COMP 551 MINI-PROJECT 4 REPORT

Edmund Wu 261033501

Tianyi Xu 261082272

Santosh Passoubady 261098017

December 5, 2023

ABSTRACT

Reinforcement Learning (RL) is a type of machine learning where an agent learns to make decisions by interacting with an environment to achieve a goal. In our study, we implemented the paper "Playing Atari with Deep Reinforcement Learning," which pioneered the use of Deep Q-Networks (DQN) to play games. We trained a DQN on the Atari Breakout environment, following as closely as possible the architecture outlined in the original paper while focusing on tuning specific hyperparameters. Our objective was to evaluate the performance impact of these adjustments by benchmarking our modified DQN against the baseline model. This project aims to investigate how hyperparameter tuning can optimize reinforcement learning models for targeted tasks.

1 Introduction

The advent of deep learning has revolutionized various fields, including the domain of reinforcement learning (RL), which focuses on training models to make a sequence of decisions. The pioneering work by Mnih et al. in their paper "Playing Atari using Deep Reinforcement Learning" marked a significant milestone in this field. Their research introduced a novel deep learning model, a convolutional neural network trained with a variant of Q-learning, capable of learning control policies directly from high-dimensional sensory input. This model, known as the Deep Q-Network (DQN), represented a breakthrough in its ability to process raw pixels and estimate future rewards, thereby enhancing decision-making in complex environments.

In this project, we delve deeper into the potential of the DQN model by applying it to a specific context: the Atari Breakout game. Our work is not just a replication of Mnih et al.'s approach but an exploration of the intricacies of model fine-tuning. By focusing on one game, as opposed to the seven different Atari 2600 games considered in the original study, we provide a more concentrated analysis of how hyperparameter adjustments can influence the model's performance. This targeted approach enables us to dissect the DQN model's behavior in a constrained setting, offering a clearer view of its strengths and limitations.

Moreover, we benchmark our modified DQN model against the original, assessing the impact of hyperparameter tuning on various performance metrics. The primary aim is to explore the extent to which fine-tuning can enhance the model's efficiency and effectiveness, specifically in the context of Atari Breakout. This exploration not only contributes to the understanding of the DQN model's adaptability but also provides insights into the broader application of deep reinforcement learning models in domain-specific tasks.

Our study's findings are intended to serve as a comprehensive resource for those interested in the nuances of model optimization in RL. By presenting a detailed comparison with the baseline model and discussing the implications of our approach, we aim to illuminate the path for future research in the field of high-dimensional sensory environment learning.

The rest of the report is organized as follows. We first provide a reminder of the structure of a deep Q-network, as well as its problems, and the current different alternative models. Afterward, we show the results of our experiments that show which components do what in the deep Q-network. We finish with a conclusion.

2 System Model: Deep Q-Network (DQN)

2.1 Structure of a DQN

Consider an environment ε and an agent that acts in ε. A basic deep Q-network can be decomposed into two components: (1) A target network, and (2) an experience replay component. We first introduce the target network. Consider a policy function π that is a conditional probability distribution over the measurable space of actions given a state. Consider also a function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ where $\mathcal{S}$ and $\mathcal{A}$ are the state space and the action space respectively. The function Q is the action-value function, and it is given by $Q^{\pi}(s, a) = E_{\pi}[R_t|S_t = s, A_t = a]$ where E is the expected value and $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ is called the accumulated return, where $\gamma \in (0, 1]$ is the discount factor.

To our knowledge, in reinforcement learning problems, the agent aims to maximize the expected return $E^{\pi}[R_t|S_t = s]$ for each state $s \in \mathcal{S}$ (this is the so-called value function). One of the two common approaches is value-based methods. One such method is called Q-learning. Q-learning aims to find the optimal action-value function $Q^* = \max_{\pi} Q^{\pi}$. A first observation is that we are optimizing over a space of probability distributions that is a non-parametric setting. As mentioned in the paper, Bellman's iteration to obtain the optimal Q^* is not feasible. The standard approach is then to approximate Q^* by a parametric $Q(\cdot, \cdot; \theta)$ which can be a neural network, the so-called Q-network. The Q-network learns by optimizing a sequence of loss functions

$$L_i(\theta_i) = E\left(r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta_{i-1}) - Q(s_t, a_t; \theta_i)\right)^2$$

The target network is a copy of the Q-network updated every few steps (a hyperparameter). The target network fixes the parameters θ every so often, which can be thought of as a way for the agent to have more time to try out a way of executing actions. The target network stabilizes training and is the network that is used in DQN algorithms.

Another important component of deep Q-networks is the so-called experience replay. The goal of experience replay is to decorrelate and to induce stationarity in the training data. Experience replay introduces a set containing previously taken transitions. We then sample uniformly from this set and use it to update our network. As mentioned in the paper, experience replay first allows for efficient use of the data, second reduces correlation and thus variance, and third induces a time average in the distribution of the training data.

Deep Q-networks have certain limitations and are perhaps by far one of the least performant models. One of the main disadvantages of DQN is that experience replay limits the approach to maximize the expected return for each state to only off-policy methods. Generating data on-policy is not possible, since experience replay considers previously generated data under a different policy. Solutions to this include n-step Q-learning 2 or asynchronous methods for deep reinforcement learning 3 which can generalize to on-policy methods. Alternative reinforcement learning models include actor-critic models such as A3C and DDPG, or generative models based on diffusion 4.

2.2 Hyperparameter Tuning

The algorithm of a deep Q-network is given as in the original paper. We will not report it here again. However, the training loop of DQN can be broken into two main steps: the first is sampling an action and taking a transition step, and the second is sampling a transition and performing a gradient descent step in the Q-network.

More precisely, the first step selects an action following an ϵ-greedy policy where a larger $\epsilon \in (0, 1]$ usually implies a larger probability of exploration. Given the action, we then perform a step in the environment and we obtain a transition in return, which is stored in the experience replay buffer. This buffer has a limited amount of memory and a buffer size. The parameters of the Q-network are then updated with a batch of transitions sampled from the replay buffer, and fed into the target network. The update is done by a step of gradient ascent which also considers a hyperparameter called the discount factor γ. The discount factor dictates the value of the reward in terms of time. Depending on the case scenario, we may give preference to immediate rewards or delayed rewards. Finally, the target network is updated depending on a rate usually denoted by τ.

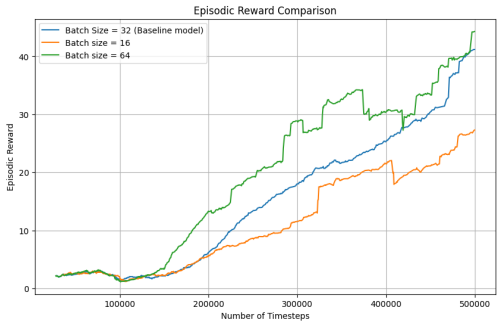
Another factor to consider is what parametric function $Q(\cdot, \cdot; \theta)$ is used to approximate Q^*. Markov decision processes are of pedagogical importance, but the extension of Q-networks to deep Q-networks usually considers neural networks as function approximators of the optimal action-value function. The discussion of the tuning of each hyperparameter will continue in the following section. There are of course more hyperparameters, but the ones mentioned were in our opinion the most interesting ones to look at. The tuning of these parameters to trivial settings, such as setting the discount factor to 1, contributes also to ablation studies in an intuitive way. Another example is setting the buffer size to be small which tells us of its contribution to the RL model.

3 Results

The set of hyperparameters that were used for our baseline DQN model across the experiment is defined in Appendix A. We standardly set the total training time steps to be 500000. Still, each experiment takes several low-digit hours to complete.

3.1 Batch Size

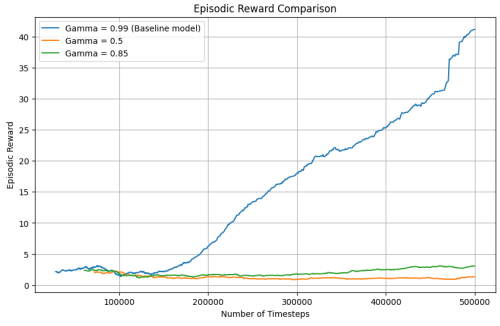
Figure 1: Batch Size Influence's Impact on Episodic Reward: Baseline (32), Smaller (16), Larger (64)



The baseline model with a batch size of 32 shows a steady and robust increase in reward, suggesting an optimal balance between learning stability and experience variety. Smaller batch sizes (16) appear to result in more volatile learning progress, possibly due to less stable gradient estimates. In contrast, larger batch sizes (64) demonstrate a smoother, yet slower increase in reward, which might indicate less frequent but more reliable updates to the network. Overall, batch size significantly impacts the learning efficiency and stability of the DQN agent.

3.2 Discount factor

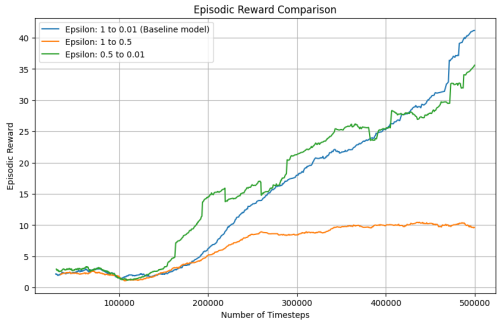
Figure 2: Gamma Sensitivity's Impact on Episodic Reward: Baseline (0.99), Lower (0.5), Higher (0.85)



The baseline model exhibits a strong performance with consistent growth in rewards, underscoring the importance of considering future rewards during training. In contrast, the models with lower gamma values plateau early, indicating a short-sighted learning approach that heavily favors immediate rewards and hinders the agent's ability to effectively learn long-term strategies. The results demonstrate the critical role of the discount factor in balancing short-term and long-term rewards within the reinforcement learning framework.

3.3 Exploration/Exploitation

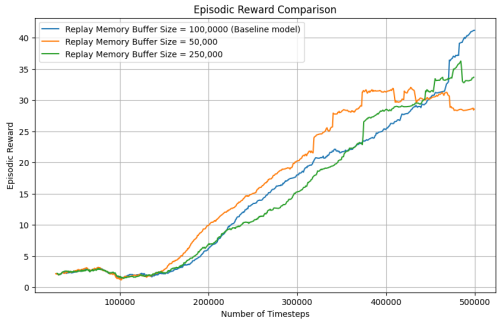
Figure 3: Epsilon Decay's Impact on Episodic Reward: Baseline (1 to 0.01), Conservative (1 to 0.5), Aggressive (0.5 to 0.01)



The baseline model, with an epsilon decay from 1 to 0.01 (blue), shows a gradual and sustained improvement, indicating a balanced exploration-exploitation trade-off throughout the training. The more conservative epsilon decay from 1 to 0.5 (orange) results in a lower overall reward, suggesting that less exploration may hinder the agent's ability to discover optimal policies. Meanwhile, the green line, representing an epsilon starting at a lower bound of 0.5 decaying to 0.01, converges to a similar performance as the baseline, implying that starting with a lower epsilon does not significantly impact long-term learning outcomes but may speed up initial performance gains

3.4 Experience Replay Buffer Size

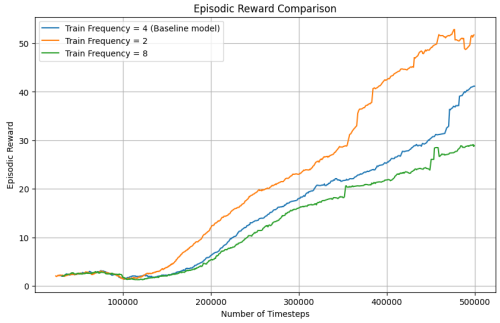
Figure 4: Replay Buffer Size's Impact on Episodic Reward: Baseline (100k), Small (50k), Large (250k)



The baseline model with a buffer size of 100,000 (blue) achieves a steady increase in rewards, indicating a good balance of experience diversity and learning stability. A smaller buffer size of 50,000 (orange) exhibits similar performance initially but slightly lower rewards in the long run, potentially due to limited diversity of experiences. The largest buffer size tested, 250,000 (green), shows a lag in the early stages but eventually surpasses the baseline, suggesting that a larger buffer may contribute to a more comprehensive learning of the environment at the expense of slower initial learning progress. The results underscore the importance of buffer size in the effectiveness of experience replay.

3.5 Training Frequency

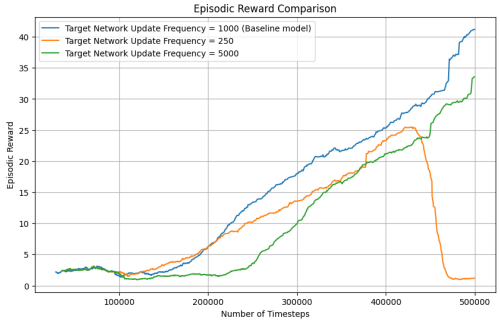
Figure 5: Training Frequency's Impact on Episodic Reward: Baseline (every 4 steps), Frequent (every 2), Infrequent (every 8)



The agent trained every 2 timesteps exhibits the fastest improvement, reaching higher rewards more quickly than the baseline, suggesting more frequent updates facilitate faster learning. The baseline model shows moderate growth, while the agent trained every 8 timesteps demonstrates the slowest progression, indicating less frequent training may impede the agent's ability to adapt and improve. These observations emphasize the significance of training frequency on the speed and quality of the agent's learning trajectory.

3.6 Target Network Update Frequency

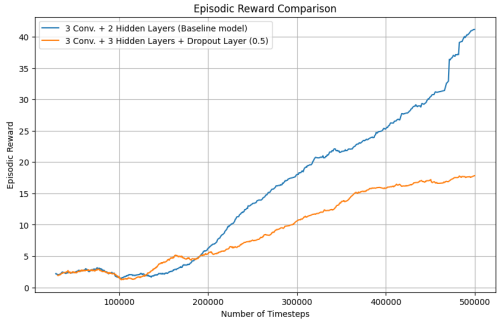
Figure 6: Target Network Update Frequency's Impact on Episodic Reward: Baseline (every 1000 steps), Frequent (every 5000), Infrequent (every 250)



The baseline model with a target update frequency of 1000 timesteps (blue) shows steady and consistent reward improvement, suggesting this frequency helps maintain a stable learning progression. A more frequent update every 250 timesteps (orange) leads to faster initial learning but encounters instability later, possibly due to overfitting to recent experiences. Conversely, updating every 5000 timesteps (green) results in slower learning, implying that the agent's Q-value estimations may become outdated, hindering its ability to effectively learn optimal policies. These results highlight the delicate balance needed in the timing of target network updates to optimize learning.

3.7 Structure of the Q-Network

Figure 7: Architecture's Impact on Episodic Reward: Baseline (2 Hidden Layers), Modified (3 Layers + Dropout of 0.5)

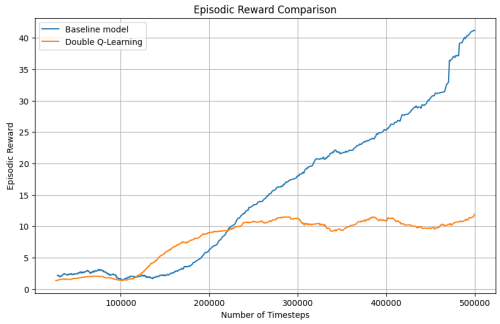


While the baseline model follows a traditional approach in its Q-network which comprises of a sequence of convolutions and linear layers, our updated Q-Network architecture increased the number of convolutions layers as well as introduced dropout layers. These changes were done in hopes for the network to extract more nuanced features from the input and mitigate over fitting(a common challenge in deep learning models deployed in high-dimensional data).

Now, surprisingly, the results from our experiments showed that the updated Q-Network architecture did not perform as well when compared to the baseline model. In fact, results from the new model showed slower learning rates and also achieved lower overall rewards. This under-performance raises some important considerations regarding model complexity. In fact, it suggests that more complex architectures do not always translate to better performance in reinforcement learning tasks. These findings emphasize the need for a careful balance between model sophistication and the practicalities of the learning environment.

3.8 Double Q-Learning

Figure 8: DQN vs. Double Q-Learning: Reward Progress Over Timesteps



Double Q-Learning is specifically designed to counter the overestimation bias that tend to happen in standard Q-Learning. This approach mainly aims to refine the action-value estimation process by both normalizing the layer inputs and independently assess the value of states and actions. However, the results from the Double Q-Learning model proved to be less effective than the baseline model as the Double Q-Learning model exhibited both a less stable learning curve and also scored lower overall. These results suggest a potential underestimation of action values or an over-regularization effect, which could be due to the model's complexity.

4 Discussion and Conclusion

4.1 Challenges

We had a few challenges. The main challenge was to choose a suitable paper that both allowed for a reasonably small compute power capacity, and that did not have an implementation that was too confusing or that relied on outdated Python dependencies. As we were still familiarizing ourselves with reinforcement learning, we wanted to choose something that was within our capacities, and truth be told, *Playing atari with deep reinforcement learning* was not our first choice. However, we have opted for this paper because of its semi-low resource demand for experimentation (depending on the total number of time steps), because it already had existing code templates, and of course because we wanted to do something related to reinforcement learning.

4.2 Relation with Class Material

Although reinforcement learning is a different machine learning paradigm and not the subject of this class, there are still a few related concepts. While obvious, we will nevertheless remark that the Q-network can be any neural network that we want; a convolutional neural network, a recurrent neural network, a multi-layered perceptron, the choice is yours. Furthermore, as is typically done in optimization contexts, updates of the parameters are done by using the gradient of a loss. Here we have the mean squared error. Generalization techniques that we have seen, such as data augmentation or regularization, should be possible to implement here when adapted to the situation.

Perhaps it is easier to mention the differences. We are not minimizing a cost function, but rather maximizing the expected return given each state. We do not perform gradient descent, but an iterative gradient ascent. The data points are not feature and target pairs, but rather states, actions, values, returns, and future states. Reinforcement learning models also include different concepts, and we would argue that RL models are more probabilistic than the models that we have seen in class.

4.3 Key Takeaways

All in all, in this mini-project, we have created a comprehensive exploration of DQN when applied to the Atari Breakout game with a special emphasis on tuning the hyper parameters for improving performance. There are multiple key takeaways from this project, most noticeably the impact of hyper parameter adjustments on DQN efficiency. In this project, we have found that tweaking the learning rate, discount factor, and exploration rate can lead to substantial differences in how quickly and effectively the model learns the game. Also, adjustments in the experience replay buffer, which stores past states, actions, and rewards, were found to influence the model's ability to learn from a diverse set of experiences, enhancing its generalization capabilities.

Some possible directions for future investigation could be to explore alternative reinforcement models such as Actor-Critic models, which combine the benefits of policy-based and value-based approaches. Moreover, we could also explore different environments such as other simulations in OpenAI's Gym.

5 Statement of Contributions

Each member contributed equally to this project. Each of us completed this lab individually and wrote the report by merging our responses and code.

References

[1] 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.

[2] J. Fernando Hernandez-Garcia, Richard S. Sutton. 2019. Understanding Multi-Step Deep Reinforcement Learning: A Systematic Study of the DQN Target. https://arxiv.org/abs/1901.07510

[3] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, Koray Kavukcuoglu. 2016. Asynchronous Methods for Deep Reinforcement Learning. https://arxiv.org/abs/1602.01783

[4] Anurag Ajay, Yilun Du, Abhi Gupta, Joshua Tenenbaum, Tommi Jaakkola, Pulkit Agrawal. 2023. Is Conditional Generative Modeling all you need for Decision-Making?. https://arxiv.org/abs/2211.15657

[5] Richard S. Sutton, Andrew G. Barto. 2018. Reinforcement Learning: An Introduction. 2nd edition. MIT Press, Cambridge. http://incompleteideas.net/book/the-book-2nd.html

Appendix A Default Model Parameters

Table 1: Hyperparameters Used in the Training of Our Base DQN Model

Hyperparameter	**Value**
Total Timesteps	500,000
Learning Rate	1e-4
Number of Environments	1
Buffer Size	100,000
Gamma (Discount Factor)	0.99
Tau (Target Network Update Rate)	1.0
Target Network Update Frequency	1,000
Batch Size	32
Starting Epsilon (Start-e)	1.0
Ending Epsilon (End-e)	0.01
Exploration Fraction	10%
Learning Starts (Timestep)	80,000
Training Frequency	Every 4 timesteps