Implementation of a Deep Reinforcement Learning Algorithm

This report describes the reinforcement learning algorithm used to train an agent to solve the environment “Banana”, where the goal is to collect as many yellow bananas as possible, while avoiding the collection of blue bananas.

Learning Algorithm

Agent

At the first place an object *Agent* is initialised using the homonymous Class (as defined in the file *dqn_agent.py*). In order to initialise the agent, we pass the state space size (37) and the action space size (4).

Training

The agent is trained using the following steps:

1. the environment is reset. This means we start from a random state s_0
2. the agent takes an action using the Ɛ-greedy policy. This means, it takes a random action with a probability equal to Ɛ, while it chooses the action with the highest Q-value with a probability equal to (1-Ɛ)
3. according to the taken action the environment is set to the new state. A reward is collected (+1 if a yellow banana is collected, -1 if a blue banana is collected, 0 otherwise). At this step we also collect the information whether the episode is finished (done = True)
4. the tuple (state, action, reward, next_state, done), also called *experience,* is used to perform an agent’s step, where it actually learns how to perform better using the following sub-steps:
   i. the *experience* is stored in a fixed-size Buffer (as defined in the class ReplayBuffer)
   ii. if enough experiences are available in the buffer (in our case 64) a random sample of experiences are collected to train the agent
   iii. in order to train the agent two neural networks are used, one which holds the *expected* Q values (Q_network_local) and one which holds the target values (Q_network_target)
   iv. the gradient descent algorithm is applied to the error defined as (expected_values - target_values) in order to get optimal values of the Q_network_local’s weights
   v. every C steps the weights of the target network are set equal to the local network’s weights
5. the scores of every episode are collected and the average of the last 100 episodes is computed
6. when the average score reach +13 the training is interrupted

Neural Networks Architecture

Both for the local and the target network the used architecture is the following:

- one linear layer with input size = 37 and output size=64
- one linear rectifier unit
- one linear layer with input size = 64 and output size=64
- one linear rectifier unit
- one linear layer with input size = 64 and output size=4

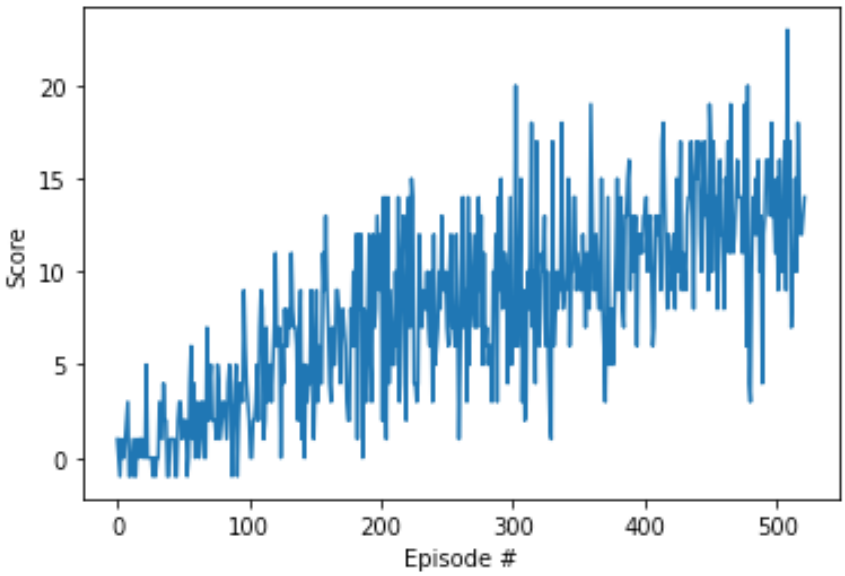
Parameters

Following parameters have been used for the learning algorithm:

$Ɛ_{start}$ = 1, initial value for the epsilon-greedy policy
$Ɛ_{end}$ = 0.01, minimum value for epsilon
$Ɛ_{decay}$ = 0.995, factor to decrease epsilon after the completion of one episode
max_t = 1000, maximum number of time steps in one episode
buffer_size = 100000, number of experience tuples stored in the memory for experience replay
batch_size = 64, number of experience samples randomly extracted from the buffer
gamma = 0.99, discount factor to lower the influence of later rewards

Plot of rewards

The following plot shows the reward obtained in each episode. A clear upward trend is recognisable, meaning the agent is becoming a little bit more intelligent as more episodes are played:



Score vs. Episode Number

The trend of the average score over the previous 100 episodes is reported below:

```
Episode 100     Average Score: 1.67
Episode 200     Average Score: 5.94
Episode 300     Average Score: 8.30
Episode 400     Average Score: 9.81
Episode 500     Average Score: 12.67
Episode 522     Average Score: 13.00
```

The environment was solved after 522 episodes, as the average score equals 13.

Ideas for future works:

There are several ways to improve the training algorithm in order to obtain higher rewards in less episodes:

- optimize the hyper parameters listed above
- Double DQN
- Prioritised Experience Replay
- Dueling DQN