Project: Learning To Drive

Paul Yoo

NetID: yoosehy

1 Overview

Driving is a complicated task with many variables that can affect the action of a driver. Some challenges include road conditions that may unexpectedly impact the dynamics of the car and predicting and coping with other vehicles at intersections and lane changes. One way to successfully train an agent to learn these behaviors is through reinforcement learning in a simulated environment. In this project, I use CARLA[1] simulation to train an agent to navigate around the simulated environment. I use a Deep Q-Learning approach to train an agent to learn controls that prevent the agent from colliding. CARLA comes with many sensors for the car to help with navigation. These sensors include but are not limited to Lidar, collision detector, IMU, and depth camera. In this project I mainly incorporated measurements from the depth camera and collision detector.

Figure 1: Measurements from the car camera

These images from the depth camera are the input to the Deep Q-Learning model and the output of the model is a prediction of the Q value over action space. In my case, the action that an agent can control is the steering angle and throttle of the car. However, since these controls are continuous values and the model can output only a finite set of numbers, I normalized and discretized the action space by letting steering angle and throttle range from a value of 0 to 1 where a value of 0 is a full left turn of the steering wheel and a value of 1 is a full right turn.
The main goal of the agent is to learn the optimal policy given a view of the road from the camera such that executing the policy maximizes reward. The reward in this can be a combination of multiple variables such as how many collisions, how smooth the agent drives(i.e. no abrupt changes in velocity), number of lines crossed, and how many traffic signals were violated. For my project, I focused mainly on training an agent that can drive around the simulated world as long as possible without any collisions. The reward of each state after taking an action was defined as follows:

$$r(s_t, a_t) = \begin{cases} -10 & s_t = collision \\ -3 & s_t = redTrafficLight \wedge speed > 0 \\ -2 & s_t = crossedSolidLine \\ -1 & speed < 5 \\ 1 & otherwise \end{cases}$$

Here a different reward is given based on the current state and action of the agent. When the agent collides with any object in the environment, it receives the most negative reward since the goal is to prevent any type of collision. Additionally, the agent will receive negative reward for violating rules on the road. For instance, if the traffic light is red and the agent still hasn't come to a stop, there is a negative reward. Also a negative reward is given when the agent crosses a solid line. Otherwise, the agent receives a positive reward for surviving without any violations and collisions. However, by allowing the agent to continuously receive reward as long as it's not violating anything, the agent will keep staying still in a fixed spot. In order to address this issue, I have applied a small penalty for agents moving very slowly.

Figure 2: Sensor detecting solid line crossing

Figure 3: Sensor detecting collision with the sidewalk

2 Method

Deep Q-Learning relies on the agent to learn the Q values associated with each state and action. Once we design a neural network to learn a Q function over the states of the environment and actions, the optimal policy is just the action that maximizes the Q-value.

$$\pi^*(s) = \arg\max_a Q(s, a)$$

Collecting data for the neural network to train on is done on the fly while the agent explores the simulation. This can be done by running the agent for multiple episodes. An episode in this case, can be a timed session that the agent explores the world after being spawned at random locations or it can be a indefinite session that keeps going on until the agent collides. To reduce training time, I have decided to set a timer for how long an episode lasts. At every episode, the environment is reset and the agent is spawned randomly again. At each state of the game, the agent saves the state which is the camera measurement

and also the reward after taking the best action output by the model into a buffer. Once the buffer reaches a certain size, we can sample small batches that can then act as a replay memory for training the model. It also makes sense to keep and train on the most recent memories by discarding old memories once the buffer reaches a certain size.

The neural network architecture is a convolutional neural network with a single fully connected layer that maps directly to the output layer which represents the action space. The convolution layers used were similar to those of general image classification networks like AlexNet. This network will take as input, the image from the depth camera and output the approximate Q-value for all possible actions. Using the replay memory, we sample a random batch and compute Q-values of those memories according to the Bellman equation. The Bellman equation states that the optimal Q-value function Q^* is the maximum expected cumulative reward achievable from a given (state, action) pair.

$$Q^*(s,a) = \mathbf{E}_{s'}[r + \gamma \max_{a'} Q^*(s',a')|s,a]$$

In the CARLA environment, the transition model is deterministic, meaning given a state and an action, we end up in another state without having chances of ending up in different states. In reality, however, the controls might not be applied properly or the road condition might be in a state that enables the car to end up in multiple possible states. The expectation over transition states becomes a single term as follows:

$$Q^*(s,a) = r + \gamma \max_{a'} Q^*(s',a')$$

If we are at a state marking the end of an episode, the Q-value is simply just equal to the reward $Q^*(s,a) = r$.
The mean squared error loss function was used with Adam optimization method to minimize the difference in expected Q-value and predicted Q-value.

The weights of the model can be saved afterwards and later be loaded to test the agent on a new environment.

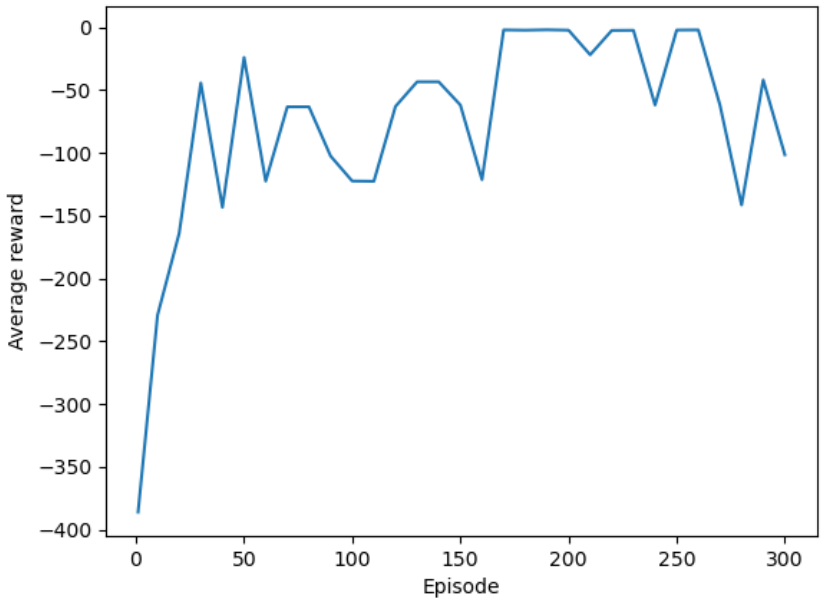

Figure 4: Average reward over episodes of training

The change in average reward of the agent over episodes trained is visualized above. The agent actually never achieves positive reward as 300 episodes of training is not enough to train an agent to steer the wheels properly. After 300 episodes, training took an indefinite amount of time due to the limited computing power of my laptop. There are lots of room for improvements for the agent other than training for more episodes. Some improvement that can be made is, having the agent perform more complicated tasks such as learning to change lanes by incorporating more sensor measurements. Additionally, the reward function can take into account collision magnitudes to better learn what series of actions lead to a more unstable trajectory.

3 References

[1] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, Vladlen Koltun. CARLA: An Open Urban Driving Simulator. In 1st Conference on Robot Learning (CoRL), 2017.

[2] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. In International Conference on Learning Representations (ICLR), 2017.