Report on Results

Deep Reinforcement Learning

Project 1: Navigation

David Neuhäuser

1 Framework

In the current project, the goal was to train an agent navigating in a two-dimensional box and collect bananas. To be more precise, the agent can meet with two different types of bananas, yellow and blue. A reward of $R_t = 1$ is provided for getting a yellow banana, and a reward $R_t = -1$ is provided for meeting a blue banana. Thus, in order to optimize the total score, the aim of the agent is to collect as many yellow bananas as possible while avoiding blue bananas. The state space $\mathcal{S}$ has 37 dimensions and contains the agent's velocity, along with ray-based perception of objects around agent's forward direction. Based on this information, the agent has to learn how to select actions in the best way. Four discrete movements can be made, corresponding to

- 0 - move forward
- 1 - move backward
- 2 - move left
- 3 - move right.

Thus, the action space is given by $\mathcal{A} = \{0, 1, 2, 3\}$. The task is episodic, i.e. $t \in [0, 1, \ldots, T]$. In order to solve the environment, the agent must get an average score of +13 over 100 consecutive episodes e_i, i.e. $\frac{1}{100(T+1)} \sum_{\{e_{j_1}, \ldots, e_{j_{100}}\}} \sum_{t=0}^{T} R_t = 13$.

2 Detailed Description of the Algorithm

The algorithm which is used to train the agent for solving the task is the deep Q-learning network algorithm. It is one of the so-called Temporal Difference Methods (others are for instance SARSA or expected SARSA) and has the advantage of updating after every

action a_t, contrary to Monte-Carlo prediction which needs the entire episode before updating. In principle, the update step is given by

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(r_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a) - Q(s_t, a_t) \right),$$

where α denotes the learning rate. Instead of a Q-table used for estimating the optimal action-value function $q_*(s, a)$ in context of discrete state spaces $\mathcal{S}$, the idea in our context with continuous $\mathcal{S}$ is estimating the action-value function via (deep) neural networks. In the provided solution, two different network architectures with respect to the hidden layers were used. In the first one, we have two hidden layers each with 64 units. In the second one, we chose three hidden layers where the first one has 128 units and the the other two have 64 units, respectively. This is just to investigate the influence of the network's architecture on the performance of the agent. Rectified linear units where used as activation functions except for the last layer in the network. In order to deal with the problems that come along with this kind of training the agent, two further ideas were implemented:

- experience replay: in order to de-correlate the state-action-reward-next state tuples $(s_t, a_t, r_{t+1}, s_{t+1})$, $(s_{t+1}, a_{t+1}, r_{t+2}, s_{t+2}), \ldots$, it is useful to create a replay buffer of certain size. Then sample at random from this buffer leading to (at least close to) i.i.d. samples instead of learning from tuples as they come over time.
- fixed Q-targets: fix the weights of the neural net in the target $\widehat{Q}$. Of course, this leads to the consequence that we need two neural networks, one target network and one online network. However, we can get rid of instatibility issues when updating the model weights.

In the following table, you can find the hyperparameters for the neural network architecture used for estimating the action-value function $q_\pi(s, a)$:

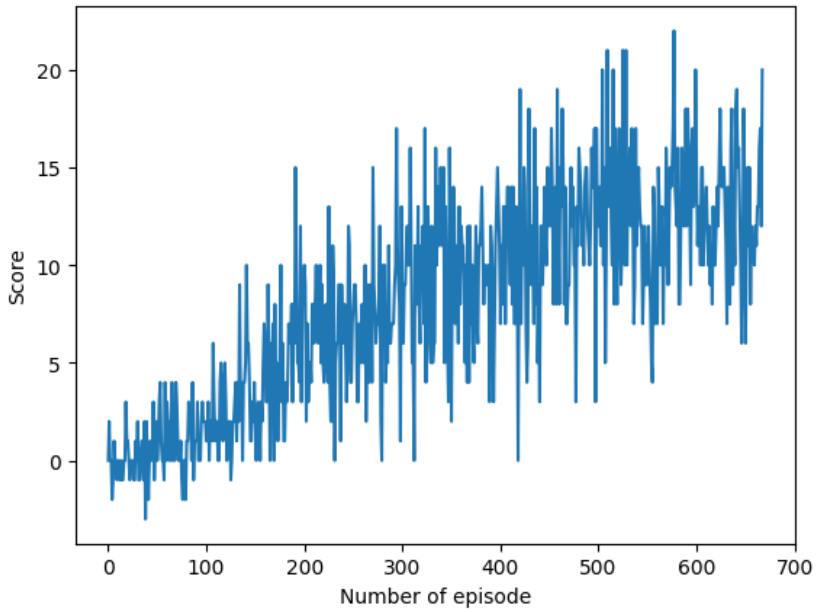
Hyperparameter	Value
size of second layer (first hidden!) in QNetwork_2Hidden	64
size of third layer (second hidden!) in QNetwork_2Hidden	64
size of second layer (first hidden!) in QNetwork_3Hidden	128
size of third layer (second hidden!) in QNetwork_3Hidden	64
size of fourth layer (third hidden!) in QNetwork_3Hidden	64

In the next table, all hyperparameters which were used for the deep Q-network learning algorithm are listed:

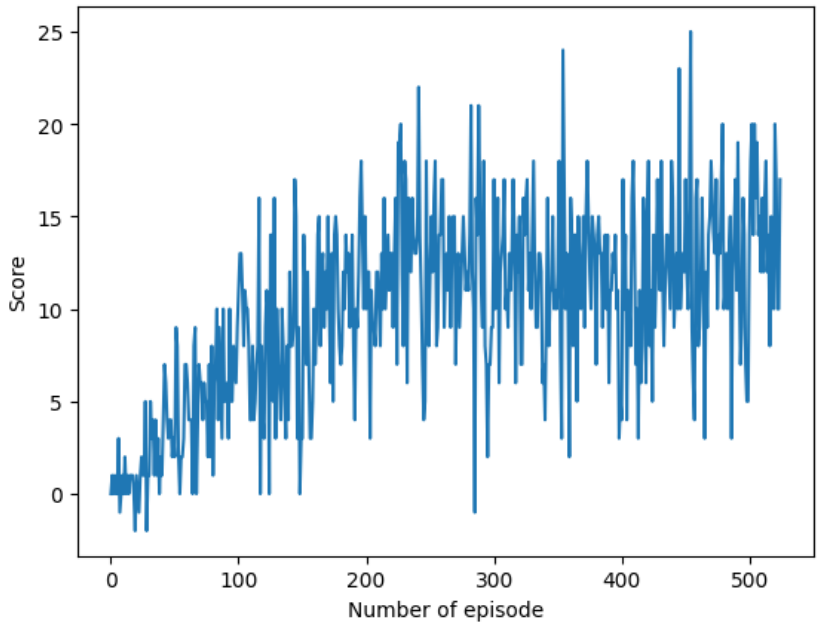
Hyperparameter	Value
replay buffer size	10000
batch size	64
discount factor γ	0.9
soft update parameter τ	0.001
learning rate	0.0005
update interval	4
number of episodes	5000
maximum number of timesteps per episode	2000
starting value of epsilon	1.0
minimum value of epsilon	0.1
multiplicative factor to decrease epsilon in QNetwork_2Hidden	0.995
multiplicative factor to decrease epsilon in QNetwork_3Hidden	0.990

3 Results

The performance of the agent via 2 hidden layers in the Q-network was as follows: The environment was solved in 568 episodes with anverage score of 13.05 over the last 100 episodes, also see the following figure.



The performance of the agent via 3 hidden layers in the Q-network was as follows: The environment was solved in 425 episodes with anverage score of 13.08 over the last 100 episodes, also see the following figure.



However, training results for the network with three hidden layers are relatively unstable and vary strongly from training to training. As a consequence, high efforts of changing the fundamental structure of the neural net do not seem to eventually bring performance increase. Instead, we should focus on the ideas of the next section in future.

4 Ideas for Future Work

As already mentioned in the previous section, the agent is trained via deep Q-learning network algorithm including experience replay and fixed Q-targets. In order to improve the agent's performance, several possibilities are thinkable:

- Architecture of the neural networks seems to have minor impact; anyways, it could be worth to adapt the neural net correspondingly
- Hyperparameter search: so far, all the hyperparameters listed in the previous section were not varied too much during the project. A systematic search for optimal parameters could significantly boost the results.

The most important ideas for improvement are however the following ones:

- Usage of double deep Q-networks: one network for choosing the best action, one for evaluating that action; or even use dueling deep Q-networks.

- Prioritized experience replay: instead of uniformly sampling from the replay buffer, there should be some kind of importance sampling, taking more relevant tuples with higher probability than minor important ones.