**Reinforcement Learning To Play Space Invaders**

Aaron Bun, Rushad Daruwalla, Andres Torres

# **Introduction**

The primary goal of our project is to train an agent in playing Space Invaders through reinforcement learning. Given the various strategies applicable to this type of challenge, we have chosen to use deep learning methods. Specifically, we utilized a convolutional neural network (DQN) to analyze raw image data from a complex environment. Every frame is then captured by an image to be further preprocessed and fed to the model. So far, we’ve completed a majority of the project, which includes data preprocessing, model implementation, and the training loop. All we need to accomplish is training the model.

# **Problem Definition**

The machine learning problem which needs to be solved is reinforcement learning. We have the specific task of training an agent to play the Atari game Space Invaders using Deep Q Networks (DQN). The agent learns to make sequential decisions by interacting with an environment to maximize a cumulative reward. The DQN is a type of neural network used to approximate the Q-function, which then produces a value representing the cumulative reward for taking a particular action in a given state. The goal is to teach the agent to take actions in the game that lead to the highest possible reward over time. This is completely different from deciding whether performing an action was the best (like moving left or right). This would introduce human bias where we humans decide what the best move is in a current state. Our agent is specifically trained to maximize reward by achieving a high score. The agent learns by interaction with the environment, observing states, taking an action, and receiving a reward.

# **Data**

Due to the nature of reinforcement learning, we do not have a traditional data set. Instead our data comes from the environment itself in the form of raw grayscale images. Every frame played within our environment state is represented as a single image. The original size of the image is 210 x 160 pixels with value ranging from 0 - 255. However, because we are working with neural networks, this data from the environment can cost a lot to compute. We use a preprocessing step to reduce the complexity of the data. We first crop the image to only include the image area. Then we reduce the image size to only 84 x 84 pixels. Lastly we normalize the grayscale values by dividing each value by 255. Since the game constantly changes, our data changes from frame to frame and all the steps from above are repeated.

# **Methods**

DQNBasic is a keras implementation of the model described in DeepMind's paper "Playing Atari with Deep Reinforcement Learning". The input for the model is a numpy array of 4 images (in grayscale) cropped to be 84 x 84 pixels. The output is a numpy array of length 6, where each action's value is represented by one of the items in the array.

The training loop uses various techniques to train our agent to play the game. The first method we used was a replay buffer to store in memory experiences from the game itself. The replay buffer maintains a dataset of the current state, the actions taken, the received reward, and the resulting state. This buffer helps train the Q-function by randomly sampling (mini-batch) from past experiences. The random sampling contributes to more stable and efficient learning processes instead of feeding the Q-function with consecutive states. Our agent predicts which is the best action and then takes that action to update the replay buffer with its respective data of the environment.

The mini-batch is then used for back-propagation to update the weights of our model. We first need to calculate the loss from what our model predicted and the expected rewards from this random sampling. After calculating the loss of the function, we take the gradient of the loss and update our weights. The function below compares our Q-value prediction and the Q-target and uses gradient descent to update the weights of our Deep Q-Network to approximate our Q-values better.

TODO: epsilon strategy

TODO: reread methods section I probably missed some things

# 

read the .idea.md to get an idea what to write about in the experiments and conclusion section

**Experiments**

**Conclusion**