Learning to play Pong with DQN

SAM_1RT747, 1RT745 – Group project

Ivar Blohm, Oskar Åsbrink, Tobias Arnehall Johansson
Group 13

May 25, 2022

1 Introduction

This report will demonstrate how Deep-Q-Networks (DQN) can be applied to solve two Reinforcement Learning (RL) problems. Results from experiements with hyperparameters will be presented to show new insights.

The first scenario is the reputed Cartpole problem, i.e. the Inverted Pendulum problem, where the task is to learn an agent to balance a cartpole by taking appropriate actions at certain states [1]. The second scenario is to learn an agent to play the Atari game Pong [2], meaning that the agent should know when to take either action up or down in order to hit the ball, with the goal of winning over its opponent.

Even though the scenarios seems to be different, both in terms of how the tasks are designed, as well as the degree of difficulty, they are somewhat similar. They only differ slightly in some of the technical aspects, such as the reward system and state space. To set up and control the environment in both scenarios we used OpenAi Gym, which is a standard API for Reinforcement learning [3].

2 DQN

In many cases when developing reinforcement learning methods, usually the state- and action space becomes too large and can therefore not be handled by using tabular methods. It has been successfully proven that using a DQN could manage those problems to some extent [4, 5]. The idea behind DQN is to use a deep neural network to approximate the state value function for each action in a given state. The neural network could consist of both convolutional- and dense layers, depending on the application. They can be used in combination to solve, for example, the Pong scenario, since it is a common approach when analyzing images.

Usually, a DQN is constructed in conjunction with a replay memory, which is being used to draw random samples from earlier experience. By using this, it improves the learning, since it minimizes the dependency between the samples (when drawn randomly).[6]

Two similar DQN's can be used when optimizing a model where one is used as a "target" network, while the other one is the actual "training" network. The use of having a target network is to have it "frozen" so that the training network could optimize its weights by trying to minimize the loss (w.r.t the target network). In order for this to work, the target network is periodically updated with the weights from the training network during the learning process.[6]

3 Results

3.1 CartPole

The results from the Cartpole problem is dependent of the initialization of the hyperparameters, one needs to consider their values in order to search for a better model. After studying the results with different ways of decreasing epsilon it shows a major impact for convergence. This could therefore be seen as a fundamental aspect regarding the exploration vs exploitation trade-off. In this case, ϵ_t-greedy was used with a starting value of 1.00 which was decreased over time to the end value 0.05. The major focus was how to decay epsilon, where in theory it is possible to construct a schedule ϵ_t that asymptotically gives logarithmic total regret when ϵ_t decreases over time t. The following two equations were analyzed

$$\epsilon_t = \epsilon_{\text{start}} + \frac{t}{\text{anneal length}} \cdot (\epsilon_{\text{end}} - \epsilon_{\text{start}}) \tag{1}$$

$$\epsilon_t = \epsilon_{\text{end}} + (\epsilon_{\text{start}} - \epsilon_{\text{end}}) \cdot e^{\frac{-t}{\text{decay}}}. \tag{2}$$

In both equations t represents each step taken and the variable *anneal length/decay* determines how fast epsilon will reach its end value. Since the first equation decreases epsilon linearly. Many different values for *anneal length/decay* were tested for both equations, where a few of them are shown in Figure 1.

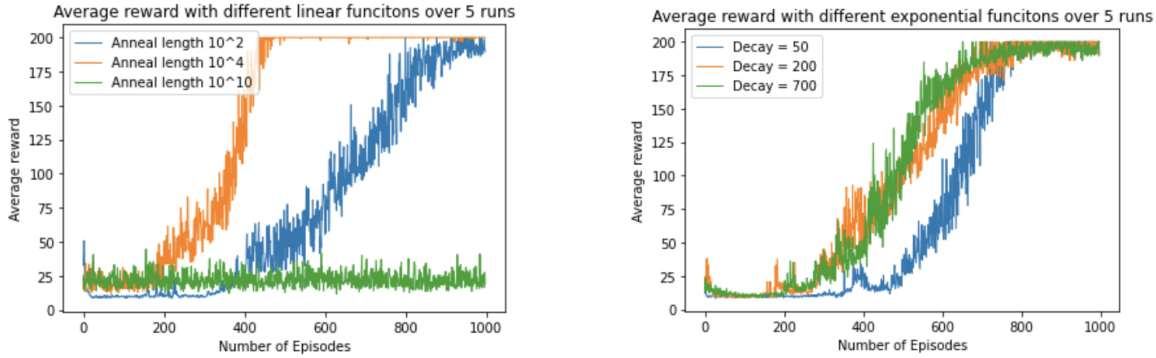

Figure 1: *Left:* Linearly decreasing epsilon (eq. (1)). *Right:* Decreasing epsilon exponential (eq. (2)).

When observing Figure 1 it is necessary to mention that these just experiments of how parameter values affect average reward. From both plots (especially the left plot) in Figure 1 it is certain that epsilon has a big impact on the exploration vs exploitation trade-off. A sufficient result was found with linearly decreasing epsilon and with *anneal length* $= 10^4$.

Experiments of the impact of different values of gamma and memory size is shown in Figure 2.

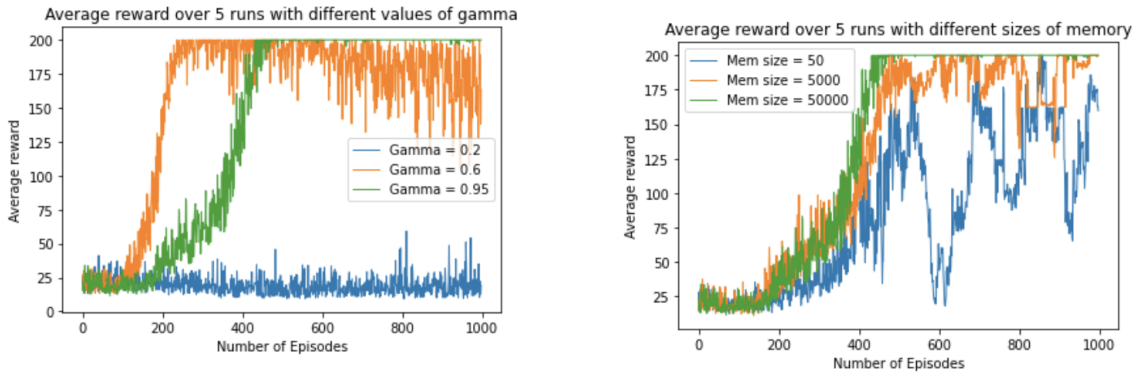

Figure 2: *Left:* Three different values of gamma. *Right:* Three diferent sizes of memory.

Note that all runs in Figure 2 is with decreasing epsilon linearly and with *anneal length* $= 10^4$, since it was proven to provide a good result in Figure 1.

The discount factor *Gamma* (γ) is the variable that is included in the target. Since Q-learning is used, the target is given as

$$G_t = R + \gamma max_a Q(S', a).$$

In other words, gamma controls the impact of the expected maximum state-action value. When it approaches zero, the target is merely defined by the reward. In Figure 2 (left plot), it is shown that gamma close to one produces a more stable convergence.

The plot on the right-hand-side of Figure 2 indicates the impact of the memory size. This plot is included in the report, since it was a new idea for us. The (replay) memory saves the last N experiences and the purpose is to provide a batch when performing updates of the DQN. In this project a batch size of 32 is used, and it's clear that a larger memory size improves the result. Since the batch is uniformly sampled from memory, this clarifies that better results were achieved when there is less dependency between the samples.

3.2 Pong

In the process of optimizing the Pong-agent, the following hyperparameters were used:

Observation stack size: 4
Replay memory capacity: 10000
Batch size: 32
Target update frequency: 1000
Training frequency: 4
Discount factor: 0.99
Learning rate: 1e-4
Initial epsilon: 1.0
Final epsilon: 0.01
Anneal length: 10^6

Epsilon was decreased with a linear function since it was proven to be successful in CartPole. Worth to mention as well was that only two of the original six actions are needed for training the agent, signaling up and down actions.

In Figure 3 we can observe the average reward per 25 episodes. It is apparent that the agent is learning over time, where approximately in episode 625 it starts to beat the opponent. This can be due to the fact that the agent is mostly taking random actions initially. There is however a major drop in episode 800, but this could be considered as random since the reward increases significantly in the following episodes.

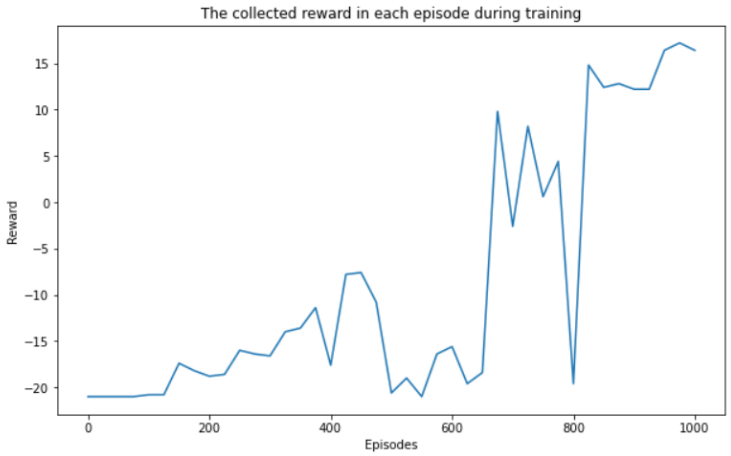

Figure 3: The collected reward in each episode during training.

When observing Figure 3, it is possible to see that the highest average reward (17.2) was achieved in episodes 950-975. Screenshots from the gameplay with the best performing model is visualized in Figure 4.

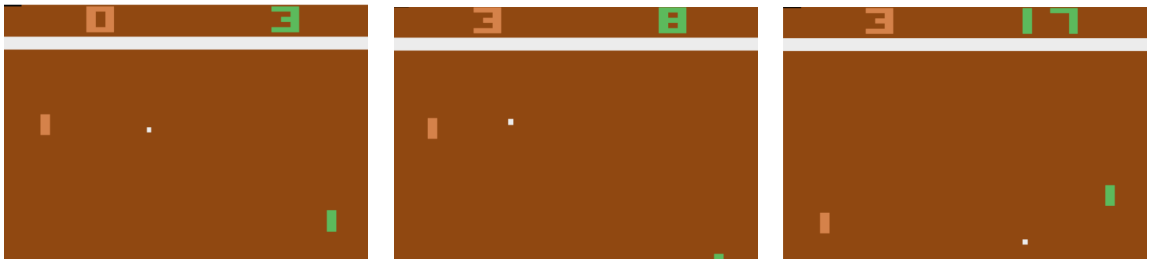

Figure 4: One example run when evaluating the best performing model. The reward in this case was 16, since it eventually won with 21-5.

In this case the agent did win with 21-5, giving a reward of +16, which is a decent result for an agent that has been trained 1000 episodes, without manipulating any hyperparameters.

4 Discussion

This report shows that one can learn an agent navigate two different problems with two similar implementations of a DQN. In terms of the Cartpole problem, an average reward of >195 over 100 consecutive trials is the definition of solving the problem according to the website for the OpenAI-CartPole [1]. This was achieved in this project. Second, a DQN was used to learn an agent to play Pong, where it was able to beat a pre-trained opponent. This indicates that the agent has learned how to play the game, even though the performance is not up to par with OpenAI [5]. For better future performance, one would have to experiment more with the hyperparameters, starting with increasing the memory size, use exponential decay, and/or increase the number of episodes.

Due to troubles and error in the implementation process, there was not enough time to carry out these experiments on the Pong environment. Some parts of the code was hard to properly implement and debug, and it was difficult to be sure if and when the code was working, since progress could not be shown until several hours of training. Google Colab was not used for training due to error in implementation, and thus training was very slow.

For future projects, we would make sure to get Google Colab or other computing assistance up and running earlier to make room for quicker debugging and further experimentation with hyperparameters.

One additional point is to debug the code in a more systematic way to catch the bugs in time, since programming in this gym-torch fashion can make it difficult to discover them. Furthermore, we would like to explore other aspects of how to improve training, more specifically in the gym environment itself. This could be wrappers, stacking more/less frames etc. Finally, we wish to see how this DQN implementation performs in other Atari games.

References

[1] https://www.gymlibrary.ml/environments/classic_control/cart_pole/?highlight=cartpole, Accessed: 2022-05-19.

[2] https://en.wikipedia.org/wiki/Pong, Accessed: 2022-05-19.

[3] https://www.gymlibrary.ml/, Accessed: 2022-05-19.

[4] Mnih, V, et al. Playing atari with deep reinforcement learning, 2013.

[5] Mnih, V, et al. Human-level control through deep reinforcenemt learning, 2015.

[6] https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html, Accessed: 2022-05-19.