Report Udacity Navigation

1 Introduction

In this report I wll describe the implemenation for the P1 Navigation project for the Udacity Nanodegree in Deep Reinforcement Learning(RL). The Algorithm itself will be explained as well as the choice of hyperparameters. The rewards of the successful agent will be documented and possible future improvement will be discussed.

2 The DDQN-Algorithm

The algorithm is a Double Deep Q-Network algorithm with (prioritized) experience replay((P)ER). It features reinforcement learning with a neural network to approximate the Q-Value-Function, which is updated via reinforcement learning to optimize the agents performance.

2.1 Backpropagation Neural Net

A backpropagation neural network(BNN) is used to approximate the Q-action-value-function for a given state. It contains 4 layers: input layer, 2 hidden layers, and the outputlayer. The input layer consists of 37 neurons which take the given states as input. The states are continous real numbers. It connects these input neurons to the first hidden layer with 64 neurons featuring ReLu-activation-functions. The second hidden layer also consists of 64 neurons with the same activation functions and is connected to the output layer of 4 neurons which gives the Q-Values for each possible action.

2.2 Q-Learning-Agent

The agent features double deep Q reinforcement learning, using (prioritized) experience replay. On each time step within the episode it supplies the best action according to the approximated Q-values from the BNN and probabilities from deciding ε-greedy. As it is a double DQN-algorithm it decides on the action according to qnetwork_local, which is updated immediately. The other network qnetwork_target is only updated at the end to the time step with an associated dampening factor tau.

On every 4th step the agent samples batch_size elements from the replay buffer and learns from that batch updating directly the qnetwork_local and only with the described lag this update is forwarded to qnetwork_target.

2.3 Prioritized ER(own 'improved' implementation)

I implemented a prioritized experience replay buffer, but all results showed a tendency to be supposedly worse than the standard. Hopefully the PER was implemented properly, hoping for that, I would suggest that PER might not be helpful in this environment. I implemented my own version of the PER to reduce complexity in a way to remove n. Instead of being O(n) or with a binary heap O(log(n)) the complexity in my version is not directly dependent on n. I proceed as follows with element i with $P(i) = \frac{p_i}{\sum p_j}$ as in the lesson:

1. Draw an element i from PER $\rightarrow p_d = \frac{1}{n}$
2. Accept the draw with $p_a = \frac{nP(i)}{PMULT}$

Note that $mean(P(i)) = \frac{1}{n}$ and first assume that $PMULT = 1$ and that we only draw $P(i) < \frac{1}{n}$. Hence $p_a = nP(i)$ and hence the probability that we will draw and accept element i is $p_a p_d = P(i)$.d So far exactly what we would like. If $P(i) \geq \frac{1}{n}$ it would be that $p_a \geq 1$. But the probability to accept cannot be larger 1. Thus I introduced PMULT, to ensure that p_a is a probability, but p_a will be reduced proportionally for all $P(i)$ so that we keep the same distribution and just need to draw around $PMULT$ times more than before, but keep the same distribution. I keep track that $PMULT$ should be larger than $max(\frac{P(i)}{n})$. This removes n from the complexity and apart from the formula, the need to control for $max(P(i))$, and acceptance of draws, it saves us the work of implementing a binary heap which would on top even have the supposedly worse performance of O(log(n)).

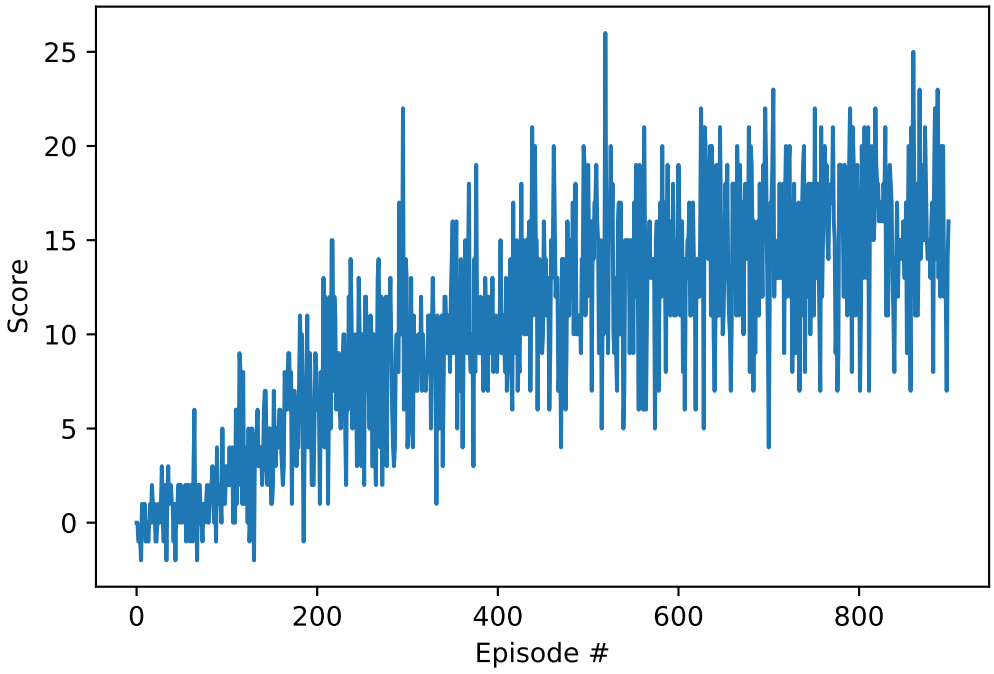
2.4 Choice Hyperparameters

I trained on various combinations of hyperparameters which are documented in *ParameterSearch.xlsx* and decided for hyperparameters which showed the best performance in these runs. These are:

Name	Value	Description
γ	0.98	discount factor
τ	0.001	Dampening factor for updating qnetwork_target
LR	0.0005	Learning Rate for BNN
batch_size	64	Batchsize of experiences drawn from replay buffer
buffer_size	100000	Number of elements in replay buffer
Prioritized ER		
α	0.5	Exponent on weights for probabilities
β	0.6	Exponent on IS weights
Decaying ε		
$\varepsilon - start$	1	Start value for ε
$\varepsilon - end$	0.01	End value for ε
$\varepsilon - decay$	0.995	Decay factor for ε

3 Agents Performance

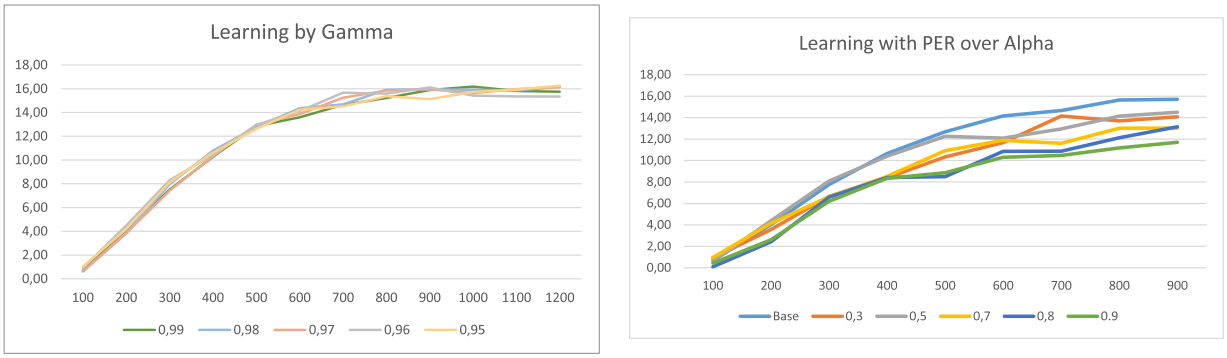
With the given hyperparameters the agent reached the average of 13 over 100 episodes usually between period 500 and 600. The following plot displays the agent scores over 900 epsiodes:



The agents scores usually became satiated between 14-16 and only fluctuated around in this interval also for longer runs over 2000+ episodes. Other parameter configurations resulted

mostly in similar or worse performance on average. Meaning learning up to 13 or 15 was slower or the score of 15 was not reached at all.

Figure 1: Scores for Gamma and PER



These plots display average scores over 100 episodes for agents desribed above as averaged over 3 training runs. Left: For different Gammas the scores show virtually no differences. Right: Compared to the base scenario with uniform ER all parameter choices for alpha resulted in worse performance for PER. More episodes did not change these outcomes.

Varying γ between 0.95 and 0.99 in uniform ER showed little difference, smaller values had tendencies for lower performances. These outcomes are shown in figure 1 left plot. Trying different values for α in PER only showed slower learning than the uniform ER as shown in figure 1 right plot. Further training scores are documented in *ParameterSearch.xslx*.

4 Discussion for future Improvements

For further improvements I would first suggest to further exploration of hyperparameters by extensive training runs over different values. This could include further tests of fixed values but also different paths of hyperparameters to increase or decrease over time. Different values could be interesting for the early part of raising scores to 10 and above, while other values could be more relevant for the later periods to stabilize values in the range of 15 or 16. Furthermore different setups for the BNN could be explored, more layers or more neurons per layer might improve on precision, to improve on later performance when scores get satiated. Or maybe later fluctuations in scores are also caused by imprecisions in Q-value approximation which might be mitigated by changed the structure of the BNN. I noticed that there are still quite significant changes in Q-values in the later satiation period. This could indicate some instabilities in the network. A closer exploration with which states these larger changes in Q-values are associated might help to detect problems here. Further exploration of the structure of the Q-function might yield further insides on problematic regions. It might be interestiing to use t-sne to visualize its structure. On the method side implementing Dueling DQN could be another option to improve performance.