Project: Navigation
Udacity Deep Reinforcement Learning

Bastian Broecker

Abstract. *This document shows the reinforcement learning approach appied in the first project "Navigation" (Value-Based Methods) as part of the udacity nanodegree: Deep Reinforcement Learning. The approaches used during this project are based on a Deep Q-Network (DQN) introduced in the paper [Mnih et al. 2013], which is responsible for estimating the quality (Q-value) of a state-action pair in a given environment. During the project I tested and compared different variations and additions to the basic DQN e.g. Double DQN, Dueling DQN, priority replay buffers and frame skipping.*

1. Project Environment and Goal Description

This is a short recap of the project's environment and goal we have to achieve. During this project we have to train an agent to navigate in a large square, whilst navigating the agent has to pick up yellow bananas and avoid blue bananas (see Figure 1). Each picked yellow banana provides a reward of +1 to the agent, a pick up of a blue banana deducts -1 from the agent's total reward.

The state space is continues and has 37 dimensions, this includes the agent's velocity and a ray based perception, indicating distances to bananas. The agent has four discrete actions for navigation:

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

The task is episodic and terminates after 300 time steps. In order to solve the task the agent has to average 13+ points over the last 100 episodes.

2. Value Estimation Networks

The approaches used during this project, are based on the idea of DQNs introduced by [Mnih et al. 2013]. The following section gives a brief overview over the DQN and variations applied in this project. Starting with the vanilla DQN in Section 2.1, going to the Double DQN in Section 2.2 and the Dueling DQN in Section 2.3. Each section describes the network and the hyper-parameters used during training.

2.1. DQN

Similar to other reinforcement approaches, DQNs are trying to estimate the optimal state-action value function $Q^*(s, a)$, which quantifies the expected "quality" (Q-value) when taken an action a in a given state s. During these approaches agents are constantly exploring the state-action space, while iteratively updating the Q-value by applying the Bellman-equation:

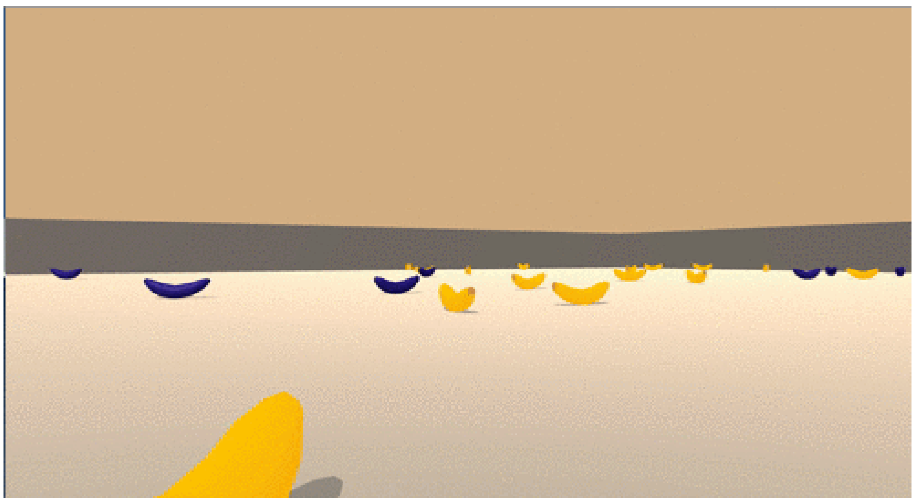
Figure 1. Environment

$$Q^*(s,a) = E[r + \gamma \max_{a'} Q^*(s',a')|s,a], \tag{1}$$

where s' and a' define the following state and action, r defines the reward given by the environment and γ represent discount factor for the estimated future rewards. DQNs use deep-neural-networks as non-linear function approximator, formulized by $Q(s,a;\theta) \approx Q^*(s,a)$, where θ defines the parameters/weights of the network. The weights θ are then updated by derivating the estimation error function in respect to the current weights. The update-rule, based on the Bellman-equation, is thereby formulated as:

$$\Delta\theta = lr \cdot \overbrace{(\underbrace{r + \gamma \max_{a'} Q(s',a',\theta)}_{\text{target}} - \underbrace{Q(s,a,\theta)}_{\text{old value}})}^{\text{error}} \nabla_\theta Q(s,a,\theta), \tag{2}$$

with $lr \in (0,1)$ as the learning rate in which the weight are adjusted. The update shown in Equation 2 has one issue, the target estimation is changing since the network is constantly updated and this can affect the training. In order to overcome this problem [Mnih et al. 2013] introduced a second target network with the weights θ^- where the weights are kept fixed, they are only copied from θ to θ^- after a certain amount of time-steps, this makes the training more stable. The updated rule can be seen in Equation 3.

$$\Delta\theta = lr \cdot \overbrace{(\underbrace{r + \gamma \max_{a} Q(s',a',\theta^-)}_{\text{target}} - \underbrace{Q(s,a,\theta)}_{\text{old value}})}^{\text{error}} \nabla_\theta Q(s,a,\theta), \tag{3}$$

Similar to this idea I apply a so called soft-update, where the weights of the target network are constantly updated, but with a much slower rate. The factor which defines the rate of the update is defines as τ and the update-rule is defined in Equation 4:

$$\theta^- = \tau\theta + (1-\tau)\theta^- \tag{4}$$

The last mechanism introduced in the DQN-paper was the use of a replay buffer, which stores the experiences $e_t = (s_t, a_t, r_t, s_{t+1})$ of each time-step t inside a fixed sized data set. During training the agent will draw a fixed size of samples (batch) from the buffer and will update the network weights according to the update-rule 3. In order to break correlation between consecutive samples [Mnih et al. 2013] uses a random sampling to assure a higher variance in the samples.

The hyperparameters used for training are derived from empirical testing and are shown in Table 1. The network itself has two hidden fully connected layers with 64 nodes and ReLu activation functions, the output layer is linear:

- Fully connected layer: input: 37 (state size) output: 64, activation: ReLu
- Fully connected layer: input: 64 output: 64, activation: ReLu
- Fully connected layer: input: 64 output: 4 (action space), activation: Linear

The training performance and comparison with the other DQN variations is illustrated in Section 5

Parameter	Value	Description
γ	0.95	Factor for reduction of future rewards (see Bellman Equation 1)
τ	1e-4	Update rate for adjusting target network (see Equation 4)
lr	3e-4	Learning rate used by the optimizer adjusting the network weights according to the estimation error (Equation 3).
ϵ_{start}	1	Start value for ϵ-greedy policy. For more detail see [Mnih et al. 2013]
ϵ_{decay}	0.995	Reducing factor for reducing ϵ (of ϵ-greedy policy) per episode.
BUFFER_SIZE	10000	Size of replay buffer.
BATCH_SIZE	64	Number of samples used during training.

Table 1. DQN hyperparameter settings

2.2. Double DQN

Double DQN (DDQN) was introduced by [van Hasselt et al. 2015] and tackles the problem of overestimation of the action values, which happens especially in the early stages of training. This approaches uses two networks, one for estimating the maximum rewarding action for state s' and the other network for estimation the Q-value for this state and action pair. If the first network selects an action where the second network doesn't agree with, the Q-value will be not that high. Since I already use a second network parameterized with θ^- to fix the moving target problem, I use this network for the Q-value estimation, this changes the target estimation of the update rule (Equation 3) to the following:

$$r + \gamma Q(s', arg \max_a Q(s', a', \theta), \theta^-) \tag{5}$$

The DDQN uses the same layers and hyper-parameters as the DQN (see Section 2.1 and Table 1). The training performance of the DDQN is analyzed in Section 5.

2.3. Dueling DQN

The last used DQN extension is the dueling DQN introduces by [Wang et al. 2015]. The basic idea of the dueling DQN is to divide/split the network into two streams. One stream for estimating the value of a state, independent of the action, and the second stream estimates the advantage of an action in a certain state. The motivation of the authors is that many environment have states where the action is has no strong influence on the state-value, the proposed architecture should therefore be better in identifying desirable/undesirable states. The state-action value is achieved by combining the two streams as follows:

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + (A(s,a;\theta,\alpha) - \max_{a'} A(s,a;\theta,\alpha)), \tag{6}$$

Where θ represents the shared weights, α the weight of the advantage layers and β the weights of the value estimation layers. In this project I used two fully connected layers with 64 nodes and ReLu activations for both parts of the network (value and advantage estimation). The training hyper-parameters are the same uses for the DQN described in Section 2.1. The training results are discussed in Section 5.

3. Prioritized Replay Buffer

Section 2.1 gave a brief introduction to the commonly used replay buffer where the agent samples uniformly from a set of past experiences. Using this uniform sampling can have the effect that the network doesn't "experiences" important states if these are only sparsely represented in the buffer. One option to overcome this issue was introduced by [Schaul et al. 2015], where samples are weighted by their importance and are sampled base on this prioritization. The weight/prioritization is calculated by using the absolute td-error shown in Equation 7, to insure that samples with zero error, still have a possibility to be sampled, a small value e is added (Equation 8). The sampling probability is calculated by the normalized weight of each sample, shown in Equation 9, where $a \in (0,1)$ determines the uniformity of the priority distribution. In order avoid overfitting high prioritized samples [Schaul et al. 2015] adapts the network update rule to the following Equation 10, this reduces the gradient anti-proportional to the priority value, since these are now samples more often. The $b \in (0,1)$ in Equation 10 determines the anti-proportional weight, usually b is increased during training. The results were achieved by $a = 0.1$, b-start=0.4, b-end=1.0 and b-increase of 0.0001 per step, other setting were tested and their training performance is discussed in Section 5.

$$\delta_t = r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}, \theta) - Q(s_t, a_t, \theta) \tag{7}$$

$$p_t = |\delta_t| + e \tag{8}$$

$$P(i) = \frac{p_i^a}{\sum_k^N p_k} \tag{9}$$

$$\Delta\theta = lr \cdot \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)^b \delta \nabla_\theta Q(s,a,\theta) \tag{10}$$

4. Frame Skipping

The last technique I tested was frame skipping, which was also used in the original DQN paper [Mnih et al. 2013]. In this technique the agent only sees and acts ever k^{th} frame and the action is kept for all intermediate frame the same. This can help in environment where the frame frequency is really high, because consecutive frame look all the "same" and don't influence the agent action decision. By skipping frames we speed up the execution time of the episodes, since the agent acts/evaluates only every k^{th} frame. Training results are discussed in Section 5.

5. Training Performance

The goal was to train an agent which is capable to achieve an average of 13+ points in 100 consecutive episodes in the banana picking environment described in Section 1. I tried different variations of the DQN approach to solve the problem. All network architectures shown in Section 2 were able to achieve the required 13 points, but the DDQN (Double DQN) had the quickest convergence time with an average of 464 episodes. The same goes for the maximum achieved average reward (17.35), see Table 5 and Figure 2.

	Episodes to Solve	Max Average Reward
DQN	511	17.05
DDQN	464	17.35
Dueling DQN	605	17.00
Dueling DDQN	469	16.84

Table 2. Network Architectures

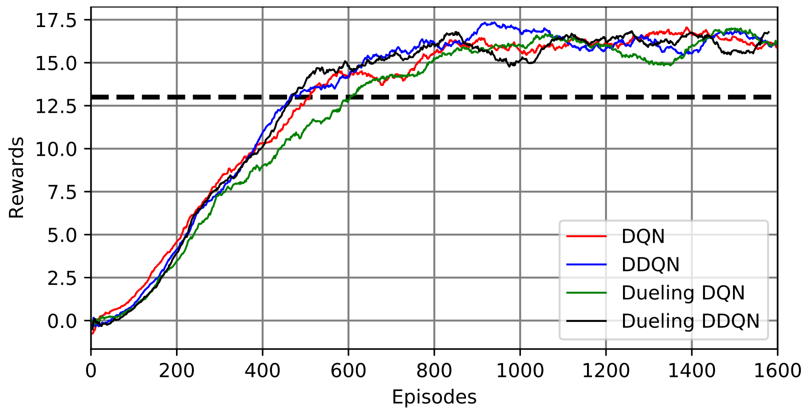

Figure 2. Network Architectures

Furthermore, I explored performance of the prioritized replay buffer (Section 3) and the affect of the hyperparameters on the training's convergence time. During training I started with a β of 0.4 and increased it slowly towards 1, I experimented with different α-values since they had the most affect on the training. α of 0 results in a random sampling an increasing of α results in a more and more prioritized based sampling distribution. I used a DDQN as value estimation network. The results show than increasing of α has an negative affect on the training, since the higher the α, the slower the convergence (see Table 5 and Figure 3). This is mostly due to over-fitting towards the prioritized samples.

	Episodes to Solve	Max Average Reward
Priority Buffer Beta(0.4), Alpha(0.1)	487	16.55
Priority Buffer Beta(0.4), Alpha(0.4)	580	15.27
Priority Buffer Beta(0.4), Alpha(0.6)	826	14.39

Table 3. Priorities Replay Buffer, Hyperparameters

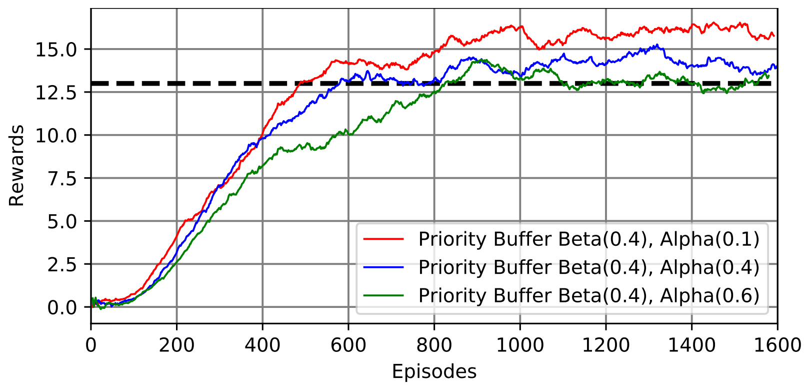

Figure 3. Priorities Replay Buffer, Hyperparameters

The last technique I tried was frame skipping (Section 4), where the agent is sensing/acting only ever k^{th} frame, which can speed up the training on environments with highly frequent frames, because consecutive frame are too similar. On the downside, frame skipping slows down the reaction time of the agent, because he acts only ever k^{th} frame. The results shows that the banana environment has exactly the right state "update rate", since the agent is not able to solve the task and converges much slower, when he skips frames (see Figure 4 and Table 5). This means that frame skipping is not suitable for this environment.

	Episodes to Solve	Max Average Reward
DDQN	464	17.35
DDQN skip frame(1)	n/a	8.09
DDQN skip frames(2)	n/a	3.02

Table 4. Frame Skipping

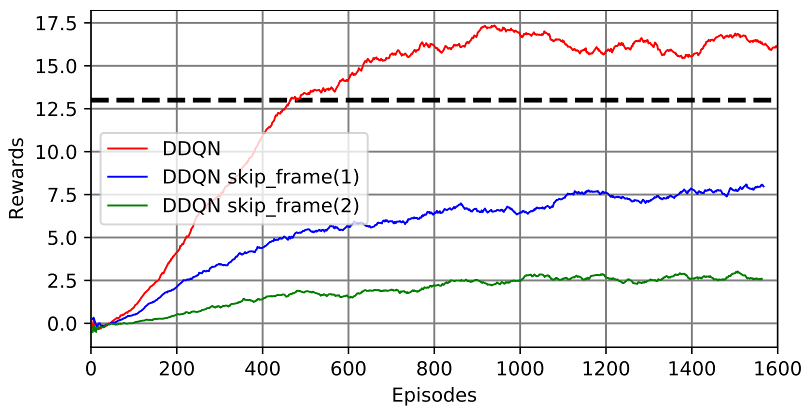

Figure 4. Frame Skipping

6. Ideas for Future Work

- Learning from pixels
- Grid-search for hyperparameter tuning
- Testing of policy based approaches

References

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. A. (2013). Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015). Prioritized experience replay. cite arxiv:1511.05952Comment: Published at ICLR 2016.

van Hasselt, H., Guez, A., and Silver, D. (2015). Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461.

Wang, Z., de Freitas, N., and Lanctot, M. (2015). Dueling network architectures for deep reinforcement learning. *CoRR*, abs/1511.06581.