Comparison of various DQN-based and A2C Approaches on SpaceInvaders

Edwin Meriaux
McGill University
edwin.meriaux@mail.mcgill.ca

David Deng
McGill University
zhiyun.deng@mail.mcgill.ca

Abstract

Exploring the state space is a central process to reinforcement learning. In simple games such as "ice-lake" and "cartpole," the state space can be simply discretized. However, in more complex cases such as Atari video games, tabular approaches are no longer enough. Q-learning and actor-critic algorithms can be combined with deep learning methods to learn effectively from games with a large state space. The individual pixels can be processed. This is the foundation of Deep Q Learning. In this project, we implement DQN, A2C, and their variants based on processing the frames of a classic Atari game, Space Invaders. Over the course of this paper, DQN and A2C will be compared for both accuracy and computational cost.

1 Introduction

The Atari arcade games have long been used as a benchmark to test RL programs. These games include one called "Space-Invaders". Developing an algorithm to beat that game is the fundamental goal of this paper. Two methods were implemented in Python. The first was Deep Q-Network (DQN) and the second is Advantage Actor Critic (A2C). Since the basic algorithm themselves do not necessarily work very well. Multiple different versions of the algorithms were developed. These versions contain computer vision frame reduction, replay buffer, and different neural networks. The main goal of the project is to partially reproduce results from [3] regarding its use of DQN-variant on Atari-related games and to partially replicate results of [5] regarding the use of A2C. This project enabled us to explore important concepts in RL, especially challenges related to learning in large state spaces and sparse rewards.

2 Background

2.1 DQN

Deep-Q-Network is fundamentally a combination of Deep Learning (DL) with Q-Learning. This method was pioneered in the early 2010s with two important papers: "Playing Atari with Deep Reinforcement Learning"[7] and "Human-level control through Deep Reinforcement Learning"[6]. These papers followed the initial development of Deep Neural Networks (DNN) in 2012 (see Figure 1). The DNN inside the basic RL Q-Learning base simplifies the state discretization process. DQN learns the action-value function with a neural network and takes care of the targeting of the space (see Figure2). DNN's ability to generalize to unseen states and to perform end-to-end learning of hierarchical representations made it superior to previous approaches of relying on hand-crafted features. Since DQN has no convergence guarantees unlike tabular Q-learning, DQN has a tendency to overestimate Q-values and can therefore become unstable. This can cause its results to even regress in score over time. There are many ways to overcome these issues. DQN has multiple variants that have been explored in [3]. Some of these variations we test in this paper take the basic DQN algorithm and add: a replay buffer, a simplified visual frame, and an increase in neural network size.

2.2 A2C

As another part of this project, we also created an A2C implementation for the game. A2C is a specific algorithm in the family of Actor-Critic methods. Advantage-actor-critic (A2C) makes use of the concept of "advantage", the difference between how good an action is and how good an average policy action is at a particular state. The advantage function serves as a dense reinforcing signal for sparse rewards and can have a lower variance compared to Monte Carlo estimates of returns. [2]. This allows the algorithm to compare how much better one action could be if used more. This has been demonstrated to be superior to the current average of all possible actions rewards. The advantage function can be approximated with TD error. A2C has an asynchronous variant, called Asynchronous Advantage Actor-critic (A3C), which was used as a baseline by [3] when evaluating DQN performance.

2.3 Space Invaders

Space Invaders is an Atari single-player game where the player has a ship and must kill all the "invaders" while dodging the bullets (see Figure 3). The player has 6 actions they can take: do nothing, shoot, move left, move right, move left and shoot, and move right and shoot. An important note is that while the "invaders" can shoot multiple bullets at the same time, the player can only shoot one at a time. The bullet must hit an enemy or leave the screen before it can shoot again. In this project, we used the version "ALE/SpaceInvaders-v5", which is a deterministic environment.

2.4 Gym and ALE

The Gym library contains the basic environment necessary for basic RL research. It has some basic games such as "cartpole" and "frozen lake". It simulates the necessary environment to alter the game state based on the action taken and to output the appropriate reward at each step. This Gym library is extended by the ALE (Arcade Learning Environment) library, which contains a framework specifically for developing agents for Atari 2600 games.[1]

3 Methodology

The two main metrics used over the course of this test were based on the two pieces of data collected. This first bit was the reward at each episode, and the second was the ten-episode moving average (average over the last ten episodes). We noted the max reward in any given episode and the max 10-episode moving average. The algorithms were compared based on these two metrics. This was a comparison between DQN, A3C, and the random set. Each set contains 5 runs of the algorithm averaged together. Each run is 250 episodes. The DQN algorithm contains multiple variants and all were compared against each other. The random set contains random actions at each timestep. This is the baseline. These metrics are similar to the ones used in papers from Deepmind ([4]), but their final scores are generally just the highest individual score, whereas we took the average across different runs.

3.1 DQN

The DQN algorithm implemented had multiple components. The first was the basic Q-learning algorithm. A one-step look ahead Temporal Difference (TD) Q learning system was implemented. Since this overall system was DQN, a Deep Neural Network had to be integrated to replace the normally tabular state space that a standard Q-Learning system would use. A 3-layered linear DNN was set up with the input being the frame state of the pixel (greyscale) and the output being the 6 possible actions of the player. The number of Neurons in the DNN was not consistent in all of the different variants of the DQN tested. The amount of video memory limits the size of the DNN. There were a few hyperparameters in the DQN algorithm. The first was the discount factor which was set to a value of 0.99. We further set the Learning Rate to be 0.001, and the number of training episodes to be 100. For each of the experiments of 250 episodes, roughly 100 episodes were mainly exploration (with a decaying epsilon) and the last 150 has epsilon set to 0.

There were 3 additional methods added to the DQN to improve accuracy. The first was an increase in the size of the Neural Network. The GPU being used had 10GB of video memory. The Space Invaders frame is 420x160 pixels. In the case, with all the video memory used, the number of neurons

per layer was the number of pixels multiplied by 40. In the case where only 4GB of video memory was used, the number of neurons is the number of pixels multiplied by 20. This demonstrates the comparison of how the sizes of the layers affects the outcome. The second additional method was a computer vision program to reduce frame sizes. The full frame for the Space Invaders game is very big (210x160), and the ship itself is only 7 pixels wide. Since the whole frame is not necessarily useful in action determination, a computer vision script was written to limit the world seen from the vertical part of the screen to just 40 pixels wide. The ship is the center of the 40 pixels (see Figure 4). This reduces the number of pixels being processed allowing the number of neurons in the DNN to be increased to the number of pixels (now 170x40) multiplied by 100. The final additional method is a replay buffer. A common strategy in RL is adding a Replay Buffer. This strategy takes well-performing episodes and retrains the DNN occasionally on those top episodes only. This is what is implemented in the code. Of the top episodes, one is randomly used to retrain the DNN every 10 episodes.

3.2 A2C

To implement A2C, we primarily sought to replicate the architecture from (Mnih, 2016)[5]. Specifically, we used a CNN network with two convolutional layers. The first has 16 filters of size 8 x 8 with stride 4. The second uses 32 filters of size 4 x 4 with stride 2. These layers are followed by a fully connected layer with 256 hidden units. All three layers were followed by ReLu.

Advantage actor-critic relies on two networks: an actor (policy) network and a critic (value) network. The three layers mentioned previously are shared between these two networks. The actor-network has a softmax layer outputting to the number of actions, and the critic network has a linear output layer that outputs a single value for the image state.

A single loss function combines sums of the actor loss and the critic loss, and a single optimizer optimizes them both simultaneously. Combining two different losses in the same loss function has its own challenges, since the policy losses and the value losses have different units and need to be scaled. There is little empirical literature on the correct scaling techniques, so we arbitrarily scaled the value loss to weigh half of the policy loss. In addition to these two loss components, we used entropy regularization (computing an entropy loss and adding it to an overall loss) to discourage the model from being overconfident in one action and to encourage exploration. A search was conducted for the best entropy regularization weight among 5 possible weights, eventually settling on 0.01. To estimate the advantage function, we used the n-step forward-return estimator approach, where $A^{\pi}_{\mathrm{NSTEP}}(s_t, a_t) \approx r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots + \gamma^n r_{t+n} + \gamma^{n+1}\widehat{V}^{\pi}(s_{t+n+1}) - \widehat{V}^{\pi}(s_t)$ and $\widehat{V}^{\pi}$ denotes the value function learned by the critic. We chose a discount factor of 1, and $n = 5$. The negative of the advantage function is incorporated into policy loss so as to maximize advantage under constraints. Other variants we have tried include using two different actor and critic networks that do not share parameters. Due to computational resource constraints, we did not run the A2C as long as the experiment in the reference paper demanded. Instead, we trained the agent on 250 episodes of the game and averaged the result across five runs. In addition to our own implementation, we used a reference A2C implementation on Stable-Baselines3[8]. We trained it on a similar Space Invader environment, SpaceInvaders-v0, for 100,000 frames. Since the environment is deterministic, we used probabilistic sticky actions to test the model's generalization by setting `repeat_action_probability` to 0.25.

4 Experiment Results and Analysis

The results show that the Basic DQN algorithm with expanded memory space, replay buffer, and the computer vision frame reduction code outperforms the rest of the DQN algorithm (see 8 and 7). The tables show this trend. While this is not very visible in the score graph, it is more visible in the moving average graph. The red line shows the overall average of the scores. All DQN variants beat the random policy baseline. One issue visible is that the training/exploration curve is not very evident in the case of the DQN training. This might simply be due to the fact that 100 training episodes are not sufficient to train this DQN. There are so many possible states that a longer training period is needed. In the comparison between the cases with small Neural Networks and larger Neural Networks, we conclude having more neurons are better. All the cases with larger layer size beat out the "smaller" NN counterparts. Examples of this in the Figure 8 would be "With CV and Replay" which scored around 300, but the case "With CV small NN and Replay" only scored 230 on average.

It can also be seen that "with CV" outperforms the "Basic DQN" algorithm showing that just the addition of the computer vision algorithm aids the system.

4.1 A2C performance

In the deterministic environment the agent operates in, an episode score of 285 can be obtained by always choosing action 1 (i.e. always firing). Incidentally, this is also the strategy learned by the our agents when trained for around 250 episodes. The rewards earned during training can be viewed in figure 9. Although it performs better than the random baseline, it does not learn in a quick and satisfactory manner. One of the most significant challenges for this project is the difficulty in distinguishing between a faulty model and a model that has simply not been trained for an adequate amount of time. After 250 episodes, in testing mode, the agent always obtains a score of 285 by choosing action 1 repeatedly. The behavior is identical for the Stable-Baselines3 implementation of A2C after 100,000 frames. The overall performance is not different enough from the random baseline to conclude that the agent is learning. When our custom agent was further trained to 2500 episodes, the reward score during testing did not increase beyond 285. In additional tests where the discount factor $\alpha = 0.98$, the reward score did not change.

Certain actions in Space Invaders do not cause a change in the state. For example, the no-op action does not alter the state. If the agent is inclined to choose that action, then it is unlikely to gather the feedback necessary to revise its own probability estimates.

4.2 Overall comparison in performance

Overall, using DQN with replay buffer gives higher scores compared to other DQN approaches we implemented and compared to A2C. However, it has a higher computational cost.

5 Potential improvements

The most straightforward way to improve the algorithms is by increasing compute availability. This means both stronger GPUs and more computing time on those GPUs. The current experiments for this report were run using an RTX 3080 and roughly took over 10hrs per algorithm. This does include the 5 runs of each algorithm. If more GPUs were available the experiments would be run for more than 250 episodes and this would give more exploration and exploitation time. This will allow the algorithm to improve better.

In the case of DQN, the results can be improved by adding a second DNN. This is the basis for Double Deep Q-Learning (DDQN). This allows the algorithm to prevent overestimation from DQN. This is done with two networks. With every action, one DNN estimates the action while the other DNN receives the update from that action. An improvement to the DNN could also be to add more layers to it. Instead of the 3 big layers used, more small layers could have been used instead. Instead of linear neural networks, convolutional neural networks could have been used instead.

In the case of the A2C algorithm, improvements include determining whether it can learn in the long run. Another eventual improvement to the A2C algorithm would be A3C. As explained earlier in the paper, A3C allows multiple agents to learn at the same time and share their data. This method uses more computing power because simultaneous agents are needed and therefore they each need the same computing power. To facilitate comparison to DQN, similar neural networks should have been used.

6 Conclusion

Overall, this project was able to implement DQN and a couple of variants of it to work in the game Space Invaders. A2c has not been verified to be successfully implemented. So, the results from this paper come mainly from the implementation of DQN. It was seen that the basic algorithm was easily able to beat out the random baseline results and the addition of more neurons to fully utilize the GPU, computer vision script to try to remove unnecessary pixels from being processed, and a replay buffer to improve training did improve the DQN algorithm and increase overall scores. In the future, more computing power could be used to better improve the system and increase training times, but given the limited amount of time and computing resources, the goal of the project (to train an RL algorithm to play Space Invaders) worked because the policy developed outperformed the random policy.

7 Attachments

This is the Youtube video for our presentation: https://www.youtube.com/watch?v=u4PmiA0-Jho

Also enclosed in our submission are three python files. The first two are IPYNB files for the A2C model (one is our model and the other is a sample A2C) The .py file is the DQN algorithm test with full GPU usage (10GB), frame reduction, and replay buffer.

References

[1] M. G. Bellemare et al. "The Arcade Learning Environment: An Evaluation Platform for General Agents". In: *Journal of Artificial Intelligence Research* 47 (June 2013), pp. 253–279.

[2] Laura Graesser and Wah Loon Keng. *Foundations of Deep Reinforcement Learning: Theory and Practice in Python*. 1st. Addison-Wesley Professional, 2019. ISBN: 0135172381.

[3] Matteo Hessel et al. *Rainbow: Combining Improvements in Deep Reinforcement Learning*. 2017. arXiv: 1710.02298 [cs.AI].

[4] Matteo Hessel et al. "Rainbow: Combining improvements in deep reinforcement learning". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1. 2018.

[5] Volodymyr Mnih et al. *Asynchronous Methods for Deep Reinforcement Learning*. 2016. arXiv: 1602.01783 [cs.LG].

[6] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (Feb. 2015), pp. 529–533. ISSN: 00280836. URL: http://dx.doi.org/10.1038/nature14236.

[7] Volodymyr Mnih et al. *Playing Atari with Deep Reinforcement Learning*. 2013. arXiv: 1312.5602 [cs.LG].

[8] Antonin Raffin et al. "Stable-Baselines3: Reliable Reinforcement Learning Implementations". In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8. URL: http://jmlr.org/papers/v22/20-1364.html.

8 Credit

1. Edwin Meriaux
   (a) DQN
      i. Initial Research
      ii. Code Implementation
      iii. Implement Replay Buffer for DQN
   (b) Computer Vision (done collaboratively)
      i. Location Determination of the "spaceship"
      ii. cutting out the unecessary parts of the frame
   (c) Report
      i. Explanation of DQN, the replay buffer, and Computer Vision in the report
      ii. Documentation of Results
      iii. Introduction Section of the Paper
      iv. Background in the Report (done collaboratively)
      v. Conclusion in the Report
   (d) Video
      i. Audio Recording
      ii. Formatting Video from fames
2. David Deng
   (a) A2c
      i. Initial Research
      ii. Code Implementation
   (b) Computer Vision (done collaboratively)
      i. Location Determination of the "spaceship"
      ii. cutting out the unecessary parts of the frame
   (c) Report
      i. Explanation of A2c and Computer Vision in the report
      ii. Documentation of Results
      iii. Background in the Report (done collaboratively)

iv. Abstract in the Report

(d) Video

i. Audio Recording

9 Appendix

9.1 Hardware Used

The hardware for this project was centered around the one Desktop where all the final tests were run. Its specs were:

- CPU: AMD 5900x (with 12 cores)
- Memory: 64GB 3600 MGHz (CL16)
- GPU: RTX 3080 (10Gb)

The GPU is specifically important for the sake of this experiment. Its video memory allows for Neural Networks (NN) with more layers and neurons than one with less memory. This will be the base test for the DQN (for example) because two sets of runs will be done. The first set will be using not more than 4Gb of the video memory and the other case will be with all 10Gb. The more memory is used the more layers and therefore more features can be learned. This could also lead to overfitting. The results will clarify this. The GPU is also important because it dictates the speed at which the computations in the video memory will be happening. This GPU is also made by NVIDIA therefore the computations can benefit from CUDA acceleration from Pytorch. If the GPU wasn't being used then the basic memory (not video memory) and CPU speeds would be important. But running GPU tasks on CPU is very inefficient because of the multiprocessing that CPUs lack. To put things into perspective the AMD 5900 has 12 cores and therefore 24 hyper cores (1 core has 2 hyper cores), but the NVIDIA RTX 3080 has 8,704 CUDA cores. There cannot be any competition between the two. Even though the CPU cores turn faster there are not enough.

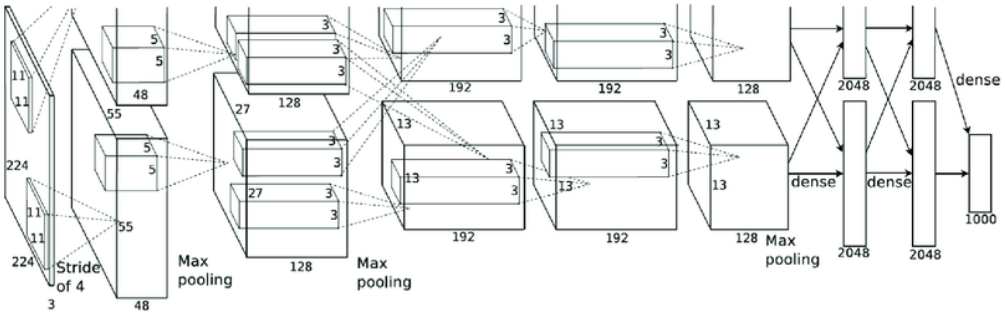

Figure 1: DNN Visual Implementation

$$Q(s_t, a) \leftarrow Q(s_t, a) + \alpha \left[r_{t+1} + \gamma \max_p Q(s_{t+1}, p) - Q(s_t, a) \right]$$

Figure 2: DQN Algorithm

Figure 3: Space Invaders

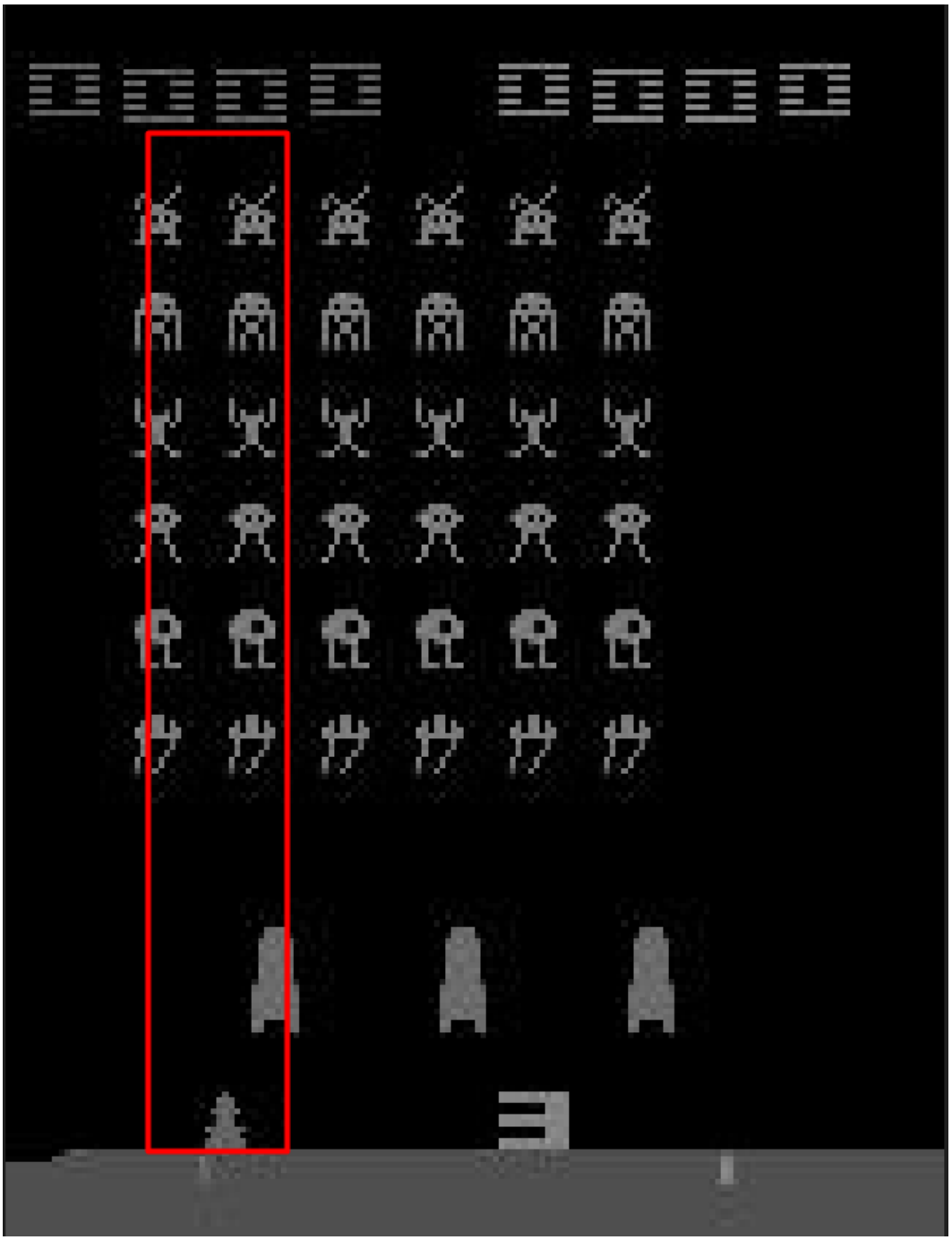

Figure 4: Space Invaders

$$\nabla_\theta J(\theta) \sim \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) A(s_t, a_t)$$

Figure 5: A2C Algorithm

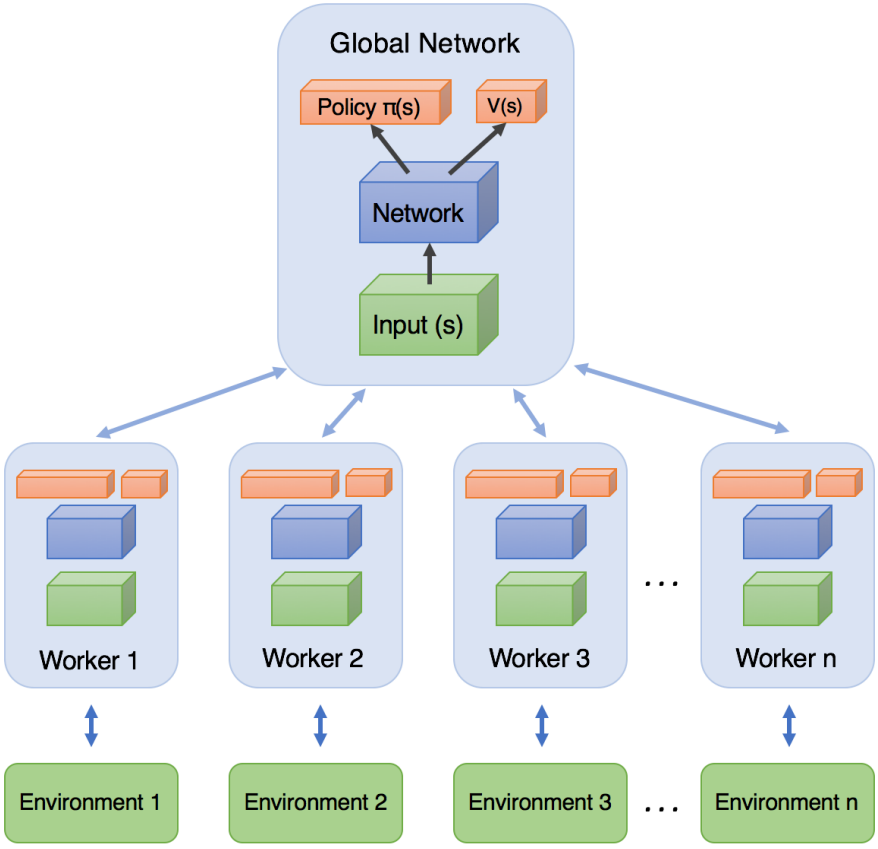

Figure 6: Visual Representation of A3C

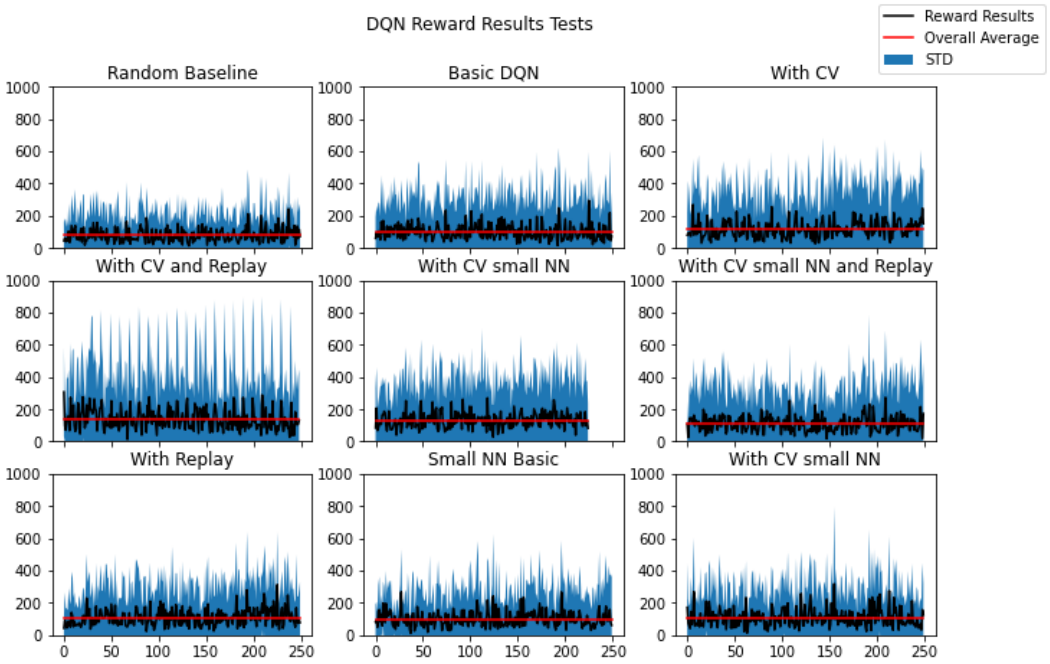

Figure 7: DQN Reward Results Plots (100 episodes of training and 150 of testing, X axis is episode count and Y axis is reward)

Figure 8: DQN Moving Average Plots (100 episodes of training and 150 of testing, X axis is episode count and Y axis is reward)

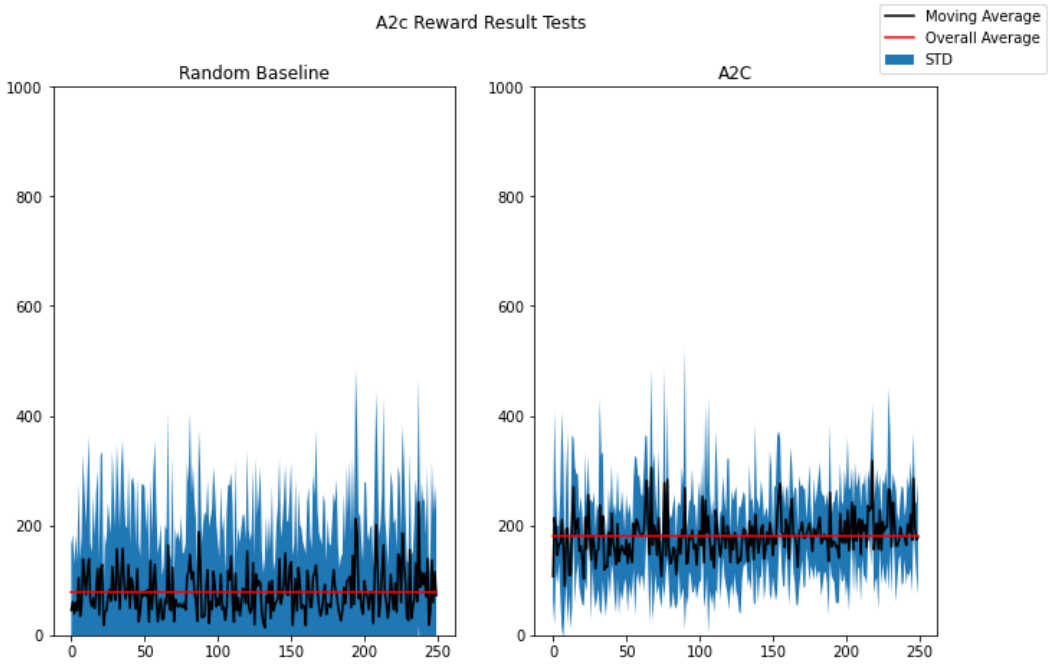

Figure 9: A2C Reward Results Plots (100 episodes of training and 150 of testing, X axis is episode count and Y axis is reward)

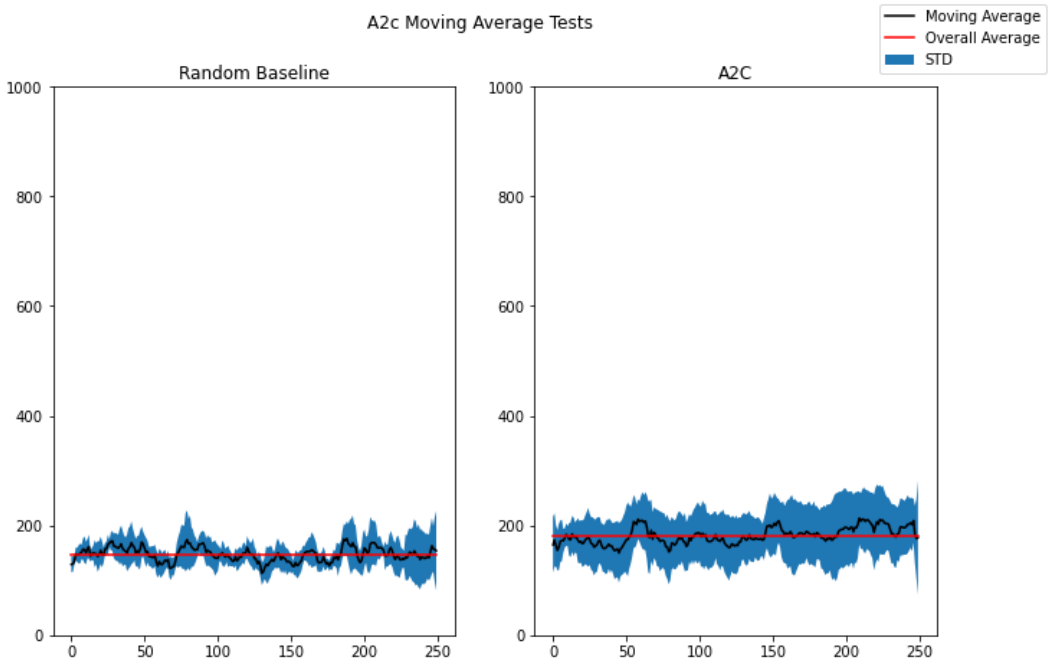

Figure 10: A2C Moving Average Plots (100 episodes of training and 150 of testing, X axis is episode count and Y axis is reward)