Navigation Project Report

Udacity Reinforcement Learning Nanodegree Program

Author: Guilherme Sales Santa Cruz

7 may 2020

Summary

This document describes the navigation project. It is divided in 4 sections:

Problem description

The problem consists of creating an **Q-Learning** based agent capable of navigating through a room and collecting yellow bananas while avoiding blue bananas.

The agent is rewarded with +1 for each yellow banana collected and -1 for each blue banana.

The observation space has 37 dimensions and contains the agent's velocity, along with ray-based perception of objects around agent's forward direction. The action space has 4 discrete options:

- Move forward.
- Move backward.
- Turn left.
- Turn right.

Algorithm description

The algorithm used to solve this problem is called **DQN (Double Q-Network)**. It consists of an agent that receive continuous informations and take discrete decisions based on it.

Components

It has 3 different components.

The first is a ***memory buffer***. It has a fixed size and saves the experiences of the agent along the steps. One *experience* consists of: the state before the action, the action taken, the reward for that action, the new state and the information on weather the episode is over or not (done). Once it is full, it discards its oldest experiences to add new ones.

The other 2 components are the ***local and target network***. These neural networks are used to approximate the Q-Function, which is the function that values a given state. Two networks are used to reduce the chances that the agent overestimates the parameters while learning.

Methods

The agent has 2 public methods.

The first is ***act***. This method receives as input one state and outputs its estimation of the Q-value for this state.

The second method is ***step***. This method saves an experience and from time to time triggers the learning routine. This method receives as input one *experience* (see the previous section for definition) and stores it in the internal *memory buffer*. At every N-*steps* the learning routine is triggered. It takes a random sample of *experiences* from the *memory buffer* and trains one of the networks by using the other as an estimator for a more stable value. Then, after training it updates the second network by interpolating its weights with the recently trained one. The interpolation is linear to the TAU parameter where 0 is no interpolation and 1 is a pure copy.

Hyperparameters

Hyperparameters are parameters that should be tuned to improve the performance of the model. The following image shows the hyperparameters used to build the final agent.

```
BUFFER_SIZE = int(1e5)  # replay buffer size
BATCH_SIZE = 64         # minibatch size
GAMMA = 0.99            # discount factor
TAU = 1e-3              # for soft update of target parameters
LR = 5e-4               # learning rate
UPDATE_EVERY = 4        # how often to update the network
```

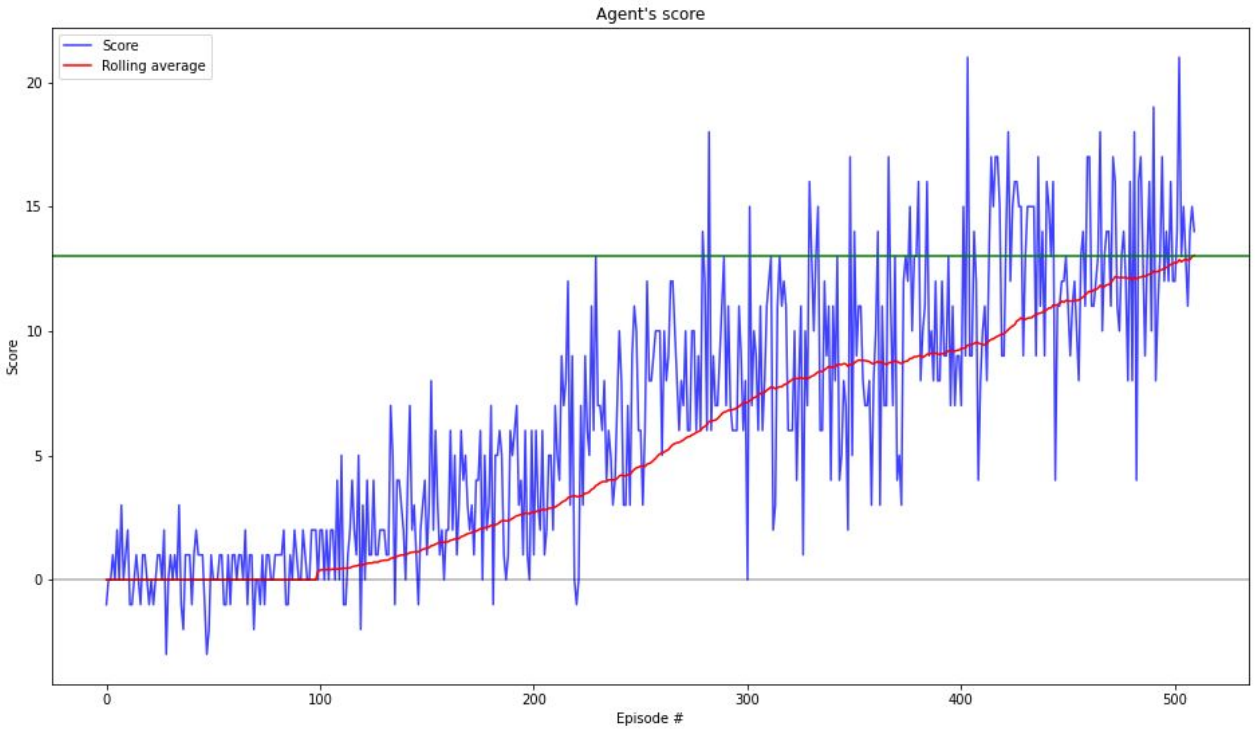
- *BUFFER_SIZE* : Configures the size of the internal *memory buffer.*
- *BATCH_SIZE* : Number of experiences randomly sampled from the *memory buffer*.
- *GAMMA* : Gives the discounted value of future rewards.
- *TAU* : Configures the update of the second network by interpolation.
- *LR* : Configures the learning rate of the network optimizer (Adam).
- *UPDATE_EVERY* : Describes how often to update the network

Network architecture

Some may consider the network architecture a hyperparameter but sometimes it goes beyond that. The network is composed of 3 fully connected layers each one with 100 neurons. The last layer has a linear activation and the previous ones have ReLU (rectified linear unit) activation. The input layer has 37 neurons, one for each state dimension. The output layer has 4 neurons, one for each action value.

Results

The agent was able to solve the problem in 510 episodes. The image below plots the score and the rolling average over the number of episodes.



The rolling average grew not monotonically but quite steadily. The variation of the scores increased too. That may be given by the initial state of the agent on the environment as it improves its performance starting at a better becomes much more profitable. Although the greater outliers shows us how it is not ready to deal with certain scenarios.

Future work

There are multiple possible ways of improving the agent such as:

- Fine tune the hyperparameters;
- Use a more complex Q-based model such a Dueling DQN;
- Improve exploration by using entropy methods instead of epsilon-greedy.

Personally, I would try to create a model capable of reducing the under average outliers. For that I would create a committee of agents where each new agent specializes on scenarios that the previous agents performs very poorly (the under average outliers). This is inspired on the boosting technique for classificators. In this case, I would be bagging experiences instead of input data. Finally, a meta-agent would decide which agent to trust for each scenario or which combination of agents.