Project 1: Navigation

Edoardo Bacci

Deep Reinforcement Learning Nanodegree

Udacity

06/03/2019

Keywords: Q-Learning

1 INTRODUCTION

For this project we were required to train an agent by using Deep Q-Learning. The testing benchmark used in this project was a simulated environment developed using Unity where the agent is supposed to collect as many yellow bananas as possible while avoiding blue bananas within some time boundary.

2 THEORY

In Q-Learning[3] we associate a value to each state-action pairs. When the state-action pairs lead to a terminal state we assign a value equal to the reward obtained.

Otherwise we assign a value according to the formula:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)) \quad (1)$$

Q-Learning requires to store every value for each state value-pair. This can quickly lead to an explosion of possible state-action combinations and it is suitable only for small problems.

For big problems we can approximate the Q-function with a neural network having weights θ. In order to train the neural network we use the loss function

$$L(\theta) = \mathbb{E}_{(s,a,r,s')}\left[\left((r + \gamma \max_a Q(s', a, \theta) - Q(s, a, \theta))\right)^2\right] \quad (2)$$

Where θ are the weights of the neural network

The above method suffers from instabilities due to the fact that θ is both updated and used as a target at the same time. The problem is solved by using two copies of the same network and keep one as a target while the other gets updated.

Every n iterations the target network gets updated with the local network and the learning continues. Another variant is to slowly update the weights of the target network by a fraction of the local network.

2.1 Double DQN

In the above method the maximum value of each state-action pair is reliable only when the weights are properly trained. When the network is still at the beginning of the training the initial estimates will probably be random.

Taking the maximum of this estimates will likely overestimate the real Q-value of each state-action pair. In order to counteract this behaviour we use a new method for selecting actions called Double DQN[2].

In Double DQN we select action using using one network but we estimate their value using another.

One way of doing this is selecting the values using the local network and estimating their value using the target network.

The updated equation is :

$$L(\theta) = \mathbb{E}_{(s,a,r,s')}\left[\left(r + \gamma Q(s', \operatorname*{argmax}_a Q(s', a, \theta), \theta') - Q(s, a, \theta)\right)^2\right] \quad (3)$$

Where θ is the local network and θ' is the target network.

2.2 Dueling DQN

Another improvement that can be made comes from the fact that in some of the states the action taken has very little influence on the final score. By separating the advantage we get by choosing some actions from the value of the state overall can benefit the trained policy.

For this reason the paper "Dueling Network Architectures for Deep Reinforcement Learning"[1] split the last layers of the neural network in two "heads".

One head of the network will estimate the value of the state while the other will estimate the "advantage" of each action, that is how good every action is compared to the average of every action in the state.

The output equation of the network is going to be:

$$Q(s, a, \theta, \alpha, \beta) = V(s, \theta, \beta) + \left(A(s, a, \theta\alpha) - \frac{1}{|\mathcal{A}|}\sum_{a'} A(s, a', \theta, \alpha)\right) \quad (4)$$

where θ represents the shared weights of the neural network and α and β represent the weights of each head.

3 METHODS

The algorithm used for this project is a combination of the Double DQN, Dueling DQN and Prioritised Experience techniques described above. The full pseudocode can be found in Algorithm 25.

The full code can be found at https://github.com/phate09/drl_banana.

3.1 Architecture

The architecture that I used was a simple feed-forward neural network with 2 fully connected layers followed by a "Value function" head and an "Advantage function" head.

The output of the network consists of the combination of the two heads as described in "Dueling Network Architectures for Deep Reinforcement Learning"[1].

The final architecture is shown in Figure 1.

Algorithm 1: Deep Q-Learning (Dueling, Double, Prioritised Exp Replay)

Input: minibatch k, step-size η, replay period K and size N, exponents α and β, episodes budget T, target update τ.
Initialise θ randomly and copy to θ'
Initialise replay memory $\mathcal{H} = \emptyset$
for $i \leftarrow 0$ **to** T **do**
  $s \leftarrow$ reset environment
  while *not done* **do**
    choose $a \sim \pi(s)$
    s', r, *done* = environment step (s,a)
    Compute TD-error $\delta =$ $r + \gamma Q(s', \underset{a}{\text{argmax}} Q(s', a, \theta), \theta') - Q(s, a, \theta)$
    store (s,a,r,s', *done*) in experience replay prioritised by δ
    $s \leftarrow s'$
    if $t \mod K = 0$ **then**
      for $j \leftarrow 0$ **to** k **do**
        Sample transition s_j, a_j, s'_j, r_j from $\mathcal{H}$
        Compute importance-sampling weight $w_j = (N \cdot P(j))^{-\beta} / \max_i w_i$
        Compute TD-error $\delta =$ $r_j + \gamma Q(s'_j, \underset{a}{\text{argmax}} Q(s'_j, a, \theta), \theta') - Q(s_j, a_j, \theta)$
        Update transition priority $p_j \leftarrow |\delta_j|$ in $\mathcal{H}$
        Accumulate weight-change $\Delta \leftarrow \Delta + w_j \cdot \delta_j \cdot \nabla_\theta Q(s_j, a_j)$
      end
      Update weights $\theta \leftarrow \theta + \eta \cdot \Delta$
      Update target weights $\theta' \leftarrow (1 - \tau) \cdot \theta' + \tau \cdot \theta$
      Reset $\Delta = 0$
    end
  end
end

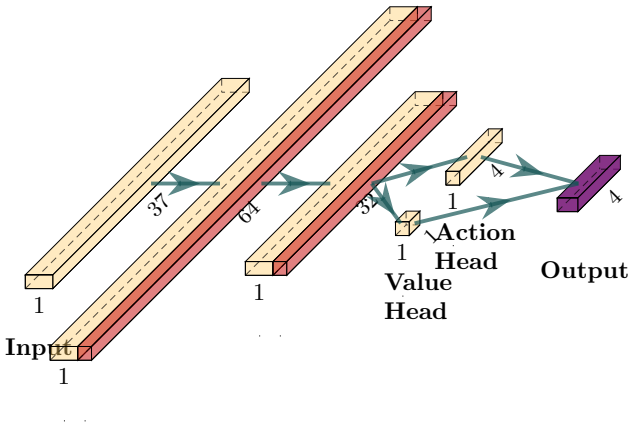

Figure 1. Architecture for the agent model (yellow layers are linear, red ones are RELU, purple combines the two heads as described in the Dueling DQN section)

3.2 Hyperameters

To reach good results, the algorithm has some hyperparameters that needs to be tuned to the problem. For this project the parameters that have been used are:

- Learning rate $\eta = 5e-4$
- Batch size $k = 64$
- Episodes budget $T = 2000$
- Epsilon $\epsilon = 1.0 \rightarrow 0.01$ during the first 20% of the episodes budget, then constant
- Experience Replay Size $k = 1e6$
- Discount factor $\gamma = 0.99$
- Replay period $K = k$
- Target update coefficient $\tau = 1e-3$
- Prioritised experience exponents $\alpha = 0.4, \beta = 0.5 \rightarrow 1.0$ annealed during entire training

4 RESULTS

The results are reported in Figure 2. The algorithm improves steadily its performance until around iteration 600. From there it slowly increment up to around a score of 16 and keeps that score for the rest of the number of iterations allowed. A video of the agent in action can be found at https://youtu.be/uQFR3ll6CqM. A graph that plots the score of a single run can be found at Figure 3.

5 FUTURE IMPROVEMENTS

As seen before, combining different techniques to counteract issues during training helped to make the agent more stable during training and converge faster to a near optimal solution. Researchers have also combined other techniques in a single algorithm called "Rainbow"[5] that manages to cope with the issues mentioned above and others which were not covered here. A possible improvement to my algorithm would be to implement the same techniques covered in "Rainbow" to make it converge even faster.

6 CONCLUSION

This project was my first ever attempt to develop an agent using Q-Learning.

It was an exciting experience where I learnt a lot. The hardest part has been the hyperparameter tuning because every problem is different and requires different hyperparameters hence I could not find appropriate values from research papers.

REFERENCES

[1]"Dueling Network Architectures for Deep Reinforcement Learning", Ziyu Wang and Nando de Freitas and Marc Lanctot, 2015
[2]"Deep Reinforcement Learning with Double Q-learning", Hado van Hasselt and Arthur Guez and David Silver, 2015
[3]"Human-level control through deep reinforcement learning", Volodymyr Mnih et al. , 2015
[4]"Prioritized Experience Replay",Tom Schaul, John Quan, Ioannis Antonoglou, David Silver, 2015
[5]"Rainbow: Combining Improvements in Deep Reinforcement Learning", Matteo Hessel et al., 2017

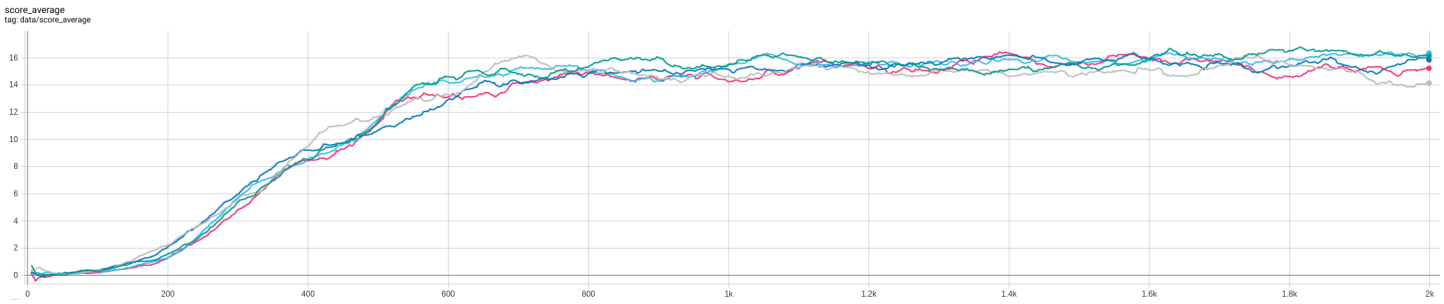

Figure 2. Plot of the score average (last 100 episodes) using different seeds. From the plot it can be seen that the algorithm already reach an average of 13+ around iteration 600

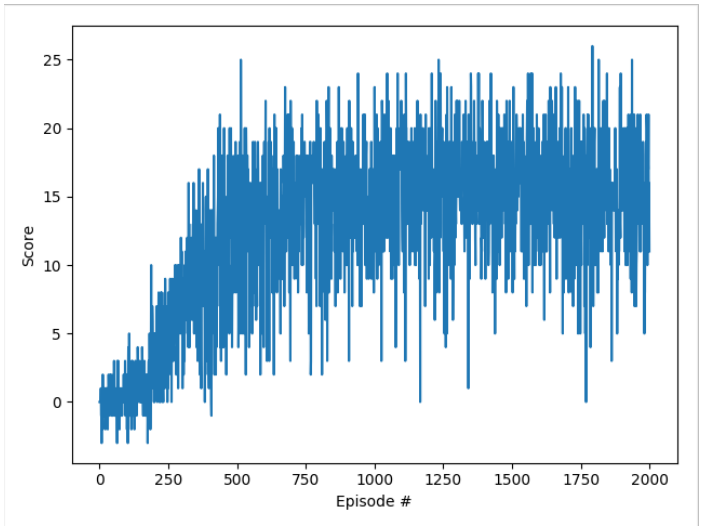

Figure 3. Score of a single run of the algorithm presented