UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR ROBOTICS
AND COGNITIVE SYSTEMS

Course: Reinforcement Learning (SS2020)
Dr. Christoph Metzner & Honghu Xue, M.Sc.
Assignment Report Nr. IV
Date: 8th July 2021

Final score: 16.75

Team Nr. 1
Rakesh Reddy Kondeti
Christopher Schmale
Tim-Henrik Traving
Contact: timhenrik.traving@student.uni-luebeck.de

1 Theoretical Understanding

1.1 Convergence

1.1.1 Convergence Conditions

Monte Carlo converges to the global optimum of the VE(Value Estimate) under linear function approximation if α is reduced over time. The semi-gradient TD(0) algorithm also converges under linear function approximation, but this does not follow from general results on Stochastic Gradient Descent. In semi-gradient linear TD(0) (Bootstrapping), the expected next weight vector:

$$\mathbb{E}[\mathbf{w}_{t+1}|\mathbf{w}_t] = \mathbf{w}_t + \alpha(\mathbf{b} - \mathbf{A}\mathbf{w}_t), \tag{1}$$

where $\mathbf{b} = \mathbb{E}[R_{t+1}\mathbf{x}_t] \in \mathbb{R}^d$ and $\mathbf{A} = \mathbb{E}[\mathbf{x}_t(\mathbf{x}_t - \gamma\mathbf{x}_{t+1})^T] \in \mathbb{R}^d \times \mathbb{R}^d$. When A is positive-semi definite, it is able to convergence. Additionally α need to be reduced over time.

Monte-Carlo are SGD methods which can converge robustly in on-policy and off-policy. Semi-gradient methods with bootstrapping does not converge under off-policy training
Non-tabular function iteration does not converge when using function approximation.

1.1.2 DQN Convergence Guarantee

DQN generally do not converge when using function approximations. This is due to the use of two contradicting norms (∞-Norm and 2-Norm).
Applying the bellman operator "moves" the value closer to the optimal value by taking the maximum (∞-norm) over all given possibilities. This value is then projected back onto the set of possible approximations (in this case neural nets) by taking the one with the smallest distance to the optimal one (2-Norm).

1.2 Gradient Methods

1.2.1 Semi- vs. True Gradient Descent

The main difference is, that in semi-gradient methods take into account the effect of changing the weight vector w_t on the estimate, but ignore its effect on the target Semi-gradient methods do not converge as robustly as gradient methods, but still do it in important cases like linear cases. Important advantages of semi-gradient methods are:

- Enable significantly faster learning
- Do not wait till the end of episode $\implies$ learning is continuous and online.

Semi-Gradient methods can use an n-step return for approximating a value.

1.2.2 SG TD(0) vs. TG MC

The semi-gradient method in TD(0) uses $R_{t+1} + \gamma\bar{v}(S_{t+1}, \mathbf{w})$, as the target, where R_{t+1} is the reward from the next state S_{t+1}. The term γ is the discounting factor and $\mathbf{w}$ is the weight vector. The weight vector is updated by the formula:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha\left[R + \gamma\hat{v}\left(S', \mathbf{w}\right) - \hat{v}(S, \mathbf{w})\right]\nabla\hat{v}(S, \mathbf{w}) \tag{2}$$

The gradient Monte-Carlo algorithm considers the true value of a state is the expected value of the return following some policy π. The weight vector is updated as:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \left[G_t - \hat{v}(S_t, \mathbf{w})\right] \nabla \hat{v}(S_t, \mathbf{w}) \tag{3}$$

Yes, under linear approximation, both the semi-gradient TD(0) and gradient Monte-Carlo converges to global optimum.

1.2.3 LS TD vs. SG TD(0)

The main advantage of Least Squares TD over semi-gardient TD(0) is that, it does not require a step size, but it does requires ϵ, if ϵ is chosen too small the sequence of inverses can vary wildly, and if ϵ is chosen too large then learning is slowed

1.3 Deep Q-Networks

1.3.1 Replay Buffer

In DQN method, we approximate a function $Q(s, a)$ with the neural network. However the training data should be independent and identically distributed, but when the agent interacts with the environment, the sequence of experience tuples can be highly correlated. To prevent this replay buffer is used.
The replay buffer is used stabilize training on highly correlated sequential data by decorrelating the training examples in each batch used to update the neural network. The replay buffer uses a large buffer of our past experience and sample training data from it, instead of using our latest experience.

1.3.2 Target Network

Consider s, a are the current state and action & s', a' as the next state and actions. Whenever we try to update the Neural Network parameters so to make $Q(s, a)$ value close to the ground truth value, we also alter the value produced for $Q(s', a')$. This makes the training highly unstable.

To make the training stable, we use *target network*, through which we keep a copy of neural network to calculate the value of $Q(s', a')$ in the Bellman equation. That is, the value of produced by this second Q-network or *target network*, is used to backpropogate and train the original Q-Network.

Two options on the update of target network:
Soft update: The whole target network is not updated at once, but instead we update it slowly and in small steps. This means that the target values are constrained to change slowly, greatly improving the stability of learning.

$$\theta' : \theta' \leftarrow \tau\theta' + (1 - \tau)\theta, \quad \tau < 1 \tag{4}$$

where θ' and θ represent the weights of the target network and the current (main) network respectively. The term τ denotes the smoothness.

Hard Update: In the above equation, if $\tau = 0$, then it is called as hard update because we update the whole target network at once.In this case the network is updated after a given number of steps.

1.4 Maximization Bias

1.4.1 Explanation

Q-learing moves to a target value that is defined as $y = r + \gamma \max_a Q_\Phi(s, a)$. This max-term becomes a problem if the result of $Q(s, a)$ is noisy. The only case where the max-term will choose is the correct Q-Value, is if the noise on all possibilities is ≤ 0. In all other cases the max-term incorrectly chooses a possibility where there is positive noise added to the correct Q-value. This leads to an overestimation of the Q-Value, which is called *maximization bias*, as the choice of the values is influenced (biased) by the maximization.

1.4.2 Mitigation

To counteract or mitigate maximization bias, the idea of *double learning (Double-DQN)* is introduced.
One of the reasons for *maximization bias* is that we are using the same samples for maximizing the action and for estimating its value. So, to mitigate this we can use two different estimates $Q_{\phi A}$) and $Q_{\phi B}(a)$, where one estimate $Q_{\phi A}$ is used to determine the maximizing action, $A^* = argmax_a(Q_{\phi A}(a))$ and the other estimate $Q_{\phi B}$ is used to provide the estimate of its value, $Q_{\phi B}(A^*) = Q_{\phi B}(argmax_a Q_{\phi A}(a))$. This decorrelates the value calculation and action selection, which results in an un-/less biased estimate.
The Q-values are updated in the Double Q-learning as:

$$Q_{\phi A}(s, a) \leftarrow r + \gamma Q_{\phi B}(s', \arg\max_{a'}(s', a')) \tag{5}$$

$$Q_{\phi B}(s, a) \leftarrow r + \gamma Q_{\phi A}(s', \arg\max_{a'}(s', a')) \tag{6}$$

2 Programming

2.1 Classic DQN

See figure 1 for the episodic rewards of the classic DQN. Every figure is made by a team member.

2.2 Double-DQN

See figure 2 for the episodic rewards of the classic DDQN. Every figure is made by a team member.
In practice the two networks are not trained seperatly. Instead, the current and target network are used to get the two different Q-functions. The current network is used to choose the action, which is then evaluated by the target network:

$$y = r + \gamma Q_{\phi'}(s', \arg\max_{a'} Q_\phi(s', a')) \tag{7}$$

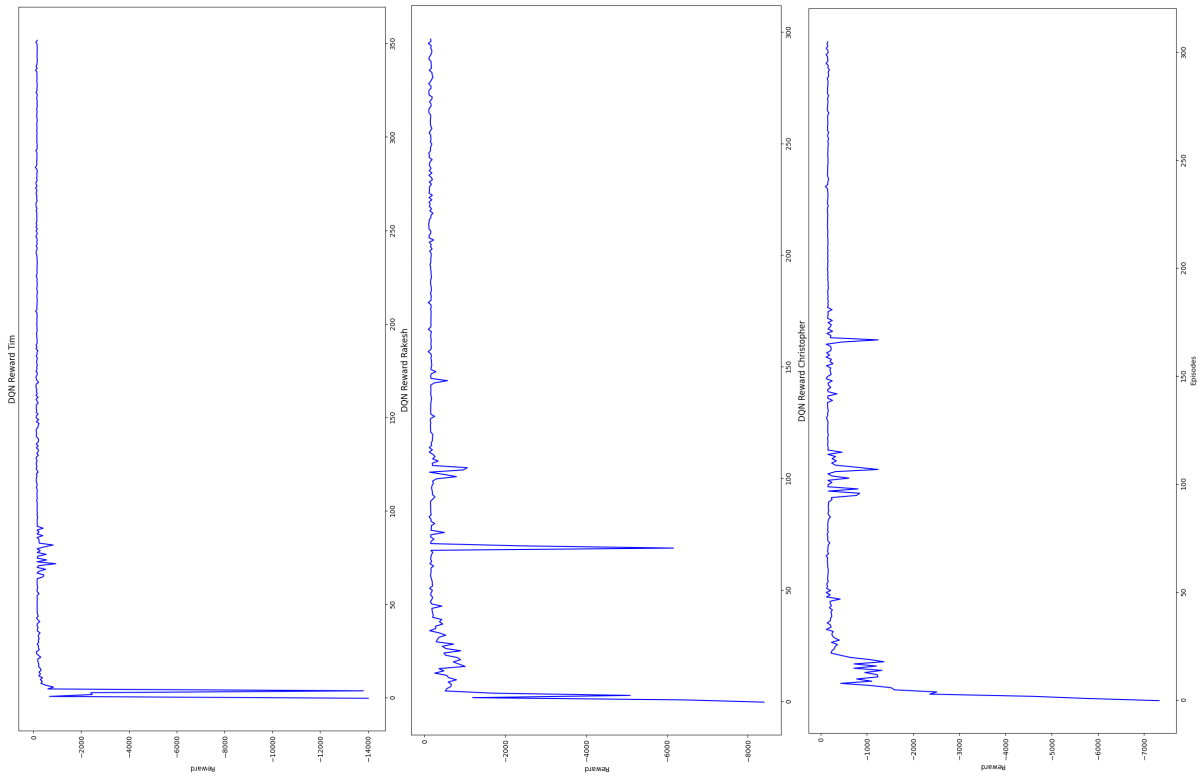
Therefore only one network is trained, while the other is being kept up-to-date through either the soft- or the hard update.

3 Going Deeper

3.1 Q-Values

3.1.1 Lowest possible Q-Value

As you can see in figure 3a the lowest possible q-value is around -90. However, the q-values do not converge to this number.



(a) Tim's run **(b)** Rakesh's run **(c)** Christopher's run

Figure 1 The DQN episodic rewards

3.1.2 Approach lowest q-value

The lowest possible q-value can be $-\infty$, for example if the agent is not able to finish an episode.

3.1.3 Optimally

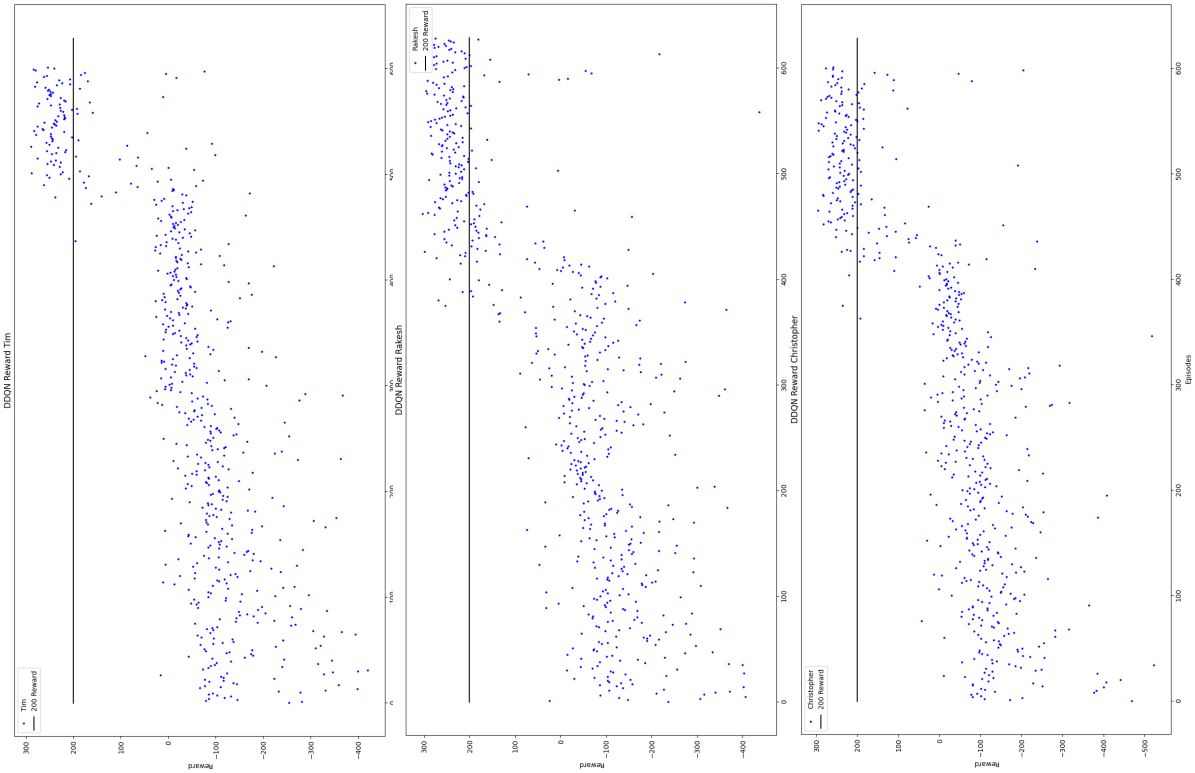
3.2 Q-loss

3.2.1 Plots of q-loss and estimated q-values

The q-loss can be seen in figure 3b and the estimated q-values can be seen in figure 3a .

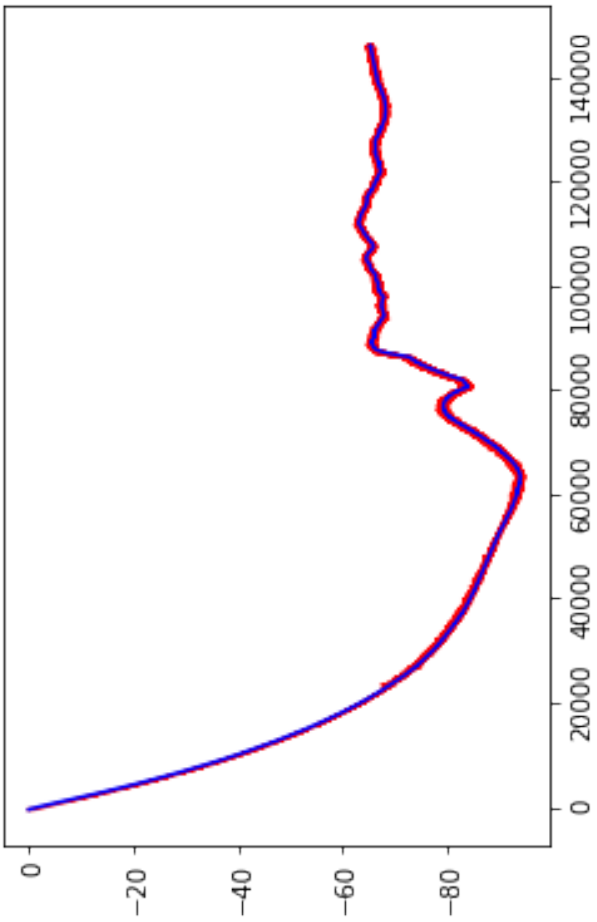
3.2.2 Q-loss values

In the beginning, the q-loss function approaches zero, because there is no episode finished. To finish the first episode, the system takes a lot of time (around 30000 steps). After the first episode is finished, there is a q-value which the q-loss can be calculated on. All in all, there are around 300 episodes.

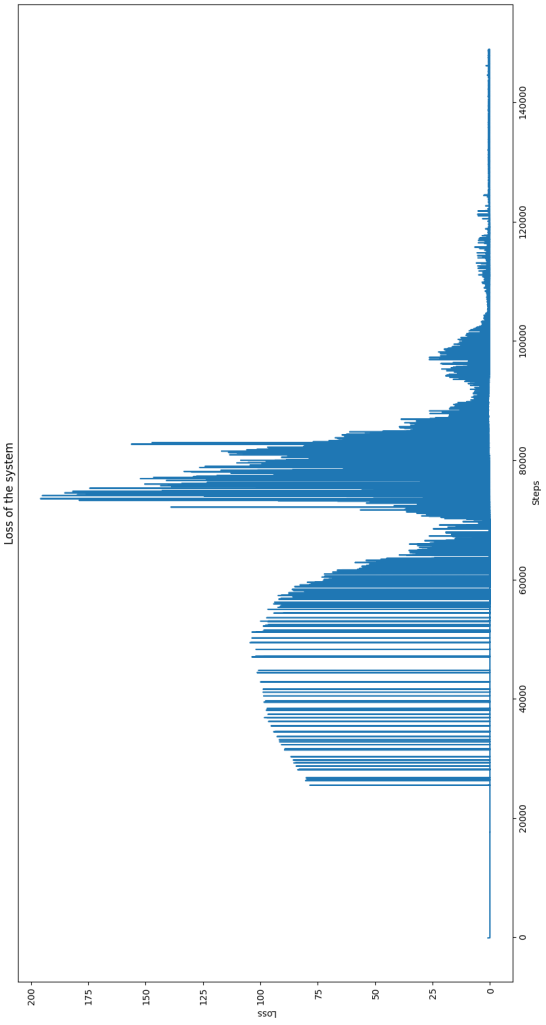


(a) Tim's run **(b)** Rakesh's run **(c)** Christopher's run

Figure 2 DDQN Episodic Rewards



(a) Q-values



(b) Q-loss

Figure 3 Q-loss and q-values