RL-Course: Final Project Report, Philipps Part

1 Method

1.1 Deterministic Policy Gradient and TD3 (Philipp)

TD3 is an actor-critic algorithm used in continous environments, which unites the advantages of value-based and policy-based methods. While value-based methods like Deep Q-Learning (DQN) enable off-policy learning, the have problems in continuous environments. This is solved by combining with an actor, resulting in an approach called Deep Deterministic Policy Gradient (DDPG). TD3 then builds upon the DDPG algorithm but tries to solve some of its key issues. The following sections explain this in more detail.

1.1.1 From DQN to DDPG

In continuous environments, Value Based Methods are hard to optimize, mostly due to the use of the Bellman Optimality equation. In the case of Deep Q-Learning [4], this occurs on two occasions. First, when computing the policy for a given state 1 and secondly in the loss function 2. The $\arg\max$ / $\max$ operator means we have to find the maximum, which is not trivial in a continuous action space.

$$\arg\max_a Q_\omega(s,a) \tag{1}$$

$$L(\omega) = \mathbb{E}_{s,a,r,s' \sim D_i}[(r + \gamma \max_{a'} Q_{\omega'}(s',a') - Q_\omega(s,a))^2)] \tag{2}$$

In contrast, deterministic policy based methods learn the policy $\mu_\theta(s) = a$ explicitly and can easily deal with continuous environments. Therefore, it seems natural to replace the $\max$ by the policy, leading to the Loss function:

$$L(\omega) = \mathbb{E}_{s,a,r,s' \sim D_i}[(r + n\gamma Q_{\omega'}(s', \mu_{\theta'}(s')) - Q_\omega(s,a))^2)] \tag{3}$$

Because the expectation only depends on state distribution D_i, we can use off-policy learning and reuse data from prior iterations. To reduce error, two target networks $Q_{\omega'}, \mu_{\theta'}$ are used with sliding updates by $\omega' = \tau\omega + (1-\tau)\omega'$.
For the optimization of the policy, we use the standard objective function of Deterministic Policy Gradient:

$$J(\mu_\theta) = \mathbb{E}_{\mu_\theta}[G_1] \tag{4}$$

with its derivative:

$$\nabla_\theta J(\mu_\theta) = \mathbb{E}_{\mu_\theta}[\nabla_\theta \mu_\theta(s) \nabla_a Q_\omega(s,a)|_{a=\mu_\theta}] \tag{5}$$

The behavioural policy is simply given by the policy μ_θ. However, to explore the state space better, some Noise ϵ is added during training, resulting in:

$$a = \mu_\theta(s) + N \tag{6}$$

1.1.2 TD3

The most prominent problems of DDPG are an overestimation of the critic target, a high variance in the Q-network predictions and a Q-Network which can easily overfit. The following sections show how these issues can be addresed, which then forms the TD3 algorithm.

Overestimation Bias

One of the problems which DDPG introduces is a value overestimation bias. Recall that in normal Deep Q-Learning, the Q-value estimate is updated by the derivative of the loss function given by equation 2. If we assume that the target $r + Q_{\omega'}(s', a')$ has some error ϵ, which is reasonable especially if approximated by a neural network, we see that

$$\mathbb{E}_\epsilon[r + \max_{a'} Q_{\omega'}(s', a') + \epsilon] \geq r + \max_{a'} Q_{\omega'}(s', a') \tag{7}$$

meaning that the estimate will always be larger than the true target. Consequently, the size of the gradient will always be overestimated. Although in the DDPG the $\max$ is replaced by the policy μ_θ, [1] showed that the overestimation bias is still present. Therefore, they transferred the technique from Double DQN [5], where two actors$\mu_{\theta_1}, \mu_{\theta_2}$ and two critics $Q_{\omega_1}, Q_{\omega_2}$ (one critic in implementation for performance purpose) were used, which leads to the loss:

$$L(\omega_1) = \mathbb{E}_{s,a,r,s',n\sim D_i}[(r + n\gamma Q_{\omega_2'}(s', \mu_\theta(s')) - Q_{\omega_1}(s, a)^2)] \tag{8}$$

However as the critics are not entirely independent, we can further reduce overestimation by just using the minimum over both targets:

$$y = r + \gamma \min(Q_{\omega_2'}(s', \mu_\theta(s')), Q_{\omega_1'}(s', \mu_\theta(s')) \tag{9}$$

Target Policy Smoothing

Deterministic policies are often prone to overfit, which means that they will produce very different actions for similar states. However, this means that the Q-value target in equation 9 will have high variance, leading to an inaccurate gradient for the critic. [1] addressed this issue by forcing the critic to learn similar values for similar actions, which can be realized by adding some noise to the policy:

$$y = r + \gamma \min(Q_{\omega'}(s', \mu_{\theta'}(s) + \epsilon), Q_{\omega'}(s', \mu_{\theta'}(s) + \epsilon)) \tag{10}$$

with $\epsilon \sim N(0, \sigma)$ noise from a normal distribution which then gets clipped to prevent to extreme differences.

Delayed updates

As the previous section showed, the Q-value estimate is often biased and has high variance. But as the update of the policy in equation 5 contains $Q_\omega(s, a)$, this means the inaccurate estimate of the Q-values results in an inaccurate estimate of the policy.
To stabilize the policy target, DDPG uses a sliding update of the target with $Q_{\omega'} = \tau \cdot Q_\omega + (1 - \tau) \cdot Q_{\omega'}$, where τ controls the influence of the current parameters onto the target. Although this approach stabilizes the Q target, it does not resolve the high variance in the policy gradient due to inaccurate Q-values. To reduce the variance, it seems reasonable that the Q-network should have a good estimation before updating the policy. The authors realize this by delaying the update of the policy, which results in updating them only every d epochs. In this way, the critic has a good estimate before the policy is updated and the gradient will be more accurate. In TD3, the combination of those two approaches leads to a more stable target, which introduces a better gradient.

2 Results

2.1 TD3 (Philipp)

2.1.1 Hyperparameter modifications

We start with the hyperparameters proposed by the authors of the paper on TD3 [1], which results in a winrate of 70%. Tuning the hyperparameters added by TD3 over DDPG should not only increase performance, but also show the advantages of TD3. Increasing the policy noise resulted in a winrate of 2.5% while decreasing resulted in a winrate of 52%. Removing the delay and sliding update resulted in an increased winrate of 82%, incressing the delay together with τ decreased it to 65%. These results show that a small policy noise helps performance. A delay with a sliding update however does not seem to help for this environment.
Regarding general hyperparameters different batch sizes and reward discount did not result in any major differences, so we stayed at the default ones. As the learning rate can be challenging to optimize, we replace the default Adam optimizer [2] by an newer version called Rectified Adam (RAdam) [3], which is less sensitive to learning rate. This results in a winrate of 88%.

2.1.2 Reward modifictaions

(Disclaimer: These tests were already done in the old environment, where no penalty was added for a standing puck in the own half.)
The initial reward of the environment is very sparse: You can only get a +10 or -10 at the end of each episode for a win/loss. This creates a very challenging environment for the agent, as for every action the agent does not receive any reward and can therefore not infer if the chosen action was good or bad.
To adress this problem, the environment already gives the agent a small penalty for moving away from the puck or leaving the puck untouched. However, we can also modify the reward ourself by adding more penalty with different strength factors by: $reward = normalreward + factor \cdot distancepenalty$. Figure 1 clearly shows that adding more penalty speeds up training. However, a large factor would result in the agent ignoring the actual reward (scoring a goal), and learning to minimize the penalty. Therefore, we also tried to decay the penalty factor with increasing epochs. Surprisingly, this does not seem to make a difference on final performance.

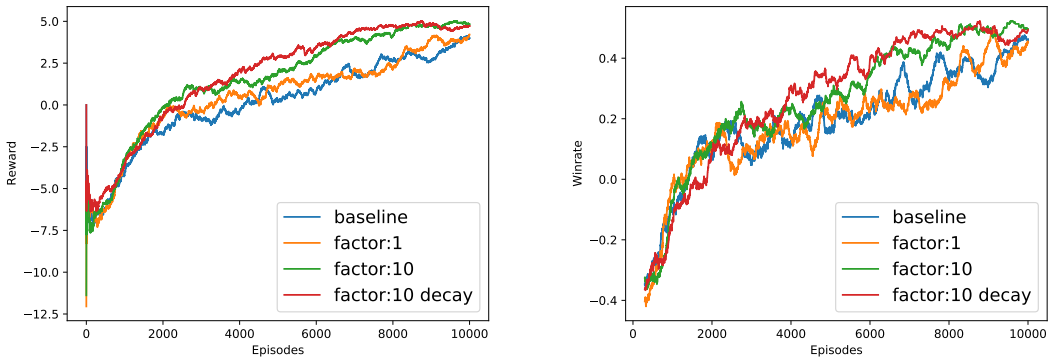

Figure 1: Effect of Reward modification with the closeness to puck reward on the agents reward and winrate. Using a lrge factor speeds up training.

2.1.3 Self Play

Even though TD3 introduced policy noise to smooth the policy and reduce overfitting, we show that this is still present in the environment. Using Self Play instead should lead to a less overfitting and more consistent performance.
Evaluating a TD3 agent trained against the hard opponent shows that overfitting still occurs. Where this agent reaches a winrate of 97% against the hard opponent, against a weak opponent he only has a winrate of 79%. We tried to use self-play to reduce the overfit and get a more consistent agent. However, learning totally from scratch, especially in an environment with sparse rewards, would be a great challenge for the agent. Therefore, it is trained for a small number of epochs against an opponent from the environment to figure out basic acting in the environment. Then training continues with a simple self-play formulation, in which the agent just trains against himself.
Table 1 shows the results of this technique. While the difference in performance is smaller for a self play agent,the overall performance against the strong agent drops significantly.

Opponent/TD3 agent	normal agent	self play
weak opponent	97	64
strong opponent	79	54

Table 1: Winrate (in %) against a weak/strong opponent. Self-play agents have a smaller difference than normally trained agents, but also a worse overall performance

When evaluating the self-play agent against the normal agent however, it wins 91% of its games, which means even though the self-play did not benefit the performance against the environment opponents, it clearly improved the general performance.

2.1.4 Evaluation of final agent

The final agent was trained for 10000 epochs against a weak opponent plus an added closeness-penalty with a factor of 10. Self-Play was not used, as this would decrease the performance against the specific opponent. The training resulted in a mean reward of 9.55 a winrate of 97.7% and a loose rate of 1.2% against the weak opponent in normal mode.

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] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[3] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*, 2019.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[5] 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.