Two Deep Q-Networks and an Actor-Critic Implementation for Learning Ice Hockey

Nicholas Aksamit [*1]

Abstract

Three reinforcement learning algorithms are applied to the Atari game Ice Hockey: two Deep Q-Networks, and the Advantage Actor-Critic. Performances express that Ice Hockey is an incredibly difficult game to learn, with learned policies being largely insufficient in solving the problem at hand. However, the on-policy advantage actor-critic algorithm was able to achieve a score much higher than the other algorithms which are off-policy.

1. Introduction

In machine learning, there are three main paradigms which algorithms follow: supervised, unsupervised, and reinforcement (Sutton & Barto, 2018). The lattermost model makes use of a reward signal, and attempts to select actions in some environment that maximizes the accumulated rewards. A function is defined which expresses a definitive way to maximize these accumulated rewards:

$$V^{\pi}(x) = \mathbb{E}[\sum_{t=1}^{T} R_t | S_1 = x] \tag{1}$$

where $\mathbb{E}$ is the expectation, R_t is the reward at time t, S_1 is the initial state of the agent, and $t = 1$ to T consist of time steps where an action is taken in a state according to policy π. A policy determines which action to take in a certain state, and $V^{\pi}(x)$ determines the value of state x. An additional function decouples states and actions, while still returning a value:

$$Q^{\pi}(x, a) = \mathbb{E}[R_1 + \gamma V^{\pi}(x') | S_1 = x, A_1 = a] \tag{2}$$

As can be interpreted, the V function gives the average between all possible actions that can be taken, while the Q function simply determines the value of taking an action in some state. As such, by calculating this Q function, one can determine the best actions to take by taking the maximum value function of all possible actions, or $\max_a Q(x, a)$. Similarly, this is what is referred to as the greedy policy, for it chooses the action that maximizes the value function.

Reinforcement learning has been applied to a wide variety of video game genres, with a diverse degree of success (Shao et al., 2019). Some of these games include: Minecraft, StarCraft, Dota2, and Quake. In this work, three algorithms are applied to the Atari 2600 game Ice Hockey. These algorithms are the Deep Q-Network (DQN), Double DQN (DDQN) with Prioritized Experience Replay (PER), and Advantage Actor-Critic (A2C).

In Section 2, each algorithm is described in detail, followed by Section 3, which includes information on the Ice Hockey environment. Afterwards, Section 4 consists of information for the experimental setup of this work, accompanied by Section 5 where results of the experiments are discussed. Concluding is Section 6, expressing the generalities found during experimentation.

2. Methods

In total, three methods were implemented for attempting to learn ice hockey for the Atari 2600, namely a DQN, DDQN with PER, and A2C algorithm. Each will be discussed in their respective subsection, starting with DQN (Section 2.1), followed by DDQN (Section 2.2), and ending with A2C (Section 2.3).

2.1. Deep Q-Network

The DQN utilizes a neural network for approximating the Q-value. To improve performance over the usage of a standard neural network, both experience replay and target networks are used (Mnih et al., 2015). The former allows the network to train on previously experienced events, while the latter makes training more stable. In addition, an ϵ-greedy search policy is utilized throughout the training. Algorithm 1 illustrates the DQN algorithm with both a target network and

[1]Department of Computer Science, Brock University, St. Catharines, Canada. Correspondence to: Yifeng Li <yli2@brocku.ca>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

experience replay included.

Algorithm 1 DQN

Input
- α Learning Rate
- B Number of Batches
- U Updates per Batch
- N Batch Size
- F Target Network Update Frequency

Initialize online network θ
Initialize target network $\theta_t = \theta$
for m=1:MAX_EPOCHS **do**
  make h experiences e based on θ, and store in E
  for b=1:B **do**
    sample batch of experiences from E
    for u=1:U **do**
      for n=1:N **do**
        $y_n = r_n + \delta_{x'_n} \gamma \max_{a'} Q(x'_n, a', \theta_t)$
      end for
      $L(\theta) = \frac{1}{N} \Sigma_{n=1}^{N} (y_n - Q(x_n, a_n, \theta))^2$
      $\theta = \theta - \alpha \nabla_\theta L(\theta)$
    end for
  end for
  if m mod F == 0 **then**
    $\theta_t = \theta$
  end if
end for

2.1.1. Experience Replay

Let each experience be represented as $e_t = (s_t, a_t, r_t, s_{t+1})$, with E being the set of all experiences. After every action that is taken in the environment, add the new experience e into the set E. E has a certain predefined capacity, where upon reaching its maximal size, the oldest experiences are continually replaced with newer ones. In this way, the experience replay is similar to a queue data structure.

A reason for utilizing this performance enhancement is due to the network being able to forget about previously occurred events. With the inclusion of experience replay, the network is able to learn from events of the distant past. This helps the approximator both remember good actions that were taken in certain situations, as well as learn from poor action selection.

2.1.2. Target Q-Network

Two DQNs are initialized, namely one that is online (θ) and the target (θ_t) network. While training, θ_t is utilized as a predictor of the target Q-value. This is used for the loss and gradient calculations with respect to the online network θ. Every n iterations, the weights θ_t are set to equal that of θ, where n is a predefined variable.

This improvement does not allow one DQN to train on itself. If this was to occur, then when predicting the value of $Q(s, a)$, the same approximator that calculates $Q(s', a')$ would be used. This can lead to instabilities, such as improving $Q(s, a, \theta)$ altering the outcome of $Q(s', a', \theta)$. In an idiomatic sense, this would equate to a moving target. Thus, when a target network is used, it calculates $Q(s', a', \theta_t)$ and remains consistent throughout n iterations, reducing the instability of the system by not allowing for a moving target.

2.1.3. ϵ-Greedy Policy

For an ϵ-greedy policy strategy, the value of ϵ determines the probability of whether a greedy action is taken, or random. Algorithm 2 demonstrates the operation of this policy.

Algorithm 2 ϵ-Greedy Policy

$val \leftarrow rand(0, 1)$ ▷ Random decimal value
if $val > \epsilon$ **then**
  return $\text{argmax}_a(Q(s, a, \theta))$
else
  return random action
end if

2.2. Double Deep Q-Network with PER

The DDQN by itself includes a mechanism for increasing the stability of target calculations (Van Hasselt et al., 2016). It does this with the use of the target network θ_t, which is described in Section 2.1.2. As was shown in Algorithm 1, the target value is calculated as Equation 3. However in the DDQN, this function is now calculated as is shown in Equation 4. Comparisons between these two equations display a difference in calculating $Q(x', a')$. Instead of using only the target network estimation as the Q-value, an action value is first selected from the approximation of the online network.

By including this performance enhancement, it reduces a level of overestimation that the DQN is prone to. As was stated in Section 2.1.2, the target network reduces the effect of a moving target. Another improvement is illustrated by Equations 3 and 4, whereby the target Q-value includes estimations from both the online and target network, rather than simply the target. Another improvement included within this algorithm combination is PER, which is discussed in Section 2.2.1. The pseudocode for DDQN is not provided as it would appear similar to Algorithm 1. Main differences include the new y_n calculation, and sampling from PER instead of a standard experience replay memory.

$$y_n = r_n + \delta_{x'_n} \gamma \max_{a'} Q(x'_n, a', \theta_t) \quad (3)$$

$$y_n = r_n + \delta_{x'_n} \gamma \max_{a'} Q(x'_n, \max_{a'} Q(x'_n, a', \theta), \theta_t) \quad (4)$$

2.2.1. Prioritized Experience Replay

Usage of PER provides an advancement over the standard experience replay, which is described in Section 2.1.1. The intuition behind this improvement is to attach a priority on each individual experience value, so as to more efficiently use those experiences in the limited capacity E. In PER, those experiences that have a higher priority will have a greater chance of being selected for sampling. To obtain the priority value, the temporal difference error is used as a basis (Schaul et al., 2015). Both Equations 5 and 6 display the temporal difference error, and priority equation respectively. The α value represents how much prioritization is used, and ϵ is a low value that ensures the priority is never reduced to a value of 0.

$$\delta_n = y_n - Q(x_n, a, \theta) \tag{5}$$

$$P(x_n) = \frac{(|\delta_n| + \epsilon)^\alpha}{\Sigma_{n'}(|\delta_{n'}| + \epsilon)^\alpha} \tag{6}$$

When utilizing PER, a sum tree implementation is incorporated to maintain experience priorities. The leaf nodes in the sum tree are priority values, whereas the intermediate parent nodes are the sum of all child nodes. This allows easy maintenance of cumulative priority calculations, which can be seen in the denominator of Equation 6.

2.3. Advantage Actor-Critic

A2C is an algorithm that behaves differently than the previously explained DQN and DDQN. As the name suggests, it consists of an actor and a critic. The critic evaluates state and action pairs with the use of the advantage function (Equation 7). This equation measures the extent to which an action is better than the average policy value at the same state. On the other hand, the actor decides which action is made. When the process is combined, critic critiques the decision that the actor arrives at. More specifically with regards to actor and critic, the former uses the policy distribution, while the latter uses the value distribution. Estimation of the advantage value can be done using n-step returns and Generalized Advantage Estimation (GAE). For the purposes of this work, the latter method of advantage estimation is used.

$$A_t^\pi(x_t, a_t) = Q^\pi(x_t, a_t) - V^\pi(x_t) \tag{7}$$

$$A_{\text{GAE}}^\pi(x_t, a_t) = \Sigma_{l=0}^\infty (\gamma\lambda)^l \delta_{t+l} \tag{8}$$

In n-step return, a variable n balances a tradeoff between the amount of bias and variance in estimation. However with GAE, the intuition is to mix different values of n, so as to reduce both variance and bias. Equation 8 expresses how the advantage value is estimated using GAE. Notice that δ is the temporal difference error. In addition, λ is a hyperparameter which also controls the bias-variance trade-off, but in a more implicit way than n. Algorithm 3 illustrates the formulation of A2C steps for completion.

Algorithm 3 A2C

Input
- α_A Actor Learning Rate
- α_C Critic Learning Rate
- β Entropy Regularization Weight

Initialize actor network θ_A
Initialize critic network θ_C
for episode=1:MAX_EPISODES **do**
  gather and store experiences for $t = 1, 2, ..., T$ by acting in environment using current policy
  for t=1:T **do**
    calculate $V(t) = V(x_t, \theta_C)$
  end for
  for t=1:T **do**
    calculate $A_{GAE}^\pi(x_t, a_t)$
    calculate target y_t
    calculate entropy of policy distribution from θ_A
  end for
  $L_{val}(\theta_C) = \frac{1}{T}\Sigma_{t=1}^T(V(x_t, \theta_C) - y_t)^2$
  $L_{pol}(\theta_A) = \frac{1}{T}\Sigma_{t=1}^T(-A_{\text{GAE}}^\pi(x_t, a_t) log\pi(a_t|x_t) - \beta H_t)$
  $\theta_C = \theta_C - \alpha_C \nabla_{\theta_C} L_{val}(\theta_C)$
  $\theta_A = \theta_A - \alpha_A \nabla_{\theta_A} L_{pol}(\theta_A)$
end for

3. Environment

The environment used for this work comes from OpenAI Gym (Brockman et al., 2016), which is an open-source software package for the Python programming language. Its purpose is to provide a standard for comparing performance between reinforcement learning algorithms. For this work, all methods are trained to play an Atari 2600 game named Ice Hockey.

In this game, the agent is expressed as one of the two players on the upper-half of the screen. An example of a random policy is illustrated as Figure 1. As the hockey puck arrives closer to the goal, the player is switched from forward to goalie, in an attempt to save from the opponent scoring. The main goal of Ice Hockey is to score as many points as possible within a 3 minute time limit. This time limit is what defines an episode. For training in this environment, a reward signal of 1 is given for scoring on the opponent, -1 for being scored on, and 0 otherwise.

Ice Hockey has the ability to provide RAM or RGB values as input to a neural network. For this work, the RAM

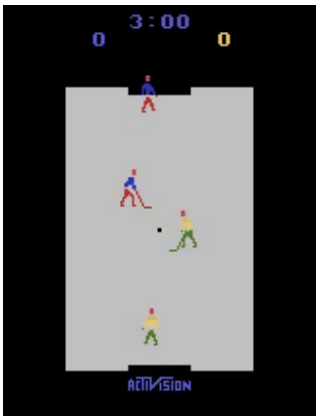

Figure 1. Random Policy on Ice Hockey

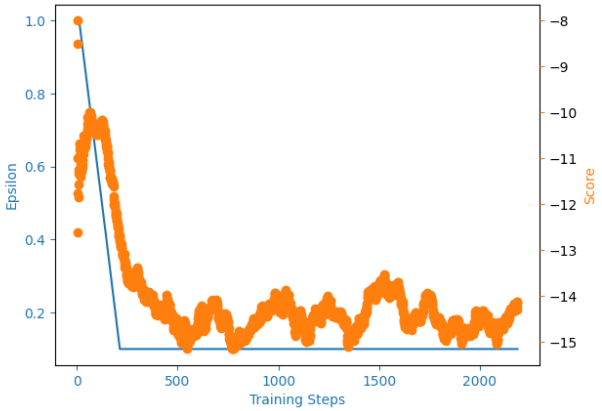

Figure 2. DQN Score Graph

method was used, which provides 128 numerical integers. These values contain information such as the position of the players, the state of the players, position and velocity of the hockey puck, and more. Ice Hockey is an already complex game to learn, and any additional complexities that come from RGB values may inhibit the learning process. It is for this reason that RAM was decided for use rather than RGB.

There are a total of 18 actions that may be taken in this environment. The agent can move in any of 8 directions, and also while slashing its hockey stick. This corresponds to 16 actions. The additional two actions are no operation, where the agent simply does not move, and no movement but slashing the hockey stick.

4. Experimental Setup

All the methods used in this work are explained in detail within Section 2. In this section, the parameter configurations used for each of the methods will be listed, along with the environment that all algorithms were used for. This encompasses the information needed to approximately replicate the results found in Section 5. Each subsection is respective to the order of methods, namely the DQN (Section 4.1), DDQN with PER (Section 4.2), and lastly, A2C (Section 4.3).

4.1. Deep Q-Network

Each of the important parameters for execution are displayed in Table 1. The notable parameter being the number of maximum frames used for training the online network. A total of 10 million frames were used, with training starting on frames after 50000. The DQN was employed specifically because of the many papers praising its performance on various Atari 2600 games (Mnih et al., 2015; 2013). In addition to this, it has a relatively simple implementation. The Python language was utilized, along with packages such as OpenAI Gym, and PyTorch. The latter allows for tensor calculations to be completed on the graphics processing unit, greatly decreasing training time.

4.2. Double Deep Q-Network with PER

Parameters for the DDQN with PER are listed in Table 2. Similarly to DQN, 10 million frames are run, and training begins after 50000 frames. The DDQN and PER are both improvements upon the regular DQN, and so it naturally arose as a method for experimenting with learning Ice Hockey. Similar parameters are used between the aforementioned algorithms, so as to not give preference to one over the other with hyperparameter selection. Similarly to DQN, DDQN with PER was implemented in the Python language using PyTorch. OpenAI Gym is used for simulating the environment.

4.3. Advantage Actor-Critic

All of the hyperparameters used for A2C are illustrated in Table 3. Differently than both DQN and DDQN, training is done using epochs rather than a set number of maximum frames. In addition, Tianshou (Weng et al., 2021) was utilized. Tianshou is a reinforcement learning platform for Python that already has major algorithms implemented, and ready for testing on various environments. However, OpenAI Gym still simulates the Ice Hockey environment.

A2C is selected as a method due to its contrast from DQN and DDQN. This contrast may shed light on possible improvements when training the models to play Ice Hockey.

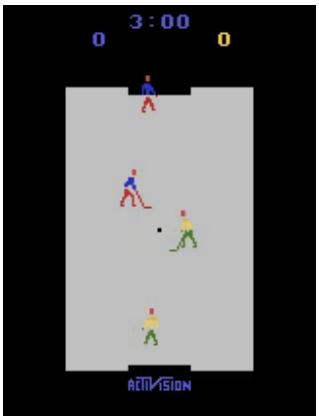

Figure 3. DQN Policy on Ice Hockey

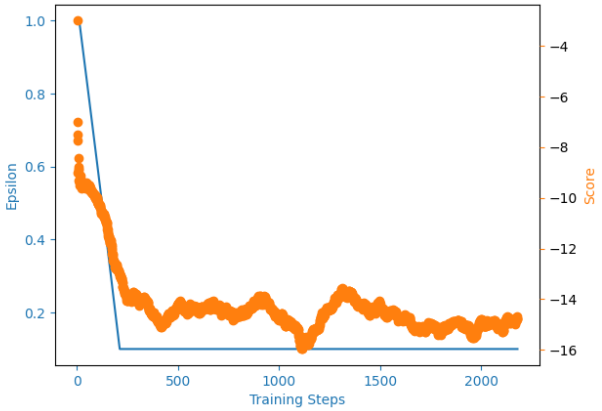

Figure 4. DDQN with PER Score Graph

5. Results

5.1. Deep Q-Network

Figure 2 illustrates the score of the DQN during the training process of 10 million frames of gameplay. Its x-axis displays the training steps in terms of episodes, of which there were over 2000. As can be seen, usage of the DQN gives an abysmal score after training. It seems as if the network does not learn anything, even after such a large amount of game time. After the epsilon value is reduced to 0.1, the score remains at a low of between -13 and -15 per episode.

Figure 3 provides the link to watch the DQN policy. As can be seen, the agent learns to only and always move upwards. Such a move is not optimal as there is no potential for shooting the puck to score a goal, and there is little chance for blocking a shot from the opponent. Due to this, many goals are scored on the agent, and it receives a large negative overall score per episode.

From results displayed on the SLM Lab leaderboards (slm), DQN regularly achieves the worst score in comparison to methods such as DDQN with PER, and A2C. Thus, it seems that the game of Ice Hockey is intrinsically too complex for the DQN to learn. Perhaps this could be due to the off-policy nature of this algorithm, or from poor experience sampling. Conflicts also might arise not from the particularities of this algorithm, but the algorithm itself. DQN may simply be not powerful enough to learn this game, even with added enhancements such as a target network and experience replay. With reference to all the algorithms used from the SLM Lab leaderboards, this seems to be a possible explanation.

5.2. Double Deep Q-Network with PER

The DDQN with PER was trained over a span of 10 million frames, where the performance is illustrated in Figure 4. The overall score, similar to DQN (Figure 2), is very poor. The DDQN does not seem to learn the environment space,

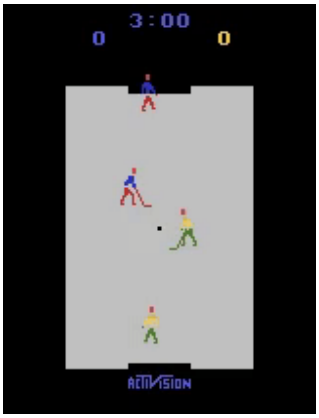

Figure 5. DDQN with PER Policy on Ice Hockey

as scores constantly degenerate at the beginning and remain at a relatively constant level throughout. At the end, training would achieve an average score of between -16 and -14 in the game of Ice Hockey.

When interpreting the results of DQN, it was said that the weak performance may have been due to the lackluster enhancements. However, the DDQN includes better estimations of the action-value function, and also a much-improved experience replay set. Even with these improvements, scores are not improved and remain relatively the same as the regular DQN. This leads to the belief that algorithm itself may not be powerful enough to encapsulate the difficulties of Ice Hockey. The poor score for DDQN follows from what is seen in the SLM Lab leaderboards (slm), where it is also very similar to that recieved from DQN.

Figure 5 expresses the policy of the DDQN with PER. It becomes quickly evident why the poor performance occurs. The DDQN has learned a very similar policy to the DQN. However, instead of moving upwards, the agent now con-

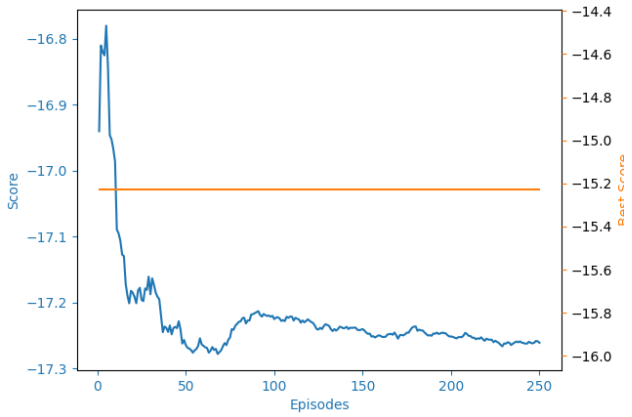

Figure 6. A2C Off-Policy Score Graph

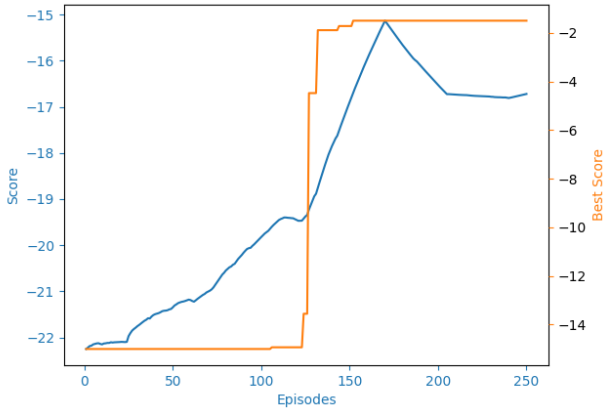

Figure 7. A2C On-Policy Score Graph

stantly moves downwards and to the left. When comparing to the policy of DQN (Figure 3), this is much worse, since the goalie will move out of the goal and allow for an easy goal by the opponent. This clearly is not the optimal policy of the game, since the overall reward is to be maximized, and a very low reward is received.

5.3. Advantage Actor-Critic

A2C can be expressed as both an on-policy and off-policy reinforcement learning algorithm. For the purposes of this work, both methods were executed to compare if a difference in policy for training leads to a difference in outcome. This could give rise to conclusions about the performance of DQN and DDQN with PER, both of which are off-policy algorithms. It should be noted that a demonstration could not be provided due to an issue with loading a policy from one that is saved (kiwi-sherbet, 2021). However, the graphs will give an accurate representation of the training by visualization of scores per epoch.

For the experimentation process, a set number of 250 epochs were utilized. In each epoch, 50,000 steps inside of the environment is taken using the current policy, with training occurring throughout these steps. Afterwards, the score is gathered as an average of scores in multiple testing environments. The best score is as named, the best score found throughout this policy evaluation.

To start, off-policy training is done. Figure 6 illustrates the scores received from this process. It becomes clear that the score received near the end of the epoch limit is very similar to those from both DQN and DDQN with PER. Thus, it does not seem like the off-policy A2C can learn the environment well. The best score found does not change after the initial epoch, remaining near -15.2. Although no demo is provided, it is likely that the agent determined a policy similar to that seen from the demo of DQN (Figure 3) and DDQN (Figure 5), where only one action is decided upon.

However, the on-policy A2C reached a different conclusion than that of its off-policy counterpart, and even DQN and DDQN. As can be seen in Figure 7, scores start at -22, and increase to levels similar to that of the off-policy A2C. Despite that, the best score increases to a high of above -2. Thus, there was an episode of testing where scores were much higher than anything that is seen in any other algorithm mentioned in this work. This leads to an interesting interpretation, where on-policy algorithms may work better for learning Ice Hockey than off-policy. This best score follows a similar result found on the SLM Lab leaderboards (slm).

6. Conclusion

Three reinforcement learning algorithms are applied for learning an Atari 2600 game named Ice Hockey. The Python libraries Tianshou (Weng et al., 2021) and OpenAI Gym (Brockman et al., 2016) were utilized. Both DQN and DDQN with PER were implemented from the ground up, while A2C was executed with help of Tianshou.

Overall, the performances from all three algorithms expressed the notion that Ice Hockey is a difficult game to learn from an algorithmic perspective. None were able to achieve a score that was neutral, and would commonly find a policy which does not solve the game. DQN and DDQN with PER, along with the off-policy A2C displayed similar performance. However, the on-policy A2C exemplifies that perhaps on-policy methods should be utilized for learning Ice Hockey. It was able to find an overall best score of -2, which is much greater than the scores found in all other methods. Future work should consist of exploit-

ing on-policy methods to determine their performance in learning Ice Hockey. An on-policy algorithm that should be experimented with in the future is the Proximal Policy Optimization algorithm (Schulman et al., 2017).

Acronyms

DQN Deep Q-Network

DDQN Double DQN

PER Prioritized Experience Replay

A2C Advantage Actor-Critic

GAE Generalized Advantage Estimation

References

Atari environment benchmark. URL https://slm-lab.gitbook.io/slm-lab/benchmark-results/atari-benchmark.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

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

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

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

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

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

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

kiwi-sherbet. Issues on loading a policy, 2021. URL https://github.com/thu-ml/tianshou/issues/443.

Hyperparameter	Value
minibatch size	32
replay memory size	1,000,000
target network update frequency	10,000
discount factor	0.99
learning rate	0.005
initial ϵ	1.0
final ϵ	0.1
final exploration frame (for ϵ)	1,000,000
replay start size	50,000
total training frames	10,000,000
loss function	MSE
optimizer	Adam
network model	128-128-18

Table 1. DQN Hyperparameter Setup

Hyperparameter	Value
minibatch size	32
replay memory size	1,000,000
target network update frequency	10,000
discount factor	0.99
learning rate	0.005
initial ϵ	1.0
final ϵ	0.1
final exploration frame (for ϵ)	1,000,000
replay start size	50,000
total training frames	10,000,000
loss function	MSE
optimizer	Adam
α	0.6
β	0.4
ϵ (PER)	0.001
network model	128-128-18

Table 2. DDQN Hyperparameter Setup

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

Weng, J., Chen, H., Yan, D., You, K., Duburcq, A., Zhang, M., Su, H., and Zhu, J. Tianshou: A highly modularized deep reinforcement learning library. *arXiv preprint arXiv:2107.14171*, 2021.

Hyperparameter	Value
minibatch size	64
replay memory size	20,000
discount factor	0.9
learning rate	0.001
epochs	250
environment steps per epoch	50000
λ	1
β	0.1
value loss weight	0.5
network model	128-128-128-18

Table 3. A2C Hyperparameter Setup