RL-Course 2024/25: Final Project Report

Sahiti Chebolu, Surabhi S Nath

1 Introduction

Over the years, deep reinforcement learning (DRL) has proven to be a powerful technique for training autonomous agents to make sequential decisions in complex environments [8], particularly in games [9, 12], robotics [7, 13], and more recently in improving reasoning in large language models [5].
In this report, we attempt to build DRL agents to play the game of laser hockey. A custom laser hockey environment `hockey` was developed by the Martius lab wherein two agents compete to hit a puck in the opposite goal. Alongside, we also test the performance of our agents in two other solved environments, namely Pendulum `Pendulum-v1`, where the goal is to keep the pendulum vertically upright and CartPole `CartPole-v0`, where the goal is to balance the pole in the cart and prevent it from falling.
DRL algorithms can be classified along several dimensions. Model-based methods (e.g., AlphaZero, World Models) learn environment dynamics for sample efficiency, whereas model-free methods (e.g., DQN, PPO) directly optimize policies or value functions. Value-based methods (e.g., DQN) estimate action values, while policy-based methods (e.g., PPO) optimize policies directly, with actor-critic approaches (e.g., DDPG, TD3) combining both. On-policy methods (e.g., PPO, TRPO) learn from the current policy, ensuring stability, whereas off-policy methods (e.g., DQN, TD3) use past experiences, improving sample efficiency. Lastly, discrete-action algorithms (e.g., DQN) are suited for finite action spaces, while continuous-action algorithms (e.g., DDPG, SAC) can handle robotic control.
Our agents are based on DQN, a model-free, off-policy, value-based algorithm, using a discrete action space. Even though the action space for Laser Hockey and Pendulum are continuous, we discretize the action space and opt for DQN due to its ease of training, stability and proven success in high-dimensional problems such as Atari games [10]. Particularly, we extend the implementation of DQN with modifications that fix its shortcomings or aim to make improvements to aspects such as prioritization (Prioritized Experience Replay (PER), Dueling DQN), overestimation (Double DQN), multi-step learning and exploration (Random Network Distillation (RND)) [6, 1].
We find that for simple environments such as `Pendulum-v1` and `CartPole-v0`, DQN is successfully able to attain high performance, and the modifications implemented are able to maintain the performance or enhance it slightly. On the other hand, for more complex environments such as `hockey`, the modifications and complexities of agents matter a lot, with particular combinations achieving high performance. We found our Dueling DQN + PER agent to be most successful compared to all other agents at playing hockey, achieving 83% wins over the weak agent and nearly no losses. Interestingly, different modifications resulted in different behavioural strategies adopted by the agents.
Our code, agents and GIFs are available on GitHub.

2 Methods

Deep Q-Network (DQN) DQN implements classic Q-learning for continuous state spaces, using neural networks for approximating Q-values of state-actions ($Q(s, a, \theta)$).

DQN consists of a neural network that takes in continuous states as inputs and returns Q-values for each discrete action. The network weights (θ) are trained by stochastic gradient descent (SGD) with the following objective:

$$L(\theta) = \mathop{\mathbb{E}}_{s,a,r,s'} \left[(r + \gamma \max_{a'} Q(s', a', \theta) - Q(s, a, \theta))^2 \right]$$

giving the following gradient:

$$\nabla_\theta L(\theta) = \mathop{\mathbb{E}}_{s,a,r,s'} \left[(r + \gamma \max_{a'} Q(s', a', \theta) - Q(s, a, \theta)) \nabla Q(s, a, \theta) \right]$$

The updates are off-policy, where the policy for learning the Q-function (here, it is greedy) is different from the policy for action-selection (we implement ϵ-greedy rule). This allows using a replay buffer to sample random mini-batches of experiences (s, a, r, s') for training by SGD. Apart from the main network, there is a target network with the same architecture. The target network weights are updated every few iterations. This is because the target Q-values for learning are derived from the target network and keeping the network constant prevents rapid oscillations or diverging policies [10].
Several useful modifications can be implemented on top of DQN for performance enhancement. A set of such modifications were proposed in Hessel et al. [6], some of which we apply and describe below.

Prioritized experience replay (PER) In DQN, experience replay samples transitions uniformly, leading to inefficient learning. To tackle this, Schaul et al. introduce a novel method to improve sample efficiency in reinforcement learning by prioritizing experiences based on their learning potential [11]: the probability of sampling a transition i is determined by the magnitude of its temporal-difference (TD) error:

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

where $p_i = |\delta_i| + \epsilon$ ensures nonzero probabilities, δ_i is the TD error, and α controls the strength of prioritization, and was set to 0.5 across all our experiments. Since prioritization introduces bias, importance-sampling weights are used to correct it:

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

β is usually annealed from 0 to 1 to ensure an unbiased estimate in the later stages of training, but we fixed it to 0.5 throughout. By prioritizing rare but informative experiences, PER trains the agent effectively by reducing the number of samples required.
PER can be efficiently implemented using Sum Trees. A Sum Tree is a binary tree where each parent node stores the sum of the priorities of its children. A priority value is random sampled and the corresponding transition is retrieved from the tree. Compared to a simple list-based implementation, *i.e.* O(N) complexity for sampling, a Sum Tree reduces storage overhead and performs sampling (and updates) in O(logN).

Dueling Networks Dueling Q-networks (Dueling DQN) improve the stability and efficiency of learning by decomposing the Q-value function into two separate estimators: the *state-value function* $V(s)$ and the *advantage function* $A(s, a)$ [15]. Instead of directly estimating $Q(s, a)$, Dueling DQN approximates:

$$Q(s, a) = V(s) + A(s, a) - \max_{a'} A(s, a') \tag{3}$$

where $V(s)$ represents the value of being in state s and $A(s, a)$ represents the advantage of taking action a over the mean action in that state. This separation helps improve generalization across actions. By learning a better state-value estimate, Dueling DQN enables more stable and efficient training, particularly in large state spaces.

Double Q-learning DQN suffers from overestimation bias, where the max operator in the Bellman equation leads to optimistic Q-value updates. Double DQN addresses this by using separate networks for action selection and evaluation [14]. The target Q-value is modified as:

$$y = r + \gamma Q_{\text{target}}\left(s', \arg\max_a Q(s', a; \theta); \theta_t\right) \tag{4}$$

where Q (with parameters θ) is used to select the best action in state s' and Q_{target} (with parameters θ_t) is used to evaluate the value of this action. By decoupling the action selection from evaluation, Double DQN reduces overestimation bias, leading to improved performance.

Multi-step learning Standard DQN relies on one-step TD learning, where updates are based on immediate rewards and next-step predictions. Multi-step learning extends this by using n-step returns:

$$G_t^{(n)} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{n-1} r_{t+n-1} + \gamma^n Q(s_{t+n}, a_{t+n}) \tag{5}$$

For small n, updates are closer to one-step TD (higher bias, lower variance). For large n, updates resemble Monte Carlo estimates (lower bias, higher variance).
Multi-step learning accelerates convergence by propagating rewards more effectively across multiple time steps. We implement n = 3, 5, 10, 100 and full MC update. We test each for `Pendulum-v1` and n = 3 for `hockey`.

Random network distillation (RND) Random network distillation (RND) is a method to implement exploration bonuses for deep RL algorithms [1]. Base DQN already includes an ϵ-greedy behavioral policy that follows the greedy policy with probability $1 - \epsilon$ and a random action with probability ϵ. While this allows exploration of new actions for a state, it can also be advantageous to explore new states in the environment in a directed manner. As counting state visitations is not straightforward in continuous spaces, RND introduces a method to quantify novelty of states and encourages visitation of novel states through intrinsic rewards. This method has improved exploration and performance especially in environments with very sparse rewards [1].
RND consists of two neural networks: a target network with randomly initialised weights ($f(\mathbf{s})$)and a predictor network ($\hat{f}(\mathbf{s}, \theta)$) that will be trained to predict the output of the target network. Both networks take states in the RL environment as inputs. The weights of the predictor network are trained by gradient descent to minimise the MSE loss between the outputs of the two networks:

$$L(\theta) = ||f(\mathbf{s}) - \hat{f}(\mathbf{s}, \theta)||^2$$

This training is performed on the experience collected by the agent, that is, the states visited by the agent. Hence, the MSE loss will be lower for states the agent has visited many times (that is, for the inputs the predictor network has been trained to predict the output of the random network) and higher for novel states. Hence, this prediction error ($i_t = L(\theta_t)$) can be used as an intrinsic reward (or exploration bonus), in addition to the extrinsic rewards (e_t) from the environment, yielding a combined reward $r_t = e_t + i_t$ that can guide better exploration of the state space of the environment.

Action space discretisation DQN only supports learning Q-values for discrete actions. Hence, for `Pendulum-v1` and `hockey` environments that have continuous actions, we discretised the action space. `Pendulum-v1` consists of a 1-dimensional action space, namely torque ranging between $(-2, 2)$. We discretised this into 5 levels, giving 5 discrete actions. `hockey` consists of three continuous actions for moving in x-coordinates, y-coordinates and rotation as well as one discrete action for shoot (or not). We derived 8 discrete actions corresponding to doing nothing, moving $\pm$ 1 unit in x, y, angle and shooting. `CartPole-v0` already has only two discrete actions: moving left and moving right.

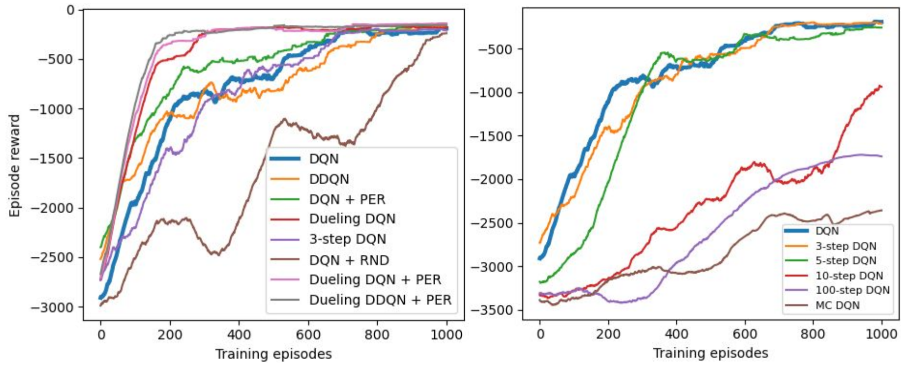
Model architectures We used fully connected neural networks for Q and Q-target networks in the DQN. The input layer size corresponds to the state space dimension in the environment: 3 in `Pendulum-v1`, 4 in `CartPole-v0` and 18 in `hockey`. The output size of the networks correspond to the number of discrete actions. We have two hidden layers with 100 units and use $tanh()$ as the nonlinear activation function. We use the same architecture for RND (although the output layer size doesn't necessarily have to correspond to the action number in RND).

Training and testing settings and procedures We used the same network architecture for all three environments. We trained the network using DQN (+ modifications) for 1000 episodes for `CartPole-v0`, 1200 for `Pendulum-v1` and 15000-20000 episodes for `hockey`. We had a maximum of 500 timesteps within each episode. We did 32 training iterations by sampling randomly from the replay buffer after every episode. The Q-target network was updated every 20 training iterations. During training, we set ϵ to 1.0 and decayed it by a factor of 0.98-0.999 every episode, up to a minimum of 0.01-0.2. We use Adam optimiser with a learning rate of 0.0002 (which we decay by a factor of 0.95 every 2000 training episodes for some algorithms). We set discount factor γ between 0.95-0.99 for different agents and environments. For RND, we use a learning rate 0.001 to update the predictor network weights.
Additionally, in `hockey`, we trained our agents to play against a weak agent (predefined), through curriculum learning, and finally, we trained our agents to play against themselves (self-play).
For `Pendulum-v1`, external rewards are from costs from deviating from $0°$and from non-zero angular velocity and torque (so, maximum reward is 0, when all three components are exactly 0). For `CartPole-v0`, the rewards are the number of timesteps lasted in the episode (so, maximum is 500 for us). In `hockey`, we included a penalty proportional to the distance from puck in own half, a binary reward if the puck is touched, reward/ penalty for direction of puck and a final reward of 10, 0 or -10 for win, draw and loss in the episode.
In RND, there is a problem with scale (of the intrinsic rewards) since the target weights are randomly initialised and hence the intrinsic rewards might be too big or too small for different environments. To deal with this, we normalised the intrinsic rewards by standard deviation of intrinsic rewards in the episode. For `hockey`, we also scaled the intrinsic rewards by a factor of 0.01 post-hoc to bring them to scale with the external rewards.
For testing, we save the weights of the agent after training and run it for 100 test episodes, where there is no more learning and with $\epsilon = 0$.

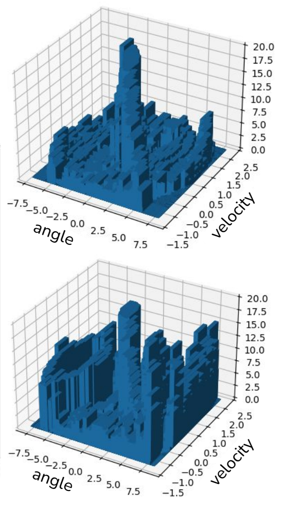
3 Results

Pendulum

The total episode rewards through the training episodes ('training trajectories') for `Pendulum-v1` are plotted in Figure 1a. We smoothen the training curves by calculating a running mean using a sliding window of size 200. The base DQN trajectory is presented in bold blue. All algorithms reach similar levels of episode rewards in 1200 episodes.
From Figure 1a (left), we see that DQN + PER and Dueling DQN and their combination learn faster (*i.e.* reach higher episode rewards earlier) compared to base DQN. This could be explained by the impact of prioritization which makes learning more efficient and hence faster.
Double DQN began by learning faster than DQN but then it's slope reduced below base DQN. This suggests that in continuous action spaces like Pendulum, overestimation may be less harmful, and overly conservative Q-value updates could lead to underestimation bias, slowing down learning. However, the combination of PER, Dueling network and Double DQN learns the fastest.
We also tested multi-step learning with different n values (n = 1, 3, 5, 10, 100 and full MC, 1a (right)). We see that for pendulum, the base DQN with n = 1 is indeed the best and as n increases, the performance



(a) Training curves for all (left) and multi-step (right) agents in `Pendulum-v1`.



(b) State visitation plots for DQN (top) and DQN + RND (bottom) agents

Figure 1: Results for `Pendulum-v1`

decreases. The full MC update performs the worst, only achieving a test performance of -2253.
The DQN + RND training trajectory shows a slower, more undulated pattern compared to the smooth increase in episode rewards with training in the other algorithms. One reason for this could be that the DQN + RND agent explores more states before reaching a solution due to the intrinsic rewards. In Figure 1b, we plot the number of times a state in the 2D state space is visited for base DQN and DQN + RND. The two states are angle and angular velocity which we discretised into bins of 100 to get this count. In the figure, we clipped the counts at 20, since the most visited states (around 0°angle and 0 angular velocity) are orders of magnitude ($\sim$10000) larger than for the other states. The RND agent has evidently explored more states other than the optimal one compared to the base DQN agent.
We tested all agents on 100 episodes in the same environment and the mean test score for each is shown in Table 1. We see that all agents (except the multi-step DQN agents with $n > 3$) performed similarly, obtaining around -150 test performance and were all able to move the pendulum to the top.

CartPole

Figure 2 shows the training curves for all agents for `CartPole-v0`. We set $\epsilon = 0.2$ for training and do not decay it in this environment. The maximum number of timesteps and hence max reward is 500. Notably, the training is very unstable and was found to be highly sensitive to initial conditions. Therefore, we picked the agents that show the best performance across multiple training seeds for each algorithm. Some trajectories keep oscillating while others reach the peak performance then start dropping. This is reported to be a common phenomenon in `CartPole-v0` [4]. However, all models obtain a mean test performance of 500 over 100 test runs as shown in Table 1.
We also looked at the test-time agent GIFs to study agent behaviours. We saw that different agent converged to different strategies to keep the pole balanced. The base DQN agent makes small, alternating left-right steps to balance the pole. The DQN + PER agent kept the pole nearly stationary throughout, with gradual movement of the cart in one direction. On the other hand, DQN + RND agent shows 'sliding' movements in some episodes, where the cart is slid in one direction (tilting the pole), and then it is balanced by sliding the cart in the other direction.

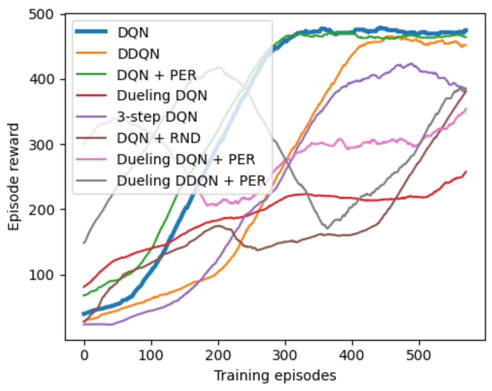


Figure 2: Training curves for all agents in `CartPole-v0`

Hockey

Figure 3 shows the training curves for all agents on `hockey` and Table 1 column Hockey shows the on wins and draws test performance against the weak agent.

		Test Performance			
	Agent	**Hockey**		**Pendulum**	**Cartpole**
		Wins	Draws		
1	DQN	0.39	0.49	-166	500
2	Double DQN	0.39	0.29	-159	500
3	DQN + PER	0.44	0.27	-152	500
4	Dueling DQN	0.42	0.36	-158	500
5	**Dueling DQN + PER**	**0.83**	**0.10**	**-147**	**500**
6	Dueling DQN + PER (14 actions)	0.43	0.23		
7	Dueling DQN + PER (20 actions)	0.74	0.15		
8	Dueling DQN + PER: strong	0.23	0.24		
9	Dueling DQN + PER: self play	0.37	0.30		
10	Dueling DQN + PER: curriculum(defense)	0.33	0.20		
11	Dueling Double DQN + PER	0.53	0.22	-150	500
12	Multi-step DQN (n = 3)	0.52	0.29	-163	500
13	Multi-step DQN (n = 5)	-	-	-182	-
14	Multi-step DQN (n = 10)	-	-	-939	-
15	Multi-step DQN (n = 100)	-	-	-2109	-
16	Multi-step DQN (MC)	-	-	-2253	-
17	DQN + RND	0.35	0.53	-148	500
18	DQN + RND: self play	0.03	0.64		
19	DQN + RND: curriculum(shooting)	0.18	0.41		
20	DQN + RND: curriculum(defense)	0.11	0.42		
21	DQN + RND: curriculum(both)	0.15	0.27		

Table 1: Test performance of different RL agents across environments.

From Figure 3 we see that most agents converged on performance in about 20000 episodes, except the Dueling DQN + PER agent. In Table 1, we see the most variance in test performance for the different agents in `hockey`, presumably since it is a more complex environment where it is tougher to learn a

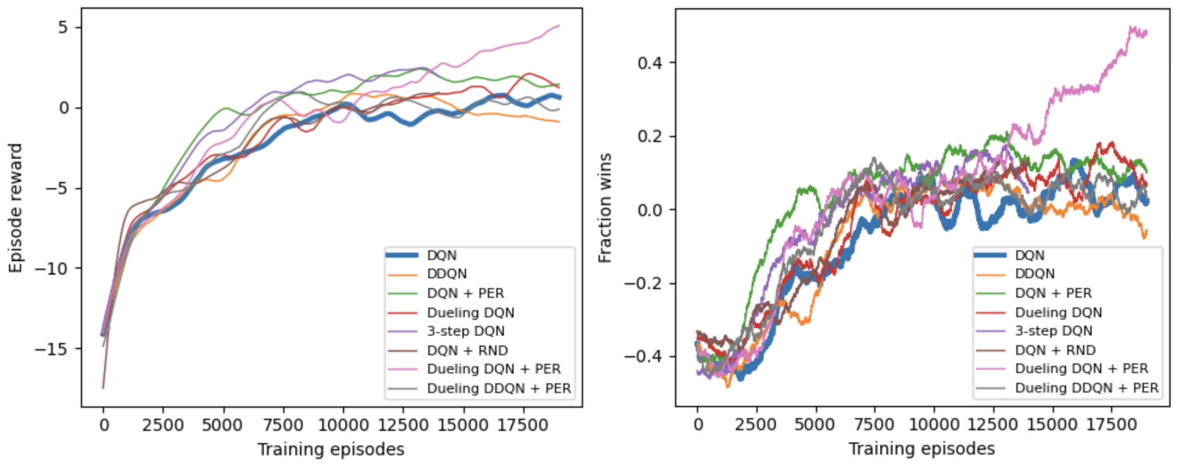


Figure 3: Training curves for all agents in `hockey`. Left: episode reward, Middle: Fraction of wins, Right: episode reward for DQN + RND experiments

good policy. Dueling DQN + PER shows the highest test performance compared to all agents with a winning 83% times and losing only 7% times against the weak agent. The DQN + RND agent(s) had a higher number of draws (53%) compared to all agents.

We analyzed the test GIFs for Dueling DQN + PER and DQN + RND. Dueling DQN + PER is highly offensive and tries to shoot at the goal several times directly into the goal in each episode. Sometimes it shoots right into the opponent who blocks the goal, but many times, it successfully steers away from the line of sight of the opponent and squeezes the puck into the goal. However, it is much weaker at defense, often unable to run to save the puck from reaching towards its own goal. The DQN + RND agent, on the other hand, is good at defense and draws a lot. It typically aims to shoot by bouncing off of the walls. This is not always a good strategy as it is harder to achieve, and the weak agent can often defend shots.

We also experimented with different action space discretizations for Dueling DQN + PER. We tested a 14 action version (including double steps of +2, -2 for each of the 6 actions) and a 20 action version (including combination of actions for example +1 in x and -1 in y etc.). We find that performance drops in both cases (Table 1). Perhaps a more discretized action space is harder to learn and would require a higher number of episodes to reach the same level of performance as with only 8 actions.

In addition to these algorithms, we also experimented with different opponent training, curriculum training for Dueling DQN + PER and DQN + RND.

For opponent-training, after training the agent on 15,000 episodes with the weak opponent, we trained the resultant agent to play either with the strong agent or with itself as an opponent (self-play) for another 10000 episodes. In case of Dueling DQN + PER, both training against the strong agent or against self decreased subsequent test performance against the weak agent significantly, more so in case of the strong agent (Table 1 rows). In case of DQN + RND, the self-play agent learns to draw with itself even by the end of training. However, on testing on the weak opponent, the self-play agent only wins only 3% of the test trials but draws even more often (64% times).

In curriculum training, we trained the Dueling DQN + PER agent on 15,000 episodes against the weak opponent following by an additional 10,000 episodes in defense mode. For the DQN + RND agent, we tested the other way around, *i.e.* we trained 5000 episodes of either defense mode, shooting mode or both followed by 10000 episodes of normal mode against the weak agent. We hypothesised that training in steps towards the final task might be helpful for the agent to learn the whole task better. For the Dueling DQN + PER defense mode training, the test GIFs show that the agent is not able to improve its defense by training for 10,000 extra episodes, and also forgets some of its offense capabilities, therefore resulting in lower test performance against the weak agent. For DQN + RND, the agents trained in defense mode are able to learn to defend the goal but are not able to improve much further in the normal mode. In shooting mode, the performance is quite low for the initial shooting training episodes and picks up in normal mode. The test GIFs show the DQN + RND agents trained by this curriculum method are neither able to shoot or defend well (sometimes not even going after the puck) resulting in lower test performance against the weak agent. Overall, self-play and curriculum training did not improve overall wins percentages for any agent across all experiments.

4 Discussion

In simple environments such as `Pendulum-v1` or `CartPole-v0`, our agents are all able to obtain good performance, but may converge on slightly different solutions. For example, in `CartPole-v0` we observed slight alternating movements to stay in place and prevent the pole from falling, or sliding movements (back and forth) that also prevent the pole from falling or small oscillations that cause the cart to move in one direction gradually but keep the pole in place. For `Pendulum-v1`, the solutions were more similar to each other: balancing the pendulum at $0^\circ$with very little torque applied or slightly off $0^\circ$with some torque applied to keep it in place.

In `CartPole-v0`, we also noticed a phenomenon where agents reach max performance during training before performance drops again, as if the agent is forgetting the learnt policy. This is something others have also found while training DQN-based algorithms on `CartPole-v0` [4]. This might be related to the phenomenon of 'catastrophic forgetting' observed in deep learning, especially deep RL [3]. This might happen when only successful states and actions are visited after a point, the neural net might forget the values it has learnt for other states and actions that it is no longer being trained on [2]. A solution proposed by Bruin et al. is to retain a small percentage of experience collected early in training, so that the network doesn't forget the values for this. We didn't implement this, but is a pointer for future experiments. Another work-around is to do early-stopping, where we stop training when the agent reaches the highest performance.

In more complex environments such as `hockey`, the performance of different agents varies to a greater degree. the Dueling DQN + PER reaches the solution that gets the most number of wins against the weak opponent. All other agents reach similar performance at the end of training (0.35 fraction of wins). The GIFs provide some clue into why this might be the case. Dueling DQN + PER has learnt a way to shoot past the weak agent – and goes for multiple shoots in every episode. DQN + RND for instance is able to defend well, and tries to shoot for example by bouncing off the walls, but is not as offensive as the Dueling DQN + PER agent, hence it has more draws than wins. We tried training other agents (DQN, DDQN, Dueling DDQN + PER, DQN + RND agent for many more episodes, but it still asymptotes at the same level of performance – perhaps indicating that it has converged on a local optimum. From the training curves, it is clear that Dueling DQN + PER has moved past this asymptote.

The lack of success from experiments with opponent and curriculum training suggest that the (1) the strategies used by the weak agent, strong agent, or self are very different to try to generalise performance across each other, and (2) the strategies learnt in defense or shooting mode where the initial position of the puck is different might actually be ill-suited for normal mode where the episode doesn't always start with the puck being with the agent or the opponent.

For further gains in the `hockey` environment, we may need more sophisticated techniques like model-based planning to efficient plan goals and theory of mind to account for the position, strategies and abilities of the opponent.

References

[1] Y. Burda, H. Edwards, A. Storkey, and O. Klimov. Exploration by random network distillation, 2018.

[2] T. de Bruin, J. Kober, and K. Tuyls. The importance of experience replay database composition in deep reinforcement learning. 2015.

[3] I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks, 2015.

[4] A. Green. Dqn debugging using open ai gym cartpole. https://adgefficiency.com/dqn-debugging/, 2020.

[5] D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[6] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[7] J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[8] Y. Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

[9] 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.

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb. 2015.

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

[12] K. Shao, Z. Tang, Y. Zhu, N. Li, and D. Zhao. A survey of deep reinforcement learning in video games. *arXiv preprint arXiv:1912.10944*, 2019.

[13] C. Tang, B. Abbatematteo, J. Hu, R. Chandra, R. Martín-Martín, and P. Stone. Deep reinforcement learning for robotics: A survey of real-world successes. *Annual Review of Control, Robotics, and Autonomous Systems*, 8, 2024.

[14] 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.

[15] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.