Value-Based Reinforcement Learning using DQN

Rithviik Srinivasan, [1] Loc Anh Tran, [1] Aswin Prasanna Suriya Prakash[1]

[1] Department of Electrical and Computer Engineering, New York University
{rs8385, as17340,lat9357}@nyu.edu

Abstract

Reinforcement learning has shown remarkable success in training agents to perform complex tasks such as playing video games. However, learning with continuous state spaces is a difficult task, and existing approaches such as deep Q-learning can sometimes lead to overestimation of the value of actions. In this project, we aim to implement Deep Q-Network (DQN)(Mnih et al. 2013), Double Deep Q-Network (Double DQN) (Hasselt 2010; van Hasselt, Guez, and Silver 2015) and Dueling Deep Q-Network (Dueling DQN) (Wang et al. 2016) and compare the performance between each. We also partly investigate the effects of hyper parameter optimization with Dueling DQN. All three architectures are implemented to play the Viz Doom game (Kempka et al. 2016). Dueling DQN with optimized hyper parameters performed the best of the three algorithms.

The project code base (GitHub repository) and video demo of final model can be found **here**.

Introduction

This project involved implementing three different deep q network techniques: DQN (Mnih et al. 2013), Double DQN (Hasselt 2010) and Dueling DQN (Wang et al. 2016) on the Viz Doom environment (Kempka et al. 2016), which provides various game scenarios that range from simple to complex, with different types of enemies, obstacles, and rewards. This involved training an agent to successfully satisfy the requirements of the chosen scenario using appropriate Deep Q-learning techniques and compare the performance among them. The ”rocket-basic” scenario is being used for this project. The agent receives input observations in the form of RGB images (which are pre processed to gray scale) of the chosen scenario's environment, which represent the current state of the game. The core focus was to identify the best possible implementation configuration that maximises the agent's ability to replicate near human performance (finding and killing the single enemy) in the chosen scenario of the game . The target metric prioritized as an indicator of performance was the overall average score achieved by the agent. Achieving a high score is the same as reducing the time taken to complete the level or minimising the number of actions taken. Using a replay buffer that stores the agent's past experiences, a data set of experiences is generated by the agent prior to training. During the agent's interaction with the environment, each experience tuple (observation, action, reward, next observation, done flag) is generated and accumulated in the replay buffer. During the training process, the agent uses random samples of batches equivalent to the batch size which allows the agent to break any temporal correlations in the sequence of experiences.

Literature Review

Reinforcement learning (RL) is a subfield of machine learning that is concerned with how an agent can learn to make intelligent decisions through interactions with an environment (Sutton and Barto 2018). The agent learns by trial-and-error, taking actions in the environment and receiving feedback in the form of rewards or penalties. The goal of the agent is to learn a policy that maximizes the cumulative reward over time. Reinforcement learning has many practical applications, including robotics, gaming, finance, and healthcare. It has been used to perform complex task like teach robots to perform complex tasks such as grasping objects and walking (Levine et al. 2016). In RL, there are typically three components: the agent, the environment, and the reward signal. The agent is the decision-making entity that interacts with the environment. The environment is the world in which the agent operates, and it can be either deterministic or stochastic. The reward signal is a scalar value that the agent receives from the environment after each action.

There are several different approaches to RL, but value-based RL is one of the most popular. In value-based RL, the agent learns to estimate the value of each state or state-action pair and chooses the action that maximizes the expected value. The most common algorithm used in value-based RL is Q-learning, which iteratively updates the estimated Q-values using the Bellman equation (Bellman 1954):

$$\underbrace{\text{New } Q(s,a)}_{\substack{\text{New}\\\text{Q-Value}}} = \underbrace{Q(s,a)}_{\substack{\text{Current}\\\text{Q-Value}}} + \underset{\substack{\text{Learning}\\\text{rate}}}{\alpha} \Big[\underbrace{R(s,a)}_{\text{Reward}} + \underset{\substack{\text{Discount}\\\text{rate}}}{\gamma} \overbrace{\max Q'(s',a')}^{\substack{\text{Maximum predicted reward, given}\\\text{new state and all possible actions}}} - Q(s,a) \Big]$$

The VizDoom environment (Kempka et al. 2016) provides various game scenarios that range from simple to complex, with different types of enemies, obstacles, and rewards. VizDoom is a simulation environment that utilizes the popular first-person shooter game Doom. It permits the creation of bots that can play Doom using only the screen buffer. The environment features a 3D world that is more similar to real-life environments compare to the Atari console games environments used by Mnih in 2013 (Mnih et al. 2013) and Wang in 2016 (Wang et al. 2016). Additionally, it incorporates a fairly accurate physics model. In order for an agent (bot) to succeed in VizDoom, it must be able to efficiently perceive, understand, and learn from the 3D world to make informed decisions about its movements and actions. The agent usually receives observations in the form of RGB images, which represent the current state of the game.
In this project, we compare the performance of two different architectures of deep Q-networks (DQNs) for value-based reinforcement learning: DQN and Duel DQN.

Technical Details

Standard DQN

The DQN architecture uses a single neural network to approximate the Q-value function. It takes the current state of the environment as input and outputs the estimated Q-values for all possible actions in that state. The agent selects the action with the highest Q-value to maximize its rewards. The model is depicted in the following graph:

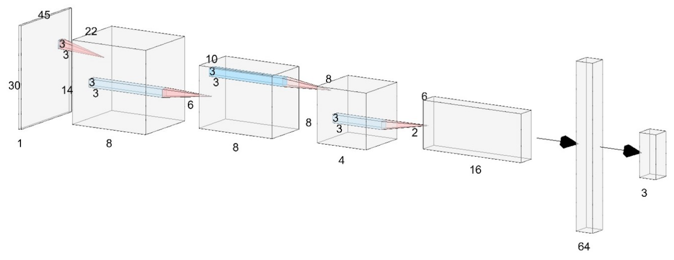


Figure 1: Single stream Q-network and Double Q-network

Double DQN

It has been shown that some DQNs perform substantial over estimations in certain conditions (with Atari 2600 console games (van Hasselt, Guez, and Silver 2015)). Thus, to reduce the risk of over-exploiting and under-exploring the environment, especially in complex environments with many states to explore, the Double DQN algorithm was proposed (Hasselt 2010; van Hasselt, Guez, and Silver 2015). Instead of using two copies of a single Q-network, the Double DQN uses two separate networks: an online network to select actions and an offline network to suggest the Q-values for the selected actions. This way, the Double DQN algorithm avoids over-exploitation and under-exploration issues of the DQN algorithm.

Duel DQN

The Duel DQN architecture (Wang et al. 2016) is an extension of the DQN architecture that separates the estimation of the state value and action advantage functions. It uses two parallel neural networks: one estimates the state value function, which measures the value of being in a particular state regardless of the action taken, and the other estimates the action advantage function, which measures the advantage of taking a particular action in a particular state over other possible actions. The Q-value function is then computed as the sum of the state value and action advantage functions. The model is depicted in the following graph:

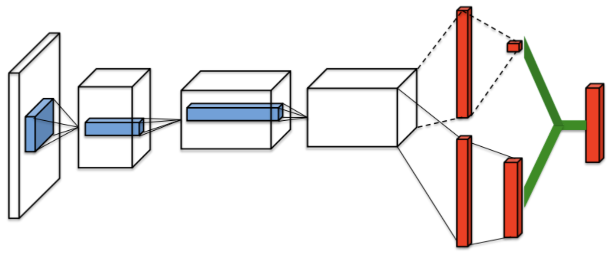


Figure 2: Dueling Q-network (Wang et al. 2016)

Hyperparameters Tuning

Hyperparameters are the parameters set for an algorithm before the learning process takes place. Such parameters have a direct effect on how well a model can learn from its environment. To reach maximum potential for learning in an environment, a model must have its hyperparameters optimized. In recent years, the complexity of deep learning methods is increasing alongside their popularity, thus there is a growing need for an efficient framework for automatic hyperparameter tuning. For this matter, the paper also investigates the effect of hyperparameter optimization on the Duel DQN Architecture.
Various software tools have been developed for hyperparameter optimization, including Google Vizier (Golovin et al. 2017), Ray Tune (Liaw et al. 2018), Hyperopt (Bergstra, Yamins, and Cox 2013),and Microsoft NNI (Microsoft 2021). In this paper, we choose the hyperparameter optimization framework Optuna (Akiba et al. 2019). The framework has many advantages over the others, including define-by-run programming that allows the user to dynamically construct the search space, efficient sampling algorithm and pruning algorithm that allows some user-customization. Furthermore, it's a versatile architecture that can be deployed for tasks of various types (Akiba et al. 2019).

Experiment

The aim of this project is to optimize a Deep Q-Network (DQN) on the ”Rocket Basic” (Olsson, Malm, and Witt

2022) scenario of VizDoom. The methodology followed involves implementing and comparing different DQN variants in order to obtain the best possible performance. The evaluation metric used is the average reward obtained by the agent over a certain number of episodes. The agent has been trained using each variant for 50 epochs each. The project uses mean square error (MSE) as the loss metric. MSE is chosen as the loss function in DQN due to its general application in finding an intuitive measure of Q-value approximation, a good amount of differentiability for efficient optimization, robustness to noisy data, statistical efficiency, and compatibility with existing frameworks. However, it can lead to overestimation of action values. To overcome this limitation, extensions like double DQN and dueling DQN is proven to have been effective. (Mnih et al. 2015)

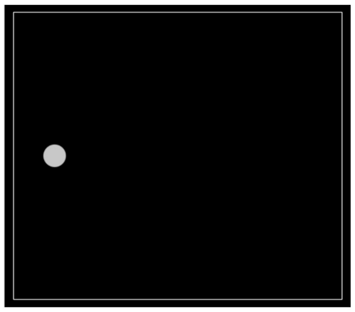


Figure 3: Basic Scenario Map (Olsson, Malm, and Witt 2022)

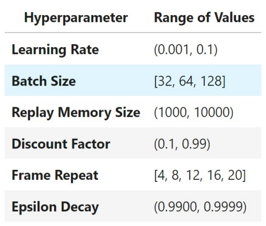
The project can be divided into the following phases:

1. Implementing a standard DQN and training and testing the model on the scenario with default parameters to get an initial performance metric.
2. Integrating double DQN with the standard DQN network and getting an initial performance metric.
3. Identifying the drawbacks in the two models by comparing the initial results
4. Implementing a Dueling DQN and get an initial performance metric.
5. Performing Bayesian hyperparameter tuning on the Dueling DQN model using Optuna for 93 trials (arbitrarily chosen).
6. Training and testing the Dueling DQN , Double DQN and standard DQN models using the best hyperparameters obtained from the optuna trials.
7. Summarizing the results obtained in the form of plots that show the improvement in performance among models.

Hyperparameter Optimization with Optuna

Optuna (Akiba et al. 2019) is an open source hyperparameter optimization framework to automate hyperparameter search. It uses a Bayesian Optimization algorithm to efficiently search for the best hyperparameters. The framework is proven to be better than some of the available hyperparamter optimization tools on Combined Algorithm Selection and Hyperparameter Optimization (CASH Optimization) Problem based on the results of a recent study (Shekhar, Bansode, and Salim 2022).

Optuna allows one to set an interval between certain values of hyperparameters and, in doing so, automatically find the best ones. Table 1 contains all hyperparameters used in the project. However, several more hyperparameters are available for each algorithm.



Hyperparameter	Range of Values
Learning Rate	(0.001, 0.1)
Batch Size	[32, 64, 128]
Replay Memory Size	(1000, 10000)
Discount Factor	(0.1, 0.99)
Frame Repeat	[4, 8, 12, 16, 20]
Epsilon Decay	(0.9900, 0.9999)

Table 1: Hyperparameter Search Space

Hyperparameters definition:

- **Learning Rate**: Determines how fast a model learns and accepts new information.
- **Batch Size**: The number of samples fed to a neural network before updating its parameters, impacting learning stability and effectiveness.
- **Replay Memory Size**: The number of past experiences stored in a reinforcement learning system's memory, affecting the learning process and performance.
- **Discount Factor**: A value in reinforcement learning that assigns importance to future rewards, impacting the learning algorithm's outcome.
- **Frame Repeat**: The number of times an action is repeated before a reinforcement learning agent receives a new observation, impacting the algorithm's training time and complexity.
- **Epsilon Decay**: A technique used in reinforcement learning to gradually decrease the exploration rate, epsilon, over time.

Results

The following plots show the train and inference mean reward score achieved by the agent against increasing number of epochs.

Deep Q-Network (DQN)

Figure 4 shows the plot of the performance during training using a standard DQN network. The y axis represents the mean reward score achieved by an agent against increasing epochs. As can be seen in the graph, the agent fails to consistently achieve a positive score and there is also a downward trend towards decreasing performance (negative result). The hyperparameters used here represent arbitrarily chosen parameters targeted at showing a baseline metric. Performance degradation is likely attributed to factors indicating the drawbacks of standard DQN. DQN networks generally work best on simple environments with low dimensional state space. VizDoom environments often have

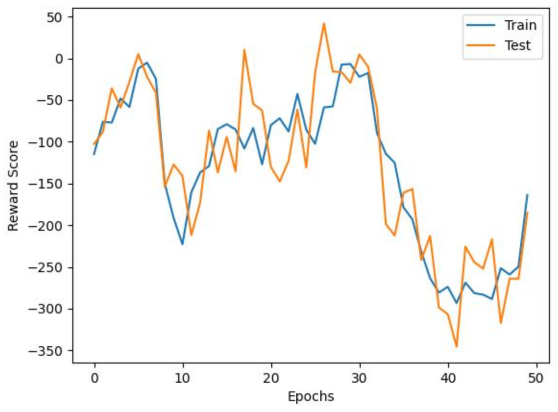


Figure 4

high-dimensional state spaces, which can make it challenging for a standard DQN network to effectively learn and generalize from the observations. The standard DQN network might struggle to capture the intricate details and correlations present in the state space, resulting in sub optimal performance. VizDoom scenarios can have non-stationary dynamics, where the optimal policy can change over time or based on the agent's actions. A standard DQN network, which assumes a stationary environment, might struggle to adapt to such changes, resulting in poor performance. DQN training is generally unstable and sensitive to hyperparameter choices. It may suffer from issues like overestimation or underestimation of action values, and it can take a considerable amount of tuning to achieve good performance. Convergence to an optimal policy is not always guaranteed.

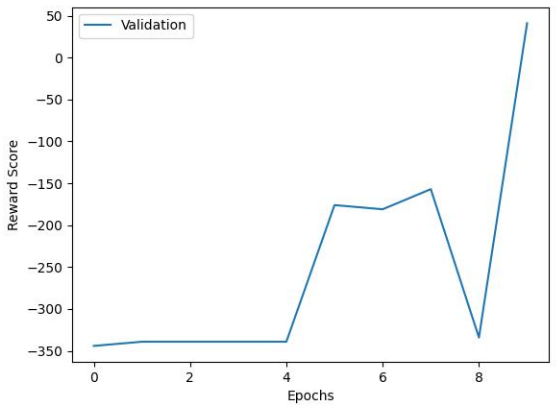


Figure 5

Figure 5 represents the testing performance of the trained agent over a few episodes. The agent does seem to perform better each episode, but the success to failure ratio (the number of episodes with positive score) over all episodes is considerably low.

Double Deep Q-Network (Double DQN)

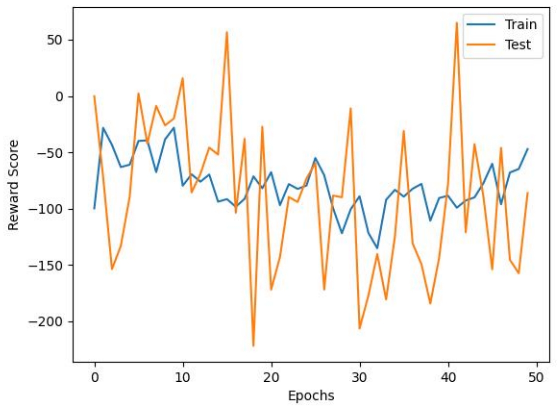


Figure 6

Figure 6 shows the plot of the performance during training using a double DQN variant. As can be seen in the graph, the agent again fails to consistently achieve a positive score even though there is no apparent downward trend towards decreasing performance (negative result). The hyperparameters used here again represent arbitrarily chosen parameters targeted at showing a baseline metric. The performance is definitely an improvement over standard DQN as there seems to be an improvement in the model's train variance. The model's does seem to be much more stable compared to standard DQN but still not considerately sufficient. This suggests some evidence confirming double DQN's intrinsic nature of mitigating result overestimation which is a common condition of standard DQN. The variability in test and train could be explained by improper epsilon decay (exploration-exploitation trade off factor) which is obvious given the DQNs are sensitive to hyperparameters, thus further enforcing the requirement for hyperparameter tuning. The graph does suggest that given more training time and tuning of hyperparameters, double DQN can possibly converge to a consistent positive reward score. However running 50 epochs itself took around 3 hours thus indicating the existence of a major bottleneck for convergence.

Figure 7 represents the testing performance of the trained agent with double DQN for around 10 episodes. Again the success rate (ratio) of the agent is unsatisfactory given the dips in reward score to -300 and the instability in the agent's ability to correctly satisfy the requirement of the chosen scenario. Even though the agent has achieved a score close to 100 it is apparent from the graph that it is lacking given the rate of success.

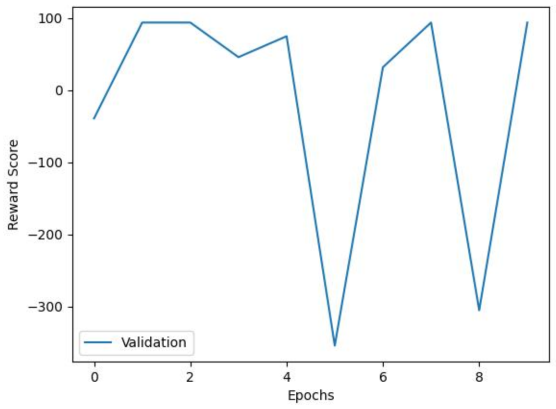


Figure 7

Dueling Deep Q-Network (Dueling DQN)

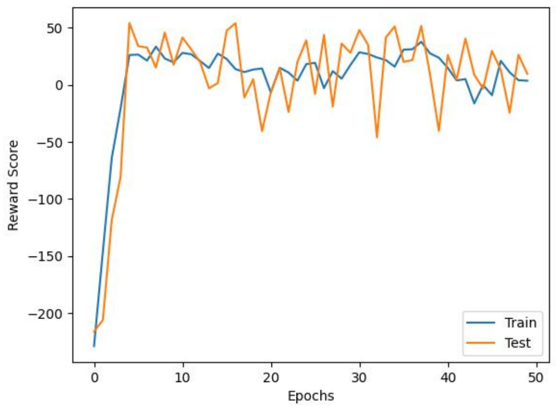


Figure 8: Baseline

Figure 8 represents the performance of the agent on dueling DQN using baseline hyperparameters (arbitrarily chosen). The agent displays highly consistent results with much stability compared to before. A major chunk of the rewards scored by the agent is positive after first few epochs on wards. This initiated the process of optimisation of the agent using dueling DQN to maximise it's performance.

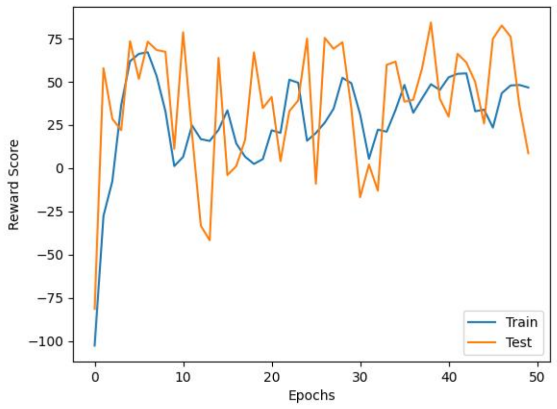
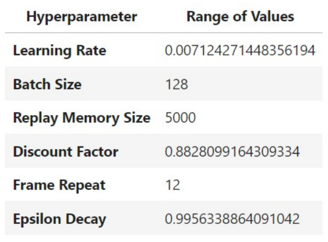


Figure 9: Optimised with best hyperparameters

Table 2 shows the best hyperparameters extracted from the Optuna study. Figure 9 shows the performance of the agent using the best hyperparameters. As can be seen, the training and testing scores are much more consistent than the previous agents trained on other variants. Overall success rate for Dueling DQN was 6x better than DQN and 1.5x better than double DQN. The agent manages to consistently achieve a higher positive score with less frequent dips (which are still not of larger magnitude).



Hyperparameter	Range of Values
Learning Rate	0.007124271448356194
Batch Size	128
Replay Memory Size	5000
Discount Factor	0.8828099164309334
Frame Repeat	12
Epsilon Decay	0.9956338864091042

Table 2: Best Hyperparameters

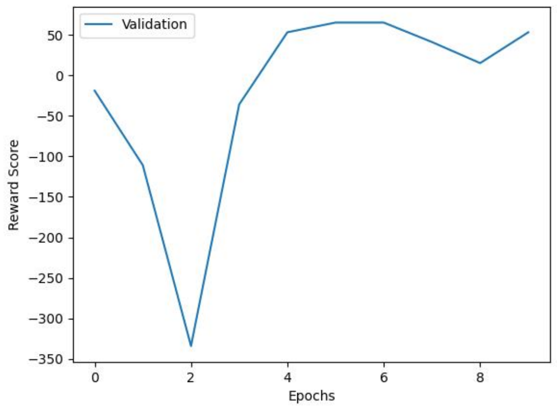


Figure 10

Figure 10 shows the final testing performance of the optimised agent on the scenario. As can be seen there is an upward trend of positive reward with each episode and the success rate of the agent is maximum in comparison to the other models thus rendering the dueling DQN hyperparameter optimised model to be the best performing of the three both in terms of model stability and convergence time and overall reward score accumulation.

Conclusion

The results show that the Duel DQN architecture outperforms the DQN architecture in terms of learning speed and final performance. The Duel DQN agent is able to achieve a higher score in the game environment and learns a more optimal policy faster than the DQN agent. This is likely due to the separation of the state value and action advantage functions, which helps to reduce overestimation of the Q-values and improve the stability of the learning process. We originally set out to compare the performance of the agent on standard DQN and double DQN. We managed to also additionally compare and optimise a dueling DQN architecture and achieve consistent reward performance for the agent.

References

Akiba, T.; Sano, S.; Yanase, T.; Ohta, T.; and Koyama, M. 2019. Optuna: A Next-generation Hyperparameter Optimization Framework. arXiv:1907.10902.

Bellman, R. 1954. The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60(6): 503 – 515.

Bergstra, J.; Yamins, D.; and Cox, D. D. 2013. Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, I–115–I–123. JMLR.org.

Golovin, D.; Solnik, B.; Moitra, S.; Kochanski, G.; Karro, J.; and Sculley, D. 2017. Google Vizier: A Service for Black-Box Optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, 1487–1495. New York, NY, USA: Association for Computing Machinery. ISBN 9781450348874.

Hasselt, H. 2010. Double Q-learning. In Lafferty, J.; Williams, C.; Shawe-Taylor, J.; Zemel, R.; and Culotta, A., eds., *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc.

Kempka, M.; Wydmuch, M.; Runc, G.; Toczek, J.; and Jaśkowski, W. 2016. ViZDoom: A Doom-based AI Research Platform for Visual Reinforcement Learning. arXiv:1605.02097.

Levine, S.; Finn, C.; Darrell, T.; and Abbeel, P. 2016. End-to-End Training of Deep Visuomotor Policies. arXiv:1504.00702.

Liaw, R.; Liang, E.; Nishihara, R.; Moritz, P.; Gonzalez, J. E.; and Stoica, I. 2018. Tune: A Research Platform for Distributed Model Selection and Training. arXiv:1807.05118.

Microsoft. 2021. Neural Network Intelligence.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2015. Human-level control through deep reinforcement learning.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. A. 2013. Playing Atari with Deep Reinforcement Learning. *CoRR*, abs/1312.5602.

Olsson, M.; Malm, S.; and Witt, K. 2022. Evaluating the effects of hyperparameter optimization in VizDoom.

Shekhar, S.; Bansode, A.; and Salim, A. 2022. A Comparative study of Hyper-Parameter Optimization Tools. arXiv:2201.06433.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book. ISBN 0262039249.

van Hasselt, H.; Guez, A.; and Silver, D. 2015. Deep Reinforcement Learning with Double Q-learning. *CoRR*, abs/1509.06461.

Wang, Z.; Schaul, T.; Hessel, M.; van Hasselt, H.; Lanctot, M.; and de Freitas, N. 2016. Dueling Network Architectures for Deep Reinforcement Learning. arXiv:1511.06581.