Reflection

Introduction

The goal of our project is to develop a deep reinforcement model to play the Atari 2000 River Raid. In recent years, AI has broken human records across a variety of games using reinforcement learning. Our motivation is to learn more about this field in a hands-on, yet fun manner. We plan to use the atari-py package to simulate the environment and interface with the game space. Our broad goal is to use a convolutional neural network to identify game elements and pass them to a reinforcement learning model, trying a few different architectures to compare performance. Some of our considerations will be to chain together sequences of game states to estimate movement and velocity of game elements, and how the algorithms can balance between greed v.s. longer term optimization.

Challenges

We made great efforts initially to set up the emulation environment. There are several different versions of the atari-py package, and finding one that could reliably work with our Python version etc. was challenging. We have frozen on using Python3.7 since that is the version on GCP's deep learning VM, with ale-py, tensorflow 2.8 and keras 2.8. Given the task complexity for the agent to learn, another challenge is our need of setting up a Deep Learning VM on GCP. Due to the limitation of computation resources, we need to thoroughly test our model locally before moving to training on GCP.

Current Model Insights

We incorporated a convolutional neural net (CNN) within the deep Q-network (DQN) model structure. The input frames of the game are of the shape (210, 160, 3). For better efficiency, we downsampled / resized the frames, converted RGB to grayscale, normalized, and concatenated k=4 frames at a time. After preprocessing, the shape of each input is (84, 84, 4). Our current model has 2 convolutional layers followed by 2 fully connected layers to compute action rewards in the deep Q-network. For an initial result of this architecture, we ran the games for 100 episodes, achieving average rewards at 773.95 with average time of 21.67 seconds per episode. The last five games have average rewards at 1534.6 and average episode training time of 16.0 seconds. (A score of 1500 is close to the performance of a human beginner)

Plan

In the next couple days, we plan to further optimize this model and run the models with more episodes of the game (5 to 10 k). The strategies we plan to try out include:

1. Experimenting on different k values for frame-skipping and Huber loss
2. Training on SARSA algorithm that utilizes a slightly different Q-value update rule
3. Replaying memory of a buffer of recent frames
4. Given enough time, we would also consider implementing the policy gradient model