# Space Invaders and Reinforcement Learning: A Comparison of Different Models

Benjamin Winslett

***Abstract*—Reinforcement learning is diverse enough to apply to many different fields of study, one of which is the Atari video game Space Invaders. There is a need to expand upon existing work already done for the issue of determining the best model configuration for obtaining the highest score possible in the game. This work expands upon previous work done to partially solve the question. Different neural network models were created, equalized as best as possible, and competed against one another to determine which model best plays the Space Invaders game. These networks include Advantage Actor Critic (A2C), Dueling Double Deep Q Networks (DDQN), Deep Q Networks (DQN), Monte Carlo Policy Gradient (PG), Proximal Policy Gradients (PPO), and Rainbow. To compare these networks, different metrics were collected during their separate runs. These metrics include time to execute training, highest score achieved both before and after training, and others. These studies present an optimal network configuration to best solve the Atari Space Invader game problem. It determined which model trained fastest and achieved the highest score over the others.**

***Key Words*—OpenAi Gym, Atari, SpaceInvaders, Reinforcement Learning**

## I. Introduction

The concept of reinforcement learning begs for the chance to be applied to video games and to be used to create intelligent agents to surpass even the most skilled of human players. This has been done since reinforcement learning was first introduce and will continue to do so into the future. Unfortunately there is not a one size fits all network model configuration, so each game will require a different configuration to achieve the best results.

The goal of this paper is to thoroughly ascertain the optimal neural network model configuration, coupled with the appropriate hyper parameters to efficiently and effectively play the Atari Space Invaders video game to achieve as high a score as possible. This is accomplished using work done by Deepanshu Tyagi (Tyagi [2]) and adding and modifying his work to gather metadata for each of the models explored. That metadata will be explored and compared to determine which configuration is best.

From that determination, it is proposed that hyper parameter data points be tweaked and modified until the highest score possible is achieved, or time expires. Thus determining not only the best model configuration for the game, but also what hyper-parameter values are best to achieve the optimal results.

## II. Methods

The environment used within this study was the Open AI Gym environment for the Atari Space Invaders video game (Farama-Foundation [1]). A screen capture of the environment can be seen in figure 1.

Fig. 1: Open AI Gym Environment

The goal of the game is pilot the spacecraft seen at the bottom of the environment away from enemy fire while simultaneously firing your own projectiles to destroy the enemy invading ships. The more you

destroy, the more points you achieve. There are also ”boss” ships that appear towards the top of the environment and are worth more points. You have 3 lives to get as high a score as possible.

The environment has a discrete action space (Table I), and its observation space is that of the RGB image displayed to human players (Figure 1)

| Num | Action |
|---|---|
| 0 | NOOP |
| 1 | FIRE |
| 2 | RIGHT |
| 3 | LEFT |
| 4 | RIGHTFIRE |
| 5 | LEFTFIRE |

TABLE I: Space Invaders Action Space

The models used by Tyagi (Tyagi [2]) included the following:

- Advantage Actor Critic (A2C)
- Dueling Double Deep Q Networks (DDQN)
- Deep Q Networks (DQN)
- Monte Carlo Policy Gradient (PG)
- Proximal Policy Gradients (PPO)
- Rainbow

The structure of the algorithms used was the same throughout all six model configurations. It first created the environment (Farama-Foundation [1]), then created the agent, and then trained the agent. After training was complete, a graph was shown, demonstrating the scores achieved across each of the episodes.

During the training, before each frame, the image captured is put through a pre-processing that reduces the size of the frame for quicker processing (Figure 2). The frames are then stacked and handed off to the model for training.

From his work I generalized the code he created to allow for a more equal basis to compare the models. I then wrapped each algorithm to time them and to output the maximum score achieved for that training session.

I ran the DQN network for 1,000 iterations, finding that the results were very subpar and did not show any formidable clues as to whether the network was learning to play the game. These steps were repeated for 5,000 and then 10,000 iterations, both of which bred similar results. See figure 3.

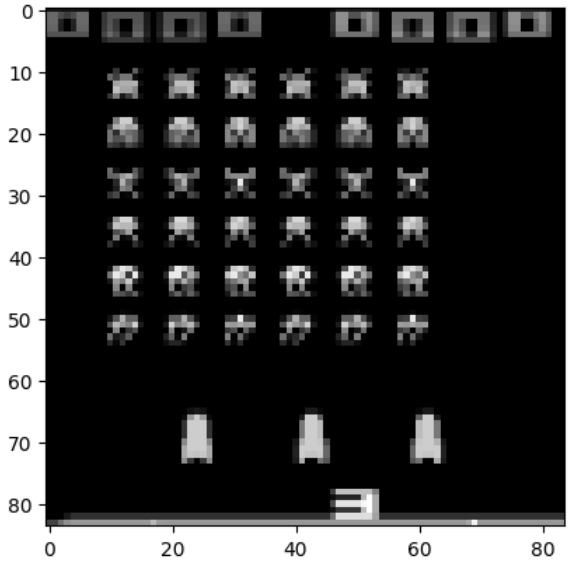


Fig. 2: Pre-processing Frame

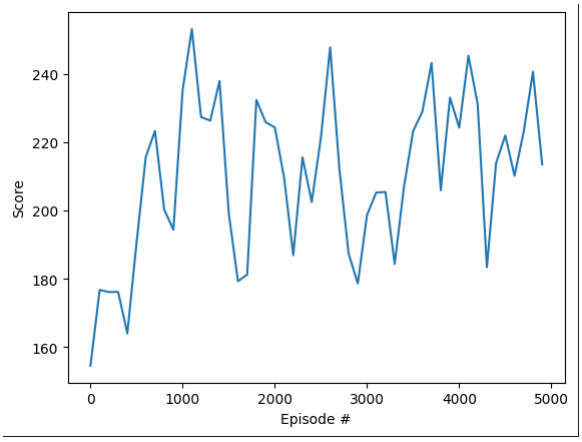


Fig. 3: DQN - 5,000 Episodes - Initial

Watching the smart agent play the game made it abundantly clear that there was a flaw in the logic. Death was not being penalized, and the agent disregarded any and all enemy fire. So I created a rewards wrapper to penalize death severely in an attempt to get the agent to avoid the enemy fire. Immediately it showed improvement to the DQN network and it was more apparent that it was indeed learning. See figure 4.

I took this wrapper and applied it to all 6 models and then ran each network for 1,000 and 5,000 iterations, noting down the execution time and max score achieved.

These simulations were ran on my home com-

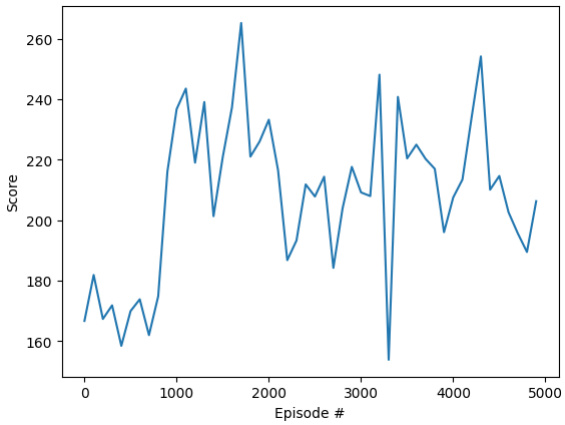


Fig. 4: DQN - 5,000 Episodes - Death Penalty

puter that has the following specifications:

- Processor: AMD Ryzen 7 5800X 8-Core Processor (16 CPUs), 3.8GHz
- RAM: 32 GB
- Storage: 1 TB Solid State Drive
- GPU: NVIDIA GeForce RTX 3070 Ti with 8 GB of dedicated VRAM and approximately 16 GB of shared memory

Upon completion of this initial study, I then took the best network and expanded upon and refined it in an attempt to get the agent to learn more, and achieve the best score possible.

## III. Results

It was quickly apparent that if by looking solely at the max score achieved, each agent performed relatively equally around the max score of 760, plus or minus about a 50 points for the 1,000 episodes. See table II for these values.

| Model | Execution Time (Seconds) | Max Score |
|---|---|---|
| A2C | 2662 | 705 |
| DDQN | 979 | 820 |
| DQN | 5802 | 785 |
| PG | 1515 | 690 |
| PPO | 2486 | 775 |
| Rainbow | 1352 | 800 |

TABLE II: Results - 1000 Episodes

Then, for the 5,000 episode runs, it showed similar results of an average max score of 870, plus or minus 50. See table III for these values as well. (Winslett [3])

| Model | Execution Time (Seconds) | Max Score |
|---|---|---|
| A2C | 12455 | 825 |
| DDQN | 4336 | 810 |
| DQN | 23762 | 955 |
| PG | 9019 | 855 |
| PPO | 10426 | 960 |
| Rainbow | 4279 | 825 |

TABLE III: Results - 5000 Episodes

Including time achieved and average score charts in the overall assessments painted a different picture. One that showed that the DDQN and Rainbow models were far quicker and showed a cleaner upward trend to the achieved average scores.

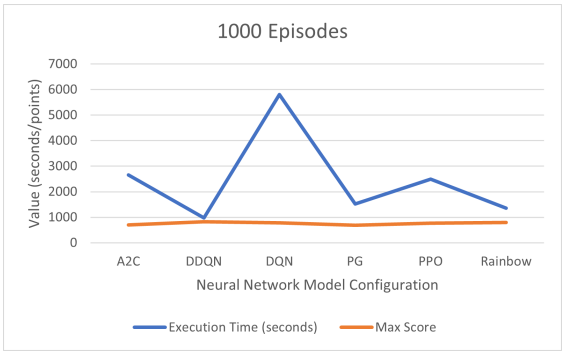


Fig. 5: Results Graph - 1000 Episodes

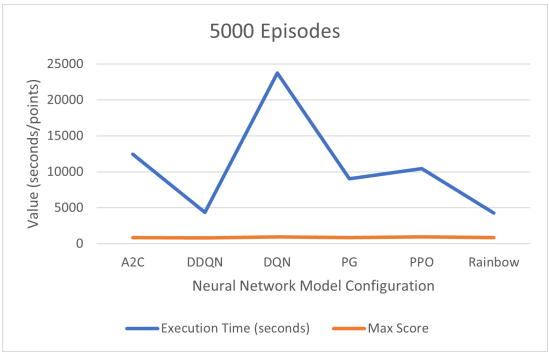


Fig. 6: Results Graph - 5000 Episodes

As seen in both figure 5 and figure 6, the DDQN and Rainbow models were completed in minimal time, therefore answering our baseline question of which model is best for the Atari Space Invaders video game. From here I decided to take at least one of these models, fine tuning the hyper parameters and running it for more episodes in an attempt to have the agent learn to achieve a max score of 5,000.

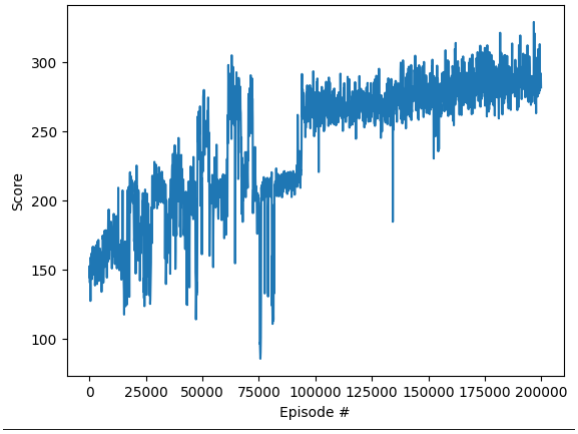


Fig. 7: DDQN - 200,000 Episodes - Slower Learning Rate Decay

Figure 7 shows the results of such an attempt. Not only did I run it for 200,000 episodes, I also modified the learning rate decay to allow for learning during all of the episodes, rather than what would have been the first 10 percent. The end results showed an upward trend that gives the impression that this could be fine tuned further and subsequently ran for much longer to achieve the desired maximum score.

## IV. Summary

Preexisting code was built upon and measured, then reworked and remeasured. The results showed 2 clear winners to the initial question of which configuration was the best for optimally playing the Atari video game Space Invaders. From that determination, further experimentation was performed to determine better hyper-parameters to allow for achieving a higher score.

## V. Conclusions

The DDQN and Rainbow model configurations were deemed the best of all the models due to their quicker computation times and smoother upward learning curves towards higher scores. The DDQN model configuration specifically was taken one step further, adjusting the learning rate decay and then ran for 200,000 episodes. The results showed promise in that its learning continued to trend upward. With more time and fine-tuning it is likely that the DDQN could achieve the desired score and be deemed the best model for the Space Invaders game.

## Acknowledgment

Special thanks to Deepanshu Tyagi for his original work in developing code that was the basis for the work done in this study. His work saved me countless hours in implementing and refining these very same models.

## References

[1] Farama-Foundation. Space invaders - gym documentation. https://www.gymlibrary.dev/environments/atari/space_invaders/, 2022.

[2] Deepanshu Tyagi. Deep reinforcement learning. https://deepanshut041.github.io/Reinforcement-Learning/, 2020.

[3] Benjamin Winslett. Deep reinforcement learning. https://github.com/WinniCdM/Reinforcement-Learning, 2022.