Project 1: Navigation - Report

Introduction

The goal of this project is to train an agent to collect as many yellow bananas as possible while avoiding blue bananas inside a given unity enviroment. The agent is provided with a reward of +1 for collecting a yellow banana and a penalty of -1 for collecting a blue banana. The agent's actions are determined by its current state, which is represented by a 37-dimensional state space that includes the agent's velocity and sensory information about objects in its forward direction. The agent has four possible actions to choose from: moving forward, backward, turning left, or turning right. The task is episodic, and the agent must achieve an average score of +13 over 100 consecutive episodes to successfully solve the environment.

Model

I used a Deep Neural Network (DNN) with 2 hidden layers. Both of them are activated with the relu function, in order to introduce non-linearity into the model. Furthermore, the DNN had a input layer with 37 neurons, matching the state space and an output layer with 4 neurons, matching the action space. The hidden layers consist of 128 neuron each. I started with 1024 neuron for the hidden layer and stepwise reduced the number resulting in better performance of the agent down until 128, which was the lowest number to achieve improvements.

Algorithm

To train the agent I used the DQN algorithm, which is basically Q-Learning (aka. SARSAmax) with a Deep Neural Network to approximate the value function instead of using a Q-Table. DQN is an off-policy TD control algorithm. That means that the policy that is evaluated and improved, differs from the policy that is used to select actions. I also used an experience replay buffer, which stores interactions of the agent with the environment. That allows to randomly sample experiences from this buffer to train the agent. By that, the risk of getting swayed by the effects of correlation of the sequential order of experiences is minimized. Experience Replay also reduces the value learning portion of reinforcement learning to a supervised learning problem. By that, techniques known to supervised learning can be used in a reinforcement learning scenarios. I my case, I used the Mean Square Error to calculate the loss, and applied back propagation to calculate the gradient, which was used by an adam optimizer to update the weights of the DNN. I also used a target and a local network. The ocal network is the main Q-network that is used to select the action and to estimate the Q-values. It is updated at every step using the backpropagation algorithm and the training samples collected by the agent. The target network is a fixed network that is used to estimate the target Q-values in the Bellman equation. The target network is not updated at every step like the local network, but it is updated periodically to stabilize the learning process.

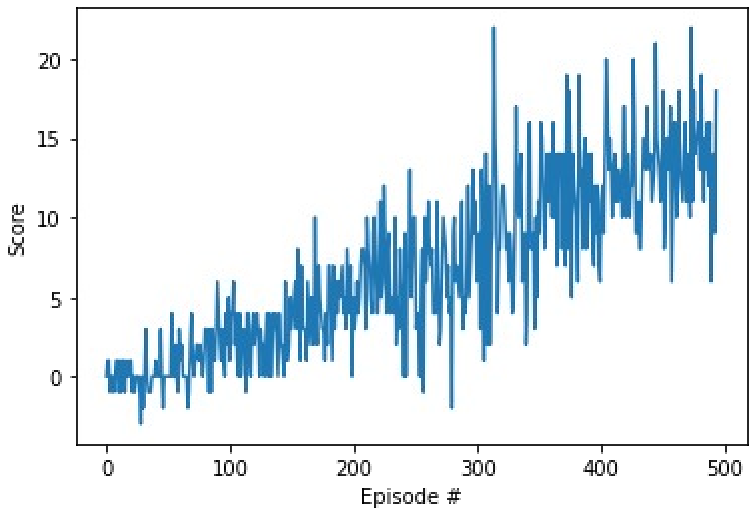
The idea behind using a target network is to decouple the learning process of the Q-network from the changing target Q-values. This helps to reduce the variance of the gradients and to stabilize the learning process. For the implementation I started with the code from the DQN coding exercise. In order to make the code work, I changed the way how to exchange data with the environment, as there a differences between openai-gym environments und unity environments. Futhermore, I changed the model as explained in the previous section. The rest I haven’t changed, as it already produced sufficient results.

The following hyperparameters have been used:

- Replay buffer size: 1e5
- Minibatch size: 64
- Discount factor: 0.99
- Soft update of target parameters: 1e-3
- Learning rate: 5e-4
- Update rate of the target network: 4

Results

With the described network and training algorithm, an average score of 13 over 100 consecutive episodes was reached after 394 episodes as shown in the plot below.



Future Work

For the future, i am planning to improve the DQN with the Rainbow method, which combines different improvements for DQN into one method. As Deep Mind has shown, this combination has the best performance, compared to the different improvements (e.g. Double DQN, A3C, Noisy DQN etc.) on there own.