April 2020

Using Deep Q-Learning to Solve Banana Environment

Vicente LÓPEZ [a]

[a] *Student at UJI. email: al341918@uji.es*

Abstract. The Reinforcement Learning is a good technique to face problems than can not be easy modelized by equations and can not make an algorithm to solve it. Because of that is a field in which they are made many studies. Q-Learning is one of the algorithms of Reinforcement Learning but have limitations due to their memory cost. To solve these problem, Neural Netwroks are used, becoming a new algorithm named Deep Q-Learning. Often find the good hyperparameters and Neural Architecture is a hard problem, so we studied the hyperparameters and Neural Architecture to solve the environment Bananas.

Keywords. Reinforcement Learning, Deep Q-Learning, Neural Network, Unity, Double Q-Learning.

1. Description of the problem

For this study, we trained an agent to navigate in a large, square world. The environment was maked using Unity.

A reward of +1 is provided for collecting a yellow banana, and a reward of -1 is provided for collecting a blue banana. Thus, the goal of the agent is to collect as many yellow bananas as possible while avoiding blue bananas.

The state space has 37 dimensions and contains the agent's velocity, along with ray-based perception of objects around agent's forward direction. Given this information, the agent has to learn how to best select actions. Four discrete actions are available, described in Table 1.

Number	Action
0	move forward
1	move backward
2	turn left
3	turn right

Table 1. Posible actions of the agent.

The task is episodic. The problem is considered solved if the agent get an average score of +13 over 100 consecutive episodes.

2. Model

2.1. Methodology

The model use Double Deep Q-Learning (DDQN) to solve the environment. Also use Memory Replay to make more robust our model. The hypeparameters used to solve the problem are avaible in Table 2

Hyperparameter	Value	Hyperparameter	Value
Replay memory size	100000	Batch size	64
Gamma	0.99	Tau	0.001
Update target every	4	Initial epsilon	1
Epsilon decay	0.995	Min epsilon	0.1
Learning rate	0.0005		

Table 2. Hyperparameters of the model.

2.2. Architecture

Our model consists in a Neural Network of 3 fully connected layers. The first layer is the input layer and have 37 input .the length of the state- and 512 output. The second is a hidden layer, which have 512 both for input and output. The last layer is the output layer and have 512 of input and 4 of output -the number of actions. In resume, our model consists in 3 layers with the form 37 - 512 - 8.

Neural Networks also need and optimizer and a way to compute the loss. Adam is used as the optimizer and the algorithm used to compute the loss is MSE.

3. Experiments and Results

The objective of the model is to play the game with a average score of 13 in 100 consecutive episodes. If we train the model until this goal is achieve we have the rewards of Figure 1 and reach them in 562 episodes. But, when we look how to play the agent, we see that it do not make a good game. In fact, It can not avoid walls and when see a walls it goes for it until it crashes.

To solve this problem, we train the agent until It reach an average score of 15 -and for so, have solved the problem too- in 1706 episodes with the evolution of scores show in Figure 2. In the figure you can see that the model can not learn much more, and for do we are closer to the maximum performance of this architecture for this problem. If we see now the agent, it can avoid walls now, but can be traped in the middle of a blue banana.

4. Conclusion and future work

This model architectura have enought power to solve the environment, but still have some problems with the learning like the blue banana trap. Also have another problem, and

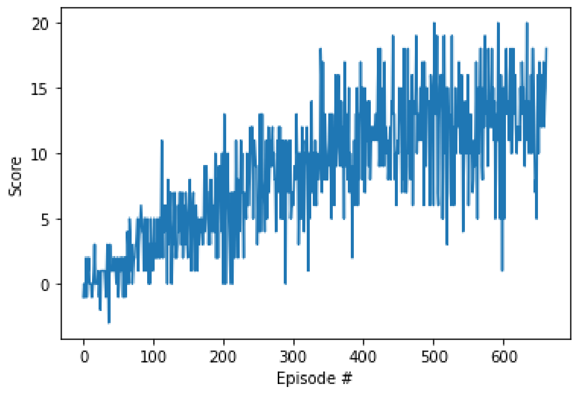

Figure 1. Rewards for model of 13 average score..

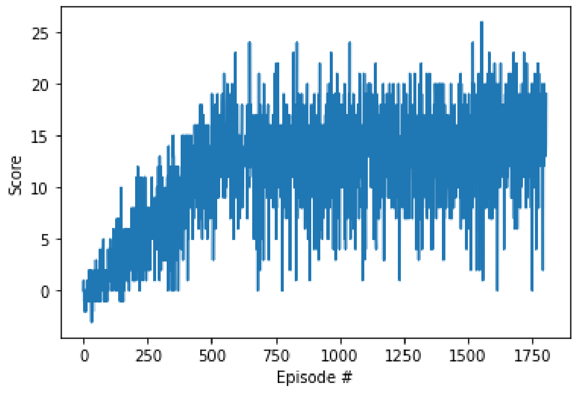

Figure 2. Rewards for model of 15 average score.

its that this model is not learning from pixels, so this architecture is not extrapolable for robotics -which usually need Computer Vision-.

As future work we will adapt this model to learn to play the same environment from pixels. Also we can implement new improvements to the DDQN algorithm like Prioritized Experience Replay, Dueling DQN, Distributional DQN and Noisy DQN.