REPORT

INTRODUCTION

Reinforcement learning has long been thought to be an important tool in achieving human level Artificial Intelligence (AI). While we are still far away from anything remotely like human level AI, the advent of deep learning has significantly improved the performance of traditional reinforcement learning algorithms.

Using a simplified version of the Unity Banana environment, Udacity defined the project objective to design an agent to navigate (and collect bananas!) in a large, square world. 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 agent's observation space is 37 dimensional and the agent's action space is 4 dimensional (forward, backward, turns left, and turn right). The task is episodic, and in order to solve the environment, the agent must get an average score of +13 over 100 consecutive episodes.

Learning Algotithm

I have chosen a Deep-Q-Network (DQN) to solve this environment. In a DQN, a neural net is used as an approximate-function for the values of the state-action pairs. Each of these values represents a reward, which will be reached if the corresponding action is chosen in a particular state. The goal of the task is to maximize the reward. The maximum reward can be reached, by choosing the actions with the maximum values. And these maximum values are reached by improving the approximator function to approximate the value for an action, from which the maximum can be chosen then. As the approximator function is the neural net, an improvement can be reached by training these net using samples of states, the chosen actions and the corresponding reward and then using gradient descent to adjust the weights of the network to better approximate these samples. The gradients in turn are calculated by back propagating the result of a loss function. This loss function calculates the difference between the value of a sample action-state pair given of the actual neural net, and the value of the same sample feed into another neural net. This other neural net, the target network, has the same architecture as the actual neural net. The target network is updated from time to time with the weights of the actual network. This is called fixed-target-network, a method to produce more stable results and let the actual network train better, by not influence itself too much as if used as input and as target value in a loss function. By training the neural net this way, the approximated values for a state-action pair get more precise, and by having precise possible values, the choice of the action with the highest value and therefore highest reward is ensured. Exploration and exploiting for choosing this action is ensured by using an epsilon-greedy strategy by choosing either the proposed action from the neural net, or randomly some of the other actions.

Model:

The model for the neural net consists of an input size of 37, which is the observation space from the environment. These inputs are forwarded through two fully-connected hidden layers, with 74 hidden nodes. The output layer maps these 74 nodes to 4, which is the action size. All layers except the output layer are activated by leaky relu.

This model architecture is used for the local and the target network.

The Hyper parameters of the model are:

```
###Hyperparameter####
hidden_layer1 = 37 * 2
hidden_layer2 = 37 * 2
eps_start = 0.1
eps_decay_rate = 0.995
eps_max_decay_to = 0.01
update_every = 4
buffer_size = 1000000
sample_batch_size = 64
gamma = 0.99
tau = 1e-3
learning_rate = 0.0006
```

hidden_layer1 - Size of the first hidden layer,

Hidden_layer2- Size of the second hidden layer

eps_start- Configures the epsilon for the epsilon-greedy strategy at start of each episode

eps_decay_rate- Configures how much the epsilon should decay after each time step

eps_max_decay_to - Configures a minimum value the epsilon should have, regardless the decay rate.

update_every- Controls how often the weights of the target network should be updated

buffer_size- Configures the maximum size of the replay buffer, older values will be discarded

sample_batch_size- Configures how much samples at each learning step should be pulled from the replay buffer

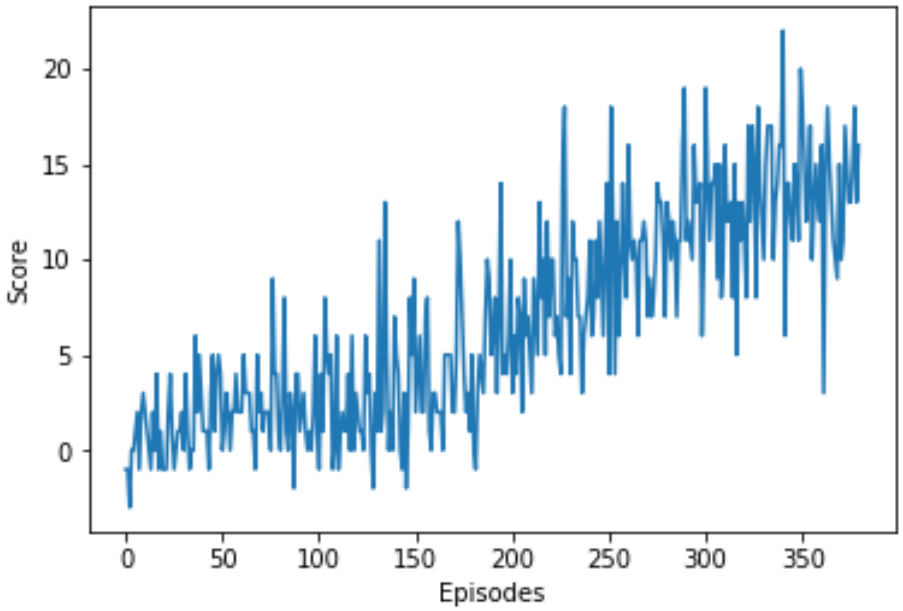
gamma - The factor how much future rewards should be noted in the valuation of the current action

tau - Configures the ratio of how much the target values in the target network should be updated with actual values during update process

learning_rate- learning rate of the optimizer

Results:

The agent has reached a mean reward of 13.06 over the last 100 episodes after episode 380



Future suggestions:

We could test a dueling DQN or Prioritized memory to improve the model efficiency.