Project 1 - Navigation from the Deep Reinforcement Learning Nanodegree of Udacity

September 25, 2019

1 Introduction

This is the report for Project 1 - Navigation from the Deep Reinforcement Learning Nanodegree of Udacity.

The goal of the project is to train an agent to navigate a flat square world and collect bananas. The agent should be able to use its experience to gradually choose better actions when interacting with the environment.

The state space has 37 dimensions and contains the agent's velocity, along with ray-based perception of objects around agent's forward direction.

The simulation contains a single agent that navigates the environment. At each time step, it has four actions at its disposal:

- 0 - walk forward
- 1 - walk backward
- 2 - turn left
- 3 - turn right

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 task is episodic and, in order to solve the environment, the agent must pick an average score of +13 over 100 consecutive episodes.

2 Learning Algorithm

At a high level the algorithm follows the usual approach of Reinforcement Learning, in which the agent interacts with the environment by choosing an action at every time step, where the environment is in a state; the environment in turn responds by providing a reward to the agent for this, and moving to the next state. The learning proceeds through a loss function, a reward function in this case, which the agent seeks to maximize by tweaking its internal parameters (weights) via steepest descent, with the various improvements over a naive implementation that we have seen in the course. The high level algorithm implemented in the notebook contains little more than a loop to implement the learning process described above. The parameter *eps* controls the level of

randomness in the agent, and it is set so that the agent is close to random at the beginning of the training (exploration), while it is very small at later stages, so that the agent behaves as learned.

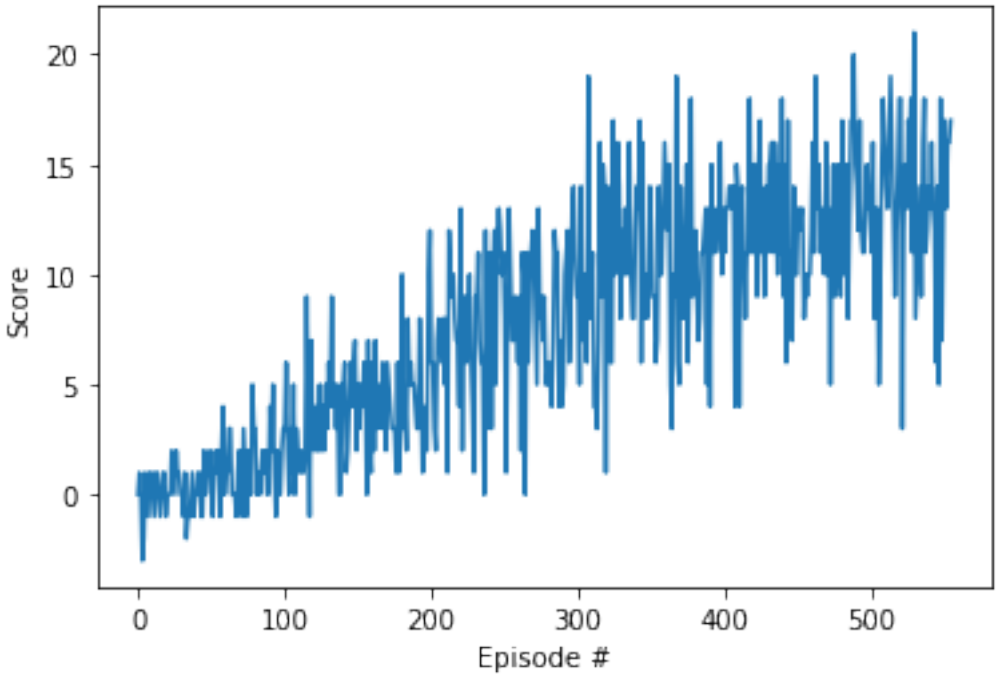
More insteresting is the internal configuration of the agent itself, provided in the python files (*agent.py* and *model.py*). These files implement the so-called Deep Q-Networks (DQN) algoritm, which essentially leverages the use of neural networks in the application of the more general Q-Learning algorithm. Q-learning is a model-free paradigm in which the goal is to learn a policy, i.e. a way of determining the actions, by means of updating the Q-table, i.e. the function giving expected future rewards for each state-action pair. As mentioned, in the case of DQN, the underlying model uses a neural network as a function approximator for the Q-function, in our case this was:

- an input layer
- a fully connected hidden layer with 64 units
- a relu activation function
- a fully connected hidden layer with 64 units
- a relu activation function
- an output layer

all networks were linear neural networks in this instance, and the target and expected Q values are calculated and used to develop a usual mean squared error (MSE) loss function to train the DNN.

In addition, the basic algorithm was enhanced with the usage of an experience replay buffer, that is a storage of a number of transitions to be sampled from, so that the agent can learn from them in different orders and avoid temporal correlations. Other techniques applied were the soft update process, and epsilon-greedy action selection to balance exploration and exploitation during learning.

With this configuration the learning rate was already quite good, as can be seen in Figure 1.



3 Future

Since the learnign rate is already quite good, and the task is not that trivial (I centainly could not beat the agent), I have not experimented much with the configuration. However approaches to improve the learning would be to change the hyperparameters in the first instance, e.g. use more or less units in the newtworks above, or even to change the configuration, say using a different type of networks or activation function.