Learning to play Pong with DQN

Group 21 - Linus Falk

May 28, 2023

1 Introduction

This project is part of the Reinforcement learning 7.5c course at Uppsala University. We are asked to implement the Deep Q-network that was first introduced by Mnih et.al [1]. In order to simplify the work and debugging of the code we were first asked to implement it in a simple environment such as the CartPole and after that move on to the more challenging Pong environment [2].

2 Deep Q-network (DQN)

Deep Q-network is a Reinforcement learning algorithm that combines the use of neural networks and the classic reinforcement learning technique, Q-learning. In Q-learning the agent learns the Q(s,a) function which estimates the total reward that can be obtained by taking an action a in state s, where $a \in \mathcal{A}$ and $s \in \mathcal{S}$, the action and state spaces. This function can be solved by using the Bellman equation when the state and action spaces are small [3]. However, in most application that is not the case and we most use function approximation instead of a matrix of action and states. Using deep learning methods is one way to solve it could be hard to train since the most deep learning algorithms expects that the data samples to be independent of each other, this is not often the case in Reinforcement learning. To overcome the obstacles of using deep learning methods to model more complex action spaces was these solutions proposed [1].

- **Experience Replay**
- **Target network**

In order to solve the correlation problem was the **Experience replay** introduce that stores sequences of state, action, reward and next state. When training the replay buffer is sampled to update the network which helps to break the correlation between consecutive samples[1]. The **Target network** was introduced to stabilize the training that was often unstable when generating the target with the same model that could affect the action values for the next step leading to catastrophic forgetting of learning. By using a model that is not updated immediately and uses old parameters it was possible to get more stable training [4]

3 Cartpole-v1

The Cartpole model was implemented using the Gymnasium tool kit for Reinforcement learning and with skeleton code provided by the instructor. The neural network was in this case very simple, see table: 1:

Layer (type)	**Input Shape**	**Output Shape**
Linear 1	4	256
ReLU		
Linear 2	256	2

Table 1: Neural Network Architecture

The model was then trained with different hyperparameters to see how it affected the training and result. The hyperparameters common for all model are presented i table: 2 and the hyperparameters that was change are presented in table: 3. To evaluate the training was the training stopped and evaluated periodically and tested for 5 times and the mean return was then plotted over the course of the training. The average return of these evaluations during training are presented in figures: 1a to 1e.

Hyperparameter	
memory_size	50000
n_episodes	1000
batch_size	32
lr	1e-4
train_frequency	1
gamma	0.95
anneal_length	10^4
n_actions	2

Table 2: Hyperparameters for CartPole-v1

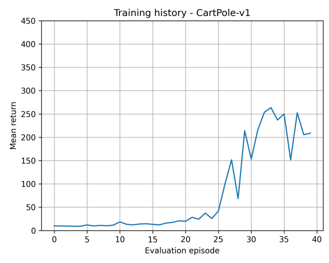
Hyperparameter	**Model 1**	**Model 2**	**Model 3**	**Model 4**	**Model 5**
target_update_frequency	100	5	150	100	100
gamma	0.95	0.95	0.95	0.95	0.95
eps_start	1.0	1.0	1.0	0.5	1.0
eps_end	0.05	0.05	0.05	0.05	0.5

Table 3: Hyperparameters for CartPole-v1

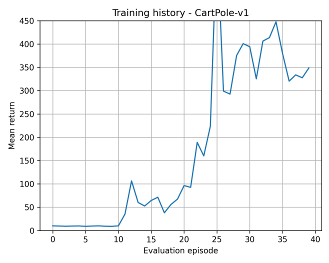
Discussion

Looking at the return plots of the different models we can see that reducing the *target update frequency* (in model 2) lead to a better model with higher return. The correlation problem was perhaps not that big in this simple environment, but we can see some tendency that it fluctuates more than the previous model. Dialing the *target update frequency* up showed instead that the training became too slow/stable and didn't learn enough the environment to improve. In model 4 and 5 the exploration strategy's was changed by changing the probability for exploration during the training. Model 4 with less experience in the beginning show slower progress than model 1 which is expected. Model 5 on the other hand keeps a high degree of exploration and is instead not able to find a more optimal policy do to this, giving it a smaller overall return than model 1 and 2.

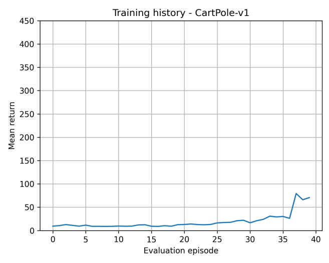
The implementation of this environment was straight forward with the skeleton given and easy to understand #TODO's. Some problems and perhaps a deviation from strategy thought of the instructor was done with the masking of the terminating states. Here was an integer introduce to "mask" the terminated samples. A simple mistake of not updating state to the next state in the end of the training loop gave more headache than i should.



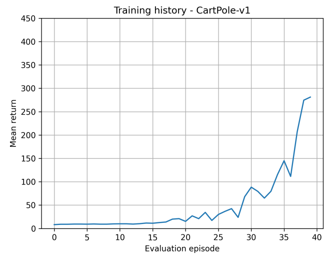
(a) Model 1



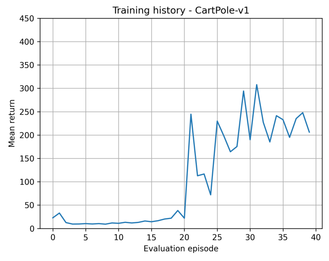
(b) Model 2



(c) Model 3



(d) Model 4



(e) Model 5

```
obs = torch.tensor(obs).to(device)
next_obs = torch.tensor(next_obs).to(device)
action = torch.tensor(action).unsqueeze(0).to(device)
reward = torch.tensor(reward).unsqueeze(0).to(device)

if terminated:
  terminated_bool = torch.tensor(1).unsqueeze(0)
else:
  terminated_bool = torch.tensor(0).unsqueeze(0)

memory.push(obs, action, next_obs, reward, terminated_bool)

##########

Q_targets = concatenated_reward + (dqn.gamma * max_next_q_values * (1 -
    concatenated_terminated))
```

4 Atari - Pong

The next step was to train a model for the Atari-Pong environment. The code from the previous environment could mostly be reused but new hyperparameters that would "guarantee" a reasonably good training was given, see table: 4. The Neural network was also updated to a Convolutional Neural Network, CNN. The architecture is presented in figure: 2.

Hyperparameter	**Value**
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

Table 4: Hyperparameters

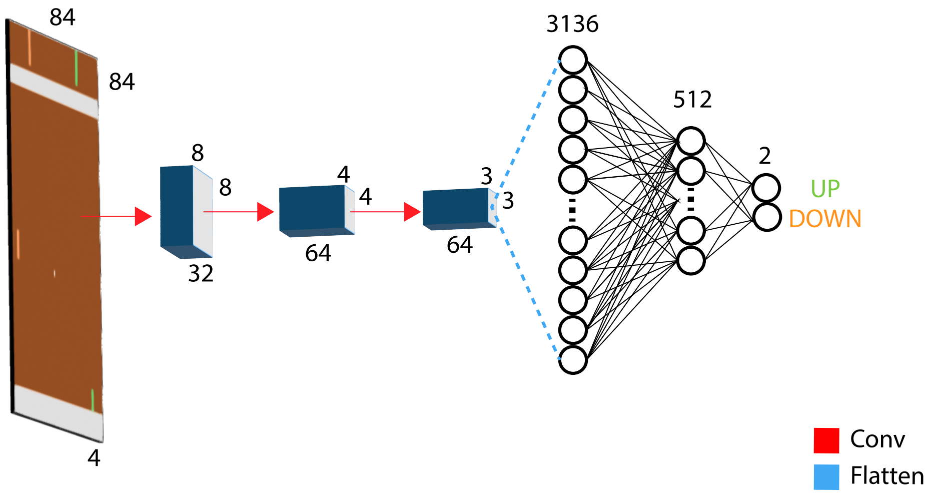


Figure 2: CNN architecture

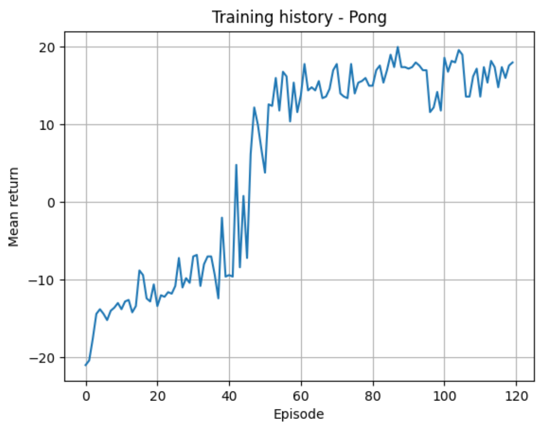


Figure 3: Neural network architecture

Discussion

The most challenging part of this part was to get the stacking of the sequence of images that was collected from the environment and save them to the replay memory correctly. But the instructions given in the task helped and the problem was solved eventually. After overcoming these minor obstacles the work was not that difficult until it was time to start to test. One of the challenges was to see if it was correctly implemented and actually could improve its policy. The training took typically around 5-6 hour training it on a *Nvidia RTX3060*, with not much improvement the first 5-10 minutes. The result is present in figure: 3. The episode on the x-axis is the number of the evaluation episode that was done every 25th normal episode.

References

[1] Volodymyr Mnih and Koray Kavukcuoglu and David Silver and Alex Graves and Ioannis Antonoglou and Daan Wierstra and Martin Riedmiller (2013). Playing Atari with Deep Reinforcement Learning

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

[3] Richard S. Sutton and Andrew G. Barto (2018). Reinforcement learning - An introduction

[4] Mnih, V., Kavukcuoglu, K., Silver, D. et al (2015) Human-level control through deep reinforcement learning.