Expert Systems With Applications 156 (2020) 113420

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

DECAF: Deep Case-based Policy Inference for knowledge transfer in Reinforcement Learning

Ruben Glatt[a,*], Felipe Leno Da Silva[a], Reinaldo Augusto da Costa Bianchi[b], Anna Helena Reali Costa[a]

[a] University of Sao Paulo, Av. Prof. Luciano Gualberto 158, Sao Paulo 05508-010, SP, Brazil
[b] FEI's University Centre, Av. Humberto de Alencar Castelo Branco 3972, Sao Bernardo do Campo 09850-901, SP, Brazil

ARTICLE INFO

Article history:
Received 20 May 2019
Revised 29 February 2020
Accepted 26 March 2020
Available online 31 March 2020

Keywords:
Deep Reinforcement Learning
Case-based Reasoning
Transfer Learning
Knowledge discovery
Knowledge management
Neural networks

ABSTRACT

Having the ability to solve increasingly complex problems using *Reinforcement Learning (RL)* has prompted researchers to start developing a greater interest in systematic approaches to retain and reuse knowledge over a variety of tasks. With *Case-based Reasoning (CBR)* there exists a general methodology that provides a framework for knowledge transfer which has been underrepresented in the *RL* literature so far. We formulate a terminology for the *CBR* framework targeted towards *RL* researchers with the goal of facilitating communication between the respective research communities. Based on this framework, we propose the *Deep Case-based Policy Inference (DECAF)* algorithm to accelerate learning by building a library of cases and reusing them if they are similar to a new task when training a new policy. *DECAF* guides the training by dynamically selecting and blending policies according to their usefulness for the current target task, reusing previously learned policies for a more effective exploration but still enabling the adaptation to particularities of the new task. We show an empirical evaluation in the Atari game playing domain depicting the benefits of our algorithm with regards to sample efficiency, robustness against negative transfer, and performance increase when compared to state-of-the-art methods.

1. Introduction

Artificial Intelligence research is often concerned with learning how to solve a given task in a most efficient way. In the past, most research efforts have focused on learning individual tasks, often modeled as a discrete time sequential decision-making problem described as *Markov Decision Process (MDP)* (Puterman, 2014).

Many methods for solving this kind of decision-making processes are specified in the field of *Reinforcement Learning (RL)* (Sutton & Barto, 1998), which has seen a surge in interest over the last years because powerful algorithms emerged from the combination with *Deep Learning (DL)* approaches (Schmidhuber, 2015), resulting in the field of *Deep Reinforcement Learning (DRL)* (Arulkumaran, Deisenroth, Brundage, & Bharath, 2017; Li, 2017). In *RL*, an agent explores the space of possible strategies to solve a task in a given environment, receives feedback (a reward) on the outcome of the actions it takes and learns a behavior policy from its observations over time. A policy then maps each state to an action, which maximizes the accumulated reward over time. The standard *RL* framework is shown in Fig. 1.

Although *RL* has been successfully used to autonomously solve complex tasks, like classic board games (Tesauro, 1995), robot soccer (Stone & Sutton, 2001), or autonomous helicopter flight (Ng et al., 2006), learning still takes a long time. That is still true for more recent approaches as for example for Atari computer games (Mnih et al., 2015), First-Person Shooter (Feng & Tan, 2016), or navigating complex environments (Mirowski et al., 2017). To learn to solve the most complex domains requires large computational resources and years of non-stop real-time experiences (heavily parallelized), as for example in real-time strategy games (Vinyals et al., 2017). This is due to the fact that agents applying *RL* techniques require a large number of samples of interactions with the environment to infer an effective solution policy even for simple tasks.

The field of *Transfer learning (TL)* (Taylor & Stone, 2009) emerged from the efforts towards solving this kind of challenges and consists of expanding *Machine Learning* algorithms with the ability to transfer knowledge across tasks to allow learning of multiple tasks more efficiently by reusing previously acquired knowledge (Pan & Yang, 2010). The idea of using accumulated knowledge in this way is inspired by the human transfer learning abilities

* Corresponding author.
E-mail addresses: ruben.glatt@usp.br (R. Glatt), f.leno@usp.br (F.L. Da Silva), rbianchi@fei.edu.br (R.A. da Costa Bianchi), anna.reali@usp.br (A.H.R. Costa).

https://doi.org/10.1016/j.eswa.2020.113420
0957-4174/

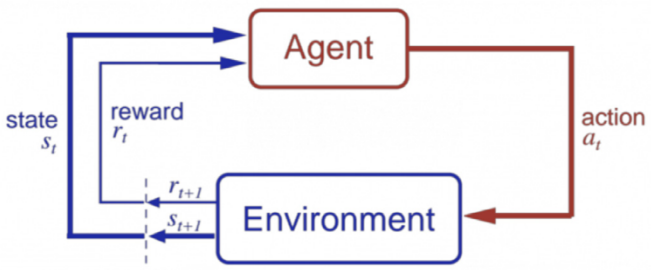

Fig. 1. A standard *Reinforcement Learning* setup: after performing an action, an agent receives a reward and an update of the current state from the environment. Source: Adapted from Sutton and Barto (1998).

which work quite similar. The goal is to create intelligent agents that can learn a variety of tasks by engaging in continuous or even lifelong learning (Thrun & Mitchell, 1995) in the form of multitask learning (several tasks at the same time) or sequential learning (one task after another), which is our focus here. This kind of setup compared to individual task learning is sketched out in Fig. 2.

Case-based Reasoning (CBR) (Aamodt & Plaza, 1994) describes a methodology to build computational models to reuse existing knowledge in a general manner (Watson, 1999). The guiding assumption for *CBR* is that similar problems have similar solutions. It has been shown to be very successful and has been widely applied in a number of fields (Cheetham & Watson, 2005). Although pursuing the same goal of properly reusing previously acquired knowledge, the *CBR* community has relatively few publications considering the acceleration of *RL* methods, maybe because differences in formulations and vocabulary hinder the exchange of ideas and methods between the communities.

In order to bridge the gap between the areas of *RL* and *CBR*, we here describe a general framework by specifying *CBR* using the terminology of *RL* problems. We also propose an implementation of this framework for *DRL* hereafter termed *Deep Case-based Policy Inference (DECAF)*, which estimates the usefulness of previously learned policies and autonomously switches the control between the new and already existing policies according to an importance measure that is updated during training. Therefore, the contribution of this paper is threefold:

1. We formulate the general *CBR* cycle in *RL* terminology, so as to facilitate the communication between the *RL* and *CBR* communities;
2. We propose the *DECAF* algorithm, which is an implementation of the framework for reusing knowledge for complex *DRL* tasks; and
3. We empirically show the benefits of *DECAF* over existing approaches.

This work presents an extension of our workshop paper (Glatt, da Silva, & Costa, 2017) to include *DECAF* to solve tasks in more complex domains that require *DRL* and make the whole framework applicable to a much wider range of more demanding applications.

2. Foundations and problem statement

An 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 provides a reward for applying action a in state s. 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'. The agent's goal is to learn an optimal policy π^*, that dictates the best action in any state.

A possible way to solve decision-making problems with *RL* is through the *Q-Learning* algorithm (Watson, 1999), which iteratively estimates a quality function $Q : S \times A \rightarrow \mathbb{R}$ for each possible action in each state. *Q-learning* has been proven to eventually converge to an optimal value $Q^*(s, a) = E\left[\sum_{i=0}^{\infty} \gamma^i r_i\right]$, where γ is a discount factor, which can be used to derive an optimal policy. In simple problems, the *Q-value* can be saved in a table with a row for each state and a column for each action to easily find a mapping from state to action over the entire state-action space. For complex problems, where it is infeasible to explicitly list the state-action space, a parametric function approximator might be used to approximate this *Q-table*. This has been published as *Deep Q-Network (DQN)* which implements a *Deep Neural Network (DNN)* architecture as sketched in Fig. 3 and has been shown to work very well in high-dimensional problem spaces (Mnih et al., 2015).

However, both regular *Q-learning* and *DQN*, suffer from sample inefficiency and therefore need extensive training processes. As an approach to alleviate this issue, knowledge from previous tasks might be reused in a *TL* setting. Even though directly copying network weights works under certain circumstances (Glatt, Silva, & Costa, 2016), it may result in *negative transfer* in which learning is even further slowed down instead of accelerated (Taylor & Stone, 2009) if the source and target tasks are not similar enough. Therefore, a more robust approach is needed for most applications. The *TL* area has implemented knowledge transfer for *RL* in many ways (Taylor & Stone, 2009), such as transferring knowledge in the form of options (Konidaris, Scheidwasser, & Barto, 2012), by transferring artificial neural network weights (Glatt et al., 2016),

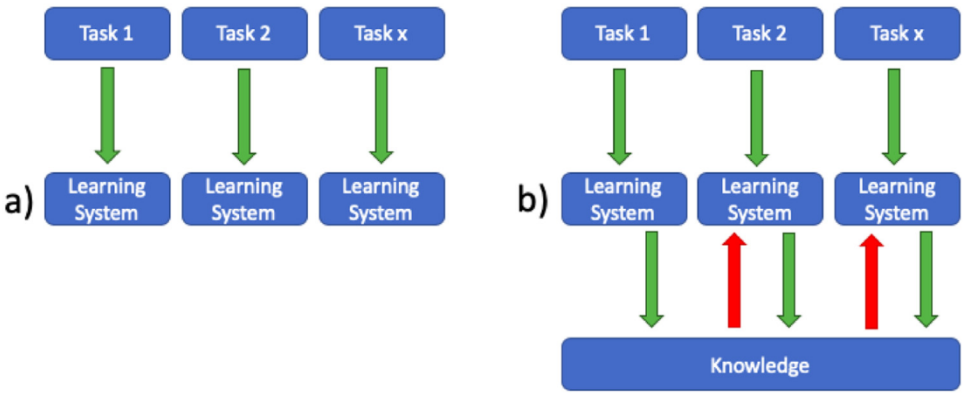

Fig. 2. The different setups for sequential task learning when *(a)* using no transfer learning and *(b)* using transfer learning to share knowledge across tasks.

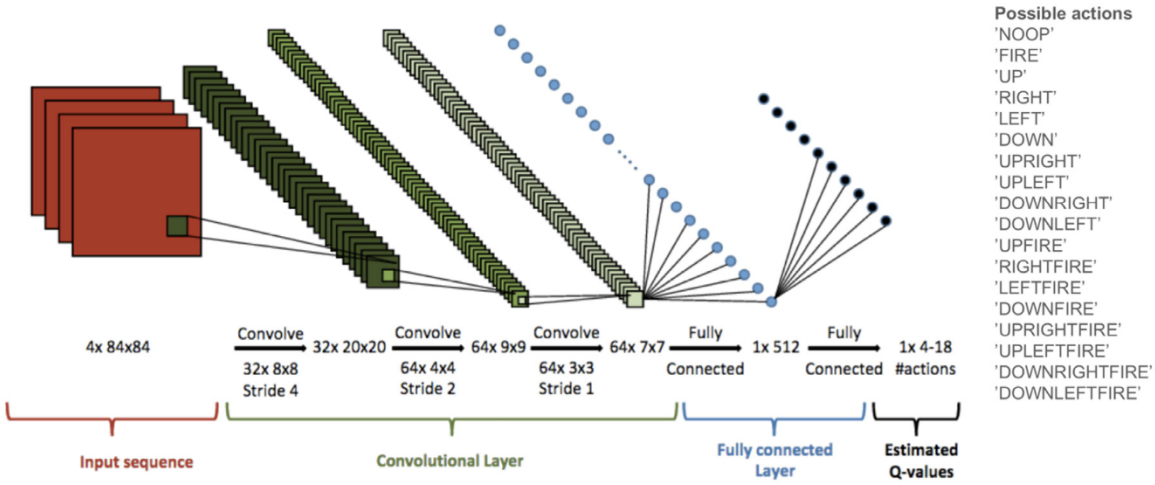

Fig. 3. Sketch of the architecture of the original *DQN* with three convolutional layers and two fully connected layers, where the nodes in the highest layer each represent a *state-action* value for all available actions. Source: Adapted from Mnih et al. (2015).

or through advice from other agents (Silva, Glatt, & Costa, 2017). Many of the works in *TL for RL* cover only specific aspects of the transfer while few works take a more systematic approach, as proposed by Glatt and Costa (2017), to build an agent that is able to gather more knowledge over time and reuse it in a directed manner.

One way that describes a systematic approach is the *CBR* methodology, which describes how knowledge can be reused and provides a general framework to solve learning challenges. A high-level description of the *CBR* cycle describes four stages during the learning of a new task with previously acquired knowledge (Aamodt & 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 learned solution in the knowledge base or discard it. *CBR* provides a clear framework for *TL* problems but nevertheless the *RL* community has only taken little interest in its methodology probably in part because of the little overlap in terminology which makes finding similar works more difficult.

3. Case-based Reasoning in Reinforcement Learning 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,

$$\vartheta := \langle \Omega_\vartheta, \pi_{\Omega_\vartheta} \rangle. \tag{1}$$

Instead of saving a policy π it is also possible to save a representation of the *Q-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

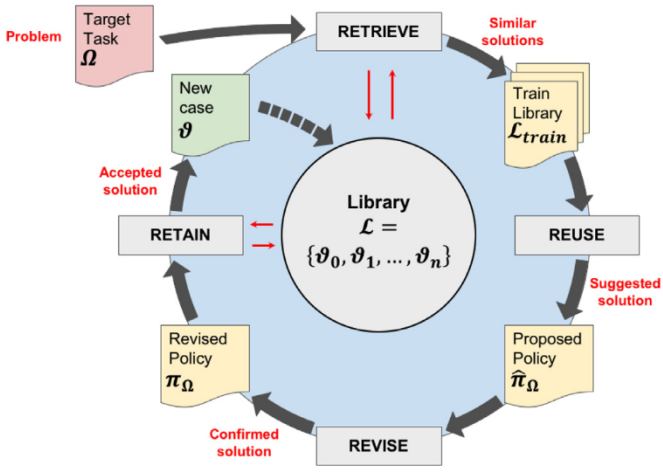

Fig. 4. A view on the *CBR* cycle adapted to RL terminology.

we are only discussing the *policy-based* formulation because the *value-based* formulation could be easily derived from that using $\pi_{\Omega_\vartheta} = \arg\max_a Q_{\Omega_\vartheta}$.

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

$$\mathcal{L} := \{\vartheta_0, \vartheta_1, ..., \vartheta_n\}, \tag{2}$$

where ϑ_i stands 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 Fig. 4.

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 intentionally wide here so that adaptions to other variants are easily possible, as for example by Aha (1998), 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.

3.1. 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. This could be achieved in a reduction function $\mathcal{L}_{train} = reduce(\mathcal{L}_{train})$ by simply selecting the most similar tasks up to l_{max} or define some condition to break a tie if similarities are the same.

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, Negny, & Montastruc, 2016), making it necessary to use different similarity functions for different tasks while allowing to add policies for very different tasks in the case base. For convenience, usually a domain and task specific similarity function sim_{task} and a similarity threshold $\sigma_{task} \in [0, 1]$ are defined to help fulfill the given criteria, with

$$sim_{task} : \Omega \times \Omega \rightarrow [0, 1]. \tag{3}$$

The main purpose of the *RETRIEVE* stage (see Algorithm 1) is properly selecting the training library $\mathcal{L}_{train}$, with

$$\mathcal{L}_{train} \leftarrow RETRIEVE(\mathcal{L}, \Omega, sim_{task}, \sigma_{task}, l_{max}). \tag{4}$$

3.2. 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 (Fernández & Veloso, 2006; Koga, Freire, & Costa, 2015). 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,

$$\hat{\pi}_\Omega \leftarrow REUSE(\mathcal{L}_{train}, \Omega). \tag{5}$$

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 & Veloso, 2006).

Algorithm 1 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}

$\mathcal{L}_{train} \leftarrow \emptyset$
for $\forall \vartheta \in \mathcal{L}$ **do**
  if $sim_{task}(\Omega, \Omega_\vartheta) \geq \sigma_{task}$ **then**
    $\mathcal{L}_{train} \cup \{\langle \Omega_\vartheta, \pi_{\Omega_\vartheta} \rangle\}$ sorted by similarity
if $|\mathcal{L}_{train}| > l_{max}$ **then**
  $\mathcal{L}_{train} = reduce(\mathcal{L}_{train})$
return $\mathcal{L}_{train}$

3.3. 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,

$$\pi_\Omega \leftarrow REVISE(\Omega, \hat{\pi}_\Omega). \tag{6}$$

Here, π_Ω resolves to the optimal policy π^*_Ω given sufficient training time because the training is not influenced by other policies anymore 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 *REVISE* 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.

3.4. 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 $\sigma_{policy} \in [0, 1]$,

$$sim_{policy} : \Pi \times \Pi \rightarrow [0, 1], \tag{7}$$

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 2) is to select and maintain relevant knowledge by updating the library $\mathcal{L}$ by comparing the new policy with existing policies,

$$\mathcal{L} \leftarrow RETAIN(\mathcal{L}, \mathcal{L}_{train}, \langle \Omega, \pi_\Omega \rangle, sim_{policy}, \sigma_{policy}). \tag{8}$$

3.5. Summary

The whole *CBR* cycle can be described in our framework using the formulations from above as shown in Algorithm 3. The library can initially already contain cases $\mathcal{L} = \{\vartheta_0, \vartheta_1, \ldots, \vartheta_n\}$ 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 2 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}

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

Algorithm 3 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: $\mathcal{L} \leftarrow RETAIN(\mathcal{L}, \mathcal{L}_{train}, \langle \Omega, \pi_{\Omega} \rangle, sim_{policy}, \sigma_{policy})$

4. Proposed algorithms

In this section, we propose two algorithms. The first one, *Case-based Policy Inference (CBPI)* is tailored to tasks that can be solved through tabular *RL* and was originally proposed in a workshop contribution (Glatt et al., 2017). The second one is the extension *Deep Case-based Policy Inference (DECAF)*, which is built to work with more complex domains that require the use of *DRL*.

4.1. Case-based Policy Inference (CBPI)

The core concept of this algorithm is based on the idea of generalizing over previously acquired knowledge and reusing it to guide the agent during training. The algorithm is based on the tabular form of *Q-Learning* and aims at finding a policy by learning the *Q-values* and then selecting the action with the highest *Q-value* in every state,

$$\forall s \in S : \pi_{\Omega}(s) = \arg\max_{a \in A} Q_{\Omega}(s, a). \tag{9}$$

In the *RETRIEVE* stage, the algorithm selects a number of previously trained policies according to a similarity measure that is task specific and will be described for our concrete example in the experiment section, Section 5.

The *REUSE* and *REFINE* stages are merged and the algorithm replaces the normally separate stages by gradually switching the control from the library over to the agent that is currently being trained. This is done by evaluating the usefulness, or *importance*, of each of the source policies π_{Ω_i} for the target task at the beginning of the training and then updating it with respect to the improvement and *importance* of the target policy π_{Ω} during training.

The *importance factor* $P_{imp}(\pi_{\Omega_k})$ follows the assumption that performance in a task is a good indicator for the usefulness of a policy for the current task. Therefore, we here define it as

$$P_{imp}(\pi_{\Omega_k}) = \frac{e^{-\bar{R}_{\pi_{\Omega_k}} * \tau_{policy}}}{\sum_j e^{-\bar{R}_{\pi_{\Omega_j}} * \tau_{policy}}}, \tag{10}$$

where $j = [0, \ldots, |\mathcal{L}_{train}| + 1]$ initially. This *softmax* function provides us with a probability for each policy in $\mathcal{L}_{train}$ and the current policy.

The average reward of each source policy $\bar{R}_{\pi_{\Omega_i}}$ is calculated once at the beginning of training for every selected source policy π_{Ω_i} while the average reward for the current policy $\bar{R}_{\pi_{\Omega}}$ is updated after every training episode. With increasing performance for π_{Ω} the *importance* should increase for the current policy while decreasing for all other policies. The temperature factor τ_{policy} is decreasing over time raising the sensitivity for small differences in $\bar{R}_{\pi}$ also helping to balance exploration and exploitation.

The *importance factor* $P_{imp}(\pi_{\Omega_k})$ is then multiplied with each respective $Q_{\Omega_k}(s, a)$ for all available policies and subsequently all these weighted *Q-values* are added together and the resulting *Q-values* are used to select the action during training. When the *importance factor* of a source policy falls below a given threshold, it is removed from the training library so that the algorithm eventually only trains on and refines the target policy as it improves.

In the *RETAIN* stage, the final policy is compared to the policies from the original training library and is only added to the library if it improves the performance on the task considering a threshold factor that is added to the performance of the other tasks.

4.2. Deep Case-based Policy Inference (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 (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 (Silva & Costa, 2017; Taylor, Stone, & Liu, 2007); but the advantage of the 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 4. 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 *RETRIEVE* (L.1) and *RETAIN* (L.18) 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.4-16). This idea is based on the *A2T* network

Algorithm 4 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 $\pi_{train}(s; \theta_{\Omega}, \theta_{\mathbf{w}})$ (Eqs. (11) and (12))
8: Observe reward r and follow-up state s'
9: Save experience $< s, a, r, s' >$ in replay memory $\mathcal{M}$
10: **with** minibatch from $\mathcal{M}$ **do**
11: Update training target $Y_{train}(s', r; \theta_{\Omega}^{-}, \theta_{\mathbf{w}}^{-})$ (Eq. (13))
12: Update importance network $\mathbf{w}(s, a; \theta_{\mathbf{w}})$ (Eq. (14))
13: Update training target network $Q_{\Omega}(s, a; \theta_{\Omega})$ (Eq. (15))
14: **if** Update interval **then**
15: Set target network parameter: $\theta_{\Omega}^{-} \leftarrow \theta_{\Omega}$, $\theta_{\mathbf{w}}^{-} \leftarrow \theta_{\mathbf{w}}$
16: $s \leftarrow s'$
17: $\pi_{\Omega} \leftarrow \pi_{train}$
18: $\mathcal{L} \leftarrow RETAIN(\mathcal{L}, \mathcal{L}_{train}, \langle \Omega, \pi_{\Omega} \rangle, sim_{policy}, \sigma_{policy})$

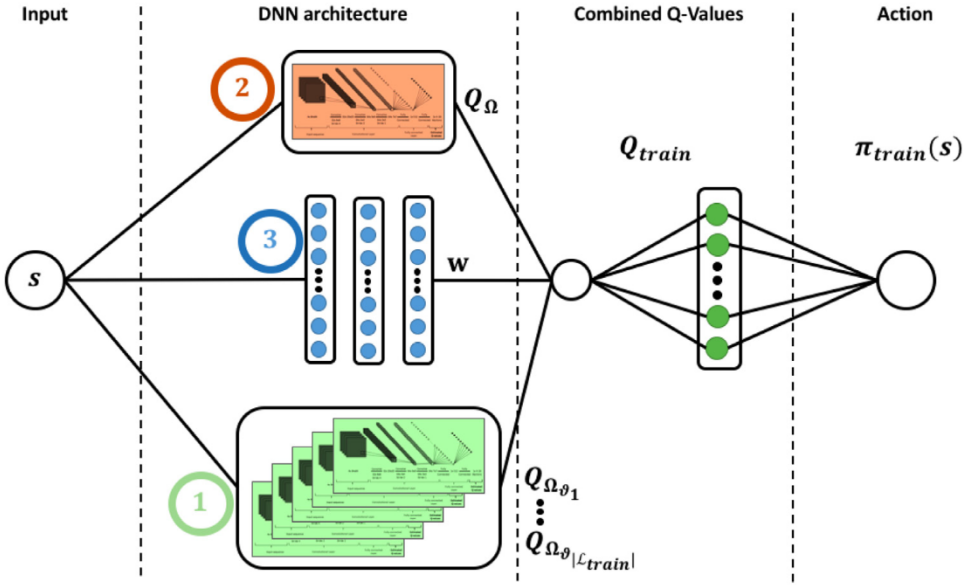

Fig. 5. Sketch of network architecture that produces the guiding training policy $\pi_{train}(s, \cdot)$ for a given input state s: (1) Selected source networks $Q_{\Omega_\vartheta}(s, \cdot)$ from $\mathcal{L}_{train}$, (2) network $Q_\Omega(s, a; \theta_\Omega)$ that we want to learn, and (3) importance network $\mathbf{w}(s, a; \theta_\mathbf{w})$.

proposed by Rajendran, Lakshminarayanan, Khapra, Prasanna, and Ravindran (2017) (also see discussion in Section 6, Related Works).

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 (Fig. 5). 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 $Q_{\Omega_\vartheta}(s, \cdot)$ 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 by linear or non-linear function approximators as long as they have the same input and output shape as required by the domain, but we here use the same non-linear 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 this non-linear deep 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 non-linear 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 $Q_{\Omega_\vartheta}(s, a)$ $(j = 1...|\mathcal{L}_{train}|)$ 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 linear combination of the three pillars with the *Q-values* of each source network in the first pillar and the target network in the second pillar multiplied by their respective importance value in the third pillar as coefficients. More specifically, each network output of the first and second pillar is weighted with the respective w_j value from the importance network and combined to

$$Q_{train}(s, \cdot) = Q_\Omega(s, a; \theta_\Omega) w_0(s, a; \theta_\mathbf{w}) + \sum_{j=1}^{|\mathcal{L}_{train}|} Q_{\Omega_j}(s, \cdot) w_j(s, a; \theta_\mathbf{w}), \tag{11}$$

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

$$\pi_{train}(s, \cdot) = \arg\max_{a \in A} Q_{train}(s, \cdot). \tag{12}$$

In our Algorithm 4, 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.6-16), 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 $< s, a, r, s' >$ in a *replay memory* $\mathcal{M}$ (L.9). The *replay memory* is a way to select the samples for the network parameter updates randomly 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 (L.10–13). 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.3,15). The training target for the updates is defined as

$$Y_{train}(s', r; \theta^-_\Omega, \theta^-_\mathbf{w}) = (r + \gamma \max_{a'} Q_{train}(s', a'; \theta^-_\Omega, \theta^-_\mathbf{w})). \tag{13}$$

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

$$L_\mathbf{w}(\theta_\Omega, \theta_\mathbf{w}) = \mathbb{E}[(Y_{train}(s', r; \theta^-_\Omega, \theta^-_\mathbf{w}) - Q_{train}(s, a; \theta_\Omega, \theta_\mathbf{w}))^2] \quad \textit{and} \tag{14}$$

$$L_{\Omega}(\theta_{\Omega}, \theta_{\mathbf{w}}) = \mathbb{E}[(Y_{train}(s', r; \theta_{\Omega}^{-}, \theta_{\mathbf{w}}^{-}) - Q_{\Omega}(s, a; \theta_{\Omega}, \theta_{\mathbf{w}}))^2]. \quad (15)$$

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 training updates in this way lets the *importance network* 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.16). This training loop is executed until a stopping criterion is reached, which could be a maximum number of steps or 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 $\vartheta_{\Omega} =< \Omega, Q_{\Omega}(s, a; \theta_{\Omega}) >$ should be added to the library $\mathcal{L}$ (L.18). We will also give more information about this in the experimental evaluation section regarding our domain specific measure.

5. Experimental evaluation

In this section, we describe experiments that show the usefulness of the proposed algorithms from Section 4. We present tests in two domains – the Gridworld and the Atari domain – to show that the use of *CBR* has a stabilizing effect and, given the proper previous knowledge, speeds up learning.

5.1. Grid world domain

Here, we describe an implementation of the framework for a classic Gridworld domain using *CBPI*. Our Gridworld domain represents 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 of $R = 0$ 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* of the target positions in *Cartesian coordinates* (x, y) of the pretrained task and the new task in the grid under the assumption that targets closer together represent similar tasks,

$$sim_{task}(\Omega, \Omega_{\vartheta}) = 1 - \frac{\sqrt{(x_{\Omega} - x_{\Omega_{\vartheta}})^2 + (y_{\Omega} - y_{\Omega_{\vartheta}})^2}}{\sqrt{x_{grid\_max}^2 + y_{grid\_max}^2}}, \quad (16)$$

and a similarity threshold provided by a percentage of the grid diagonal, here $\sigma_{task} = 0.7$.

We compared our results with the standard *Q-Learning* algorithm for individual tasks and the *Policy Library Policy Reuse (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.

5.1.1. Learning a new policy

In this domain, 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 Fig. 6).

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 $\mathcal{L}_{2,3,4} = \{\pi_2, \pi_3, \pi_4\}$ (see Fig. 7).

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 $H = 100$ steps is performed).

As can be seen in Fig. 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 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 Fig. 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 *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.

5.1.2. 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. After training each task the score will always be highest for the new policy, but we only want to keep it if the new policy is not too similar to existing ones. Therefore, the similarity measure that we use here is the actual similarity of the policy by selecting a number n_{sim} of random states from the environment and comparing the action recommendation between the policies $\pi_{\Omega_{\vartheta}}$ from the training library $\mathcal{L}_{train}$ and the newly learned policy π_{Ω},

$$sim_{policy}(\Omega, \Omega_{\vartheta}) = \frac{\sum_{i=0}^{n_{sim}} (\pi_{\Omega_{\vartheta}} == \pi_{\Omega})}{n_{sim}}, \quad (17)$$

and only keep a newly trained policy if the overlap is less than 60% for all existing policies, resulting in a threshold factor of $\sigma_{policy} = 0.6$.

The results as shown in Fig. 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 hand, *PLPR* provides a set of 14 policies and seems to emphasize these con-

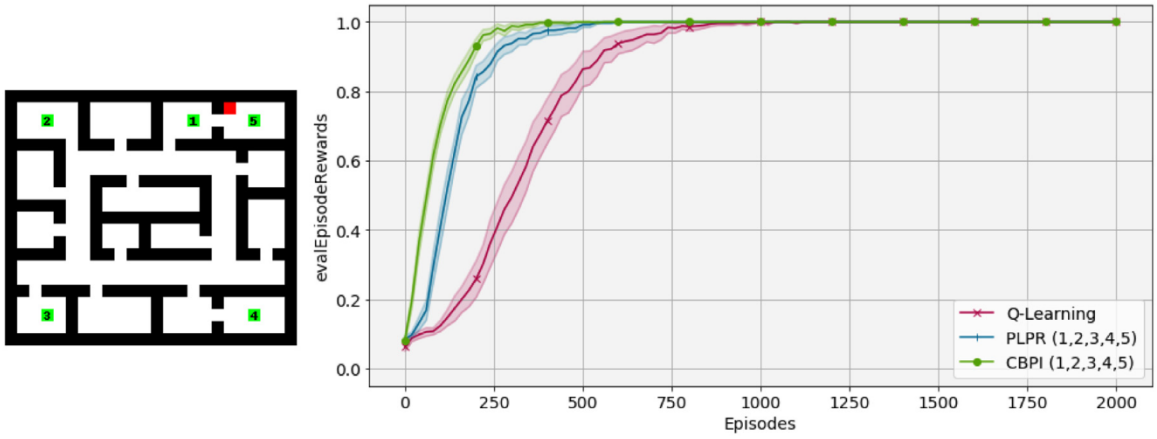

Fig. 6. Resulting rewards (right) when using a policy library that also contains policies from very similar tasks, Ω_1 and Ω_5 (left). The red square represents the goal position in the target task.

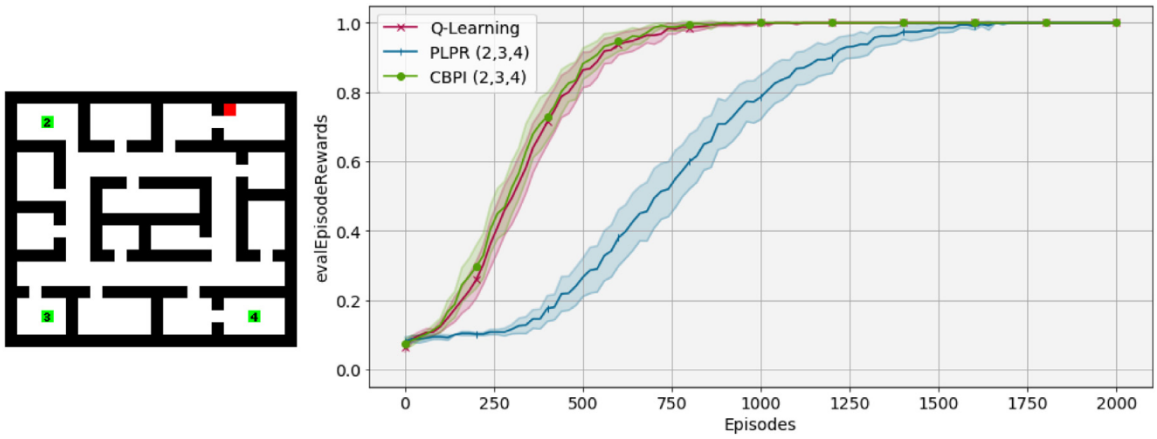

Fig. 7. Resulting rewards (right) when using a library only containing policies from very unrelated tasks, Ω_2, Ω_3, and Ω_4 (left). The red square represents the goal position in the target task.

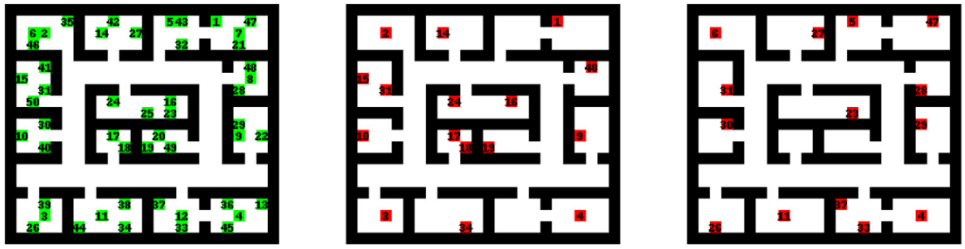

Fig. 8. The same grid world domain was evaluated for the library building experiment. The graphs show all 50 goal positions that were learned during the experiment (left) and the extracted core policies using *CBPI* (middle) and *PLPR* (right), respectively.

nection 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*.

5.2. Atari domain

In this section, 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 agents from the library
3. ***A2T (mixed)***: *A2T* using all agents from the library
4. ***DECAF***: Selects autonomously the best tasks for training and then applies *A2T* using only favorable tasks from the library

All graphs in this subsection 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

Fig. 9. Tasks contained in the knowledge base of experiment 1 from left to right: *Pong* and *SpaceInvaders*.

1080, V100) and operating systems (Ubuntu, CentOs).[1] In each run, the experiment was performed for 20 eras, where each era consists of a training and a test epoch. The training epoch consists of 250.000 steps in the environment and a test epoch of 10 full episodes in the environment which are used to report results in Figs. 10 and 12.

The network architecture and hyper-parameter settings follow the *DQN* implementation described by Mnih et al. (2015) and *A2T* is based on the work from Rajendran et al. (2017). The DQN networks each consist of an input layer, 4 hidden layers, and an output layer. The input layer takes in a 84*x*84*x*4 stacked image. The first hidden layer is a convolutional layer with 32 kernels of size 8*x*8 and a stride of 4, the second hidden layer is a convolutional layer with 64 kernels of size 4*x*4 and a stride of 2, the third hidden layer is a convolutional layer with 64 kernels of size 3*x*3 and a stride of 1, and the fourth hidden layer is a fully connected layer with 512 nodes. The output layer is fully connected and consists of nodes in the number of available actions for the given task, for example in the game *Pong* there are 6 actions available. All hidden layers have rectifier activation functions whereas the input and output layer have linear activations. The network is trained using RMSProp (Tieleman & Hinton, 2012) as optimizer with a learning rate of 0.00025 using batches of size 32.

5.2.1. 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 Fig. 9).

In Fig. 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 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.

5.2.2. 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 of six pretrained agents on *Pong* with obscured areas (upper half, middle, lower half)[2], *SpaceInvaders*, *Pooyan*, and *Qbert* (see Fig. 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.

Fig. 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 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 share the same observation space (RGB image of size 84 by 84 pixel), while the action space (6 actions but different for different tasks) is only similar. 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 be limited by the actual concrete actions that are available, leaving only the various *Pong* versions and *SpaceInvaders* here, following the similarity function

$$sim_{task}(\Omega, \Omega_{\vartheta}) = \begin{cases} 1, & \text{if same domain, state- and observation-space} \\ 0, & \text{else} \end{cases}. \quad (18)$$

The similarity threshold can be set to $\sigma_{task} = 1$ and will result in $\mathcal{L}_{train} = \{PongBlurredUpper, PongBlurredMiddle, PongBlurredLower, SpaceInvader\}$ which then can be reduced to $l_{max} = 3$ by performing 100 test episodes in the target task using each source policy and picking the highest performers according to average reward achieved (*PongBlurredUpper*: $R = 14.16$, *PongBlurred-*

[1] Full code, hyper-parameter settings, and results available on github at https://github.com/cowhi/pub_DECAF.

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

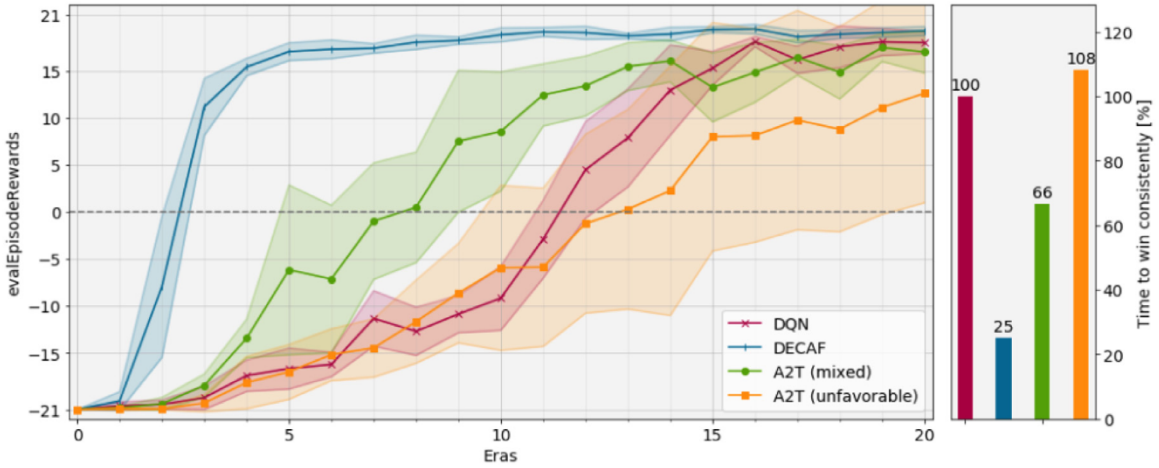

Fig. 10. Resulting rewards using different pretrained agents to highlight the importance of pre-selecting source tasks. The graph shows average and confidence intervals of the performance of 6 experiments per setting with 10 evaluation periods per era and setting.

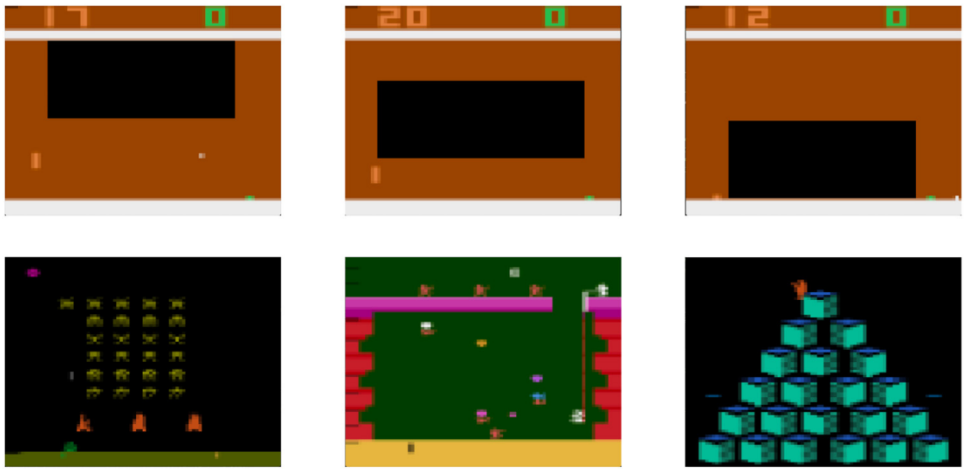

Fig. 11. Tasks contained in the knowledge base of experiment 2 from left to right, upper row: *Pong* with obscured areas (upper half, middle, lower half); lower row: *SpaceInvaders*, *Pooyan*, and *Qbert*.

Middle: $R = 6.57$, *PongBlurredLower*: $R = -12.21$, *SpaceInvader*: $R = -21.00$).[3]

In the *REUSE* 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 and describe the policy similarity as

$$sim_{policy}(\Omega, \Omega_{\vartheta}) = \begin{cases} 1, & \text{if } -R_{max} \leq R_{\Omega} \leq R_{\Omega_{\vartheta}\,\max} \\ \frac{R_{\Omega} - R_{max}}{R_{\Omega_{\vartheta}\,\max} - R_{max}}, & \text{if } R_{\Omega_{\vartheta}\,\max} < R_{\Omega} \leq R_{max} \end{cases}, \tag{19}$$

with $R_{max} = 21$ and an achieved average reward $R_{\Omega} = 18.52$ over 100 evaluation episodes. Following this result, the new policy was added to the policy library even when using a generous threshold factor of $\sigma_{policy} = 0.5$ to compare against the best performing already existing case with $R_{\Omega_{\vartheta}\,\max} = 14.16$ (*PongBlurredUpper*).

6. Related work

The creation of a policy library to reuse previously learned findings has been investigated in the past already. We here introduce some of the most related works and provide a differentiation to our approach.

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. Sharma et al. (2007) propose a *CBR* framework that is used as an instance-based state function approximator in a layered *RL* architecture. Similarly, Gabel and Riedmiller (2005) also make use of *CBR* for function approximation. We, on the other hand, directly use previously learned policies for accelerating learning in the new task.

Bianchi, Ros, and De Mantaras (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. While an extension of this work by Bianchi, Santos, da Silva, Celiberto, and de Mantaras (2018) also takes into consideration transferring knowledge within a domain, they consider cases at every step making the approach computationally expensive.

Jiang and Sheng (2009) propose a *Case-based Reinforcement Learning (CRL)* algorithm for dynamic inventory control. Their focus is on combining *CRL* with multi-agent systems in a *Supply-Chain Management (SCM)* setting under non-stationary conditions. Fang and Wong (2010) also focus their efforts on a *SCM* application, where they use a case base for the pre- and post-negotiation phases to support adaptive negotiation strategies. Both approaches

[3] 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.

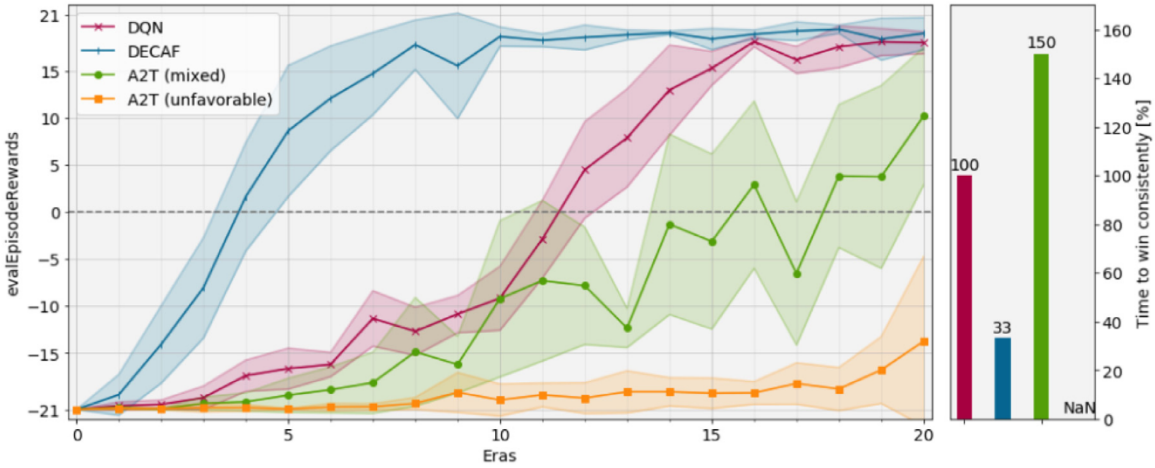

Fig. 12. Resulting rewards using different tasks from the policy library to show the positive effects of DECAF when dealing with a larger library. The graph shows average and confidence intervals of the performance of 6 experiments per setting with 10 evaluation periods per era and setting.

are concerned with a multi-agent setting while we are concerned with a single agent.

Other works focused more on the *RL* aspect without explicitly considering the *CBR* methodology.

As mentioned earlier, Fernández and Veloso (2006) introduced the *PLPR* algorithm that we used to compare our *CBPI* algorithm with. In their work, a library of core policies for a given domain is autonomously built and reused. While they select a whole policy to be followed 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, Narvekar, Leonetti, and Stone (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.

An interesting network architecture is 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, Hawasly, and Ramamoorthy (2016) also propose an algorithm for *policy reuse* that selects policies during training according to a 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 *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.

Some publications have focused more on applied problems for using *CBR* in clinical settings. Marie et al. (2019) combine *CBR* with *DNN*s to present a system to conduct the segmentation of tumoral kidneys to build a library of stored cases for better detection of cancerous regions. During training they only use a single case while we select a number of cases to guide the training process. Another work by Torrent-Fontbona, Massana, and López (2019) describe a *CBR* system to provide recommendations of levels of insulin infusions for type 1 diabetes patients. Their approach focuses on maintaining the case-base efficient and accurate without training additional networks.

More works have used *CBR* in combination with techniques from other areas of computational intelligence. Cabrera and Sànchez-Marrè (2018) introduce a stochastic method for learning new cases for environmental data stream mining, Ali, Khatak, Chow, and Lee (2018) combine it with meta-learning and reasoning for selecting data mining classifiers, Abdelwahed, Saleh, and Mohamed (2018) use *CBR* to speed up single-query sampling-based algorithms, Mustapha (2018) combines it with methods for social media analysis to identify knowledge leaders in online communities, Craw and Aamodt (2018) add cognition and meta-cognition to drive more robust and explainable artificial intelligence, Ching-Hung, Chun-Hsien, Fan, and An-Jin (2019) combine it with the *Theory of Inventive Problem Solving* to accelerate customized innovative service design, and Silva, Carvalho, and Caminhas (2020) introduce a combination with the *Artificial Immune Systems* paradigm for fault detection and diagnosis in electric machines.

7. Conclusions and future work

We here proposed a framework to integrate *Case-based Reasoning* with *Reinforcement Learning* called *Deep Case-based Policy Inference* which consistently reuses gathered knowledge from previously learned tasks in order to accelerate learning of a new target task while building a growing library of policies over time. *DECAF* exploits previously learned policies from cases that are similar to the target task and blends those policies during training, progressively switching the control to the new target policy.

We compared *DECAF* with *A2T* and *Vanilla DQN* to show that our proposal has benefits in scenarios with advantageous previ-

ous knowledge compared to single task learning (*DQN*) and also over other transfer methods (*A2T*) in scenarios with disadvantageous knowledge, being more robust against negative transfer.

The results reported here show the usefulness of our approach for sequential task learning combining the *CBR* framework with *DRL*. Future works on this framework could include better initialization of the random networks which we are training for the target policy using network distillation from previous learned tasks, but also ways that could compress knowledge from different tasks in a meta-policy to reduce hardware requirements and improve generalization. Another important aspect that could improve the framework would be to define similarity between tasks in a more general way that might not necessarily be domain specific which could be based on an object-oriented representation (Silva, Glatt, & Costa, 2019).

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Credit authorship contribution statement

Ruben Glatt: Conceptualization, Methodology, Software, Writing - original draft. **Felipe Leno Da Silva:** Methodology, Validation, Writing - review & editing. **Reinaldo Augusto da Costa Bianchi:** Resources, Writing - review & editing, Supervision. **Anna Helena Reali Costa:** Writing - review & editing, Supervision, Funding acquisition.

Acknowledgments

Funding: This work was supported by CAPES; CNPq [grant 311608/2014-0]; and FAPESP [grants 2015/16310-4, 2016/18792-9, 2016/21047-3, 2018/00344-5].

References

Aamodt, A., & Plaza, E. (1994). Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI communications, 7*(1), 39–59.

Abdelwahed, M. F., Saleh, M., & Mohamed, A. E. (2018). Speeding up single-query sampling-based algorithms using case-based reasoning. *Expert Systems with Applications, 114*, 524–531.

Aha, D. W. (1998). The omnipresence of case-based reasoning in science and application. *Knowledge-based systems, 11*(5), 261–273.

Ali, R., Khatak, A. M., Chow, F., & Lee, S. (2018). A case-based meta-learning and reasoning framework for classifiers selection. In *Proceedings of the 12th international conference on ubiquitous information management and communication* (p. 31). ACM.

Arulkumaran, K., Deisenroth, M. P., Brundage, M., & Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine, 34*(6), 26–38.

Bianchi, R. A. C., Ros, R., & De Mantaras, R. L. (2009). Improving reinforcement learning by using case based heuristics. In *Proceedings of the 8th international conference on case-based reasoning (ICCBR)* (pp. 75–89). Springer.

Bianchi, R. A. C., Santos, P. E., da Silva, I. J., Celiberto, L. A., & de Mantaras, R. L. (2018). Heuristically accelerated reinforcement learning by means of case-based reasoning and transfer learning. *Journal of Intelligent & Robotic Systems, 91*(2), 301–312.

Cabrera, F. O. n., & Sànchez-Marrè, M. (2018). Environmental data stream mining through a case-based stochastic learning approach. *Environmental modelling & software, 106*, 22–34.

Chazara, P., Negny, S., & Montastruc, L. (2016). Flexible knowledge representation and new similarity measure: Application on case based reasoning for waste treatment. *Expert Systems with Applications, 58*, 143–154.

Cheetham, W., & Watson, I. (2005). Fielded applications of case-based reasoning. *The Knowledge Engineering Review, 20*(03), 321–323.

Ching-Hung, L., Chun-Hsien, C., Fan, L. I., & An-Jin, S. (2019). Customized and knowledge-centric service design model integrating case-based reasoning and TRIZ. *Expert Systems with Applications*, 113062.

Craw, S., & Aamodt, A. (2018). Case based reasoning as a model for cognitive artificial intelligence. In *International conference on case-based reasoning* (pp. 62–77). Springer.

Fang, F., & Wong, T. N. (2010). Applying hybrid case-based reasoning in agent-based negotiations for supply chain management. *Expert Systems with Applications, 37*(12), 8322–8332.

Feng, S., & Tan, A.-H. (2016). Towards autonomous behavior learning of non-player characters in games. *Expert Systems with Applications, 56*, 89–99.

Fernández, F., & Veloso, M. (2006). Probabilistic policy reuse in a reinforcement learning agent. In *Proceedings of the 5th international conference on autonomous agents and multiagent systems (AAMAS)* (pp. 720–727).

Gabel, T., & Riedmiller, M. (2005). Cbr for state value function approximation in reinforcement learning. In *Proceedings of the 6th international conference on case-based reasoning (ICCBR)* (pp. 206–221). Springer.

Glatt, R., & Costa, A. H. R. (2017). Policy reuse in deep reinforcement learning. In *Thirty-first AAAI conference on artificial intelligence* (pp. 4929–4930).

Glatt, R., da Silva, F. L., & Costa, A. H. R. (2017). Case-based policy inference for transfer in reinforcement learning. In *Workshop on scaling-up reinforcement learning at the 28th european conference on machine learning (ECML)* (pp. 1–8).

Glatt, R., Silva, F. L. d., & Costa, A. H. R. (2016). Towards knowledge transfer in deep reinforcement learning. In *Proceedings of the 5th brazilian conference on intelligent systems (BRACIS)* (pp. 91–96). IEEE.

Hullermeier, E. (2007). Credible case-based inference using similarity profiles. *IEEE Transactions on Knowledge and Data Engineering, 19*(6), 847–858.

Jiang, C., & Sheng, Z. (2009). Case-based reinforcement learning for dynamic inventory control in a multi-agent supply-chain system. *Expert Systems with Applications, 36*(3), 6520–6526.

Koga, M. L., Freire, V., & Costa, A. H. R. (2015). Stochastic abstract policies: Generalizing knowledge to improve reinforcement learning. *IEEE Transactions on Cybernetics, 45*(1), 77–88.

Kolodner, J. (2014). *Case-based reasoning*. Morgan Kaufmann.

Konidaris, G., Scheidwasser, I., & Barto, A. G. (2012). Transfer in reinforcement learning via shared features. *Journal of Machine Learning Research (JMLR), 13*(1), 1333–1371.

Li, Y. (2017). Deep reinforcement learning: An overview. *arXiv e-prints*. arXiv:1701.07274

Marie, F., Corbat, L., Chaussy, Y., Delavelle, T., Henriet, J., & Lapayre, J.-C. (2019). Segmentation of deformed kidneys and nephroblastoma using case-based reasoning and convolutional neural network. *Expert Systems with Applications, 127*, 282–294.

Mirowski, P., Pascanu, R., Viola, F., Soyer, H., Ballard, A. J., Banino, A., & Hadsell, R. (2017). Learning to navigate in complex environments. In *Proceedings of the 5th international conference on learning representations (ICLR)*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature, 518*(7540), 529–533.

Mustapha, S. S. (2018). Case-based reasoning for identifying knowledge leader within online community. *Expert Systems with Applications, 97*, 244–252.

Ng, A. Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., … Liang, E. (2006). Autonomous inverted helicopter flight via reinforcement learning. In *Experimental robotics IX* (pp. 363–372). Springer.

Pan, S. J., & Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering, 22*(10), 1345–1359.

Puterman, M. L. (2014). *Markov decision processes: discrete stochastic dynamic programming*. Hoboken, NJ, USA: John Wiley & Sons.

Rajendran, J., Lakshminarayanan, A., Khapra, M. M., Prasanna, P., & Ravindran, B. (2017). Attend, adapt and transfer: Attentive deep architecture for adaptive transfer from multiple sources in the same domain. In *Proceedings of the 5th international conference on learning representations (ICLR)*.

Rosman, B., Hawasly, M., & Ramamoorthy, S. (2016). Bayesian policy reuse. *Machine Learning, 104*(1), 99–127.

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., … Hadsell, R. (2016). Progressive neural networks. *arXiv e-prints*. arXiv:1606.04671

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

Sharma, M., Holmes, M. P., Santamaría, J. C., Irani, A., Isbell Jr, C. L., & Ram, A. (2007). Transfer learning in real-time strategy games using hybrid CBR/RL. In *Proceedings of the 20th international joint conference on artificial intelligence (IJCAI): vol. 7* (pp. 1041–1046).

Silva, F. L. D., & Costa, A. H. R. (2017). Towards Zero-Shot Autonomous Inter-Task Mapping through Object-Oriented Task Description. In *Proceedings of the 1st workshop on transfer in reinforcement learning (tiRL)* (pp. 1–10).

Silva, F. L. d., Glatt, R., & Costa, A. H. R. (2017). Simultaneously learning and advising in multiagent reinforcement learning. In *Proceedings of the 16th international conference on autonomous agents and multiagent systems (AAMAS)* (pp. 1100–1108).

Silva, F. L. d., Glatt, R., & Costa, A. H. R. (2019). Moo-mdp: An object-oriented representation for cooperative multiagent reinforcement learning. *IEEE Transactions on Cybernetics, 49*(2), 567–579. doi:10.1109/TCYB.2017.2781130.

Silva, G. C., Carvalho, E. E. O., & Caminhas, W. M. (2020). An artificial immune systems approach to case-based reasoning applied to fault detection and diagnosis. *Expert Systems with Applications, 140*, 112906.

Sinapov, J., Narvekar, S., Leonetti, M., & Stone, P. (2015). Learning inter-task transferability in the absence of target task samples. In *Proc. 14th international conference on autonomous agents and multiagent systems (AAMAS)* (pp. 725–733).

Stone, P., & Sutton, R. S. (2001). Scaling reinforcement learning toward robocup soccer. In *Proceedings of the 18th international conference of machine learning (ICML)* (pp. 537–544). ACM.

Sutton, R. S., & Barto, A. G. (1998). *Introduction to Reinforcement Learning*. Cambridge, MA, USA: MIT Press.

Taylor, M. E., & Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research, 10*, 1633–1685.

Taylor, M. E., Stone, P., & Liu, Y. (2007). Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research, 8*(1), 2125–2167.

Tesauro, G. (1995). Temporal difference learning and TD-gammon. *Communications of the ACM, 38*(3), 58–68.

Thrun, S., & Mitchell, T. M. (1995). *Lifelong robot learning*: vol. 15. Elsevier.

Tieleman, T., & Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning, 4*(2), 26–31.

Torrent-Fontbona, F., Massana, J., & López, B. (2019). Case-base maintenance of a personalised and adaptive CBR bolus insulin recommender system for type 1 diabetes. *Expert Systems with Applications, 121*, 338–346.

Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhnevets, A. S., Yeo, M., & Tsing, R. (2017). Starcraft ii: A new challenge for reinforcement learning. *arXiv e-prints*. arXiv:1708.04782

Watson, I. (1999). Case-based reasoning is a methodology not a technology. *Knowledge-based systems, 12*(5), 303–308.