# Playing and Experimenting on Atari Games with Deep Reinforcement Learning

(Dated: January 23, 2021)

In this paper we aim to examine the Atari games Gym environments and winning them using Deep Q-Network (DQN) as a policy, as well as transferring knowledge to win new games faster. So far, we have experimentally confirmed that, without change of hyperparameters, DQN agents are successfully learning to win various games. However, our experiments with transfer learning show that knowledge from previous games increases the training time for new ones.

## I. INTRODUCTION

### A. Related Work

OpenAI Gym's environments are widely used for development and testing of new reinforcement learning algorithms. There have been numerous breakthroughs in the field that have been demonstrated using Gym, one of the most notable being DQN (Mnih et al.) [1]. It introduces the first usage of a deep convolutional neural network as a policy in a reinforcement learning problem and its algorithm has later been adopted for various problems.

Recently, Deepmind's Agent57 [2] managed to beat the human benchmark on all Atari games. It is the first model to generalize a solution for all games at the same time.

Asawa et al. [3] have documented an approach to apply transfer learning methods and investigate whether a trained DQN reduces the training speed of a new game when compared to training from scratch. Their results show that this is not the case. Even worse, using the parameters of an already trained network decreases the training time of the next agent. We aim to replicate their results and improve them.

### B. Problem Description

Teaching an agent to play by itself means that it needs to learn how to react to environment changes, which happens over lots of iterations. In order to do that, most approaches examine the visual data (observation) of the environment and based on that they aim to make the best conclusions as to what needs to happen next. Training with visual input is also what we base our experiments on.

In this context one observation in an Atari Environment consists of one frame of the game, or a 210x160x3 RGB image. A selected action is applied for a couple of consecutive frames (often 4). In Mnih et al., a state is defined to be a tuple $(s_t, a_t, r_t, s_{t+1})$, where:

- $s_t$ is the current observation;
- $a_t$ is the action we have taken based on the observation according to our policy;
- $r_t$ is the reward we have received after taking this action;
- $s_{t+1}$ is the next received observation.

These states are gathered throughout the training and they represent the training data for the Q-network. It is trained to predict the expected reward if a certain action is taken. Hence, we define our loss function as $Loss(\operatorname{argmax}_a Q(s_t, a), r_t + \gamma * Q(s_t, a_t))$ where $Q$ is our action-value function (or just Q-function) and $\gamma$ is a discount factor. $Loss$ is MSE in the DQN paper; in our implementation we use Huber loss. The DQN is described in more details in Mnih et al. [1].

We wanted to find out if there is a way to achieve good training results in less time. In order to achieve this we used a transfer learning approach which is proven to reduce training times in other deep learning problems [4], [5]. Our goal is to apply the transfer learning approach to an already trained DQN agent for a different game and then compare the training time of that agent with the training times of another agent which had been trained on the same game from scratch. We aim to apply transfer learning across games that look similar in their logic. With that, we expect the first agents' networks to learn higher level features that are applicable in the next games as well. Two such pairs of games for transfer learning are $Breakout \rightarrow Pong, Pacman \rightarrow Asterix$.

## II. METHODS

In order to do our experiments, we went with a Deep Q-Learning algorithm with experience replay. The way this algorithm works is it uses a memory buffer to store past transitions and a DQN which computes the necessary Qvalues. The algorithm steps are as follows:

```
Algorithm 1: DQN Algorithm
Initialize the memory buffer $M$ to capacity $pretrain\_length$;
Initialize DQN with random weights;
for $iteration \leftarrow 0$ to $N$ do
    Obtain initial state $s_t$;
    $done \leftarrow False$;
    while not done do
        With a probability $\epsilon$ select random action $a_t$;
        Otherwise $a_t \leftarrow \max_a Q(s,a)$;
        Execute action $a_t$ and observe reward $r_t$, next state $s_{t+1}$ and $done_t$;
        Store transition $(s_t, a_t, r_t, s_{t+1})$;
        $done \leftarrow done_t$;
        $s_t \leftarrow s_{t+1}$;
        Use batch from memory buffer to train DQN;
    end
end
```

The architecture for the DQN we chose consists of 2 convolutional layers one layer that flattens the convolution output, one fully connected layer and an output layer. The input of the DQN is either a single state or a batch of states, where the state is the aforementioned tuple $(s_t, a_t, r_t, s_{t+1})$. The observations $s_t$ and $st + q$ in the state consist of 4 stacked consecutive frames from the environment.

During execution of the DQN algorithm, we keep two copies of the Q-Network - *policy* and *target*. We use *target* to predict the target expected reward when calculating the loss function. The *policy* network is the one that is being updated and every $k$ iterations *target* and *policy* are synced. $k$ is a hyperparameter.

Using the algorithm and DQN architecture mentioned above, we approached transfer learning the following way:

- We trained our agent on game $X$ for $N$ iterations
- Simultaneously we trained a second agent on game $Y$ for $N$ iterations as well
- Once the first agent had finished training on game $X$ we started training it on game $Y$ immediately, using the already stored weights from the previous training

We did not change the DQN architecture and we did not reset any of the weights in the DQN for the transfer learning. Because of this, the games on which we experimented were chosen in such a way that they had the same action space, so that the network could be applied to both games. For the experiment we chose 2 pairs of games with the same action spaces.

## III. RESULTS

Due to limited time and resources, we could not afford to do enough training to achieve competitive scores. We trained long enough so that a steady and obvious increase of average reward is observable.

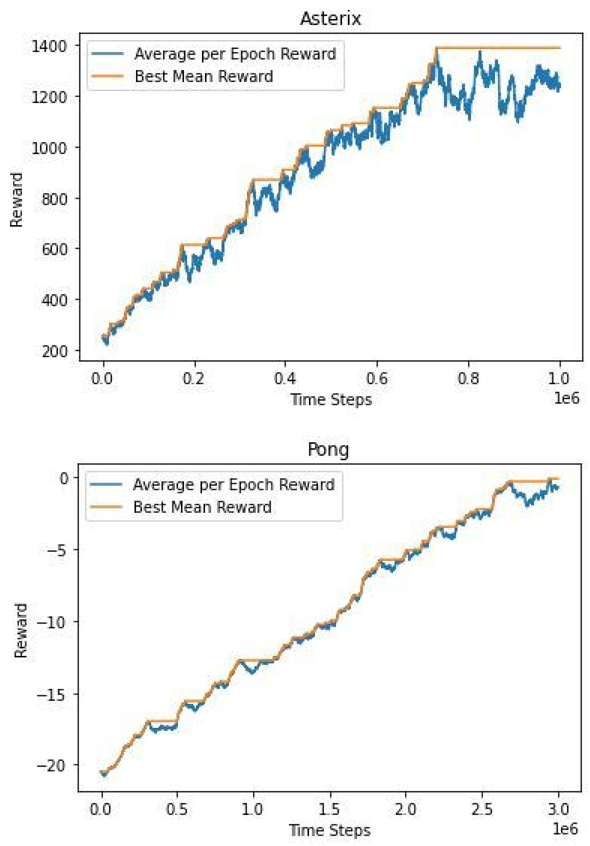


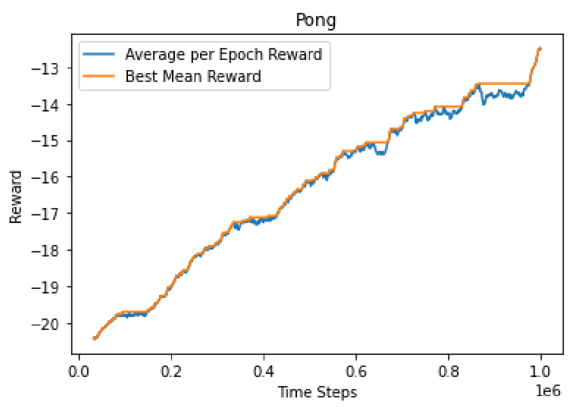
FIG. 1: Trainings from scratch on Asterix and Pong. Both agents improve their average reward over time.

We trained two agents to play respectively Asterix and Pong from scratch (see Fig. 1). As noted in Mnih et al., the DQN approach improves the agent's performance over time. We saved these agents' performances to compare them with the agents we produce after we apply reinforcement learning.

Parallel to that, on different machines we trained agents to play Pacman and Breakout and then train the same agents on respectively Asterix and Pong. When compared to the agents that we trained from scratch, these achieved lower average reward for the same training time (see Fig. 2).

## IV. CONCLUSION

In this paper we replicated the results from the original DQN paper [1] and received satisfying results over relatively short training times. On top of that, we tried to accelerate the training of future agents by using already trained Q-Networks. Contrary to our expectations and similar to the outcome of Asawa et al. [3], we did not observe a decrease of training times. Even worse, the



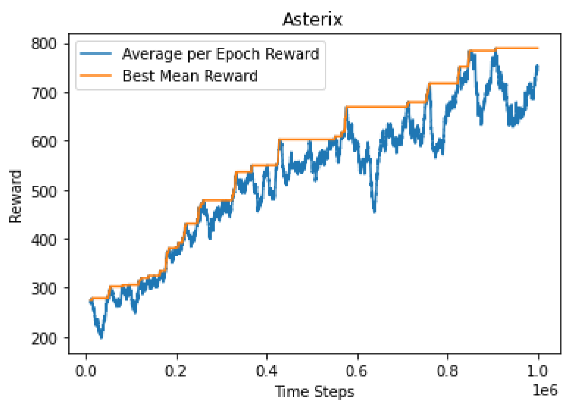


FIG. 2: Agents trained on Pong and Asterix using transfer learning for 1M iterations. The Q-Network was trained on respectively Breakout and Pacman beforehand. Compare with Fig. 1 to see that, at the same timestamps, the freshly trained agents achieved higher rewards.

agents improved their average rewards slower than the ones trained from scratch.

## V. DISCUSSION AND FUTURE STEPS

We expected that the Q-Network would learn generalized and transferable features that could aid the training of future agents. Our hypothesis why this did not happen is that the architecture we used does not contain enough trainable parameters to generalize and learn high level features. Indeed, Badia et al. [2] use a much deeper neural architecture and also have implemented techniques for ”guiding” the network into generalization. However, in our case we aim to build a training transition between only two games. For this goal, we believe increasing the trainable parameters could improve our results. Thus it is our first planned step in the future. Furthermore, other transfer learning methods reinitialize the weights before the output layer since they can be very specific to the previous problem the Q-Network has been used to solve [3].

## VI. ACKNOWLEGEMENTS

The source code of the project is available on GitHub: `https://github.com/rl-game-training/multi-game-agent`.

Most of the development and training has been done on Google Colab. We would like to acknowledge three sources that aided us in implementing DQN:

- Dr. Marin Bukov’s implementation of DQN;
- DQN tutorial in Pytorch’s website: `https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html`;
- David Reiman’s implementation of DQN: `https://github.com/davidreiman/pytorch-atari-dqn`.

We would also like to thank Viktor Velev (`https://github.com/VikVelev`) for letting us train our models on his GPU overnight.

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” (2013), arXiv:1312.5602 [cs.LG] .

[2] A. P. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, D. Guo, and C. Blundell, “Agent57: Outperforming the atari human benchmark,” (2020), arXiv:2003.13350 [cs.LG] .

[3] C. Asawa, C. Elamri, and D. Pan, “Using transfer learning between games to improve deep reinforcement learning performance and stability,” .

[4] H.-W. Ng, V. D. Nguyen, V. Vonikakis, and S. Winkler, in *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction*, ICMI ’15 (Association for Computing Machinery, New York, NY, USA, 2015) p. 443–449.

[5] H. Shin, H. R. Roth, M. Gao, L. Lu, Z. Xu, I. Nogues, J. Yao, D. Mollura, and R. M. Summers, IEEE Transactions on Medical Imaging **35**, 1285 (2016).