

KNOWLEDGE REUSE FOR DEEP REINFORCEMENT LEARNING

Ruben Glatt

Universidade de São Paulo

June 2019

Thesis submitted to Escola Politécnica da Universidade de São Paulo in
fulfillment of the requirements for the degree of
Doctor of Science.

Research Area: Computer Engineering

Advisor: **Prof. Dr. Anna Helena Reali Costa**

– Revision January 2022: New format, Updated references, English only –

Abstract

With the rise of *Deep Learning* the field of *Artificial Intelligence (AI) Research* has entered a new era. Together with an increasing amount of data and vastly improved computing capabilities, *Machine Learning* builds the backbone of *AI*, providing many of the tools and algorithms that drive development and applications. While we have already achieved many successes in the fields of image recognition, language processing, recommendation engines, robotics, or autonomous systems, most progress was achieved when the algorithms were focused on learning only a single task with little regard to effort and reusability. Since learning a new task from scratch often involves an expensive learning process, in this work, we are considering the use of previously acquired knowledge to speed up the learning of a new task. For that, we investigated the application of *Transfer Learning* methods for *Deep Reinforcement Learning (DRL)* agents and propose a novel framework for knowledge preservation and reuse. We show, that the knowledge transfer can make a big difference if the source knowledge is chosen carefully in a systematic approach. To get to this point, we provide an overview of existing literature of methods that realize *knowledge transfer* for *DRL*, a field which has been starting to appear frequently in the relevant literature only in the last two years. We then formulate the *Case-based Reasoning* methodology, which describes a framework for knowledge reuse in general terms, in *Reinforcement Learning* terminology to facilitate the adaption and communication between the respective communities. Building on this framework, we propose *Deep Case-based Policy Inference (DECAF)* and demonstrate in an experimental evaluation the usefulness of our approach for sequential task learning with knowledge preservation and reuse. Our results highlight the benefits of knowledge transfer while also making aware of the challenges that come with it. We consider the work in this area as an important step towards more stable general learning agents that are capable of dealing with the most complex tasks, which would be a key achievement towards *Artificial General Intelligence*.

Keywords: Artificial Intelligence. Machine Learning. Deep Reinforcement Learning. Case-based Reasoning. Transfer Learning.

*"How can you learn from
your mistakes if you don't re-
member them?"*

**Bernard, Westworld, Sea-
son 1 Episode 10, min 48**

List of Figures

2.1	A standard <i>Reinforcement Learning</i> setup. After performing an action, an agent receives a reward and an update of the current state from the environment. Source: Adapted from Sutton and Barto [2018].	7
2.2	Sketch of the architecture of the original <i>DQN</i> with three convolutional layers and two fully connected layers, where the nodes in the highest layer each represent a <i>state-action</i> value for all available actions. Source: Adapted from Mnih et al. [2015].	12
2.3	Potential benefits of knowledge transfer: <i>Jumpstart</i> (1), <i>Time to Threshold</i> (2), and <i>Asymptotic Performance</i> (3). Source: Adapted from Taylor and Stone [2009].	15
2.4	Parameter transfer can be applied to only part (left) or all layers (right) of a previously trained network on a source task, and is commonly used as initialization when training the network of the target task. Source: Author.	18
2.5	In <i>Sample transfer</i> many workers are generating independent samples by interacting with the environment under different pretexts which are then used to stabilize training of the target task. Source: Author.	19
2.6	<i>Partial solutions</i> are used to directly guide the training of the target task. Source: Author.	20
2.7	In <i>Curriculum Learning</i> , an agent progressively trains on more complex challenges with respect to either the task at hand or the agent representations. Source: Author.	20
2.8	<i>Self-play</i> uses an agents own policy to play against a copy of itself, updating the opponent with the current policy from time to time. Source: Author.	21

2.9	<i>Meta solutions</i> describe approaches where previously learned tasks are combined to a meta solution, which is then used to initialize the target task or to guide the target task training. Source: Author.	22
3.1	A view on the <i>CBR</i> cycle adapted to <i>RL</i> terminology. Source: Author.	27
3.2	Sketch of network architecture that produces the guiding training policy $\pi_{train}(s, \cdot)$ for a given input state s : (1) Selected source networks $Q_{\Omega_\theta}(s, \cdot)$ from \mathcal{L}_{train} , (2) network $Q_\Omega(s, a; \theta_\Omega)$ that we want to learn, and (3) importance network $\mathbf{w}(s, a; \theta_w)$. Source: Author.	34
4.1	The games for which policies were trained: (a) Atlantis, (b) Boxing, and (c) Breakout. Source: Screenshot from Bellemare et al. [2013].	40
4.2	Comparison between different settings for a <i>DQN</i> trained for the game Breakout. Source: Author.	43
4.3	Comparison between random initialization of Breakout and initialization with the best performing network for a previously trained <i>DQN</i> for Breakout. Source: Author.	44
4.4	Comparison between random initialization of Breakout and initialization with the best performing network for a previously trained <i>DQN</i> for Atlantis. Source: Author.	45
4.5	Comparison between random initialization of Breakout and initialization with the best performing network for a previously trained <i>DQN</i> for Boxing. Source: Author.	45
4.6	Resulting rewards (right) when using a policy library that also contains policies from very similar tasks, Ω_1 and Ω_5 (left). The red square represents the goal position in the target task. Source: Author.	48
4.7	Resulting rewards (right) when using a library only containing policies from very unrelated tasks, Ω_2 , Ω_3 , and Ω_4 (left). The red square represents the goal position in the target task. Source: Author.	49
4.8	The same grid world domain was evaluated for the library building experiment. The graphs show all 50 goal positions that were learned during the experiment (left) and the extracted core policies using <i>CBPI</i> (middle) and <i>PLPR</i> (right), respectively. Source: Author.	50

4.9	Tasks contained in the knowledge base of experiment 1 are <i>Pong</i> (left) and <i>SpaceInvaders</i> (right). Source: Screenshot from Bellemare et al. [2013].	54
4.10	Resulting rewards using different pretrained agents. The graph shows average and confidence intervals of the performance of 6 experiments per setting with 10 evaluation periods per era and setting. Source: Author.	55
4.11	Tasks contained in the knowledge base of experiment 2 from left to right, upper row: <i>Pong</i> with obscured areas (upper half, middle, lower half); lower row: <i>SpaceInvaders</i> , <i>Pooyan</i> , and <i>Qbert</i> . Source: Screenshot from Bellemare et al. [2013].	56
4.12	Resulting rewards using different tasks from the policy library. The upper graph shows the performance during training periods and the lower graph shows the performance during evaluation periods. Source: Author.	58

List of Abbreviations

A2T	Attend, Adapt and Transfer
AI	Artificial Intelligence
CBPI	Case-based Policy Inference
CBR	Case-based Reasoning
CRL	Case-based Reinforcement Learning
DECAF	Deep Case-based Policy Inference
DL	Deep Learning
DNN	Deep Neural Network
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
MDP	Markov Decision Process
ML	Machine Learning
NN	Neural Network
PLPR	Policy Library Policy Reuse
RL	Reinforcement Learning
TD	Temporal Difference
TL	Transfer Learning

List of Symbols

α	Learning rate
A	Action space for Markov Decision Processes
a	Action
a'	Follow-up action
a_t	Action at time step t
\mathcal{D}	Domain
\mathcal{D}^Ω	Domain of a specific task Ω
e	Experiences in the form of a tuple $\langle s, a, r, s' \rangle$
ε	Probability for choosing a random action a
γ	Discount factor
L	Loss function
l	Layer number in a Deep Neural Network
l_{max}	Maximum size for training library \mathcal{L}_{train}
L	Loss function
\mathcal{L}	Policy library
\mathcal{L}_{train}	Policy library for training
\mathcal{M}	Replay memory

$\mathcal{M}_{S_i} / \mathcal{M}_T$	Replay memory a source task / of a target task
Ω	Task
Ω_{S_i}	Source task
Ω_T	Target task
Ω_{ϑ}	Task of a case ϑ
Π	Space of all policies
π	Policy for agent behaviour
π^*	Optimal policy for agent behaviour
π_{Ω}	Policy for agent behaviour for a task Ω
$\pi_{\Omega_{\vartheta}}$	Policy for agent behaviour for a task Ω of a case ϑ
$\hat{\pi}_{\Omega}$	Proposed policy for agent behaviour for a task Ω
π_{o_i}	Partial policy for agent behaviour (also <i>Option</i>)
Q	State-action value function
$Q_{\Omega_{\vartheta}}$	State-action value function for a task Ω of a case ϑ
\mathbb{R}	Space of real numbers
R	Reward function for Markov Decision Processes
r_t	Reward at time t
S	State space for Markov Decision Processes
s	State

s'	Follow-up state
s_t	State at time step t
$sim_{policy/task}$	Similarity between policies / tasks
$\sigma_{policy/task}$	Threshold factor for similarity between policies /tasks
T	Transition function for Markov Decision Processes
θ / θ^-	Network parameter
θ_Ω	Network parameter for a given task Ω
θ_Ω^l	Network parameter of a specific layer l for a given task Ω
ϑ	Case in Case-based Reasoning
V	State value function
w	Importance vector
Y	Training target

Contents

1	Introduction	1
1.1	Objectives	3
1.2	Contributions	4
1.3	Organization	5
2	Background	6
2.1	Deep Reinforcement Learning	6
2.1.1	Reinforcement Learning	6
2.1.2	Deep Reinforcement Learning	10
2.1.3	Deep Q-Network	11
2.2	Transfer for Deep Reinforcement Learning	13
2.2.1	Definition and Motivation	13
2.2.2	Dimensions in Transfer Learning for Deep Reinforcement Learning	15
2.2.3	Discussion	21
2.3	Case-based Reasoning	22
2.3.1	General introduction	23
2.3.2	Case-based Reasoning and Reinforcement Learning	24
3	Proposal	25
3.1	Motivation	25
3.2	Formulating CBR in RL terminology	26
3.2.1	RETRIEVE	28
3.2.2	REUSE	28

3.2.3	REVISE	29
3.2.4	RETAIN	30
3.2.5	Summary	31
3.3	Proposing a novel algorithm: DECAF	31
3.4	Discussion of related work	37
4	Experiments	39
4.1	Parameter initialization for agent networks	39
4.1.1	Importance of optimization method and network architecture	42
4.1.2	Importance of parameter initialization	43
4.1.3	Discussion	46
4.2	First steps with CBR for RL	46
4.2.1	Learning a new policy	47
4.2.2	Building a core policy library	50
4.2.3	Discussion	51
4.3	Using DECAF for knowledge building and reuse	51
4.3.1	Training parameters for <i>DECAF</i>	52
4.3.2	Determine the importance of pre-selection of source tasks	54
4.3.3	Working with a larger library	55
4.3.4	Discussion	58
5	Final considerations	60
5.1	Conclusion	60
5.2	Future work	61
A	Related works for transfer in Deep Reinforcement Learning	64
B	Productivity and Accomplishments	66
	Bibliography	72

Chapter 1

Introduction

After the enthusiasm for *Artificial Intelligence (AI)* was not met with the desired and overestimated results in the late 80s and early 90s, general interest in the field was severely reduced and replaced with disappointment and criticism, leading to funding cuts and an almost two decade long phase known as the *AI winter*. With an increasing amount of data and vastly improved computing capabilities, a need for new solutions evolved at the beginning of this century which sparked new interest and led to new funding in the field, which was the beginning of the rapid development and massive investments that we are seeing today.

Today *AI* is one of the most important and highest funded research areas because it can be applied not only to related fields, like Robotics, Computer Vision, or Data Sciences, but also to any field that generates or uses a lot of data, like Biology, Genetics, Medicine, Mechanics, Material Sciences, or Economics. In fact, many solutions are already part of every ones life. The range of examples is huge and not only consists of driver assistance systems in vehicles, speech recognition and *Natural Language Processing* in mobile phones or other computing devices, object or face recognition in photos and videos, financial trading on stock markets, product/video/music-recommendations on service platforms, gene decoding to understand the human body, but also many others that are deeply embedded in our everyday life. *Machine Learning (ML)* provides many of the tools and algorithms that drive applications and development in *AI* and is also at the center of this work.

The driving force behind these developments is the field of *Deep Learning (DL)* originating from the comeback of non-linear function approximators based on *Neural Network (NN)* architectures [Schmidhuber, 2015]. Mainly influenced by the work of Geoff Hinton, Joshua Bengio and Yann LeCun, *DL* architectures became one of the most powerful tools in *ML*, beating long standing records not just in one, but in almost any domain [LeCun et al., 2015]. *DL* allows to learn abstract representations of high dimensional input data and its ability to generalize can be used to enhance many existing techniques.

Another well-researched area in *ML* is the field of *Reinforcement Learning (RL)*, which enables an agent to learn a task through repetition and integrating feedback of state changes and reward updates into the learning process, while exploring a given environment [Sutton and Barto, 2018]. While the classic *RL* approaches were only suitable for simple problems, the integration of *DL* techniques in the learning process made it possible to move to more complex tasks [Mnih et al., 2015].

A remaining challenge of most of the techniques from this areas is the restriction to single task learning. In an expensive learning process, agents need to learn every new task from scratch without considering previously gained knowledge.

The abilities to gather and to reuse knowledge is formulated in the area of *Transfer Learning (TL)*. The approaches in this area aim at building more autonomous and more capable *ML* agents that are closer aligned with the workings of the human brain. The brain stores and organizes acquired knowledge into concepts, models and categories, which are based on the input of senses and states of muscles. At the same time, it recognizes similarities between tasks and domains, and generates connections between them. This ability offers the advantage that we do not have to learn every new task from scratch, but can rely on abstractions of past experiences.

These assumptions also provide the basis for the research efforts described here, where we are concerned with the development of intelligent agents that are able to learn new tasks by relying on knowledge transferred from previously learned tasks, turning the learning experience faster and more efficient. While research on single task learning has already produced many excellent results, research on systems for multiple tasks still offers many opportunities for improve-

ment. Hence, our research focuses on developing a more general learning approach, mimicking the human *TL* abilities towards systems that enable a kind of *Lifelong Machine Learning*, initially introduced as *Lifelong Robot Learning* [Thrun and Mitchell, 1995].

The underlying techniques, which we find most suited for this endeavour are a combination of *RL* and *DL* together with a systematic approach to reusing knowledge, described under the *Case-based Reasoning (CBR)* methodology [Kolodner, 2014]. Therefore, we are here concerned with the research and development of *Deep Reinforcement Learning (DRL)* agents that are able to utilize *TL* techniques to improve learning of multiple tasks in succession. This area is especially interesting for us because we consider the areas of knowledge transfer and *DL* a big step for *ML* in the direction of building smarter agents, which are able to build up knowledge, autonomously choose a suitable learning model for a new challenge, and improve the generalization of knowledge.

1.1 Objectives

We here investigate methodologies for systematic approaches to *knowledge building* and *knowledge reuse* for intelligent agents.

We focus on the area of *TL* for *DRL* which is currently an emerging field because of its high potential for solving very complex problems with the least amount of necessary manual intervention for a great variety of challenges. Since most of the existing literature neglects a systematic approach for solving more than a single task while preserving the gathered knowledge, we focus our efforts here.

More specifically, the objectives of this research are to

- evaluate the existing literature for *knowledge transfer* in *DRL*;
- identify suitable frameworks for a systematic approach for *knowledge reuse* for *DRL* agents;
- propose an algorithm based on the framework to improve the state of the art;

- empirically show the usefulness of the framework in experiments comparing our approach to other methods.

1.2 Contributions

In this thesis, we contribute a novel approach to systematically reuse knowledge based on the combination of *DRL* with the *CBR* methodology.

We first provide an overview of methods to perform *knowledge transfer* in the existing literature for *DRL*, a field which has just been starting to appear frequently in the relevant literature in the last two years. As a means to add a more systematic approach for *knowledge transfer* in *RL*, we formulate the *CBR* methodology in *RL* terminology to facilitate the adaption and communication between the respective communities. Building on this framework we then propose an algorithm, termed *Deep Case-based Policy Inference (DECAF)*, that is able to successfully build up knowledge and to reuse it as it learns new tasks. In our experimental evaluation, we then show that knowledge transfer can play an important role, but also show that it is very difficult to achieve benefits if the source for knowledge is not carefully chosen. Then we evaluate our framework in the *TL* context and demonstrate the importance of the systematic approach in a number of experiments, where highlight the benefits of our approach.

During this work, we published a number of articles in relevant international conferences and workshops, as well as recognized journals, participated and organized research activities, and openly shared some of the source code that we developed.

We consider this work important towards building more autonomous agents that are not limited to single task learning, but are able to build up knowledge over a variety of tasks and consider the learned knowledge to speed up and to improve the learning of new tasks.

1.3 Organization

This first chapter (Chapter 1), *Introduction*, gave a short introduction to our work, providing a context to the whole manuscript, laying out our objectives, and summarizing the achieved contributions. In Chapter 2, *Background*, we will cover the underlying basic concepts that form the foundation for this research proposal, specifically *DRL*, *TL* for *DRL*, and *CBR*. Chapter 3, *Proposal*, details our research proposal with the underlying motivation, the concrete proposal, and a demarcation to the existing literature. We then describe the conducted experiments and discuss the achieved results in Chapter 4, *Experiments*. In the final chapter (Chapter 5), *Final considerations*, we provide a summary and a conclusion of the conducted research and propose future work on the topic.

In the appendix, we give some more details about related work (Appendix A) and accomplishments and productivity during the Ph.D. program (Appendix B).

Chapter 2

Background

In this chapter, we are introducing the underlying core concepts, algorithms, and methodologies that are essential for the understanding of this work and which provide the context for the following chapters. Section 2.1 lays out the basics of *RL*, specifically *DRL* and the *Deep Q-Network (DQN)*, Section 2.2 provides an overview for *TL* methods for *DRL*, and Section 2.3 introduces *CBR* in general.

2.1 Deep Reinforcement Learning

In recent years, *RL* has seen a massive surge in interest driven by new findings in representation learning that make it possible to work with problems of a much higher complexity than was possible before. The combination of these findings with *RL* is forming the new area of *DRL* which was kickstarted with outstanding results in learning how to play *Atari* games using an approach based on a *DQN* architecture.

2.1.1 Reinforcement Learning

A *Markov Decision Process (MDP)* is described by a $\langle S, A, T, R \rangle$ tuple where S is the set of possible states, A is the set of applicable actions, $T : S \times A \times S \rightarrow [0, 1]$ is a transition function that defines the probability of observing a follow-up state s' after applying action a in state s , and $R : S \times A \rightarrow \mathbb{R}$ is a reward function that

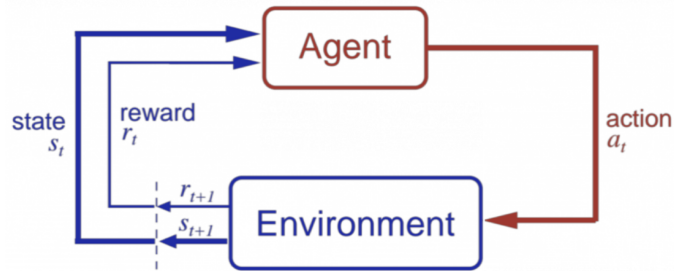


Figure 2.1: A standard *Reinforcement Learning* setup. After performing an action, an agent receives a reward and an update of the current state from the environment. Source: Adapted from Sutton and Barto [2018].

provides a reward for applying action a in state s [Puterman, 2014]. The agent’s goal is to find an optimal policy π^* , that dictates the best action in any state. In learning problems, the agent does not know T and R , hence it has to learn through samples of $\langle s, a, r, s' \rangle$, where r is the reward observed after applying a in s and reaching follow-up state s' .

Many possible ways to solve this kind of decision-making problems are described in the field of *RL* which is a well-researched area in *ML* [Sutton and Barto, 2018]. In *RL*, an agent explores the space of possible strategies or actions in a given environment, receives a feedback (reward or cost) on the outcome of the choices made and deduces a behavior policy from his observations. As shown in Figure 2.1, an agent can interact with its environment by performing an action at every discrete time-step. The environment then answers by updating the current state to a follow-up state and by giving a reward to the agent, indicating the value of performing an explicit action in a concrete state. In general, an agent initially interacts with the environment in a trial-and-error manner to explore the state space with as little bias as possible before learning how to act in the best possible way for each possible state. The policy is then learned by trying to maximize the accumulated reward over the lifetime of the agent.

Successes have been achieved through various techniques like *Dynamic Programming* [Bellman, 1966], *Monte Carlo Methods* [Gilks et al., 1995], or *Temporal Difference (TD) Learning* [Tesauro, 1992], and in a variety of applications, like the board-game domain [Tesauro, 1995], robot soccer [Stone and Sutton, 2001] or

autonomous helicopter flight [Ng et al., 2006].

A differentiation of agents can be based on the way it learns to solve a task. In a *model-free* method, the agent directly learns a policy without learning transition and reward functions explicitly. A *model-based* method learns the whole MDP first and then determines an optimal policy based on planning algorithms. In this work, we are only concerned with *model-free* methods.

Another important distinction is based on the way the agent performs the learning updates. An *on-policy* learner improves the policy the agent is currently following, while an *off-policy* learner learns the value of the optimal policy independently of the current policy of the agent by estimating future behaviour.

The internal representation of *RL* agents can be divided into three main groups: *critic-only*, *actor-only*, and *actor-critic* methods. *Critic-only* methods are value-based, meaning the agent learns a value function which gives an indication of either the value of being in a given state $V(s)$ or the value of taking a specific action in a given state, also called quality value $Q(s, a)$, where the *state value* can also be expressed through the *state-action value* as $V(s) = \max_{a \in A} Q(s, a)$. *Actor-only* are policy-based methods and learn a policy $\pi(s)$ that directly tells the agent which policy to take regardless of the actual value that a state might have for the agent. *Actor-critic* methods on the other hand learn a policy but are guided by the actual *state-action* value, combining the two methods to benefit from their respective advantages while reducing their shortcomings. In this work, we restrict the observations to *critic-only* methods although in many cases the results would be expected to behave similar for the other methods with small changes to the algorithms.

In our algorithms and experiments, most considerations are based on *Q-Learning* which is an *off-policy TD* method and described in Algorithm 1 [Watkins and Dayan, 1992]. In *Q-Learning*, the agent is concerned with discovering an optimal policy π^* for every state by finding an optimal solution for the *state-action* value. The optimal policy π^* can then be described as

$$\begin{aligned} \pi^*(s_t) &= \arg \max_{a \in A(s_t)} Q^*(s_t, a) \\ &= \arg \max_{a \in A(s_t)} [r_t + \gamma \max_{a \in A(s_{t+1})} Q^*(s_{t+1}, a)], \end{aligned} \tag{2.1}$$

Algorithm 1 Q-Learning

```
1: Initialize  $Q(s, a)$  with random values for all  $s \in S$  and  $a \in A$ 
2: repeat (for each episode)
3:   initialize  $s_t$ 
4:   repeat (for each time-step in episode)
5:     Choose action  $a_t$  in  $s_t$  following  $\varepsilon$ -greedy strategy
6:     Observe new state  $s_{t+1}$  and reward  $r_t$  and
7:     Update Q-Value:  $Q(s_t, a_t) \leftarrow (1 - \alpha)Q(s_t, a_t) + \alpha[r_t + \gamma \max_a Q(s_{t+1}, a)]$ 
8:     Set  $s_t \leftarrow s_{t+1}$ 
9:   until episode ends
10: until no more episodes
```

where s_t is the state at time t , s_{t+1} the follow-up state, a_t the action taken at time t , r_t the reward received at time t when taking action a_t in s_t , and γ a discount factor for future rewards. The term in brackets represents a one step look-ahead where we receive the reward for taking a specific action and assume that we follow an optimal policy from that point on. The *Q-Learning* approach has been researched extensively and various extensions and adaptations have been proposed, as for example the use of *Multi Layer Perceptrons* in *Fitted Q-Learning*, an early approach of using *NN* architectures with *RL* [Riedmiller, 2005].

During training many states have to be explored on a trial-and-error basis until the agent is able to make a reliable suggestion for a good policy. For this purpose, the agent needs to find a balance between exploration and exploitation, where exploration refers to exploring the state space to find new states and exploitation refers to following an already existing policy π to strengthen the gained insights on how to act in the environment [Kaelbling et al., 1996]. A popular strategy to achieve this is the ε -greedy strategy, where ε indicates the probability of choosing a random action a at a given time step and is continuously decreasing during training [Watkins and Dayan, 1992]. The algorithm also features a learning rate α , which determines how to update the *state-action* value, where a value too high might make learning very difficult and a value too low might make learning very slow.

2.1.2 Deep Reinforcement Learning

With the raise of *DL* [LeCun et al., 2015], it became possible to use *RL* successfully on increasingly complex tasks with very large state spaces like computer games [Mnih et al., 2015] by benefiting from the generalization abilities of *Deep Neural Network (DNN)* architectures.

DNN architectures are characterized by multiple layers of *artificial NN* structures [Schmidhuber, 2015]. In this work we are only interested in *Artificial Neural Networks* in contrast to *Biological Neural Networks* and therefore dispense the prefix. Those architectures are very flexible and allow the training of intermediate levels of representation using either supervised or unsupervised learning, which can be applied locally at each level or globally. Because each hidden layer computes a non-linear transformation of the previous layer, a *DNN* can have significantly greater representational power (i.e., can learn a significantly more complex representation) than a shallow one (one with only one or few layers).

A big advantage of *DNN* architectures is the fact that their parameters do not need to be carefully designed but are learned from data using a general-purpose learning procedure. Also, the network can generalize non-locally and approximate outputs for unknown input data. The progress is mainly driven by a massive increase of available data [Kitchin, 2014], increasing computational power and parallelization ability, e.g. *Graphic Processing Units (GPU)* with *CUDA* cores [Kirk and Wen-me, 2012], and advances in algorithms like layer-wise pre-training [Bengio et al., 2007], optimization methods [Ngiam et al., 2011], activation functions like Rectified Linear Units (ReLU) [Nair and Hinton, 2010] and approximation methods [Gal and Ghahramani, 2016]. *DL* algorithms already have produced revolutionary results in many fields like image recognition [Krizhevsky et al., 2012], speech recognition [Hinton et al., 2012], *Natural Language Processing* [Collobert et al., 2011], or Genetics [Xiong et al., 2015], and they are quickly covering new areas.

Combining *DL* with *RL* allows learning of abstract representations of high dimensional input data while learning a behaviour policy at the same time in end-to-end learning models. The resulting field of *DRL* is still a young research field but promises to solve the most complex tasks and has been met with a lot of

research efforts in the recent past. There are, however, a number of challenges that have not yet been overcome and limit the application to real world problems.¹

Successful applications of *DRL* include areas, but are not limited to, such as maze navigation [Mirowski et al., 2017], robotics [Levine et al., 2016], learning real world physics [Denil et al., 2016], and Atari games [Hessel et al., 2017].

2.1.3 Deep Q-Network

Mnih et al. [2015] were the first to propose a novel agent based on a *DNN* architecture (see Figure 2.2). Their approach was based on the original *Q-Learning* algorithm and hence the name *Deep Q-Network (DQN)*. Since in many domains the state space is too large to tractably store a tabular representation of the *state-action* values, the *DQN* approach uses a deep function approximator to represent a *state-action* value function instead. They show that the algorithm can successfully learn policies directly from simply observing the game-play on the screen while acting in a *Atari game playing* environment. They also demonstrate that the same architecture can learn control policies in a variety of different environments without using prior knowledge and without changing any initial parameters.

Training a *NN* is hard even with a perfect training signal as in *Supervised Learning*. Before this publication no one had succeeded to apply a *DNN* architecture for *RL* directly because the training signal was too unreliable. The difference to previous approaches were three small changes that helped to stabilize the training signal and led to highly improved performance. They can best be described by first looking at the original loss function L , also referred to as *TD* error,

$$L(\theta) = \mathbb{E}[r + \gamma \max_{a'} Q(s', a', \theta) - Q(s, a, \theta)]. \quad (2.2)$$

Here, the first part of the expectation is the training target

$$y(\theta) = r + \gamma \max_{a'} Q(s', a', \theta), \quad (2.3)$$

¹For a more thorough discussion please read <https://www.alexirpan.com/2018/02/14/rl-hard.html>

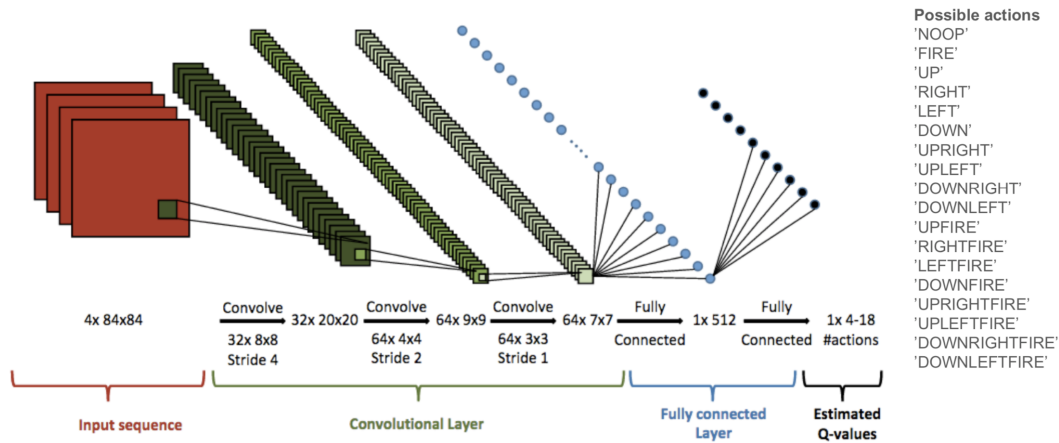


Figure 2.2: Sketch of the architecture of the original *DQN* with three convolutional layers and two fully connected layers, where the nodes in the highest layer each represent a *state-action* value for all available actions. Source: Adapted from Mnih et al. [2015].

which is the actual reward r that was achieved plus the expectation of the next optimal *state-action* value $Q(s', a', \theta)$ multiplied by the discount factor γ .

The first change was to apply *reward clipping* to the rewards to keep updates at reasonable levels, here $\{r \in \mathbb{R} \mid -1 \leq r \leq 1\}$, and to avoid wildly increasing *state-action* values.

The next change was to introduce a second network, termed *target* network, with parameters θ^- that was fixed during training and only updated with the current parameters from the *online* network θ in given time intervals. This served to provide a more stable training signal for the agent and also to keep network updates from making big jumps.

The third change was the introduction of a replay memory \mathcal{M} which stores its experiences in the form of $e_t = \langle s_t, a_t, r_t, s_{t+1} \rangle$ at every time step t . Instead of using the current experiences during training, the agent can now draw random experiences from the replay memory to break correlations between successive experiences, improve data efficiency, and eliminate unwanted feedback-loops.

The improved loss function of the agent, considering all these small changes,

is now

$$L(\theta, \theta^-) = \mathbb{E}_{s,a,r,s' \sim \mathcal{M}} [r + \gamma \max_{a'} Q(s', a', \theta^-) - Q(s, a, \theta)], \quad (2.4)$$

and can successfully be applied to a number of challenging learning problems.

2.2 Transfer for Deep Reinforcement Learning

In this work, we focus on *TL* approaches that can be used for *DRL*. Although some methods from classic *RL* are still applicable, using *DNN* architectures rules out many methods specialized for tabular and linear *RL*. Moreover, the knowledge about the agent representation can be leveraged for building more efficient methods tailored for non-linear *DNN*-based agents. Therefore, we specifically searched for *TL* methods that can directly be applied to *DRL* agents [Glatt et al., 2020].

Other surveys have focused on different aspects of knowledge transfer like transfer for supervised learning [Pan and Yang, 2010, Weiss et al., 2016], unsupervised learning [Bengio, 2012], or multiagent systems [Silva and Costa, 2019]. Available surveys for *TL* for *RL* provide great overviews for the area before the introduction of *DRL* but are not considering the latest developments [Taylor and Stone, 2009, Lazaric, 2012]. Although there is an intersection of interests between those previous surveys and ours, none of them clearly lists all the types of knowledge reuse techniques that can be applied to *DRL*.

2.2.1 Definition and Motivation

As mentioned earlier, in *RL* a task can fully be described by its underlying *MDP*, consisting of a tuple $\langle S, A, T, R \rangle$ [Puterman, 2014]. The domain \mathcal{D} of a task Ω can then be described by the state and action spaces as $\mathcal{D}^\Omega : A^\Omega \times S^\Omega$. Agents can generate experiences $e_t = \langle s_t, a_t, r_t, s_{t+1} \rangle$ which are typically stored in a replay memory \mathcal{M} from where the agent samples during training.

Normally, *RL* is concerned with problems where we have only partial information of the task objective by means of rewards and need to learn a behavior policy $\pi_\Omega(s)$ that provides a suggestion of which action to perform in any given state to

maximize the cumulative reward over the lifetime of an agent when performing task Ω . In *DRL* the policy for a task is encoded in the parameters (weights) θ_Ω of a *DNN* architecture. The network consists of stacked layers with network parameters θ_Ω^l for each layer l that can consist of different kind of network structures like convolutional, recurrent, or fully connected layers. For transfer, it is often beneficial to learn partial policies (also skills or options) $\pi_o(s)$ that are used to solve smaller sub-tasks and can be more specialized than whole policies [Konidaris and Barto, 2007].

In this context *TL* describes algorithms that aim at improving the learning speed and overall performance when learning a target task Ω_T using internal knowledge in the form of previously acquired knowledge from a single task or multiple source tasks Ω_{S_i} . The way the knowledge of the source tasks Ω_{S_i} was acquired only plays a secondary role while the focus lies on how and when it is beneficial to use transfer. Transfer can happen between tasks within the same domain where $\mathcal{D}^{\Omega_T} = \mathcal{D}^{\Omega_{S_i}}$, but also across domains where $\mathcal{D}^{\Omega_T} \neq \mathcal{D}^{\Omega_{S_i}}$. In the second case, often some kind of inter-task mapping can be learned or defined to still achieve benefits from transferring knowledge [Taylor et al., 2007].

Using previous knowledge to solve a new problem can have a number of benefits. They are shown in Figure 2.3 and include *Jumpstart*, *Time to Threshold* and *Asymptotic Performance*.

The *Jumpstart* is a metric that shows the vertical improvement in performance at the beginning of the training and can be considered very important for algorithms that are aiming at quick successes in the training of a new task. The *Time to Threshold* on the other hand is concerned with the temporal improvement until reaching a certain level of performance and is shown as the horizontal improvement at that level which is especially important for tasks that can be considered solved even if the optimal policy has not yet been found. Finally, the *Asymptotic Performance* is again a metric that is concerned with the improvement of the vertical performance, but this time at the end of training, indicating that the transfer led to a better generalization and improved learning.

In cases where it is not important which of those metrics are the goal for improvements or if there is a given time frame for training, one can measure the success by either comparing the *total accumulated reward* achieved during train-

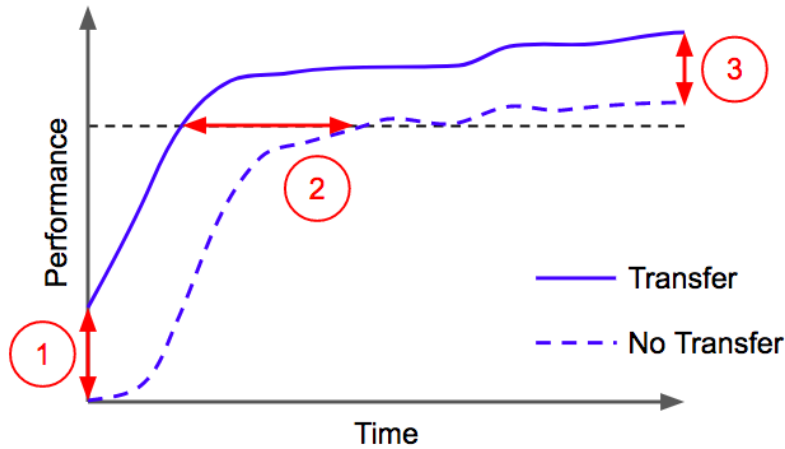


Figure 2.3: Potential benefits of knowledge transfer: *Jumpstart* (1), *Time to Threshold* (2), and *Asymptotic Performance* (3). Source: Adapted from Taylor and Stone [2009].

ing or by looking at a *transfer ratio* calculated by the difference of the area under both *performance over time* curves divided by the area under the curve for the baseline algorithm.

2.2.2 Dimensions in Transfer Learning for Deep Reinforcement Learning

As a pretext to our research contribution, we investigated the general use of knowledge transfer in *DRL*. We describe three core dimensions that characterize the recent lines of research on *TL* for *DRL* and place our own work in this context. The dimensions are *Type of Learner*, *Agent Quantity*, and *Strategies for Knowledge Transfer*. In general, most works can be classified over these three dimensions even though some are overlapping and fit into several categories within a single dimension.

Type of Learner

Depending on the desired scenario, new tasks might be approached in different ways, which naturally will affect the way knowledge can be reused most effec-

tively.

Multi-task Learner are agents that learn a variety of tasks at the same time. This is a special case of transfer where transferable knowledge might not necessarily be available when the learning starts but might be build up at the same time for multiple target tasks. For example, Parisotto et al. [2016] propose an approach where an agent learns in various tasks simultaneously and then generalizes to new tasks while remaining with the same model complexity as a single task agent.

Imitation Learner are agents that are able to master a task by observing demonstrations and adapting the solution quickly. Although imitation learning has a long history and can be seen as a whole field on its own, we are focusing only on works utilizing *DRL*. For example, Stadie et al. [2017] propose third-person imitation learning where the learning is aligned close to human learning, in that we observe other humans perform tasks, infer that task, and then accomplish the same task without having the same perspective as the demonstrator.

Lifelong Learner are defined as intelligent agents that receive a sequence of tasks building up knowledge over time [Thrun, 1998]. Using this definition most of the works fall in this category where a learner only concerns itself with one task at a time but may use a variety of knowledge sources for that purpose. A single new task is considered for each transfer problem. For example, Finn et al. [2017] present a method for semi-supervised reinforcement learning motivated by real-world lifelong learning in continuous control robot tasks, where the reward is inferred when it is not available.

In this work, we are focusing on *Lifelong Learner* agents to build up knowledge over time while learning one task after another.

Agent Quantity

This dimension is concerned with the type of learning with respect to the quantity of active learners in the target task. Most works until today are considering single agents when training a new target task, but improved computational resources are paving the way for parallelization and the capability to employ more than a single agent.

Single agent Learner are agents that are solely learning a target task (the

amount of agents used for knowledge generation is irrelevant). This is still one of the most common settings in the research literature where an agent is faced with sequential tasks often in a hierarchical order. Designing an algorithm for this setting is also simpler because one can use straight forward implementations of well-researched algorithms. For example, the original *DQN* algorithm is based on single agent learning [Mnih et al., 2015].

Multiagent Learner basically cover two scenarios, one in which multiple agents are actuating in the same environment and another one in which multiple copies of the environment are independently and simultaneously executed by multiple agents. Often, multiple agents are leveraged through simultaneously updating a common policy or a shared sample memory for offline training. An example of such an algorithm is *Asynchronous Advantage Actor-Critic (A3C)*, where multiple workers explore the environment simultaneously for sharing experiences to stabilize a single policy while updating network parameters through asynchronous parallel processes [Mnih et al., 2016].

Even though multiagent approaches show a stabilizing effect on learning performance with better generalization abilities, we are focusing on *Single agent Learner* because of computational restrictions and the fact that they can often easily be adapted to multiagent approaches later.

Strategies for Knowledge Transfer

The most diverse dimension to classify *TL* for *DRL* algorithms is the strategy they follow for reusing knowledge.

Direct parameter transfer covers transfer approaches where the network parameters of a previously learned task are used as initialization for the network that learns the target task while fine-tuning the network. This happens either for the whole network $\theta_{\Omega_s} \rightarrow \theta_{\Omega_t}$ or only for selected layers $\theta_{\Omega_s}^l \rightarrow \theta_{\Omega_t}^l$ (see Figure 2.4). As an early work in this category, we have contributed with a publication that investigated the effectiveness of network initialization in the target task by directly copying parameters learned in a source task and then continuing training in the target task without fixing any layers instead of randomly initializing them [Glatt et al., 2016]. We will provide more details about this in Section 4.1.

Sample transfer is another straightforward yet effective way to reuse knowledge across tasks. Here the agent reuses sample experiences from one or several source tasks in a target task, $\{\mathcal{M}_{S_1}, \dots, \mathcal{M}_{S_i}\} \rightarrow \mathcal{M}_T$ (see Figure 2.5). If the tasks are very similar, simply reusing the samples as if the agent was experiencing it in the new task might be enough to consistently accelerate learning. Another advantage of sample reuse is the fact that it can stabilize learning because it might force the policy to be more general which helps to protect against overfitting. Gu et al. [2017], for example, propose to accelerate learning by distributing policy updates across multiple similar robots, which provides a drastic speed up in the learning process. They use a learner thread, which is separated from the experience-collecting worker threads and dispatches updated policy parameters to each of the worker threads.

Partial solution transfer can be used if there exist solutions to parts of a task that can be learned separately or extracted from the full solution. This is done because very often only a part of the acquired knowledge is suitable for transfer between tasks and would help in the new task. Consequently, some works focus on transferring only partial solutions to new tasks, $\{\pi_{o_1}, \dots, \pi_{o_i}\} \rightarrow \pi_{\Omega_T}$, aiming at reusing behaviors that are similar across tasks while trying to avoid the negative influence of task-specific knowledge (see Figure 2.6). However, how to transfer those behaviors is a challenge, both because it is hard to extract only part of a solution and very difficult to adequately compute similarities to select the right partial solutions for transfer. Vezhnevets et al. [2017], for example, describe *Feudal networks*, an architecture to learn and manage sub-skills when learning an individual task to make long-term credit assignment and memorization more tractable. The

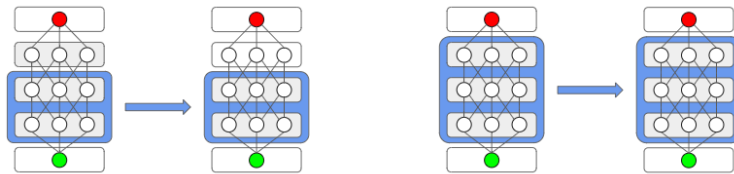


Figure 2.4: Parameter transfer can be applied to only part (left) or all layers (right) of a previously trained network on a source task, and is commonly used as initialization when training the network of the target task. Source: Author.

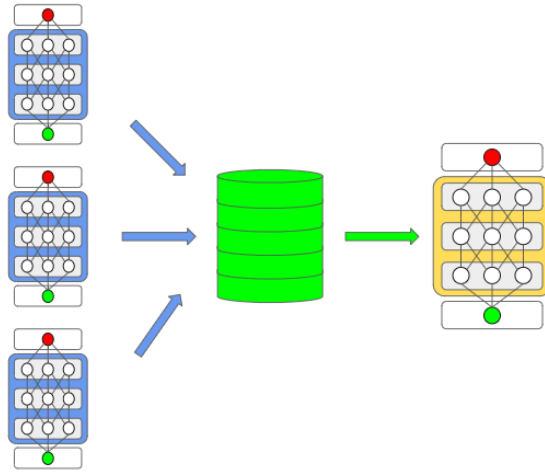


Figure 2.5: In *Sample transfer* many workers are generating independent samples by interacting with the environment under different pretexts which are then used to stabilize training of the target task. Source: Author.

goal is to decompose agent behavior into meaningful primitives and then reuse them to more efficiently acquire new behavior.

Curriculum Learning is organizing the learning of several tasks according to difficulty of tasks or complexity of the agents. This makes sense since, sometimes, agents struggle to learn complex tasks directly and it might help to have the agent learn easier tasks first and then gradually increase the difficulty while transferring the knowledge continuously from task to task until it is able to learn the hard tasks (see Figure 2.7). Shao et al. [2018], for example, show the applicability of *curriculum learning* to a Real-Time Strategy game where they use the curriculum to train agents in increasingly difficult levels. A different approach is the *Mix & Match* framework that forms a curriculum over agents to progressively train more complex agents bootstrapped from simpler agents, increasing the granularity of the agents abilities over time [Czarnecki et al., 2018].

Self-play is a very powerful approach to train agents, often specialized for adversarial tasks. In *self-play* agents play against copies of themselves until they safely beat their own strategy, then update the opponent to the current strategy, $\pi_{\Omega_{T_{new}}} \rightarrow \pi_{\Omega_{T_{old}}}$, and then repeat the process (see Figure 2.8). Enabling an agent

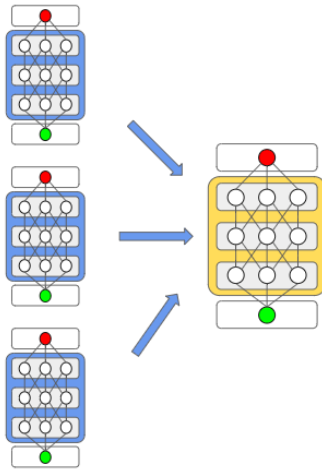


Figure 2.6: *Partial solutions* are used to directly guide the training of the target task. Source: Author.

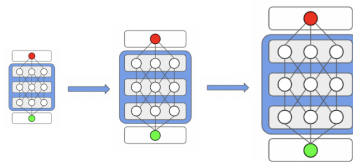


Figure 2.7: In *Curriculum Learning*, an agent progressively trains on more complex challenges with respect to either the task at hand or the agent representations. Source: Author.

to build up knowledge in this open-ended manner may lead to the development of very complex behaviors. The difficulty lies in avoiding overfitting and getting stuck in a kind of strategy loop that would stop the agent from evolving further. One of the most popular applications of *DRL* was presented by *Deepmind* when they beat the worlds best Go player [Silver et al., 2017] and shortly after even produced an algorithm that was more general and also learned the games of Chess and Shogi [Silver et al., 2018].

Meta solutions or *Meta Learning* is one of the most popular approaches in the recent past. Even though solutions can be used sometimes directly as in the previous sections, network parameters are very specialized and usually hard to general-

ize across tasks. A possible way to reuse previous policies is by combining them into a *meta solution* which generalizes over multiple solutions and can be used to initialize learning in a new task, $\{\pi_{\Omega_{S_1}}, \dots, \pi_{\Omega_{S_i}}\} \rightarrow \theta_{\Omega_M} \rightarrow \theta_{\Omega_T}$ (see Figure 2.9). Rusu et al. [2016], for example, introduce *Progressive networks* which establish lateral connections to previously learned features in parallel networks and are able to reuse previous knowledge while being immune to catastrophic forgetting. Teh et al. [2017] propose a network architecture they call *DISTRAL* (DIstill & TRAnsfer Learning) as a new approach for joint training of multiple tasks, where agents are sharing a distilled policy that captures common behavior between tasks which stabilizes training and makes *DRL* more robust against hyperparameter settings.

Our focus in this category was firstly set on *direct parameter transfer* but after initial investigations we were more interested in pursuing the direction of *Meta solutions*, which we think offers greater generalization abilities and has much more potential for impact in real-world challenges.

2.2.3 Discussion

After having a slow start following the introduction of *DRL* approaches, *TL* for *DRL* has become a very important research topic with a quickly increasing number of publications, especially in the last two years. Although even *DRL* itself is a very young field, there is now already a number of groups actively researching to reuse knowledge for *DRL*, providing interesting results, and publishing them in high-quality venues.

Up until today, most work in the area can be categorized in a *Lifelong Learning* setting utilizing only *single agent* algorithms to build up knowledge over time

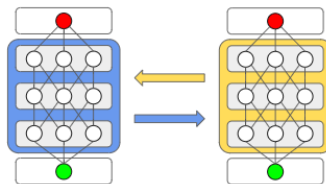


Figure 2.8: *Self-play* uses an agents own policy to play against a copy of itself, updating the opponent with the current policy from time to time. Source: Author.

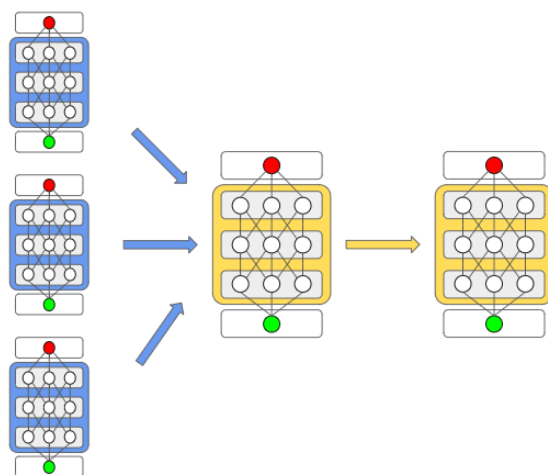


Figure 2.9: *Meta solutions* describe approaches where previously learned tasks are combined to a meta solution, which is then used to initialize the target task or to guide the target task training. Source: Author.

as can be seen in Table A.1 in Appendix A. *Meta Learning* is quickly gaining popularity and, in general, the trend seems to be going towards more complex architectures that heavily borrow from insights from other domains and progress in *DL* research. But the decision on which approach to follow is often task-specific and should be considered under the constraints of the domain, available access to the environment, the difficulty of the tasks at hand, or the architecture of the agent.

2.3 Case-based Reasoning

An important concept to benefit from previously acquired knowledge is defined under the term *Case-based Reasoning (CBR)* [Aamodt and Plaza, 1994] which describes a methodology to build computational models to reuse existing knowledge in a general manner [Watson, 1999].

2.3.1 General introduction

In *CBR* new problems are solved by adapting successful previous solutions to similar problems under the general guiding assumption for *CBR* that similar problems have similar solutions. Specifically, *CBR* makes assumptions about *regularity* (same actions in same situation produce same outcomes), *typicality* (similar situations appear repeatedly), *consistency* (similar situations require similar solutions), and *adaptability* (small changes in the situation only require small changes in the solution) [Kolodner, 1996].

CBR has been shown to be very successful and has been widely applied in a number of fields [Cheetham and Watson, 2005]. A high-level description of the *CBR* cycle describes four stages during the learning of a new task with previously acquired knowledge [Aamodt and Plaza, 1994, Kolodner, 2014]. First, the agent has to *RETRIEVE* the most similar cases from its knowledge base and then *REUSE* the contained information to propose a solution to the new task. After that, the proposed solution is *REVISED* to find a final solution to the target task. In the end, the agent can decide if it should *RETAIN* the gathered solution in the knowledge base. We will describe these steps in more detail in Section 3.2.

The benefits of *CBR* are that the knowledge is easily available since it does not need to be specifically extracted (*Elicitation*), cases can be used for several different problems (*Problem solving bias*), no need to build abstractions (*Incremental Learning*), suited for complex solution spaces (*Disjunctive Solution Space*), possibility to make assumptions about the new solution from retrieved solutions (*Precedent Explanations*), and accumulates more knowledge over time (*Sequential Problem Solving*) [Aha, 1998b].

The downsides of *CBR* are that it might have large storage requirements depending on the problems space and the limitations set for knowledge preservation, and that the functions for retrieving and storing knowledge are often not trivial and require careful adaption.

2.3.2 Case-based Reasoning and Reinforcement Learning

Other works considered ideas from both *CBR* and *RL* before, even though they do not combine them into a single framework as we do here. Gabel and Riedmiller [2005] make use of *CBR* for function approximation to assess the desirability of the state it finds itself in. Similarly, Sharma et al. [2007] propose a *CBR* framework that is used as an instance-based state function approximator in a layered *RL* architecture. We, on the other hand, will directly use previously learned policies for accelerating learning.

Bianchi et al. [2009] also rely on a *CBR* approach to accelerate learning in *RL* tasks. However, their focus is on extracting heuristics from the source tasks, rather than explicitly reusing policies as we do.

Jiang and Sheng [2009] propose a *Case-based Reinforcement Learning* algorithm for dynamic inventory control. Their focus is on combining *Case-based Reinforcement Learning* with multi-agent systems under non-stationary conditions. Fang and Wong [2010] also focus their efforts on a *Supply Chain Management* application, where they use a case base to support adaptive negotiation strategies. Both approaches are concerned with a multi-agent setting while we are concerned with a single agent.

Domínguez-Estévez et al. [2017] use *CBR* to replace the *Q-table* in the classic *Q-Learning* algorithm focusing on a single video game, *Ms. Pac-Man vs. Ghosts*. This approach focuses on reusing knowledge within a single task while our goal is to facilitate learning across tasks.

Zhao et al. [2017] propose a framework for the generation and evolution of software adaptation rules, improving existing rule-based adaptation approaches in the flexibility of the adaptation logic and the quality of the adaptation rules. Bianchi et al. [2018] accelerate learning in robot soccer with small-scale robots and humanoid-robot stability learning by using *CBR* as heuristics within a *TL* setting to accelerate *RL*. While those approaches address learning over tasks their focus is rather on the application of classic *RL* algorithms than *DRL*, as we do.

To the best of our knowledge there has been no work combining *CBR* and *DRL* in the same way that we do, as a means to systematically build up a library of specific knowledge that can be reused for learning new tasks faster and better.

Chapter 3

Proposal

In this chapter, we provide the motivation for our research proposal (Section 3.1), describe the *CBR* framework in *RL* terminology (Section 3.2), and formulate the proposed algorithm based on our efforts (Section 3.3).

3.1 Motivation

Pure *RL* algorithms generate excellent results for single task learning with successes in low dimensional state and action spaces in the classic domains but with the use of *DNN* architectures also increasingly powerful algorithms for high dimensional problems. Still, these algorithms suffer from *task specificity*, *sample inefficiency*, and *high volatility in learning performance*. *Task specificity* here means that a learned model can in general only be applied to one specific task in a fixed setting and does not generalize well to new tasks or unseen challenges. *Sample inefficiency* refers to the fact that *RL* algorithms need to see a high number of samples before significant learning happens, extending the computational requirements and making the whole process very time consuming. *High volatility in learning performance* is based on observations that show that different implementations of the same algorithm using the same hyper parameters can lead to vastly different results where even the influence of the random seed can be significant [Henderson et al., 2017].

Specifically, the need of having to learn every new task from scratch seems

unnecessary considering that we are trying to build more human-like learning agents since humans barely learn anything from scratch from the minute they are born. The introduced technologies all have their advantages and might be the single right choice for some specific problem but few of them consider a systematic approach for knowledge building and conservation.

In this work, we are proposing a systematic approach for knowledge building and conservation based on the combination of *DRL* with the *CBR* methodology for *TL*. This approach addresses two of the challenges mentioned here, *task specificity* and *high volatility in learning performance*. We will show that we can massively speed up learning of a new task using the algorithm based on our framework while at the same time reducing the volatility when training with different random seeds and on different hardware.

3.2 Formulating CBR in RL terminology

In this section we provide our interpretation of the *CBR* framework in *RL* terminology. As is common for subfields of a research area, *CBR* has evolved its own terminology and perspectives, making it less tangible for other communities. Therefore, we attempt to provide a formulation that defines the *CBR* framework using *RL* terminology with the goal of making it more accessible for the *RL* community and making *RL* more attractive to the *CBR* community.

In *CBR* terminology, a *case* describes a problem and its solution. A *case base* is comprised of a number of retained *cases*, which have been learned in the past and whose solutions can be reused to solve future problems.

In *RL*, a *case* ϑ contains the task Ω_ϑ (problem) and a policy π_{Ω_ϑ} (solution) associated with that task,

$$\vartheta := \langle \Omega_\vartheta, \pi_{\Omega_\vartheta} \rangle. \quad (3.1)$$

Instead of saving a policy directly it is also possible to save a representation of the *state-action* values which provides a quality estimate for every possible action a in a given state s , $\vartheta := \langle \Omega_\vartheta, Q_{\Omega_\vartheta} \rangle$. The policy can then, for example, simply be defined by choosing the action with the highest *Q-value*. For the remainder of this section, we will use the term *policy* regardless if we are dealing with a

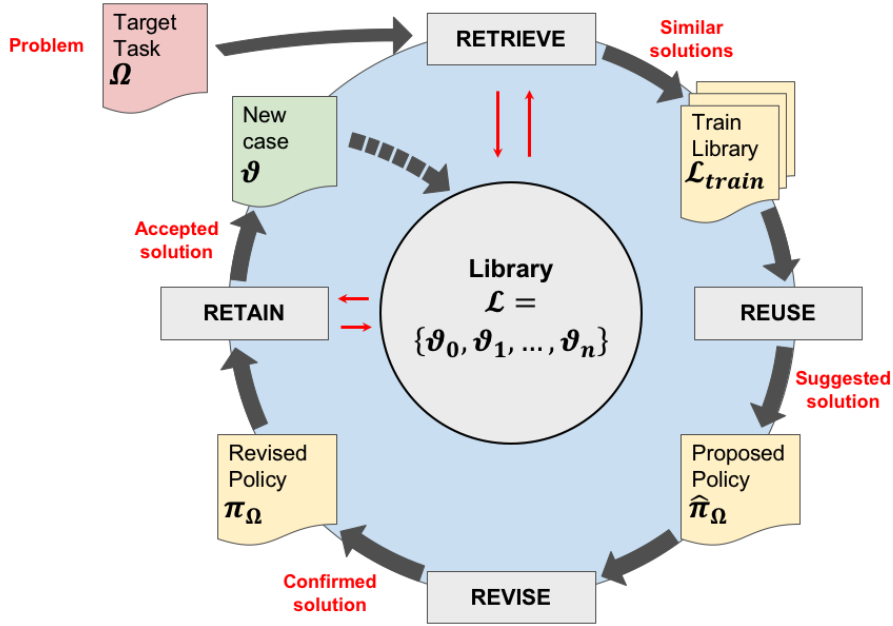


Figure 3.1: A view on the CBR cycle adapted to RL terminology. Source: Author.

value- or *policy-based* approach, because we can easily derive the policy from the *value-based* approach by using $\pi_{\Omega_\theta} = \underset{a}{\operatorname{argmax}} Q_{\Omega_\theta}$.

A *case base* can be defined as a *library* in RL and formally described as

$$\mathcal{L} := \{\vartheta_0, \vartheta_1, \dots, \vartheta_n\}, \quad (3.2)$$

where the ϑ_i stand for individual *cases* that have successfully been solved and added to the library. Each of those cases is ideally distinct from each other to avoid duplicate information and uncontrolled growth of the library. A visualization of the learning process is shown in Figure 3.1.

When a new (target) task Ω is presented to the agent, a set of relevant existing cases \mathcal{L}_{train} is selected in the *RETRIEVE* phase. \mathcal{L}_{train} is then processed during the *REUSE* phase and a policy $\hat{\pi}_\Omega$ is suggested which will be fine-tuned during the *REVISE* phase to produce the finally learned policy π_Ω . When the final policy is achieved, we decide in the *RETAIN* phase if Ω and π_Ω compose an interesting case that should be stored for posterior use. We are leaving the definitions intention-

ally wide here so that adaptations to other variants are possible, as for example by Aha [1998a], who introduces an additional *REVIEW* step between *REVISE* and *RETAIN*, or by Hullermeier [2007] who also describes a framework but focuses on the *RETAIN* stage and proposes to build a *credible* set that contains some (possibly all) solutions for a given problem. In the next subsections we further detail each step.

3.2.1 RETRIEVE

Not every case in \mathcal{L} is expected to be useful when learning a new task Ω . Ideally, a more confined subset $\mathcal{L}_{train} \subseteq \mathcal{L}$ should be extracted, in which the task of each selected case is (i) similar to Ω , so as to enable the agent to find commonalities between tasks and profit from knowledge reuse, and (ii) dissimilar from the other selected cases, so as to avoid ambiguities and redundant knowledge. Also \mathcal{L}_{train} can be limited by a maximum number of elements l_{max} to accommodate for computational restrictions.

Defining the similarity measures between cases is a difficult step and is always also dependent on the context of the agent and the task [Chazara et al., 2016]. For convenience, usually a domain specific similarity function sim_{task} and a similarity threshold $\sigma_{task} \in [0, 1]$ are defined to help fulfill the given criteria, with

$$sim_{task} : \Omega \times \Omega \rightarrow [0, 1]. \quad (3.3)$$

The main purpose of the *RETRIEVE* stage (Algorithm 2) is properly selecting the training library \mathcal{L}_{train} , with

$$\mathcal{L}_{train} \leftarrow RETRIEVE(\mathcal{L}, \Omega, sim_{task}, \sigma_{task}, l_{max}). \quad (3.4)$$

3.2.2 REUSE

In this stage, the main goal of the agent is to use \mathcal{L}_{train} to achieve a satisfactory performance in Ω as fast as possible. However, reusing past policies is not simple and has been a long-studied challenge for the *RL* community [Koga et al., 2015,

Algorithm 2 *RETRIEVE*

Require: Case library \mathcal{L} , Target task Ω , Task similarity function sim_{task} , Task similarity threshold σ_{task} , Maximum training library size l_{max}

- 1: $\mathcal{L}_{train} \leftarrow \emptyset$
 - 2: **for** $\forall \vartheta \in \mathcal{L}$ **do**
 - 3: **if** $sim_{task}(\Omega, \Omega_{\vartheta}) \geq \sigma_{task}$ **then**
 - 4: $\mathcal{L}_{train} \cup \{\langle \Omega_{\vartheta}, \pi_{\Omega_{\vartheta}} \rangle\}$ sorted by similarity
 - 5: **if** $|\mathcal{L}_{train}| > l_{max}$ **then**
 - 6: Remove the last $|\mathcal{L}_{train}| - l_{max}$ cases from \mathcal{L}_{train}
 - 7: **return** \mathcal{L}_{train}
-

Fernández and Veloso, 2006]. Even if only good cases are selected for \mathcal{L}_{train} , the agent still needs to decide when their policies are expected to be useful, which is not easy to define because the agent does not have a complete model of the target task. Often, a good strategy is to only use a source policy where the solved task has commonalities with the new one.

The main purpose of the *REUSE* stage is to exploit the knowledge in the provided source policies from \mathcal{L}_{train} to discover a policy $\hat{\pi}_{\Omega}$ that already performs well on the target task,

$$\hat{\pi}_{\Omega} \leftarrow REUSE(\mathcal{L}_{train}, \Omega). \quad (3.5)$$

Notice that $\hat{\pi}_{\Omega}$ represents the behavior that the agent will start to follow. This policy might be induced, for example, by reusing multiple source policies alternatively [Fernández and Veloso, 2006].

3.2.3 REVISE

Although some of the cases in the library may contain very similar tasks, none of them are expected to be exactly equal to the new target task. For this reason, selecting one (or multiple) policies to guide the training is a good way to find a reasonable policy fast, but this needs to be followed by further training and fine-tuning to adapt the suggested policy for the current target task Ω . Therefore, a learning algorithm should update the policy $\hat{\pi}_{\Omega}$ given by the *REUSE* method to

further improve it,

$$\pi_{\Omega} \leftarrow REWISE(\Omega, \hat{\pi}_{\Omega}). \quad (3.6)$$

Here, π_{Ω} resolves to the optimal policy π_{Ω}^* given sufficient training time because the training is not influenced by other policies any more and has the same convergence properties as regular *RL* approaches for single task learning. Depending on the used algorithm, it often makes sense to have the *REUSE* and *REWISE* stages blend into each other and have some selection criteria that gradually switches the guiding policy from past experiences to the new policy, as we propose with our algorithm in the next section.

3.2.4 RETAIN

When the training for Ω ends (whether by finding the optimal policy or by meeting a termination condition), it must be decided if the new case $\vartheta = \langle \Omega, \pi_{\Omega} \rangle$ should be added to the general library \mathcal{L} . This is achieved by comparing the new case with the previous ones in the library. A new case is only added if its policy is sufficiently different from the previous ones (i.e. will add new knowledge into the library). The comparison might be performed using a function to determine the similarity between policies sim_{policy} and a similarity threshold $\sigma_{policy} \in [0, 1]$,

$$sim_{policy} : \Pi \times \Pi \rightarrow [0, 1], \quad (3.7)$$

with higher values meaning greater similarity. The comparison is only performed on the previously selected cases for \mathcal{L}_{train} to make sure we are not comparing to tasks that are not similar which might lead to catastrophic failure.

The main objective of the *RETAIN* stage (see Algorithm 3) is to select and maintain relevant knowledge by updating the library \mathcal{L} by comparing the new policy with existing policies,

$$\mathcal{L} \leftarrow RETAIN(\mathcal{L}, \mathcal{L}_{train}, \langle \Omega, \pi_{\Omega} \rangle, sim_{policy}, \sigma_{policy}). \quad (3.8)$$

Algorithm 3 *RETAIN*

Require: Case library \mathcal{L} , Training library \mathcal{L}_{train} , New case $\langle \Omega, \pi_\Omega \rangle$, Policy similarity function sim_{policy} , Policy similarity threshold σ_{policy}

- 1: **for** $\forall \vartheta \in \mathcal{L}_{train}$ **do**
- 2: **if** $sim_{policy}(\pi_\Omega, \pi_{\Omega_\vartheta}) \geq \sigma_{policy}$ **then**
- 3: **return** \mathcal{L}
- 4: **return** $\mathcal{L} \cup \{\langle \Omega, \pi_\Omega \rangle\}$

3.2.5 Summary

The whole *CBR* cycle can then be described in our framework using the formulations from above as shown in Algorithm 4. The library can initially already contain cases $\mathcal{L} = \{\vartheta_0, \vartheta_1, \dots, \vartheta_n\}$ or be empty $\mathcal{L} = \{\}$. The similarity functions for the *RETRIEVE* (sim_{task}) and *RETAIN* stages (sim_{policy}) as well as the respective similarity thresholds (σ_{task} and σ_{policy}) are domain specific, while the maximum size of the training library l_{max} is often restricted by hardware constraints.

Algorithm 4 *CBR* using *RL* notation

Require: Case library \mathcal{L} , Target task Ω , Task similarity function sim_{task} , Task similarity threshold σ_{task} , Policy similarity function sim_{policy} , Policy similarity threshold σ_{policy} , Maximum training library size l_{max}

- 1: $\mathcal{L}_{train} \leftarrow RETRIEVE(\mathcal{L}, \Omega, sim_{task}, \sigma_{task}, l_{max})$
- 2: $\hat{\pi}_\Omega \leftarrow REUSE(\mathcal{L}_{train}, \Omega)$
- 3: $\pi_\Omega \leftarrow REVISE(\Omega, \hat{\pi}_\Omega)$
- 4: $\mathcal{L} \leftarrow RETAIN(\mathcal{L}, \mathcal{L}_{train}, \langle \Omega, \pi_\Omega \rangle, sim_{policy}, \sigma_{policy})$

3.3 Proposing a novel algorithm: DECAF

Here, we propose an implementation of the framework adapted for learning with high-dimensional input states and name it *Deep Case-based Policy Inference* or in short *DECAF*.

We focus on the case where the observation space and the available action space remain the same across all tasks under consideration. If the observation or

action space change across tasks, an inter-task mapping could be used to apply our method [Taylor et al., 2007, Silva and Costa, 2017]. But one of the advantages of our method proposed here is that the algorithm would be able to discard unhelpful knowledge when selecting previously learned cases for the training library, avoiding negative influence on the training process even if no inter-task mapping is provided.

The complete learning process is described in Algorithm 5. It requires as input the existing library \mathcal{L} , the target task Ω that we want to solve, a task similarity function sim_{task} for selecting tasks, a task similarity threshold factor σ_{task} to select only sufficient similar tasks, a policy similarity function sim_{policy} for evaluating the new policy, a policy similarity threshold factor σ_{policy} to avoid having duplicate knowledge in the library, and the maximum number of similar cases l_{max} that we consider for the training library \mathcal{L}_{train} .

The algorithm describes how knowledge is selected from the existing case library, how it is applied to learn the new task, and how the case library is eventually updated. During the training process, the agent keeps performing training episodes until a stopping condition is reached for each episode (either by reaching a maximum number of steps per episode or reaching a terminal condition) and eventually for the whole training process (in our case we used a maximum number of training steps). At each step, the agent performs an update of the network parameters θ_{Ω} and θ_w of the online networks by using random samples drawn from the replay memory \mathcal{M} in the form of mini batches. The weights of the target networks θ_{Ω}^- and θ_w^- , that are used for the training target $Y_{train}(s', r; \theta_{\Omega}^-, \theta_w^-)$ when updating the online network parameters, are only updated after a given step interval to provide a more stable training signal.

The *RETRIEVE* (L.1) and *RETAIN* (L.17) stages are domain specific and should be optimized accordingly. As described in the previous section, the *RETRIEVE* stage (L.1) relies on a similarity function sim_{task} for selecting previous cases that are similar to the current task. In the experimental evaluation section we will provide more details on this function for our target domain.

We here combine the *REUSE* and *REVISE* stages by performing a gradual transition between them, rather than defining explicit hard transitions (L.4-15). This idea is based on the *A2T* network proposed by Rajendran et al. [2017] which

Algorithm 5 Learning a task with *DECAF*

Require: Case library \mathcal{L} , Target task Ω , Task similarity function sim_{task} , Task similarity threshold σ_{task} , Policy similarity function sim_{policy} , Policy similarity threshold σ_{policy} , Maximum training library size l_{max}

- 1: $\mathcal{L}_{train} \leftarrow RETRIEVE(\mathcal{L}, \Omega, sim_{task}, \sigma_{task}, l_{max})$
 - 2: Randomly initialize online networks: $Q_{\Omega}(s, a; \theta_{\Omega}), \mathbf{w}(s, a; \theta_{\mathbf{w}})$
 - 3: Set target network parameter: $\theta_{\Omega}^- \leftarrow \theta_{\Omega}, \theta_{\mathbf{w}}^- \leftarrow \theta_{\mathbf{w}}$
 - 4: **while** training **do**
 - 5: Restart environment and observe starting state s
 - 6: **while** Episode **do**
 - 7: Select and perform a based on $\pi_{train}(s; \theta_{\Omega}, \theta_{\mathbf{w}})$ (Equations 3.9, 3.10)
 - 8: Observe reward r and follow-up state s'
 - 9: Save experience $\langle s, a, r, s' \rangle$ in replay memory \mathcal{M} minibatch from \mathcal{M}
 - 10: Update training target $Y_{train}(s', r; \theta_{\Omega}^-, \theta_{\mathbf{w}}^-)$ (Equation 3.11)
 - 11: Update importance network $\mathbf{w}(s, a; \theta_{\mathbf{w}})$ (Equation 3.12)
 - 12: Update training target network $Q_{\Omega}(s, a; \theta_{\Omega})$ (Equation 3.13)
 - 13: **if** Update interval **then**
 - 14: Set target network parameter: $\theta_{\Omega}^- \leftarrow \theta_{\Omega}, \theta_{\mathbf{w}}^- \leftarrow \theta_{\mathbf{w}}$
 - 15: $s \leftarrow s'$
 - 16: $\pi_{\Omega} \leftarrow \pi_{train}$
 - 17: $\mathcal{L} \leftarrow RETAIN(\mathcal{L}, \mathcal{L}_{train}, \langle \Omega, \pi_{\Omega} \rangle, sim_{policy}, \sigma_{policy})$
-

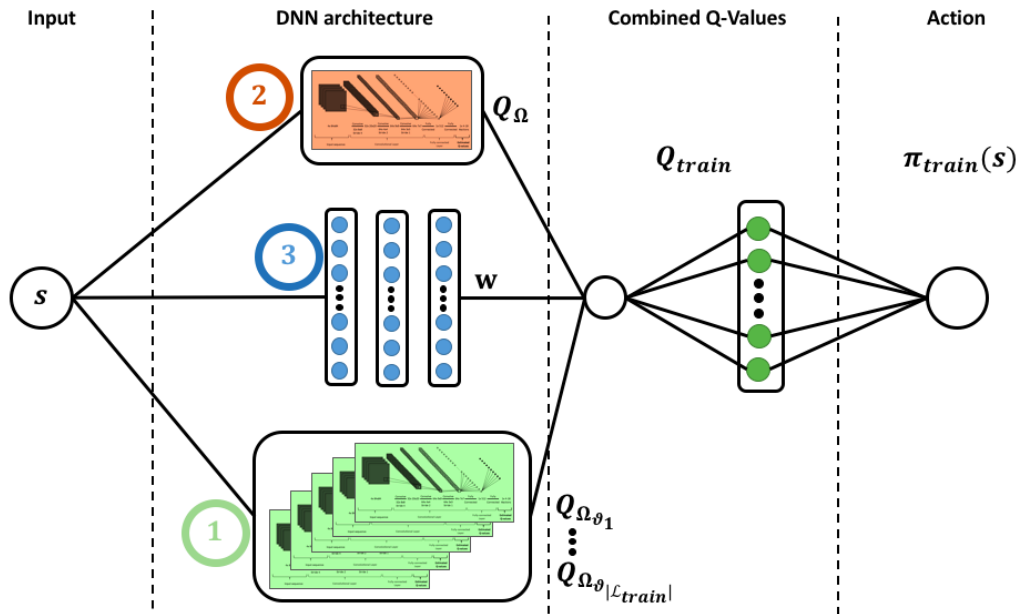


Figure 3.2: Sketch of network architecture that produces the guiding training policy $\pi_{train}(s, \cdot)$ for a given input state s : (1) Selected source networks $Q_{\Omega_{\theta}}(s, \cdot)$ from \mathcal{L}_{train} , (2) network $Q_{\Omega}(s, a; \theta_{\Omega})$ that we want to learn, and (3) importance network $w(s, a; \theta_w)$. Source: Author.

is also used as a baseline in the experimental evaluation of this algorithm in Section 4.3.

The learning architecture for the new task is described by a *DNN* architecture that consists of three pillars that are combined at the top to produce the training policy $\pi_{train}(s, \cdot)$ that guides the agent during training (Figure 3.2). The input of the architecture is the state representation s that is given to all three pillars.

The first pillar consists of all the $|\mathcal{L}_{train}|$ source values $Q_{\Omega_{\theta}}(s, \cdot)$ that were selected according to their task similarity to be helpful during training. In our experiments, we are also using neural networks and the weights remain fixed during training. In general, those source networks could be represented in different ways as long as they have the same input and output shape as required by the domain, but we here use the same network architecture for all policies.

The second pillar consists of the new network $Q_{\Omega}(s, a; \theta_{\Omega})$ that we want to

train in the form of the *DQN* with *online* and *target* network. The weights of the network are initialized randomly at the beginning of the training (L. 2). During evaluation periods, the agent only uses the output of this pillar to decide its actions and ultimately, the *online* network for this pillar is the one that will be added to the library if it is found to improve the knowledge base.

The third pillar consists of another single network, in the following referred to as *importance network*, that learns a weight vector $\mathbf{w}(s, a; \theta_{\mathbf{w}})$ with individual output values w_j for each of the $|\mathcal{L}_{train}|$ source networks $Q_{\Omega_j}(s, a)$ ($j = 1 \dots |\mathcal{L}_{train}|$) and the new network $Q_{\Omega}(s, a; \theta_{\Omega})$ ($j = 0$). The difference to the other networks is that the final layer additionally applies a softmax function to the output to make sure it generates values with $\forall j : w_j \in [0, 1], \sum_j w_j = 1$. We interpret those values as a measure for the usefulness of each policy with respect to the other available policies, and expect the network to learn when to use which policy for guidance during training. The weights of this network are also initialized randomly at the beginning of the training (L. 2).

The final *Q-values* of the architecture during training are then a combination of the three pillars. Each network output of the first and second pillar is weighted with the respective w_j value from the importance network and combined to

$$Q_{train}(s, \cdot) = Q_{\Omega}(s, a; \theta_{\Omega})w_0(s, a; \theta_{\mathbf{w}}) + \sum_{j=1}^{|\mathcal{L}_{train}|} Q_{\Omega_j}(s, \cdot)w_j(s, a; \theta_{\mathbf{w}}), \quad (3.9)$$

from which the guiding policy $\pi_{train}(s, \cdot)$ is derived by choosing the action with the highest *Q-value*,

$$\pi_{train}(s, \cdot) = \arg \max_{a \in A} Q_{train}(s, \cdot). \quad (3.10)$$

The training loop starts with the initialization of a new episode and the observation of the starting state s (L.5). During an episode (L.6-15), the agent first selects and performs an action a based on $\pi_{train}(s, \cdot)$ (L.7) after which it observes a reward r and the follow-up state s' (L.8). The agent then saves this experience as tuples of $\langle s, a, r, s' \rangle$ in a *replay memory* \mathcal{M} (L.9). As described, the *replay memory* is a way to select the samples for the network parameter updates ran-

domly from uncorrelated experiences to stabilize learning instead of taking the experiences directly from the *online* (acting) agent.

The networks are updated at every step using mini batches from the replay memory by making the according calculations of the losses for every experience in the batch and then taking the averages over the batch (L.9-12). A training signal (or training target) $Y_{train}(s', r; \theta_{\Omega}^-, \theta_{\mathbf{w}}^-)$ is calculated here as our best estimation of future rewards considering the reward we received and acting optimally from there on (according to our current beliefs) which is different from supervised-learning, where we have a true target value that is the absolute truth and not changing. The network parameters θ^- of the *target* network are updated only periodically with the parameters θ from the *online* network (L. 3,14). The training target for the updates is defined as

$$Y_{train}(s', r; \theta_{\Omega}^-, \theta_{\mathbf{w}}^-) = (r + \gamma \max_{a'} Q_{train}(s', a'; \theta_{\Omega}^-, \theta_{\mathbf{w}}^-)). \quad (3.11)$$

The respective loss functions for the online network parameter θ are evaluated with respect to their target parameters θ^- and calculated as

$$L_{\mathbf{w}}(\theta_{\Omega}, \theta_{\mathbf{w}}) = \mathbb{E}[(Y_{train}(s', r; \theta_{\Omega}^-, \theta_{\mathbf{w}}^-) - Q_{train}(s, a; \theta_{\Omega}, \theta_{\mathbf{w}}))^2] \quad \text{and} \quad (3.12)$$

$$L_{\Omega}(\theta_{\Omega}, \theta_{\mathbf{w}}) = \mathbb{E}[(Y_{train}(s', r; \theta_{\Omega}^-, \theta_{\mathbf{w}}^-) - Q_{\Omega}(s, a; \theta_{\Omega}))^2]. \quad (3.13)$$

The gradients $\nabla_{\theta_{\Omega}} L_{\Omega}(\theta_{\Omega}, \theta_{\mathbf{w}})$ and $\nabla_{\theta_{\mathbf{w}}} L_{\mathbf{w}}(\theta_{\Omega}, \theta_{\mathbf{w}})$ are then, respectively, used for the parameter updates of the neural networks.

Performing the training updates in this way drives the *importance network* to learn which source policy is good for which state, while at the same time improving the target policy by imitating the source policies in that state. As the current *online* network is getting better over time, the *importance network* realizes this and gradually gives more weight to it. This is used to balance the trade-off between exploration and exploitation, using the guidance of the previously trained networks until the new policy is consistently the most useful for the current task.

After the updates, the current state s is set to the follow-up state s' to initialize the next step of the episode (L.15). This training loop is executed until a

stopping criterion is reached, which could be a maximum number of steps or the termination of the episode by reaching a goal state.

Finally, in the *RETAIN* phase, the newly trained network $Q_{\Omega}(s, a; \theta_{\Omega})$ is compared against the policies of the similar tasks that were selected for \mathcal{L}_{train} to decide if the new case $\vartheta_{\Omega} = \langle \Omega, Q_{\Omega}(s, a; \theta_{\Omega}) \rangle$ should be added to the library \mathcal{L} (L.17). We will also give more information about this procedure in the experimental evaluation section regarding our domain specific measure.

3.4 Discussion of related work

The creation of a policy library to reuse previously learned findings has been investigated in the past already. Systematic approaches that rely on *CBR* have already been discussed in Section 2.3.2. We here introduce some of the most related works and provide a differentiation to our approach that are taken from the general *RL* literature. These works focus more on the *RL* aspect without explicitly considering the *CBR* methodology.

One of the few works that uses a policy library during training is from Fernández and Veloso [2006] who introduced the *Policy Library Policy Reuse (PLPR)* algorithm that we will also use in the experiment section (Section 4.2) to compare against our framework. In their work, a library of core policies for a given domain is autonomously built and reused. While they select a whole policy that they follow for a certain amount of time during training, we estimate the usefulness of policies and mix the policies during training instead of following a single policy per episode.

Koga et al. [2015] propose to blend multiple policies into a single abstract policy, which is used at the beginning of learning in any new task (whether the new task is similar to the source tasks or not). In spite of following a similar idea, *DECAF* stores multiple concrete policies, and selects only the most promising ones by taking similarity with the target task into consideration.

Sinapov et al. [2015] evaluate user-defined task features so as to enable the estimation of the similarity between tasks and choose only the most similar one(s) for the target task. However, they are more focused on source-task selection,

while we focused on providing a consistent framework to select and reuse the source policies as good as possible for continuously building the knowledge base.

In the *DRL* domain, an interesting network architecture has been proposed by Rusu et al. [2016]. Their *Progressive networks* introduce lateral connections to previously learned features in parallel networks and are able to reuse previous knowledge while being immune to catastrophic forgetting. While their approach works well, they are also not considering similarity to previously learned tasks and all previous knowledge has to be used for training, making it impossible to scale to more than a few tasks.

Rosman et al. [2016] also propose an algorithm for *policy reuse* that selects policies during training according to an Bayesian belief over the nature of the task build from offline-captured correlations, continuously updating the belief state to better select a policy to execute in the next episode. While this approach takes a similar stance on selecting only useful source tasks for training, the difference lies in the fact that only one policy is selected during each episode.

Rajendran et al. [2017] introduced *Attend, Adapt and Transfer (A2T)*, which we have used as a benchmark for our algorithm and also implemented within the *DECAF* framework. While *A2T* presents a good approach for reusing past policies, it has insufficient protection against reusing disadvantageous previous knowledge and also does not provide a way to only consider core policies for addition to the library. We, on the other hand embed *A2T* in the *CBR* cycle and augment it with the ability to do these things, making it more robust and usable for a wider range of domains, improving and speeding up training while reducing memory requirements.

Chapter 4

Experiments

In this chapter we describe the conducted experiments and their results. We first describe the initial investigations on *direct parameter transfer* in *DQN* (Section 4.1), then the first steps with our proposed framework for knowledge transfer (Section 4.2), and finally the experiments using the full *DECAF* algorithm (Section 4.3).

4.1 Parameter initialization for agent networks

In this section we will present the experiments we conducted in the area of *direct parameter transfer* in *DRL* which was one of the first published works in the area [Glatt et al., 2016].

The *DRL* approach has its application in high-dimensional input spaces and has been used on visual domains since its introduction [Mnih et al., 2013], of which computer games represent a suitable playground for the development and comparison of algorithms and agents. Here, we used the Atari emulator ALE by Bellemare et al. [2013] to analyse our algorithms in the game playing domain.

In the Atari game playing domain the agent controls the actions while playing a selected game with the goal of maximizing the game score. For each game the agent can only use the actions that are available for the specific game, which represent a subset of $A = \{ \text{'NOOP'}, \text{'FIRE'}, \text{'UP'}, \text{'RIGHT'}, \text{'LEFT'}, \text{'DOWN'}, \text{'UPRIGHT'}, \text{'UPLEFT'}, \text{'DOWNRIGHT'}, \text{'DOWNLEFT'}, \text{'UPFIRE'}, \text{'RIGHTFIRE'}, \text{'LEFTFIRE'}, \text{'DOWNFIRE'} \}$,

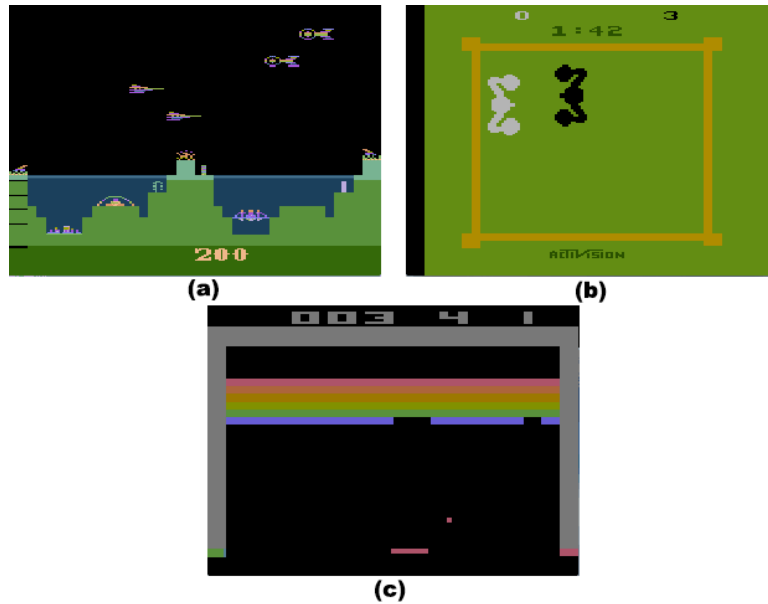


Figure 4.1: The games for which policies were trained: (a) Atlantis, (b) Boxing, and (c) Breakout. Source: Screenshot from Bellemare et al. [2013].

'UPRIGHTFIRE', 'UPLEFTFIRE', 'DOWNRIGHTFIRE', 'DOWNLEFTFIRE'}). To keep in line with the nomenclature in the Atari game literature we refer to one game (finishes after loss of all lives if more than one) as one episode and to the achieved score per game step as the reward per game step. The state space in such games is high dimensional, which is the reason a *DQN* is used as an approximation for the state instead of saving all different states in a *Q-table*.

We limited our experiments to the Atari games *Atlantis*, *Boxing*, and *Breakout*, shown in Figure 4.1, to provide us with a controlled scenario and ease the analysis of our results. A reason for the limited observation range was also the difficulty of training combined with the long computing cycles while having only access to very limited computational resources at the time.

Our ultimate goal here is to evaluate the possible effects of *TL* in the form of *parameter transfer* on *DRL*. For this, we first investigated different training architectures and parameter optimization methods to select the best one for the second phase, where we actually perform the knowledge transfer.

The first phase intended to verify if the choice of optimization method or the

number of output nodes would have a greater impact on the training of a *DQN*. We compared two state-of-art optimization methods, *RMSProp* [Dauphin et al., 2015] and *ADAM* [Kingma and Ba, 2014], and chose the game *Breakout* as the benchmark game for this phase, because we achieved the most stable training results with this game in initial training runs. The *DQN* can be trained using only the actions that are available for the specific game (6 for *Breakout*), as in Mnih et al. [2015], or use all 18 available actions of the Atari controller. Here, we compared *RMSProp* both with 6 and 18 available actions and *ADAM* with only 6 available actions.

The second phase of our experiments simulated the following situation: A source task is given to a learning agent which is initialized randomly before the training starts. After learning an effective policy, a new target task is presented to the agent. Even though the agent has no information about the new task, we retain the previous knowledge and start training in the target task with the parameters learned when we ended training in the source task. In order to evaluate the efficiency of *TL* in different situations, where we use source tasks that have different degrees of similarity, we firstly trained a *DQN* for *Breakout* from scratch (network parameter initialized randomly) and then compared the learning results under the following scenarios:

1. *TL from similar tasks*: *TL* is expected to present better results when the source and target tasks are very similar. In order to simulate this situation, we performed a new training phase for *Breakout* initializing the *DQN* with the parameters of a previously trained *DQN* also on *Breakout* (highest possible similarity).
2. *TL from neutral tasks*: While in *Atlantis* an optimal policy can be achieved by only using the *fire* actions, they are mostly useless in *Breakout* (but they do not lead to terrible situations either). We here evaluate *TL* when the source task is different from the target task, but the optimal actuation in the source task is not expected to be worse than a random policy. After training a *DQN* for the game *Atlantis*, we used its parameters as initialization to train a new *DQN* for *Breakout*.

3. *TL from different tasks*: *TL* is reported to result in negative transfer when applied carelessly. We here evaluate the learning performance when starting with a very bad policy. The game *Boxing* offers a very different game-play than *Breakout*, and also has very different outcomes when using the same actions. So we trained a *DQN* in *Boxing* and used it to train a new *DQN* for *Breakout*.

In all experiments in this section, an epoch consists of 250.000 training steps followed by 125.000 steps for evaluation. Our graphs show the number of episodes per training epoch (left), where a lower number is favorable (not dying very often), and the average reward per episode per epoch (right), where a higher value is favorable, as a running average over 5 epochs.

4.1.1 Importance of optimization method and network architecture

The results for the first phase are shown in Figure 4.2, where we compared the three configurations over 100 epochs. One can see that they all have similar behaviour.

The *ADAM* optimizer seems to learn better at the very beginning of training, rising faster on the average game reward while learning not to die earlier. Later, between epochs 10 and 15, *RMSProp* catches up in both configurations and remains the algorithm with higher average reward and fewer loss of live for the rest of the 100 epochs. Interestingly, *ADAM* also seems to converge faster but at a much lower average reward indicating that the learner got stuck in a local minima from which it did not manage to escape in this experiment.

For *RMSProp*, the distinction between the number of available actions does not have a significant impact on the overall maximum reward but the volatility during training seems to be a little higher when using all available actions instead of only the game relevant actions. Comparing the training times for the number of actions in the network showed that, in this case, the additional computations were not significant enough to have an impact on the duration of the training time.

Based on these results we chose the *RMSProp* optimizer for the second phase

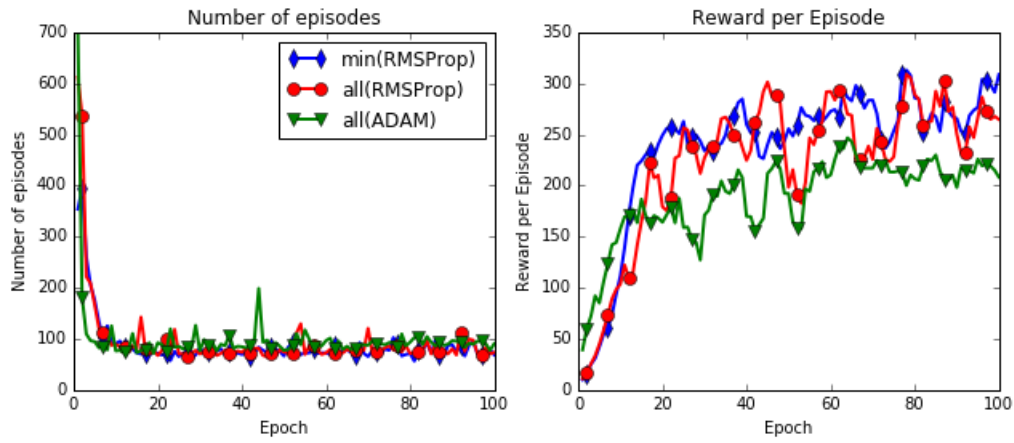


Figure 4.2: Comparison between different settings for a DQN trained for the game Breakout. Source: Author.

of the experiment because of the higher rewards it achieved. For practical reasons, we also chose to use all available actions for the networks in the second phase to avoid having to find a fitting mapping between the available actions for each game.

4.1.2 Importance of parameter initialization

Figure 4.3 shows the results for the scenario where we have expectatly favorable parameters for the network initialization. The achieved rewards per episode in the first epochs are much greater when starting with the transferred DQN , while the number of episodes per epoch has already decreased to a good level, meaning that fewer lives are lost during playing the game. Since we here used knowledge from the same game for the initialization this *jumpstart* effect was expected and we show these results to indicate the potential of TL when transferring knowledge across very similar tasks. We can also see that training with a better initialized network has a stabilizing effect on the training results specifically towards the end of training.

Figure 4.4 depicts the results for the scenario where we have a task in which a similar actuation in the target task as in the source task is expected to lead to results

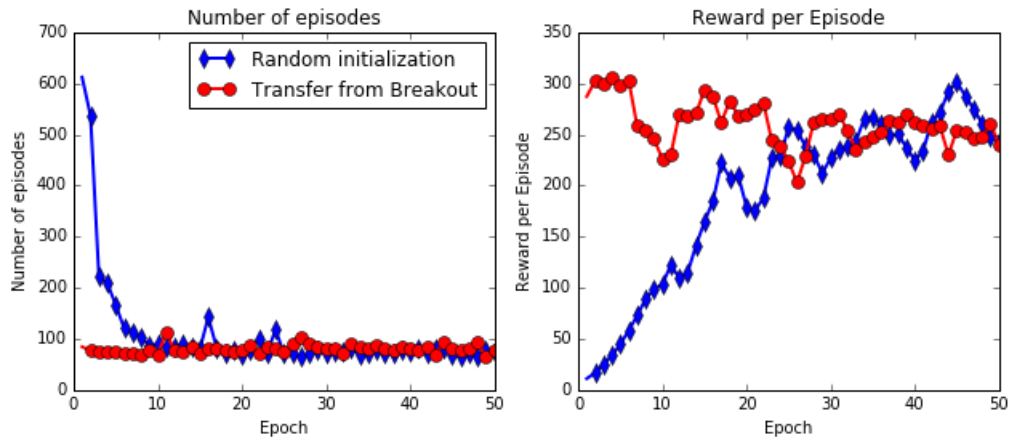


Figure 4.3: Comparison between random initialization of Breakout and initialization with the best performing network for a previously trained *DQN* for Breakout. Source: Author.

that are better than a random initialization. The minimum number of episodes and the maximum average reward per episodes are similar for both network initializations, indicating that the transferred knowledge did not help with the *DQN* training but also did not lead to worse performance compared to a random initialization. It is however notable, that again, the pretrained network seems to be more stable and shows a lower variance in the results.

Finally, Figure 4.5 presents the results for the third scenario where we have two tasks with very different strategies. In this case, it is clear that the *DQN* for the target task suffers from some kind of *negative transfer*. Apparently, the use of a very unfit policy when starting the learning process greatly hampered the *DQN* optimization, which can be seen in the average reward per episode which is much worse compared to the random initialization. Interestingly, the new agent also learned to survive quickly but remained constantly at a 10-15% worse performance of the randomly initialized agent. The influence seems so dominant that the agent is unable to recover from his bad performance until the end of the training process even though in the last episodes it finally seems to slowly start a positive development again.

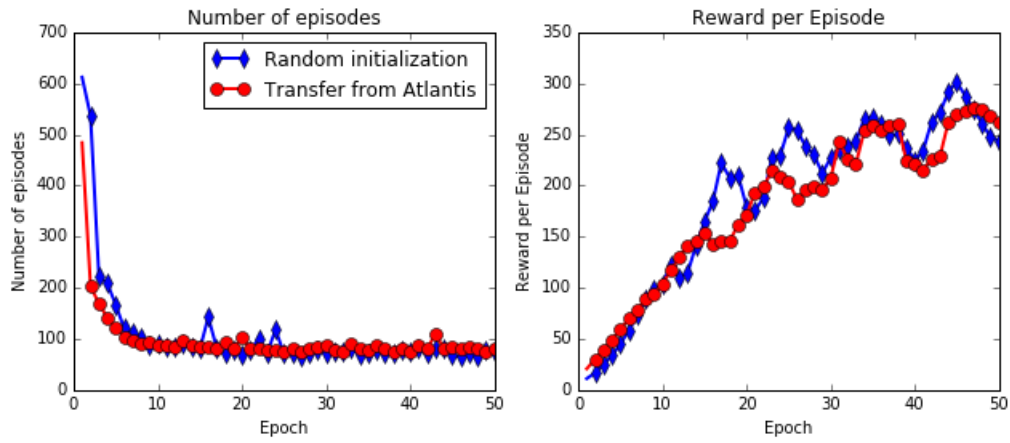


Figure 4.4: Comparison between random initialization of Breakout and initialization with the best performing network for a previously trained DQN for Atlantis. Source: Author.

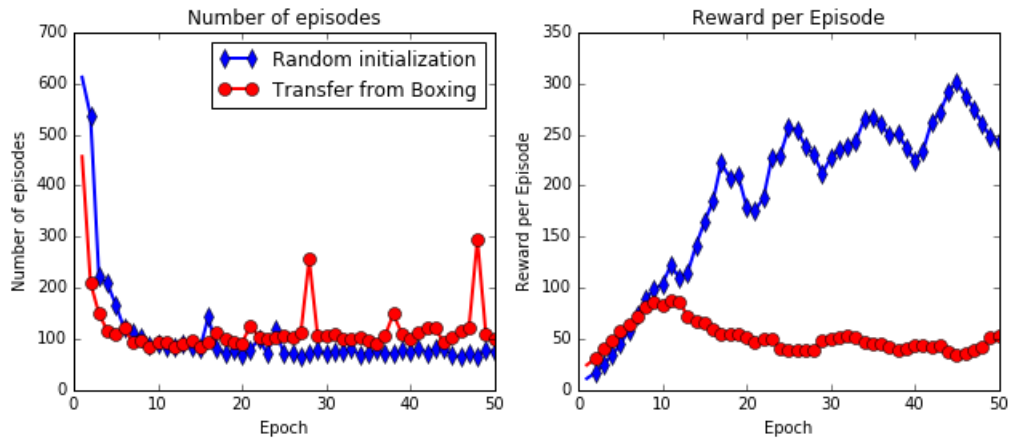


Figure 4.5: Comparison between random initialization of Breakout and initialization with the best performing network for a previously trained DQN for Boxing. Source: Author.

4.1.3 Discussion

Here, we evaluated the influence of different training settings on the learning success of *DQN* agents. Specifically, we looked at the optimization method, the number of available actions an agent uses, and the initialization of the network parameter, where we focused on the applicability of *parameter transfer* by measuring the performance in the learning process when transferring knowledge from source tasks with different degrees of similarity to the target task.

Our results show that the initialization of the *DQN* plays a far more important role than the choice of optimization algorithm or the number of available actions for the agent. We showed the potential for transferring knowledge from similar tasks which achieved a greatly accelerated learning process, realizing results closer to the optimal actuation already shortly after the beginning of training. However, when applied to unrelated tasks, there is a possibility for *negative transfer* from which it is very hard to recover confirming concerns presented from researchers when applying *TL* in the classic *RL* domain.

These results also indicate a great potential to accelerate learning if the source knowledge is carefully selected considering similarity between source and target task. However, copying weights in an unprincipled manner is not as successful as in *Supervised Learning* and is likely to lead to negative transfer, and thus we need a better framework for knowledge transfer.

4.2 First steps with CBR for RL

Here, we describe an implementation of the *CBR* methodology for a classic *RL* domain and propose the algorithm *Case-based Policy Inference (CBPI)* which is a framework for knowledge transfer that is able to deal with simple problem spaces [Glatt et al., 2017b].

Our agent is tasked with solving challenges in a gridworld domain, representing a simple navigation task based on an office layout with rooms that are connected through corridors. The agent has to learn how to get from any starting position in the environment to a target position by moving in any of the four directions (left, right, up, down). The state that the agent perceives are the coordinates

in the grid and at every step it receives a reward signal which is $R = 0$ unless it finds the goal state, where it receives a reward of $R = 1.0$. Different tasks share the same environment but can be distinguished by the position of the goal.

The similarity measure for the task selection that we use here is the *Euclidean distance* in the grid of the target positions of the pretrained task and the new task under the assumption that targets closer together represent similar tasks.

We compared our results with the standard *Q-Learning* algorithm for individual tasks and the *PLPR* algorithm proposed by Fernández and Veloso [2006]. *PLPR* calculates a *W-value* after every episode that indicates the usefulness of a policy for the current task according to the results achieved as running averages. At the beginning of every episode, the *W-values* are transformed into probabilities for each policy and then a single policy is chosen as guiding policy for the next episode based on this probability.

4.2.1 Learning a new policy

In this challenge, we evaluated two scenarios with the same training settings. In the first scenario, we pretrained an agent on a number of hand-selected target positions. We then evaluate if the transfer algorithm is able to select relevant previous policies and reuse them successfully if we use all available policies for the policy library, $\mathcal{L}_{1,2,3,4,5} = \{\pi_{\Omega_1}, \pi_{\Omega_2}, \pi_{\Omega_3}, \pi_{\Omega_4}, \pi_{\Omega_5}\}$ (see Figure 4.6, left).

In the second scenario, we evaluate the worst-case scenario, where all available policies are expected to hurt learning performance by deliberately choosing only policies from tasks that are unrelated to our target task, so the policy library becomes $\mathcal{L}_{2,3,4} = \{\pi_{\Omega_2}, \pi_{\Omega_3}, \pi_{\Omega_4}\}$ (see Figure 4.7, left).

Each curve in these experiments shows the average reward and confidence interval of the test episodes of 50 runs with 2000 training episodes with 10 test episodes after every training episode (where an episode ends when the goal state is reached or a maximum of 100 steps is performed).

As can be seen in Figure 4.6, both algorithms that reuse knowledge outperform regular *Q-Learning* by far but perform very similar to each other when using a library that also contains advantageous knowledge for the current task. It is notable, that our algorithm performs better at the beginning of the training, due to

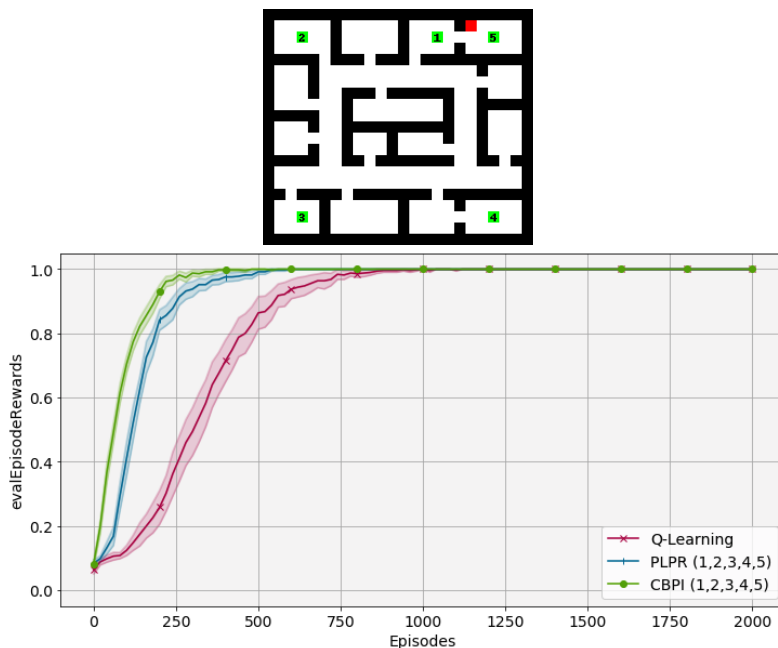


Figure 4.6: Resulting rewards (right) when using a policy library that also contains policies from very similar tasks, Ω_1 and Ω_5 (left). The red square represents the goal position in the target task. Source: Author.

the fact that it evaluates the available policies already the first time before it starts training, while *PLPR* starts randomly and learns its *W-values* for each policy on the fly.

In Figure 4.7, we can see the scenario where the library only contains disadvantageous knowledge. The *PLPR* approach uses all available tasks for its training library \mathcal{L}_{train} , while the similarity metric in *Case-based Policy Inference (CBPI)* detects that there are no similar tasks available, ignoring the available cases. Here, *CBPI* and *Q-Learning* perform equally while *PLPR* takes much longer to converge suffering from *negative transfer*.

Those results show that *CBPI* benefits from existing favorable knowledge while in the worst case performs as if no *a priori* knowledge was available. This behaviour indicates that it can better deal with the problem of *negative transfer* during training, as long as a well-defined similarity metric is given for the initial task selection.

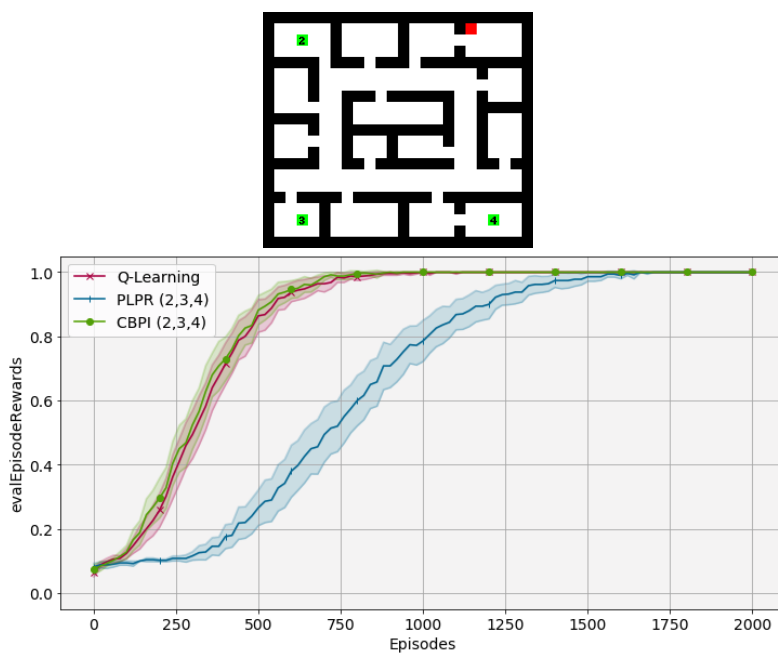


Figure 4.7: Resulting rewards (right) when using a library only containing policies from very unrelated tasks, Ω_2 , Ω_3 , and Ω_4 (left). The red square represents the goal position in the target task. Source: Author.

4.2.2 Building a core policy library

A *core policy* is an essential policy for solving tasks in a given domain that is easily generalizable and which facilitates quickly learning similar tasks. The ability to build a library of these *core policies* is shown in this experiment, where the agents are tasked to build a library out of 50 randomly chosen tasks. The similarity measure that we use here is the actual similarity of the policy by selecting a number of random states from the environment and comparing the action recommendation between the policies and only keep a newly trained policy if the overlap is less than 60%. The results as shown in Figure 4.8 make clearly visible that *CBPI* and *PLPR* algorithms have similar behaviour and both select a good set of policies.

It is however notable, that *CBPI* provides a set of 16 policies that contains a policy for every room apart from the connecting rooms, which makes sense since those can already be reached by the policy for the next room. On the other

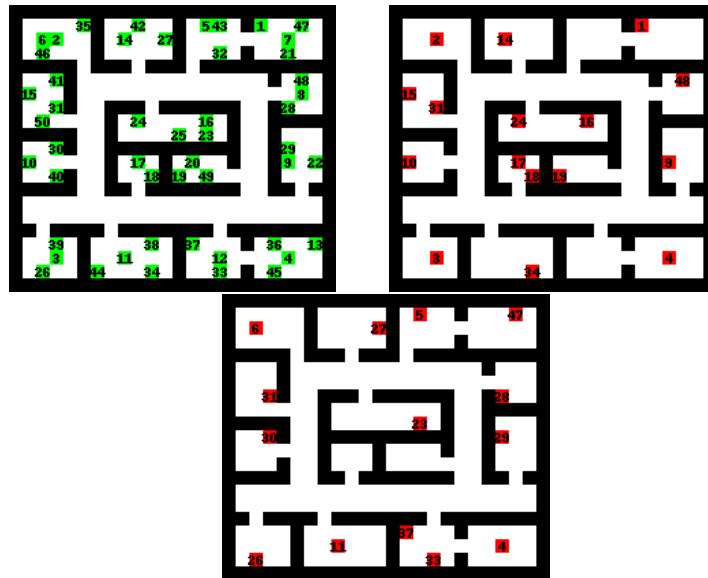


Figure 4.8: The same grid world domain was evaluated for the library building experiment. The graphs show all 50 goal positions that were learned during the experiment (left) and the extracted core policies using *CBPI* (middle) and *PLPR* (right), respectively. Source: Author.

hand, *PLPR* provides a set of 14 policies and seems to emphasize these connection rooms, however it does not provide policies for two of the rooms in the middle of the grid.

We therefore conclude that our selection approach in the *RETAIN* stage is sensitive enough for selecting core policies in this domain and provides acceptable results compared to *PLPR*.

4.2.3 Discussion

Here, we investigated the use of the *CBR* framework for the classic *RL* domain in a simple gridworld navigation environment. Our results demonstrate that this combination is a powerful tool to build a library of core policies while continuously reusing past knowledge to speed up the learning of a new task. Our approach shows that the selection of the right source knowledge is important to become more robust against *negative transfer*.

However, these results are achieved in a simple domain using classic *RL* approaches for learning. This limits *CBPI* to a very small number of problems and the algorithm as it does not translate well to more complex challenges where a *DNN* architecture is needed to deal with the high-dimensional state spaces. A more adequate algorithm would be needed to handle complex environments more efficiently and enable learning in a wider range of more challenging domains.

4.3 Using DECAF for knowledge building and reuse

In this section, we describe experiments using the *Deep Case-based Policy Inference (DECAF)* algorithm that is able to deal with highly complex domains. We show that the use of *DECAF* has a stabilizing effect and, given the proper previous knowledge, speeds up learning. More specifically, we describe two experiments combining *CBR* with *DRL* in the Atari game playing domain, using *DECAF* to train agents on new tasks. In the first one, we show that the algorithm which we use for training the new agent is sensitive to the quality of the previously acquired knowledge and, in the second one, we show the application of *DECAF* to an existing library of previously trained agents.

In these experiments, we show the progress of a learning agent under four different settings:

1. *DQN*: Vanilla *DQN* without using any previous knowledge as baseline.
2. *A2T (unfavorable)*: *A2T* using only unfavorable knowledge from the library.
3. *A2T (mixed)*: *A2T* using all knowledge from the library.
4. *DECAF*: Selects autonomously the best knowledge for training and then applies *A2T* using only favorable knowledge from the library.

4.3.1 Training parameters for *DECAF*

We are here describing all parameters that we used in the *DECAF* experiments in more detail and explain our choices to improve understanding and reproducibility. Our implementation is available online¹ and has taken inspiration from Castro et al. [2018]. The settings mostly follow the *DQN* implementation from Mnih et al. [2015] and the *A2T* implementation described by Rajendran et al. [2017] unless otherwise stated.

The state representation is generated from observing the high-dimensional screen output of the Atari game emulator. The emulator produces color images of 210×160 *pixel* at a frequency of 60 *Hz*. The state is then represented by 4 consecutive frames that are stacked and modified to grayscale and scaled to 84×84 *pixels* in size each to reduce the complexity. Another modification is the use of the maximum pixel value over two consecutive frames to avoid missing artifacts that are only visible every second frame because of limitations of the emulator.

The actual network architecture is shown in Figure 2.2 and consists of an input layer for the stacked images with dimensions $84 \times 84 \times 4$. The first hidden layer is a convolutional layer with 32 convolutional filters of size 8×8 with stride 4 and *ReLU* activation [Nair and Hinton, 2010]. The second hidden layer is also a convolutional layer but with 64 convolutional filters of size 4×4 with stride 2

¹https://github.com/cowhi/pub_DECAF

and *ReLU* activation. The third hidden layer is the last convolutional layer with 64 convolutional filters of size 3×3 with stride 1 and *ReLU* activation. The fourth layer is then a fully connected layer with 512 nodes and *ReLU* activation. The fifth and final layer then is also a fully-connected layer with a single node for every available action (in *Atari* games between 4 and 18) and linear activation. This architecture has proven itself as reasonable for working with the available data and learning complex behaviours.

Every training episode was initialized with 30 random steps to achieve different starting states and an episode was forcibly ended if the number of steps reached 27,000 to avoid getting stuck in possible endless loops.

When learning a new task, the network training starts after filling the *replay memory* with a minimum of 50,000 experiences. After reaching the maximum capability for experiences of 250,000 the *replay memory* remains at this level and handles new experiences in a *First-In-First-Out* manner. Here, we deviate from the original implementation because we seemed to achieve better results over all experiments.

The algorithm used for updating the network parameters is *RMSProp* with minibatches of size 32 (randomly drawn from the *replay memory*) after every 4th training step. We set the learning rate to 0.00025 and the decay rate of the gradient to 0.95 with a momentum of 0.0 and a small error constant of 0.00001, which were also slightly changed from the original implementation for performance reasons. For the calculation of the training target we used *reward clipping* to achieve values between -1 and 1 and set the discount factor of the agent to 0.99.

The balance between exploration and exploitation was achieved using the ϵ – *greedy* approach with a starting value of 1.0 and a decay over 1,000,000 training steps down to a minimum value of 0.1 that was kept consistent until the end of training. The ϵ value was set to 0.05 during testing episodes to avoid that an agent would get stuck.

As described earlier the weights of the *target* network were updated periodically with the weights of the *online* network in intervals of 10,000 steps.

For the *A2T* network, we kept most of these settings with the difference that the ϵ value was set to the minimum of 0.1 for the training steps from the beginning to rely on the past knowledge for guidance of generating new experiences during

training.

All graphs in the following subsections show results consisting of average reward and confidence intervals of 6 runs using different random seeds and in part different hardware (Nvidia GPUs Titan X/GTX 1080/V100) and operating systems (Ubuntu, CentOS).²

We show our results in graphs with average rewards over eras. An era consists of a training epoch of 250.000 steps followed by a test epoch of 10 test episodes. The training is performed for 20 eras with one data point for each era.

4.3.2 Determine the importance of pre-selection of source tasks

For this experiment, our goal was to enable the agent to learn how to play the Atari game *Pong* using pretrained agents to guide the learning. The tasks we chose to learn before learning *Pong* using the policy library were (i) *Pong* itself, to make sure we have an example that we can learn from without doubt, and (ii) *SpaceInvaders*, which is a game that has the same state and action space but has very different objectives, dynamics, and reward structures, to have an example of something that we would expect is not helping at all during training (see Figure 4.9).

In Figure 4.10 one can clearly see the influence of the pre-selection for the training library from all available cases. The setting *A2T (unfavorable)* shows the worst performance and highest variance in the results due to the negative influence

²Full code, hyper-parameter settings, and results available at https://github.com/cowhi/pub_DECAF.



Figure 4.9: Tasks contained in the knowledge base of experiment 1 are *Pong* (left) and *SpaceInvaders* (right). Source: Screenshot from Bellemare et al. [2013].

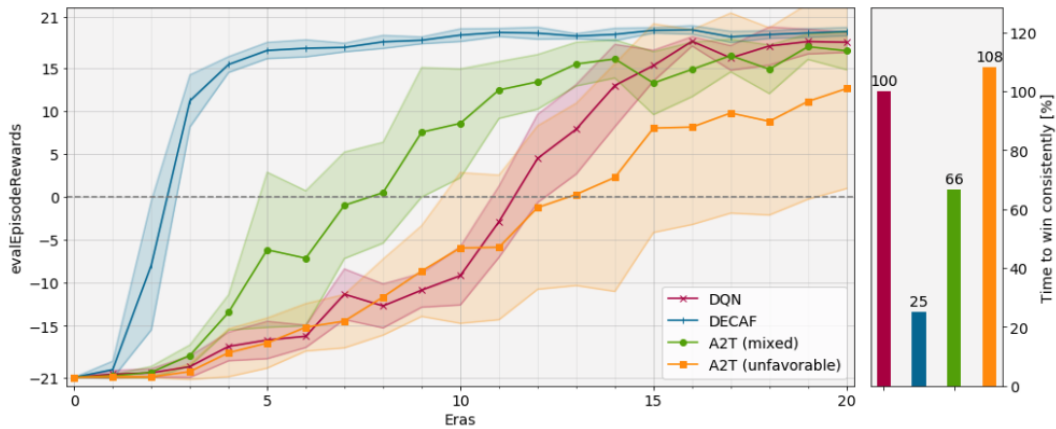


Figure 4.10: Resulting rewards using different pretrained agents. The graph shows average and confidence intervals of the performance of 6 experiments per setting with 10 evaluation periods per era and setting. Source: Author.

of the previous gathered knowledge, even under-performing the setting with no previous knowledge, *DQN*, in the final average reward but also in the time to start winning the game consistently, e.g. the average reward remains greater than 0 from that point on. *A2T (mixed)* shows that the algorithm can deal with some bad influence if it also has positive influence for learning and out-performs the baseline especially in the early phase of learning reaching the point where the agents starts to win the game consistently around 34% faster. *DECAF* is performing the best by far and starts winning permanently four times faster than the baseline algorithm and also has a marginally higher performance at the end of training.

It is also interesting to see that with a better training library the variance between individual runs seems to be reduced and the selection has a positive impact on stability of the agent, making the algorithm more reliable and apparently more robust against outliers.

4.3.3 Working with a larger library

In this second experiment, we investigated how the results change when we are working with a larger library of pretrained agents. Again, our goal was to learn the game *Pong* under several different settings. Our library consisted this time

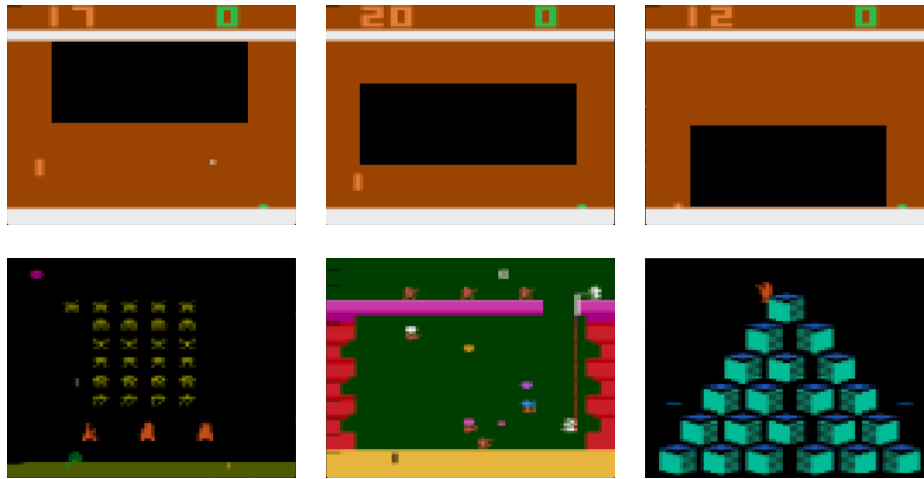


Figure 4.11: Tasks contained in the knowledge base of experiment 2 from left to right, upper row: *Pong* with obscured areas (upper half, middle, lower half); lower row: *SpaceInvaders*, *Pooyan*, and *Qbert*. Source: Screenshot from Bellemare et al. [2013].

of six pretrained agents on *Pong* with obscured areas (upper half, middle, lower half)³, *SpaceInvaders*, *Pooyan*, and *Qbert* (see Figure 4.11). The *Pong* versions are expected to have a positive influence on learning while still being distinct from the original *Pong*, and the other games are very different and expected to have a negative influence on training.

Figure 4.12 shows that for *A2T (unfavorable)* we can see that the influence of the three unfavorable agents is so dominant that the agent only seems to start learning at the end of our training time and never gets to a point where it wins the game. Using the whole library for *A2T (mixed)* shows that the agent is learning but it is slowed down and takes 50% longer than the baseline to start winning consistently but never reaches the same absolute performance until the end of training. For *DECAF* on the other hand, we can again see a clear improvement with a speedup of three times over the baseline and a higher absolute performance as well.

In the *RETRIEVE* stage our agent is relying on a policy library with cases

³The black areas in the figure are just for visualization of the blurred area while the environment used the actual background color during training.

about which we have some knowledge that helps to determine the similarity, but also if training is necessary at all, e.g. we already have a good policy in the library. The agent could save and use very different cases with very different state and action spaces, and *DECAF* would be able to only select policies from actually useful cases for training a new task. To do so the agent only needs to remember some additional information like the domain of the task in the case and the name of its environment. With this information the agent can query the environment to extract information about the observation space as well as the action space. In the event that this information is not enough to narrow the available cases down to a reasonable number, a number of test runs can be used to evaluate the performance in the new task and rank the cases accordingly.

In this experiment, all cases are from the same *Atari* domain and the observation space (RGB image of size 84 by 84 pixel) and action space (6 actions) are also the same. Using *A2T* without pre-selecting would work in this example, but in more realistic scenarios with cases with different observation and action spaces this could lead to catastrophic failure of the agent. The selection of the right source cases can further be limited by the actual concrete actions that are available, leaving only the various *Pong* versions and *SpaceInvaders* here. We are limiting the number of source cases for *DECAF* to 3 by performing 100 test episodes in the target task using each source policy and picking the highest performers according to average reward achieved (*PongBlurredUpper*: $R = 14.16$, *PongBlurredMiddle*: $R = 6.57$, *PongBlurredLower*: $R = -12.21$, *SpaceInvader*: $R = -21.00$).

The differences in the average rewards for the different versions of *Pong* might be explainable with an unbalanced use of the playing field in the game, making it harder to generalize depending on which part is blurred.

In the *RETAIN* stage, we perform the same test run we did for the source policies in the *RETRIEVE* stage but this time using our newly trained policy. The achieved average reward $R = 18.52$ over 100 episodes was greater as the already existing cases even when adding a threshold factor of 10% of the maximum score, $\sigma_{policy} = 2.1$, to the best performing policy from the library, *PongBlurredUpper*. Following this result, the new policy was added to the policy library.

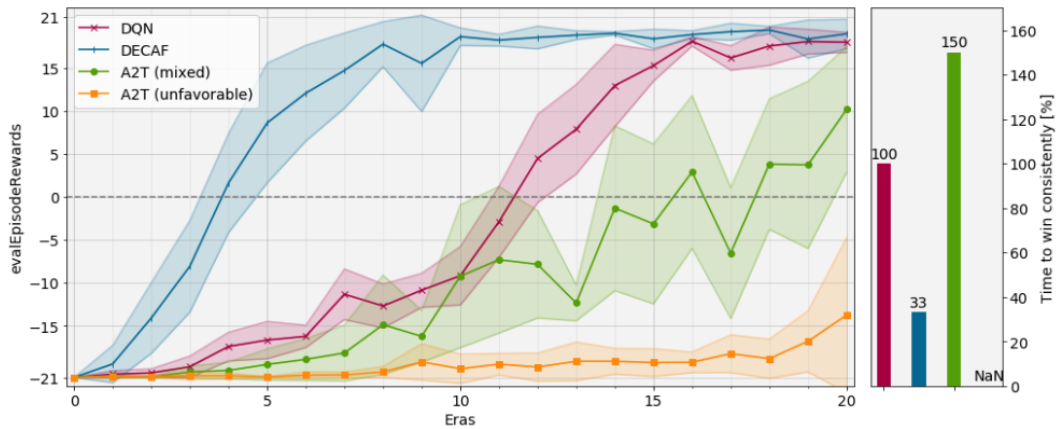


Figure 4.12: Resulting rewards using different tasks from the policy library. The upper graph shows the performance during training periods and the lower graph shows the performance during evaluation periods. Source: Author.

4.3.4 Discussion

In this section, we have shown that our proposed algorithm can deal well with high-dimensional problem spaces. We were able to demonstrate again the importance to select the proper source knowledge for transfer and show that *DECAF* is good at doing that given the right similarity measures between tasks. *DECAF* performs better than approaches that do not consider similarity and has clear speed-ups and even improves learning performance on an absolute scale for our problem space.

The advantage of learning faster might be diminished by the effort to define and determine similarity as well as the increased computational requirements for integrating the training library and the importance network. In our experiments, the time for the similarity determination was very short and could be neglected for the total consideration, but the size and quality of the training library played an important role. We observed that reaching the point, where the agent starts to win consistently (average reward greater than 0), was still achieved faster on an absolute timescale, where *DECAF* needed only 55% as the baseline *DQN* algorithm, while *A2T* with all policies from the library needed almost 320%.

Another important aspect is the scalability of this approach which is limited

by the maximum size of the case library. This could be extended, for example, by applying more abstraction or increasing generalization over tasks.

The most limiting factor to our framework is the need for the similarity determination for tasks, but also for policies. This requirement is very hard to define and is the base for a whole branch of research itself and a deep discussion here would be out of the scope of our considerations.

Chapter 5

Final considerations

In this last chapter, we are giving a short summary and conclusion of the conducted research (Section 5.1) and a brief outline of potential future work (Section 5.2).

5.1 Conclusion

In this work, we investigated the application of *TL* methods to the *DRL* domain.

First, we investigated the impact of *direct parameter transfer* when transferring knowledge from source tasks with different degrees of similarity to a target task. Our results show that the initialization of a *DRL* agent plays a far more important role than the choice of the optimization algorithm for network updates or the number of available actions in the output layer of the network. While transfer from a very similar task can have great benefits for an agent, we also show that transfer from an unrelated task can lead to *negative transfer*, hampering the learning success of the agent.

Having shown the importance of a systematic approach to knowledge transfer, we proposed a framework for exactly that purpose. We combined the *CBR* methodology with *RL* and proposed two algorithms based on this framework. We used the first one, called *Case-based Policy Inference*, to show the usefulness of the framework for building a library of core policies over time while speeding up learning and avoiding *negative transfer*.

The second algorithm is the extension of the framework to *DRL*, called *Deep Case-based Policy Inference*, which we evaluate in the *Atari* game playing domain. It exploits previously learned policies to transfer knowledge from similar source tasks to the target task and blends those policies during training. We compared *DECAF* with *A2T* and *Vanilla DQN* to show that our algorithm has benefits in scenarios with advantageous previously attained knowledge compared to single task learning (*DQN*), but also over other transfer methods in scenarios with disadvantageous previously attained knowledge, while being more robust against negative transfer and helping to produce more stable results over different runs.

In summary, we proposed a novel framework and algorithm for knowledge transfer for *DRL* agents and published results that show the usefulness of our approach for sequential task learning with knowledge conservation and reuse.

5.2 Future work

Even though the goals are similar, we divide this section and consider future works specifically for the proposed framework and then for future works in general for the area of intelligent agents.

Future works on this framework could focus on various aspects of the process for gradual improvements. The following is an incomplete list of suggestions that might have an impact of the performance of the framework or challenges that might occur in certain situations:

- Most importantly, the **definition and discovery of similarity** between tasks to improve the selection of useful source tasks would benefit from a more general approach. One way that comes to mind would be the use of an object-oriented representation as introduced by Silva et al. [2019] for multi-agent systems.
- The framework could easily be extended with the ability to not start learning from a random initialization but from a **parameter initialization** that has been generated specifically for the current task or transferred from a very similar task.

- When training in complex environments an agent could develop a strategy that results in short-term gains but has disadvantages over a long-term strategy. In this **optimality mismatch**, an agent that relies on transferred knowledge from an agent trained on the long-term strategy only for part of the task might then be triggered into performing sub-optimal actions for its current objective.
- Another interesting aspect would be to see how the saved knowledge could be better organized by distilling the available policies into fewer policies to **reduce storage requirements** and generalization abilities of the agent.
- Yet another approach could be to replace the *Neural Network* policies with programmatic policies as proposed by Verma et al. [2018] that are specifically designed to be more easily interpretable than *Neural Networks*, which in turn could serve to better detect similarities in the knowledge that we chose to retain.

When looking at the big picture for developing more intelligent agents it becomes obvious that there is still a lot of room to grow. The following is a list that reflects some of the possible directions that we think would most probably lead to progress towards that goal:

- Even though there have been first results and it is an emerging research area, *TL* for *DRL* still needs more attention to achieve more breakthrough results. In general, we would welcome more attention to approaches that combine *Lifelong* and *Multi-task Learning* to resemble a closer similarity to the human learning approach which is able to continuously learn multiple tasks in parallel if necessary. These approaches could combine seamlessly with *Meta Learning* approaches that could provide a structure to the process.
- Another important aspect for making gained insights more valuable is generating better abstractions of this knowledge to make it more applicable to a wider range of problems without having to be sure about every detail, as for example shown in Koga et al. [2015]. Another approach could be to rethink the way we are looking at state values at the moment by factoring

in more information than simply the state representation as has been investigated with Universal Value Function Approximators introduced by Schaul et al. [2015]. Both, abstraction and augmentation of states, would benefit from a better conceptualization of knowledge, which already is a well researched topic in cognitive psychology and which provides a structure to form a proper representation of a situation in the right context [Kelter and Kaup, 2019].

- Another reason, why we often only encounter works on restricted problem spaces is the absence of training environments that would require an agent to use varying skills while encountering dynamically changing environment conditions. One way to do this would be to have a state representation that consists of multi-modal data streams where there are not always all streams available while at the same time having a similar restriction on the available action space. The first part is being explored in *Partial Observable MDPs* and has gained a lot of interest lately as for example in Igl et al. [2018].

Despite those open challenges, we believe that the extension of *DRL* with *TL* abilities will certainly lead to increased generalization and robustness for learning agents soon. Eventually, this combination will produce the most stable and general learners that are capable of dealing with the most complex tasks, making it a key achievement towards reaching *Artificial General Intelligence*.

Appendix A

Related works for transfer in Deep Reinforcement Learning

Publications	Type			Quantity			Strategy				
	Lifelong	Multi-task	Imitation	Single agent	Multitagent	Parameter	Sample	Skills	Curriculum	Self-play	Meta
[Glatt et al., 2016],[Du et al., 2016],[Levine et al., 2016]	✓			✓		✓					
[Isele and Cosgun, 2018]	✓			✓			✓				
[Vezhnevets et al., 2017],[Tessler et al., 2017], [Bacon et al., 2017],[Finn et al., 2017]	✓			✓				✓			
[Wu and Tian, 2017]	✓			✓					✓		
[Silver et al., 2018],[Silver et al., 2017],[Bansal et al., 2018]	✓			✓						✓	
[Schaul et al., 2015],[Rusu et al., 2016],[Gupta et al., 2018]	✓			✓							✓
[Gu et al., 2017],[Mnih et al., 2016]	✓			✓			✓				
[Shao et al., 2018]	✓			✓			✓		✓		
[Fernando et al., 2017]	✓			✓			✓				✓
[Czarnecki et al., 2018]	✓	✓		✓					✓		
[Teh et al., 2017]	✓	✓		✓							✓
[Parisotto et al., 2016],[Rusu et al., 2015], [Oh et al., 2017]	✓	✓		✓							✓
[Omidshafiei et al., 2017]	✓	✓		✓			✓				✓
[Finn et al., 2016],[Stadie et al., 2017]	✓		✓	✓							✓

Table A.1: Classification of a selection of publications that represent the recent development in the field.

Appendix B

Productivity and Accomplishments

This section describes the academic achievements and conducted activities during the course of the Ph.D. program.

B.1 Publications

The following list indicates the research productivity during the Ph.D. program:

DECAF: Deep Case-based Policy Inference for Knowledge Transfer in Reinforcement Learning Glatt et al. [2020]

MOO-MDP: An Object-Oriented Representation for Cooperative Multiagent Reinforcement Learning. Silva et al. [2019]

A Framework to Discover and Reuse Object-Oriented Options in Reinforcement Learning. Bonini et al. [2018]

Case-based Policy Inference for Transfer in Reinforcement Learning. Glatt et al. [2017b]

Simultaneously Learning and Advising in Multiagent Reinforcement Learning. Silva et al. [2017b]

Case-based Policy Inference (Short Paper). Glatt et al. [2017a]

Transferring Probabilistic Options in Reinforcement Learning (Short Paper). Bonini et al. [2017]

Improving Deep Reinforcement Learning with knowledge transfer (Doctoral Consortium Extended Abstract). Glatt and Costa [2017b]

Policy Reuse in Deep Reinforcement Learning (Student Abstract). Glatt and Costa [2017a]

An advising framework for Multiagent Reinforcement Learning systems (Student abstract). Silva et al. [2017a]

Towards Knowledge Transfer in Deep Reinforcement Learning. Glatt et al. [2016]

Object-Oriented Reinforcement Learning in Cooperative Multiagent Domains. Silva et al. [2016]

B.2 Openly available software

This is a list of software that was made available to the general public in connection with my research:

- https://github.com/cowhi/pub_DECAF: Software to replicate experiments conducted for Glatt et al. [2020].
- https://github.com/cowhi/lab_rl: Software to conduct experiments in the *Deepmind Lab* environment [Beattie et al., 2016].
- <https://github.com/cowhi/CBPI>: Software to replicate the experiments conducted in Glatt and Costa [2017b].
- <https://github.com/cowhi/deepatari>: Framework to support the development of algorithms for intelligent agents in the Atari game playing domain.

B.3 Honors

The general idea of this research has been well received and the performance has been rewarded with the following honors:

- **2019 - Turnitin Scholarship:** Winner. Awarded to attend the Deep Learning Bootcamp at the *University of California, Berkeley*.

- **2018 - IEEE Upsilon Pi Epsilon Honor Society Award:** Winner. Awarded for academic excellence for students in the computing discipline by the *IEEE Computer Society*.
- **2017 - Outstanding Young Researcher:** Invited to join the *5th Heidelberg Laureate Forum (HLF)*.
- **2017 - Best Student Poster:** Winner. Awarded for representation of our research at the *31st AAAI Conference on Artificial Intelligence (AAAI)*. [Silva et al., 2017a]
- **2017 - Doctoral Consortium Travel Grant:** Awarded to present and discuss my research efforts at the *31st AAAI Conference on Artificial Intelligence (AAAI)*.
- **2016 - Google Research Award Latin-America¹** Winner. The award is intended as an honor to world-class permanent faculty members and their students who are pursuing cutting-edge research in specific fields related to Computer Science at top universities in Latin America.
- **2016 - NVIDIA GPU Grant Program²** Winner. The award is intended to empower researcher and to collaborate with universities worldwide to support cutting-edge technological innovation.
- **2016 - Best Paper Award 1st Place:** Winner. Awarded at the *5th Brazilian Conference on Intelligent Systems (BRACIS)* [Silva et al., 2016].
- **2016 - Distinguished Work Award:** Winner. Awarded at the *5th Postgraduate Workshop in Computer Engineering (WPGEC)*.
- **2015 - Google Research Award Latin-America:** Winner.
- **2015 - CAPES Ph.D. scholarship:** Research scholarship for the duration of the Ph.D. program.

¹[Google Research Award Latin-America Homepage](#)

²[Nvidia GPU Grant Homepage](#)

B.4 Internships

The following internship was conducted:

- **2018 - Microsoft Research, Redmond, USA:** 3 month engagement as *Research Intern* to work at the intersection of *Software Testing* and *Machine Learning* utilizing *Reinforcement Learning* methods.

B.5 Volunteering

The following list shows engagements in the research community:

- **2019 - Workshop Chair:** 2nd Scaling-Up Reinforcement Learning (SURL) Workshop at IJCAI 2019.
- **2018 - Conference Volunteer:** 32nd Conference on Neural Information Processing Systems (NeurIPS)
- **2017 - Workshop Chair:** 1st Scaling-Up Reinforcement Learning (SURL) Workshop at the European Conference on Machine Learning (ECML) 2017.
- **2017 - Local Organizer:** 1st Workshop on Transfer in Reinforcement Learning (TiRL) at the 6th International Conference on Autonomous Agents and Multiagent Systems (AAMAS) 2017.
- **2016 - Conference Volunteer:** 33rd International Conference on Machine Learning (ICML)
- **2016 - Conference Volunteer:** 30th Conference on Neural Information Processing Systems (NIPS)

B.6 Disciplines

The minimum amount of credits (40) for the conclusion of the Ph.D. program has been acquired in full by attending seven disciplines with a total of 44 credits and an average grade of A.

- **PCS5024-1/1 - Aprendizado Estatístico** by *Anna Helena Reali Costa, Fabio G. Cozman & Denis D. Mauá*. Duration: 23/02/2015 - 29/05/2015, Workload: 120h, Credits: 8, Situation: concluded, Frequency: 100%, **Grade: A**.
- **PMR5241-1/1 - Robôs Sociáveis e Computação Afetiva** by *Marcos Ribeiro Pereira Barretto*. Duration: 09/06/2015 - 31/08/2015, Workload: 120h, Credits: 8, Situation: concluded, Frequency: 100%, **Grade: A**.
- **MAC5784-2/1 Inteligência Artificial em Jogos de Computador** by *Flavio Soa-res Correa da Silva*. Duration: 10/08/2015 - 27/11/2015, Workload: 120h, Credits: 8, Situation: concluded, Frequency: 100%, **Grade: A**.
- **PCS5012-5/1 Metodologia de Pesquisa Científica em Engenharia de Computação** by *Anarosa Alves Franco Brandão & Anna Helena Reali Costa*. Duration: 14/09/2015 - 18/12/2015, Workload: 120h, Credits: 8, Situation: concluded, Frequency: 92%, **Grade: A**.
- **EAD5966-1/1 Ciência, Tecnologia e Inovação: A Dimensão Internacional** by *Guilherme Ary Plonski & Alberto Pfeifer at Faculdade de Economia, Administração e Contabilidade*. Duration: 01/03/2016 - 24/06/2016, Workload: 120h, Credits: 8, Situation: concluded, Frequency: 100%, **Grade: A**.
- **PCS5118-1/1 Seminários em Engenharia de Computação** by *Anna Helena Reali Costa & Alessandro Antonucci*. Duration: 13/11/2017 - 03/12/2017, Workload: 30h, Credits: 2, Situation: concluded, Frequency: 100%, **Grade: A**.
- **PCS5119-1/1 Tópicos Avançados em Engenharia de Computação** by *Alessandro Antonucci*. Duration: 19/02/2018 - 04/03/2018, Workload: 30h, Credits: 2, Situation: concluded, Frequency: 100%, **Grade: A**.

Additionally the prerequisite to participate in the teaching internship as demanded by the scholarship regulations has been acquired by attending the lecture series on pedagogical preparation in the *Programa de Aperfeiçoamento de Ensino*

(PAE) of the Departamento de Engenharia Metalúrgica e de Materiais da EPUSP from 10/10/2015 - 26/10/2015. The following lectures were attended:

- **Lecture 1:** As múltiplas inteligências e sua importância na escolha de atividades para elaborar uma aula, *Prof. Dr. Sérgio Duarte Brandi* (13/10/2015, Duration: 2h)
- **Lecture 2:** Estratégias de educação em engenharia e experiências na Escola Politécnica da USP, *Prof. Dr. José Aquiles Baesso Grimoni e Prof. Dr. Osvaldo Shigueru Nakao* (14/10/2015, Duration: 2h)
- **Lecture 3:** Erro Humano, *Prof. Dr. Sérgio Médici de Eston* (19/10/2015, Duration: 2h)
- **Lecture 4:** Orientação no uso dos recursos informacionais disponíveis para elaboração de trabalhos acadêmicos, *Maria Aparecida Gabriel e Maria Cristina Olaio Villela* (21/10/2015, Duration: 2h)
- **Lecture 5:** A Escola Politécnica e o ensino paulista: ensino e pesquisa, *Prof. Dr. Macioniro Celeste Filho* (26/10/2015, Duration: 2h)

The following one semester teaching internship was subsequently conducted under the supervision of *Profa. Anarosa Alves Franco Brandão* in the discipline *PCS3335-1 Laboratório Digital A* in the 2nd semester of 2016.

Bibliography

- A. Aamodt and E. Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI communications*, 7(1):39–59, 1994.
- D. W. Aha. The omnipresence of case-based reasoning in science and application. *Knowledge-based systems*, 11(5):261–273, 1998a.
- D. W. Aha. The omnipresence of case-based reasoning in science and application. *Knowledge-based systems*, 11(5-6):261–273, 1998b.
- P.-L. Bacon, J. Harb, and D. Precup. The option-critic architecture. In *Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)*, pages 1726–1734, 2017.
- T. Bansal, T. Pachocki, S. Sidor, S. I., and I. Mordatch. Emergent complexity via multi-agent competition. In *Sixth International Conference on Learning Representations (ICLR)*, 2018.
- C. Beattie, J. Z. Leibo, D. Teplyaev, T. Ward, M. Wainwright, H. Küttler, A. Lefrancq, S. Green, V. Valdés, A. Sadik, et al. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016.
- M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.
- R. Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

- Y. Bengio. Deep learning of representations for unsupervised and transfer learning. *Unsupervised and Transfer Learning Challenges in Machine Learning*, 7: 19, 2012.
- Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, et al. Greedy layer-wise training of deep networks. *Advances in Neural Information Processing Systems*, 19: 153–160, 2007.
- R. A. Bianchi, R. Ros, and R. L. De Mantaras. Improving reinforcement learning by using case based heuristics. In *Eighth International Conference on Case-Based Reasoning (ICCBR)*, pages 75–89. Springer, 2009.
- R. A. Bianchi, P. E. Santos, I. J. da Silva, L. A. Celiberto, and R. L. de Mantaras. Heuristically accelerated reinforcement learning by means of case-based reasoning and transfer learning. *Journal of Intelligent & Robotic Systems*, 91(2): 301–312, 2018.
- R. C. Bonini, F. L. d. Silva, R. Glatt, and A. H. R. Costa. Transferring probabilistic options in reinforcement learning (short paper). In *First Workshop on Transfer in Reinforcement Learning (TIRL)*, 2017.
- R. C. Bonini, F. L. d. Silva, R. Glatt, E. Spina, and A. H. R. Costa. A framework to discover and reuse object-oriented options in reinforcement learning. In *Seventh Brazilian Conference on Intelligent Systems (BRACIS)*. IEEE, 2018. doi: 10.1109/BRACIS.2018.00027.
- P. S. Castro, S. Moitra, C. Gelada, S. Kumar, and M. G. Bellemare. Dopamine: A Research Framework for Deep Reinforcement Learning. *arXiv preprint arXiv:1812.06110*, 2018.
- P. Chazara, S. Negny, and L. Montastruc. Flexible knowledge representation and new similarity measure: Application on case based reasoning for waste treatment. *Expert Systems with Applications*, 58:143–154, 2016.
- W. Cheetham and I. Watson. Fielded applications of case-based reasoning. *The Knowledge Engineering Review*, 20(03):321–323, 2005.

- R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011.
- W. M. Czarnecki, S. M. Jayakumar, M. Jaderberg, L. Hasenclever, Y. W. Teh, N. Heess, S. Osindero, and R. Pascanu. Mix & match agent curricula for reinforcement learning. In *Thirty-Fifth International Conference on Machine Learning (ICML)*, pages 1095–1103, 2018.
- Y. N. Dauphin, H. de Vries, J. Chung, and Y. Bengio. RMSProp and equilibrated adaptive learning rates for non-convex optimization. *arXiv preprint arXiv:1502.04390*, 2015.
- M. Denil, P. Agrawal, T. D. Kulkarni, T. Erez, P. Battaglia, and N. de Freitas. Learning to perform physics experiments via deep reinforcement learning. *arXiv preprint arXiv:1611.01843*, 2016.
- F. Domínguez-Estévez, A. A. Sánchez-Ruiz, and P. P. Gómez-Martín. Training pac-man bots using reinforcement learning and case-based reasoning. In *Fourth Congreso de la Sociedad Española para las Ciencias del Videjuego (CoSE-Civi)*, pages 144–156, 2017.
- Y. Du, V. Gabriel, J. Irwin, and M. E. Taylor. Initial progress in transfer for deep reinforcement learning algorithms. In *DRL Workshop at the International Conference on Machine Learning (ICML)*, 2016.
- F. Fang and T. Wong. Applying hybrid case-based reasoning in agent-based negotiations for supply chain management. *Expert Systems with Applications*, 37(12):8322–8332, 2010.
- F. Fernández and M. Veloso. Probabilistic policy reuse in a reinforcement learning agent. In *Fifth International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 720–727, 2006.
- C. Fernando, D. Banarse, C. Blundell, et al. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017.

- C. Finn, S. Levine, and P. Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *Thirty-Third International Conference on Machine Learning (ICML)*, pages 49–58, 2016.
- C. Finn, T. Yu, J. Fu, P. Abbeel, and S. Levine. Generalizing skills with semi-supervised reinforcement learning. In *Fifth International Conference on Learning Representations (ICLR)*, 2017.
- T. Gabel and M. Riedmiller. Cbr for state value function approximation in reinforcement learning. In *Sixth International Conference on Case-Based Reasoning (ICCBR)*, pages 206–221. Springer, 2005.
- Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *Thirty-Third International Conference on Machine Learning (ICML)*, 2016.
- W. R. Gilks, S. Richardson, and D. Spiegelhalter. *Markov chain Monte Carlo in practice*. Chapman and Hall/CRC, 1995.
- R. Glatt and A. H. R. Costa. Policy reuse in deep reinforcement learning (student abstract). In *Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)*, pages 4929–4930. AAAI Publications, 2017a.
- R. Glatt and A. H. R. Costa. Improving deep reinforcement learning with knowledge transfer (extended abstract). In *Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)*, pages 5036–5037. AAAI Publications, 2017b.
- R. Glatt, F. L. d. Silva, and A. H. R. Costa. Towards knowledge transfer in deep reinforcement learning. In *Fifth Brazilian Conference on Intelligent Systems (BRACIS)*, pages 91–96. IEEE, 2016.
- R. Glatt, F. L. d. Silva, and A. H. R. Costa. Case-based policy inference (short paper). In *First Workshop on Transfer in Reinforcement Learning (TIRL)*, 2017a.
- R. Glatt, F. L. d. Silva, and A. H. R. Costa. Case-based policy inference for transfer in reinforcement learning. In *First Workshop on Scaling-Up Reinforcement Learning (SURL)*, pages 1–8, 2017b.

- R. Glatt, F. L. Da Silva, R. A. da Costa Bianchi, and A. H. R. Costa. Decaf: deep case-based policy inference for knowledge transfer in reinforcement learning. *Expert Systems with Applications*, 156:113420, 2020.
- S. Gu, E. Holly, T. Lillicrap, and S. Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *International Conference on Robotics and Automation (ICRA)*, pages 3389–3396, 2017.
- A. Gupta, R. Mendonca, Y. Liu, P. Abbeel, and S. Levine. Meta-reinforcement learning of structured exploration strategies. *Advances in Neural Information Processing Systems*, 31:5307–5316, 2018.
- P. Henderson, R. Islam, P. Bachman, et al. Deep reinforcement learning that matters. *arXiv preprint arXiv:1709.06560*, 2017.
- M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning. *arXiv preprint arXiv:1710.02298*, 2017.
- G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.
- E. Hullermeier. Credible case-based inference using similarity profiles. *IEEE Transactions on Knowledge and Data Engineering*, 19(6):847–858, 2007.
- M. Igl, L. Zintgraf, T. A. Le, F. Wood, and S. Whiteson. Deep variational reinforcement learning for pomdps. *Thirty-Fifth International Conference on Machine Learning (ICML)*, 2018.
- D. Isele and A. Cosgun. Selective experience replay for lifelong learning. In *Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*, 2018.
- C. Jiang and Z. Sheng. Case-based reinforcement learning for dynamic inventory control in a multi-agent supply-chain system. *Expert Systems with Applications*, 36(3):6520–6526, 2009.

- L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, pages 237–285, 1996.
- S. Kelter and B. Kaup. Conceptual knowledge, categorization, and meaning. *Semantics-Typology, Diachrony and Processing*, 2019.
- D. Kingma and J. Ba. ADAM: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- D. B. Kirk and W. H. Wen-mei. *Programming massively parallel processors: a hands-on approach*. Newnes, 2012.
- R. Kitchin. *The data revolution: Big data, open data, data infrastructures and their consequences*. Sage, 2014.
- M. L. Koga, V. Freire, and A. H. Costa. Stochastic abstract policies: Generalizing knowledge to improve reinforcement learning. *IEEE Transactions on Cybernetics*, 45(1):77–88, 2015.
- J. Kolodner. *Case-based reasoning*. Morgan Kaufmann, 2014.
- J. L. Kolodner. Making the implicit explicit: Clarifying the principles of case-based reasoning. *Case-based Reasoning: Experiences, Lessons and Future Directions*, pages 349–370, 1996.
- G. Konidaris and A. G. Barto. Building portable options: Skill transfer in reinforcement learning. In *Twentieth International Joint Conference on Artificial Intelligence (IJCAI)*, volume 7, pages 895–900, 2007.
- A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 25:1097–1105, 2012.
- A. Lazaric. Transfer in reinforcement learning: a framework and a survey. In *Reinforcement Learning*, pages 143–173. Springer, 2012.
- Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

- S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(1):1334–1373, 2016.
- P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, et al. Learning to navigate in complex environments. In *Fifth International Conference on Learning Representations (ICLR)*, 2017.
- V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *Computing Research Repository*, abs/1312.5602, 2013.
- V. Mnih, D. Silver, A. A. Rusu, M. Riedmiller, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.
- V. Mnih, A. P. Badia, M. Mirza, et al. Asynchronous methods for deep reinforcement learning. In *Thirty-Third International Conference on Machine Learning (ICML)*, pages 1928–1937, 2016.
- V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Twenty-Seventh International Conference on Machine Learning (ICML)*, pages 807–814. ACM, 2010.
- A. Y. Ng, A. Coates, M. Diehl, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang. Autonomous inverted helicopter flight via reinforcement learning. In *Experimental Robotics IX*, pages 363–372. Springer, 2006.
- J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, Q. V. Le, and A. Y. Ng. On optimization methods for deep learning. In *Twenty-Eighth International Conference on Machine Learning (ICML)*, pages 265–272, 2011.
- J. Oh, S. Singh, H. Lee, and P. Kohli. Zero-shot task generalization with multi-task deep reinforcement learning. In *Thirty-Fourth International Conference on Machine Learning (ICML)*, pages 2661–2670, 2017.

- S. Omidshafiei, J. Pazis, C. Amato, J. P. How, and J. Vian. Deep decentralized multi-task multi-agent reinforcement learning under partial observability. In *Thirty-Fourth International Conference on Machine Learning (ICML)*, pages 2681–2690, 2017.
- S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.
- E. Parisotto, J. L. Ba, and R. Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. In *Fifth International Conference on Learning Representations (ICLR)*, 2016.
- M. L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, Hoboken, NJ, USA, 2014. ISBN 1118625870.
- J. Rajendran, A. Lakshminarayanan, M. M. Khapra, P. Prasanna, and B. Ravindran. Attend, adapt and transfer: Attentive deep architecture for adaptive transfer from multiple sources in the same domain. *arXiv preprint arXiv:1510.02879v5*, 2017.
- M. Riedmiller. Neural fitted q iteration—first experiences with a data efficient neural reinforcement learning method. In *Sixteenth European Conference on Machine Learning (ECML)*, pages 317–328. Springer, 2005.
- B. Rosman, M. Hawasly, and S. Ramamoorthy. Bayesian policy reuse. *Machine Learning*, 104(1):99–127, 2016.
- A. A. Rusu, S. G. Colmenarejo, C. Gulcehre, et al. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015.
- A. A. Rusu, N. C. Rabinowitz, G. Desjardins, et al. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.
- T. Schaul, D. Horgan, K. Gregor, and D. Silver. Universal value function approximators. In *Thirty-Second International Conference on Machine Learning (ICML)*, pages 1312–1320, 2015.

- J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.
- K. Shao, Y. Zhu, and D. Zhao. Starcraft micromanagement with reinforcement learning and curriculum transfer learning. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 99:1–12, 2018.
- M. Sharma, M. P. Holmes, J. C. Santamaría, A. Irani, C. L. Isbell Jr, and A. Ram. Transfer learning in real-time strategy games using hybrid cbr/rl. In *Twentieth International Joint Conference on Artificial Intelligence (IJCAI)*, volume 7, pages 1041–1046, 2007.
- F. L. d. Silva and A. H. R. Costa. Towards Zero-Shot Autonomous Inter-Task Mapping through Object-Oriented Task Description. In *First Workshop on Transfer in Reinforcement Learning (TiRL)*, pages 1–10, 2017.
- F. L. d. Silva and A. H. R. Costa. A survey on transfer learning for multiagent reinforcement learning systems. *Journal of Artificial Intelligence Research*, 69: 645–703, 2019.
- F. L. d. Silva, R. Glatt, and A. H. R. Costa. Object-oriented reinforcement learning in cooperative multiagent domains. In *Fifth Brazilian Conference on Intelligent Systems (BRACIS)*, pages 19–24. IEEE, 2016.
- F. L. d. Silva, R. Glatt, and A. H. R. Costa. An advising framework for multiagent reinforcement learning systems. In *Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)*, pages 4913–4914. AAAI Publications, 2017a.
- F. L. d. Silva, R. Glatt, and A. H. R. Costa. Simultaneously learning and advising in multiagent reinforcement learning. In *Sixteenth International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2017b.
- F. L. d. Silva, R. Glatt, and A. H. R. Costa. Moo-mdp: An object-oriented representation for cooperative multiagent reinforcement learning. *IEEE Transactions on Cybernetics*, 49(2):567–579, 2019. ISSN 2168-2267. doi: 10.1109/TCYB.2017.2781130.

- D. Silver, J. Schrittwieser, K. Simonyan, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.
- D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.
- J. Sinapov, S. Narvekar, M. Leonetti, and P. Stone. Learning inter-task transferability in the absence of target task samples. In *Fourteenth International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 725–733, 2015.
- B. C. Stadie, P. Abbeel, and I. Sutskever. Third-person imitation learning. In *Fifth International Conference on Learning Representations (ICLR)*, 2017.
- P. Stone and R. S. Sutton. Scaling reinforcement learning toward robocup soccer. In *Eighteenth International Conference on Machine Learning (ICML)*, pages 537–544. ACM, 2001.
- R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
- M. E. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10:1633–1685, 2009.
- M. E. Taylor, P. Stone, and Y. Liu. Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*, 8(1): 2125–2167, 2007.
- Y. Teh, V. Bapst, R. Pascanu, et al. Distral: Robust multitask reinforcement learning. *Advances in Neural Information Processing Systems*, 30:4497–4507, 2017.
- G. Tesauro. Practical issues in temporal difference learning. *Advances in Neural Information Processing Systems*, 5:259–266, 1992.

- G. Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.
- C. Tessler, S. Givony, T. Zahavy, et al. A deep hierarchical approach to lifelong learning in minecraft. In *Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)*, pages 1553–1561, 2017.
- S. Thrun. Lifelong learning algorithms. In *Learning to learn*, pages 181–209. Springer, 1998.
- S. Thrun and T. M. Mitchell. *Lifelong robot learning*, volume 15. Elsevier, 1995.
- A. Verma, V. Murali, R. Singh, P. Kohli, and S. Chaudhuri. Programmatically interpretable reinforcement learning. *Thirty-Fifth International Conference on Machine Learning (ICML)*, 2018.
- A. S. Vechnyevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *Thirty-Fourth International Conference on Machine Learning (ICML)*, pages 3540–3549, 2017.
- C. J. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.
- I. Watson. Case-based reasoning is a methodology not a technology. *Knowledge-based systems*, 12(5):303–308, 1999.
- K. Weiss, T. M. Khoshgoftaar, and D. Wang. A survey of transfer learning. *Journal of Big Data*, 3(1):9, 2016.
- Y. Wu and Y. Tian. Training agent for first-person shooter game with actor-critic curriculum learning. In *Fifth International Conference on Learning Representations (ICLR)*, 2017.
- H. Y. Xiong, B. Alipanahi, L. J. Lee, H. Bretschneider, D. Merico, R. K. Yuen, Y. Hua, S. Gueroussov, H. S. Najafabadi, T. R. Hughes, et al. The human splicing code reveals new insights into the genetic determinants of disease. *Science*, 347(6218):1254806, 2015.

T. Zhao, W. Zhang, H. Zhao, and Z. Jin. A reinforcement learning-based framework for the generation and evolution of adaptation rules. In *Forteenth International Conference on Autonomic Computing (ICAC)*, pages 103–112. IEEE, 2017.