Project Proposal: Deep RL with Double Q-Learning

Jacob Heglund, Nitin Tangellamudi, Xinbo Wu, Jake Headings, Sahil Kumar

1. Paper Summary

We will be implementing the paper "Deep Reinforcement Learning with Double Q-learning" by Hado van Hasselt, Arthur Guez, and David Silver for our project this semester. Traditional deep Q-learning is known to overestimate the value of actions in reinforcement learning problems. These overestimates cause less-than-optimal policies to be generated by the algorithm. This paper shows that the addition of a second Q value to the Q-Learning architecture not only reduces overestimation of state values, but generates more-optimal policies. The two Q values are utilized in the following way: during each step of training, one of the two sets of parameters are randomly chosen to be updated at each timestep. One of the sets of parameters are then used to generate the policy, and the other set is used to evaluate the value of this policy. This architecture decouples the policy generation and the policy evaluation.

The Q-value is estimated using a deep neural network, with the input being a state and an output being an optimal action. Reinforcement learning and Q-Learning are fields where hyperparameter selection and value initialization has great importance on the way the model trains and behaves. This specific paper closely implements the algorithm and experiment found in *Human-level control through deep reinforcement learning* by Minh et al (2015), but with Double Q-Learning.

A standard method for evaluating the performance of Reinforcement Learning algorithms is to use a standardized environment. In the case of this paper, a series of Atari games were used. The experiment trains separate instances of identical neural network to play 49 Atari 2600 games. All networks are different instances of the same network, sharing the same structure and hyperparameters. Each game was trained for 200M frames on one node; this training took the approximately one week per game.

2. Plan for Implementation

Datasets

1. Our data does not come from a single formatted file, but uses the Arcade Learning Environment (ALE) (Bellemare et al., 2013). This framework has an implementation of an Atari 2600 emulator in order to output frames for each game. The paper uses this for the full game set of 57 different games, and the implementation used took about a week to train in full. Even with reducing the total amount of data used (the researchers used about 200 million frames total) it's still somewhat

unfeasible to properly train all of this with all the testing we're going to be doing as well. This is where we will be using OpenAI Gym, a framework similar to ALE, to select a smaller subset of games to easily play with our model, and continue refining our final program until we are comfortable submitting the entire program for training.

Plan for coding

1. Nitin: Build model in Pytorch and implement methods to optimize training times and convergence.
2. Sahil: Implement Forward and Backward Steps and document the mathematics of this for our final paper
3. Jacob: Graph metrics, incorporating the theory behind Q-Learning into our project
4. Jake, Xinbo(Jason): Implement environments

Computations

We believe each game should take anywhere from 20 to 50 hours to train. We will (try to) be using the same hyperparameters that the paper uses. The paper trains different instanced of the same network on 49 games. If we were to do this, it would take 980 to 2450 GPU hours to train all games.

Our hopes and dreams

We want to provide a unique contribution to the methods proposed in this project. These are possible ways we can contribute to the field:

- Split the 49 games into different subgroups based on the kind of game. For example, a maze-type game like Pacman and a spaceship game like Air Raid would belong to separate groups. We would initialize one neural network for each group. The network would be trained on one game from the group, and tested on the whole group. Then the game would be trained on a second game from the group, and then tested on the whole group again. This would continue until we train on the whole subgroup. The results should provide insight on transfer learning. (or we could find out how the algorithm "groups" games by which ones it performs similarly on?)
- Work on optimizing the training method in the paper by applying different methods used in RL to speed up training and convergence, and document a benchmark of improvement.
- Train a network on one game, and classify games into groups by seeing which other games this network solves "well". This can provide insight on how transfer learning works in this specific RL case by seeing what kinds of subgroups end up being formed and lay a path for further analysis.