Air Hockey Using Deep Reinforcement Learning

*

1st Apoorva Mani, 2nd Aishwarya Virigineni, 3rd Maturi Tanuj, 4th Amudha J.
[1,2,3,4] *Department of Computer Science and Engineering*
Amrita School of Computing, Bengaluru, India
bl.en.u4aie19007@bl.students.amrita.edu,bl.en.u4aie19068@bl.students.amrita.edu,bl.en.u4aie19041@bl.students.amrita.edu,
j-amudha@blr.amrita.edu

***Abstract*—The performance of reinforcement learning algorithms is quite strong when playing difficult board games and computer games, lately. The ability of reinforcement learning algorithms to generalise their findings is the subject of an increasing number of research projects. The objective of the project is to create a game that entails air hockey. The environment consists of a two-player hockey game, in which the agents play against each other in order to secure maximum goals score goals against each other on a table using their respective disc strikers and a puck. The agent's aim is to block the opponent's attack as much as possible and also strike in such a way that it scores a goal. The two players in the game are the two agents. Agents are expected to attain the essential skills to play the game. The main goal is to train the agents to learn how to play the game. Even though The game might seem simple and easy, the environment consists a lot of obstacles and complexities within itself. At the end of each trial, reward will be provided accordingly and the correct policy learning will be directed.**

***Index Terms*—Dueling DQN, Deep Deterministic Policy Gradient, Soft Actor Critic Approach, Deep Reinforcement Learning**

I. Introduction

Two player simultaneous action games are a significant category of games to take into account in reinforcement learning. predator-prey games, Rock-Paper-Scissors, iterated matrix games, Pong and many real-world situations like corporate competition or wars are typical examples. Additionally, as we will demonstrate later, many team games in which agents on different teams have the same goal are also adaptable to this situation. Sports like basketball and soccer as well as video games like DOTA or Overwatch fall under this category.

Action games for two players at once and multi-agent games in general present some particular difficulties. The fact that the environment is non-stationary from the perspective of any single agent presents the biggest and most fundamental challenge. This is referred to as the moving-target issue. Credit allocation presents another challenge; how can any specific agent determine whether the prize they earned was the result of their own efforts or merely the errors of their rivals? The constraint of dimensionality is a more pervasive problem in machine learning and reinforcement learning in particular. spaces. Robustness of policy is a final obstacle. It is challenging to guarantee that a specific action or policy will succeed regardless of how the other agent behaves. It happens rather frequently in various games that a specific action will only work well against a specific opponent. Although agents can continue to use a Nash equilibrium strategy, this frequently results in overly cautious behaviour that has little chance of outperforming breaking even. Finding and acting on an opponent's flaws is frequently viable and effective, especially in two player games when considerably more work may be invested into modelling one's opponent. A major driving force behind the algorithms we provide is learning effective policies while also taking advantage of enemies' flaws. Since more agents produce exponentially larger state and action spaces, this challenge is made worse in multi-agent games. Robustness of policy is a final obstacle. It is challenging to guarantee that a specific action or policy will succeed regardless of how the other agent behaves. It happens rather frequently in various games that a specific action will only work well against a specific opponent. Although agents can continue to use a Nash equilibrium strategy, this frequently results in overly cautious behaviour that has little chance of outperforming breaking even. Finding and acting on an opponent's flaws is frequently viable and effective, especially in two player games when considerably more work may be invested into modelling one's opponent. A major driving force behind the algorithms we provide is learning resilient policies while also taking advantage of adversaries' flaws.

Every interaction between an independent framework and its environment is driven by the difficulty of mastering a skill, a mapping between actions and states to achieve a goal in a continuous space. In this essay, we investigate how an agent may learn to efficiently defend against and strike the puck in an air hockey game. Two players will be playing against one another on a low-friction table in this game, which could be quiet and tough. The players must master techniques like blocking and striking if they want to win and score as many points as they can. The traditional methods for hitting the puck entail multiple stages of planning and execution. Start by developing a workable plan based on the present objective.

II. Literature Survey

Study on sports robotics has been going on for quite some time, and numerous research on the topic have been carried out in the most recent few years. Particularly, research has been conducted and published on the application of deep reinforcement learning to the sports of table tennis [6, badminton [7], sword fighting [8], and air hockey [9]. Automated

Fig. 1. Laser Hockey Game

algorithms that can compete effectively with humans need to be capable of significant real-time recognition as well as high-speed movement.

Solves the laser-hockey gym environment with Reinforcement Learning (RL) using the Twin Delayed Deep Deterministic policy gradient algorithm (TD3) [2]. In addition to that, prioritized experience replay (PER) [3] and observation normalization [4] have been implemented to analyze the effect of these modifications on TD3 in the laser-hockey environment.[4] illustrates how to create a policy for all-purpose robotic manipulators for the game of air hockey using two Kuka Iiwa 14. a technique for optimization that makes use of the manipulator's redundant mechanisms for dealing with joint velocity constraints. while an optimization algorithm on a desktop computer runs in real-time[4]. The technique coupled a temporally correlated noise with E-greedy exploration [5]. The Twin Delayed Deep Deterministic Policy Gradient algorithm (TD3), an actor-critic algorithm that takes into account the interaction between function approximation error in both policy and value updates, is created by applying our modifications to the state-of-the-art actor-critic method for continuous control, Deep Deterministic Policy Gradient algorithm (DDPG) (Lillicrap et al., 2015).

III. Formulation

A. Reinforcement Learning

Its primary objective is to build an experience history for the AI, or "agent," as it is called in the field, through multiple cycles of the allowed actions in the environment, analyzing the outcomes of such acts, and applying this experience to future actions. a procedure of trial-and-error that is well suitable to an autonomous agent that operates without human intervention. To learn to play air hockey effectively, an AI agent must play numerous rounds of the game, evaluate what and all actions work best for a source image, and thereafter alter its behavior to raise the frequency of the actions that work for the input, and vice versa.

In Air Hockey, reinforcement learning uses rewards to determine which actions to perform, and the reward is a +1 for every round the agent wins, and a -1 for every round the competitor agent wins. Real-world applications might be difficult to compute, particularly when there is no evident single score or aim. In some games, such as Space Invaders, rewards might be related to defeating distinct types of aliens.

Reinforcement learning describes a policy as a rule, strategy, or behavior function that evaluates and proposes the next action given a particular state. It is essentially a map from the state to the action. Since the main objective of Reinforcement learning agents is to maximize their rewards, we want to choose the policy that maximizes the future expected reward for a given action, whether it is deterministic or stochastic.In the case of Air Hockey, the policy helps the agent choose an action, which can be either moving the striking, blocking, or not doing anything at all.

A Markov Decision Process (MDP) is used in Reinforcement Learning, It is a state graph consisting of different nodes, each of which represents a possible state in an environment, and links and edges between those states, which represent possible actions that an agent might take, their probability, and their rewards.There are also two possible state transition strikes coming out of each configuration of the mallets and puck, and every state has a reward of 0, except at the terminal state, which could have a reward of +1 or -1. Air hockey can also be modeled as an MDP. By formulating our problem as a MDP, we can use the Markov property, which states that the future is independent of the past when we consider the present.

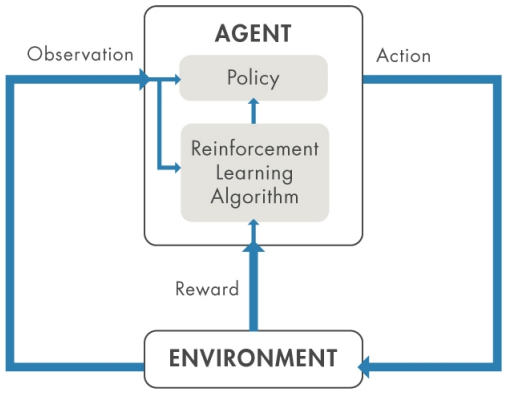

Fig. 2. A Pictorial Representation of the Reinforcement Learning Model

A key insight of MDPs is that the present state characterizes the problem fully and allows the agent to make the best decision possible, independent of past experiences, because past experiences can be encoded in the present.

When selecting an action to try to transition the agent into a future state, value functions can be used to evaluate the quality of a state or state-action pair. Value functions come in several types, each based on a different assumption about the policy.

By determining how good an action is relative to another, we can determine whether our RL agent learns enough to perform well at the task, and not whether it is absolutely good.

B. TD3

TD3 comes after and succeeds the Deep Deterministic Policy Gradient (DDPG). Up until recently, for situations involving continuous control, such as those encountered in robotics and autonomous driving, one of the algorithms that

saw the most application was DDPG. In spite of the fact that it is capable of delivering outstanding results, DDPG does have some downsides. Training DDPG, like training many other RL algorithms, can be unstable and is highly dependent on finding the appropriate hyper parameters for the task in hand. This is due to the algorithm repeatedly overestimating the Q values of the critic network. These estimating errors compound over time, which might result in the agent reaching a local optimal solution or suffering catastrophic forgetting. In order to solve this problem, TD3 places its primary emphasis on lowering the overestimation bias that was present in earlier algorithms. This is accomplished through the incorporation of the following three essential elements:

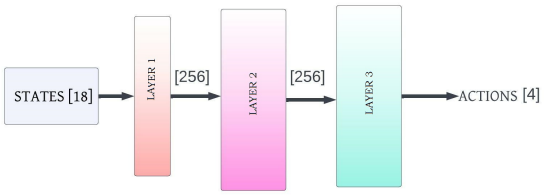

Fig. 3. Actor Architecture

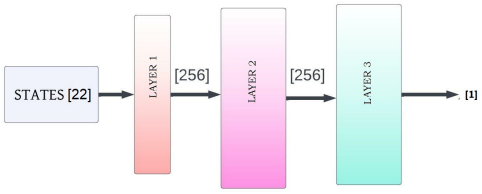

Fig. 4. Critic Architecture

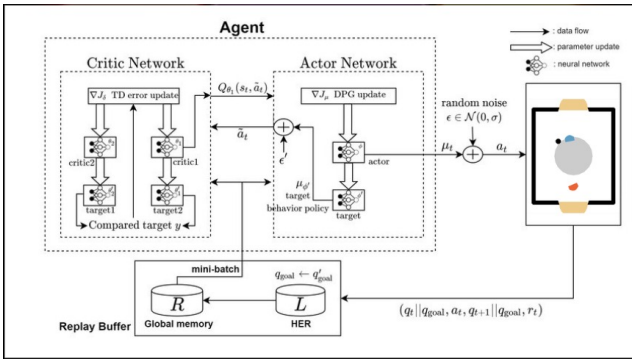

Fig. 5. Twin Delayed Deep Deterministic Policy Gradient Architecture

- Using a pair of critic networks
- Delayed updates of the actor
- Action noise regularisation

a) ***Twin Critic Networks****:* The utilisation of two separate critic networks is the first new feature introduced in TD3. The method used in Deep Reinforcement Learning with Double Q-learning (Van Hasselt et al., 2016), which involves predicting the current Q value using a distinct target value function in order to reduce bias, served as an inspiration for this. On the other hand, the method is not without its flaws when used to actor critic methods. This is due to the fact that both the policy and the target networks are only updated very infrequently, making them appear quite similar to one another and bringing bias back into the picture. An older implementation, which can be found in Double Q Learning (Van Hasselt, 2010), is utilised instead. The clipped double Q learning method is utilised by TD3, and this method selects the critic network with the lower value.

The Q values are more likely to be underestimated when using this strategy. The fact that the low values will not be propagated through the algorithm, in contrast to the overestimate values, means that the underestimating bias is not a concern. This results in a more stable approximation, which in turn contributes to an improvement in the algorithm's overall stability. When determining its targets, TD3 makes use of the critic network with the lower value, which is comprised of two distinct networks.

b) ***Delayed Updates****:* Although target networks are a useful technique for bringing consistency into an agent's training, actor critic approaches provide various challenges that must be overcome because of the nature of the method. The interaction between the policy (actor) networks and the critic (value) networks is the root cause of this phenomenon. Whenever an ineffective policy is exaggerated, the agent's training goes in a different direction. Because it is currently updating on states where there are many errors, our agents' policy will then continue to deteriorate. In order to rectify this situation, all that is required of us is to do updates on the value network more frequently than on the policy network. Before the value network is used to update the policy network, this enables the value network to become more stable and lower the number of errors it produces. In actual fact, the policy network is only updated after a predetermined number of time steps, whereas the value network continues to update itself after each and every time step. These less frequent policy updates will result in value estimates that have a reduced variation, and as a result, a better policy should be the end result. TD3 makes advantage of a delayed update of the actor network; rather than updating it after each time step, it is only updated every 2 time steps. This results in training that is both more stable and more effective.

c) ***Noise Regularisation****:* Smoothing the targeted policy is the topic of discussion during the final section of TD3. When updating the critic, deterministic policy techniques have a propensity to generate target values that have a large degree of volatility. This occurs as a result of the value estimate being overfit to spikes in the data. Target policy smoothing is a method of regularisation that is utilised by TD3 in order to bring this variance down to a more manageable level. In an ideal world, there would be no difference between the target values, and acts that were very similar would receive very comparable values. This volatility is reduced by TD3 with the application of a tiny amount of random noise to the target and through the application of averaging over mini batches. In order to maintain a goal value that is somewhat near to the action that was first performed, the range of the noise is trimmed.

Policies have a tendency to be more stable as a result of the inclusion of this additional noise to the value estimate. This is because the target value is returning a higher value for activities that are more resistant to noise and interference. During the process of computing the targets, clipped noise is added to the action that was selected. This gives higher priority to actions that have higher values and are more reliable.

C. Prioritized Experience Replay

Experience replay is yet another important component of DQN's success. As it learns, the agent builds up a dataset $t = o_1, o_2, ..., o_t$ of experience called $\emptyset_t(t, a_t, r_t, s_t+1)$, using which we sample minibatches uniformly to train the agent. Experience replay vastly reduces the variance as sampling decreases the interconnection between the given samples used in the next iteration. from the buffer. However, regardless of their importance, this method just reruns transitions at the same frequency as when they were first experienced. To overcome this, [Schaul et al., 2015] introduced Prioritized Experience Replay, where samples are sampled with probability proportional to the respective TD-error. In particular, the sampling transition $\imath_s$ probability is defined as:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \tag{1}$$

with priority pi = $\delta_i + \epsilon \geq 0$. Stochastic updates that have the same distribution as the anticipated value are necessary for the estimate of the expected value using these updates. Prioritized replay increases bias because it uncontrollably modifies this distribution, which modifies the response that the estimates will eventually converge to (even if the policy and state distribution are fixed). By utilising importance-sampling (IS) weights, we can eliminate this bias.

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)^\beta \tag{2}$$

The unbiasedness of the updates is crucial in a typical reinforcement learning situation approaches convergence at the conclusion of training because the process is highly non-stationary due to shifting policies, state distributions, and bootstrap goals. Because of this, it is usual to define a (linear) schedule on the exponent to anneal the amount of importance-sampling adjustment over time.

D. Implementation of TD3

The preceding pseudo code has been divided into logical steps that you can use to carry out the TD3 algorithm as follows:

a) ***Initialise Networks****:* Both the Actor and the Critic networks often operate with a configuration similar to this one. In reality, the critic class incorporates both of the aforementioned networks into its structure. The Q values for both critics are returned by the forward() method of the critics, which can be used later. The getQ method is quite straightforward and only returns the very first critic network.

b) ***Initialise Buffer****:* This is a typical buffer for playing back replays..

c) ***Select Action with Exploration Noise****:* This is a typical stage in the Markov decision process that the environment goes through. At this point, the agent will select an action, and exploration noise will be added to it.

d) ***Store Transitions****:* We record the information for that time step in the replay buffer after performing an action. We'll use these transitions later when we update our networks.

e) ***Update Critic****:* Our model is trained across a number of iterations after we have completed a full time step over the environment. The update's critic is involved in the initial step. The majority of the TD3 extra features are implemented here, making it one of the algorithm's most crucial components. The first step is to sample a small batch of transitions that have been placed in the replay buffer. The target policy smoothing will then be applied to the action we have chosen for each of the states that we have included in our mini batch. As previously mentioned, all that is required for this is the target actor network to select an action, and then we add noise to the clipped action to make sure that the noisy action doesn't deviate too far from the original action value.

Calculating our goal Q values for the critic comes next. The double critic networks are useful in this situation. The Q values for each target critic will be obtained, and the target Q value will be the smaller of the two. The loss for the two active critic networks is then calculated. The desired Q value we just determined and the MSE of each existing critic are obtained to accomplish this. The critic's optimization is then completed as usual.

f) ***Update Actor****:* Comparing the actor to the critic, the actor is much easier to update. First, we confirm that we are updating the actor no more frequently than every d time steps. The actor was updated every second time step in both our case and the paper. Our actor decides what action to take based on the micro batch of states, and the loss function simply takes the mean of the -Q values that are provided by our critic network. Backpropagation, as it was before, is the method by which we optimise our actor network.

g) ***Update Target Networks****:* At long last, we do a gentle upgrade on our previously frozen target networks. This is done simultaneously with the actor update, which also causes a delay.

IV. RL Model for the Application

A. Environment

Laser-hockey gym environment which was taken from Open-AI gym for the two-player air hockey game, in which the agents play against each other in order to secure maximum goals score goals against each other on a table using their respective disc strikers and a puck. The agent's aim is to block the opponent's attack as much as possible and also strike in such a way that it scores a goal. The two players in the game are the two agents.

B. State Space

1) x pos player one
2) y pos player one
3) angle player one

Algorithm 1 TD3

1 Initialize critic networks $Q_{\theta_1}, Q_{\theta_2}$, and actor network π_ϕ with random parameters θ_1, θ_2, ϕ
Initialize target networks $\theta'_1 \leftarrow \theta_1, \theta'_2 \leftarrow \theta_2, \phi' \leftarrow \phi$

2 Initialize replay buffer $\mathcal{B}$

for $t = 1$ **to** T **do**

3 Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward r and new state s'

4 Store transition tuple (s, a, r, s') in $\mathcal{B}$

5 Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
$\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
$y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$
Update critics $\theta_i \leftarrow \text{argmin}_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$

6 **if** t mod d **then**
Update ϕ by the deterministic policy gradient:
$\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$

7 Update target networks:
$\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i$
$\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$

end if

end for

Fig. 6. The TD3 algorithm with emphasis placed on essential areas

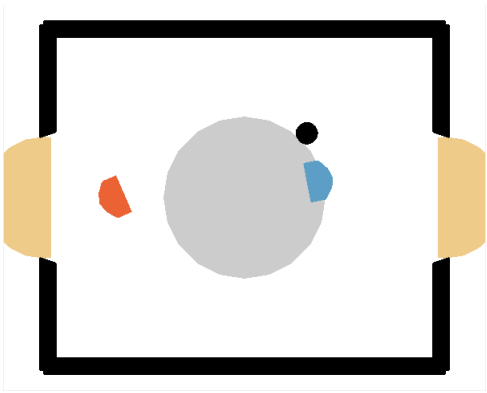

Fig. 7. Laser Hockey Gym Environment

4) x vel player one
5) y vel player one
6) angular vel player one
7) x player two
8) y player two
9) angle player two
10) y vel player two
11) y vel player two
12) angular vel player two
13) x pos puck
14) y pos puck
15) x vel puck
16) vel puck
17) Keep Puck Mode
18) time left player has puck
19) time left other player has puck

C. Action Space

The set of all the possible actions is what the ”action space” (also known as ”the action set”) describes. Regarding the undertaking, we will use the discrete action space, which is a group of four different actions, i.e., at 0, 1, 2, 3. This indicates movement of striker: move right, move left, move up and move down respectively.

Fig. 8. Action space

D. Reward Function

The reward function determines the reward received as a result of a trajectory’s activities. The agent’s performance is totally dependent on the rewards it receives for its actions, therefore having the optimal reward function is crucial. We use the infinite-horizon discounted return, whose discount factor is a coefficient. The discount factor, which ranges from zero to one, reflects how far into the future the rewards were received. R( ) = X t=0 trt (3) Following each and every decision, the game simulator will carry out the action and provide us a reward. The reward will either be +1 if the ball was successful in getting past the opponent, -1 if it was unsuccessful, or 0 if it was not successful at all. It goes without saying that we want to move the paddle so that we can accumulate a lot of rewards.

E. Neural Fitted Q iteration algorithm(NFQ)

There are total seven steps or components involved for implementing NFQ algorithm to solve different Deep reinforcement learning problems. These seven components are chosen to solve our Air Hockey game using Deep Reinforcement learning and they are listed below.

First decision point is to select a value function - where we are selecting the action value function Q(s,a;). Second decision point is to select a neural network architecture where choosing the state in Values out architecture which is more efficient compared to other architectures. Third decision point is to selecting what to optimize to achieve our goal where we will optimize the action value function to get the approximate optimal action-value function q*(s,a). Fourth decision point is selecting the target and we are choosing TD target off policy version Q-learning target. Fifth decision point is selecting an exploration strategy where we will implement the Epsilon greedy exploration strategy to maintain the balance between exploration and exploitation. Sixth decision point is to select a loss function to measure how accurate are the neural network predictions where we will be using Huber Loss Function as loss function. Seventh decision point is selecting an optimization method for the neural network where we will used the Adam optimizer.

These are all the seven components we will be using for our project to implement NFQ algorithm to solve our Air Hockey game using deep reinforcement learning problem.

V. EXPERIMENTS

In this section we will perform an ablation/sensitivity study, where we explore what effect certain hyperparameters and architecture choices have on the performance. The metric of interest is the percentage of won games on 10,00,000 timesteps. We perform an evaluation rollout after training episodes in order to track the progress of each algorithm. For the purpose of the task, we train a feed forward dueling architecture using prioritized experience replay, with the relu function and tanh activation function. Moreover, we use three schedulers: a step scheduler that halves the learning rate at predefined checkpoints; a linear scheduler that anneals beta (for Prioritized Experience Replay) to 1; and a linear scheduler that decreases epsilon (for epsilon-greedy) to 0.1 in an attempt to mitigate the exploration-exploitation dilemma. In total, there are well over 20 hyper-parameters that can be tuned, which was not a trivial task. In the rest of this chapter, due to space constraints, we only cover a small portion of the intuitions gained in training the agent to beat the Opponent.

TABLE I
DEFINING HYPER PARAMETERS

Hyperaparameter	Default Value
start timesteps	5e4
eval freq	5e3
self play freq	15e5
max timesteps	1e6
max episode timesteps	500
max buffer size	1e6
expl noise	0.15
hidden dim	256
batchsize	256
learning rate	3e-4
discount	0.99
alpha	0.6
beta	1.0
tau	0.01
policy noise	0.1
noise clip	0.5
policy freq	2

VI. RESULTS

A. Performance evaluation against the weak and strong opponent

- TD3: Win-rate = 0.87, Tie-rate = 0.12, Loss-rate = 0.01
- TD3 + PER: Win-rate = 0.54, Tie-rate = 0.41, Loss-rate = 0.05
- TD3 + ON: Win-rate = 0.94, Tie-rate = 0.04, Loss-rate = 0.02

When evaluating against weak and strong opponent, TD3 with observation normalization has proved to give better results with a high win-rate and less loss-rate and tie-rate. Regular TD3 has given a better result in terms of win-rate and loss-rate compared to TD3 with prioritized experience replay.

B. Performance evaluation against the weak opponent

- TD3: Win-rate = 0.97, Tie-rate = 0.03, Loss-rate = 0.00
- TD3 + PER: Win-rate = 0.99, Tie-rate = 0.01, Loss-rate = 0.00

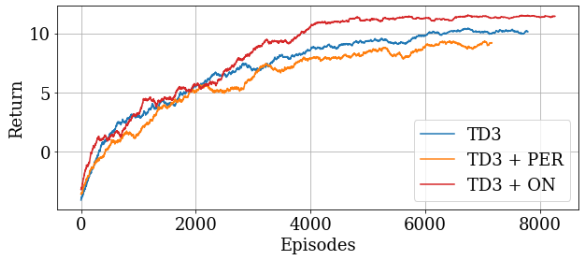

Fig. 9. Episode return during training - Performance evaluation against the weak and strong opponent

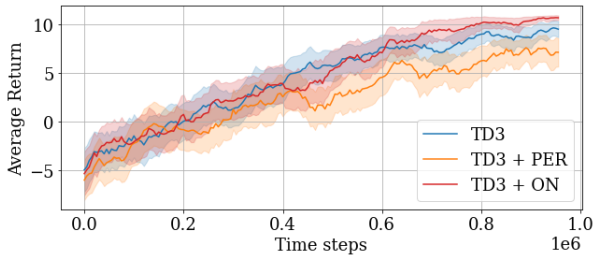

Fig. 10. Average episode return in the evaluation during training - Performance evaluation against the weak and strong opponent

- TD3 + ON: Win-rate = 0.95, Tie-rate = 0.05, Loss-rate = 0.00

When evaluating against the weak opponent, TD3 with prioritized experience replay has proved to give better results with a high win-rate and less tie-rate followed by regular TD3 and TD3 with observation normalization.

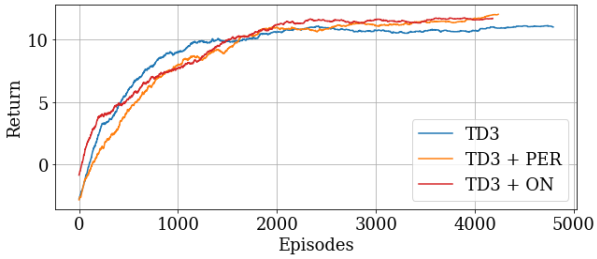

Fig. 11. Episode return during training - Performance evaluation against the weak opponent

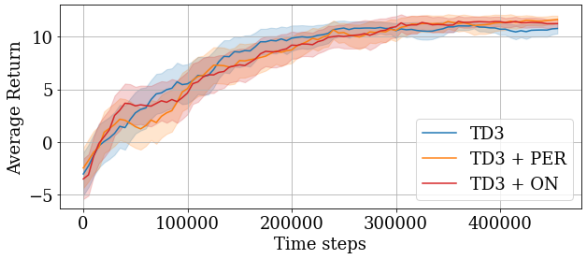

Fig. 12. Average episode return in the evaluation during training - Performance evaluation against the weak opponent

C. Performance evaluation of batch sizes

- batch size = 100: Win-rate = 0.87, Tie-rate = 0.12, Loss-rate = 0.01
- batch size = 256: Win-rate = 0.99, Tie-rate = 0.01, Loss-rate = 0.00

When evaluating based on the batch sizes, it is seen that with higher batch-size, better were the results.

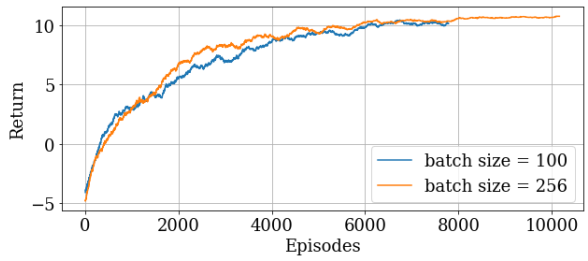

Fig. 13. Episode return during training - Performance evaluation of batch sizes

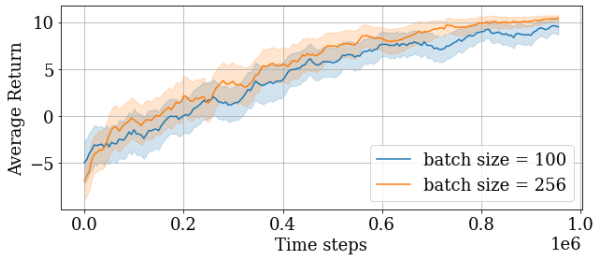

Fig. 14. Average episode return in the evaluation during training - Performance evaluation of batch sizes

D. Performance evaluation of hidden dimensions

- hidden dim = 128: Win-rate = 0.40, Tie-rate = 0.53, Loss-rate = 0.07
- hidden dim = 256: Win-rate = 0.87, Tie-rate = 0.12, Loss-rate = 0.01
- hidden dim = 512: Win-rate = 0.82, Tie-rate = 0.17, Loss-rate = 0.01

When evaluating based on hidden dimensions, it is seen that a moderate value of 256, has produced better results compared to 128 and 512.

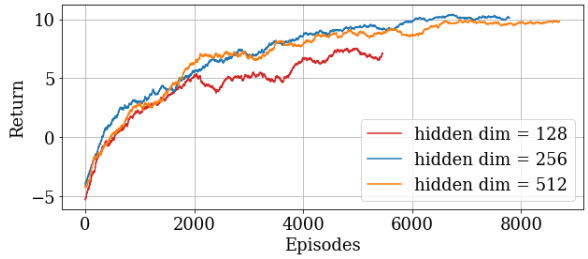

Fig. 15. Episode return during training - Performance evaluation of hidden dimensions

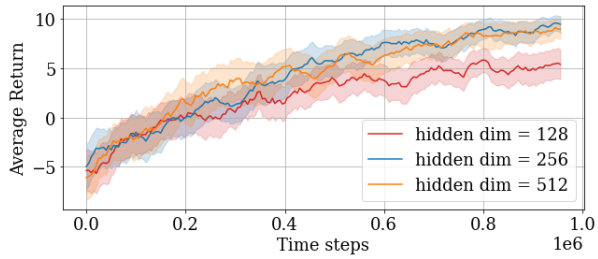

Fig. 16. Average episode return in the evaluation during training - Performance evaluation of hidden dimensions

E. Performance evaluation of the target update proportion tau

- tau = 0.0025: Win-rate = 0.52, Tie-rate = 0.45, Loss-rate = 0.03
- tau = 0.005: Win-rate = 0.87, Tie-rate = 0.12, Loss-rate = 0.01
- tau = 0.010: Win-rate = 0.96, Tie-rate = 0.03, Loss-rate = 0.01

When evaluating based on the target update proportion tau, it is seen that as the value of tau increase, the results got better.

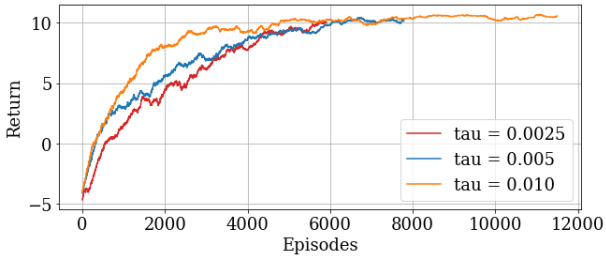

Fig. 17. Episode return during training - Performance evaluation of the target update proportion tau

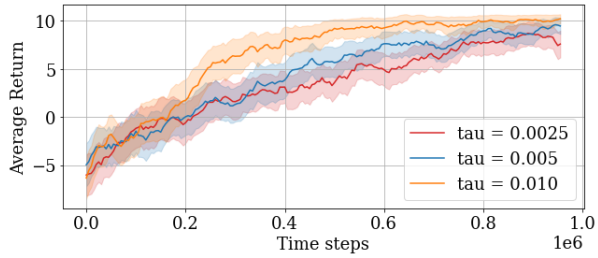

Fig. 18. Average episode return in the evaluation during training - Performance evaluation of the target update proportion tau

F. Performance evaluation of the policy noise sigma tilde

- sigma = 0.1: Win-rate = 0.99, Tie-rate = 0.01, Loss-rate = 0.00
- sigma = 0.2: Win-rate = 0.87, Tie-rate = 0.12, Loss-rate = 0.01
- sigma = 0.4: Win-rate = 0.74, Tie-rate = 0.25, Loss-rate = 0.01

When evaluating based on the policy noise sigma tilde, it is seen that as the value of sigma decreases, the results got better.

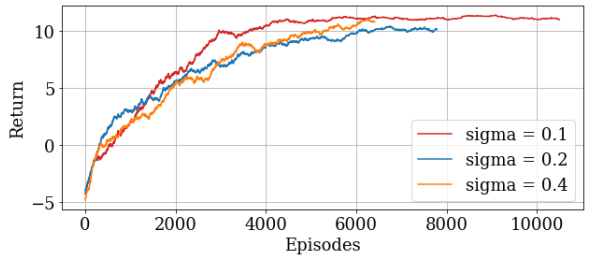

Fig. 19. Episode return during training - Performance evaluation of the policy noise sigma tilde

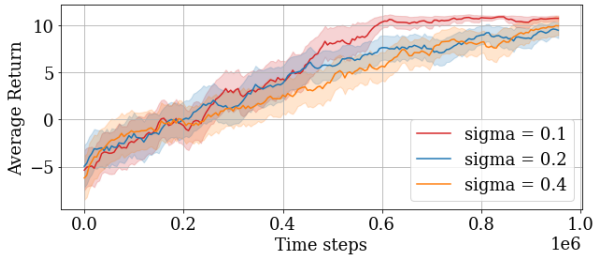

Fig. 20. Average episode return in the evaluation during training - Performance evaluation of the policy noise sigma tilde

G. Performance evaluation of the TD3 with self-play

- TD3 with self-play: Win-rate = 0.99, Tie-rate = 0.00, Loss-rate = 0.01

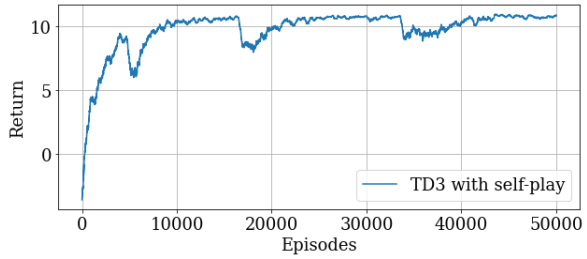

Fig. 21. Episode return during training - Performance evaluation of the TD3 with self-play

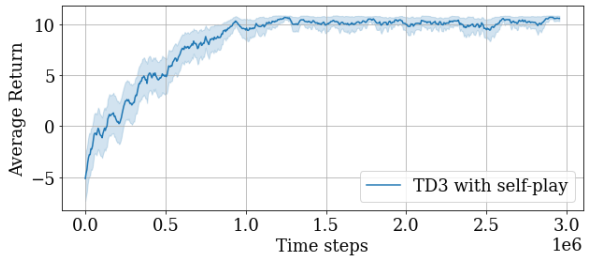

Fig. 22. Average episode return in the evaluation during training - Performance evaluation of the TD3 with self-play

VII. CONCLUSION

The performance of reinforcement learning algorithms has been quite strong when playing difficult board games and computer games. In this project, Twin Delayed Deep Deterministic policy gradient algorithm (TD3) algorithm has been implemented which can be used to train two agents (i.e. two independent neural networks) to play the game Air Hockey against each other in the laser-hockey gym environment. Therefore the agents play the game and save the experience into a replay memory which poses as a data set for training the neural networks. However, remarkably the TD3 algorithm does not require a supervisor, but instead uses an old version of itself to label the data. An updating scheme derived from the Bellman equation is the used to train both agents. With this approach agents have been trained that were able to consistently reach the maximal game time of 1e6 time steps. It needs to be emphasized that the TD3 algorithm described in this report does not know any of the internal mechanisms of the game. With very little changes the same algorithm could be used to learn all sorts of different environment, e.g. other game or also real-life scenarios. Also the algorithm does not depend on the network architecture which could be exchanged with any other.

In addition to TD3, prioritized experience replay (PER) and observation normalization (ON) have been implemented to analyze the effect of these modifications on TD3 in the laser-hockey environment. The result graphs shows the most important evaluation results conducted during training of the above agents. It includes the influence of certain hyperparameters, the influence of the modifications prioritized experience replay and observation normalization.

REFERENCES

[1] S. Fujimoto, H. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. ICML, 2018.
[2] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. arXiv preprint arXiv:1511.05952, 2015.
[3] M. Andrychowicz, A. Raichuk, P. Stanczyk, et al. What matters in on-policy reinforcement learning? A large-scale empirical study. arXiv preprint arXiv:2006.05990. 2020.
[4] Liu, Puze, Tateo, Davide, Bou Ammar, Haitham, Peters, Jan. (2021). Efficient and Reactive Planning for High Speed Robot Air Hockey. 586-593. 10.1109/IROS51168.2021.9636263.
[5] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver,D., and Wierstra, D. Continuous control with deep reinforcement learning. arXivpreprint arXiv:1509.02971 (2015).
[6] K. Yamada, "Robot table tennis tutor as an example of "harmony between human and robot"," J. of The Institute of Electrical Engineers of Japan, Vol.137, No.2, pp. 81-84, 2017 (in Japanese).
[7] S. Mori, K. Tanaka, S. Nishikawa, R. Niiyama, and Y. Kuniyoshi,"High-speed and lightweight humanoid robot arm for a skillful badminton robot," IEEE Robotics and Automation Letters, Vol.3, No.3,pp. 1727-1734, 2018.
[8] W. Chen, T. Liao, Z. Li, H. Lin, H. Xue, L. Zhang, J. Guo, and Z. Cao, "Using ftoc to track shuttlecock for the badminton robot,"Neurocomputing, Vol.334, pp. 182-196, 2019.
[9] A. Namiki and F. Takahashi, "Motion generation for a swordfighting robot based on quick detection of opposite player's initial motions," J. Robot. Mechatron., Vol.27, No.5, pp. 543-551, 2015