Project Navigation: Report

This report describes the techniques used and the ideas behind the chosen methods. It will not repeat the content of the ReadMe file.

Intro

The Reinforcement Learning (RL) problem posed by the Banana world, from a RL-agents perspective is defined by environment.

Environment

The environment in turn is defined by its interaction with the agent. Therefore by its state space, its action space and the reward that is returned when acting on the environment and thereby moving a time step ahead.

Our task is episodic, therefore the environment needs to return to the agent, whether an episode has passed or not. If an episode has passed, the environment has to reset to its original state.

Specifications:

State space dimension: 37 continuous features
Action space dimension: 4 discrete actions
Rewards: Collect yellow banana: +1, Collect blue banana: -1
Episode over: done = 1, else 0

Model

Instead of a Q-table, a Q-network is used as knowledge storage. This Q-network is a deep neural network.

Why fully connected layers?

I chose to use a neural network with 3 hidden layers. These 3 hidden layers are fully connected layers. The idea behind choosing fully connected layers is, that the input features are not related. (In contrast to the "Learning from Pixels" task)

In the following I will describe the neural network from input layer to output layer.

Input layer

The input layer is a 1-d array of length 37. Each input feature is represented as an element of this array.

Fully connected layer 1

The first hidden layer is a fully connected layer with 256 neurons. We increase the number of dimensions in order to be able to find hidden meta information, that lies in the combination of the input features.

RELU activation function

This layer is succeeded by a rectified Linear Unit (RELU) activation function, that is applied on each neuron. This activation function maps the output of the neuron to max(0,output).

Fully connected layer 2

We then gradually decrease the number of dimensions to intensify the physical meaning of the dimensions(neurons).

The second hidden layer is a fully connected layer with 128 neurons. This layer is succeeded by another RELU activation function.

Fully connected layer 3: Output layer

We further decrease the dimensions to concentrate the meaning in the output neurons.

The third hidden layer is a fully connected layer with 64 neurons. This layer outputs 4 dimensions. They represent the 4 available actions.

No softmax activation function

We do not use a softmax activation function, as we would want to compute the error from the logits.

Code

The model is defined in **model.py**

Agent

The agent implements the necessary functions it requires to perform RL with an environment. The agent itself can be used on different environments. Therefore its functions are general to all environments.

An RL agent needs to implement these functions:

- Choose action
- Update the knowledge

As this agent implements the advanced techniques of “Experience Replay” and “Fixed Q-targets” some additional functionality is required:

- Append experience to replay buffer (Experience Replay)
- Update the knowledge of the target network (Fixed Q-targets)

Agent functions

The agent is implemented in dqn_agent.py. The following explanations are regarding the functions in that file:

- Act(): Chooses an action epsilon greedy, to allow for less greedy action selection in the beginning of learning, therefore high exploration, by random action selection. Increasing greedy action selection over the course of learning and decreasing exploration.
- Step(): Stores a State-Action-Reward-Next_State (SARS) tuple in the Replay buffer, that was created by acting on the environment.

- Learn(): Uses Deep Learning to learn from a batch of experiences from the Replay Buffer and update the knowledge (Q-network)
- Soft_update(): As the error function for the Deep Learning task is computed using the "Fixed Q-targets" technique, a target Q-network exists. This Q-network is updated, by simply copying the weights and biases of the original Q-network, that was trained (after the training batch has passed)

Experience Replay: Replay Buffer

The idea behind experience replay is, to keep SARS tuples in memory. This makes sense, because these tuples are not affected training the agent. The same action in the same state will always yield the same reward, no matter how smart the agent is. Therefore keeping these tuples in memory is useful, as they can be used at a later point in time.

Fixed Q-targets

This removes the risk of learning a correlation of weight adjustment to the output. While learning adjusting weights can be interpreted as a correlation to the error. This will lead to an increasing or oscillating error function. It is countered by introducing a second set of weights (second Q-network) similar to the original one. The weights of this Q-network are only used to compute the error, and are not changed in the course of learning.

This second "target" Q-network is only updated once a learning batch from the Replay Buffer has passed the original network.

Learning Algorithm

The learning algorithm is similar to the one of a RL agent. The main difference is that the discrete Q-table is replaced by a Q-network.

Code in Jupyter notebook. Function "dqn()"

First loop:

This algorithm iterates over the number of episodes specified. In each episode the environment is reset to an initial state.

Second loop:

In the first loop: While the environment does not indicate that the episode is over, the following is done:

- Agent selects an action: act()
- Action is acted onto environment: env.step()
- The environment passes the new state, along with the reward, and whether the episode is over
    - This SARS tuple is written to the replay buffer: agent.step()
- Current score is updated with the reward received

- Epsilon for the epsilon greedy action selection is decreased, by multiplying by 0.999

The agent internally decides when to learn. It does so, once it has enough experiences collected in the Replay Buffer.

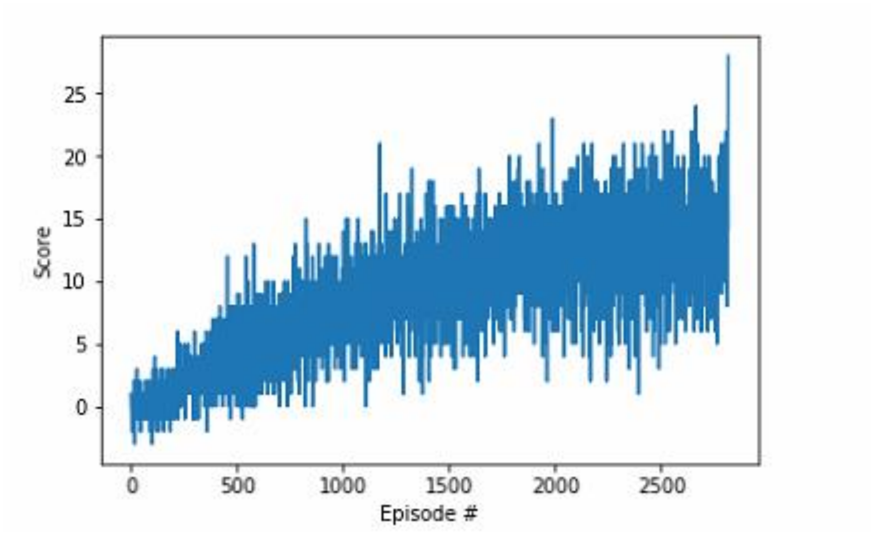
Hyperparameters

The following hyperparameters are used:

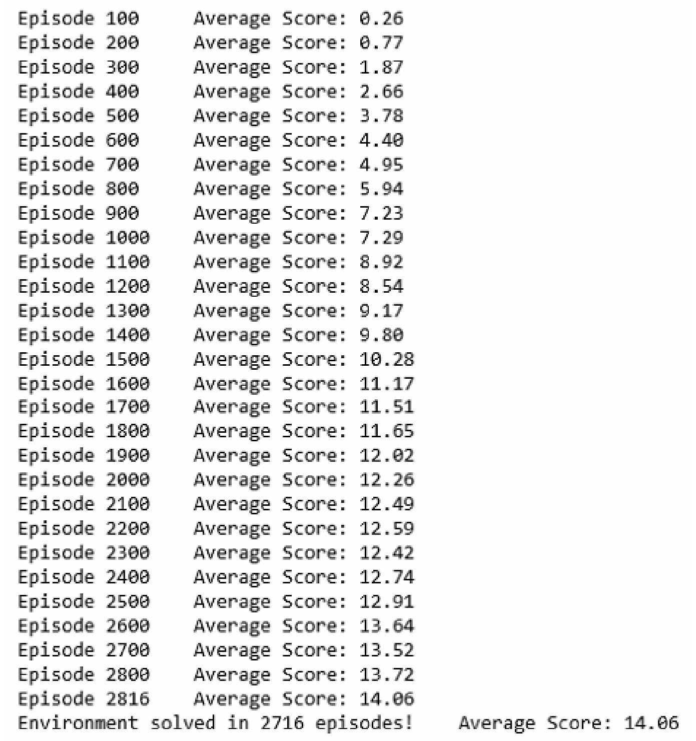
- Epsilon start: 1.0
- Epsilon min: 0.01
- Epsilon decay rate: 0.999 (Epsilon greedy action selection)
- Replay buffer Size: 100000 (Experience Replay)
- Batch Size for learning: 64 (Experience Replay)
- Gamma: 0.99 (for discounting)
- Tau: 0.001 (Fixed Q-Targets: update parameter for interpolating original Q-network and target Q-network, when writing updated weights to target Q-network)
- Learning Rate for Adam optimizer: 0.0005
- UPDATE_EVERY: 4 (number of experiences after which agents checks if enough experiences are in the Replay Buffer to learn(update Q-network))

Result

The rewards per episode evolved in the following way:



After an estimated 2550 episodes the average score per episode was above +13 and remained there.



```
Episode 100     Average Score: 0.26
Episode 200     Average Score: 0.77
Episode 300     Average Score: 1.87
Episode 400     Average Score: 2.66
Episode 500     Average Score: 3.78
Episode 600     Average Score: 4.40
Episode 700     Average Score: 4.95
Episode 800     Average Score: 5.94
Episode 900     Average Score: 7.23
Episode 1000    Average Score: 7.29
Episode 1100    Average Score: 8.92
Episode 1200    Average Score: 8.54
Episode 1300    Average Score: 9.17
Episode 1400    Average Score: 9.80
Episode 1500    Average Score: 10.28
Episode 1600    Average Score: 11.17
Episode 1700    Average Score: 11.51
Episode 1800    Average Score: 11.65
Episode 1900    Average Score: 12.02
Episode 2000    Average Score: 12.26
Episode 2100    Average Score: 12.49
Episode 2200    Average Score: 12.59
Episode 2300    Average Score: 12.42
Episode 2400    Average Score: 12.74
Episode 2500    Average Score: 12.91
Episode 2600    Average Score: 13.64
Episode 2700    Average Score: 13.52
Episode 2800    Average Score: 13.72
Episode 2816    Average Score: 14.06
Environment solved in 2716 episodes!    Average Score: 14.06
```

This is how the score evolved over the number of episodes. Training was stopped once it surpassed +14. With this model and hyper parameters, the upper limit for learning seemed to be around +16:

```
Episode 1900 Average Score: 12.16 Episode 2000 Average Score: 12.82
Episode 2100 Average Score: 12.90 Episode 2200 Average Score: 13.07
Episode 2300 Average Score: 13.27 Episode 2400 Average Score: 13.87
Episode 2500 Average Score: 14.34 Episode 2600 Average Score: 13.58
Episode 2700 Average Score: 12.37 Episode 2800 Average Score: 13.46
Episode 2900 Average Score: 13.52 Episode 3000 Average Score: 14.29
Episode 3100 Average Score: 15.22 Episode 3200 Average Score: 14.79
Episode 3300 Average Score: 15.08 Episode 3400 Average Score: 14.41
Episode 3500 Average Score: 15.15 Episode 3600 Average Score: 15.08
Episode 3700 Average Score: 14.84 Episode 3800 Average Score: 15.99
Episode 3900 Average Score: 15.07 Episode 4000 Average Score: 15.27
Episode 4100 Average Score: 14.93 Episode 4200 Average Score: 14.84
Episode 4300 Average Score: 14.22 Episode 4400 Average Score: 14.40
Episode 4500 Average Score: 14.49 Episode 4600 Average Score: 15.28
Episode 4700 Average Score: 15.17 Episode 4800 Average Score: 15.40
Episode 4900 Average Score: 16.34 Episode 5000 Average Score: 16.13
Episode 5100 Average Score: 16.77 Episode 5200 Average Score: 15.70
Episode 5300 Average Score: 14.92 Episode 5400 Average Score: 15.59
Episode 5500 Average Score: 15.44 Episode 5600 Average Score: 15.15
Episode 5700 Average Score: 15.47 Episode 5800 Average Score: 15.31
Episode 5900 Average Score: 15.81 Episode 6000 Average Score: 15.43
Episode 6100 Average Score: 16.02 Episode 6200 Average Score: 15.19
Episode 6300 Average Score: 15.08 Episode 6400 Average Score: 15.49
Episode 6500 Average Score: 14.28 Episode 6600 Average Score: 15.82
Episode 6700 Average Score: 15.17 Episode 6800 Average Score: 14.54
Episode 6900 Average Score: 14.95 Episode 7000 Average Score: 14.18
Episode 7100 Average Score: 15.39 Episode 7200 Average Score: 15.87
Episode 7300 Average Score: 15.54 Episode 7400 Average Score: 15.48
Episode 7500 Average Score: 16.44 Episode 7600 Average Score: 15.66
Episode 7700 Average Score: 15.33 Episode 7800 Average Score: 15.89
Episode 7900 Average Score: 16.12 Episode 8000 Average Score: 16.33
Episode 8100 Average Score: 16.02 Episode 8200 Average Score: 14.94
Episode 8300 Average Score: 15.36 Episode 8400 Average Score: 14.84
Episode 8500 Average Score: 14.85 Episode 8600 Average Score: 15.23
Episode 8700 Average Score: 15.95 Episode 8800 Average Score: 15.59
Episode 8900 Average Score: 15.74 Episode 9000 Average Score: 15.07
Episode 9100 Average Score: 16.15 Episode 9200 Average Score: 15.93
Episode 9300 Average Score: 15.95 Episode 9400 Average Score: 16.03
Episode 9500 Average Score: 16.04
```

Highscore

After 5100 episodes an average score of 16.77 was reached.

Enhancements

The result could possibly be enhanced by applying the advanced techniques of

- Prioritized Experience Replay
- Dueling DQN

Also a less complex neural network, possibly with one layer less, would speed up reaching the score of +13. The currently used network overperforms. Therefore complexity can be reduced, as this high performance is not required.

Learning from Pixels

I did not yet cover the “Learning from Pixels” task in this report.

I put large hope in the use of convolutional neural networks instead of Multi Layered Perceptrons (MLP) for this task, as this task relies on image data. Pixels in images are related location wise. Therefore it makes sense to use convolutional neural networks, that make use of this location information, instead of MLPs, which loose this information.