Project 1 - Deep reinforcement learning - Navigation

Gianluca Scarpellini (gianluca@scarpellini.dev)

April 4, 2020

Contents

1 Introduction

Deep Reinforcement Learning is today becoming a relevant field of research. The current work aims to evaluate the ability of value-based methods of solving a simple task in a constrained environment. In particular, we implemented Deep Q-learning algorithm as presented during Udacity "Deep Reinforcement Learning" as well as in the original paper. We describe the environment in section 2. A more in depth explanation of Deep Q-learning is presented in section 3. In section 4 we comment the results obtained with the algorithm. Finally, we take our conclusions in section 5

2 Environment and task

The presented project is developed in order to solve the "Banana" environment. The task consists in collecting yellow banana in a plain space while avoiding blue banana. The agent needs to learn the optimal policy. It gets a reward of +1 for each yellow banana and a reward of -1 for each blue banana. The action space is limited to 4 possible choices: move left, right, forward and backward.

3 Algorithm and Implementation

We decided to solve the environment with Deep Q-learning algorithm. Deep Q-Learning has already shown its potential in solving Atari games as well (or better then) human players. It's an off-policy learning algorithm. 2 different policies are kept simultaneously: an evaluation policy and the learned policy. Q-learning, and temporal-difference learning algorithms in general, learn an action-value function Q from each step of each episode. Compared to Monte-carlo methods, TD-methods reduces learning time at the cost of an additional **bias**. The equation in figure 1 refers to the vanilla pnpQ-Learning update. Q function is updated using a **future** value estimation (which introduces bias in the learning process). Modern development of Deep Learning and Neural Networks can be exploited for **approximating** function Q with function approximators.

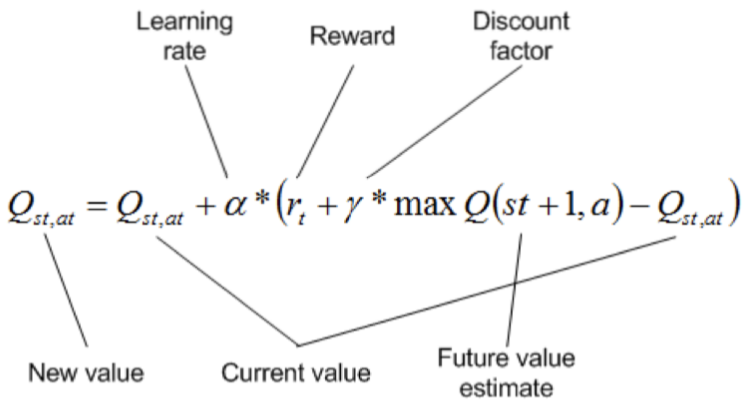
Learning rate
Reward
Discount factor
$Q_{st,at} = Q_{st,at} + \alpha * (r_t + \gamma * \max Q(st+1, a) - Q_{st,at})$
New value
Current value
Future value estimate

Figure 1: Q-Learning equation

The main idea of Deep Q-Learning is contained in figure 2. In particular, Deep Q-learning implements 2 techniques in order to stabilize the training process and allow the neural network to learn the **optimal** action-value function.

- Fixed Q-target: 2 different networks are combined in order to keep the method

off-policy. A target network is updated at every iteration while an evaluation network is exploited for future value estimation. The target network parameters are copied to the evaluation network at the end of the updating phase.

- Experience Replay: In order to decouple sequential states of each episode,

Deep Q-Learning create a buffer of tuple <S, a, R, S'>. At each iteration a random batch is pulled from the buffer. The main goal of experience replay is to feed into the learning algorithm independents batch of tuples.

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$$

Figure 2: Deep Q-Learning equation weights update equation

3.1 Hyperparameters

For the experiment we used the following hyperparameters:

Hyperpameter	Value
lr	5e-4
tau	1e-3
gamma	0.99
update every	4
buffer size	1e5
n. hidden layers	2

4 Results

In figure 3 it's presented the algorithm result per episode. In particular, we were able to solve the environment in less then 300 epochs. The scores kept growing until an optimal maximum of 13, probably due to limited time play per episode. The learning phase was revealed to be a noisy. More stables approaches are discussed in section 5.

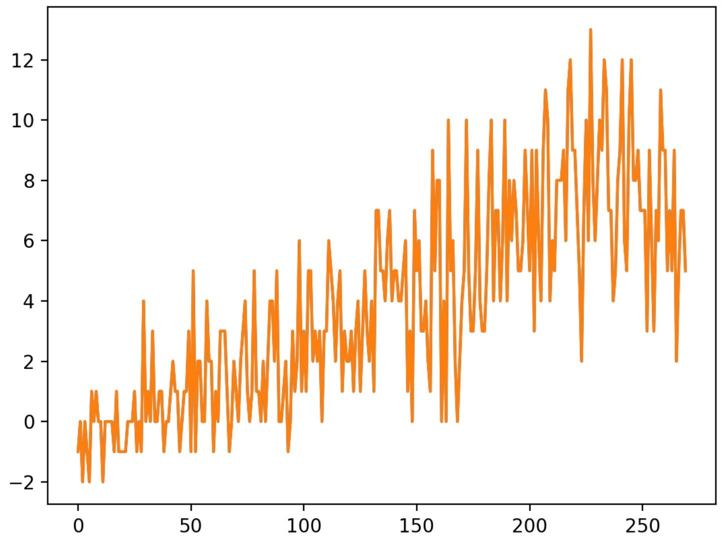

Figure 3: Learning scores per epoch

5 Conclusion and further

We developed a pipeline in order to solve ‘banana‘ environment with Deep Reinforcement Learning. In particular, we implemented **Deep Q-Learning** following the paper specification. Deep Q-learning is an off-policy algorithm which has proven stability and optimal results in multiple tasks. We got the algorithm to hack the environment in less then 300 epochs. As discussed above, we believe the noisy of the learning could be alleviated with further developments of both learning phase and the buffering phase. We would consider integrating **Prioritized Replay** as a way to optimally sample mini-batches from the queue. Further improvements are available in implementing **Double DQNs** strategy in order to disentangle the calculation of Q-targets for best action estimation from Q-values.