RL-Course 2023: Final Project Report

The Random Action Rascals: Christian Eberle, Felix Pfeiffer

June 19, 2024

1 Introduction

In this report we describe the methods and algorithms we used to train a reinforcement learning agent for the Laser Hockey Challenge of the lecture Reinforcement Learning. Moreover we present and discuss our results in the following.

Laser Hockey is a game for two players where the aim is to shoot a puck into the goal of the opponent on the opposite half of the field. It is based on the gym environment from openAI using the Box2D engine. The player has four different ways to interact with the enviroment. He can move in the x and y direction on the playing field with a speed value between -1 and 1. It can also rotate around its own axis at a speed between -1 and 1. Furthermore, it is possible to shoot the puck once the agent has picked it up. The state of the environment is fully described with 18 values. Six values are needed to describe the state of a player. The x and y position as well as the rotation angle. In addition, the current speed can be observed for all these values. To define the state of the puck four values are needed. Its x and y position as well as its speed into x and y direction. The remaining two values indicate the remaining time that the two players can still hold the puck if they have it.
Although the game has a relatively small observation and action space, it is much more challenging to implement a reinforcement learning agent for it than the first impression might suggest.

For this project we implemented an agent with a discrete and an agent with a continuous action space:

- **Clipped Dueling Double Deep Q-Network** - Christian Eberle
- **Twin Delayed Deep Deterministic Policy Gradient (TD3)** - Felix Pfeiffer

2 Clipped Dueling Double Deep Q-Network

2.1 Methods

The aim of Q-Learning is to find the action-value function $Q^\pi(s, a)$ of an MDP. This function maps a state-action pair (s, a) to the expected return R_t when taking action a in state s and then following policy π:

$$Q^\pi(s, a) = \mathbb{E}[R_t|(s, a), \pi] \tag{1}$$

Given the optimal action-value function Q^*, the optimal policy π^* is simply choosing the action that maximizes $Q*$:

$$\pi(s) = \arg\max_a Q^*(s, a) \tag{2}$$

To find Q^*, we iteratively apply the following update rule derived from the Bellmann optimality equation:

$$Q_{i+1}(s,a) \leftarrow \mathbb{E}[r + \gamma \arg\max_a Q^i(s,a)] \tag{3}$$

With this update rule, Q will eventually converge to $Q*$. However, in MDPs with larger action or state spaces this is often infeasible since the Q-table containing the Q-value for each state-action pair grows exponentially. Mnih et al. [4] thus proposed Deep Q-Learning, in which the action-value function is approximated with a neural network. To find network parameters θ such that $Q(s,a,\theta) \approx Q^*(s,a)$, we optimize the following loss function:

$$\mathcal{L}i(\theta_i) = \mathbb{E}_{s,a,r,s' \sim \rho(.)}[(y_i - Q(s,a,\theta_i))^2] \tag{4}$$

$$y_i = r + \gamma \arg\max_a Q(s,a,\theta_{i-1}) \tag{5}$$

The idea behind this loss function is to minimize the temporal difference error $y_i - Q(s,a,\theta_i)$ that represents the mismatch between two successive predictions of the Q-value.
Since the network selecting the action is the same as the network evaluating the Q-value of the selected action, this leads to a significant maximization bias. To overcome this bias, Van Hasselt et al. [7] propose Double DQN, an approach using two neural networks: The online network Q_{online} is used to select the action, and the target network Q_{target} is used to compute the target y_i. While the online network is updated each training step, the parameters of the target network are periodically updated by copying the parameters of the online network.
However, if the target network parameters are copied too frequently or if the policy changes too slowly then Double DQN can still suffer from maximization bias. A more conservative estimation of Q-values is Clipped Double DQN [1]. Here, the target y is calculated using the minimum of the two Q-functions:

$$y = r + \gamma \min\left(Q_1(s', \arg\max_a Q_1(s',a,\theta_1)), Q_2(s', \arg\max_a Q_2(s',a,\theta_2))\right) \tag{6}$$

Another major improvement on DQN is Dueling DQN [8]: This architecture splits the Q-value into a state value $V^\pi(s)$ and an advantage value $A^\pi(s,a)$:

$$V^\pi(s) = \mathbb{E}_{a\sim\pi}(Q^\pi(s,a)) \tag{7}$$

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s) \tag{8}$$

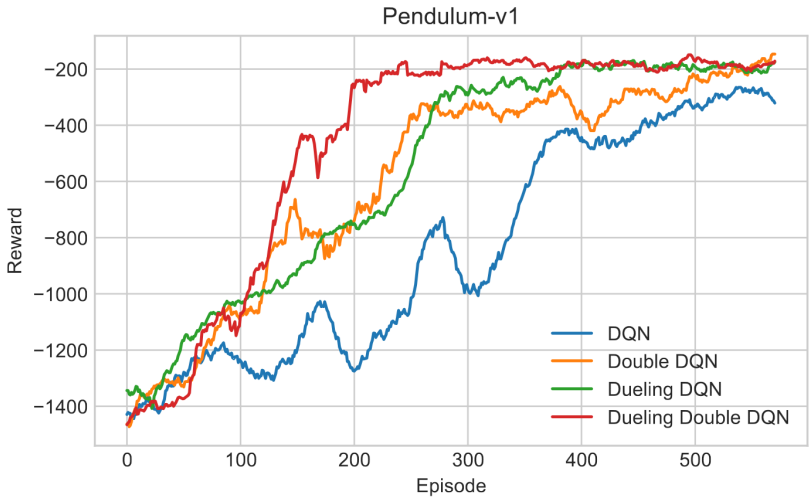
The state value measures how good a state is for our agent, while the advantage value quantifies how good an action is compared to other actions. A Dueling DQN splits its final network layer into two streams: One to estimate the state value, and the other to estimate the advantage value. From these, the Q-Value is calculated as:

$$Q^\pi(s,a,\theta,\alpha,\beta) = V^\pi(s,\theta,\beta) - (A^\pi(s,a,\theta,\alpha) - \frac{1}{|\mathcal{A}|}\sum_{a'} A^\pi(s,a',\theta,\alpha)) \tag{9}$$

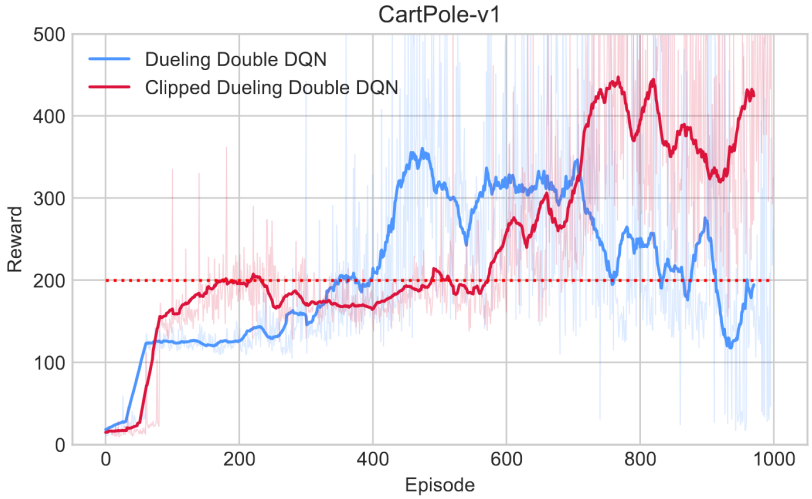
Here, $\mathcal{A}$ denotes the set of available actions. The advantage of separating the state and the advantage value is that it allows the network to learn which states are valuable without having to learn the effect of each action in every state.

2.2 Experiments

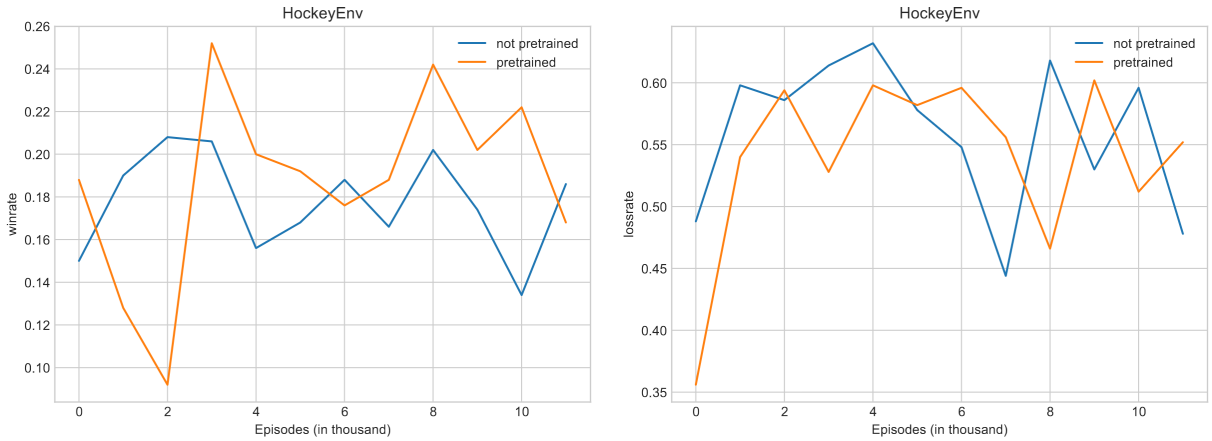
The DQN code solution of Exercise 7 served as a basis for the Dueling Double DQN implementation used in the following experiments. The added output streams (state and advantage) each consist of one hidden layer of size 64 with ReLU activation, followed by the output layer. If clipping was enabled, the temporal difference target was calculated according to (6). We also introduced epsilon-decay to speed up convergence by focusing on exploration in the earlier stages of training. In total, there were 14 hyperparameters to be optimized which was a challenging task. The default hyperparameter used in the following experiments are listed in the appendix.
The following experiments are limited to easier gym environments because the our DQN agent failed to solve the hockey environment (view appendix for details). In the first experiment, we compared the performance of different DQN architectures in the Pendulum environment (200 steps per episode). Since reward variance was high, the reward was smoothed by taking the running mean of the last 30 episode rewards. As expected, the performance of the vanilla DQN was significantly worse than all other agents. For all other agents, the performance in the later episodes was similar. However, Dueling Double DQN converged the fastest which is why we used this architecture in the following experiments.



The next experiment explores the effect of using a clipped target. Each episode in the cart pole environment was terminated after a maximum of 500 steps and the environment is considered to be solved if reward > 200, indicated by the dashed red line. Both agents were able to solve the environment.



Again, Dueling Double DQN converged the fastest but the performance decreased toward the end of training. Clipped Dueling Double DQN was observed to be more stable, resulting in a much better final performance. This is also why we chose the Clipped Dueling Double DQN agent for our final and most challenging environment, hockey.
Since the goal of this project was to train an agent that can consistently beat the weak opponent, our performance metrics were the win-, loss- and drawrates against the weak opponent. The metrics were calculated over a period of 500 evaluation episodes to reduce variance. Initially, we trained our agent in the normal game mode against the weak opponent, evaluating every 1000 steps. We can observe that the agent is unable to improve over time, with performance stagnating at a winrate of 0.2 and a lossrate of 0.6. This is different for the "train defense" mode: Here, the agent seems to increases its reward over time. However, these improvements don't seem to translate to the normal game mode, as the agent pretrained in defense-mode performs similarly bad as the other agent.



3 Twin Delayed Deep Deterministic Policy Gradient

3.1 Methods

Twin Delayed Deep Deterministic Policy Gradient (TD3) introduced by Fujimoto et. al. [2] is the successor of Deep Deterministic Policy Gradient (DDPG) by Lillicrap et. al. [3], which combines the ideas from DQN and deterministic policy gradient (DPG). DDPG suffers from the problem of overestimation bias, because in Q-learning the Q-values that indicate how well a certain action is rated in a certain state is chosen greedy, $y = r + \gamma \max_{a'} Q(s', a')$. Thrun&Schwatz [6] have shown that if the target is vurnerable to an error ϵ, the Q-value will be greater than the true maximum, $\mathbb{E}_\epsilon[\max_{a'} Q(s', a') + \epsilon] \geq \max_{a'} Q(s', a')$. Resulting in a suboptimal policy that the agent would learn. Furthermore, this overestimation accumulates over time leading to false predictions. Since Actor-Critic algorithms learn from each other an, overestimation would then lead to feedback loops harming the performance further. TD3 makes three improvements to solve this problem.

1. Clipped Double Q-Learning for Actor-Critic
2. Target Networks and Delayed Policy Updates
3. Target Policy Smoothing Regularization

Since TD3 builds on Clipped Double Q-learning, it consists of two critic networks Q_{θ_1} and Q_{θ_2}, their target networks $Q_{\theta'_1}$ and $Q_{\theta'_2}$ and an actor π_{ϕ_1} as well as its target $\pi_{\phi'_1}$. Because the two critic networks

use the same experiences from the same replay buffer for some states s we have $Q_{\theta_2}(s, \phi_{\phi_1}(s)) > Q_{\theta_1}(s, \phi_{\phi_1}(s))$. This is problematic because Q_{θ_1} generally overestimates for some states s as described above. Thus, Fujimoto et. al. propose to take the minimum of the two values called clipped Double Q-Learning for Actor-Critic, resulting in the following target update equation (10) for the algorithm.

$$y_1 = r + \gamma \min \left(Q_{\theta_1'}(s', \pi_{\phi_1'}(s')), Q_{\theta_2'}(s', \pi_{\phi_1'}(s')) \right) \tag{10}$$

Target y_1 is the used to update both critic networks, subsequent the loss of the critic networks is defined by equation 11.

$$\begin{aligned} \mathcal{L}_{\text{critic1}} &= MSE\left(y_1 - Q_{\theta_1}(s, a)\right) \\ \mathcal{L}_{\text{critic2}} &= MSE\left(y_1 - Q_{\theta_2}(s, a)\right) \end{aligned} \tag{11}$$

The interplay of actor and critic can lead to divergent behavior while training. An inaccurate value estimation of the critic through overestimation provides an unstable target for updates. This results in the actor learning a suboptimal policy, which subsequently exacerbates the inaccuracy of value estimation. The proposed improvement Target Networks and Delayed Policy Updates, keeps the target networks fixed for a few training steps and only updates the other networks in each step to converge, reducing the value error and allowing more stable updates. Moreover, the target networks are updated slowly using a parameter τ. This results in the update equation (12) of the target networks every other step:

$$\begin{aligned} \theta_1' &\leftarrow \tau \cdot \theta_1 + (1 - \tau) \cdot \theta_1' \\ \theta_2' &\leftarrow \tau \cdot \theta_2 + (1 - \tau) \cdot \theta_2' \\ \phi_1' &\leftarrow \tau \cdot \phi_1 + (1 - \tau) \cdot \phi_1' \end{aligned} \tag{12}$$

Since the action space is continuous (The Laser Hockey environment accepts values between -1.0 and 1.0) similar actions should have similar Q-values. When dealing with deterministic policies function approximation errors inaccuracies increase the variance of the target. As a consequence, this results in a policy overfits on certain actions. To address this issue, TD3 adds random Gaussian noise to the target policy that gets clipped to ensure only small changes are made, resulting in the modified target update (13).

$$\begin{aligned} y &= r + \gamma Q_{\theta'}\left(s', \pi_{\phi'}(s') + \epsilon\right) \\ \epsilon &\sim \text{clip}\left(\mathcal{N}(0, \sigma), -c, c\right) \end{aligned} \tag{13}$$

Because of computational reasons TD3 has only one actor π_{ϕ_1} that is optimized against Q_{θ_1}. Since we want to maximize, we takes the negative value returned by the critic to update the actor. Equation 14 outlines the loss function for the actor.

$$\mathcal{L}_{\text{actor}} = -Q_{\theta_1}\left(s, \pi_{\phi_1}(s)\right) \tag{14}$$

3.2 Experiments

After implementing the algorithm, we tested the function by solving simpler environments than Laser-Hockey. Figure 2 shows that after a relatively short time of about 100 episodes the environment Pendulum-v1 was solved. This requires a reward between about -200 and 0. The slightly more complex environment LunarLanderContinuous-v2 could also be considered solved after about 1000 episodes after a reward of 200 was reached. After experimentally verifying the correct implementation of the TD3 algorithm, we could start tuning the hyperparameters for the LaserHockey environment.

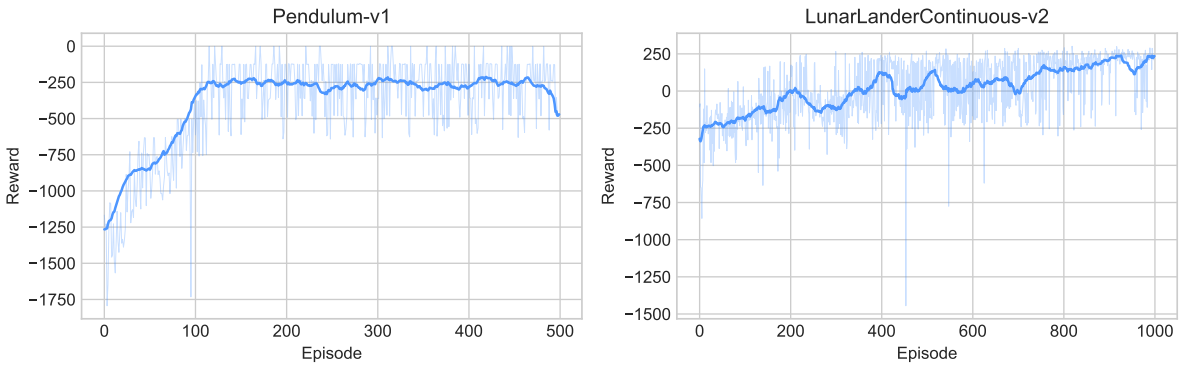
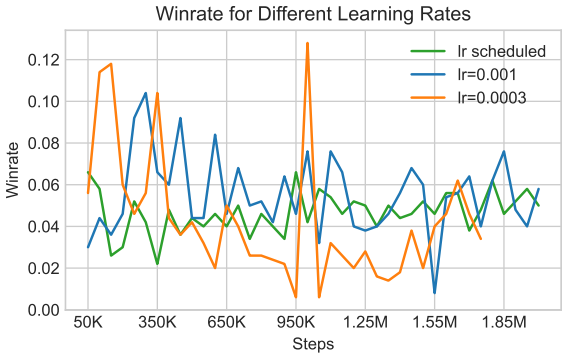
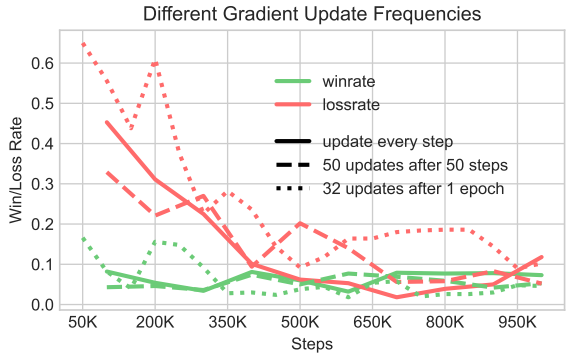


Figure 2: Reward over time during training of TD3 on simple to solve environments.

For the first attempts we started the training against the weak opponent. Since we used a learning rate of 0.001 to solve the pendulum environment and used a learning rate of 0.0003 as suggested by Fujimoto et. al. for the TD3 algorithm, the first step was to try these different learning rates. We also used a learning rate scheduler with an initial learning rate of 0.0005 that gets multiplied with $\gamma = 0.9$ every 50k steps. Figure 3a shows that this did not have a big impact on the agent's win rate, which was always around 6%. A closer look at the results shows that the agent defends well, but most of the games end in a draw. Plots can be found in the appendix.
Another experiment was to vary the number and frequency of gradient updates. Figure 3b shows a similar result with a constant low win rate but a decreasing losing rate. Three different experiments were conducted. The solid line is the result when the gradient is updated at each step. In the run of the dashed line, the gradient was updated 50 times after 50 steps. Finally, in the run shown with the dotted line, the gradient was updated 32 times after each epoch. When visualising the game-play of the agent, it was noticed that the puck is well defended, but that he does not actively move towards it, pick it up and shoot it down if at the beginning of an episode the puck spawns in his own half of the field.



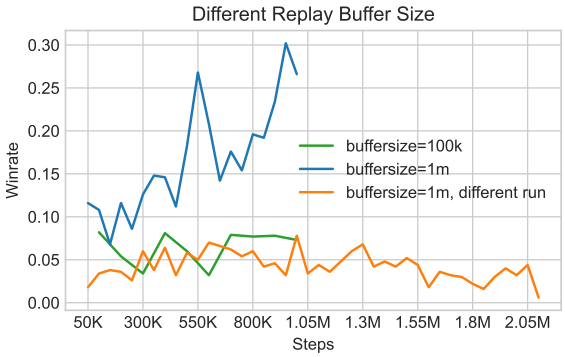
(a) Experiments with different learning rates. Winrate against the weak opponent.



(b) Experiments with the update frequency of the gradients. Win and lossrate against the weak opponent.

To get the agent to pick up the puck and score a goal, we have added the various rewards that the environment gives returns at each step to the standard reward. These include 'reward_closeness_to_puck', 'reward_touch_puck', 'reward_puck_direction'. We have also increased the size of the replay buffer from 100 thousand to one million replays for the experiments. Since we won only few games, we wanted to ensure that these good replays are not lost and contribute positively to training in the future.

In the first experiment with a larger replay buffer, the win rate rose to about 30%, while loosing only around 10% of the games against the weak opponent. In the end, this agent was used for the tournament. However, this result could not be repeated, as the orange line in figure 4a indicates. This leads us to believe that the previous low winrates cannot be explained by the replay buffer size.
Figure 4b also shows that by using additional rewards, the hoped-for result could not be achieved, even though the winrate increased slightly.



(a) Experiments with different replay buffer sizes. Winrate against the weak opponent.

(b) Experiments with different rewards. Win and lossrate against the weak opponent.

Since all agents have always learned not to lose against the weak opponent, but to draw most of the time, we pretrained an agent in shooting mode. after 250 thousand steps we trained against the weak opponent. After two million training steps we achieved a winrate of nearly 80%, our best result. In figure 5 one can observe that the winrate drops again, just as the agent in shooting mode loses performance after about 700 thousands steps. A possible reason could have been the high learning rate of 0.001. Unfortunately, this performance was reached only after the tournament, so that this agent could not participate in it.

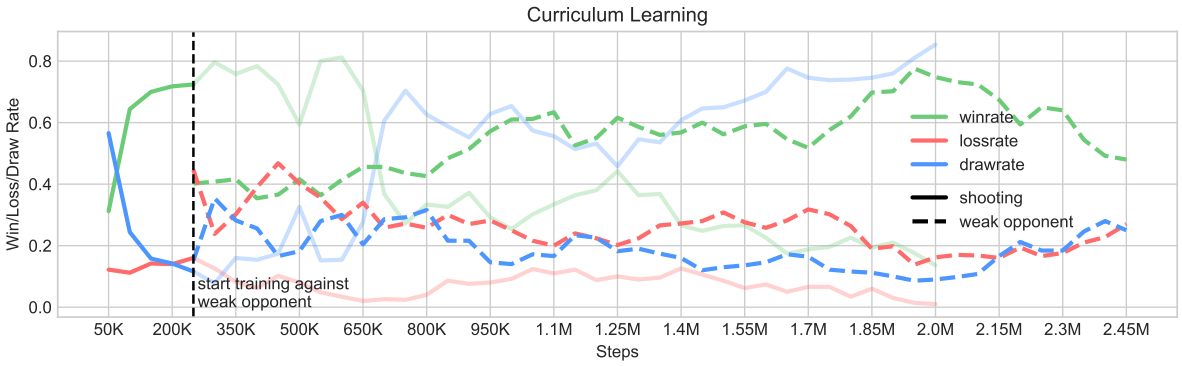


Figure 5: Results against the weak opponent after training in shooting mode for 250k steps.

4 Discussion

4.1 Clipped Dueling Double Deep Q-Network

The DQN Architecture worked well for easier gym environments, such as pendulum or cart pole. Using a Dueling Double DQN seemed to improve convergence speed, while using a Clipped Dueling Double DQN seemed to make the training process more stable. Unfortunately, the performance of the DQN agent was quite underwhelming in the hockey environment. While the agent's behavior looked reasonable most of the time (i.e. consistently moving to the ball and shooting toward the opponent's side), it lacked in precision when moving and shooting. Moreover, when the ball was out of reach on the opponent's side the agent often moved away from the goal, allowing the opponent to score easy goals. We speculate that this poor performance stems mainly from a suboptimal choice of hyperparameters. While we did experiment with different hyperparameters (namely learning rate, target update frequency, batch size and training time length), our search was far from exhaustive. Furthermore, we believe our performance could have been improved by using Prioritized Experience Replay [5] to sample from the replay buffer, thereby training more efficiently. One last possible way of improving our performance could be to add more action classes to the discrete action space. It could be that the default discrete action space is too limiting for a good performance.

4.2 Twin Delayed Deep Deterministic Policy Gradient

The experiments with simple environments have shown that TD3 can converge very quickly and reliably in easy to solve scenarios. However, since the LaserHockey environment is much more complex, it is also much more challenging to tune the hyperparameters for it. In the end, we managed to have an agent with 30% win and 10% loosing rate against the weak opponent to compete in the RL tournament 2023. The biggest problem in training was that the agent defends well, but doesn't score many goals. For this it might have been useful to have a custom reward that rewards the agent as soon as he has the puck. In addition, we might have achieved faster convergence if we filled the replay buffer with some experiences from the gamplay of two strong opponents against each other. This way, the agent would have already known useful actions at the beginning of the training. Finally, due to the large number of hyperparameters and the stochastic environment, which does not allow exact replication of results, tuning a reinforcement learned agent is a very demanding and time-consuming task. At the very end we managed to train an agent with $\sim 80\%$ winrate and $\sim 15\%$ looserate against the weak opponent through many experiments with different hyperparameters and curriculum learning.

4.3 Comparison

TD3 is a much more sophisticated architecture than Clipped Dueling Double DQN. We can see from our experiments that even in simple environments like pendulum, it outperforms the DQN architectures. TD3 was also better suited for the hockey environment, since the algorithm is designed for a continuous action space. On the other hand, DQN can only handle discrete action spaces and needs to discretize the hockey action space first. This can be quite limiting, since often the best possible action is not included in the discrete action classes available to DQN. It is therefore not a big surprise that TD3 performed much better in the hockey environment than DQN did. However, TD3 was also more computationally expensive, with its training runs needing much more time than DQN.

References

[1] S. Fujimoto, H. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

[2] S. Fujimoto, H. van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1587–1596, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[3] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[5] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[6] S. Thrun and A. Schwartz. Issues in using function approximation for reinforcement learning. In *Proceedings of the Fourth Connectionist Models Summer School*, volume 255, page 263. Hillsdale, NJ, 1993.

[7] H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[8] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas. Dueling network architectures for deep reinforcement learning, 2016.

5 Appendix

5.1 DQN Hyperparameters

The following table contains the hyperparameters used to train the agents in the experiment section:

	Pendulum	Cart Pole	Hockey
gamma (reward discount)	1	1	0.99
epsilon min	0.01	0.01	0.02
epsilon max	1	1	1
epsilon decay	0.95	0.95	0.995
target update frequency	20	20	200
learning rate	1e-4	1e-3	1e-4
max. buffer size	100 000	100 000	500 000
train step frequency	1 episode	1 episode	1 episode
train episodes total	600	1000	10 000
epsilon (Adam optimizer)	1e-6	1e-6	1e-61
DQN backbone hidden size	[100, 100]	[100, 100]	[100, 100]
DQN heads hidden size (Dueling only)	[64]	[64]	[64]
batch size	32	32	128

5.2 TD3

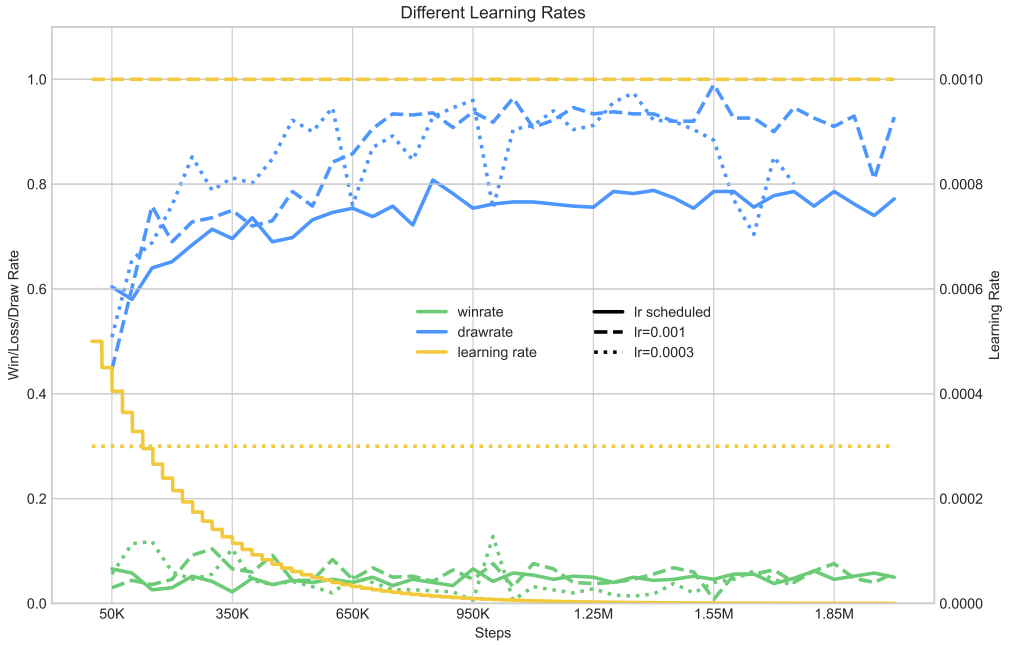


Figure 6: This plot shows the influence off different learning rates to the winrate against the weak opponent. One can observe that the agents learns how to defend, but not how to score goals. The performance is nearly independent of the learning rate.

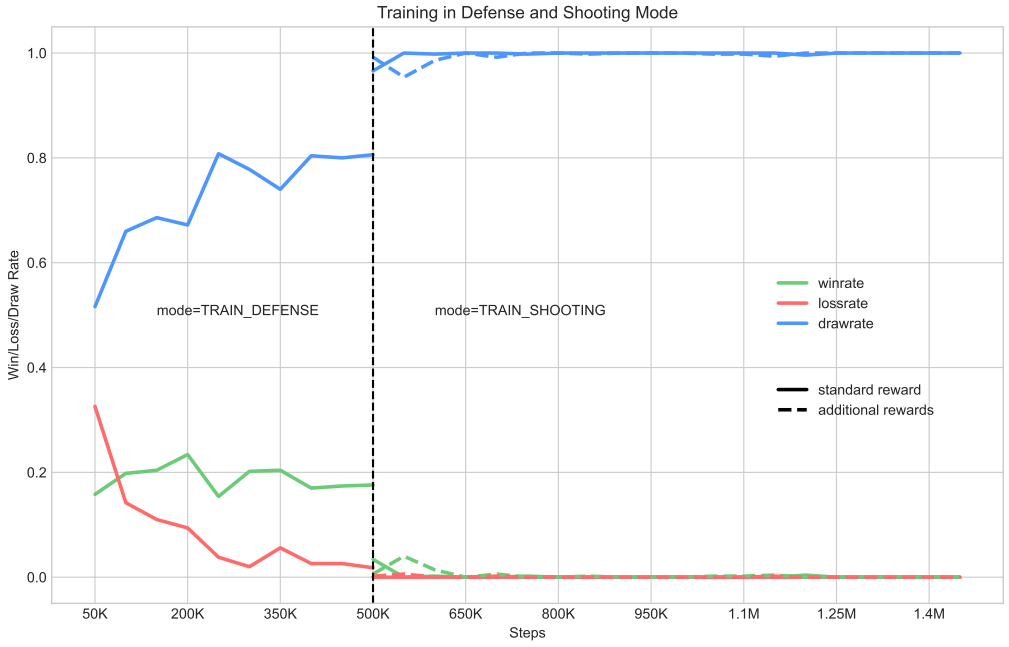


Figure 7: This plot shows the performance in shooting mode after pretrained in defense mode for 500 thousand steps. One can see that the agent is really good at defending, but it does not learn how to score goals. The reason for this is that it does not move towards the puck if it spawns it its own half at the beginning of an episode. Even when added the additional rewards provided by the environment the performance does not increase.