

# Machine Learning Engineer Nanodegree

## Capstone Proposal

### Proposal

I would like to develop an A3C-based agent successfully as described in [1](#). I plan to evaluate the algorithm's performance for Breakout, an ATARI game where baseline methods are Double DQN<sup>2</sup> with Prioritized Experience Replay<sup>3</sup>. It is true there's hardly anything novel, but for the sake of personal learning, I think this is a challenging, interesting problem.

### Domain Background

The field of Reinforcement Learning has recently seen breakthrough results by approximating Q action-value functions with Deep Neural Networks (DQN). DQN has been shown to beat expert human players on a number of the ATARI games by training only with raw visual inputs of the games and without any domain knowledge specific to the games given<sup>4</sup>. Despite DQN's promising results, it is known to be unstable in its vanilla form. There are three main causes to this. First the samples an agent experiences are highly correlated (non-iidness). Second, a small update to the Q action value function can lead to a large change to the data the agent experiences (non-stationarity). Third, the action values (Q) and the target (reward +  $\gamma \max Q$ ) are correlated, just like a cat chasing its own tail (non-stationarity). To remedy the instability, Experience Replay (EP) and the trick of using a separate target were developed, and achieved a stabilizing effect.

However, EP has several drawbacks. First, it requires more memory and computation because the agent must keep track of the storage of experiences and in the case of Prioritized EP, the task of updating the priorities of the samples can be computationally heavy. Second, EP forces us to do off-policy learning where on-policy learning algorithms like SARSA, actor-critic methods is impossible.

Asynchronous Advantage Actor Critic (A3C) is an algorithm proposed to solve the issues mentioned above. A3C is deemed outperforming DQN, in terms of the simplicity of implementation, the training time, and performance. Unlike DQN with EP that uses a single agent interacting with the environment, A3C runs multiple agents where each agent interacts, in parallel (asynchronously), with a different instance of the environment. This scheme has the effect of reducing the correlations between experience samples, and making the learning more stationary because the variety of experiences will increase. Practically, the parallel scheme allows the training time to be faster than in DQN, and the training can be performed on a single computer with a standard multi-core CPU.

### Problem Statement

The problem is to produce similar results in the A3C paper for Breakout, an ATARI game environment. Concretely, I hope the agent will learn to acquire over an arbitrary amount of 500 points.

### Datasets and Inputs

1. The environment: OpenAI's gym
  - <https://gym.openai.com/envs/Breakout-v0>
2. Existing implementations to reference:
  - <https://github.com/devsisters/DQN-tensorflow>
  - <https://dbobrenko.github.io/2016/11/03/async-deeprl.html>

### Solution Statement

The solution is an A3C algorithm. The A3C model will be trained on my local machine: Dell XPS 9560 with quad-core i7 CPU and NVIDIA GTX 1050 (4GB memory). The trained model will be evaluated on my machine whose result will be posted publicly on the OpenAI's scoreboard. 500 points is on par with some of the highest scores posted there.

## Benchmark Model

1. DQN with Experience Replay This will be trained on my local machine. Since training DQN successfully may take days, I will try to use Google Cloud or EC2 for training it. If possible, I may try to evaluate double DQN(D-DQN), D-DQN with prioritized experience replay, or dueling D-DQN.
2. CEM (Cross Entropy Method) This is an evolutionary algorithm that is considered a well-performing baseline. The implementation will share the implementations of [5](#) and [6](#).

## Evaluation Metrics

1. training time to achieve an appropriate score level (e.g. 100 points)
2. total average points per episode (moving average over 100 episodes) acquired by a successfully trained model.

## Project Design

I will follow the strategy of writing the final report from the beginning and keep adding details until the report is complete. Therefore the workflow will be similar to how one would go about reading the final report from one section to another.

1. related work I will read papers that feature relevant studies (6 referenced plus any more papers) to understand both the theoretical grounds and the technical details needed for implementation. The result
2. environment setup I will set up necessary dependencies to run an agent on an OpenAI gym environment. I will also acquire an account from OpenAI to be able to post results on their scoreboard.
3. system setup The system to implement will be A3C, DQN and CEM. The system will be written in Python 3.
4. training The training will be done on my local machine. If I conclude DQN is hard to train on my local machine, I will run it on Amazon EC2 or Google Cloud Compute Engine.
5. evaluation The evaluation will be done on my local machine where the evaluated metrics will be visualized in python-based plots.
6. reporting The end-result will be shown on Jupyter notebook. I will report findings and share areas of improvement for further work.