXiv preprint arXiv:1811.12560* (2018).

[35] Jonathan Frankle and Michael Carbin. "The lottery ticket hypothesis: Finding sparse, trainable neural networks." In: *arXiv preprint arXiv:1803.03635* (2018).

[36] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. "Linear mode connectivity and the lottery ticket hypothesis." In: *arXiv preprint arXiv:1912.05671* (2019).

[37] Jonathan Frankle, G Karolina Dziugaite, DM Roy, and M Carbin. "Stabilizing the Lottery Ticket Hypothesis." In: *arXiv preprint arXiv:1903.01611* (2019).

[38] Scott Fujimoto, Herke Hoof, and David Meger. "Addressing function approximation error in actor-critic methods." In: *International Conference on Machine Learning*. PMLR. 2018, pp. 1587–1596.

[39] Scott Fujimoto, Edoardo Conti, Mohammad Ghavamzadeh, and Joelle Pineau. "Benchmarking Batch Deep Reinforcement Learning Algorithms." In: *arXiv preprint arXiv:1910.01708* (2019).

[40] Matthieu Geist, Bruno Scherrer, et al. "Off-policy learning with eligibility traces: a survey." In: *J. Mach. Learn. Res.* 15.1 (2014), pp. 289–333.

[41] Varun Gohil, S Deepak Narayanan, and Atishay Jain. "One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers." In: *ReScience-C* (2020).

[42] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor." In: *International Conference on Machine Learning*. PMLR. 2018, pp. 1861–1870.

[43] Song Han, Huizi Mao, and William J Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding." In: *arXiv preprint arXiv:1510.00149* (2015).

[44] Hado Philip van Hasselt. *Insights in reinforcement rearning: formal analysis and empirical evaluation of temporal-difference learning algorithms*. Utrecht University, 2011.

[45] Ji He, Jianshu Chen, Xiaodong He, Jianfeng Gao, Lihong Li, Li Deng, and Mari Ostendorf. "Deep reinforcement learning with a natural language action space." In: *arXiv preprint arXiv:1511.04636* (2015).

[46] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification." In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[47] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[48] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. "Deep reinforcement learning that matters." In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[49] J Fernando Hernandez-Garcia and Richard S Sutton. "Understanding Multi-Step Deep Reinforcement Learning: A Systematic Study of the DQN Target." In: *arXiv preprint arXiv:1901.07510* (2019).

[50] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. "Rainbow: Combining improvements in deep reinforcement learning." In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[51] Matteo Hessel, Hado van Hasselt, Joseph Modayil, and David Silver. "On inductive biases in deep reinforcement learning." In: *arXiv preprint arXiv:1907.02908* (2019).

[52] Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. "Densely connected convolutional networks." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. Vol. 1. 2. 2017.

[53] Gary B Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller. *Labeled faces in the wild: A database for studying face recognition in unconstrained environments*. Tech. rep. Technical Report 07-49, University of Massachusetts, Amherst, 2007.

[54] Tommi Jaakkola, Satinder P Singh, and Michael I Jordan. "Reinforcement learning algorithm for partially observable Markov decision problems." In: *Advances in neural information processing systems* (1995), pp. 345–352.

[55] Ou Jin, Nathan N Liu, Kai Zhao, Yong Yu, and Qiang Yang. "Transferring topical knowledge from auxiliary long texts for short text clustering." In: *Proceedings of the 20th ACM international conference on Information and knowledge management*. 2011, pp. 775–784.

[56] Philipp Kainz, Harald Burgsteiner, Martin Asslaber, and Helmut Ahammer. "Training echo state networks for rotation-invariant bone marrow cell classification." In: *Neural Computing and Applications* 28.6 (2017), pp. 1277–1292.

[57] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. "Model-based reinforcement learning for atari." In: *arXiv preprint arXiv:1903.00374* (2019).

[58] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. "Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation." In: *arXiv preprint arXiv:1806.10293* (2018).

[59] Simon Kornblith, Jonathon Shlens, and Quoc V Le. "Do Better ImageNet Models Transfer Better?" In: *arXiv preprint arXiv:1805.08974* (2018).

[60] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks." In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[61] Stephen H Lane, David A Handelman, and Jack J Gelfand. "Theory and development of higher-order CMAC neural networks." In: *IEEE Control Systems Magazine* 12.2 (1992), pp. 23–30.

[62] Pat Langley. "Transfer of knowledge in cognitive systems." In: *Talk, workshop on Structural Knowledge Transfer for Machine Learning at the Twenty-Third International Conference on Machine Learning*. 2006.

[63] Romain Laroche and Merwan Barlier. "Transfer reinforcement learning with shared dynamics." In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 31. 1. 2017.

[64] Alessandro Lazaric. "Transfer in reinforcement learning: a framework and a survey." In: *Reinforcement Learning*. Springer, 2012, pp. 143–173.

[65] Alessandro Lazaric, Marcello Restelli, and Andrea Bonarini. "Reinforcement learning in continuous action spaces through sequential monte carlo methods." In: *Advances in neural information processing systems* 20 (2007), pp. 833–840.

[66] Alessandro Lazaric, Marcello Restelli, and Andrea Bonarini. "Transfer of samples in batch reinforcement learning." In: *Proceedings of the 25th international conference on Machine learning*. 2008, pp. 544–551.

[67] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." In: *nature* 521.7553 (2015), p. 436.

[68] Donghun Lee, Boris Defourny, and Warren B Powell. "Bias-corrected q-learning to control max-operator bias in q-learning." In: *2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*. IEEE. 2013, pp. 93–99.

[69] Donghun Lee and Warren B Powell. "Bias-corrected Q-learning with multistate extension." In: *IEEE Transactions on Automatic Control* 64.10 (2019), pp. 4011–4023.

[70] Yuxi Li. "Deep reinforcement learning: An overview." In: *arXiv preprint arXiv:1701.07274* (2017).

[71] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. "Continuous control with deep reinforcement learning." In: *arXiv preprint arXiv:1509.02971* (2015).

[72] Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. "Runtime neural pruning." In: *Advances in Neural Information Processing Systems*. 2017, pp. 2181–2191.

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

[74] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. "Microsoft coco: Common objects in context." In: *European conference on computer vision*. Springer. 2014, pp. 740–755.

[75] Jun S Liu and Rong Chen. "Sequential Monte Carlo methods for dynamic systems." In: *Journal of the American statistical association* 93.443 (1998), pp. 1032–1044.

[76] Libin Liu and Jessica Hodgins. "Learning to schedule control fragments for physics-based characters using deep q-learning." In: *ACM Transactions on Graphics (TOG)* 36.3 (2017), pp. 1–14.

[77] Lin Ma, Zhengdong Lu, Lifeng Shang, and Hang Li. "Multimodal convolutional neural networks for matching image and sentence." In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 2623–2631.

[78] Raphaël Marée, Loïc Rollus, Benjamin Stévens, Renaud Hoyoux, Gilles Louppe, Rémy Vandaele, Jean-Michel Begon, Philipp Kainz, Pierre Geurts, and Louis Wehenkel. "Collaborative analysis of multi-gigapixel imaging data using Cytomine." In: *Bioinformatics* 32.9 (2016), pp. 1395–1401.

[79] Dominic Masters and Carlo Luschi. "Revisiting Small Batch Training for Deep Neural Networks." In: *arXiv preprint arXiv:1804.07612* (2018).

[80] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity." In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[81] Rahul Mehta. "Sparse Transfer Learning via Winning Lottery Tickets." In: *arXiv preprint arXiv:1905.07785* (2019).

[82] Thomas Mensink and Jan Van Gemert. "The rijksmuseum challenge: Museum-centered visual recognition." In: *Proceedings of International Conference on Multimedia Retrieval*. 2014, pp. 451–454.

[83] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. "Playing atari with deep reinforcement learning." In: *arXiv preprint arXiv:1312.5602* (2013).

[84] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. "Human-level control through deep reinforcement learning." In: *Nature* 518.7540 (2015), p. 529.

[85] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. "Asynchronous methods for deep reinforcement learning." In: *International conference on machine learning*. 2016, pp. 1928–1937.

[86] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. "Pruning convolutional neural networks for resource efficient inference." In: *arXiv preprint arXiv:1611.06440* (2016).

[87] Ari Morcos, Haonan Yu, Michela Paganini, and Yuandong Tian. "One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers." In: *Advances in Neural Information Processing Systems*. 2019, pp. 4933–4943.

[88] Romain Mormont, Pierre Geurts, and Raphaël Marée. "Comparison of deep transfer learning strategies for digital pathology." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2018, pp. 2262–2271.

[89] Romain Mormont, Pierre Geurts, and Raphaël Marée. "Multitask pre-training of deep neural networks for digital pathology." In: *IEEE journal of biomedical and health informatics* (2020).

[90] Karthik Narasimhan, Tejas Kulkarni, and Regina Barzilay. "Language understanding for text-based games using deep reinforcement learning." In: *arXiv preprint arXiv:1506.08941* (2015).

[91] Johan S Obando-Ceron and Pablo Samuel Castro. "Revisiting Rainbow: Promoting more insightful and inclusive deep reinforcement learning research." In: *arXiv preprint arXiv:2011.14826* (2020).

[92] Sinno Jialin Pan and Qiang Yang. "A survey on transfer learning." In: *IEEE Transactions on knowledge and data engineering* 22.10 (2009), pp. 1345–1359.

[93] Sinno Jialin Pan and Qiang Yang. "A survey on transfer learning." In: *IEEE Transactions on knowledge and data engineering* (2010), pp. 1345–1359.

[94] Jooyoung Park and Irwin W Sandberg. "Approximation and radial-basis-function networks." In: *Neural computation* 5.2 (1993), pp. 305–316.

[95] Ross Parry. "Digital heritage and the rise of theory in museum computing." In: *Museum management and Curatorship* (2005), pp. 333–348.

[96] Jing Peng and Ronald J Williams. "Incremental multi-step Q-learning." In: *Machine Learning Proceedings 1994*. Elsevier, 1994, pp. 226–232.

[97] Andreas Pentaliotis and Marco Wiering. "Variation-resistant Q-learning: Controlling and Utilizing Estimation Bias in Reinforcement Learning for Better Performance." In: (2021).

[98] Fred Phillips and Brandy Mackintosh. "Wiki Art Gallery, Inc.: A case for critical thinking." In: *Issues in Accounting Education* 26.3 (2011), pp. 593–608.

[99] Martin L Puterman. "Markov decision processes." In: *Handbooks in operations research and management science* 2 (1990), pp. 331–434.

[100] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[101] Pengjiang Qian, Yizhang Jiang, Zhaohong Deng, Lingzhi Hu, Shouwei Sun, Shitong Wang, and Raymond F Muzic. "Cluster prototypes and fuzzy memberships jointly leveraged cross-domain maximum entropy clustering." In: *IEEE transactions on cybernetics* 46.1 (2015), pp. 181–193.

[102] Aniruddh Raghu, Matthieu Komorowski, Leo Anthony Celi, Peter Szolovits, and Marzyeh Ghassemi. "Continuous state-space models for optimal sepsis treatment: a deep reinforcement learning approach." In: *Machine Learning for Healthcare Conference*. PMLR. 2017, pp. 147–163.

[103] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. "CNN features off-the-shelf: an astounding baseline for recognition." In: *Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on*. IEEE. 2014, pp. 512–519.

[104] Angie K Reyes, Juan C Caicedo, and Jorge E Camargo. "Fine-tuning Deep Convolutional Networks for Plant Recognition." In: *CLEF (Working Notes)*. 2015.

[105] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*. Vol. 37. University of Cambridge, Department of Engineering Cambridge, England, 1994.

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

[107] Matthia Sabatelli. *Learning to Play Chess with Minimal Lookahead and Deep Value Neural Networks*. Tech. rep. Faculty of Science and Engineering, 2017.

[108] Matthia Sabatelli, Mike Kestemont, and Pierre Geurts. "On the transferability of winning tickets in non-natural image datasets." In: *arXiv preprint arXiv:2005.05232* (2020).

[109] Matthia Sabatelli, Gilles Louppe, Pierre Geurts, and Marco Wiering. "Deep Quality-Value (DQV) Learning." In: *Advances in Neural Information Processing Systems, Deep Reinforcement Learning Workshop*. Montreal. 2018.

[110] Matthia Sabatelli, Mike Kestemont, Walter Daelemans, and Pierre Geurts. "Deep transfer learning for art classification problems." In: *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*. 2018, pp. 631–646.

[111] Matthia Sabatelli, Gilles Louppe, Pierre Geurts, and Marco A Wiering. "The deep quality-value family of deep reinforcement learning algorithms." In: *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2020, pp. 1–8.

[112] Iman Sajedian, Heon Lee, and Junsuk Rho. "Design of high transmission color filters for solar cells directed by deep Q-learning." In: *Solar Energy* 195 (2020), pp. 670–676.

[113] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. "Prioritized experience replay." In: *arXiv preprint arXiv:1511.05952* (2015).

[114] Juergen Schmidhuber. "Reinforcement Learning Upside Down: Don't Predict Rewards–Just Map Them to Actions." In: *arXiv preprint arXiv:1912.02875* (2019).

[115] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. "High-dimensional continuous control using generalized advantage estimation." In: *arXiv preprint arXiv:1506.02438* (2015).

[116] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. "Trust region policy optimization." In: *International conference on machine learning*. PMLR. 2015, pp. 1889–1897.

[117] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal policy optimization algorithms." In: *arXiv preprint arXiv:1707.06347* (2017).

[118] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." In: *arXiv preprint arXiv:1409.1556* (2014).

[119] Satinder Singh, Tommi Jaakkola, Michael L Littman, and Csaba Szepesvári. "Convergence results for single-step on-policy reinforcement-learning algorithms." In: *Machine learning* 38.3 (2000), pp. 287–308.

[120] James E Smith and Robert L Winkler. "The optimizer's curse: Skepticism and postdecision surprise in decision analysis." In: *Management Science* 52.3 (2006), pp. 311–322.

[121] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. "The German traffic sign recognition benchmark: a multi-class classification competition." In: *Neural Networks (IJCNN), The 2011 International Joint Conference on*. IEEE. 2011, pp. 1453–1460.

[122] Eric Van den Steen. "Rational overoptimism (and other biases)." In: *American Economic Review* 94.4 (2004), pp. 1141–1151.

[123] Gjorgji Strezoski and Marcel Worring. "Omniart: multi-task deep learning for artistic data analysis." In: *arXiv preprint arXiv:1708.00684* (2017).

[124] Gjorgji Strezoski and Marcel Worring. "Omniart: a large-scale artistic benchmark." In: *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 14.4 (2018), pp. 1–21.

[125] Tianxiang Sun, Yunfan Shao, Xiaonan Li, Pengfei Liu, Hang Yan, Xipeng Qiu, and Xuanjing Huang. "Learning Sparse Sharing Architectures for Multiple Tasks." In: *arXiv preprint arXiv:1911.05034* (2019).

[126] Richard S Sutton. "Learning to predict by the methods of temporal differences." In: *Machine learning* 3.1 (1988), pp. 9–44.

[127] Richard S Sutton. "Generalization in reinforcement learning: Successful examples using sparse coarse coding." In: *Advances in neural information processing systems*. 1996, pp. 1038–1044.

[128] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[129] Richard S Sutton, David A McAllester, Satinder P Singh, Yishay Mansour, et al. "Policy gradient methods for reinforcement learning with function approximation." In: *NIPs*. Vol. 99. Citeseer. 1999, pp. 1057–1063.

[130] Richard Stuart Sutton. "Temporal credit assignment in reinforcement learning." In: (1984).

[131] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. "Rethinking the inception architecture for computer vision." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 2818–2826.

[132] Nima Tajbakhsh, Jae Y Shin, Suryakanth R Gurudu, R Todd Hurst, Christopher B Kendall, Michael B Gotway, and Jianming Liang. "Convolutional neural networks for medical image analysis: Full training or fine tuning?" In: *IEEE transactions on medical imaging* (2016), pp. 1299–1312.

[133] Matthew E Taylor, Nicholas K Jong, and Peter Stone. "Transferring instances for model-based reinforcement learning." In: *Joint European conference on machine learning and knowledge discovery in databases*. Springer. 2008, pp. 488–505.

[134] Gerald Tesauro. "TD-Gammon, a self-teaching backgammon program, achieves master-level play." In: *Neural computation* 6.2 (1994), pp. 215–219.

[135] Sebastian Thrun and Anton Schwartz. "Issues in using function approximation for reinforcement learning." In: *Proceedings of the Fourth Connectionist Models Summer School*. Hillsdale, NJ. 1993, pp. 255–263.

[136] Tijmen Tieleman and Geoffrey Hinton. "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude." In: *COURSERA: Neural networks for machine learning* (2012), pp. 26–31.

[137] Andrea Tirinzoni, Rafael Rodriguez Sanchez, and Marcello Restelli. "Transfer of value functions via variational methods." In: *Advances in Neural Information Processing Systems*. 2018, pp. 6179–6189.

[138] Andrea Tirinzoni, Andrea Sessa, Matteo Pirotta, and Marcello Restelli. "Importance weighted transfer of samples in reinforcement learning." In: *International Conference on Machine Learning*. PMLR. 2018, pp. 4936–4945.

[139] Denis Tomè, Federico Monti, Luca Baroffio, Luca Bondi, Marco Tagliasacchi, and Stefano Tubaro. "Deep convolutional neural networks for pedestrian detection." In: *Signal Processing: Image Communication* (2016), pp. 482–489.

[140] Huan-Hsin Tseng, Yi Luo, Sunan Cui, Jen-Tzung Chien, Randall K Ten Haken, and Issam El Naqa. "Deep reinforcement learning for automated radiation adaptation in lung cancer." In: *Medical physics* 44.12 (2017), pp. 6690–6705.

[141] John N Tsitsiklis and Benjamin Van Roy. "An analysis of temporal-difference learning with function approximation." In: *IEEE transactions on automatic control* 42.5 (1997), pp. 674–690.

[142] Hado Van Hasselt. "Double Q-learning." In: *Advances in Neural Information Processing Systems*. 2010, pp. 2613–2621.

[143] Hado Van Hasselt, Arthur Guez, and David Silver. "Deep reinforcement learning with double Q-learning." In: *Thirtieth AAAI Conference on Artificial Intelligence*. 2016.

[144] Hado Van Hasselt, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil. "Deep Reinforcement Learning and the Deadly Triad." In: *arXiv preprint arXiv:1812.02648* (2018).

[145] Harm Van Seijen, Mehdi Fatemi, and Arash Tavakoli. "Using a logarithmic mapping to enable lower discount factors in reinforcement learning." In: *arXiv preprint arXiv:1906.00572* (2019).

[146] Ryan Van Soelen and John W Sheppard. "Using winning lottery tickets in transfer learning for convolutional neural networks." In: *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2019, pp. 1–8.

[147] Zheng Wang, Yangqiu Song, and Changshui Zhang. "Transferred dimensionality reduction." In: *Joint European conference on machine learning and knowledge discovery in databases*. Springer. 2008, pp. 550–565.

[148] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando Freitas. "Dueling Network Architectures for Deep Reinforcement Learning." In: *International Conference on Machine Learning*. 2016, pp. 1995–2003.

[149] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. "Sample efficient actor-critic with experience replay." In: *arXiv preprint arXiv:1611.01224* (2016).

[150] Christopher JCH Watkins and Peter Dayan. "Q-learning." In: *Machine learning* 8.3-4 (1992), pp. 279–292.

[151] Qinglai Wei, Derong Liu, and Hanquan Lin. "Value iteration adaptive dynamic programming for optimal control of discrete-time nonlinear systems." In: *IEEE Transactions on cybernetics* 46.3 (2015), pp. 840–853.

[152] Stuart Weibel, John Kunze, Carl Lagoze, and Misha Wolf. *Dublin core metadata for resource discovery*. Tech. rep. 1998.

[153] Marco A Wiering. "Explorations in efficient reinforcement learning." PhD thesis. University of Amsterdam, 1999.

[154] Marco A Wiering. "Convergence and divergence in standard and averaging reinforcement learning." In: *European Conference on Machine Learning*. Springer. 2004, pp. 477–488.

[155] Marco A Wiering. "QV (lambda)-learning: A new on-policy reinforcement learning algorithm." In: *Proceedings of the 7th European Workshop on Reinforcement Learning*. 2005, pp. 17–18.

[156] Marco A Wiering and Hado Van Hasselt. "The QV family compared to other reinforcement learning algorithms." In: *Adaptive Dynamic Programming and Reinforcement Learning, 2009. ADPRL'09. IEEE Symposium on*. IEEE. 2009, pp. 101–108.

[157] Marco Wiering and Jürgen Schmidhuber. "Speeding up Q (λ)-learning." In: *European Conference on Machine Learning*. Springer. 1998, pp. 352–363.

[158] Ronald J Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning." In: *Machine learning* 8.3-4 (1992), pp. 229–256.

[159] Jos van de Wolfshaar, Mahir F Karaaba, and Marco A Wiering. "Deep convolutional neural networks and support vector machines for gender recognition." In: *Computational Intelligence, 2015 IEEE Symposium Series on*. IEEE. 2015, pp. 188–195.

[160] R. Wollheim. *On Art and the Mind. Essays and Lectures*. Allen Lane, 1972.

[161] Han Xiao, Kashif Rasul, and Roland Vollgraf. "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms." In: *arXiv preprint arXiv:1708.07747* (2017).

[162] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. "Aggregated residual transformations for deep neural networks." In: *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*. IEEE. 2017, pp. 5987–5995.

[163] Corinna Cortes Yann LeCun and Christopher J. C. Burges. *The MNIST database of handwritten digits*. http://yann.lecun.com/exdb/mnist/, 1994. 1996.

[164] Haonan Yu, Sergey Edunov, Yuandong Tian, and Ari S Morcos. "Playing the lottery with rewards and multiple languages: lottery tickets in RL and NLP." In: *arXiv preprint arXiv:1906.02768* (2019).

[165] Dongbin Zhao, Haitao Wang, Kun Shao, and Yuanheng Zhu. "Deep reinforcement learning with experience replay based on SARSA." In: *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE. 2016, pp. 1–6.

[166] Sheng-hua Zhong, Xingsheng Huang, and Zhijiao Xiao. "Fine-art painting classification via two-channel dual path networks." In: *International Journal of Machine Learning and Cybernetics* 11.1 (2020), pp. 137–152.

[167] Rong Zhu and Mattia Rigotti. "Self-correcting Q-Learning." In: *arXiv preprint arXiv:2012.01100* (2020).

[168] Xiaofeng Zhu, Zi Huang, Yang Yang, Heng Tao Shen, Changsheng Xu, and Jiebo Luo. "Self-taught dimensionality reduction on the high-dimensional small-sized data." In: *Pattern Recognition* 46.1 (2013), pp. 215–229.

[169] Xiaofeng Zhu, Xuelong Li, Shichao Zhang, Chunhua Ju, and Xindong Wu. "Robust joint graph sparse coding for unsupervised spectral feature selection." In: *IEEE transactions on neural networks and learning systems* 28.6 (2016), pp. 1263–1275.

[170] Zhuangdi Zhu, Kaixiang Lin, and Jiayu Zhou. "Transfer Learning in Deep Reinforcement Learning: A Survey." In: *arXiv preprint arXiv:2009.07888* (2020).

[171] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. "A comprehensive survey on transfer learning." In: *Proceedings of the IEEE* 109.1 (2020), pp. 43–76.

[172] Zhuangwei Zhuang, Mingkui Tan, Bohan Zhuang, Jing Liu, Yong Guo, Qingyao Wu, Junzhou Huang, and Jinhui Zhu. "Discrimination-aware channel pruning for deep neural networks." In: *Advances in Neural Information Processing Systems*. 2018, pp. 875–886.

DECLARATION

Put your declaration here.

Saarbrücken, September 2015

Matthia Sabatelli

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

https://bitbucket.org/amiede/classicthesis/

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

http://postcards.miede.de/

Final Version as of May 10, 2021 (`classicthesis` version 4.2).