Playing SeaQuest with Deep Q-Network Variants

Jakob Kirby
Computer Science & Engineering
Texas A&M University
College Station, USA
kirbyj3684@tamu.edu

Noam Gariani
Computer Science & Engineering
Texas A&M University
College Station, USA
noamgariani@tamu.edu

***Abstract*—We investigated the effectiveness of various improvements to Deep Q Networks to improve the performance of an agent at the Atari game SeaQuest. For each variant we experimented with different hyper parameter settings, found the best performing variant, and compared it to the baseline vanilla DQN implementation. From our results we found that each individual variant can improve performance but integrating the individual variants into one DQN variant will require careful tuning of hyper parameters to maintain stability and effective learning. Also we found that the variants require longer training periods than our given time frame to demonstrate the improvements of the enhancements done over the baseline DQN. However, our work showcases the impact each variant can have in improving the performance.**

***Index Terms*—Deep Q-Network (DQN), SeaQuest, Atari, Categorical DQN (C51), Noisy Networks (NoisyNet), Stochastically, Reward Distribution Learning**

Video Link: https://youtu.be/vrJe15SYtBo

I. Introduction

SeaQuest is a classic Atari 2600 game where the player controls a submarine navigating underwater to rescue divers and collect points. The submarine must avoid or destroy enemy submarines, sharks, and other threats while managing an oxygen meter that depletes over time. The player earns points by picking up divers and resurfacing to replenish oxygen. The game becomes progressively harder as the enemies increase in number and speed, challenging the player's reflexes and strategy.

SeaQuest is a popular benchmark in Deep Q-Network (DQN) research because of its challenging game play, which requires both strategic planning and quick decision-making. DQNs are well-suited for such tasks, as they use reinforcement learning to train an agent to maximize cumulative rewards over time. However, the game highlights key difficulties for DQNs, such as discovering the need to surface for air before the oxygen runs out—a behavior not explicitly rewarded but essential for survival. Balancing immediate rewards, like shooting enemies or rescuing divers, with these implicit survival strategies tests the DQN's ability to learn complex, non-linear reward structures. This makes SeaQuest an excellent testbed for evaluating how well a DQN handles long-term planning and exploration challenges.

One of the key problems in the baseline Deep Q-Network (DQN) is that it has no intuition for trade-offs like balancing immediate rewards (e.g., shooting enemies) with long-term survival strategies (e.g., managing oxygen).

For instance, with categorical DQN the submarine learns not just a single "best action" but a range of possible outcomes, like a weather forecast for underwater survival. For example, it can better judge when surfacing is worth the risk by learning the potential rewards of each scenario. With NoisyNet DQN, the submarine experiments with diverse strategies by adding learnable randomness to its decisions. This ensures it doesn't get stuck in predictable patterns and finds new, better tactics over time.

The idea is by using these variants to improve on issues found in baseline DQN over time it should perform better.

To do this we have done an in-depth exploration of Deep Q-Network (DQN) variants for the Atari game SeaQuest. The focus is on enhancing baseline DQN performance by introducing and evaluating modifications that address exploration challenges, reward prediction, and computational efficiency. This paper contributes the following:

- **Comprehensive Analysis of DQN Variants:** We evaluate multiple DQN variants, including Categorical DQN, NoisyNet DQN, and their combination, to improve performance in SeaQuest.
- **Architectural and Hyper parameter Improvements:** Enhancements to network architecture and hyper parameters are implemented to optimize the learning process.
- **Empirical Evaluation:** Experiments are conducted with consistent metrics, including mean reward and episode length, to compare performance across 200,000 time steps and longer durations for selected methods.
- **Insights into Sparse Reward Handling:** Techniques to address sparse rewards in SeaQuest are evaluated, includ-

ing reward distribution learning and enhanced exploration mechanisms.

- **Foundations for Future Work:** We propose integrating additional components from Rainbow DQN, such as prioritized experience replay and multi-step learning, to further improve stability and scalability.

The claims made in this paper are supported by rigorous experimentation and analysis, detailed in Section 2 (Approaches) and Section 3 (Experiments). We provide evidence for how specific modifications impact the agent's performance and offer insights into their long-term implications.

By addressing key limitations of the baseline DQN, this paper aims to advance the understanding of reinforcement learning techniques and their application to challenging environments like SeaQuest.

A. Problem Statement

Deep Q Networks are a widely used and popular choice for reinforcement learning in Atari games. Our Atari game of choice SeaQuest follows deterministic rules meaning the same action in the same state will always lead to the same next state. However, the Arcade Learning Environment and OpenAI Gym introduce stochastic elements like frame skipping and sticky actions to add randomness into the training process [4]. While DQN performs well, there are variants of DQN that aim to improve performance. In this project, we want to explore different variants of DQN to see if we can enhance the agent's performance in SeaQuest. The reinforcement learning approach we're using is sequential in nature because our DQN agent will make decisions based on the current state and try to maximize rewards over time. In SeaQuest, our DQN agent will need to make a series of decisions to navigate the submarine, collect divers, avoid or destroy enemies, and manage oxygen levels. Each action the agent takes will affect the next state and the reward it receives. The stochastic elements added by ALE and OpenAI Gym make the agent consider the randomness in the environment when making decisions.

II. Approaches

A. Methodology and Experiments Setup

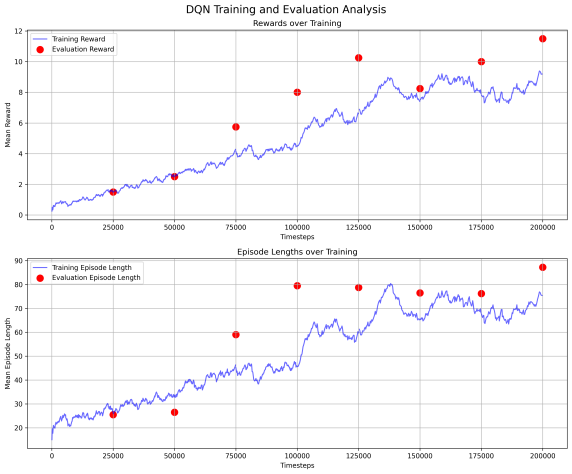
We will be using the original implementation of Deep Q-Network (DQN) as a baseline on SeaQuest and then multiple other implementations that will be considered. DQN has already been shown to have good results on Atari games like Seaquest but the various searched approaches are done for the purpose of finding better results.

We considered the following approaches: DQN, Categorical DQN, NoisyNet DQN, NoisyNet/Categorical DQN, Dynamic Action Repetition DQN, Double Q-Learning DQN, Dynamic Frame Skip DQN, Rainbow DQN, and Prioritized Experience Replay DQN.

These approaches have been talked about in the literature review section more in-depth and the following subsections will talk more to each implemented algorithm along with metrics.

B. DQN

By importing stable_baselines3 that baseline version of DQN could be easily implemented and after running the 'Rewards over Training' and 'Episode Length over Training' graphs were made.

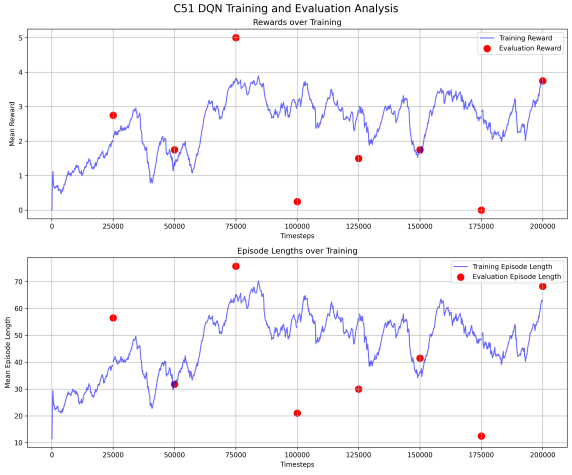


These results show good because the baseline version of DQN performs well even when the time steps are low. One of the constraints of the project was not being able to run with many time steps due to Google Colab constraints. This resulted in generally max 200,000 runs per algorithm run. In these conditions plain baseline DQN tends to perform better, but in the long run around >2 Million iterations these modifications should perform better than baseline DQN.

You can see a constant upwards trend of the rewards over the time steps in the first graph.

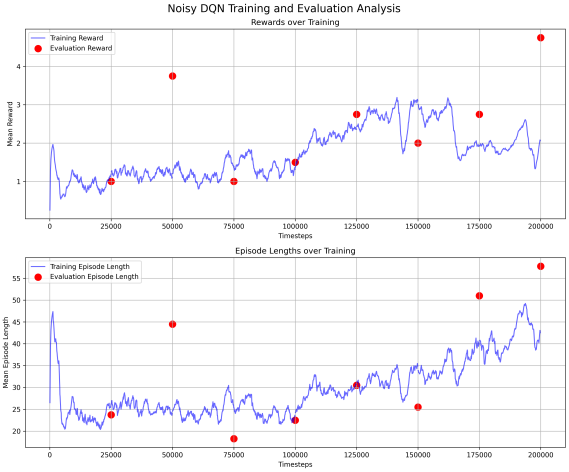
C. Categorical DQN

Categorical DQN improves on Deep Q Network by learning from a full distribution of possible return values instead of just a single expected return value. The approach discretizes this distribution into 51 atoms by default, where each atom represents a different potential return value with its associated probability. This distributional perspective allows the agent to better capture uncertainty in value estimation.



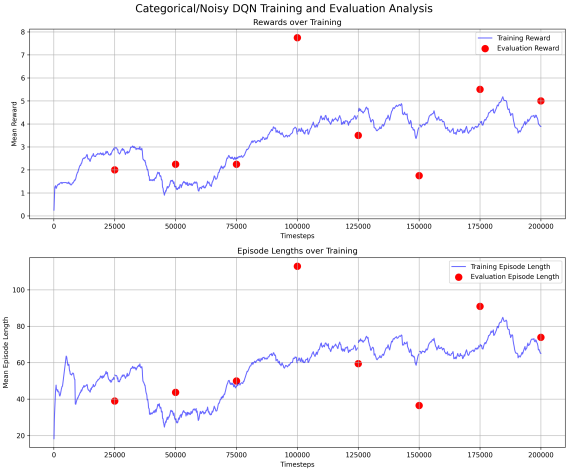
D. NoisyNet DQN

NoisyNet DQN enhances the Deep Q Network architecture by replacing standard fully connected layers with noisy layers that add learnable randomness to the weights and biases. Unlike traditional DQN which uses epsilon-greedy exploration, NoisyNet enables state-dependent exploration by learning the noise parameters through gradient descent along with the network weights. This allows the agent to learn how much noise to add in different states, automatically adjusting its exploration strategy based on uncertainty in different situations. The approach leads to a deeper exploration of the game compared to the selection of random actions.



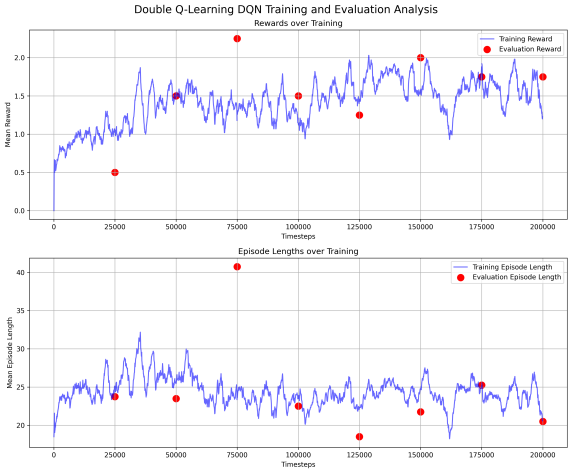
E. NoisyNet/Categorical DQN

NoisyNet/Categorical DQN combines the enhancements of both approaches into a single DQN variant. So from Categorical it learns a distribution of possible return values. From NoisyNet it learns from noise in network. With both the agent will better understand uncertainty in value estimation and explore the game deeper through noise. So the idea is that with both the agent will perform better at the game then just using one of the methods alone.



F. Double Q-Learning DQN

Double Q-Learning DQN is an improvement over the standard Deep Q-Network (DQN) designed to reduce overestimation bias in Q-value predictions. Traditional DQN uses the same network for both action selection and value estimation, which can lead to overvalued actions. Double Q-Learning addresses this by decoupling these steps through the use of two networks: the online network, which selects the action with the highest Q-value, and the target network, which evaluates the value of that action. This separation improves the stability and accuracy of the Q-value updates. A more detailed description of this method and its applications is expanded upon in the literature review section [9].

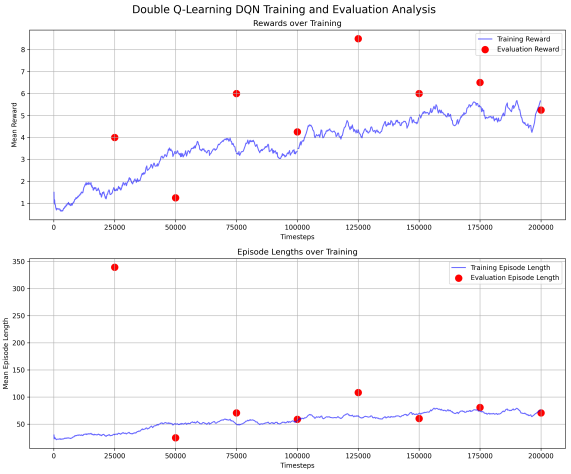


The updated Double DQN implementation introduces several improvements over the original by refining key hyper parameters and mechanisms to enhance training stability and performance. The changes are summarized as follows:

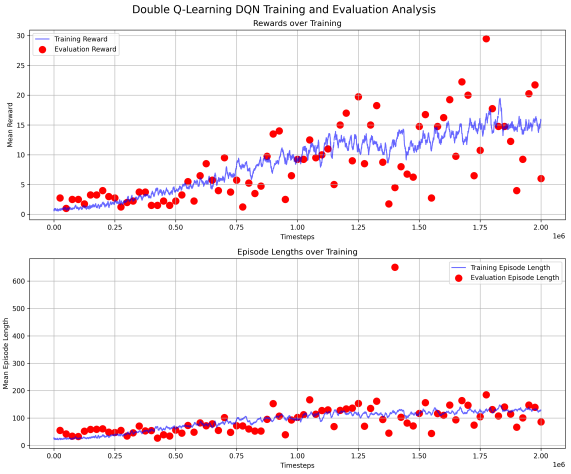
- **Learning Rate:** Increased from `1e-4` to `5e-4`. A higher learning rate accelerates the adaptation of the model to new information, enabling faster convergence during training.
- **Replay Buffer Size:** Expanded from `30,000` to `100,000`. A larger buffer provides more diverse experiences for training, reducing the risk of over fitting to recent episodes and enabling better generalization.
- **Batch Size:** Increased from `64` to `128`. Using larger batches improves gradient estimates, which can result in more stable and effective learning updates.
- **Tau (Polyak Averaging):** Reduced from `1.0` (hard updates) to `0.005` (soft updates). This change ensures smoother transitions in the target network updates, improving training stability.
- **Target Update Interval:** Reduced from `10,000` to `500`. More frequent updates allow the target network to provide faster and more consistent feedback, enhancing learning efficiency.
- **Exploration Fraction:** Lowered from `0.3` to `0.2`. This adjustment accelerates the transition from exploration to exploitation, allowing the agent to focus on learning an optimal policy sooner.
- **Final Exploration Epsilon:** Decreased from `0.05` to `0.01`. This enables the agent to exploit learned policies more effectively in later stages of training.

In addition to these hyper parameter changes, the policy net-

work architecture was enhanced by using two fully connected layers with 256 units each, improving the model's capacity to learn rich feature representations. Furthermore, gradient clipping with a maximum norm of 10 was added to the loss function to ensure stable training by mitigating the risk of exploding gradients. These improvements collectively aim to enhance the agent's learning process and overall performance.



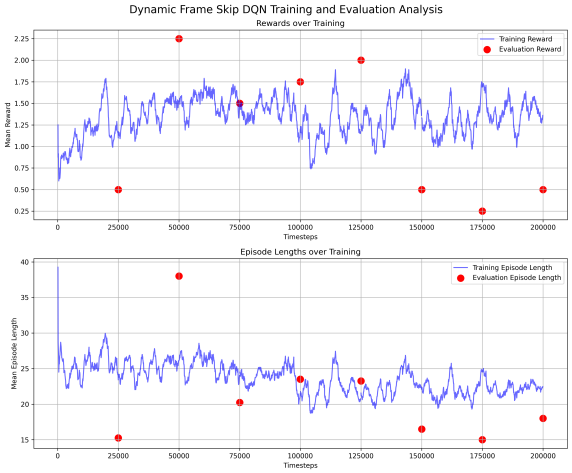
As can be seen in the above graph the performance increased by a lot. Because this was one of the best performing algorithms at 200,000 out of all the experiments, it was run again at 2,000,000 iterations to see long term improvement of the model.



In this graph it can be seen with more iterations the performance increases significantly. So by keeping most of the testing at 200,000 time steps is not the most ideal for testing but is a technical constraint. This graph shows the positive performance of this method though. With more time steps a direct comparison with baseline DQN may show better performance in this method.

G. Dynamic Frame Skip DQN

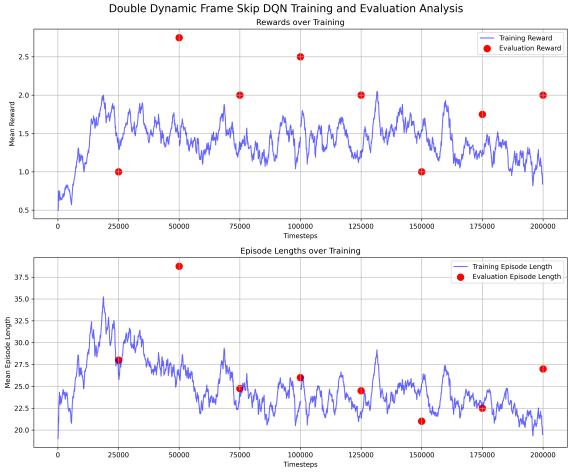
Dynamic Frame Skip DQN is a variation of the standard Deep Q-Network (DQN) that aims to improve efficiency and performance by dynamically adjusting the frame-skip rate during training. Frame skipping is a common technique where the agent selects actions less frequently, executing the same action for multiple consecutive frames to reduce computational cost. However, fixed frame-skip rates can lead to suboptimal decisions in dynamic environments. Dynamic Frame Skip DQN overcomes this by using a model-based approach to adaptively adjust the skip rate based on the complexity of the current state, allowing the agent to focus on critical moments requiring precise control while skipping less important frames. This method is further analyzed in the literature review section [6].



The results shown in this graph are not that great as it did not increase much over the span of 200,000 time steps in contrast to other methods like Double DQN or Baseline DQN. However, hypothetically, when testing with millions (>2M) time steps it should perform better.

H. Double Dynamic Frame Skip DQN

The combination of Double DQN and Dynamic Frame Skip DQN leverages the strengths of both methods to enhance learning efficiency and decision-making in complex environments. Double DQN addresses overestimation bias by decoupling action selection and evaluation, ensuring more stable Q-value updates. Meanwhile, Dynamic Frame Skip DQN optimizes computational efficiency by adaptively adjusting the frame skip rate based on state complexity. Together, these approaches improve both the accuracy of value predictions and the agent's ability to respond effectively to critical environmental changes.



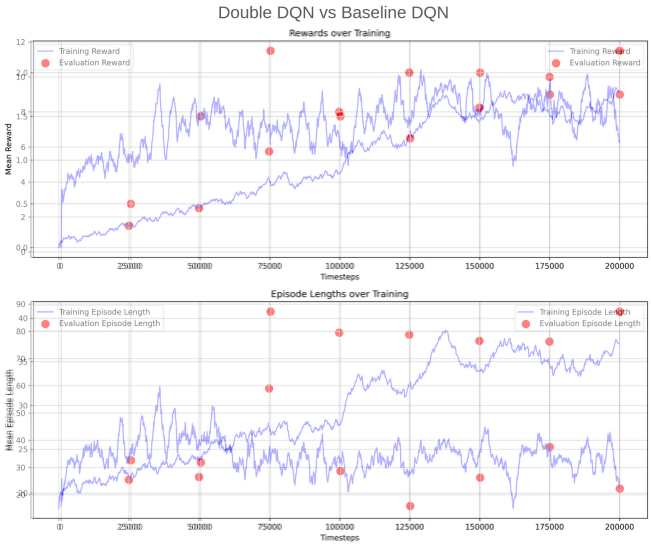
This method based on the graph with 200,000 time steps seems to perform worse that just Dynamic Frame Skip DQN however it is unclear the long term performance if tested with ¿2M time steps. It could show more positive performance or continue a plateau or downward trend.

I. Other methods

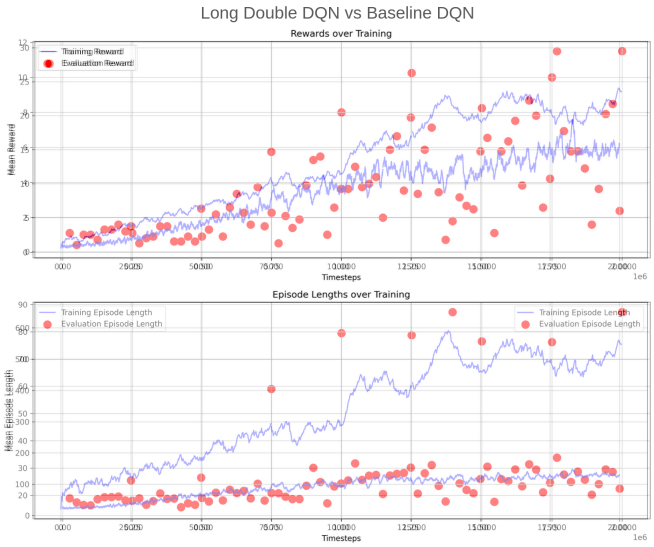
Dynamic Action Repetition, Rainbow and Prioritized Experience Replay DQN were considered algorithms that had code for them but struggled to run on Google Colab. This is discussed more in the future works section.

J. Comparative Analysis

Looking at all of the graphs, you can see improvement after certain changes. As mentioned before even with 2,000,000 iterations it is hard to see the full impact of the algorithm in comparison to DQN. Most of our algorithms were run with 200,000 time steps and Double DQN was shown to be the best out of the ones run at that level. The chart can be seen below:



In this graph there is an overlay that shows the growth rate rather than actual reward. It can be confusing because it may seem that Double DQN which is the one that has a more sharp increase is better than the more steady increasing Baseline DQN; however, Baseline DQN has a larger reward at the end and a more steady increase. However, this has been the best one in the context of 200,000 time steps.



In this next comparative graph is comparing the increased time steps to 2,000,000 for the improved Double DQN model against the 200,000 time steps of Baseline DQN. Here it shows the increase of Baseline DQN to be larger than double DQN with 2M time steps however the reward for Double DQN is higher in the long run.

Overall, when comparing the graphs baseline DQN seems to perform better because of lack of testing with large amounts of time steps. However, there are promising future results based on the code written if we would be able to run it with more GPU power.

K. Best Method

The best method was found to be Double DQN because of the comparative analysis. However, it is possible that other methods that modified the original baseline DQN perform better long term in ¿2M time steps like 5-6M and it was not seen.

L. Implementation Details

Our neural network architecture builds upon the foundations established in the DQN research papers. While maintaining the basic structure, we introduced several architectural modifications to potentially improve performance.

Base Architecture: The original DQN architecture used three convolutional layers conv1 is 16 filters 8×8 kernal 4 stride, conv2 is 32 filters 4×4 kernel 2 stride, and conv3 64 filters 3×3 kernel stride 1). [7] Our modified architecture is as follows:

- **Input Processing**:
  - Raw Input: Box(0, 255, (210, 160, 3), uint8)
    - Height: 210 pixels
    - Width: 160 pixels
    - Channels: 3 (RGB)
    - Data type: uint8 (0-255)
  - Preprocessing:
    - Normalization: x.float() / 255.0 (scales values to [0,1])

- **Convolutional Layers**:
  - Conv1: 32 filters, 8x8 kernel, stride 4
  - Conv2: 64 filters, 4x4 kernel, stride 2
  - Conv3: 128 filters, 3x3 kernel, stride 1
  - Conv4: 128 filters, 3x3 kernel, stride 1
- **Fully Connected Layers**:
  - FC1: feature_size $\rightarrow$ 512 units, ReLU
  - FC2: 512 $\rightarrow$ 256 units, ReLU
  - Output Layer (varies by DQN variant):
    - Categorical DQN: 256 $\rightarrow$ n_actions * n_atoms
      - Outputs a distribution over said number of atoms for each action
      - Total output size: number of actions $\times$ number of atoms
    - Other DQN variants: 256 $\rightarrow$ n_actions
      - Outputs a single Q value for each action
      - Total output size: number of actions
- ReLU activation functions between convolutional layers

Key Modifications: Our architecture differs from the original papers in the following ways:

- Increased filter count for each layer to capture more complex features
- Added an additional convolutional layer (Conv4) to increase network depth
- Maintained ReLU activations but with deeper architecture

III. Experiments

A. Training Metrics

- **Episode Length (rollout/ep_len_mean)**: Average number of steps per episode, indicating how long the agent survives in the environment.
- **Episode Reward (rollout/ep_rew_mean)**: Average reward obtained per episode, measuring the agent's performance in achieving the task objectives.
- **Exploration Rate (rollout/exploration_rate)**: The epsilon value in ϵ-greedy exploration, tracking how the exploration strategy evolves during training.
- **Training Speed**:
  - *Frames per Second (time/fps)*: Processing speed of the training
  - *Total Time Steps (time/total_timesteps)*: Cumulative number of environment interactions
  - *Time Elapsed (time/time_elapsed)*: Total training duration

B. Evaluation Metrics

During training their are evaluation phases for every 25,000 time step intervals, during evaluation we measure:

- **Mean Episode Length (eval/mean_ep_length)**: Average length of evaluation episodes
- **Mean Reward (eval/mean_reward)**: Average reward achieved during evaluation episodes

C. Hyper parameter Selection

a) Common Parameters: For all experiments unless stated otherwise, we use a buffer size of 30,000 due to Google Colab RAM limitations, target update interval of 10,000, learning rate of 1e-4, tau of 1.0, and gamma of 0.99. All experiments run for 200,000 time steps.

b) Categorical DQN (C51): We conducted three experiments varying the number of atoms and value distribution range:

- Experiment 1: Default values from research paper (51 atoms, range [-10, 10])
- Experiment 2: Wider distribution (51 atoms, range [-50, 50])
- Experiment 3: Increased atoms for wider range (81 atoms, range [-100, 100])

The batch size was set to 32 for all C51 experiments.

c) Noisy DQN: We tested three values of std_init to evaluate different exploration strategies:

- 0.1: Low exploration
- 0.5: Default value recommended from research paper allows balanced exploration and exploitation
- 1.0: High exploration

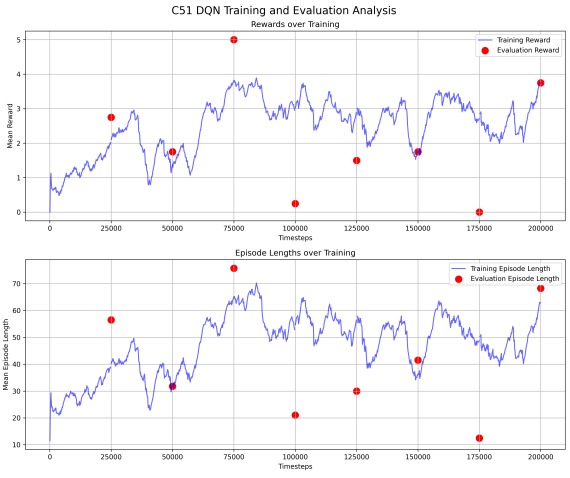
Following the NoisyNet research paper, we used a batch size of 128. All exploration-related parameters exploration fraction, initial eps, final eps were set to zero as exploration is handled by the noisy networks.

d) Noisy Categorical DQN: We conducted two experiments:

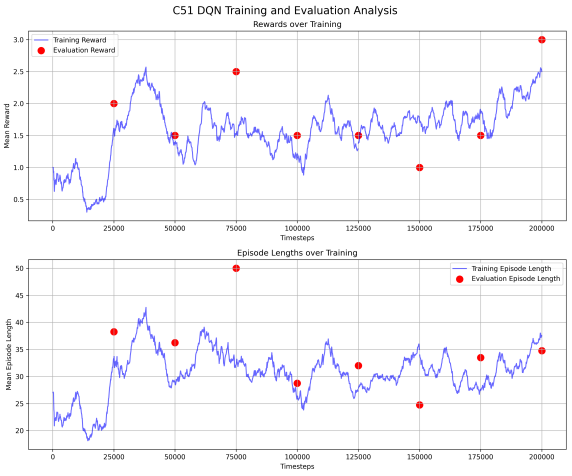
- Experiment 1: Combined best parameters from C51 and Noisy DQN (std_init = 0.5, 51 atoms, range [-10, 10])
- Experiment 2: Optimized parameters found through trail and error after poor initial performance (std_init = 0.1, 51 atoms, range [-100, 100], batch size = 64)

D. Results

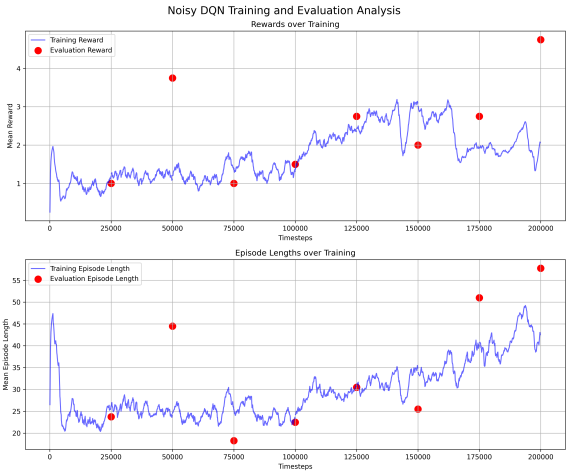
1) C51 Experiment Results: Experiment one



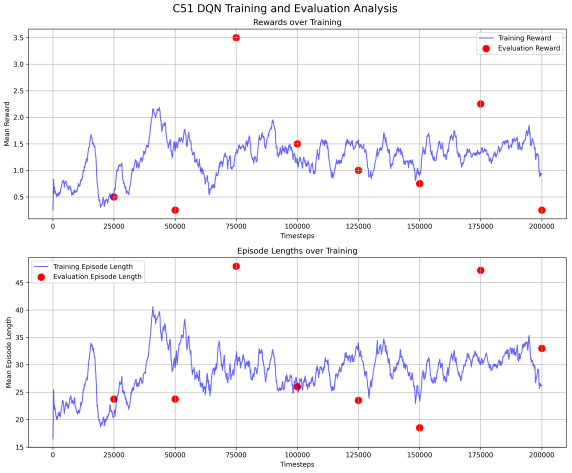
Experiment two



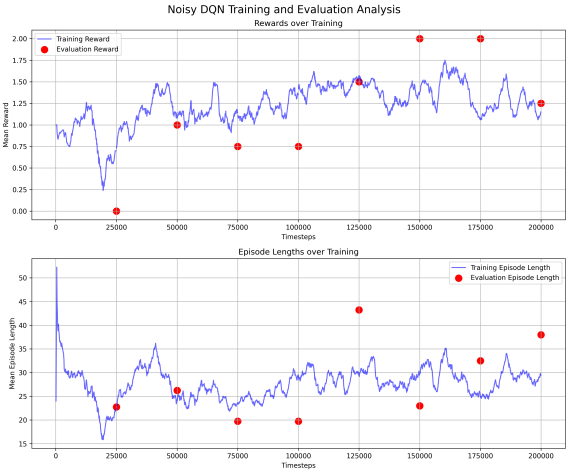
Experiment three



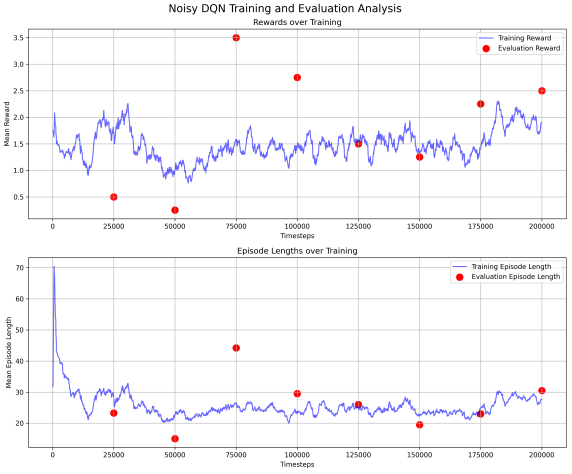
Experiment three



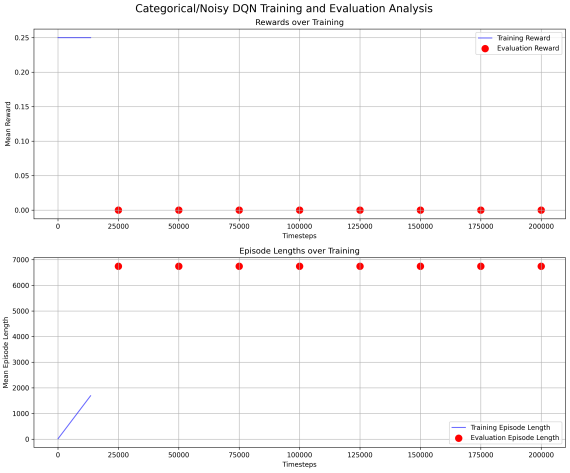
2) NoisyNet Experiment Results: Experiment one



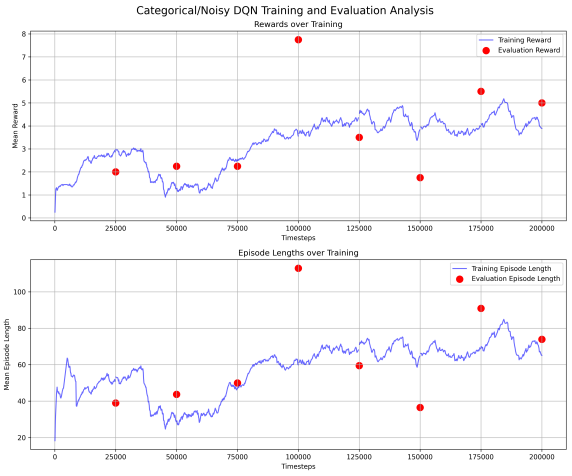
3) NoisyNet/Categorical DQN Experiment Results: Experiment one



Experiment two



Experiment two



E. Analysis

a) Categorical DQN: Looking at the results of experiment one, the training stays stable with some oscillation in performance. For example, the agent's mean reward can jump back and forth between 2 and 4. In the early phase of training, the agent is learning well. The middle phase is where we see oscillation in performance, while in the later phase the agent performs at its best. If training continued for longer, we would likely see a gradual increase in performance. The mean episode length starts around 20 to 30 steps and ends with a mean of around 60 to 70 steps.

In experiment two, the mean reward and episode length do not reach the same levels of performance as in experiment one. The agent has to learn from a wider value range of -50 to 50 compared to -10 to 10. This may have introduced complexity that requires more training time. The mean reward oscillates between 1 and 2 it reaches 2.5 early in training but not achieving that level again until the end of training. The episode length ends around 35 to 40 steps, which is lower than the 60 to 70 seen in experiment one.

Experiment three shows results similar to experiment two, with performance sometimes being even lower. The mean reward oscillates between 1 and 2, sometimes dropping below 1. The mean episode length ends around 25 to 30, the lowest of the three experiments.

Among the three setups, the first experiment with 51 atoms, v_min=-10, and v_max=10 performed the best. In experiments two and three, increasing the distribution range and number of atoms did not yield the same performance as experiment one within 200,000 time steps. Based on these results, for the game SeaQuest, the recommended default values for Categorical DQN with 51 atoms, v_min=-10, and v_max=10 is the optimal configuration.

b) Noisy DQN: In experiment one, the training shows variance in the early phase with the reward bouncing between 1 and 2. As it progresses into the middle phase it becomes more stable oscillating between 1.25 and 1.75. In the later phase, the performance remains stable with a gradual increase but the overall performance is low with the mean reward ending around 2. The episode length remains consistent between 20 and 30 steps throughout the training.

Experiment two shows stable training in the early and middle phases with a gradual increase in performance. However, in the later phase the mean reward oscillates between 2 and 3. Although it exhibits better performance compared to the NoisyNet in experiment one there is more variance. The mean episode length remains stable starting around 25 steps and slowly increasing to 40.

In experiment three, there is variance in performance during the early phase. However in the middle and later phases, it stabilizes slightly oscillating between 1 and 1.75. There is no significant improvement seen with the performance staying around 1 and 1.75. The episode length shows high variance and oscillation in the beginning phase but stabilizes in the middle and later phases, remaining around 25 to 30 steps.

Comparing the three experiments, experiment two using the recommended default value, which is std_init set to 0.5, performs the best. The NoisyNet DQN with std_init set to 0.5 in experiment two learns faster than the one with std_init set to 0.1 in experiment one. Although experiment two exhibits some variance in performance, it achieves a higher mean reward and episode length compared to experiments one and three.

c) Noisy/Categorical DQN: For the first experiment, we can see that the agent is not learning as intended. It has developed a survival bias, resulting in high episode length and low reward for each episode.

In experiment two, we used trail and error to adjust the hyper parameters, we found hyper parameter values that reduced exploration and increased the distribution range, allowed us to achieve a balance between exploration and exploitation, allowing the agent to learn instead of staying in a local safe policy. The training remains stable and gradually increases. Although there is a drop in performance around 50,000 time steps, it gradually improves, reaching a mean reward of 5. In the late phase, the agent appears to stabilize, oscillating between a mean reward of 4 and 5. For the mean episode length, there is variance in the early phase, jumping between 25 and 60 steps. In the middle and later phases, it stabilizes and improves, staying around 60 to 80 steps.

From the experiments, we can see that we couldn't simply take the hyper parameters that worked best for individual DQNs and combine them together. The noisy/categorical DQN behaves differently than the noisy DQN or categorical DQN alone. Because of this difference in behavior, the hyper parameters needed to be adjusted accordingly.

IV. Literature Review

The goal of our project is to enhance the performance of an agent playing SeaQuest by building upon the baseline Deep Q-Network (DQN). Specifically, we aim to modify the reward function to predict a distribution of future rewards rather than a single value and incorporate noisy layers to improve exploration. These enhancements will address limitations in the baseline DQN, such as its fixed epsilon-greedy exploration

strategy, and may be tested in a combined variant to leverage both improvements.

The game Sea Quest appears as a simple Atari game but has complexity for an agent to decide on short term rewards or long-term rewards depending on state-action pair. An agent could go for a short-term reward or wait and take a risk for a long-term reward. For example, the agent may shoot enemies to get points but rescuing divers and resurfacing six divers will be a larger reward that takes longer. When to resurface also matters. When the agent resurfaces, they get points for each of the six divers and for how much oxygen is left in the tank. So, while the agent may want to resurface as soon as they get six divers, they may want to time it to get six divers and have a large amount of oxygen left to get the bigger reward. The agent also must resurface even if the agent does not have six divers to refill the oxygen tank and continue playing the game without losing a life.

Sparse reward is an issue in the game Sea Quest. In the game, the agent must collect six divers then resurface to collect a reward. Initially collecting a diver has no benefits. This is discussed in Curiosity-driven Exploration by Self-supervised Prediction where when a reward is not immediately given for an action, an agent can struggle to correlate an action with an eventual reward. The paper also says, "Hoping to stumble into a goal state by chance (i.e. random exploration) is likely to be futile for all but the simplest of environments". So, hoping an agent can get a reward from random exploration is nearly impossible to happen.

The Intrinsic Curiosity Module has potential to enhance the agent's exploration in our game environment Sea Quest by encouraging the agent to explore unseen states. For example, the agent in SeaQuest will not realize that oxygen level is important until the agent dies from no oxygen. How long it would take the agent to survive until it reaches this point would take a long period of training to perform well enough to survive that long. The result of the paper shows that ICM can drive exploration of agents in environments with sparse rewards. However, the value ICM brings to our project depends on how we balance extrinsic rewards, the rewards needed for the agent to play the game well, and the intrinsic rewards, the rewards needed for the agent to explore unfamiliar states.

Intrinsic Curiosity Module has only been tested with games DOOM and Super Mario Bros. These games are more complex than most Atari games, but it is an open question to how effective it will be in our Atari game environment. However, the research paper provides valuable information about the issue of sparse rewards and how it handles the issue. For this paper I understand that it is important to explore and use methods that will deepen our agent's exploration.

Categorical DQN would be beneficial for helping agents learn short and long term rewards. C51 focuses on an agent learning distribution of returns for taking an action instead of one reward for one action. In the baseline DQN the value of an action is represented as the average of all possible future rewards for that action. This does not capture rewards that can vary in value. For example, when the agent resurfaces with six divers, the agent gets points for each of the six divers and also the amount of oxygen left. The amount of oxygen left creates variation in reward. Categorical DQN learns distribution of returns by using atoms.

Also, C51 DQN is shown to perform very well at the game Sea Quest, which is the environment we are working on for the project. C51 DQN ability to handle sparse rewards will improve the performance of the agent over our baseline making it a suitable choice for our project goals.

From the research paper, Revisiting the Arcade Learning Environment, we understand the importance of stochastically to prevent the agent from over fitting and improve generalization performance. While the paper already goes over ways that stochastic is added to the environment, it mentions that adding stochastic in other aspects of reinforcement learning would be beneficial. With this in mind, researching other ways to add stochastic has led us to Noisy DQN. Noisy DQN adds stochastic into the agent's action selection. Introducing noise into the agent's action selection process will allow the agent to explore the game deeper. The agent will possibly perform actions and reach states that were not possible to encounter in a fixed exploration schedule or a more deterministic less stochastic environment. This will help with our agent learning in the game Sea Quest. For example, early in training the agent does not know anything about the game rules. With epsilon greedy exploration the agent may explore these states but as training progresses and the exploration decreases, the agent may not have explored all states. It would be less likely in later training to learn anything new. With noisy layers, the agent will continue to learn from new states in the later training period if the agent has not already experienced these states.

In the Noisy Nets for Exploration, the authors use three baselines DQN, dual clip variant of Dueling algorithm, and A3C. Noisy layers are added to the three baselines to evaluate their impact on exploration and performance. The authors say in theory noisy layers could be added to any variant of DQN. The research paper Rainbow: Combining Improvements in Deep Reinforcement Learning puts this into effect combining noisy layers with other variants of DQN. Categorical DQN and Noisy DQN are both variants of DQN that address different aspects of baseline DQN. Categorical focuses on predicting the distribution of future rewards while Noisy DQN focuses on dynamic exploration. Combining both Categorical DQN and Noisy DQN into a single variant is possible and could offer the benefits of both variants. This combined variant potentially will perform better than either variant by itself as seen in Rainbow DQN. The improvement of performance from combining variants adding their strength into one algorithm will help us on the project create a more powerful DQN that performs better at SeaQuest than only one of the variants.

Categorical DQN as talking about in "A Distributional Perspective on Reinforcement Learning" [1] is a variant of DQN that uses atoms as discrete points to show a distribution of future rewards instead of an average value for reward. Commonly, 51 atoms are used so for one action an agent learns a distribution over 51 possible values of a future reward

instead of one value for a future reward.

Two important aspects of Categorical DQN that help operate are Distributional Bellman Update and the loss function. In baseline DQN, the bellman equation is used to find the value of future reward for a state-action pair. C51 DQN uses Distributional Bellman Update that instead of returning the mean of future rewards it returns distribution over all possible rewards. The distribution is represented by atoms that are the discrete values in distribution.

The loss function is used for when the agent moves to the next state, and it needs to update the distribution of possible future rewards for the new state-action pair. It measures the difference between the new distribution and the old distribution. This difference helps to ensure the new distribution is reflected in the atoms. So, the loss function ensures the updating of the atoms to the new distribution is smooth and there are no errors. The paper goes over the use of the Wasserstein metric for theoretical analysis of convergence and distribution. However for training or practical purposes, the cross entropy loss is used instead because it is more efficient.

The result of categorical DQN is that it performs well with games with sparse rewards such as Venture or Private Eyes. These games can be challenging because rewards can be infrequent, delayed, or take a long time to reach. With C51 DQN an agent better handles uncertainty and variability in rewards.

Noisy DQN as talked about in "Noisy Networks for Exploration" [2] focuses on replacing the epsilon-greedy algorithm used in baseline DQN and using exploration with adding noise into the network's weights and biases. This change is applied to the fully connected layers of the network, transforming weights and biases into random variables. These modified fully connected layers are referred to as noisy layers.

As a result, more stochastic is introduced both during training and action selection by making the q values stochastic. This approach leads to improving performance in games that require deep exploration or having sparse rewards.

In the epsilon-greedy algorithm the agent has a fixed exploration schedule during training. The agent starts with high exploration at the beginning but as training progresses the exploration rate decreases to a fixed value. This schedule is static and does not change regardless of game or agent's learning progress. This is different from what is seen from Noisy Net. The noise added to the agent's neural network results in a dynamic exploration schedule. At the beginning of training, the noise level is high prompting the agent to explore. However, depending on the game noise may decrease during the training process as the agent learns or it remains high throughout the entire training time. This shows that Noisy Nets adapts noise levels based on the game and the agent's progress in learning.

The Rainbow DQN as talked about in "Rainbow: Combining Improvements in Deep Reinforcement Learning" [3] is a variant of DQN that takes six existing variants of DQN and combines them together into one DQN as it is called Rainbow. The paper goes over how each variant on its own performs then how when combined forming Rainbow performs better. The paper also shows when removing one of the variants from Rainbow how the performance is impacted. The two most important components that helped Rainbow perform well were Prioritized Replay and Multi Step DQN. Distributional and Noisy DQN also contribute well to Rainbow's performance but not as much as Prioritized and Multi Step. The dueling network and Double Q learning had the least significant impact on improving performance but there was still some increase in better performance even by a little.

The paper, "Revisiting the Arcade Learning Environment: Evaluation Protocols and Open Problems for General Agents" [4] talks about the purpose and use of Arcade Learning Environment or ALE. It is a tool used to train and evaluate reinforcement learning agents in Atari games. It is useful because it is the standardized foundation in Reinforcement learning. A section of the paper talks about deterministic and stochastic elements in ALE. Atari games by nature follow deterministic rules. This means that the same action and situation will always have the same result. ALE adds stochastic or randomness to the games by using techniques like sticky actions and frame skips. Sticky action is a technique where the previous action is repeated in the next frame instead of the next action the agent wants to take. Frame skipping is the process of ALE only processing an agent's action at set intervals of frames. For example, if the agent's action is to go right, the right action is processed to say four frames before a new action is processed by ALE. The stochastic added is important to the agent's learning process. Without the randomness added agents will most likely over fit by memorizing a specific action sequence instead of generalizing. This means in a deterministic environment an agent will perform well in training but in testing will perform poorly. So, the stochastic added by ALE encourages the agent to adapt to changes and to generalize. This ensures that the agent does not memorize and can perform well to variations in the environment.

The paper, "Curiosity-driven Exploration by Self-supervised Prediction" [5] talks about the common issue of sparse rewards or rewards that take a long time to obtain in reinforcement learning. The solution is to use curiosity driven exploration or Intrinsic Curiosity Module (ICM). The algorithm works as usual in reinforcement learning with the agent in a current state it takes an action, and it moves into the next state. ICM works by taking the current state and the next state and encoding it into feature vectors. ICM will then use two models forward and inverse. The forward model will predict what the next state will be based on current state and action. The prediction error from the forward model is the curiosity of the agent. If the prediction error is high, then the intrinsic reward is high, and it encourages the agent to explore more. If the prediction error is low or accurate then the agent will receive little to no intrinsic reward. So, in summary the intrinsic reward is based on the prediction error of the forward model that encourages an agent to explore and find states that are hard to predict. The inverse model will predict what action was taken to move from the current state to the next state. The inverse model is

used to help learn the relationship between actions and their effect when transitioning to a new state. The reward given to the agent is a mix between extrinsic and intrinsic reward. Extrinsic reward is the usual reward from the environment such as score from Atari games. Intrinsic reward is new and is created specifically for the ICM. The behavior of the agent will depend on how much extrinsic and intrinsic reward it receives. If an agent is receiving little to no reward because it is sparse so the extrinsic reward is low, then the agent will focus on exploration and maximizing intrinsic reward. If an agent is receiving high value in extrinsic reward, then the agent will focus less on exploration.

From the paper, when an agent is trained with only curiosity and no extrinsic reward in the game Super Mario Bros it can perform effectively if using level 1 as pre training when going into level 2 and level 3. In level 3 the agent gets stuck at a difficult part of the level and performance declines.

Instead, a better method used was having an agent pre-trained only using curiosity then during training adding extrinsic reward. This resulted in the agent learning more quickly and performing better.

Dynamic Frame Skip Deep Q-Network (DFDQN) as talked about in the paper "Dynamic Frame skip Deep Q Network" [6] is an extension to DQN that dynamically optimizes the frame skip rate to improve performance in Atari games. The frame skip rate determines how many frames the agent skips between actions. In the traditional DQN algorithm the frame skip rate is a fixed value which can lead to inefficiencies and potentially be quite bad depending on the game it is being implemented on. This new interpretation basically makes it an adaptive skip rate depending on the agent's current state which can make it more flexible to the game environment. To do this they have a second neural network where the whole purpose is to predict the best and optimal frame skip rate. This has shown many improvements particularly in where fast reactions in games are needed. Overall, it saves computation time and resources while either maintaining or improving performance depending on the environment or games that it is being implemented on.

One of the considered methods was Deep Q Learning to where a key paper in the field, "Playing Atari with Deep Reinforcement Learning" [7] covers the use of Deep Q-Networks (DQN) in reference to Atari games. The main takeaway is that is works well with such games and shows how a Reinforcement Learning (RL) could learn complex tasks with high-dimensional inputs like raw pixel data using deep convolutional neural networks (CNN) for function approximation. The DQN architecture has multiple convolutional layers followed by fully connected layers which process the frames and learn the spatial and temporal features needed to make the decisions.

Many RL algorithms just estimate the action-value function with the Bellman equation as an iterative update which can be seen as impractical in most cases because the function is estimated separately for each sequence without generalization. In contrast, approximate the Q-value function Q(s,a) where this represents the expected cumulative rewards while taking a given action, a, in the state, s.

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}\left[(y_i - Q(s,a;\theta_i))^2\right] \tag{1}$$

The equation seen above is the loss function for DQN. The variable y_i is the target value as represented in the equation and shown below.

$$y_i = r + \gamma \max_{a'} Q(s',a';\theta^-) \tag{2}$$

The second part of the equation $Q(s',a';\theta^-)$ is the predicted Q-value and then the whole thing is squared to find the error in such a way that makes all the outputs non-negative and makes the larger errors more intense.

In the overall loss function equation $\mathbb{E}_{s,a\sim\rho(\cdot)}$ is the expectation over the state (s) and action (a) pairs to where they are sampled over the distribution of p(.).

Overall this loss function helps the DQN algorithm to minimize the error between the predicted Q values and the target Q values over time and by doing so it can learn to approximate the optimization Q value function. One of the main mechanisms in DQN that help stabilize the learning process, experience replay and target network. Experience replay stores past experiences in a replay buffer instead of sequentially which could be more unstable.

For a series of 7 Atari games tested (B. Rider, Breakout, Enduro, Pong, Q*bert, Seaquest, and S. Invaders) against different agents (Random, SARSA, DQN, Human) it was shown that DQN performs the best in every game by a large margin.

The paper, "Dynamic Action Repetition for Deep Reinforcement Learning" [8] covered Dynamic Action Repetition (DAR) as a method for improving the efficiency of DRL by the dynamic adjustment of the repetition of actions. Typically you would see the agent execute one action per time step which can be bad in situations where repetitive actions could give the same outcome. The method outlined in the paper allows the algorithm to learn the optimal action reaction rate so that it can repeat an action multiple times when needed which can accelerate the learning process and lower the computational cost. To implement this method you would need a second neural network to predict the optimal action repetition rate and then adapt that action frequency to the demands of the environment in a dynamic manner.

The paper, "Deep Reinforcement Learning with Double Q-learning" [9] covered Double Q-Learning reduces the overestimation bias in the Q-value estimates. Standardly, the same Q-network is used to select/evaluate actions which leads to being overly optimistic with the value estimates. The new method decouples the action selection from the action evaluation by having one network select the action and the other estimates the value. This is done by having the main network select the action with the highest value in the next state and the target network to evaluate that action value. This helps with overestimates of action values and helps the stability and performance of DQN in many Atari games.

The paper, "Prioritized Experience Replay" [10] covered Prioritized Experience Replay (PER) enhances the performance of the standard experience replay typically found in DQN. Normally the translations from state to action to reward and so on are sampled uniformly from a replay buffer to where it treats them all as equally important. However this method assigns higher sampling priority to transitions that have a higher temporal-difference (TD) errors which show a greater learning opportunity. This helps with the prioritization and helps the agent learn better overall.

V. Future Work and Conclusions

From our experiments with our categorical/noisy DQN variant we were able to achieve good performance, but we found that training stability is heavily dependent on careful hyper parameter tuning. Also the given time frame of 200,000 time steps is too short to show our variants performing better than the baseline DQN. Our time frame was able to showcase that the variants can learn over time and remain stable, but in the future, we would need to increase training time for our variants to reach their full potential in performance and surpass the baseline.

For future work we would focus on adding additional components from Double DQN, Dynamic Frame Skip DQN that were ran successfully along with ones that weren't fully tested yet like Dynamic Action Repetition, Rainbow, and Prioritized Experience Replay DQN. By testing more with the hyper parameters and seeing the results as well as combining relevant methods the stability and performance could increase significantly.

Rainbow DQN, Double DQN, Dynamic Skip Frame DQN, which successfully integrated both noisy networks and categorical DQN along with other improvements like prioritized experience replay and multi step learning. By adding these components from Rainbow DQN into our model, we believe we could improve the stability of training and make the model less sensitive to hyper parameter choices, while maintaining the benefits we observed from the noisy/categorical combination. Additionally, increasing the training time would allow our variants to demonstrate their improvements over the baseline DQN.

References

[1] Bellemare, M. G., Dabney, W., & Munos, R. (2017). A distributional perspective on reinforcement learning. arXiv. https://arxiv.org/abs/1707.06887

[2] Fortunato, M., Azar, M. G., Piot, B., Menick, J., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., Blundell, C., & Legg, S. (2017). Noisy networks for exploration. arXiv. https://arxiv.org/abs/1706.10295

[3] Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M. G., & Silver, D. (2017). Rainbow: Combining improvements in deep reinforcement learning. arXiv. https://arxiv.org/abs/1710.02298

[4] Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M. J., & Bowling, M. (2017). Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. arXiv. https://arxiv.org/abs/1709.06009

[5] Pathak, D., Agrawal, P., Efros, A. A., & Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction. arXiv. https://arxiv.org/abs/1705.05363

[6] A. Srinivas, S. Sharma, and B. Ravindran, "Dynamic Frame skip Deep Q Network," arXiv.org, 2016. Available: https://arxiv.org/pdf/1605.05365

[7] V. Mnih et al., "Playing Atari with Deep Reinforcement Learning," Dec. 2013. Available: https://arxiv.org/pdf/1312.5602

[8] A. Lakshminarayanan, S. Sharma, and B. Ravindran, "Dynamic Action Repetition for Deep Reinforcement Learning," Proceedings of the AAAI Conference on Artificial Intelligence, vol. 31, no. 1, Feb. 2017, doi: https://doi.org/10.1609/aaai.v31i1.10918.

[9] H. Van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-learning." Available: https://arxiv.org/pdf/1509.06461

[10] T. Schaul, J. Quan, I. Antonoglou, D. Silver, and G. Deepmind, "Published as a conference paper at ICLR 2016 PRIORITIZED EXPERIENCE REPLAY." Available: https://arxiv.org/pdf/1511.05952