Report

DQN

First attempt standard DQN with the set of params.
The algorithm is taken from the DQN exercise and then modified.

The basic implementation follow the original paper: https://arxiv.org/pdf/1312.5602.pdf

The model used consists of a neural network with two hidden layers respectively with 64 and 32 units. It is bigger than what is necessary and the reason is that a bigger network helps with the problem of moving targets.
To help with this problem the target network is also implemented.
To help with the problem of having experience samples not identically distributed a simply replay buffer is implemented.

How the model works:

1. The observation comes from the environment. In the project we use the full state made of 37 dimensions.
2. We select an action following the epsilon greedy strategy. The neural network provides q values for each action possible in the current state. Then we follow the EpsilonGreedy strategy.
3. We use the action to obtain another state and the reward
4. At this point we have everything to perform a SARSA update.
    1. We add the sample to the memory
    2. If we have enough samples in the memory to get a batch
    3. We sample randomly from the memory
    4. Compute the Q values for next states and the expected ones
    5. We compute the loss and perform a step to update the weights
    6. We also perform a soft update to move the target network weights closes to the online model
5. We repeat this for each tilmestep in the episode

I used the RMSProp optimiser with LR 0.0005.
Exploration strategy EpsilonGreedy with epsilon start of 1.0 and final 0.01 with a decay of 0.995

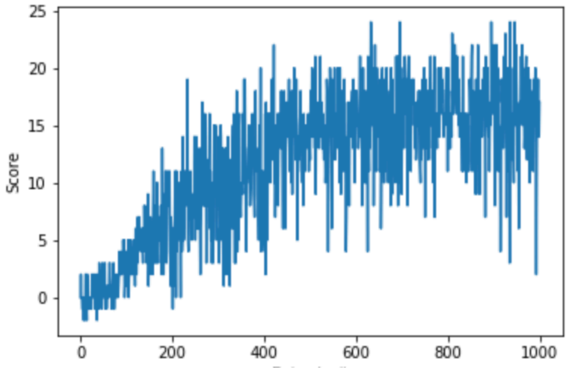
Other parameters (standard worked well):

```
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
```

I train the agent for 1000 episodes and each episode made of 500 time steps

With this setup the agent is able to complete the task with an average reward over 100 episodes greater than 13.0 in 494 episodes.

```
Episode 100	Average Score: 0.79
Episode 200	Average Score: 5.10
Episode 300	Average Score: 8.32
Episode 400	Average Score: 10.01
Episode 494	Average Score: 13.01
Environment solved in 494 episodes!	Average Score: 13.01
Episode 500	Average Score: 13.27
Episode 600	Average Score: 14.71
Episode 700	Average Score: 15.21
Episode 800	Average Score: 16.06
Episode 900	Average Score: 16.24
Episode 1000	Average Score: 15.52
```



Improvements:

1. Double DQN

The first improvement was to use a double DQN following the paper: https://arxiv.org/pdf/1509.06461.pdf

This helps in mitigating the overestimation of action - value function of DQN decoupling action selection from action evaluation.

The improvement consists in:
1. get the index of the best action from the online network
2. use the q value from the target network to select the Q targets
3. Compute the Q expected and compute the loss

2. Dueling architecture

Another improvement is shown in the paper: https://arxiv.org/pdf/1511.06581.pdf
This improvement uses the fact that actions of a state are correlated and when we learn from one we can use the information to learn something about the other actions of the same state.

This improvement touches the neural network architecture.
Instead of providing Q values for all actions as output we split the network output in two.
1. Value of the state
2. Advantage for each action

We use the relation between value function and advantage function to generate the Q function returned as output.

3. Prioritized Experience Replay

We use experience replay because we need to have identically distributed samples across experience.
But we also know some samples provide more information then other as explained in this paper:
https://arxiv.org/pdf/1511.05952.pdf
We are going to prioritize the samples from the information they contain. The more information they contain the more learning opportunity there is of using the sample.
The way of understanding how much information is contained in the sample is the TD error generated by using that sample. The absolute error is used to set the priority.

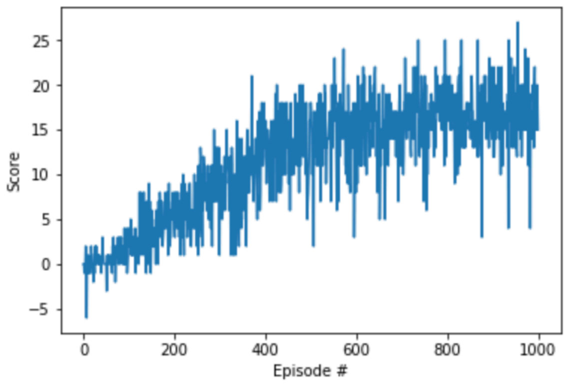
Note: the implementation is inspired from the book: Gokking Deep Reinforcement Learning (https://www.manning.com/books/grokking-deep-learning)

Parameters (standard from the book):
alpha=0.6,
beta0=0.4,
beta_rate=0.99992

Epsilon: 0.5 # used for weighted importance-sampling

We train for the same amount of time and episodes as before.
The agent is able to win in 477 episodes. The learning is more stable and the score higher.

```
Episode 100	Average Score: 0.51
Episode 200	Average Score: 3.85
Episode 300	Average Score: 6.92
Episode 400	Average Score: 9.79
Episode 477	Average Score: 13.07
Environment solved in 477 episodes!	Average Score: 13.07
Episode 500	Average Score: 13.21
Episode 600	Average Score: 14.37
Episode 700	Average Score: 15.23
Episode 800	Average Score: 16.50
Episode 900	Average Score: 16.40
Episode 1000	Average Score: 16.87
```



Further improvements:

There are several improvements possible and i plan to work on them later.
1. Noisy DQN: https://arxiv.org/pdf/1706.10295.pdf
2. Distributional Q-Learning: https://arxiv.org/pdf/1707.06887.pdf
3. Asyncronous Learning: https://arxiv.org/pdf/1602.01783.pdf

We can also improve the Replay Buffer using the Segment Tree as shown in the OpenAI baseline:
https://github.com/openai/baselines/tree/master/baselines

DQN from raw pixels:

I provided the same implementation of the improved DQN to work with visual observations.
The main difference are:

1. Observations reshaped form (1,3,84,84) to (3,84,84)
2. Stacked frames input(12,84,84)
3. Convolutional Neural Netwrok

CNN architecture:

The input is passed through 3 convolution layers with 32, 64, 64 filter.
After the convolutions there is a fully connected layer and from there the standard Dueling Architecture is followed.

There are several improvements that could be done.
As example skipping frames.

Unfortunately i have no GPUs and i was not able to try this version. It also doesn't work with the Udacity Workspace.
I tried using Google Colab but so far it did not work. I decided to move to Atari Games.