Outperforming the Baseline: Transfer Learning in Atari via Parallelized Q-Networks

Bachelor's Project Thesis

Andjela Matic, s5248736, a.matic.1@student.rug.nl,
Supervisor: Rafael Fernandes Cunha, r.f.cunha@rug.nl

Abstract: Transfer Learning (TL) is an emerging technique present in reinforcement learning (RL), showing great potential and contributions to decreasing computational costs, improving efficiency, and maximizing generalization. This study looks at TL in the scope of the Parallelized Q-Network (PQN) algorithm, which has demonstrated vast improvements in cost-effectiveness and overall performance compared to peer algorithms, such as Rainbow, IQL, PPO, etc. Building on previously conducted research in the MinAtar environment, this paper focuses on applying and evaluating TL in the Atari environment. Due to its favourable dynamics and structure, the Atari environment addresses limitations of previous research and presents a suitable extension. The results from multiple configurations show significant improvements in TL simulations compared to the baseline. In the example of the *Pong-Breakout* pair, training is noticeably accelerated, with convergence being reached more than twice as fast compared to the baseline. These findings illustrate the potential of TL in complex environments, especially when the source and target environments share underlying visual and mechanical features.

1 Introduction

In recent years, reinforcement learning (RL) has become one of the leading areas of research in machine learning, showing significant improvement in domains such as game playing [14], autonomous systems, and robotics [20]. With the growing complexity of the tasks, researchers have found ways to maximize efficiency while simultaneously improving performance through methods such as algorithmic enhancement [10], and environment parallelization [9].

A noted approach utilizes the power of the GPU as leverage to perform computationally expensive simulations. This technique allows extensive use of parallelization, where numerous environments operate simultaneously to accelerate the learning process. Such implications are suited for complex environments, often requiring vast resources and time.

As the basis for this paper, we take the Parallelized Q-Network (PQN) algorithm, as introduced by Gallici et al.. It is an example of a GPU-based method, which utilizes astute workarounds to enable fast runtime while maintaining, or even improving on, previously achieved performance. Namely, the paper eliminates the need for a replay buffer and a target network, both data-intensive structures requiring ample memory to function. PQN addresses these by implementing vectorized parallelization and regularization, respectively. The results have shown scalable, consistent performance, indicating a promising future for GPU-accelerated algorithms.

Another explored method aiming to improve RL benchmarks is transfer learning (TL). The core idea is exploiting previously gathered knowledge to bootstrap performance in an unseen environment. It is presented as a solution to real-life machine learning scenarios, where sparse and difficult-to-obtain data is frequent; training on readily available or simpler domains can reduce costs. Therefore, transfer learning puts forward an optimistic outlook [26]. However, it also comes with many drawbacks. Difficulties such as negative transfer [26], discrepancies and conflicts between domains, as well as determining and filtering

which information to transfer [22], are only a few issues that can arise from this method. On the other hand, TL has shown auspicious results when combined with more refined methods, such as meta-learning [8], embedding transfer and fine-tuning [17], and transfer with successor features [2], among others. In this paper, we will look at direct parameter transfer - training an agent on a source game, and reusing the complete set of learned parameters of the Q-network in a target game.

This experiment closely follows a previously conducted experiment as part of a Bachelor's Thesis carried out by Zaharia. Their project focused on the MinAtar environment [27] - a simplified, miniaturized version of the Atari environment. The MinAtar environment consists of five games - *Seaquest, Asterix, Breakout, Freeway*, and *Space Invaders*, confined to a 10x10 grid. An additional channel is included to form a state representation of 10x10n, n representing a game-specific object.

The paper carried out the experiment by comparing the transfer learning configurations with the baseline. *Seaquest* is excluded from this study, as the other four were present in the original paper [9], making them more easily comparable. The transfer occurs between each environment and additionally introduces **layer reinitialization**. This is a toggleable feature, where either/both the input and output layers are reset before loading the Q-network parameters. This results in a total of 13 learning curves (plus an additional four for the baseline simulations).

The results of the paper do not show improvement in learning time compared to the baseline. This was mostly attributed to PQN's innate architecture, which is not specifically designed to handle transfer, as its implementation lacks structures that support filtering of advantageous features. Moreover, it does not employ any safeguards to combat negative transfer. A further cause of insignificant results is the *MinAtar* environment itself. Due to its lowered visual complexity, the usability of relationships between states and actions plummets. In addition, *MinAtar*'s lack of stochasticity eliminates the need for abstraction in training; therefore, certain generalizations might be redundant and are omitted in the network.

However, the *Atari* environment addresses the suggested limitations, making it a suitable candidate for further research. Most notably, its much more refined observation space could encourage the agent to form knowledge about hierarchical policies and abstract representations, which could, in turn, be useful as transfer material.

The results of this research provide insights into the usability of TL in complex RL environments, by analyzing the effectiveness of transfer between various configurations of *Atari* games. The research question it aims to answer is: "*What is the effect of transferring pretrained weights on the agent's performance using the PQN algorithm in the Atari environment?*", where performance is defined as the time it takes for the return to converge (the learning speed), where a shorter time entails a better-performing agent. Additionally, performance is evaluated through the direct comparison of the return values at predetermined milestones, in the case when convergence is not reached. A consistent improvement in return values during early to mid training indicates a better-performing agent. The transfer occurs between two different *Atari* games, where the network weights obtained after training in the source game are directly used from the start of training in the target game.

In this paper, I empirically show the following:

- I demonstrate that TL configurations are able to match or significantly outperform the baseline, underlining the potential of TL in complex, but similar domains, such as that of *Atari* games.
- I discuss the potential correlation between an environment's visual representation and its ability to transfer effectively.
- I list possible limitations and challenges, and suggest methods to overcome them.

2 Background

2.1 Reinforcement Learning

Reinforcement learning (RL) is a machine-learning principle in which an agent interacts with the environment, receives feedback, and adjusts its behaviour accordingly. The agent freely explores the environment, intending to develop the optimal

policy - a mapping between states and actions, which maximizes the sum of rewards the agent receives.

The key framework to formally describe RL is the **Markov Decision Process** (MDP) - an intuitive and fundamental formalism for interactions between the agent and the environment. MDPs have become standard for modelling planning problems, game-playing problems, robot controls, etc [23]. An MDP is represented by a collection of objects $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$ [19], where:

- $\mathcal{S}$: the set of possible states of the system, representing the finite lot of environment positions the agent can find itself in.
- $\mathcal{A}$: the set of allowable actions that can be performed by the agent.
- $P(s'|s, a)$: the transition probability function (probability of the system being in state s', if action a was chosen from state s).
- $R(s, a)$: the reward received if the current state is s, and action a is selected.
- γ: the discount factor in the range [0,1), which indicates how valuable future rewards are for the agent at the current time. A discount factor closer to 0 will prioritize short-term rewards. Meanwhile, γ closer to 1 will "play the long game" and incorporate potential future rewards into its decision-making process.

The goal of reinforcement learning lies in maximizing the return, a.k.a. the cumulative discounted reward. The return is calculated by predicting and summing the rewards (scaled according to the discount factor) that the agent will receive. It is formulaically defined as [19]:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \ + R_T$$

where T represents the final time step. This corresponds to those tasks where the notion of a final time step is viable, such as episodic tasks that have a *terminal state*. In the case of infinite horizon tasks (there is no natural termination to the agent-environment interaction), it can be defined as:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

In order to achieve maximal return, the agent explores the environment by, e.g., taking random actions until it eventually gathers enough information to form a well-performing policy. A policy determines which action to take in a given state to maximize return. The goal is to uncover the **optimal policy** $\pi*$ - a mapping between states and actions that always maximizes the expected cumulative sum of discounted reward in all possible states.

During training, the agent can also exploit the information it has gathered thus far to speed up the process. This is where the exploration/exploitation balancing act comes in. To maximize training speed and performance, different strategies can be applied to find the optimal combination of exploration and exploitation.

2.2 Q-Learning

Q-learning is one of the more well-known reinforcement learning algorithms. It is a model-free, off-policy method that aims to provide agents with the capability to act optimally in Markovian domains. [25]
Q-learning relies on Q-values - the expected discounted sum of rewards for executing an action in a given state according to a policy. [25] Q-values are unique to state-action pairs, i.e., the agent can decide which action to take in a given state that maximizes the discounted sum of rewards. We can also define the action-value function or the Q-function, which evaluates specific actions by calculating the expected return from state s when choosing an action a, and subsequently following the policy π:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$$

The Q-values an agent gathers throughout the training process are stored in a Q-table, where each entry is the Q-value of a state-action pair. The values of the Q-table are updated at each step, following the Bellman equation:

$$Q_{new}(S_t, A_t) \leftarrow (1 - \alpha) \cdot Q(S_t, A_t)$$

$$+\alpha \cdot (R_{t+1} + \gamma \cdot \max_a Q(S_{t+1}, a))$$

where:

- α is the learning rate [0,1).
- γ is the discount factor [0,1).

- $Q(S_t, A_t)$ is the current Q-value of the state-action pair at time t.
- R_{t+1} is the reward to obtain if action A_t is chosen in state S_t.
- $\max_a Q(S_{t+1}, a)$ is the maximum expected discounted return that can be obtained from state S_{t+1}.

Q-learning is an off-policy method, which denotes that the agent's behaviour does not reflect its current learned policy. In Q-learning, the agent uses the maximal discounted sum of rewards of a state-action pair to determine its Q-value, but the action associated with the highest return might not be chosen. This method ensures exploration while maintaining a somewhat accurate representation of the preferability of states and actions.

2.3 Deep Q-Learning (DQN)

Deep Q-Learning (DQN) introduces deep machine learning techniques to the Q-Learning algorithm. Namely, the Q-value function from Q-Learning is replaced by a neural network, the Q-network. The Q-network serves as a function; however, the inner workings correspond to those of a neural network, including multiple layers containing neurons. This structure allows for more nuanced learning and pattern recognition, which is more suitable for highly stochastic environments.
DQN uses additional structures that enhance the algorithm's stability and effectiveness: the **replay buffer** and the **target network**.

A replay buffer is a data structure that stores the agent's experiences in the forms of tuples (s, a, r, s', d) where s is the current state, a is the chosen action, r is the received reward, s' is the next state, and d is the "done" flag -indicates whether the episode has terminated at this step. The state, action, and reward components align with the elements of the Markov Decision Process (see Section 2.1). To enhance an agent's performance, we can use its gathered experiences to bootstrap estimates for unexplored actions. This allows experiences to be used more efficiently, as, otherwise, they would be discarded after each update. However, the agent's interaction with the environment occurs sequentially. This poses a challenge - the collected data of the events are highly correlated, as they are not independent and identically distributed (IID). This temporal correlation can harm the bootstrapped results and the agent's performance by introducing instability and divergence in learning. To avoid this issue, the replay buffer allows random sampling within stored experiences. Moreover, a sampled batch can improve stability in training by reducing variance and preventing drastic oscillations in policy updates.

The target network addresses problems of instability in training. Since updates to the Q-network are made at each step, the network is simultaneously used for training and estimation of Q-values. This creates a moving target for the Q-values and contributes to the divergence. A target network is created as a copy of the Q-network, acting as a baseline to the online network. It is updated less frequently, which stabilizes estimations.

2.4 Parallelized Q-Learning (PQN)

Parallelized Q-Learning, as introduced by Gallici et al. [9], is a deep reinforcement learning algorithm that directly responds to issues we see with deep reinforcement learning methods. The algorithm aims to utilize parallelization to overcome issues of instability and computational expenses. It realizes this through incorporating vectorized environments that can be run simultaneously, enabling multiple agents to explore the environment independently, as shown in Figure 2.1. This, in turn, creates many data samples used among the agents in parallel environments, which eliminates the need for a replay buffer.

Moreover, PQN can either use LayerNorm or BatchNorm - two forms of regularization that address issues concerning off-policy instability and divergence. As a result, the algorithm eliminates the use of a target network.

The main advantage we notice from these changes is that the algorithm does not use memory-intensive structures (such as the replay buffer and target network), allowing it to fully operate on a GPU. This decreases computational expenses drastically compared to other deep reinforcement learning algorithms, such as DQN.

PQN has shown favourable results when run on multiple benchmarks, such as the Baird's Counterexample [19], Craftax [13], Atari [12], and

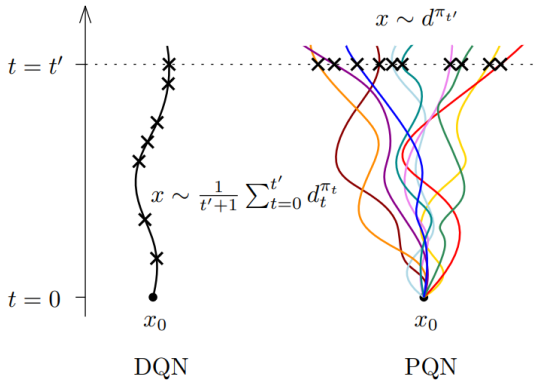


Figure 2.1: Graphical representation of the PQN algorithm, demonstrating its parallelization approach. [9]

MinAtar [27]. Namely, in certain cases, it has shown convergence in fewer steps, faster simulations (time-wise), and even higher average scores compared to high-ranking peers.

2.5 Transfer Learning in Reinforcement Learning

Transfer learning (TL) is one of the many up-and-coming techniques in reinforcement learning. It entails the use of previously gained knowledge in a new environmental setup to improve early performance. The core idea is that the agent can effectively utilize its training by generalizing to unseen environments or tasks. The agent can, therefore, identify the basic mechanics, underlying patterns, etc., and use this knowledge to overcome the initial performance gap shown through the learning curve. In other words, convergence should be reached faster, which can be beneficial as it is both less computationally expensive and time-consuming. [29]

2.6 Atari

The Arcade Learning Environment (ALE) [5] is an RL environment that adapts the original *Atari* 2600 games to the machine learning field. It features a variety of games, ranging from 1v1 combats to card-playing games, etc. The diversity in these games leverages the perfect platform to host artificial intelligence (AI) agent evaluations - it provides a multitude of tasks requiring general competence.[12]
The *Atari* environments share the same observation space - a screen consisting of three channels of dimensions 210, 160, and 3 representing the height, width, and colours, respectively. There are a total of 40 possible actions available in the ALE space, with one game being limited, usually in the range of 3 to 18. *Atari* games are all episodic, i.e., they have a definitive terminal state, and deterministic, e.g., taking the same action in the same state results in the same outcome.

This environment was chosen as its inherent structure could benefit TL. Due to its consistent observation space size, and its complex visual representations, agents can form general knowledge from one game that can easily be applied to neighboring games. Additionally, the *Atari* environment was present as one of the benchmarks in the original paper [9], which allows results to be directly compared.

3 Methods

3.1 Objective

This research aims to investigate the effects of transfer learning in *Atari*. Namely, the transfer occurs between two different environments, which have a similar subjective visual appearance. This way, we can test if the transfer between two similar environments will result in faster convergence. This would result in decreased effort needed to achieve convergence, while reducing the computational cost.

3.2 Environments

The family of environments in question (*Atari*) is one of the listed benchmarks as presented in [9]. For this experiment, out of the approximately 100 featured games, four were chosen, namely: *Pong, Breakout, Space Invaders, and Demon Attack*. Moreover, these environments are paired in groups of two, in their respective order.

These environment pairs were chosen due to the similarity of their appearance and overall gameplay. The criterion according to which this is evaluated is purely subjective and only serves as an additional point of investigation. The transfer will occur bidi-

rectionally - both members of a pair will receive a transfer from the other.

Pong and *Breakout* both share a similar structure of a round object bouncing from a flat rectangle. *Space Invaders* and *Demon Attack* feature a player avatar at the bottom of the screen, able to shoot at enemies coming from above. Additionally, they possess the same action size (6), which could produce improved results, as more of the Q-network can be retained. These environments were chosen arbitrarily as opposed to other environments with comparable descriptions.

As an additional point of investigation, a fifth environment was included in the experiment. The environment in question, *Wizard of Wor*, serves as an outlier due to its dissimilar appearance and gameplay compared to the aforementioned four main environments. By performing a transfer onto *Wizard of Wor*, we can get a clearer picture of whether the physical features of the environment affect transfer learning.

3.3 Experimental Metrics

There are numerous ways to assess the performance and benefits of a transfer learning agent. Some noted metrics include [21]:

- *Jumpstart* - the TL agent's initial performance in the target after transfer from the source task.
- *Asymptotic Performance* - the final learned performance achieved by the TL agent.
- *Total Reward* - the total reward obtained by the TL agent. Alternatively, this can be represented as the Area Under the Curve (AUC).
- *Transfer Ratio* - the ratio of the total reward obtained by the TL agent compared to the source agent.
- *Time to Threshold* - the learning time required by the TL agent to achieve a predetermined performance checkpoint.

The main point of interest of this research is the time it takes for convergence to take place, which is defined as the point when the return during training plateaus. This functions as an indicator of the agent's successful adaptation to the environment, as it can greedily achieve a consistent return. The final return is reported and compared to the baseline, together with the plotted graphic representation containing the learning curve which allows us to identify the point of convergence. This falls under the *Asymptotic Performance* methodology, shown in section A, tables A.1, A.2, and A.3.

As an additional metric used to evaluate the agents' performances, we introduce checkpoints at the 2, 5, and 7 million environment step marks. The performance metric chosen to be compared is the return. If we notice a consistently higher return at the given checkpoints from the transfer agent, we can deduce it performs better than the baseline. This serves as a quantification of the learning speed directly tied to the AUC, which is especially beneficial when convergence is not reached (which is the case in the majority of the simulations in this paper), since we have not obtained convergent results. The recorded metric can be found in section A, tables A.4, and A.5.

These metrics were chosen because they encapsulate the notion of comparing results either at full convergence or at predetermined baselines, which helps avoid misleading comparisons during early training [3][4].

3.4 Configurations

As per the previous section, the following configurations are performed:

- **Pong - Breakout**: *Pong* receives the Q-network from *Breakout*.
- **Breakout - Pong**: *Breakout* receives the Q-network from *Pong*.
- **Space Invaders - Demon Attack**: *Space Invaders* receives the Q-network from *Demon Attack*.
- **Demon Attack - Space Invaders**: *Demon Attack* receives the Q-network from *Space Invaders*.
- **Wizard of Wor - Pong** - *Wizard of Wor* receives the Q-network from *Pong*.

3.5 Hardware Specifications

The experiments were conducted on a personal Windows device using WSL to emulate Ubuntu, equipped with an NVIDIA GeForce RTX 3080 GPU and an AMD Ryzen 9 6900HS CPU. This setup was readily available and is slightly higher-end compared to that of a standard user. However, it is significantly downgraded from the hardware used by the researchers in [9], resulting in lengthier simulations and increased computing costs.

3.6 Reproducibility

In order to ensure reproducibility, the used code is freely available in the GitHub repository. The configuration settings are kept constant for every simulation, except for the seed, which is varied. The full configuration is available in the form of a `config.yaml` file. This is to create more randomized results, which ensures the results are as accurate as possible. The code can be run without any additional requirements to the original PQN *Atari* script. Additionally, library versions used when running the code are also available on GitHub, as finding the correct configuration can be time-consuming.

3.7 Method Evaluation

To ensure that transfer learning between environments occurs as expected, we test the TL code by saving a model and its parameters and loading it into a fresh simulation of an identical environment. We expect the results of this test to be as follows: recorded return starts off directly from where the baseline simulation has terminated.

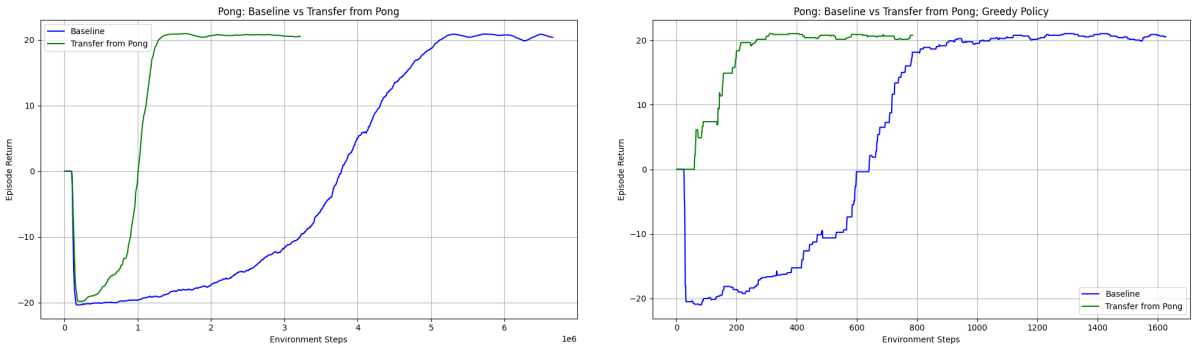
In Figure 3.1, we can see a test run where the two compared curves are the baseline *Pong* run and a transfer of parameters from the said run to another instance of *Pong*. *

The behaviour policy in Figure 3.1a shows a dip at the start of the simulation, reminiscent of the ϵ parameter being restarted to 1 (Figure 3.2a), after which the learning curve skyrockets to the previously convergent return value. This causes the new environment to explore extensively, instead of exploiting the already learned values.

As for the target policy, the expected curve after saving and loading the model on the same environment would be one starting at the previously recorded final return. In the case of a converged simulation, we expect the newly initialized simulation to maintain the plateaued curve. However, as shown in Figure 3.1b, the beginning of the simulation starts at 0 return. After further investigation, this was traced back to how the return is calculated. Since multiple vectorized environments are ran in parallel, the return at any timestep is computed as a mean across them. Intuitively, each environment takes a certain number of steps to complete a full episode, wherein the initial mean return will be null, as no environment has reached the end of the episode. As more and more environments complete it, the return gradually increases. To support this claim, we ran a testing simulation with only one testing environment, set in the `config.yaml` file through the `TEST_ENVS` variable. Figure 3.2b shows the results of this experiment. We can see that the return initially starts from 0; however, after around 60 steps (per environment steps; scale shows values for the total 128 environments corresponding to the training environment), we notice a sudden spike in return, reaching the previously converged value. This confirms the claim that the return is calculated as a mean across all the environments. Moreover, it can be argued that the exact mechanics behind the calculations are irrelevant to the experiment. With each trial run, we notice similar trends - the return is close to 0 up until a certain step mark, after which it abruptly increases to approximately the convergent value. Since these observations are consistent across multiple runs, it can be ruled off as a bug or coincidence, which allows us to directly compare the results.

Despite the initial dips in return, after discussing possible causes, we can confirm the successful loading of network parameters due to the noticeable spike in return, which leads to almost instantaneous convergence. We have, therefore, ensured reliability and eliminated risks of potential errors in further experiments.

*To reduce computational costs and save time, both simulations were cut short of their full runtime, as convergence was evident in both before they reached the step limit.



(a) Behaviour Policy **(b) Target Policy**

Figure 3.1: Behaviour (a) and Target Policy (b) learning curves of the Pong environment when saving and loading the parameters from the same environment compared to the baseline. This plot serves to evaluate whether transfer is taking place effectively. The initial dips in the policies are due to: (a) resetting ϵ to 0, making the agent explore the environment without exploiting already learned parameters; (b) the recorded return is the mean across multiple environments. Some may not have ended the episode, and thus, have not produced a return, which brings down the overall mean.

3.8 Experimental Setup

To reduce computational costs and maintain consistency, each environment was given exactly 10 million steps during training. Although not all environments reach convergence within the given timeframe, most of them will show clear signs of learning and approach convergence. To ensure accuracy and reliability, each configuration is run five times using different random seeds, and the final result will be calculated as a mean. This will smooth out any inconsistencies or outliers that might be encountered. The training simulations are run on 128 parallel environments, and the test simulations on 8. A full list of hyperparameters can be found in B.

3.9 Implementation Details

The TL code was adapted from [28], with marginal changes. Since the observation space of both the MinAtar and Atari is three-dimensional (only differs in size), the convolutional network structure was reused. Varying action sizes affect the output layer, so if the environments share the same action size, the full Q-network is retained. Otherwise, the output layer is reinitialized.

To speed up the simulation running process, a shell script was composed to enable fast and robust adjustments to accommodate the various configurations of the experiment.

4 Results

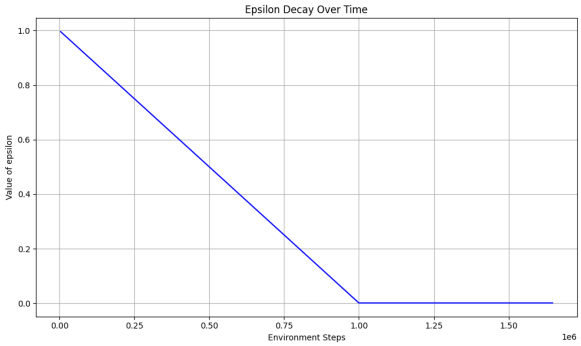
For a full table of results, refer to A.

4.1 Pong and Breakout

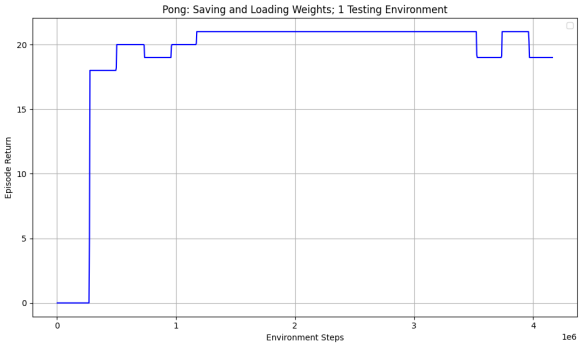
4.1.1 Pong Transfer from Breakout

Figure 4.1 showcases the results of PQN agents learning the *Pong* environment, both with and without transfer from the *Breakout* environment. Image 4.1a represents the training policy (ϵ-greedy), while 4.1b shows the target policy (greedy). In both cases, we see positive results from transfer learning - a steeper learning curve that eventually converges to the final return value as per the baseline. The baseline simulation achieves convergence around the 4 million environment steps mark, while we see a significant improvement from the transfer run, achieving convergence near the 2 million step mark, noting double the convergence speed.

The convergent values stand at around 20, namely (20.85 ± 0.08) and (20.60 ± 0.22) for target policies of the baseline and transfer runs, respectively. These results indicate that transfer was suc-



(a) Plot of ϵ-values for a *Pong* run.



(b) Target Policy with 1 testing environment.

Figure 3.2: Experimental proofs of method evaluation hypotheses for the behaviour and target policies. Figure (a) shows a plot of ϵ values, depicting maximal values of 1 at the start of the simulation, which persist until the 1 million step mark. This explains why we see a dip in initial behaviour performance, as the agent is exploring the environment with a very high probability. Figure (b) shows a saving and loading run in the testing setting in the *Pong* environment, when the `TEST_ENVS` variable is set to 1, signifying that testing was performed on 1 environment. The graph shows an immediate spike to the previously converged value after an initial period. This confirms that the return value is calculated as a mean of the parallel environments, which explains why we see the return start from 0 initially, as the environment has not yet finished the episode.

cessful, as the learning curve is steeper and achieves the same convergent value.

4.1.2 Breakout Transfer from Pong

In Figure 4.2, we notice results reminiscent of *Pong*. Although convergence is uncertain from the graphs due to the limited environment step count, the TL curve shows a significantly steeper gradient. Moreover, the TL curve shows greater signs of convergence, as the final 2 million environment steps seem to flatten. The final recorded value of the training baseline after ten million environment steps is approximately (289.70 ± 9.78), while the TL run improves to (332.09 ± 8.17). This could be attributed to the fact that the baseline run did not fully converge at the 10 million environment step mark.

To better compare the acceleration of the learning process, we can compare the difference in return across a few milestones, namely, at the 2 million, 5 million, and 7 million step marks. At the 2 million step mark, the transfer configuration records a significant improvement in the return value compared to the baseline: (111.92 ± 22.92) as opposed to (45.71 ± 2.57), noting a staggering 144.8% increase. The difference between the two configurations decreases in the later milestones (71.1% and 39.2% in the 5 million and 7 million step marks, respectively); however, there is clear evidence to show faster learning.

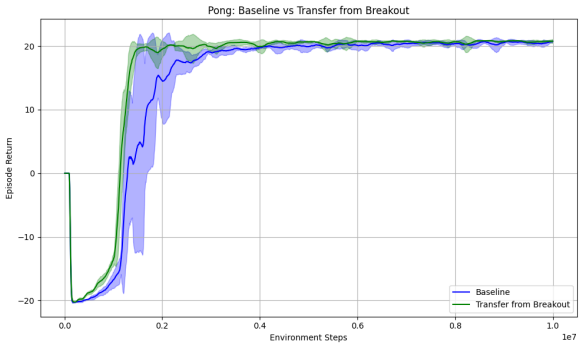
We also see significant fluctuations in values throughout the run, depicted by the large standard deviation region. However, the mean still stands at a higher value all the way throughout. These fluctuations could be due to the small number of runs (5), where large oscillations are present, and the results are affected by outliers.

4.2 Space Invaders and Demon Attack

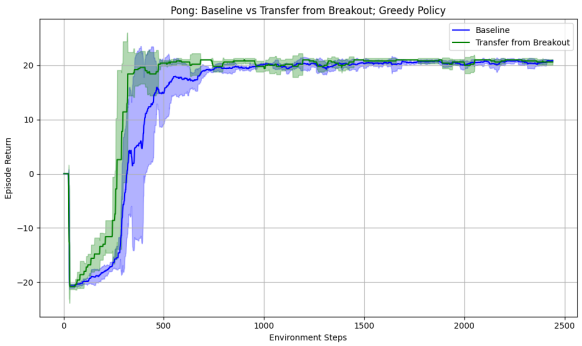
4.2.1 Space Invaders Transfer from Demon Attack

Figure 4.3 contains the results of the transfer from *Demon Attack* to *Space Invaders*, alongside the *Space Invaders* baseline. Similarly to *Breakout*, convergence is not reached, however, the results show that the transfer run is highly favourable, as the curve consistently sits above the baseline (Figure 4.3a), with a 21.6% increase in the return in early training (2 million steps).

Although significant oscillations are present in the target policy (Figure 4.3b)), the overall trend

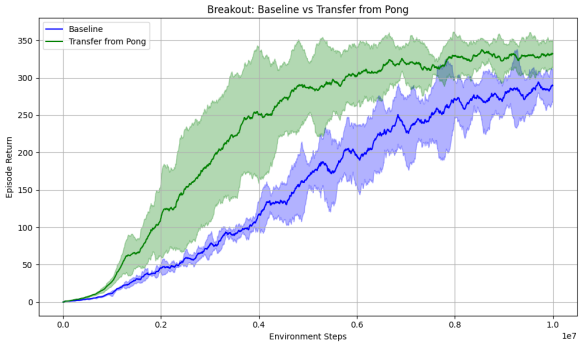


(a) Behaviour Policy

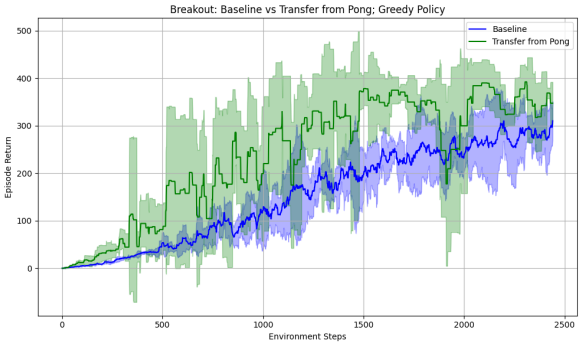


(b) Target Policy

Figure 4.1: Behaviour and Target Policy learning curves of the Pong environment with transfer from the Breakout environment compared to the baseline. The curves represent the mean return across 5 runs with different random seeds, and the shaded region shows 1 standard deviation. We can see that in both the behaviour and target policy, the agent replicates the baseline's convergent return, however, it reaches this value roughly twice as fast.



(a) Behaviour Policy



(b) Target Policy

Figure 4.2: Behaviour and Target Policy learning curves of the Breakout environment with transfer from the Pong environment compared to the baseline. The curves represent the mean return across 5 runs with different random seeds, and the shaded region shows 1 standard deviation. Although convergence is not reached within the given timeframe, we see that the curve is steeper, demonstrating faster learning.

showcases improved performance compared to the baseline. The final recorded test values are (1568.00 ± 478.71) for the TL run and (1035.63 ± 58.18) for the baseline, capturing the enhanced mean performance. However, at certain times, the TL curve underperforms the baseline, such as at the 2 million step mark. This could be a result of outliers stemming from the total number of runs, but could also be due to early lower-quality performance, which we see improve in the later milestones.

4.2.2 Demon Attack Transfer from Space Invaders

As for the transfer from *Space Invaders* to *Demon Attack* in Figure 4.4, we once again see a similar trend. Training performance is improved (Figure 4.4a), although we notice some overlap between the shaded regions. Learning speed is evidently faster with the transfer run, showing more than double the return (119.7%) in the early 2 million step milestone.

The target policy (Figure 4.4b) is fairly noisy and even underperforms the baseline at certain times. A similar sentiment is exhibited in the reverse transfer, as mentioned earlier. Since both *Space Invaders* and *Demon Attack* have the same action size (6), the full Q-network remains, without any reinitialization. Therefore, these results could have arisen from negative transfer, which was not previously present in the *Pong-Breakout* pair. A further show of potential negative side-effects is the actual return value: (12334.00 ± 4623.41) for the TL run, and a better (12841.86 ± 2043.75) for the baseline at the final timestep, but also in the 7 million step milestone: (6339.00 ± 3183.04) compared to(7343.38 ± 902.81).

4.3 Wizard of Wor Transfer from Pong

As for the additional trial with Wizard of Wor, we see a similar outcome as to that of *Space Invaders-Demon Attack*, shown in Figure 4.5. The training curve (Figure 4.5a) consistently outperforms the baseline, with return values almost doubling in the early to mid milestones (2 million and 5 million), and eventually landing at (4364.32 ± 329.75) compared to (3424.75 ± 443.02) in the final return.

On the other hand, the target policy (Figure 4.5b) shows large fluctuations, but maintains a general upwards trend, slightly higher than the baseline. This time, though, the transferrer environment (Pong) has a smaller action size compared to the transferee (Wizard of Wor), which means that the output layer is reinitialized, and less knowledge is retained. Since the two environments are vastly different in both their appearance and overall gameplay, reinitialization could benefit the results.

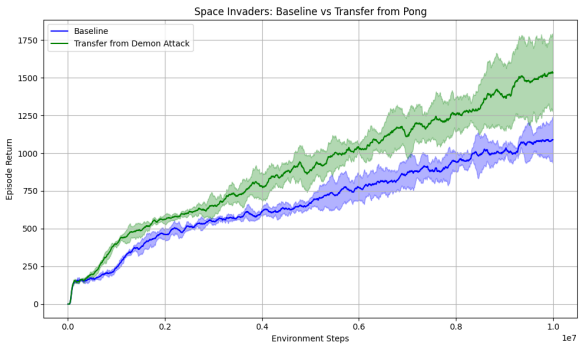
4.4 Overall Results

To summarize the findings of the experiment, we can see general improvements across the board. TL configurations in both environment pairs have shown improved learning speeds in both directions, with satisfactory results in testing. Noise and fluctuations are present throughout testing, possibly as a result of low seed variations and short runtimes. Although convergence is not explicitly reached within the given timeframe for certain environments, generally faster learning is evident through the use of consistent recorded milestones, with *Pong* showing clear evidence of faster convergence to the target value. We also notice that the *Pong-Breakout* pair shows more consistent results, as opposed to *Space Invaders-Demon Attack*. Overall, this shows that transfer learning can be beneficial for the *Atari* environment when using the PQN algorithm, as it presents faster convergence.

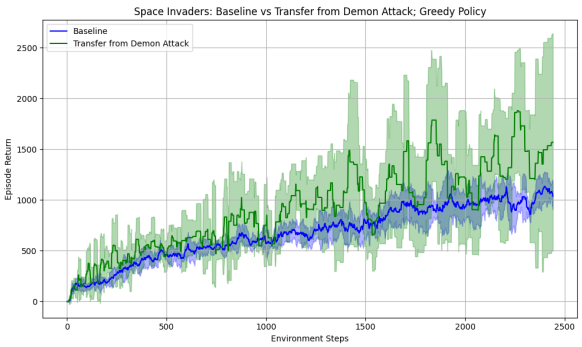
5 Discussion

Although transfer learning has shown varying results across different domains and environments, it remains an important and potentially revolutionary approach to reinforcement learning. With the results from this paper's experiment, we see that, when combined with the right environment, TL can achieve significant results, which might be highly beneficial to reducing computational costs and addressing issues of data sparsity.

Compared to the results of the study by Zaharia, the *Atari* environment seems to be significantly more suited towards transfer learning. As discussed earlier, *MinAtar*'s inherent structure was potentially the leading culprit of underwhelming results. As the observation space differs among

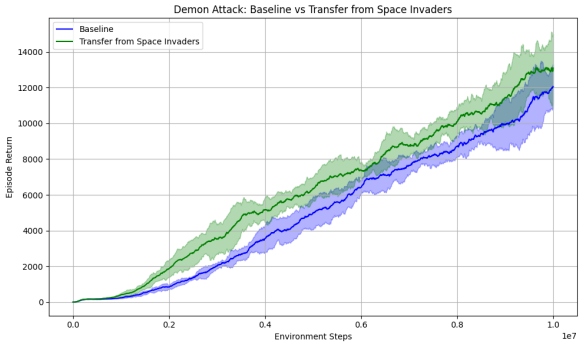


(a) Behaviour Policy

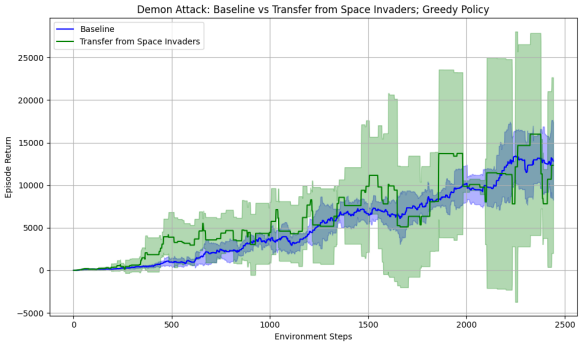


(b) Target Policy

Figure 4.3: Behaviour and Target Policy learning curves of the Space Invaders environment with transfer from the Demon Attack environment compared to the baseline. The curves represent the mean return across 5 runs with different random seeds, and the shaded region shows 1 standard deviation. Although convergence is not reached within the given timeframe, we see that learning is accelerated in (a); on the other hand, in (b), we notice severe fluctuations where the transfer run frequently underperforms the baseline. This indicates the negative implications of transfer between environments without any parameter reinitialization.

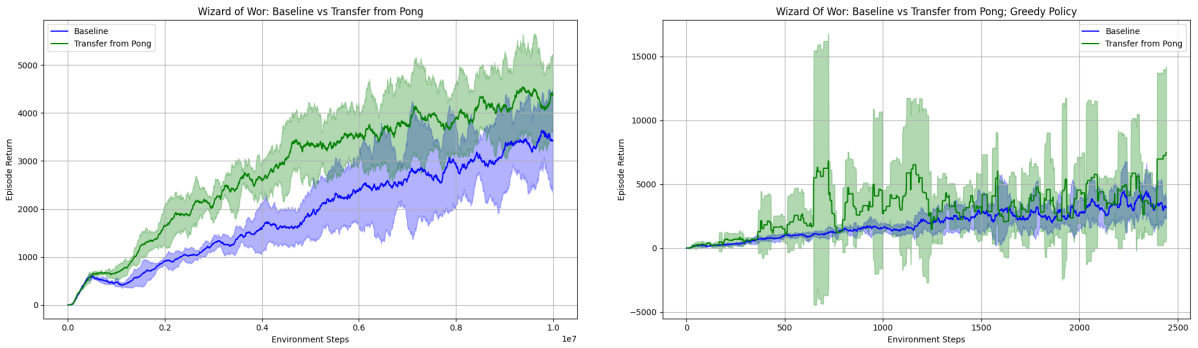


(a) Behaviour Policy



(b) Target Policy

Figure 4.4: Behaviour and Target Policy learning curves of the Demon Attack environment with transfer from the Space Invaders environment compared to the baseline. The curves represent the mean return across 5 runs with different random seeds, and the shaded region shows 1 standard deviation. Similarly to its reversed pair, transfer from *Space Invaders* to *Demon Attack* shows accelerated learning, however, often underperforms the baseline when testing, showcasing the risks of full parameter transfer.



(a) Behaviour Policy **(b) Target Policy**

Figure 4.5: Behaviour and Target Policy learning curves of the Wizard Of Wor environment with transfer from the Pong environment compared to the baseline. The curves represent the mean return across 5 runs with different random seeds, and the shaded region shows 1 standard deviation. Once again, we notice faster learning and oscillating, unsatisfactory results from the testing environments. This highlights the potential negative outcomes of transferring parameters between two dissimilar environments.

the different games, large portions of the training network are scrapped due to their incompatibility with the transferee. Furthermore, the simplicity of the environment did not enable the agent to form abstract or generalized knowledge, effectively plummeting the chances of the transfer succeeding.

With that being said, although this study's results are satisfactory, certain notes and limitations are to be addressed. This is to ensure credibility and contribute to an in-depth analysis of the results. Additionally, this section will delve deeper into the implications and theoretical relations of the results.

5.1 Analysis of Results

From the results obtained in the previous section, we made some observations regarding the improved performance of the *Pong-Breakout* pair compared to *Space Invaders-Demon Attack*. Research conducted on progressive nets (another form of TL) showed similar results in the Atari environment. According to Rusu et al., the largest amount of negative transfer occurs between environments with complete dependence of the transferred network. This is explained as the new agent's inability to overwrite the existing knowledge effectively, leading to unsatisfactory results that lead to a suboptimal local minimum. The transfer showed the most positive results when certain layers of the network were augmented by new features of the receiving environment.

A separate research done by Wang et al. showed that positive transfer heavily relied on overlapping features present in earlier layers of the network. This implies that certain visual patterns or gameplay mechanics (like object motion, enemy behavior, and scoring zones) are consistently recognized by the network across different games. They highlight that there are benefits from feature transfer, as they contain shared representations that encode generalized structures.

The trends we notice in our results are likely to be a combination of multiple variables, as it is difficult to pinpoint the exact source of the positive transfer.

The success of the results can also be attributed to the PQN algorithm and its positive implications when crossed over with TL. As mentioned earlier, a replay buffer is a structure used to store the agent's previous experiences, with the intent of using them to bootstrap estimations. However, a recurring problem arises - the primacy bias is encountered. As described by Nikishin et al., the primacy bias can be characterized as obtaining and retaining potentially negative information that was present at the very beginning of training. This is because

the agent effectively learns incorrect patterns that are then difficult to eradicate. Oftentimes, this is particularly present when using a replay buffer - the agent is encouraged to return to its previous experiences, exposing it to harmful data it gathered during early training. When translated to TL, faulty knowledge can be transferred to a new agent, making it disadvantaged right from the get-go. Since the PQN algorithm does not use a replay buffer, it avoids the risk of transferring negative information through the notion of the primacy bias.

Another inherent characteristic of the PQN algorithm is its omission of the target network. Studies such as [18] show unfavourable results when applying TL to the DQN algorithm. Although not explicit about the effects of the target network, this is additional evidence of negative outcomes when performing transfer with the presence of a target network. A study done by Piché et al. showcased an alternative structure to the target network (Functional Regularization), which demonstrated effects such as faster convergence and improved performance in the *Atari* environment compared to the baseline target network. The results highlighted that the target network can act as an implicit regularizer, oftentimes restricting fine-tuning on new tasks, in turn constraining fast adaptation to a new game environment. Since PQN omits the target network and opts for alternative regularization methods, it eliminates the risks and drawbacks associated with the target network.

Another viable concept to consider is plasticity in RL. Plasticity refers to the ability of a pretrained model to adapt effectively to a new task by modifying its internal representations. The critical part of reaching peak plasticity is finding the right balance between flexibility and rigidity. While having flexible parameters can help accelerate learning, excessive rigidity could hinder the model's adaptation to a new environment. In a study done by Kirkpatrick et al., an algorithm with variable plasticity was introduced. By keeping certain parts of the pre-trained network frozen during retraining, they were able to avoid catastrophic forgetting and maximize performance. This puts forward interesting ideas about selective retention, where identifying and preserving key features among environments creates stability during training, while also allowing effective learning.

5.2 Limitations and Challenges

One of the key limitations of this study was the computational expenses and time. As evident from the graphs, we face large fluctuations and many outliers are present, which hinder us from seeing a clear picture of the results. The study could be significantly elevated through exhaustive simulations, e.g., instead of five runs per configuration, opting for 10-20, as this would more accurately represent the results and would be more effective in eradicating outliers. Moreover, the environment step limit should be increased from the current 10 million steps to 50 million steps, as specified in the original study [9]. Since the hardware setup was not on par with the one available to the original researchers, the difference in the length of simulations was apparent, as we would encounter a doubling or even tripling of the runtime. Due to a tight schedule, the step limit was dramatically decreased, which caused convergence of certain runs to be sacrificed.

5.3 Future Work

To address the limitations mentioned in the previous subsection, there are a few experiments that can be conducted.

Firstly, running the simulations for a larger quantity of random seeds will allow us to smooth out any potential outliers, creating a clearer graph with fewer fluctuations. When coupled with high variability, a small number of runs leads to substantial statistical uncertainty, especially when reporting point estimates. [1] Moreover, research shows that increasing the number of random seeds reduces uncertainty and helps obtain reliable results [6]. This experiment was conducted using 5 random seeds in the range [0,4], which is within the usually used 3-10 in the domain of RL. However, due to the clearly statistically uncertain results, increasing the number of random seeds could drastically improve reliability. On a similar note, running this experiment on further game pairs can produce more reliable and generalizable results. Comparing multiple pairs that cross over can give a more comprehensible overview of which variables come into play, e.g., tracking if games with the same action size perform better than those with differing ones can

determine if augmentation of the network is more beneficial compared to full dependence or leveraging of shared representations.

Secondly, to better ensure convergence, the number of environment steps allowed for each environment should be increased. The original paper ([9]) set the default limit to 50 million steps, which was outside the available scope for our setup. However, computational costs would suffer dramatically with this tweak. In order to reach a middle ground, one could test-run each needed game until convergence is reached to get an estimate of the required steps, and subsequently assign each game a suitable limit. Further checks can be carried out by referring to a benchmark such as [7], where the average returns of leading algorithms can be found. This verifies what the intended convergence value for each game should be, and the step limit can be adjusted accordingly.

Lastly, with the idea of plasticity in mind, further work can focus on identifying key features of *Atari* games and selectively freezing the corresponding parameters when transferring. By analyzing the maximum return and fastest convergence from various configurations of game pairs and pinpointing which layers play the most important role in a successful transfer, we can then selectively retain them and maximize the quality of data that is transferred.

6 Conclusion

This study underlines the potential of implementing transfer learning techniques in reinforcement learning domains. It does this through the analysis of the PQN algorithm in the *Atari* environment, serving as a benchmark for experimental results. Its findings showcase the possibility of successful transfers between different environments, where convergence is faster while maintaining benchmark performance.

The results obtained from this experiment demonstrate the importance of shared representations and sufficient complexity of environments, which allow the agent to learn underlying patterns and utilize them in unseen spaces. Moreover, PQN's inherent structure, characterized by the omission of the replay buffer and target network, seems especially suited for parameter transfer tasks - the architecture works around issues such as primacy bias and allows for faster learning. Compared to results produced in the *MinAtar* environment [28], this experiment highlights the significance of rich visual representations and task diversity, both present in *Atari*. The key may also lie in the environments' inherent similarities, as discussed by [24]. This shows the benefits of feature transfer, containing valuable generalized information that can be used across multiple agents in separate environments. On the other hand, studies like [17] highlight the risk of negative transfer between fully dependent environments. Reusing the complete network might result in suboptimal plateaus, as exploration is not encouraged enough to contribute to the already formed knowledge base. This dichotomy requires further work, focusing on identifying whether either is more suitable for an environment such as *Atari*, or if finding an equilibrium through notions such as variable plasticity is better.

In conclusion, transfer learning is a promising, viable direction to pursue in reinforcement learning. With more research and analysis, the field will continue to grow and improve through the contributions of refined experimentation and investigation. By identifying blind spots and constructively tackling them, transfer learning will become a leading technique used in various fields and real-life decision-making dilemmas.

References

[1] R. Agarwal, M. Schwarzer, P. S. Castro, A. C. Courville, and M. Bellemare. Deep reinforcement learning at the edge of the statistical precipice. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 29304–29320. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/f514cec81cb148559cf475e7426eed5e-Paper.pdf.

[2] A. Barreto, W. Dabney, R. Munos, J. J. Hunt, T. Schaul, H. P. van Hasselt, and D. Silver. Successor features for transfer in reinforcement learning. In I. Guyon, U. V.

Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/350db081a661525235354dd3e19b8c05-Paper.pdf.

[3] T. Bartz-Beielstein, C. Doerr, D. van den Berg, J. Bossek, S. Chandrasekaran, T. Eftimov, A. Fischbach, P. Kerschke, W. L. Cava, M. Lopez-Ibanez, K. M. Malan, J. H. Moore, B. Naujoks, P. Orzechowski, V. Volz, M. Wagner, and T. Weise. Benchmarking in optimization: Best practice and open issues, 2020. URL https://arxiv.org/abs/2007.03488.

[4] V. Beiranvand, W. Hare, and Y. Lucet. Best practices for comparing optimization algorithms. *Optimization and Engineering*, 18(4): 815–848, 2017.

[5] 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.

[6] C. Colas, O. Sigaud, and P.-Y. Oudeyer. How many random seeds? statistical power analysis in deep reinforcement learning experiments, 2018. URL https://arxiv.org/abs/1806.08295.

[7] J. Fan. A review for deep reinforcement learning in atari:benchmarks, challenges, and solutions, 2023. URL https://arxiv.org/abs/2112.04145.

[8] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks, 2017. URL https://arxiv.org/abs/1703.03400.

[9] M. Gallici, M. Fellows, B. Ellis, B. Pou, I. Masmitja, J. N. Foerster, and M. Martin. Simplifying deep temporal difference learning, 2025. URL https://arxiv.org/abs/2407.04811.

[10] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), Apr. 2018. doi: 10.1609/aaai.v32i1.11796. URL https://ojs.aaai.org/index.php/AAAI/article/view/11796.

[11] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. In *PNAS*, 2017.

[12] M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. Hausknecht, and M. Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018. doi: 10.1613/jair.5569. URL https://jair.org/index.php/jair/article/view/11182.

[13] M. Matthews, M. Beukman, B. Ellis, M. Samvelyan, M. Jackson, S. Coward, and J. Foerster. Craftax: A lightning-fast benchmark for open-ended reinforcement learning, 2024. URL https://arxiv.org/abs/2402.16801.

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

[15] E. Nikishin, M. Schwarzer, P. D'Oro, P.-L. Bacon, and A. Courville. The primacy bias in deep reinforcement learning. In K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 16828–16847. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/nikishin22a.html.

[16] A. Piché, V. Thomas, R. Pardinas, J. Marino, G. M. Marconi, C. Pal, and M. E. Khan. Bridging the gap between target networks and functional regularization, 2023. URL https://arxiv.org/abs/2106.02613.

[17] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. Progressive neural networks, 2022. URL https://arxiv.org/abs/1606.04671.

[18] M. Sabatelli and P. Geurts. On the transferability of deep-q networks, 2021. URL https://arxiv.org/abs/2110.02639.

[19] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA, 2nd edition, 2018. ISBN 978-0-262-03924-6. URL http://www.incompleteideas.net/book/the-book.html. Second edition; online version available.

[20] C. Tang, B. Abbatematteo, J. Hu, R. Chandra, R. Martín-Martín, and P. Stone. Deep reinforcement learning for robotics: A survey of real-world successes. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(27): 28694–28698, Apr. 2025. doi: 10.1609/aaai.v39i27.35095. URL https://ojs.aaai.org/index.php/AAAI/article/view/35095.

[21] M. E. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(7), 2009.

[22] L. Torrey and J. Shavlik. Transfer learning. In *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques*, pages 242–264. IGI Global, 2010. doi: 10.4018/978-1-60566-766-9.ch011. URL https://doi.org/10.4018/978-1-60566-766-9.ch011.

[23] M. van Otterlo and M. Wiering. *Reinforcement Learning and Markov Decision Processes*, pages 3–42. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-27645-3. doi: 10.1007/978-3-642-27645-3_1. URL https://doi.org/10.1007/978-3-642-27645-3_1.

[24] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas. Dueling network architectures for deep reinforcement learning. In M. F. Balcan and K. Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1995–2003, New York, New York, USA, 20–22 Jun 2016. PMLR. URL https://proceedings.mlr.press/v48/wangf16.html.

[25] C. J. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3):279–292, 1992. doi: 10.1007/BF00992698.

[26] K. Weiss, T. M. Khoshgoftaar, and D. Wang. A survey of transfer learning. *Journal of Big Data*, 3(1):9, 2016. doi: 10.1186/s40537-016-0043-6. URL https://doi.org/10.1186/s40537-016-0043-6.

[27] K. Young and T. Tian. Minatar: An atari-inspired testbed for thorough and reproducible reinforcement learning experiments. *arXiv preprint arXiv:1903.03176*, 2019.

[28] C. Zaharia. Transfer learning in reinforcement learning: A study on the parallelized q-network, 2025. URL https://fse.studenttheses.ub.rug.nl/id/eprint/34861.

[29] Z. Zhu, K. Lin, A. K. Jain, and J. Zhou. Transfer learning in deep reinforcement learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45 (11):13344–13362, 2023. doi: 10.1109/TPAMI.2023.3292075.

A Mean Return Tables

Table A.1: Final mean return for the Pong-Breakout game pair after 10 million environment steps. The results are averaged across 5 runs, and also include an uncertainty of 1 standard error. Both in the behaviour and target policy, the return is matched to that of the baseline, showing that the transfer learning configuration maintains accuracy.

Configuration	Target Policy Mean ± SE	Behavior Policy Mean ± SE
Pong Baseline	20.85 ± 0.08	20.75 ± 0.08
Pong Transfer from Breakout	20.60 ± 0.22	20.77 ± 0.14
Breakout Baseline	310.10 ± 18.25	289.70 ± 9.78
Breakout Transfer from Pong	347.20 ± 19.90	332.09 ± 8.17

Table A.2: Final mean return for the Space Invaders-Demon Attack game pair after 10 million environment steps. The results are averaged across 5 runs, and also include an uncertainty of 1 standard error. The recorded TL return is higher than the baseline for transfer to Space Invaders; however, we see a slight decrease in the target policy when transferring to Demon Attack. Overall, the results demonstrate that the transfer was successful, but not as significantly as that of the *Pong-Breakout* pair.

Configuration	Target Policy Mean ± SE	Behavior Policy Mean ± SE
Space Invaders Baseline	1035.63 ± 58.18	1089.65 ± 67.24
Space Invaders Transfer from Demon Attack	1568.00 ± 478.71	1532.20 ± 115.14
Demon Attack Baseline	12841.86 ± 2043.75	12048.15 ± 552.63
Demon Attack Transfer from Space Invaders	12334.00 ± 4623.41	13094.01 ± 844.61

Table A.3: Final mean return for Wizard of Wor after 10 million environment steps. The results are averaged across 5 runs, and also include an uncertainty of 1 standard error. The final recorded mean shows overall improvement. However, due to high oscillations of the return values throughout the training process, the final recorded return does not clearly show the full picture of TL's performance compared to the baseline.

Configuration	Target Policy Mean ± SE	Behavior Policy Mean ± SE
Wizard of Wor Baseline	3150.00 ± 374.17	3424.75 ± 443.02
Wizard of Wor Transfer from Pong	7460.00 ± 3018.37	4364.32 ± 329.75

Table A.4: Training returns at milestones of 2 million, 5 million, and 7 million environment steps for each configuration. The return represents a mean across 5 runs, taken at a time step closest to the milestone mark. This table aims to draw attention to the progress of each configuration during the training process, as most do not converge. The results highlight that early performance is significantly improved in the transfer learning configuration compared to the baseline. As we approach the end of the simulations, improvements decrease; however, they maintain the lead to the baseline throughout. This indicates that the transfer learning configuration has an advantage to the baseline, and shows accelerated learning.

Configuration	Performance at 2M Steps		Performance at 5M Steps		Performance at 7M Steps	
	Baseline	Transfer	Baseline	Transfer	Baseline	Transfer
Pong (from Breakout)	14.54 ± 2.84	19.49 ± 0.60	19.91 ± 0.35	20.65 ± 0.08	20.39 ± 0.28	20.77 ± 0.04
Breakout (from Pong)	45.71 ± 2.57	111.92 ± 22.92	167.58 ± 7.67	286.73 ± 15.86	229.47 ± 18.82	319.40 ± 11.62
Space Invaders (from DA)	460.73 ± 16.63	560.02 ± 12.92	667.58 ± 15.55	890.20 ± 41.40	869.08 ± 42.63	1111.87 ± 63.96
Demon Attack (from SI)	845.78 ± 50.34	1857.93 ± 211.50	4986.58 ± 416.28	6335.74 ± 288.93	7646.68 ± 224.57	8762.16 ± 256.81
Wizard of Wor (from Pong)	907.20 ± 28.24	1640.03 ± 187.67	1871.17 ± 229.26	3318.41 ± 342.98	2617.81 ± 441.46	3721.14 ± 392.85

Table A.5: Testing returns at milestones of 2 million, 5 million, and 7 million environment steps for each configuration. The return represents a mean across 5 runs, taken at a time step closest to the milestone mark. This table aims to draw attention to the progress of each configuration during the testing process, as most do not converge. The results highlight that oftentimes the transfer learning configuration shows improved results early on, with occasional underperformances compared to the baseline.

Configuration	Performance at 2M Steps		Performance at 5M Steps		Performance at 7M Steps	
	Baseline	Transfer	Baseline	Transfer	Baseline	Transfer
Pong (from Breakout)	14.78 ± 2.77	20.40 ± 0.22	20.23 ± 0.50	21.00 ± 0.00	20.57 ± 0.17	21.00 ± 0.00
Breakout (from Pong)	44.55 ± 6.04	81.40 ± 23.70	152.59 ± 26.64	288.00 ± 23.03	220.97 ± 8.47	361.80 ± 10.96
Space Invaders (from DA)	478.87 ± 10.67	443.00 ± 68.20	620.00 ± 36.62	907.00 ± 130.81	974.39 ± 73.06	1582.00 ± 264.97
Demon Attack (from SI)	968.59 ± 214.94	3884.00 ± 1317.04	4708.50 ± 712.47	5331.00 ± 1987.86	7343.38 ± 902.81	6339.00 ± 3183.04
Wizard of Wor (from Pong)	918.91 ± 86.65	1584.38 ± 447.21	2037.50 ± 324.23	4080.00 ± 2929.25	2227.50 ± 394.11	3040.00 ± 1274.56

B Configuration file

Table B.1: Hyperparameters applied to the baseline and transfer learning runs. These are kept the same as the original paper [9], with the exception of the `TOTAL_TIMESTEPS` and `TOTAL_TIMESTEPS_DECAY` which are reduced from 50 million to 10 million in order to decrease computational costs.

Parameter	Value
TOTAL_TIMESTEPS	1e7
TOTAL_TIMESTEPS_DECAY	1e7
NUM_ENVS	128
NUM_STEPS	32
EPS_START	1.00
EPS_FINISH	0.001
EPS_DECAY	0.1
NUM_EPOCHS	2
NUM_MINIBATCHES	32
NORM_TYPE	”layer_norm”
LR	0.00025
MAX_GRAD_NORM	10
LR_LINEAR_DECAY	False
GAMMA	0.99
LAMBDA	0.65
TEST_DURING_TRAINING	True
TEST_ENVS	8
EPS_TEST	0