ON THE TRANSFERABILITY OF DEEP-Q NETWORKS

ABSTRACT

Transfer Learning (TL) is an efficient machine learning paradigm that allows overcoming some of the hurdles that characterize the successful training of deep neural networks, ranging from long training times to the needs of large datasets. While exploiting TL is a well established and successful training practice in Supervised Learning (SL), its applicability in Deep Reinforcement Learning (DRL) is rarer. In this paper, we study the level of transferability of three different variants of Deep-Q Networks on popular DRL benchmarks as well as on a set of novel, carefully designed control tasks. Our results show that transferring neural networks in a DRL context can be particularly challenging and is a process which in most cases results in negative transfer. In the attempt of understanding why Deep-Q Networks transfer so poorly, we gain novel insights into the training dynamics that characterizes this family of algorithms.

1 INTRODUCTION

Over the last years, the marriage between Reinforcement Learning (RL) algorithms and deep neural networks, commonly denoted as Deep Reinforcement Learning (DRL) (François-Lavet et al., 2018) has gained tremendous attention (Henderson et al., 2018). Neural networks have, in fact, proven to be extremely successful both in a model-free RL setting as in a model-based one. Large part of their success can be attributed to their ability of serving as feature extractors as well as function approximators, a property that allows them to successfully learn optimal value functions (Mnih et al., 2013; 2015; Van Hasselt et al., 2016; Zhao et al., 2016; Wang et al., 2016b; Sabatelli et al., 2020), stochastic policies (Lillicrap et al., 2015; Schulman et al., 2015b;a; Wang et al., 2016a; Mnih et al., 2016; Schulman et al., 2017; Haarnoja et al., 2018; Fujimoto et al., 2018), and models of an environment (Ha & Schmidhuber, 2018; Kaiser et al., 2019; Hafner et al., 2019a;b; 2020) that is usually formalized as a Markov Decision Process (MDP) Puterman (1990). Despite the many remarkable achievements, training a DRL agent is a process that can be very time-consuming. The task of solving an optimal decision making problem is, in fact, a challenging problem of its own, which is sometimes made even more difficult by the DRL community itself, which requires DRL practitioners to test the performance of their algorithms on benchmarks that are computationally very expensive (for a position paper about this topic see (Obando-Ceron & Castro, 2020)). One way of overcoming the need of individually training a DRL agent from scratch each time a new RL problem is encountered is based on Transfer Learning (TL). TL focuses on designing training strategies that allow machine learning models to retain and reuse previously learned knowledge when getting trained on new, unseen problems (Pan & Yang, 2009; Zhuang et al., 2020). Within deep learning, TL is largely adopted by the Supervised Learning (SL) community (Huh et al., 2016; Mormont et al., 2018; Sabatelli et al., 2018a; Domínguez Sánchez et al., 2019; Vandaele et al., 2021; Ho & Kim, 2021), as it allows to train neural networks on problems that are characterized by a lack of appropriate training data or sufficient computational resources; however, typical TL approaches such as off the shelf feature extraction, or fine-tuning (Sharif Razavian et al., 2014) have rarely been thoroughly studied from a DRL perspective. Therefore, the degree of transferability of DRL algorithms is not yet known. In this work, we focus on value-based, model-free algorithms, a family of techniques which focuses on training neural networks with the intent of learning an approximation of an optimal value function. While several of such algorithms, commonly denoted as Deep-Q Networks, exist, research studying their TL properties is, on the contrary scarce, and a clear answer to the question *"How transferable are Deep-Q Networks?"* has yet to be given. In the attempt to clearly answering this question, we present the following three contributions:

- We present a first **large scale empirical study** that analyses the TL properties of popular model-free DRL algorithms on the Atari Arcade Learning Environment (ALE), where we show that transferring pre-trained networks in a DRL context can be a very challenging task.
- We design a set of **novel, control experiments** which allows us to thoroughly characterize the TL dynamics of Deep-Q Networks.
- While studying Deep-Q Networks from a TL perspective, we discover **novel learning dynamics** that provide a better understanding of how this family of algorithms deals with RL tasks.

2 A large-scale Empirical Study

In this section, we carry out a large-scale TL experiment on several games from the Atari Environment (Sec. 2.1). The experimental protocol is detailed in Sec. 2.2 and results are discussed in Sec. 2.3.

2.1 The Atari Environment

In this study, we use the Atari Arcade Learning Environment (ALE) (Bellemare et al., 2013). Next to being one of the most popular benchmarks in DRL, the ALE is particularly well suited for TL research as it allows to choose among a set of 57 Atari games that can be used as source $\mathcal{M}_S$ and target $\mathcal{M}_T$ MDPs within a deep transfer learning setting. Since training a model-free agent on the games of the ALE is a process which can be computationally very expensive, we have carefully selected a subset of 10 different environments. Numerous reasons guided the game selection process. First, we have selected games for which we guarantee that a model-free DRL agent can learn a good policy for. Since, as discussed by Lazaric (2012), one of the key requirements of TL is that of correctly identifying and transferring knowledge across source and target tasks, we naturally ensured that some knowledge coming in the form of neural network parameters representing a near-optimal value function was available for transfer. Second, while it is true that all of the selected games result in an agent that can improve its policy over time, some games were chosen because the learned policy resulted in a final performance that was not on par with that of a human expert player. This is, for example, the case of the `Frostbite` game, where the gap in performance between an agent trained with the DQV-Learning algorithm (Sabatelli et al., 2018b) ($\approx$ 270) and a human expert player ($\approx$ 4300) is particularly significant. It follows that `Frostbite` is an interesting target task for transfer, as the agent's performance can potentially be improved through TL. Furthermore, we have also ensured that among the selected games, some environments are more similar to each other than others. This is, for example, the case for the `Ms. Pacman` and `Bank Heist` games which, as can be seen in Fig. 1, are two games where the state space is represented as a maze, and where the end goal of an agent is that of learning how to navigate it. In like manner, we have also included games that are very different from each other as is, e.g., the case for the `Crazy Climber` and `Pong` games, where it is clear from Fig. 2 that no visual similarities are shared among the two environments. Including visually similar and dissimilar games allows us to investigate whether, as is the case for supervised learning, a source task is particularly well suited for transfer if it is similar to its respective target task (Mensink et al., 2021).

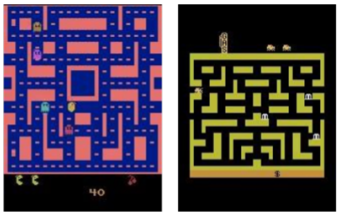

Figure 1: The visually similar `Ms Pacman` and `Bank Heist` games.

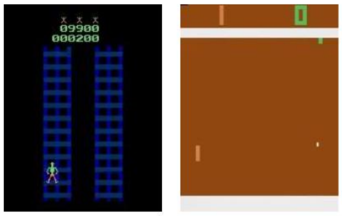

Figure 2: The highly different `Crazy Climber` and `Pong` games.

2.2 Experimental Setup

We investigate the TL performance of agents that get trained with the DQV-Learning algorithm (Sabatelli et al., 2018a), and with the DDQN algorithm (Van Hasselt et al., 2016). We take models which come as pre-trained on 10 Atari games and transfer them to all remaining environments. We mostly consider the same games for both algorithms (`Bank Heist`, `Boxing`, `Crazy Climber`, `Fishing Derby`, `Frostbite`, `James Bond`, `Ms. Pacman`, `Pong` and `Zaxxon`) with the only exception that for DDQN `Frostbite` is replaced by `Gopher`, and `Zaxxon` is replaced by `Ice Hockey` as the `Frostbite` and `Zaxxon` DDQN agents failed to improve their policy whilst training. It is worth noting that TL is particularly easy to perform as both algorithms learn an approximation of the optimal state-action value function $Q(s, a; \theta)$ by training a convolutional neural network (see Appendix A for its exact architecture and all hyper-parameter settings) directly on the images representing the state of the game. Since the state space across Atari games is always represented as an $84 \times 84 \times 4$ tensor, it is straightforward to transfer the same neural architecture among various Atari environments without needing special modifications. However, the only modification that we apply to a pre-trained network concerns its last layer responsible for estimating the different Q values, which we always replace and randomly re-initialize. Following the typical deep transfer learning literature, we investigate whether in DRL it is as beneficial as it is in supervised learning to transfer a network that comes as pre-trained on $\mathcal{M}_S$ and fine-tune it on $\mathcal{M}_T$. We do this by quantitatively assessing the transfer learning benefits on each $\mathcal{M}_S/\mathcal{M}_T$ pair by computing the area ratio metric r (Taylor & Stone, 2009). Specifically, given a learning curve representing the performance of an agent pre-trained on $\mathcal{M}_S$, and that of an agent that is instead trained from scratch, we compute r as follows:

$$r = \frac{\text{area of } \mathcal{M}_S - \text{area of } \mathcal{M}_T}{\text{area of } \mathcal{M}_T}. \tag{1}$$

2.3 Results

The results on each $\mathcal{M}_S/\mathcal{M}_T$ pair for both the DQV and DDQN algorithms are presented in Table 1. In each cell of the tables, we report the area ratio metric defined in Eq. 1: the lower (resp. higher) this score, the less (resp. more) beneficial it is to transfer and fine-tune a pre-trained agent. When it comes to the DQV algorithm, we can see that, out of nine target environments, there is only one Atari game for which it is always beneficial to transfer and fine-tune a pre-trained model: `Fishing Derby`. In fact, a positive area ratio score is obtained no matter which source environment is used for pre-training, although the best results have been obtained when starting from an `Enduro` or `Pong` pre-trained network, which both resulted in an area ratio score of ≈ 0.72. Positive transfer can also be observed on the `Frostbite` and `James Bond` games, but only for a limited number of source games. For example, a `Bank Heist` pre-trained agent transfers well to both target games as it obtains an area ratio score of 0.729 and 0.973 respectively, but the same cannot be said for an `Enduro` pre-trained network, which on `Frostbite` results in absent transfer (the area ratio score is, in fact, -0.017), and yields negative transfer on `James Bond` ($r = -0.41$). We can also observe that there are environments where it is surprisingly never beneficial to transfer and fine-tune a pre-trained agent. This is, for example, the case for the `Bank Heist` and `Pong` games, where independently from which source game $\mathcal{M}_S$ is used for pre-training, a negative area ratio score is always obtained. Furthermore, it can also be observed that transfer learning across environments is not symmetric, as one source game $\mathcal{M}_S$ can result in positive transfer when it gets transferred to a certain target game $\mathcal{M}_T$, but the same outcome is not obtained when transfer is performed in the opposite direction. As an example we can consider the `Boxing/Fishing Derby` games: positive transfer is obtained when transferring from `Boxing`→`Fishing Derby` ($r = 0.552$), but negative transfer is obtained when transferring from `Fishing Derby` → `Boxing` ($r = -0.893$). When it comes to the DDQN algorithm, similar conclusions can be drawn: we can again observe that there are only very few cases for which it is beneficial to transfer and fine-tune a pre-trained DRL agent. Examples of such cases are networks that are pre-trained on `Ice Hockey` and `James Bond` which get transferred to `Boxing` ($r = 0.245$ and $r = 0.232$ respectively), or `Boxing` and `Enduro` models that get transferred to `Pong` ($r = 0.936$ and $r = 0.248$). `Bank Heist` and `Pong` are again the two target environments for which most of the transferred source models resulted in negative transfer, while differently from the experiments performed with the DQV algorithm, this time no positive transfer can be observed on `Fishing Derby`. Overall, the process of fine-tuning a pre-trained DDQN agent mostly results in

Table 1: The results obtained when fine-tuning ten different pre-trained agents (rows) on nine other Atari games (columns), with DQV (top table) and DDQN (bottom table). Positive values (in cyan) represent positive transfer, while negative values (in orange) represent negative transfer. The darker the color, the higher the absolute value of the area ratio score.

DQV	BankHeist	Boxing	CrazyClimber	Enduro	FishingDerby	Frostbite	JamesBond	MsPacman	Pong	Zaxxon
BankHeist	-	-0.019	-1	-0.317	0.5	0.729	0.973	-0.089	-1.238	-0.998
Boxing	-0.494	-	-0.278	-0.852	0.552	-0.01	0.247	-0.184	-0.841	-0.999
CrazyClimber	-0.569	-0.261	-	-0.593	0.19	0.277	0.621	-0.111	-1.206	-0.178
Enduro	-0.571	-0.018	-0.25	-	0.726	-0.017	-0.41	-0.08	-0.466	-0.164
FishingDerby	-1	-0.893	-0.093	-0.45	-	0.068	0.197	-0.136	-3.083	-0.999
Frostbite	-0.933	0.024	-1	-0.348	0.222	-	0.569	0.009	-0.663	-0.076
JamesBond	-0.123	-0.106	-0.131	-0.033	0.519	0.262	-	0.218	-1.329	-1
MsPacman	-0.985	-0.219	-0.012	-0.494	0.6	0.346	0.398	-	-1.646	-0.997
Pong	-1	-0.083	-0.428	-0.476	0.725	-0.024	0.896	0.123	-	-0.729
Zaxxon	-0.76	-0.028	0.037	-0.116	0.385	0.16	-0.253	0.06	-1.602	-

DDQN	BankHeist	Boxing	CrazyClimber	Enduro	FishingDerby	Gopher	IceHockey	Jamesbond	MsPacman	Pong
BankHeist	-	0.121	-0.378	-0.006	-0.107	0.042	-0.006	-0.058	0.001	-3.013
Boxing	-0.316	-	-0.104	-0	0.038	0.06	0.015	-0.225	-0.027	0.936
CrazyClimber	-0.192	-0.487	-	-0.012	-0.084	0.016	0.015	0.016	-0.015	-2.64
Enduro	-0.296	0.193	-0.167	-	0.039	0.03	0.019	-0.235	-0.039	0.248
FishingDerby	-0.212	-0.545	-1	-0.085	-	0.016	0.001	-0.055	-0.026	-0.935
Gopher	-0.466	0.044	-0.108	-0.005	0.007	-	-0.005	-0.094	-0.02	-1.816
IceHockey	-0.046	0.245	-0.067	0.014	-0.178	0.072	-	0.037	-0.015	0.112
Jamesbond	-0.145	0.232	-0.064	0.005	-0.267	0.031	-0.092	-	-0.006	-1.578
MsPacman	-0.173	-1.179	-0.129	-0.06	0.003	-0.019	0.007	0.071	-	-2.774
Pong	-0.127	0.028	-0.12	0.01	0.037	0.042	0.002	-0.174	-0.006	-

absent transfer, as can be observed by the area ratio scores obtained on `Enduro, Fishing Derby`, `Gopher` and `Ice Hockey` which are all ≈ 0 on average.

3 Control Experiments

The results presented in the previous section seem to be questioning the level of transferability of DRL agents. In fact, the training strategy of fine-tuning a pre-trained model on $\mathcal{M}_T$ does not result in the same type of performance gains that have been extensively observed in a supervised learning context. To better characterize their TL properties, we have designed a set of simple control experiments that allow us to examine their transfer learning behavior in training conditions that do not require extraordinarily long training times for learning an optimal policy.

3.1 The Catch Environments

To this end, we have implemented four different versions of the `Catch` game, a simple RL task that was first presented by Mnih et al. (2014), and that has been widely used within the literature for investigating the performance of DRL algorithms in a fast, and computationally less expensive manner than the one required by the `Atari` games (van de Wolfshaar et al., 2018; Aittahar et al., 2020). In the game of `Catch`, an agent controls a paddle at the bottom of the environment, represented by a 21×21 grid, and has to catch a ball falling from top to bottom, which can potentially bounce off walls. At each time step, the agent can choose between three actions: move the paddle one pixel to the right, move it to the left, or keep it in the same position in the grid. An RL episode ends either when the agent manages to catch the ball, in which case it receives a reward of 1, or when it misses the ball, which naturally results in a reward of 0. Following the design choices presented in (van de Wolfshaar, 2017), we model the ball to have vertical speed of $v_y = -1 \; cell/s$ and horizontal speed of $v_x \in \{-2, -1, 0, 1, 2\}$. From now on, we will refer to this version of the game as `Catch-v0`, as it is the most basic and simplest form of the game that will be used throughout our experiments. Next to `Catch-v0` we have implemented three slightly different and arguably more complex versions of the game as well: `Catch-v1`, where we increased the complexity of the game by reducing the size of the paddle that the agent controls. While for `Catch-v0` its size is of five pixels, in `Catch-v1` it is of two pixels, therefore requiring the agent to be more precise if it wants to successfully catch the falling ball. The second alternative version of `Catch` is `Catch-v2`. In this case, the dynamics of the game are identical to the ones that define `Catch-v0`; however, the way the 21×21 grid is represented changes. While in `Catch-v0` as well as in `Catch-v1` the state is represented by a binary grid where all pixels, but the ones representing the paddle and the ball have a value of 0, in `Catch-v2` the cells around the paddle and the ball can have a random value between 0 and 255. This design choice

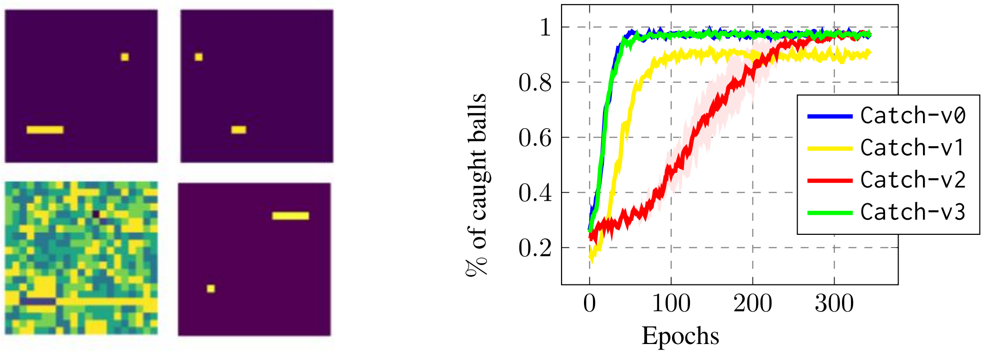

Figure 3: Image on the left: the four different versions of the Catch environment. In clockwise order: Catch-v0, Catch-v1, Catch-v3 and Catch-v2. Image on the right: learning curves obtained by a DQN agent that is trained from scratch on the aforementioned Catch versions. Shaded areas correspond to ±1 std. obtained over 5 different random seeds.

makes it much harder for a convolutional network to correctly locate and identify the position of the paddle and of the falling ball and makes Catch-v2 the arguably most complex version among the different Catch environments. Lastly, we have implemented Catch-v3, a version of Catch which is identical to the one that is modeled by Catch-v0 with the main difference that the representation of the state is now mirrored, therefore requiring the agent to look at different parts of the grid if it wants to locate the paddle, and understand that the ball is unnaturally moving from the bottom to the top. For an impression of all four Catch versions see the left image of Fig. 3. Given the overall simplicity of the different Catch environments, we now train a DQN agent instead of the arguably more complex DQV and DDQN agents that we considered in Sec. 2. As we can see from the results reported in the right plot of Fig. 3, averaged over five different runs, the agent is able to successfully learn a near optimal policy for all Catch versions. When it comes to Catch-v0, Catch-v2 and Catch-v3 we can observe that by the end of training, the agent is able to catch $\approx 100\%$ of the falling balls, whereas its performance is slightly worse ($\approx 90\%$) when it comes to Catch-v1 [1]. We can also observe that among the different Catch versions, Catch-v0 and Catch-v3 appear to be the easiest ones, as the agent requires significantly less training episodes to converge when compared to Catch-v1 and Catch-v2. Furthermore, in line with the explanation presented beforehand, our results also confirm the hypothesis that Catch-v2 is the overall most complicated Catch version, as learning requires significantly more, and potentially unstable, training epochs.

3.2 From one Catch to Another

We now replicate the TL study presented in Sec. 2 on the aforementioned Catch environments, with the hope of identifying why the process of fine-tuning a pre-trained convolutional neural network in a DRL context, seems to not be as beneficial as it is in the supervised learning one. Our goal is to find at least one pair of Catch environments which results in positive transfer, and to then potentially identify some properties within the different Catch versions that could also hold for the pairs of Atari games which have yielded positive transfer in Table 1. We formulate two hypothesis, based on which, we expect to experimentally observe positive transfer. First, we foresee that positive transfer will happen for all possible Catch combinations, as in the end the source MDP $\mathcal{M}_S$ and the target MDP $\mathcal{M}_T$ do not significantly differ from each other: in fact, the main task across different Catch versions remains that of catching a falling ball; the action space is identical; and so is the reward function that always returns a value of 1 when the agent succeeds in catching the falling ball. This hypothesis is also motivated by supervised learning research which shows that the higher the similarity between the source domain $\mathcal{D}_S$ and the target domain $\mathcal{D}_T$, the better the performance of a

[1] Please note that the performance on Catch-v1 can be improved by increasing the complexity of the DQN agent by adding one more convolutional layer. However, as the goal is to transfer the same type of model across Catch games, we did not modify the architecture of the DQN agent, at the cost of having a slightly worse performing model.

transferred pre-trained network (Sabatelli et al., 2018a). Second, in the case the previous hypothesis will not be empirically supported, we expect to at least observe positive transfer when using a model that comes as pre-trained either on `Catch-v1` or on `Catch-v2`. In fact, as described above and also shown by the performance reported in Fig. 3, these are the two most complicated versions of the `Catch` environment. In supervised learning, one of the main factors that makes a certain source task $\mathcal{T}_S$ good for transferring is its complexity (Mensink et al., 2021), which is usually defined in terms of dataset size and number of classes to classify, therefore we expect the complexity of the source game to play an important role within DRL as well. Similarly to what we did for DQV and DDQN in Sec. 2, we take the four different models that have been trained from scratch on their respective `Catch` version, randomly re-initialize their last layer (responsible for estimating the different state-action values $Q(s, a)$) and fully fine-tune the pre-trained network on the three remaining `Catch` environments.

Table 2: The area ratio obtained after fine-tuning a pre-trained DQN agent on the different `Catch` environments. We can see that no matter which source game is used for pre-training, transfer learning surprisingly never results in positive transfer.

	Catch-v0	Catch-v2	Catch-v3	Catch-v4
Catch-v0	-	-0.026	-0.486	-0.479
Catch-v2	-0.16	-	-0.121	-0.248
Catch-v3	-0.406	-0.313	-	-0.465
Catch-v4	-0.016	-0.24	-0.179	-

The results of this study are reported in Fig. 4, where, from left to right, we show the performance that is obtained when considering `Catch-v0`, `Catch-v1`, `Catch-v2` and `Catch-v3` as target MDP $\mathcal{M}_T$. The performance of each transferred network is compared against the performance that is obtained after training a DQN agent from scratch which matches with the results reported in Fig. 3. Surprisingly we found that fine-tuning a pre-trained DQN agent never resulted in positive transfer learning. This can clearly be seen in all plots represented in Fig. 4 and by the results reporting the area ratio metric in Table 2. The only case where starting from a pre-trained network appeared to be at least in part beneficial is represented by the first plot of Fig. 4 when `Catch-v1` is considered as source MDP $\mathcal{M}_S$. In this case we can in fact observe some learning speed improvements within the first 25 learning epochs. This is not surprising as an agent which is able to catch a ball with a small paddle (as defined by the game `Catch-v1`) should in principle also be able to do this when the size of its paddle is larger (which is the case for `Catch-v0`). What is more surprising, however, is that while training progresses we see that the performance of a `Catch-v1` pre-trained model starts deteriorating and that this model barely converges to the same performance that is obtained by a model trained from scratch. When it comes to all other $\mathcal{M}_S/\mathcal{M}_T$ pairs we see that pre-trained networks always perform significantly worse than randomly initialized models trained from scratch, with some extreme cases, as the one reported in the last plot of Fig. 4, where a `Catch-v0` pre-trained agent is barely able to improve its policy over time at all. These results invalidate our two hypotheses mentioned above as they clearly show that positive transfer in DRL does not arise when $\mathcal{M}_S$ and $\mathcal{M}_T$ are similar, nor when $\mathcal{M}_S$ is more complex than $\mathcal{M}_T$.

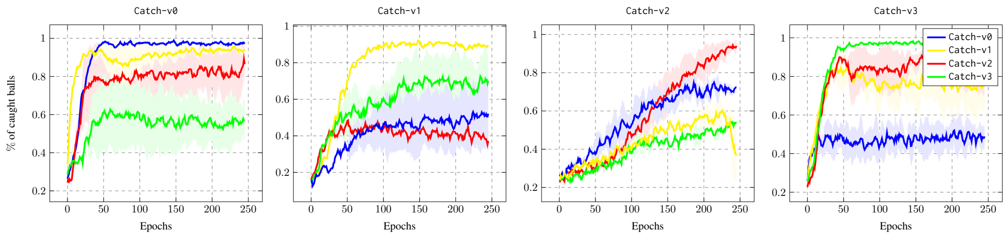

Figure 4: The results obtained after using a pre-trained `Catch` agent and fine-tuning it on a different `Catch` version. We can observe that despite all `Catch` versions being very similar no positive transfer is ever observed, as a model trained from scratch always outperforms a pre-trained, fine-tuned network.

3.3 Self-Transfer

The surprisingly poor transfer learning performance observed in the previous experiment made us question the level of transferability of Deep-Q Networks even more. To further characterize their TL properties, we decided to investigate whether pre-trained DRL agents are at least able to transfer to themselves. To this end, we studied what happens when a DQN agent gets transferred to a version of Catch that matches with the version of the game that was also used during the pre-training stage. This experiment is in large part identical to the one presented in Sec. 3.2, with the only difference being that now $\mathcal{M}_S = \mathcal{M}_T$. Moreover, differently from the previous study, we now also investigate what happens if instead of fine-tuning the network completely, we just use the pre-trained DQN agent as a simple feature extractor, therefore only training its last layer (the head) responsible for estimating the different state-action values. Our results are presented in Fig. 5, where for each Catch version, the full lines represent the performance of a network that is trained from scratch, whereas the dashed and dotted lines respectively report the performance that is obtained when the network is either used as simple feature extractor or entirely fine-tuned. We can see that if a pre-trained Deep-Q Network is used as a simple feature extractor, the agent can converge to the optimal policy almost immediately. In fact, as can consistently be observed in all plots of Fig. 5 and from the results presented in Table 3, training only the last layer of a pre-trained network yields positive transfer for all different Catch versions. However, when a fine-tuning training strategy is adopted, much more surprising results have been obtained. First, and more importantly, we can see that despite all models showing some learning speed improvements at early training iterations, their final performance is never on par with the one that is obtained when the same kind of model is either used as a feature extractor or trained from scratch (the dotted lines are consistently below the dashed and full lines). While when it comes to Catch-v0 the policy learned by a fine-tuned model still allows the agent to successfully catch $\approx 95\%$ of the falling balls, the same cannot be said when the models are tested on Catch-v1, Catch-v2 and Catch-v3, where the difference in terms of performance between a model trained from scratch and a fine-tuned one is much more significant. Please also note that special attention should be given to Catch-v2, which is an environment where the area ratio score reported in Table 3 can be misleading as it does not entirely reflect the quality of the final policy learned by the agent. In fact, while it is true that an r value of 0.393 is obtained, it is worth noting that a fine-tuned network converges to a policy that is significantly worse than the one of a network trained from scratch, as the agent is only able of catching $\approx 80\%$ of the falling balls. Second, as highlighted by the large variance across different training runs, fine-tuning on Catch-v1 and Catch-v3 resulted in highly unstable learning as well.

Table 3: The area ratio scores obtained after performing self-transfer. We can see that if only the last linear layer is trained, then positive transfer is obtained on all Catch environments, whereas if the network is fine-tuned, positive transfer is (in part) only obtained on Catch-v2.

	Catch-v0	Catch-v1	Catch-v2	Catch-v3
Only-Head	0.05	0.141	0.674	0.059
Fine-Tuning	0.017	-0.218	0.393	-0.236

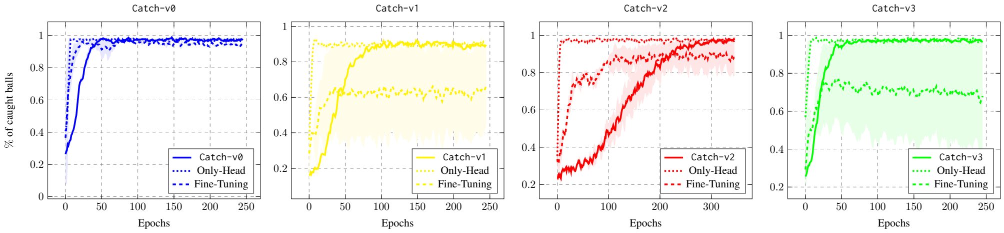

Figure 5: The results of our self-transfer experiments. From left to right the performance obtained on Catchv-0, Catch-v1, Catch-v2 and Catch-v3 after either training only the last linear layer of a pre-trained Deep-Q Network (dotted lines), or after wholly fine-tuning the model (dashed lines). We can see that the former transfer learning strategy yields significantly better results, and that a fine-tuning approach results in networks that in three cases out of four are not even able to transfer to themselves.

4 The Two Learning Phases of Deep-Q Networks

A prototypical Deep-Q Network takes as input an image representing the state of the environment and processes it through a series of convolutions and a fully connected layer. When this is done, it outputs as many $Q(s,a)$ values as there are actions available to the agent, a process that corresponds to learning a linear policy in the latent feature space of the network. It follows that by the end of training, such a model has to serve two purposes: it has to perform as a feature extractor as well as an optimal value function approximator. Extracting relevant features from high dimensional inputs and learning an optimal value function can arguably be seen as two separate tasks; yet, despite their dissimilarity, we believe that they are more interconnected than one might expect. Specifically, we hypothesize that while learning, a Deep-Q Network has to carefully find a balance between training the parts that serve as feature extractors and the components that are responsible for estimating a policy. The poor TL performance observed throughout this work could therefore be the result of using models where the feature extractor component of an agent, as it comes as pre-trained, is too detached from the respective final layer of the network, which is randomly initialized instead. To show that a Deep-Q Network has to carefully coordinate training its feature extractor components and its final layer, let us consider the left image of Fig. 6. The figure depicts how the weights of an agent, whose feature extractor layers are represented by a square and the linear layer is represented through circles, change according to the self-TL experiments presented in Sec. 3.3. Each experiment is represented through two networks, one on the left side of the arrow representing the source model, and a second one, on the right part of the arrow, obtained by the end of training. From top to bottom, and from left to right, the first three network pairs represent the training process of: a randomly initialized model trained from scratch, a pre-trained model whose only last linear layer is trained after random initialization, and a pre-trained agent who gets fully fine-tuned. Following the results presented in Sec. 3.3, we know that positive transfer is only obtained when the last linear layer is trained in isolation after being randomly re-initialized, whereas negative transfer is obtained if a fine-tuning training strategy is adopted.

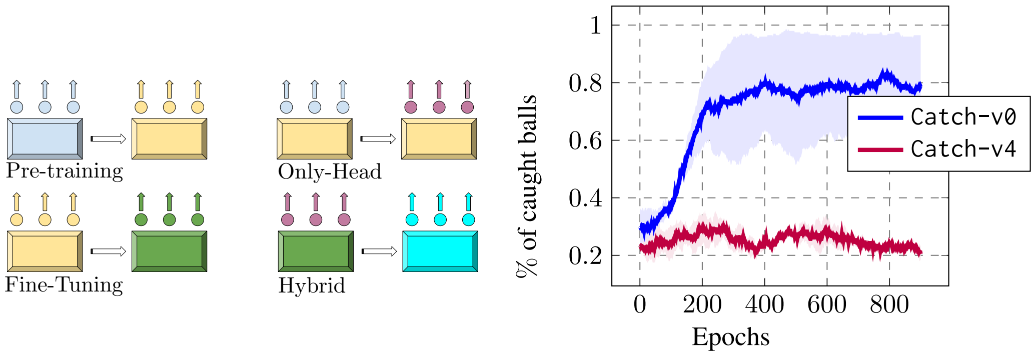

Figure 6: Image on the left: a visualization of differently initialized Deep-Q Networks before and after training. Image on the right: a successful example of positive transfer.

We now investigate the TL performance of a model that is a combination of the Only-Head and Fine-Tuning settings (see bottom right image of Fig. 6 for a visualization). Specifically, we fine-tune a Deep-Q Network whose last linear layer is initialized with the parameters that yielded positive transfer in Sec. 3.3 (Only-Head in Fig. 6), whereas its convolutional and fully connected layers are initialized with the parameters that yielded negative transfer (Fine-Tuning in Fig. 6). We visualize the self-transfer performance of these models, denoted as ‘Hybrid”, as they are a hybrid combination of two differently pre-trained networks, in Fig. 7 with a cyan dashed dotted line. We can observe that learning is characterized by a very atypical behavior: the network starts by improving its performance (thanks to the already trained final layer); it then goes through a second stage where it starts to perform more poorly (due to the poor feature extractor part), and then finally starts learning stably (when the feature extractor and the head of the model are synchronized). We believe that the poor TL performance observed throughout this work is, therefore, the result of models which could not find a balance between a randomly initialized head and their respective pre-trained layers which are too biased towards the source task.

Based on these results, one critical question still remains to be answered: how come positive transfer for some Atari games was observed in Sec. 2?. We believe that the answer to this question does not lie within the feature representations that a Deep-Q Network learns, but rather in some inner properties of the environment that is used as target task and that favors a Deep-Q Network to correctly synchronize its components. As a proof of concept, we have created one final `Catch` environment, called `Catch-v4`, which is identical to `Catch-v0` with the only difference being that a positive reward is returned to the agent only if it manages to catch five falling balls in a row. We can see from the right image of Fig. 6 that a model trained from scratch (represented by the purple line) is not able to improve its policy over time at all, as the reward signal is probably too sparse for learning, whereas a `Catch-v0` model is now able to yield positive transfer. We hence believe that some environments, if combined with certain learning algorithms, are more prone to positive transfer than others, as was, e.g., the case for `Fishing Derby` and DQV-Learning where positive transfer was observed no matter what source task was used for pre-training. This is not due to the representations that are learned by a pre-trained network but rather because of some specific dynamics within the target MDP $\mathcal{M}_T$.

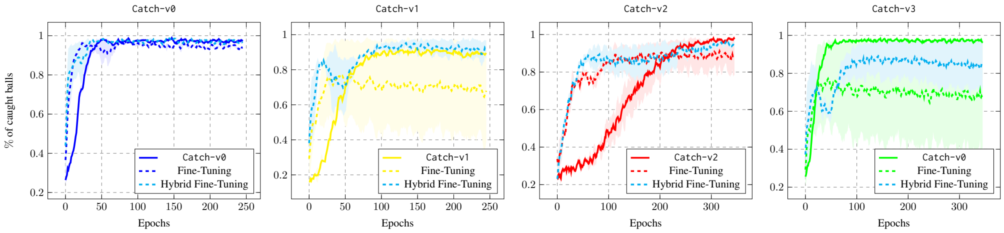

Figure 7: The performance (in cyan) of a fine-tuned pre-trained network whose last layer is initialized with parameters that yielded positive transfer, whereas its convolutional and fully conntected layers are initialized with parameters that yielded negative transfer.

5 Related Work & Conclusion

The closest research to the one presented in this work is certainly that of Farebrother et al. (2018) and Tyo & Lipton (2020), who also studied the generalization properties of Deep-Q Networks in a model-free DRL context. Our extensive experiments confirm some of the preliminary claims that were made by the former about the potentially poor TL properties of Deep-Q Networks, but contradict the study of the latter, who suggested that fine-tuning DRL agents results in positive transfer when moving from a simpler task to a harder. While both works are certainly valuable, we also believe that their experimental results are not as thorough and on par with the ones of this study as they both considered a very limited number of RL problems (four and three respectively), therefore leaving the question *"How transferable are Deep-Q Networks?"* unanswered. We believe that the answer to it is *"barely"*, but we also believe that their poor TL properties, as well as the learning dynamics identified in Sec. 4, are inherent to this family of algorithms only. In fact, it is worth noting that several works describing the benefits of TL in RL do exist (but they all differ from the study presented in this work): Tirinzoni et al. (2018) show that it possible to successfully transfer value functions across tasks, yet their work does not consider deep networks as function approximators but rather Gaussian mixtures. Parisotto et al. (2015) show that it can be beneficial to fine-tune a pre-trained DRL agent, but they consider multi-task learning and policy gradient algorithms as a way of pre-training. Rusu et al. (2016) also show that fine-tuning can be beneficial, but in the context of progressive networks and again of policy gradient techniques. Similar conclusions for Actor-Critic algorithms can also be found in the works of Zhu et al. (2017) and Chen et al. (2021). Furthermore, Landolfi et al. (2019) and Sasso et al. (2021) show that fine-tuning a pre-trained network can be beneficial for DRL tasks, but for model-based RL approaches, which are again part of a family of techniques that is different from the ones analyzed in this work. To conclude, we would like to stress out that despite the overall poor TL performance observed throughout this paper, positive transfer can nevertheless be obtained in a model-free DRL setup, and hope that this paper can serve as a solid starting point for the DRL community which is interested in designing general and transferable agents.

REFERENCES

Samy Aittahar, Raphaël Fonteneau, and Damien Ernst. Empirical analysis of policy gradient algorithms where starting states are sampled accordingly to most frequently visited states. *IFAC-PapersOnLine*, 53(2):8097–8104, 2020.

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

Lili Chen, Kimin Lee, Aravind Srinivas, and Pieter Abbeel. Improving computational efficiency in visual reinforcement learning via stored embeddings. *arXiv preprint arXiv:2103.02886*, 2021.

H Domínguez Sánchez, M Huertas-Company, M Bernardi, S Kaviraj, JL Fischer, TMC Abbott, FB Abdalla, J Annis, S Avila, D Brooks, et al. Transfer learning for galaxy morphology from one survey to another. *Monthly Notices of the Royal Astronomical Society*, 484(1):93–100, 2019.

Jesse Farebrother, Marlos C Machado, and Michael Bowling. Generalization and regularization in dqn. *arXiv preprint arXiv:1810.00123*, 2018.

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

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

David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

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

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019a.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pp. 2555–2565. PMLR, 2019b.

Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020.

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

Namgyu Ho and Yoon-Chul Kim. Evaluation of transfer learning in deep convolutional neural network models for cardiac short axis slice classification. *Scientific reports*, 11(1):1–11, 2021.

Minyoung Huh, Pulkit Agrawal, and Alexei A Efros. What makes imagenet good for transfer learning? *arXiv preprint arXiv:1608.08614*, 2016.

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

Nicholas C Landolfi, Garrett Thomas, and Tengyu Ma. A model-based approach for sample-efficient multi-task reinforcement learning. *arXiv preprint arXiv:1907.04964*, 2019.

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

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

Thomas Mensink, Jasper Uijlings, Alina Kuznetsova, Michael Gygli, and Vittorio Ferrari. Factors of influence for transfer learning across diverse appearance domains and task types. *arXiv preprint arXiv:2103.13318*, 2021.

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

Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. In *Advances in neural information processing systems*, pp. 2204–2212, 2014.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

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

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

Johan S Obando-Ceron and Pablo Samuel Castro. Revisiting rainbow: Promoting more insightful and inclusive deep reinforcement learning research. *arXiv preprint arXiv:2011.14826*, 2020.

Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

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

Martin L Puterman. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434, 1990.

Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

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

Matthia Sabatelli, Gilles Louppe, Pierre Geurts, and Marco Wiering. Deep quality-value (dqv) learning. In *Advances in Neural Information Processing Systems, Deep Reinforcement Learning Workshop*. Montreal, 2018b.

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)*, pp. 1–8. IEEE, 2020.

Remo Sasso, Matthia Sabatelli, and Marco A Wiering. Fractional transfer learning for deep model-based reinforcement learning. *arXiv preprint arXiv:2108.06526*, 2021.

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

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015b.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 806–813, 2014.

Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(7), 2009.

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

Jacob Tyo and Zachary Lipton. How transferable are the representations learned by deep q agents? *arXiv preprint arXiv:2002.10021*, 2020.

Jos van de Wolfshaar. *Deep Reinforcement Learnig of Video Games*. PhD thesis, Faculty of Science and Engineering, 2017.

Jos van de Wolfshaar, Marco A Wiering, and Lambert Schomaker. Deep learning policy quantization. 2018.

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

Remy Vandaele, Sarah L Dance, and Varun Ojha. Deep learning for automated river-level monitoring through river-camera images: an approach based on water segmentation and transfer learning. *Hydrology and Earth System Sciences*, 25(8):4435–4453, 2021.

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

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International Conference on Machine Learning*, pp. 1995–2003, 2016b.

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

Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *2017 IEEE international conference on robotics and automation (ICRA)*, pp. 3357–3364. IEEE, 2017.

Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1): 43–76, 2020.

A Appendix

We hereafter provide additional information about how the model-free DRL algorithms considered throughout this paper were trained. In Table 4 and 5, we provide the hyper-parameters that were used for the large scale empirical study presented in Sec. 2, while we then describe the DQN architecture used for the control experiments presented in Sec. 3.

DQN: The agent comes in the form a two-hidden layer convolutional neural network which is followed by a fully connected layer of 256 hidden units preceding the final linear layer responsible for estimating the different $Q(s, a)$ values. The first convolutional layer has 32 channels whereas the second one has 64 channels. All layers of the network are activated by a ReLU non-linearity. We use an experience replay memory buffer that is set to contain 400000 trajectories, and start training the model as soon as 5000 trajectories have been collected. For exploration purposes, we adopt the

Table 4: Hyper-parameters used when training a DDQN agent from scratch. We follow the experimental setup introduced in the original paper Van Hasselt et al. (2016).

Hyper-parameter	
Atari Arcade Learning Version	`Deterministic-v4`
Frame-Skipping	True
Reward Clipping	$[-1, 1]$
Epsilon Greedy ϵ	0.1
Discount Factor γ	0.99
Pre-processing scheme	$84 \times 84 \times 4$
Q-optimizer	RMSprop
Q Learning rate	0.00025
Optimizer ρ	0.95
Optimizer ϵ	0.01
Memory size S	1M trajectories

Table 5: Hyper-parameters used when training a DQV agent from scratch. We can see that they mostly correspond to the ones presented in Table 4 with the main difference being the epsilon greedy parameter ϵ that is set to 0.5 instead of 1.0, and the additional information given for the V network which is trained for learning an approximation of the state-value function.

Hyper-parameter	
Atari Arcade Learning Version	`Deterministic-v4`
Frame-Skipping	True
Reward Clipping	$[-1, 1]$
Epsilon Greedy ϵ	0.5
Discount Factor γ	0.99
Pre-processing scheme	$84 \times 84 \times 4$
Q-optimizer	RMSprop
Q Learning rate	0.00025
V-optimizer	SGD
V Learning rate	0.001
Optimizer ρ	0.95
Optimizer ϵ	0.01
Memory size S	1M trajectories

popular epsilon-greedy strategy with an initial ϵ value of 1 which gets linearly annealed over time to 0.1. Learning a near optimal policy on the aforementioned `Catch` environments with this type of DQN agent can require between the 3 and the 5 hours of training time.