*Reinforcement Learning for Atari Breakout*

Name:David Kindlarski Email: djkindlarski@crimson.ua.edu

*Abstract*—Reinforcement learning (RL) is a rapidly growing field of research that has been forced into the forefront of society in recent months. In this study, I implement a RL method to the Atari game Breakout, using policy gradients to attempt convergence. This study was based on a previous one performed on the game Pong, which shares some key characteristics but differs in environment setup and end goal. The implementation included a two-layer policy network with ReLU activation and RMSProp optimization. Despite a slow start, adjustments to the hyperparameters resulted in consistent growth of the mean score as well as steady improvement, signifying convergence. This project offers insight into the world of reinforcement learning and implementation of methods taught at a high level to undergraduate students.

Keywords—reinforcement learning, stochastic gradient policy, neural networks

# Introduction (*Heading 1*)

The field of reinforcement learning is a very new focus of machine learning with a very high ceiling for its potential. In reinforcement learning, or RL, an agent interacts with a given environment to try and achieve a predetermined goal. The agent then receives rewards or penalties based on its decisions and adjusts its decision-making process accordingly. The agent’s objective is to maximize its rewards over time. Over thousands of iterations, it will slowly develop an optimal policy.

Of course, RL has its own set of challenges. To start, the lack of training data means the agent is completely clueless when it begins learning. The first hundreds or thousands of interactions may all be complete failures. Next, the process for finding success may involve delayed rewards, meaning that the agent will not always know what it did to be successful, just that it did something right. The agent will have to dissect everything it did before getting its reward to find out what is the true successful trait. Reinforcement learning also has issues with finding local maxima, or training for small success without knowing that an even better strategy is available. It is crucial for the agent to test a wide variety of strategies, good and bad, to identify the best policy possible.

Although it is only used in a video game here, reinforcement learning is a field with massive potential. It can solve a wide variety of problems and improve many industries. In the business world, reinforcement learning could optimize the supply chain systems of businesses large and small. In healthcare, RL could find the optimal recovery programs for patients as well as test the potential methods for operations. The manufacturing industry could be completely redefined through reinforcement learning, both through the advancements of robots and the innovation of new factory implementations.

In this project, the implementation of stochastic policy gradients is used to train an agent to play Breakout. Breakout is an ideal environment for the process due to its challenging gameplay, wide variety of strategies, and easily measurable goal for the agent to try and achieve. It also features some of the difficult objectives that RL models struggle with: delayed rewards and an inconsistent environment. It is a game well known for being easy to play, hard to master, and is the ideal environment for training a reinforcement learning model.

# Experimental Environment

## Selecting Breakout

Breakout was initially chosen from a list of over sixty Atari games listed on the OpenAI gym website. OpenAI gym is a collection of machine learning environments that are easily accessible to the average programmer. They provide hundreds of models in everything from classic video games, very early 2D training models, and even 3D multi-joint dynamic creations. Breakout was decided upon after initially learning about a Pong project built in a very similar fashion. Breakout has some of the same characteristics as Pong but brings enough unique traits to be a separate experience. It also has a level of sentimental value, being a game that frequented my childhood.

## Maintaining the Integrity of the Specifications

Identify applicable funding agency here. If none, delete this text box.

In most RL models, the environment is simplified to allow for faster, more reliable computations. The same applies here, but the identity of the game is not compromised. To increase processing speed and avoid extraneous errors, the pixels were scaled down and converted into black and white. The edges of the environment were also cropped until the final result of a 80x80 workspace was acquired. Even with all of this, the same game is still being simulated. The paddle is the agent, and the ball and bricks all work in the exact same manner as the original version. The agent also retains the ability to fire, a small yet critical additional action to increase the difficulty of the experiment.

# Background or Related Work

Reinforcement learning research has become increasingly common in recent years, especially with the advent of resources like OpenAI gym. One of the earlier experiments, which was also studied in the course, was Pong**[1]**. The Pong study was also performed using stochastic policy gradients and was a key inspiration for this paper. It focused on treating one of the paddles as the agent, and a very simple reward system of +/-1 depending on if the ball gets past the opponent or missed. The research uses Pong as a simple demonstration of the more complex system at hand, focusing on the policy network and the individual functions. Beyond that, there have been many studies focused on OpenAI gym and the Atari games it employs, whether they focus on implementing RL agents attempting to recreate the human brain**[2]**, use 3D models to simulate autonomous vehicles**[3]**, or even find superior alternatives to reinforcement learning**[4]**. Each of these experiments pushes forward reinforcement learning in a upward direction, promising rapid growth and development in the community. The human brain is already one of the most studied components of modern science, and the usage of machine learning can help rapidly advance the knowledge and understanding of its intricacies and operational procedures. Additionally, replicating the processes of the brain would result in massive growth in processing data as well as understanding the human body. Meanwhile, autonomous vehicles have been on the horizon for years, and with each new training model they become closer to a reality. Using 3D models is also a much safer alternative to real world testing, and additional focus has been placed on maneuvering situations where there is not prior GPS information or hazardous environments with few traditional datapoints. Finally, the potential growth of the machine learning industry is always welcomed, as each model has flaws and struggles that it must overcome when training. One specific alternative is the Non-Axiomatic Reasoning System, or NARS. NARS is a general-purpose cognitive reasoning network and has shown signs of superiority over RL in non-deterministic, diverse environments. These are crucial developments that will allow for rapid growth into studies that do not have as conclusive of goals and can expand the reach of machine learning into even more industries.

# Methods

For this project I decided to use a stochastic policy gradient method similar to the Pong project previously assigned. I felt most comfortable with this environment due to the aforementioned practice environment, and wanted to see if it could be replicated and implemented in a similar game. The policy gradient method is designed to directly optimize the policy. It does not use any predetermined models or outside training data. The policy implemented here is a very simple two-layer neural network. The neural network includes a single hidden layer and over 200 neurons, with the actual number fluctuating throughout testing. The ReLU activation function is implemented in order to include non-linearity. ReLU was chosen for its simplicity and generally quicker convergence rates. The input layer is an 80x80 matrix, the size of the environment after preprocessing. The output layer provided four different neurons, each corresponding to a potential action within the game.

For the gradient descent I used RMSProp, which is an adaptive gradient descent algorithm. It updates all the model parameters individually, adjusting the learning rate for each one to help seek convergence. I primarily had the learning rate set to 1e-3, although I did try out 1e-4 as well. My decay rate was set to 0.95, a good medium to encourage growth without too quickly forgetting previous iterations.

In training the process is quite simple. The agent would gain experience through “episodes”, sets of states, actions, and rewards. The gradients would be calculated throughout each episode, and at the end of a “batch” of episodes, which was usually 20 or 25 long, the model parameters would be updated. The discount factor was set to match the decay rate at 0.95, and the discount rewards were all normalized before gradients were computed. This leads directly into convergence, which was assessed on a running mean based on the rewards system. The current reward was valued at .01, while the previous reward was valued at .99. These were then combined to find the running mean. This created a measurable metric for convergence, which was then saved every 100 episodes.

All the chosen methods were purposely simple and efficient. The goal of this project was to create a working RL system, not to overcomplicate measures beyond my means. The two-layer neural network and policy gradient method are sufficient for learning game dynamics, but deeper neural networks would offer much more utility with superior results if implemented. I focused on creating what I could do best with my given environment and constraints, and focused on an objective that was measurable and attainable.

# Implementation or Results

Before actually running the process, I needed to make some adjustments to the framework to be able to actually handle Breakout. The initial inspiration for the project came from Pong, which has a simpler setup with fewer potential actions. As a result, all the functions had to be rewritten to accommodate for this additional data. This led to multiple rounds of debugging where some functions would still be based on one input or output with others being converted to four. Additionally, there were issues with preprocessing and getting the environment to the ideal sizing. Lots of trial and error occurred here, with a focus on removing all unnecessary information as well as always maintaining the integrity of Breakout.

For the actual implementation of my project, I focused on a straightforward, analytical approach. Beginning with values identical to the ones used in Pong from Pixels**[1]**, I began observing the growth of the agent. To start, the agent struggled with understanding the concept of Breakout. I would miss the ball time and time again, with the mean score being around 0.2 to start. Only after over a couple thousand iterations did the paddle begin consistently hitting the ball even once. From here the paddle struggled with understanding the physics of the ball, often hitting it at weird angles that would make recovery nearly impossible, and continuing the trend of low scoring games. The growth overall felt very slow, with a running mean of 2 points after 5,000 episodes. It was at this point that I stopped the first run and changed some of the hyperparameters. I decided that the decay rate and gamma were both too high, leading to a very conservative playing style with very little growth. These were both decreased to 0.95. Additionally, the batch size was too small, so I doubled from 10 to 20. After this I began running the process again. As hoped, the growth was quicker than the initial process, hitting the same benchmark of 2 points a thousand episodes sooner. However, a recurring issue was beginning to display itself with the agent finding a local maximum. It found that if it stuck to a corner of the playing area, it would have a higher chance of getting at least 1 point. As a result, it would become stuck against a wall, never leaving or exploring much further. This is understandable, due to the ball consistently going to one of the corners when it is initially fired. To fix this, I increased the batch size further to 25 and the hidden layer to 225. This is also a situation where I found patience was one of the best solutions, as the agent would eventually overcome this local maximum with enough time and iterations. The increase in neurons was possibly the most beneficial, as this was saw the largest increase in growth rate amongst all the changes. At these settings I found the most consistent and rapid growth, demonstrating an upward trend that indicated the reinforcement learning method was converging. Unfortunately, I did not save all the data as accurately as it possibly could have been, having several failed exports of data. As a result, my graph is based off written observations and general trends within the data. Also, it would eventually take over 20,000 iterations to have a mean of 9 points, which was impossible to replicate with my time and resource constraints.

# *Chart, line chart Description automatically generated*Discussion and Conclusion

Overall, this project has been a fascinating foray into the world of reinforcement learning and a very fun finishing touch on my time as a student. This is the last project I will get to work on as an undergraduate computer science student, and it means a lot to me to do something that I have been fascinated with for years. My only regret is that I did not have as much time as I wished to work on this project, but I hope to expand my horizons as I have more free time and continue experimenting with RL and other MLOC opportunities. Currently I am in the final round of interviews for an automation consultant position, but I believe that focuses more on current ML practices and not as much on research. I hope to continue learning more about this industry on my own, with my next project hopefully being “Adventure”, which is also available through OpenAI Gym. I hope to utilize some of the more complex methods we discussed in this course to master that game, and I also plan on diving further into our textbook and all its wonderful activities. This class has given me a unique opportunity to explore the most forefront research of my industry, and for that I am forever grateful. I hope to continue expanding my understanding of RL and MLOC going into the future, and I would like to thank Dr. Gong for offering this opportunity.

##### References

1. A. Karpathy, “Deep reinforcement learning: pong from pixels,” unpublished
2. I. Pivovarov, S. Shumsky, “MARTI-4, new model of human brain, considering neocortex and basal ganglia – learns to play atari game by reinforcement learning on a single cpu,” in Lecture Notes in Computer Science, vol. 13539, 2023.
3. S. Sivashangaran, A. Khairnar, A. Eskandarian, “AutoVRL: a high fidelity autonomous ground vehicle simulator for sim-to-real deep reinforcement learning,” unpublished
4. A. Beikmohammadi, S. Magnusson, “Comparing NARS and reinforcement learning: an analysis of ONA and Q-learning algorithms,” unpublished