CS342 FINAL PROJECT WRITEUP

JEFFREY JIANG, KURTIS DAVID, QUANG DUONG

Racing Strategy

We decided that there were several things that could be hard coded, in order to simplify the task of our neural network. The hard coded things did not need to be outputs of our network. When playing the game, we take the action output the network and add on several values.

(1) Our kart will always accelerate. Therefore, we always add 4 to the output of our network
(2) Items aren't particularly important in learning how to drive. Therefore, we also hard coded for items to be used immediately when acquired. We also use the boost whenever it is acquired as well.
(3) Using the state variable given by the function step, we can retrieve the position_along_track value, which we used to check if our agent was idling. If it idled too long, we hard coded a rescue.

We also decided that self-supervised learning was going to be much worse than if we could fully supervise training, so we chose to do imitation learning.

Data collection

In order to do imitation learning, we have to have a dataset recorded from something we want our agent to imitate. One option was to record one of us playing the game, but we decided to hard code a basic AI to play the game and record its actions for use in our dataset. The gist of the AI's strategy is:

(1) The AI always accelerates
(2) The AI attempts to minimize the state variable distance_to_center, which we use in conjunction with the angle variable to determine whether to turn left or right. Our reasoning is that our ideal player would simply follow the center of the track at all times, accelerating the entire time.

We chose to go with the AI for several reasons

(1) Player inputs aren't very clean. For example, there are several steps where we pressed both left and right at the same time, and some times where we didn't press any buttons. This makes player input pretty unpredictable. In contrast, an AI has extremely predictable behavior, which makes the classification task much easier for the network.
(2) The AI has a very well defined goal in terms of the state, which again should make the agent easier to train, since we give the state as input. Therefore, it should be relatively straightforward to predict the AI's action.

To record the data, we simply recorded the action our AI output, as well as the state of the game at the time step to numpy arrays and saved them. Some of the variables did not seem very useful, so we omitted them in the recording, we chose position_along_track, distance_to_center, speed, angle, and smooth_speed, as our input variables to be recorded.

In addition, we found that collecting images and using them as input to harmful, as they added too many additional variables into our network, when the state already provides all the information we wanted for our policy. As another note, our basic AI does not do any image processing, so making the input to the network the same as the variables used by the AI's policy is quite reasonable.

Network Architecture

What we tried

Some other things we tried

(1) **Gradient Free:** Our first idea was to use a gradient free method similar to HW10, trying both a genetic algorithm and a hill climbing algorithm. However, the results we had were not particularly good. We felt like the task was too complex to learn in a gradient free manner. In addition, evaluating the fitness value of a set of weights would take a long time, since it would have to play the game for several seconds. Therefore, optimization would take a long time, unless we limited ourselves to a very small amount of samples and a low amount of training steps
(2) **Q-Learning:** We also tried a self-supervised method. Again, we found that the task is probably too complex to learn without supervision, and our Q-Learning algorithm didn't produce very good results. Even though we thought there was a relatively natural reward function to optimize (distance to center), the agent we trained in the manner did not perform very well.
(3) **Fully Connected:** We also tried a very straightforward classifcation network that we trained by taking states and feeding it through 5 fully connected layers, with the AI's output as the labels. This worked quite well in replicating the AI, but we decided that LSTM's were a more interesting architechture.

One particular thing to noe with gradient free and Q-learning is that both are heavily reliant on a reward/fitness function, which is quite difficult to define, since it's hard to quantify how well you are driving. Also, reward functions can have unexpected behavior which is optimal for the reward, but is not desirable for actually playing the game.