SUS - Project 2 - Report

Jakub Pniewski

June 2025

1 Introduction

The task was to train a model that achieves the highest possible total reward in the "CartPole-v1" environment. I chose the Deep Q Learning approach and succeeded in getting the highest possible score.

I based my solution on two sources:

- Lecture from Stanford University
- PyTorch dosc tutorials

The solution doesn't implement any of these sources directly. From Stanford, I took the basic training algorithm. From PyTorch, I took some library nuances. More details will follow in the next paragraphs.

2 Learning Algorithm

The idea from Stanford University is based on Google DeepMind's work on DQN that was done for agents playing Atari games and for the AlphaGo model.

We have two agents - policy and target. Policy agent is the one we train. We iterate through episodes (games). In every state of every game, the policy agent chooses the action. We don't train the model on that state immediately – instead, we add that state and action to replay memory.

After each state of the game, we take a batch from replay memory and train the policy model on it. We do it with a method derived from the Bellman equation:

$$pred = Q_{policy}(s, a)$$

$$y = r + \gamma \cdot max_{a'}(Q_{target}(s', a'))$$

We calculate the loss and update the policy model. From time to time, we copy the parameters of the policy model to the target model.

Detailed description is available in the lecture linked above.

I also use the epsilon greedy approach described in the Stanford lecture. At times I choose a random action instead of using the one determined by the policy model. This improves exploration. I also decay the amount of times I use a random action over time (epsilon decay). This improves exploitation in the later phases of the training.

I also use early stopping - if the model scores perfectly several times in a row I stop the training.

This method was shown by researchers to work well for much more complex games and to work very well for simple Atari games. That is why I decided to use it.

3 Network Architecture

I used a very simple neural network. It consists of only 4 layers:

- input layer
- 2 hidden layers
- output layer

I use the SiLU activation function between each pair of layers.

Input dimension is 4, output is 2. I found that 32 is sufficient as the hidden dimension! This means that the model has less than 2500 parameters!

Before the data goes into the neural network, I preprocess it using values from gym documentation and values derived empirically.

4 Training Setup

I use simple MSE as my loss function. The optimizer I decided to use is Adam. I have a relatively low learning rate that decreases linearly with respect to the number of played episodes.

I had to make replay memory pretty small. This game is probably a bit too simple for it to give any improvement.

The script supports GPU acceleration. It automatically detects a CUDA device.

The parameters I ended up using:

- batch size: 128
- learning rate 0.001
- number of episodes: 300
- ϵ: 0.1 (10%)
- ϵ decay: 0.95 (multiplied per episode)
- γ: 0.99
- steps without updating target: 100
- quality assesment steps: 25
- nn hidden dimension: 32

I tested smaller models but couldn't achieve perfect results, so I chose a hidden dimension of 32 as the minimal size that consistently performs well.

The number of training episodes is somewhat redundant — in practice, training usually completes much earlier due to early stopping.

The number of quality assessment steps is a trade-off: I want a model that consistently achieves good results, but I don't want to discard a potentially good model due to an unlucky batch. To mitigate this risk, I also implemented gradient clipping in the training script.

5 Training and Evaluation

Training time is about 3 minutes on my PC and results in a perfect model! Evaluation confirms these results.

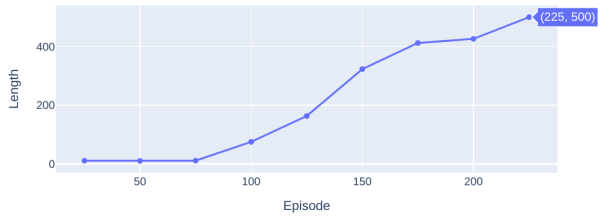

Figure 1: Average result of 25 consecutive episodes in training

6 Final Thoughts

I was able to train a relatively small and efficient model that consistently solves the CartPole-v1 task with perfect performance. Used algorithm is a very fast way of training such model.