university of groningen / faculty of science and engineering

Transfer of Experience Replay in Deep Reinforcement Learning

Bachelor's Project Thesis

Luca Mueller, s3745503, d.l.mueller@student.rug.nl,
Supervisor: Matthia Sabatelli, m.sabatelli@rug.nl

Abstract: Transfer Learning (TL) is an area of machine learning concerned with adjusting existing models to fit new data. Deep Reinforcement Learning (DRL) combines Reinforcement Learning (RL) and Deep Learning to solve games and real world problems that can be modeled as Markov Decision Processes (MDP). Since training of DRL algorithms is expensive and often not possible due to limited availability of data, this area would greatly benefit from TL. Attempts to transfer model parameters have not been successful. A possible solution is transferring trajectories stored in an experience replay buffer to provide training data from a different task, provided that both source and target task are similar in nature. Two environments from the OpenAI gym were chosen in combination with three DRL algorithms. Agents were trained on the provided environments as well as a slightly modified version of each environment. Transfer was tested from standard to modified environments and vice versa. For reference a transfer of model parameters and a transfer of both experience replay buffer and model parameters was included. The results show an improvement in one of the tasks, regardless of transfer method, and no significant difference in the second.

1 Introduction

Over the last decade Deep Reinforcement Learning (DRL) has been established as an advancement of Reinforcement Learning (RL). It builds on the same principles but uses Deep Learning (DL) to increase the capabilities of RL algorithms. This makes it possible to train agents in increasingly complex environments such as the Atari Learning Environment (ALE) (Bellemare, Naddaf, Veness, and Bowling, 2013) or the game StarCraft (Shantia, Begue, and Wiering, 2011; Vinyals, Babuschkin, Czarnecki, Mathieu, Dudzik, Chung, Choi, Powell, Ewalds, Georgiev, Oh, Horgan, Kroiss, Danihelka, Huang, Sifre, Cai, Agapiou, Jaderberg, and Silver, 2019).

One popular variant of RL is Q-learning (Watkins, 1989, as cited in Sutton and Barto, 1998). It is an off-policy temporal difference (TD) learning algorithm that learns a function for all state-action pairs based on the reward the agent receives. The resulting values, called Q-values, are used to determine an optimal policy for the agent. For discrete state-action spaces it is possible to store these Q-values in a lookup table. However, this becomes difficult or even impossible when dealing with sufficiently large state-action spaces or continuous state spaces.

This is where the power of artificial neural networks (ANNs) comes into play. ANNs can be used as function approximators, in this case approximating the state-action value function. The input is a state (collected from the environment) and the outputs are the Q-values for each action. Instead of updating values in a table, a loss function is used to determine the error between the model output and the updated Q-value. Using backpropagation an ANN can thus approximate the state-action value function.

Another feature of DRL agents is experience replay (ER). In online TD learning an agent performs an action in the environment, resulting in a reward and a new state. After each action the corresponding Q-value is updated. However, this method is not suitable for training an ANN (Lin, 1993, as cited in Mnih, Kavukcuoglu, Silver, Graves, Antonoglou, Wierstra, and Riedmiller, 2013). Instead, the agent

stores this data in a buffer. At each step, a batch of transitions is sampled from the buffer. Then the loss is calculated for the entire batch, followed by backpropagation.

One proplem with DRL, and DL in general, is that it takes enormous amounts of data, time and computing power to train a model. This problem is adderssed by transfer learning (TL), which has been shown to aid training of models in different areas of DL, most prominently computer vision (Razavian, Azizpour, Sullivan, and Carlsson, 2014; Yosinski, Clune, Bengio, and Lipson, 2014). The idea is to make use of parameters optimized for a different but similar task. Using them as a starting point can significantly speed up training times.

These findings indicate that DRL, too, might benefit from TL. There has been success with TL for model-based RL (Sasso, Sabatelli, and Wiering, 2021) and actor-critic algorithms (Parisotto, Ba, and Salakhutdinov, 2015). However, in model-free settigns a transfer of model parameters between various ALE environments has not been proven to work (Sabatelli and Geurts, 2021). Another possible method to transfer knowledge from a source to a target domain is the transfer of an ER buffer. Transitions are collected by a trained agent in the source domain and subsequently transferred to a different (untrained) agent in the target domain. Ideally this would lead the agent in the target domain to converge faster and/or to a better solution than an agent starting with an empty buffer.

In the following sections we will investigate whether the transfer of an ER buffer can improve training of an agent in a similar environment. To test this, three DRL algorithms will be trained in four different environments. The first two environments are taken directly from the OpenAI gym (Brockman, Cheung, Pettersson, Schneider, Schulman, Tang, and Zaremba, 2016), the other two are slight modifications of the former two. In both cases the transfer happens from standard to modified environment and vice versa. For each DRL algorithm and source-target pair four agents are trained. The first learns without any transfer and is used as a baseline while the second receives a full ER buffer with data from the source domain. A third agent receives the model parameters of an agent trained in the source domain and a fourth receives both buffer and parameters.

Scores are recorded in between training steps and plotted for visual analysis. In addition the area ratio scores are calculated.

2 Background

2.1 Reinforcement Learning

RL tasks feature an agent and an environment. A state is observed by the agent, based on which it chooses and executes an action. That action results in a reward and a new state. The agent's task is to achieve some goal and in order to do so it has to learn a policy that guides its actions.

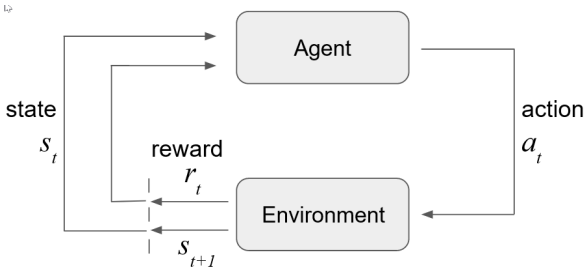


Figure 2.1: Reinforcement Learning cycle as shown in Sutton and Barto (1998)

Interactions take place in a sequence of discrete time steps, t, each of which has a state s_t, an action a_t and a reward r_t associated with it. At time step t the agent is in state $s_t \in \mathcal{S}$, where $\mathcal{S}$ is the state space. It performs an action, $a_t \in \mathcal{A}(s)$, leading to a new state s_{t+1} drawn from a probability distribution $p(s_{t+1}|s_t, a_t)$ and a reward $r_t \in \mathbb{R}$. From this the agent learns a policy, $\pi(s)$, which determines the probability of selecting an action based on the observed state.

This cycle repeats until the goal is reached or the process is interrupted, e.g. after a number of time steps. A series of time steps starting at t and ending at the final time step T is called an episode. After an episode is finished, the cumulative reward (also called return) is calculated by summing the rewards $\{r_t, r_{t+1}, r_{t+2}, ..., r_{T-1}\}$ received during the entire episode. The goal of the agent is to learn the optimal policy $\pi^*(s)$ that maximizes cumulative reward.

Shown below are the probabilities of observing a state s_{t+1} and receiving a reward r_t based on a history $s_0, a_0, r_0, s_{t-1}, a_{t-1}, r_{t-1}, ..., s_t, a_t$.

$$\Pr\{r_t = r, s_{t+1} = s' | s_0, a_0, r_0, s_{t-1}, a_{t-1}, r_{t-1}, ..., s_t, a_t\} \tag{2.1}$$

$$\Pr\{r_t = r, s_{t+1} = s' | s_t, a_t\} \tag{2.2}$$

If (2.1) is equal to (2.2) for all s', r and histories $s_0, a_0, r_0, s_{t-1}, a_{t-1}, r_{t-1}, ..., s_t, a_t$, then the task has the Markov property and can be modeled as a Markov Decision Process (MPD) (Puterman, 2014). More specifically, if the state space $\mathcal{S}$ and action space $\mathcal{A}(s)$ are finite it is a finite MDP. All tasks considered in this work are (finite) MDPs and can therefore be abbreviated $\mathcal{M}$.

Moving forward, the state space $\mathcal{A}(s)$ will be abbreviated $\mathcal{A}$ since for the environmets used the actions available in any given state are identical and therefore do not depend on the state s.

In order to determine which states are beneficial to the agent, state values can be calculated. They depend on the agent's policy π and the expected cumulative reward when follwing that policy from the given state. This state-value function can be formalized as

$$V^\pi(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \middle| s_t = s, \pi\right] \tag{2.3}$$

where $\mathbb{E}[.]$ is the expected discounted return when following policy π starting from state s and γ is the discount factor. The discount factor is a number in the range $[0, 1)$ where low values mean recent rewards are prioritized and high values mean distant rewards have more impact.

Likewise, an action-value function evaluating expected return of state-action pairs is described by

$$Q^\pi(s, a) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \middle| s_t = s, a_t = a, \pi\right] \tag{2.4}$$

The optimal state value function $V^*(s)$ and optimal state-action value function $Q^*(s, a)$ return the respective values under the optimal policy. In other terms, these are the functions that are approximated by equations (2.3) and (2.4).

One of the most famous algorithms in RL is Q-learning (Watkins, 1989, as cited in Sutton and Barto, 1998). It is a TD learning algorithm, which means that it learns by bootstrapping from existing estimates. The update equation for Q-values is given by

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) - \alpha[r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{2.5}$$

where α is the learning rate and the part after it is the TD error.

This has the advantage of allowing updates to the value function without knowing the cumulative reward of an episode. Q-learning is also characterized as an online method, meaning that updates to the Q-value function happen immediately after the agent takes a step in the environment. The simplest method to implement a Q-value function is to use a lookup table that stores the Q-values for all state-action pairs. Updates and predictions are quick, however memory requirements make this a poor choice for larger MDPs. Other ways of approximating $Q^*(s, a)$ are linear function approximation and DL. Another category that Q-learning is a part of is model-free RL. It learns values relating to states or state-action pairs by interacting with the environment to determine its policy. Model-based RL, on the other hand, uses an estimated model of the underlying MDP and learns from interacting with that model in addition to interactions with the environment.

Finally, an exploration policy is needed to ensure that the agent learns from what it has seen while also exploring new actions and states. The most common exploration policy is the ε-greedy policy, where ε denotes the probability of selecting a random action and greedy simply means choosing the best action according to the learned policy. Usually, ε is very high (close to 1) at the start of training and decays over time. This ensures that the agent tries out many possible actions while it has little training data and gradually chooses more actions based on its policy π.

2.2 Transfer Learning

Transfer Learning aims to make training of machine learning (ML) models more efficient. The idea is to use knowledge gained from one task to improve

learning performance in a different but similar task. This technique has been shown to be very successful in computer vision tasks (Razavian et al., 2014; Yosinski et al., 2014) where convolutional neural networks learn to extract features from image data which are also useful for tasks using different training data. Some examples of successful TL are pavement distress detection (Gopalakrishnan, Khaitan, Choudhary, and Agrawal, 2017) and monocular depth estimation (Alhashim and Wonka, 2018). Other fields of DL such as natural language processing also make use of TL (Ruder, Peters, Swayamdipta, and Wolf, 2019). DRL is another area of ML which could benefit from TL (Lazaric, 2012) as it often proves to be difficult and time consuming.

3 Methods

3.1 Agents

Three model free DRL agents are used for the experiments, namely DQN (Mnih et al., 2013), Double DQN (Van Hasselt, Guez, and Silver, 2016) and DQV (Sabatelli, Louppe, Geurts, and Wiering, 2018). All of them optimize an ANN with parameters θ to approximate the optimal Q-value function $Q^*(s, a)$. In addition to the policy model with parameters θ there is also a target model with parameters θ^-. It is used to calculate the future Q-values used by the loss function and synchronizes its parameters in a fixed interval with θ.

To train the model the quadratic loss of the TD error δ is calculated for mini-batches of transitions sampled uniformly from the ER buffer. The notation δ_θ means that the TD error is used to calculate the loss L_θ.

$$L_\theta = \mathbb{E}_{(s_t, a_t, r_t, s_{t+1}) \sim U(D)}[\delta_\theta^2] \tag{3.1}$$

The ER buffer consists of a memory buffer D of size N. It contains four-tuples of the shape $\langle s_t, a_t, r_t, s_{t+1} \rangle$ which are called transitions. This makes data usage more efficient as transitions can be used multiple times. Additionally, they are decorrelated due to uniform sampling. The size of the buffer used for the experiments is 10,000 and the size of the mini-batches sampled from it is 32. Training for the baseline agents begins with an empty buffer. Once the buffer is full, the oldest transitions are discarded in favor of newly collected ones.

Transfer happens between a source MDP $\mathcal{M}_S$ and a target MDP $\mathcal{M}_T$. For ER transfer, a buffer is filled by a trained agent acting on $\mathcal{M}_S$. Then it is transferred to an untrained agent that learns $\mathcal{M}_T$.

Finally, the MLPs used to approximate $Q^*(s, a)$ (and $V^*(s)$ for DQV) are small networks with two hidden layers of 32 nodes each.

3.1.1 DQN

DQN, first described by Mnih et al. (2013), was a breakthrough in model-free DRL achieving some of the best results int the Atari Learning Environment (ALE). Using a stack of four 84×84 greyscale images as state representations, it trains a convolutional neural network (CNN) that outputs Q-values for all actions. The same principle can be used with a multi layer perceptron (MLP) which is appropriate for the two selected tasks since their states are represented by four and six floating point numbers respectively.

ER plays a central role in the success of DQN. It allows the combination of supervised learning methods and reward signals which lack the consistency required of labels. Training ANNs with online methods is problematic for a number of reasons. As mentioned previously, online methods are not data efficient because they only consider the latest transition(s) and thus learn from strongly correlated transitions. Another problem is that samples collected online are biased by the agent's policy.

The TD error for DQN is calculated from the Q-value predicted by θ, the associated reward and the maximum Q-value for state s_{t+1} predicted by θ^-.

$$\delta_\theta = r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a; \theta^-) - Q(s_t, a_t; \theta) \tag{3.2}$$

3.1.2 Double DQN

Double DQN (Van Hasselt et al., 2016) is a slight variation of DQN that uses a different calculation for the future Q-value of the TD error. Where DQN chooses the maximum Q-value for the next state s_{t+1} according to the target model θ^-, Double DQN uses the policy model θ to find the action which maximizes the Q-value for state s_{t+1} and uses it to calculate the future Q-value.

$$\delta_\theta = r_t + \gamma Q(s_{t+1}, \arg\max_{a \in \mathcal{A}} Q(s_{t+1}, a; \theta); \theta^-) - Q(s_t, a_t; \theta) \tag{3.3}$$

3.1.3 DQV

DQV, introduced by Sabatelli et al. (2018), is based on QV(λ) (Wiering, 2005) and uses a state-value function $V(s)$ in addition to the action-value function $Q(s, a)$. This means a second ANN with its own set of parameters ϕ and loss function L_ϕ has to be trained.

$$L_\phi = \mathbb{E}_{(s_t, r_t, s_{t+1}) \sim U(D)}[\delta_\phi^2] \tag{3.4}$$

Both ANNs have the same architecture with exception of the last layer, which has only one output in the case of ϕ where θ has one output for every action.

The TD error δ_ϕ is calculated similar to δ_θ, replacing Q-values with V-values. This also eliminates the need for max and argmax operators as actions are not considered in the formula.

$$\delta_\phi = r_t + \gamma V(s_{t+1}; \phi^-) - V(s_t; \phi) \tag{3.5}$$

Finally, the TD error δ_θ is calculated using the V-value $V(s_{t+1}; \phi^-)$ instead of a future Q-value.

$$\delta_\theta = r_t + \gamma V(s_{t+1}; \phi^-) - Q(s_t, a_t; \theta) \tag{3.6}$$

The target model θ^- is not used for this type of agent, so the parameters needed are θ, ϕ and ϕ^-.

While training two separate networks is more computationally expensive, the function $V(s)$ converges faster than $Q(s, a)$, resulting in an overall improvement (Sabatelli et al., 2018; Sabatelli, Louppe, Geurts, and Wiering, 2020).

3.2 Environments

Experiments are carried out in two environments, `CartPole-v0` and `Acrobot-v1`, from the OpenAI gym (Brockman et al., 2016). A modified version of each environment is created, `CartPole-mod` and `Acrobot-mod`, so that one can be used as the source MDP $\mathcal{M}_S$ while the other is used as target MDP $\mathcal{M}_T$. There is no transfer between `CartPole-v0` and `Acrobot-v1` as their state representations and action spaces differ. In both cases, transfer is conducted from default to modified environment and vice versa, resulting in four source-target pairs.

3.2.1 CartPole

In the `CartPole` environment a pole is balanced on a cart that moves on a horizontal axis. If the pole tips over too far or the cart moves out of bounds the episode ends. An episode also ends if the maximum number of time steps is reached, which is 200 for `CartPole-v0`. Each time step gives a reward of +1, so the cumulative reward (or score) is in the range $(0, 200]$. A successful agent will get an average score of ≥ 195 over the course of 100 episodes. States are represented as a four-tuple containing position of the cart, velocity of the cart, angle of the pole and the pole's velocity at the tip. Available actions are pushing the cart left or right.

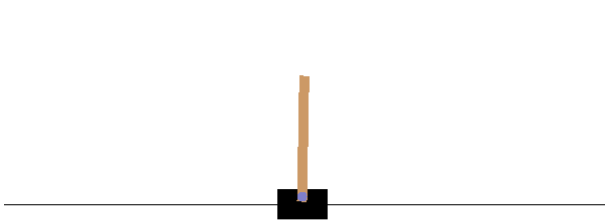
3.2.2 Acrobot

`Acrobot` consists of two joints and two links. The first joint is fixed in a central location and attaches to the inner link. The second joint is actuated and connects the inner and outer link. To reach the goal the agent has to swing the tip of the outer link above a threshold height by applying torque to the actuated joint between the links. A reward of -1 is given per time step and the maximum number of time steps is 500, so scores are in the range $[-500, 0)$. Counting as successful is an agent that scores an average of ≥ -100 over 100 episodes. States are six-tuples containing cosine, sine and angular velocity of each joint's angle. The three actions from which an agent can pick are applying torque in either direction or simply applying no torque.

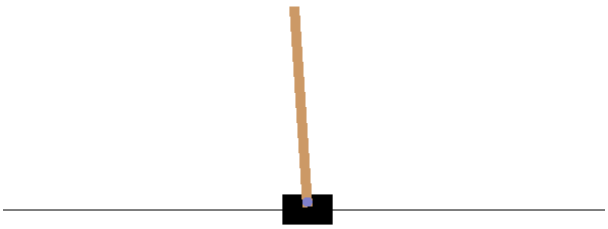
3.3 Experiment Setup

3.3.1 Hardware

The machine used for the experiments uses an AMD Ryzen 3700x CPU, Nvidia RTX 2060 Super (8 GB) GPU and 16 GB of RAM (2666 MHz). All computations were done by CPU as the small networks used do not benefit from parallel processing.



(a) CartPole-v0



(b) CartPole-mod

(c) Acrobot-v1

(d) Acrobot-mod

Figure 3.1: Gym Environments

3.3.2 Preparation

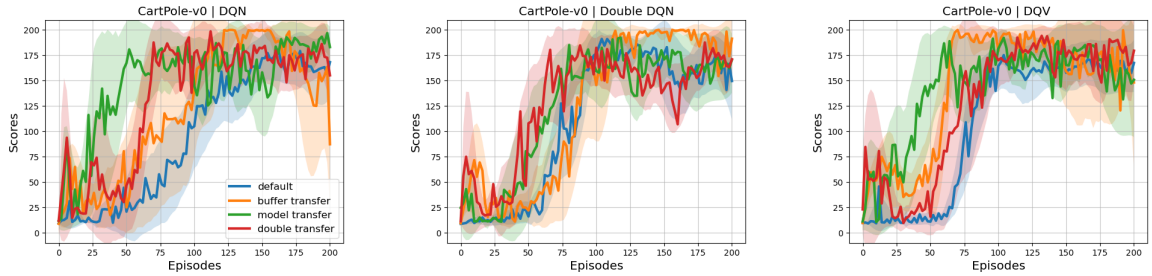
First, a modified version of each environment is created, `CartPole-mod` and `Acrobot-mod` respectively.

`CartPole-mod` uses a pole length of 0.8 instead of the default 0.5. For `Acrobot-mod` the length of both links is modified: The inner link changes from 1.0 to 0.8 and the outer link changes from 1.0 to 1.2.

An untrained agent of each type is trained in each of the four tasks until it is successful according to the aforementioned criteria and its model parameters are saved. Next, the trained agents collect transitions in the same environments ($\mathcal{M}_s$), using the learned Q-policy, and save the ER buffer.

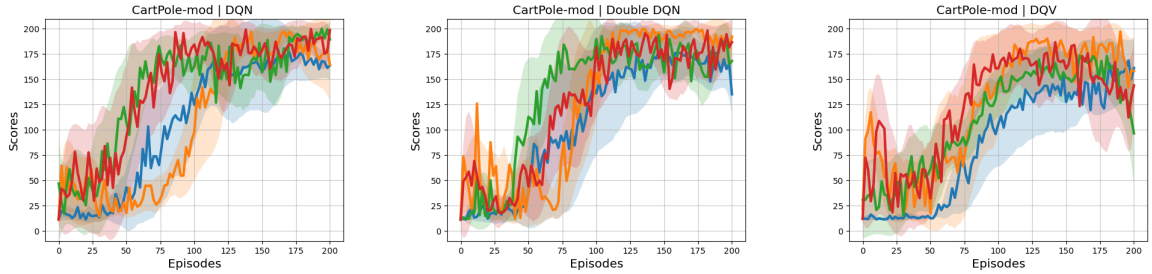
3.3.3 Training procedure

With everything set up, four variants of each agent in each environment ($\mathcal{M}_t$) are trained. The first agent does not make use of TL and simply serves as a baseline. One agent receives a full buffer from $\mathcal{M}_s$ (buffer transfer), a second agent uses the model parameters θ of an agent trained in $\mathcal{M}_s$ (model transfer) and the third agents uses both forms of transfer (double transfer). In the case of buffer and double transfer the last layer (i.e. the head) of the model is reinitialized. Scores are recorded in between training steps (every two steps for `CartPole` and every 20 steps for `Acrobot`) and use the learned policy $Q^\pi(s_t, a_t)$ instead of the ε-greedy policy used for training. The procedure is repeated five times for each agent to generate an average.



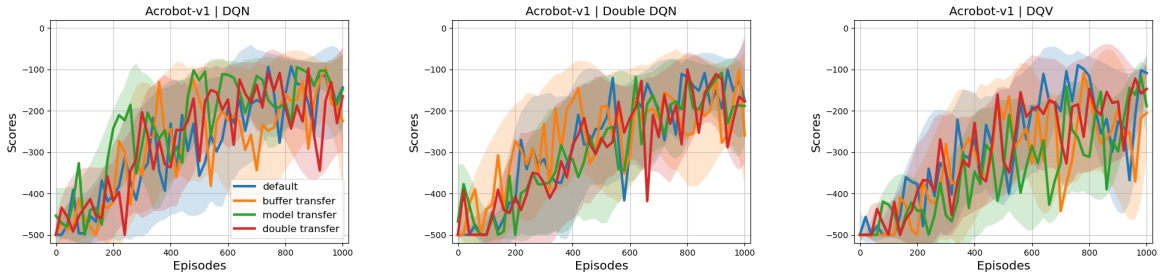
(a) DQN **(b) Double DQN** **(c) DQV**

Figure 3.2: Results CartPole ($\mathcal{M}_s$ = mod, $\mathcal{M}_t$ = v0)



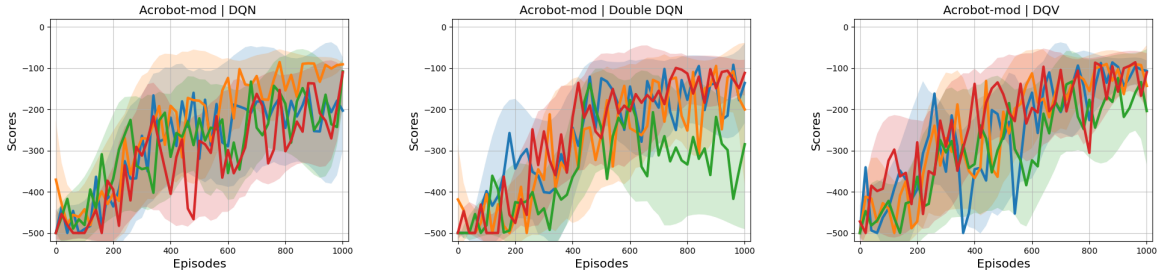
(a) DQN **(b) Double DQN** **(c) DQV**

Figure 3.3: Results CartPole ($\mathcal{M}_s$ = v0, $\mathcal{M}_t$ = mod)



(a) DQN **(b) Double DQN** **(c) DQV**

Figure 3.4: Results Acrobot ($\mathcal{M}_s$ = mod, $\mathcal{M}_t$ = v1)



(a) DQN **(b) Double DQN** **(c) DQV**

Figure 3.5: Results Acrobot ($\mathcal{M}_s$ = v1, $\mathcal{M}_t$ = mod)

Table 3.1: Area Ratio Scores CartPole

Transfer	DQN		Double DQN		DQV	
	mod → v0	**v0 → mod**	**mod → v0**	**v0 → mod**	**mod → v0**	**v0 → mod**
Buffer	0.21	0	0.10	0.13	0.26	0.30
Parameters	0.36	0.23	0.11	0.19	0.26	0.22
Double	0.28	0.24	0.13	0.12	0.17	0.29

Table 3.2: Area Ratio Scores Acrobot

Transfer	DQN		Double DQN		DQV	
	mod → v0	**v0 → mod**	**mod → v0**	**v0 → mod**	**mod → v0**	**v0 → mod**
Buffer	0	0.17	0.04	-0.14	-0.19	0.04
Parameters	0.19	-0.10	-0.08	-0.58	-0.37	-0.17
Double	0.05	-0.21	-0.08	0.01	-0.04	0.11

3.4 Analysis

Scores (i.e. cumulative reward) are plotted, one plot per $\langle \mathcal{M}_s, \mathcal{M}_t, \text{agent} \rangle$ triple, resulting in 12 plots. These are analyzed according to the criteria outlined in Lazaric (2012). Visually, the criteria are (1) learning speed, (2) asymptotic and (3) jumpstart improvement. To quantify the success of transfer, the area ratio (AR) scores are calculated.

$$\mathbf{R} = \frac{\text{area with transfer} - \text{area without transfer}}{\text{area without transfer}} \tag{3.7}$$

4 Results

The learning curves for all experiments can be seen in Fig. 3.2 to 3.5. Curves represent an average over five training cycles and the shaded areas show the standard deviation. Colors are blue for a default agent with no transfer, orange for buffer transfer, green for model transfer and red for double transfer. Area ratio scores corresponding to the same plots are found in tables 3.1 and 3.2.

4.1 CartPole

`CartPole` experiments show learning speed improvement for buffer transfer in three cases (Fig. 3.2a, 3.2c, 3.3c) and impairment in one case (Fig. 3.3a). Asymptotic imporovement can be observed to some extent in all cases while jumpstart improvement is absent in all cases.

Parameter transfer leads to learning speed improvement in all `CartPole` experiments and the same is true for double transfer. In all cases except for Fig. 3.2b there is some asymptotic improvement though not very distinctive except for Fig. 3.3c. Again, jumpstart improvement is absent in all experiments. Area ratio scores are positive for all transfer variants in all experiments, so TL for all `CartPole` experiments was successful. An interesting observation is that the addition of ER buffer transfer to parameter transfer does not seem to have a significant effect when compared to only parameter transfer.

4.2 Acrobot

For `Acrobot` experiments the situation is very different. Training curves are overlapping for the most part and do not show much of an effect of either transfer method. Exceptions are asymptotic improvement with ER buffer transfer seen in Fig. 3.5a and sudden deterioration with parameter transfer as seen in Fig. 3.5b. The area ratio scores in table 3.2 confirm these findings as they are not conclusive but show a trend towards negative transfer.

5 Discussion

5.1 Findings

For two environments from the OpenAI gym (Brockman et al., 2016), `CartPole-v0` and `Acrobot-v1`, modified environments were created to study TL between two similar tasks. Experiments were run with three agents, DQN (Mnih et al., 2013), Double DQN (Van Hasselt et al., 2016) and DQV (Sabatelli et al., 2018). Three types of knowledge transfer were studied, (1) ER buffer transfer, (2) model transfer and (3) double transfeer (ER buffer + model), by comparing their learning curves visually and numerically.

Results of the `CartPole` experiments are distinctly positive for all types of transfer, showing learning speed improvements as well as convergence to a higher cumulative reward. Area ratio scores comparing the area under the curve between an agent trained from scratch and a transfer agent are all positive, further substantiating these findings.

In the case of the `Acrobot` environment results are less impressive and not indicative of any particular trend. Visually, only two scenarios show a clear difference in learning success, one of them positive and one negative. Area ratio scores show mixed results but tend towards negative impact of knowledge transfer on learning performance.

Overall the results are not conclusive as they differ enormously between tasks. The findings in the `Acrobot` environment are in line with those of Sabatelli and Geurts (2021) which stated no positive knowledge transfer between tasks. However, the results produced in the `CartPole` environment are pointing towards a different conclusion as both types of transfer, model and ER buffer, had a positive impact.

A notable difference between this research and that of Sabatelli and Geurts (2021) is in the usage of state features. Both environments provide numerical values representing perfect sensor data. On top of that, the features are identical between source and target task. In the case of ALE environments, CNNs have to learn to extract useful features from a series of greyscale images and these features also differ between tasks. Due to hardware and time constraints it was not possible to gather meaningful data for such tasks. A first attempt of transferring knowledge between the ALE environments `PongDeterministic-v4` and `EnduroDeterministic-v0` is shown in Figures 5.1 and 5.2.

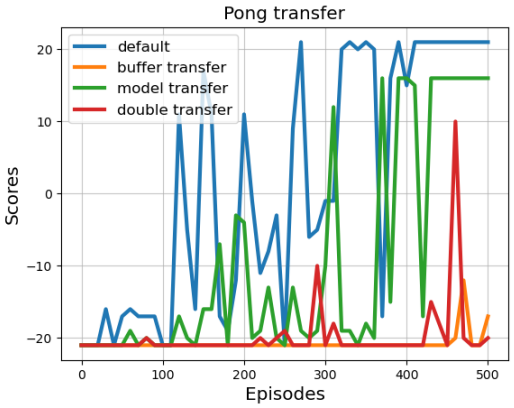


Figure 5.1: Transfer Enduro → Pong

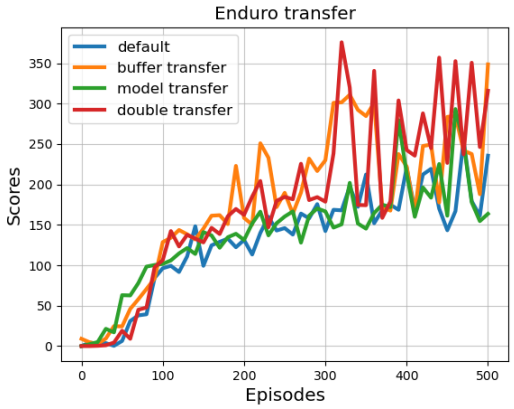


Figure 5.2: Transfer Pong → Enduro

Transferring ER from `Enduro` to `Pong` lead to the agent being unable to learn, but a transfer of ER from `Pong` to `Enduro` seems to work. A possible explanation is that the (visual) features of `Pong` are less complex than those of `Enduro`, which would be in line with the hypothesis of Farebrother, Machado, and Bowling (2018) (as cited in Sabatelli and Geurts (2021)), who suggest that transfer from an easier to a more difficult task are more likely to result in positive transfer.

5.2 DQV2

DQV2 is an agent concept that was developed as part of this research and is based on the DQV architecture. The motivation was to reduce the number of model parameters to decrease the amount of required memory and computational cost. Underlying is the assumption that both the value function $V(s)$ and the Q-value function $Q(s, a)$ learn from similar information extracted from states by an ANN. If this assumption holds, then a trained Q-value model should provide enough information to a V-value model, so that only the last layer has to be replaced and trained. The general idea is outlined below

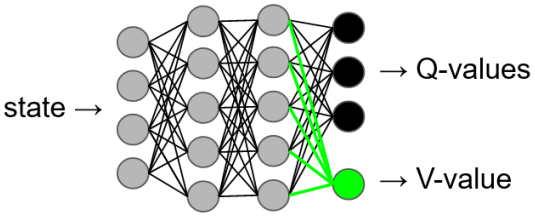


Figure 5.3: DQV2 network architecture

with black connections showing weights adjusted with the loss function L_θ of the Q-value model and green connections showing weights trained with the V-model's loss functions L_ϕ.

Results for this agent can be found below in Figures 5.4 and 5.5.

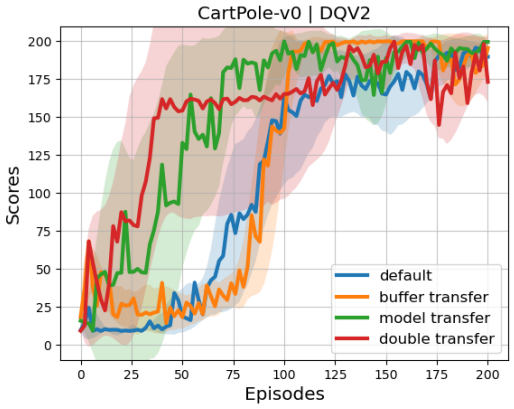


Figure 5.4: CartPole-v0 (DQV2)

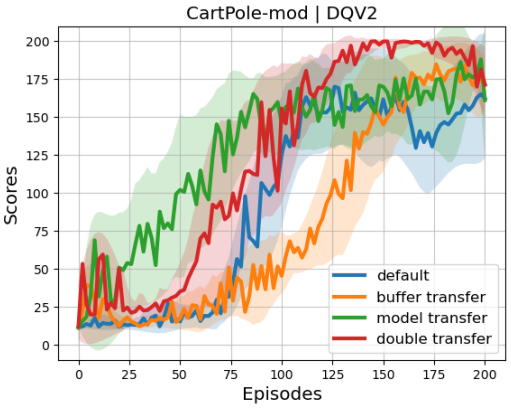


Figure 5.5: CartPole-mod (DQV2)

Clearly, the agent is able to learn the `CartPole` tasks. However, it is not obvious how it compares to other types of agents, especially DQV. It is also unknown if the agent is able to learn other, more challenging tasks such as ALE games. Therefore, a comparison of DQV and DQV2 across various tasks is required to investigate its usefulness.

6 Conclusion

In this work we investigated the effectiveness of transferring ER in model-free DRL settings. Agents were trained in a source task, collected ER data and transferred it to an agent learning a slightly modified version of the task. Learning performance was compared between regular agents without TL and three types of TL: (1) ER buffer transfer, (2) model transfer and (3) double transfeer (ER buffer + model). All types of TL were succesful for the first type of task, while for the second type the results were mixed. Since the source and target tasks were very simple and almost identical, it has to be shown how ER transfer performs in tasks with fewer similarities. A logical next step would be a series of similar experiments using different ALE environments. Another limitation is that only full ER buffers were transferred. It seems possible that too much ER data from a source task is detrimental to learning performance, so comparing transfer of different amounts of ER data seems worthwhile.

References

Ibraheem Alhashim and Peter Wonka. High quality monocular depth estimation via transfer learning. *CoRR*, abs/1812.11941, 2018. URL http://arxiv.org/abs/1812.11941.

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, jun 2013.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Jesse Farebrother, Marlos C. Machado, and Michael Bowling. Generalization and regularization in DQN. *CoRR*, abs/1810.00123, 2018. URL http://arxiv.org/abs/1810.00123.

Kasthurirangan Gopalakrishnan, Siddhartha K. Khaitan, Alok Choudhary, and Ankit Agrawal. Deep convolutional neural networks with transfer learning for computer vision-based data-driven pavement distress detection. *Construction and Building Materials*, 157: 322–330, 2017. ISSN 0950-0618. doi: https://doi.org/10.1016/j.conbuildmat.2017.09.110. URL https://www.sciencedirect.com/science/article/pii/S0950061817319335.

Alessandro Lazaric. Transfer in Reinforcement Learning: a Framework and a Survey. In Martijn van Otterlo Marco Wiering, editor, *Reinforcement Learning - State of the art*, volume 12, pages 143–173. Springer, 2012. doi: 10.1007/978-3-642-27645-3_5. URL https://hal.inria.fr/hal-00772626.

Long-Ji Lin. Reinforcement learning for robots using neural networks, 1993.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*, 2015. URL https://arxiv.org/abs/1511.06342.

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

Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. CNN features off-the-shelf: an astounding baseline for recognition. *CoRR*, abs/1403.6382, 2014. URL http://arxiv.org/abs/1403.6382.

Sebastian Ruder, Matthew E. Peters, Swabha Swayamdipta, and Thomas Wolf. Transfer learning in natural language processing. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Tutorials*, pages 15–18, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-5004. URL https://aclanthology.org/N19-5004.

Matthia Sabatelli and Pierre Geurts. On the transferability of deep-q networks. *CoRR*, abs/2110.02639, 2021. URL https://arxiv.org/abs/2110.02639.

Matthia Sabatelli, Gilles Louppe, Pierre Geurts, and Marco A Wiering. Deep quality-value (dqv) learning. *arXiv preprint arXiv:1810.00368*, 2018. URL https://arxiv.org/abs/1810.00368.

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

Remo Sasso, Matthia Sabatelli, and Marco A. Wiering. Fractional transfer learning for deep model-based reinforcement learning. *CoRR*, abs/2108.06526, 2021. URL https://arxiv.org/abs/2108.06526.

Amirhosein Shantia, Eric Begue, and Marco Wiering. Connectionist reinforcement learning for intelligent unit micro management in starcraft. In *The 2011 International Joint Conference on Neural Networks*, pages 1794–1801, 2011. doi: 10.1109/IJCNN.2011.6033442.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning I: Introduction*. The MIT Press, 1998.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

Oriol Vinyals, Igor Babuschkin, Wojciech Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John Agapiou, Max Jaderberg, and David Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575, 11 2019. doi: 10.1038/s41586-019-1724-z.

Christopher Watkins. Learning from delayed rewards. 01 1989.

Marco A Wiering. QV(λ)-learning: A new on-policy reinforcement learning algrithm. In *Proceedings of the 7th european workshop on reinforcement learning*, pages 17–18. Citeseer, 2005.

Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.