CS205: High-Level Project Description

Rudra Barua
rudrabarua@college

Charlie Harrington
charlesharrington@g

Fiona Henry
fhenry@g

Michael Krumdick
mkrumdick@g

Deep Reinforcement Learning is a field of machine learning that deals with learning behaviors. In contrast to the standard supervised learning setup, these models will learn from "experience." The models, or agents, will interact with an environment and generate data which will then be used to further optimize the model. There is no pre-existing dataset, all the data needs to be generated by the model itself.

These algorithms are famously sample inefficient. Even learning simple tasks can require tens of millions of samples from an environment. Most works looking at improving the efficiency of these algorithms focuses on improving the sample complexity of the model. However, the bottleneck is often times in interacting with the environment.

The environment is whatever game or simulation that agent is trying to learn how to behave in. Typically, the models will be implemented using some hyper-optimized CUDA code that makes them incredibly efficient at processing data in batches. On the other hand, the environments can be pure python code. They will be a sequential program that receives the action from the agent, steps the environment forward a single time-step, and then returns some data. In order to quickly generate data for our model, these environments will be run in parallel to both handle and generate batches of data. This will normally be orchestrated via python's multiprocessing library, due to the issues with multi-threading in python.

One inspiration for our work is *Sample Factory: Egocentric 3D Control from Pixels at 100000 FPS with Asynchronous Reinforcement Learning*. This work analyzes potential increases in efficiency by mainly focusing on the throughput from the environment. Instead of taking the normal approach and scaling the total amount of compute across many different machines, they are able to make improvements in the wall clock times of these algorithms by maximizing the efficiency on a single machine.

Our idea for this project is to take an environment and see how far we can maximize the throughput. Ideally, we would like to do a process similar to the speed ups for matrix multiplication table: implementing a baseline and demonstrating how much all of the improvements speed up performance. We do not plan on trying to improve or modify existing reinforcement learning algorithms. Speeding up the time it takes to sample will unilaterally improve the wall clock performance. Meaning we can time a single learning process on the baseline environment and estimate how much we can speed that entire process up by analyzing how much it cuts down on the environment interaction time.