Università degli Studi di Padova

Homework 3: Reinforcement Learning

Neural Networks and Deep Learning

Chelsea Owen
Student ID: 2006036

September 13, 2022

In this homework the task was to implement neural network models for Reinforcement Learning (RL) problems. First the model will solve the CartPole v1 game. The input is the state representation, (cart position, cart velocity, pole angle, pole angular velocity). The model hyperparameters are then optimised using a random search in order to find the model which solves the game the most efficiently. Additionally another model will learn to solve the Acrobot-v1 game.

1 Introduction

RL is about taking actions which maximise the reward in a particular environment. The rewards and penalties are specified according to the goals of the trainer. The goal is to find the best possible behaviour or set of actions that should be taken for maximum reward, or in other words 'solving' the problem. Rewards are given at for each action that the agent takes, and overall total reward is the final sum of all of the individual rewards. Like in unsupervised learning there is also no 'answers' aka labels to use. The network, or agent, must decide what to do itself and then learn from its experience in order to find the optimal policy. The learning process thus takes the trial-and-error approach. For these homework tasks a neural network will be trained as a Deep Q-Learning (DQ) Agent.

The most important elements in a RL system are: the agent, the environment, the policy of the agent and the rewards basis by which the agent is scored.

2 Gym environment

For the environments we will use OpenAI Gym which has multiple Atari games. We will use it for both of the following old school games:

- **CartPole v1:** A pole has to balance on top of a cart. You can move the cart either left or right in order to prevent the pole from toppling, which it will do if the angle exceeds 15 degrees from vertical. The game ends if it reaches maximum points, the pole falls or the cart goes out of frame. The maximum score of the game is set at 500 points.

- **LunarLander v2:** A space ship descends and has to land within the landing flags on the moon's surface. The space ship can use its engines to thrust left, right or up. The game ends when it lands or crashes or goes out of frame.

3 CartPole v1

The gym environment for this game is a cart which may move frictionless on a 1-dimensional track. Atop the cart is a pole, attached at its base to the centre of the cart but free to oscillate. At the beginning, the pole is always in an upright position, and then the aim of the game is to move the cart in order to prevent the pole from falling. See figure 1. *Gym* provides the set up space which includes the cart position (between the range of -2.4 and +2.4), the cart velocity ($-\infty$ and $+\infty$), the pole angle ($-15°$ and $+15°$) and the pole angular momentum ($-\infty$ and $+\infty$).

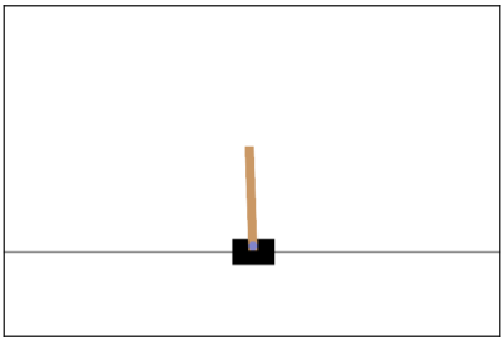

Figure 1: CartPole v1 Environment

3.1 Methods

First the ReplayMemory class is defined in order to implement the Experience Replay. We use the 'deque' object from python for a queue of a predefined length. The agents experiences at each step are added to the list until the maximum length is reached, and then it starts remove the first (oldest) step elements of the queue and add the last (newest) ones.

Next we define the fully connected feed forward network within the DQN class with the state space dimensions as inputs and the action space dimensions as outputs. This policy network provides the Q-value for each of the possible actions. The Q-value is the expected long-term reward from taking that action. The network is composed of three linear layers and hyperbolic tangent activation function. The two hidden layers both have 128 neurons.

From the Q-values the action must be chosen. Here there is a choice of prioritising the known pathways to get high reward (exploitation), or choosing new actions which may or may not lead to better rewards (exploration). This trade-off is varied with a parameter. Initially, when little is known, exploration should be favoured and then later the balance can tip more towards exploitation.

The two exploration policies are ϵ*-greedy* and *Softmax*. The ϵ-greedy policy chooses a non optimal action with probability ϵ, and the optimal action with probability $1 - \epsilon$. The softmax policy instead chooses and action based on a distribution of softmax with a temperature τ applied to the estimated Q-values. At temperature $\tau = 0$ the policy will choose the action with the highest Q-value. In this part of the homework the softmax strategy is used.

Thus, τ and ϵ will change during the learning process and that change must be defined. For this, two *exploration profiles* are defined. In the first, decays exponentially and in the second it decreases linearly. See Figure 2.

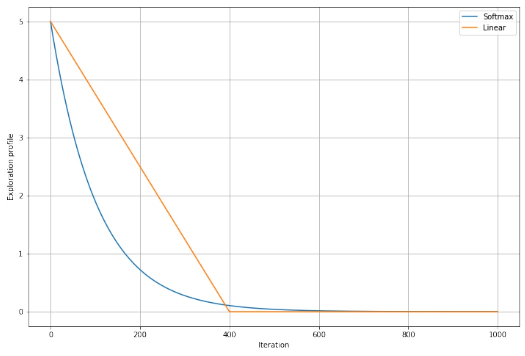

Figure 2: The two types of exploration profile, with initial $\tau = 5$ in both cases

The parameters which will be optimised in the random search are:

- **SGD Learning rate:** sampled between 1e-3 and 1e-1
- **Temperature profile:** Linear or SoftMax
- **Initial temperature:** [1,2,3,4,5,6,7,8,9,10]
- **Number of steps between policy network updates:** [5,10,15,20,25]
- **Gamma:** sampled from $(0.9, 0.99)$
- **Batch size:** [32,64,128,256]

3.2 Results

50 different combinations of exploration profiles and hyperparameters were tested, with the optimal combination being chosen as the one which arrived at the top score of 500 within the least number of training step. Due to computation time considerations, the models were trained for 500 iterations each. This is acceptable because at this stage the goal is to find the network which gets to 500 points quickest. An example of a network is shown in Figure 3

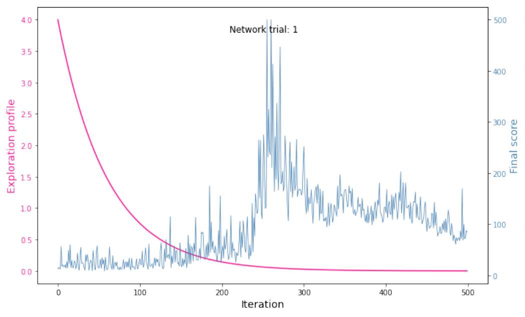

Figure 3: Example of a randomised network training

The optimised hyperparameters were:

- **SGD Learning rate:** 0.0599
- **Temperature profile:** Linear
- **Initial temperature:** 2
- **Number of steps between policy network updates:** 5
- **Gamma:** 0.98
- **Batch size:** 128

The model training with these parameters got to 500 points in only 338 iterations. See Figure 4.

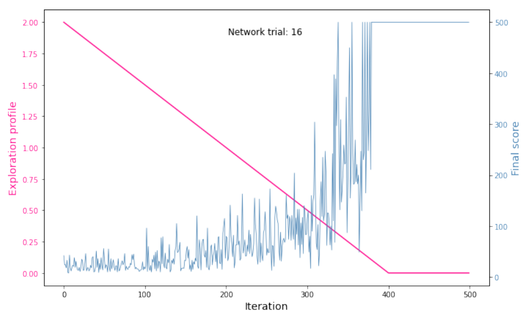

Figure 4: Training of the best network showing it 'solves' the game quickly

Click on this link to see an example video of the solved game.

4 Acrobot-v1

In Acrobot-v1, the goal of the game is for the pendulum to reach the horizontal line above its starting position. The gym environment in this case has six variables, the $sin()$ and $cos()$ of the first angle relative to the vertical and the join angle relative to the first angle, as well as the joint angular velocities of the two limbs. Figure 5 shows a still from the game.

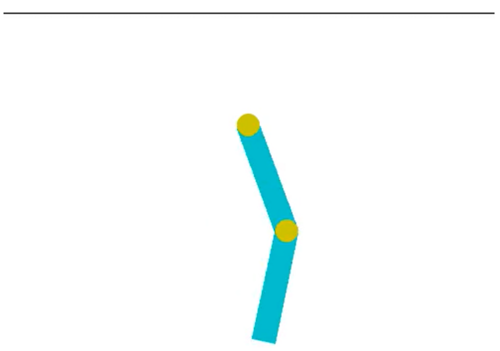

Figure 5: Acrobat-v1 Environment

There are three actions available at any point: applying either a $+1, 0$ or -1 torque to the joint between the two limbs. The reward is 0 if in that timestep it reaches the horizontal line, 6, or -1 if it does not. There is a 500 timestep limit, so the score will be within the range $(-500, 0)$ with 0 meaning it reaches in every step and -500 meaning it never reaches.

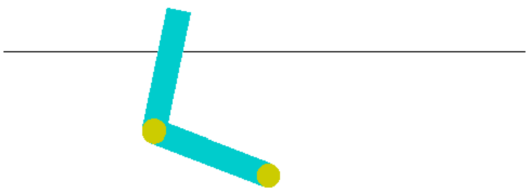

Figure 6: Example of Acrobat getting reward = 0 for that particular timestep

4.1 Methods

We will use the same DQN class and ReplyMemory class as for the CartPole game. The input and output parameters of the network, possible actions and the reward system however are now different.

The environment is tested with an agent which chooses the action randomly, and the score for each of the ten episodes was indeed -500, meaning it never once reached the horizontal line.

4.2 Results

The initialised network was trained for 800 iterations using the following hyperparameters:

'gamma': 0.99, 'lr': 0.001, 'target_net_update_steps': 10, 'batch_size': 256, 'initial_temp': 5

The training process can be visualised in Figure 7.

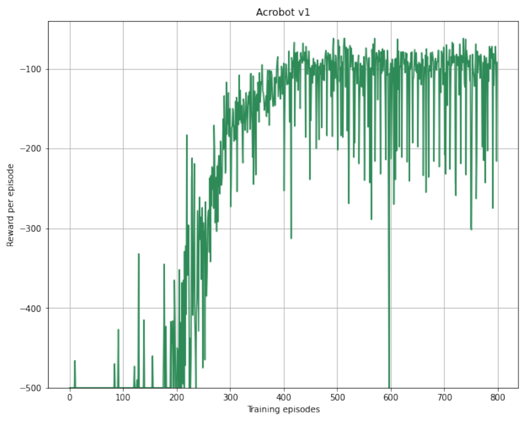

Figure 7: Training for the Acrobot game

The graph shows the learning process, and after about 200 iterations, once it has gained enough information, it starts to successfully get higher scores with each passing episode. This is when the temperature decreases and the network starts to prioritise exploitation over exploration. Unlike the CartPole game however it is not able to find the perfect solution for the game, it can only try to minimise its score.

Testing the final performance on the training network for ten episodes had these results:

Episode 1	-85.0
Episode 2	-82.0
Episode 3	-97.0
Episode 4	-91.0
Episode 5	-116.0
Episode 6	-72.0
Episode 7	-104.0
Episode 8	-157.0
Episode 9	-70.0
Episode 10	-84.0

View the best scoring episode, episode 9, at this link.

5 Conclusion

To conclude, the optimised CartPole-v0 network was able to solve the game in only 338 iterations. After the network learned the best set of actions and the temperature was set to zero, it was able to always get the maximum score of 500 showing that the strategy is optimal for the environment.

The Acrobot-v1 game cannot exactly be solved, however the network was able to learn to minimise the loss in about 400 iterations, where the score for each iteration converges at around -100.