Robotics 1 – HW3 Write Up

GitHub Repo link (please use the “main” branch): https://github.com/rho-selynn/592-HW3

Team Members: Michael Holm, Roselynn Conrady

1. Using the provided GitHub repo link (https://github.com/udacity/RoboND-Rover-Project) for this assignment, we were able to run the rover simulation in Training Mode. We navigated the rover around and picked up objects. The data that we used can be seen in the “RoverTrainingData_Grid” folder. Within that folder is a .csv file called “robot_log.csv” that has the Path, Steer Angle, Throttle, Brake, Speed, X_Position, Y_Position, Pitch, Yaw, and Roll of the rover. And we also have a folder called “IMG” that has the images from capturing the video of the rover moving and picking up objects.

2. After loading in the images and data, we normalized the data so that way we worked with values between 0 and 1 for the steer angle, throttle, and image values. The images were loaded in grayscale, because it was believed that RGB value would not be valuable information to the model, and would potentially just confuse it. To create our training and testing datasets: From the images and data collected, we used 800 images to train the model. Then we used the remaining 200 images to test the model. These 800 images were paired with labels of both steering angle and throttle value, gathered from the simulation program.

The overall design of our model is a regression model that ends with a 1 dimensional output, being either the predicted steer value, or the predicted throttle value. Many architectures were tested for the model, in order to create the best possible outcome. This architecture included several linear layers, with a convolution layer at the beginning, several linear layers with various activation functions (e.g. ReLU, Sigmoid, and Threshold) and several linear layers of various sizes. Different optimization algorithms were tested, with us settling on a Mean Squared Error approach, in order to more harshly punish large errors in predicted value.

Finally, various batch sizes and epoch numbers were tested for the model training. We settled on batch sizes of 128 as providing the most optimal output for the models created. We also tried large epoch numbers, and found the models never reached a point of overtraining, although both models did plateau in their validation losses. Another thing we tried was to put random dropouts in the model, but that did not improve the models either.

Finally, we settled on 4 linear layers of sizes (160*320 by 128), (128 by 10), (10 by 5), and (5 by 1) in our model. Then we performed a forward pass on these models, and computed backpropagation in order to follow a gradient descent approach, to find the most optimal weight values for both networks. It was found that the steer model plateaued with a training loss value of approximately 0.17 and test loss value of roughly 0.02. However the throttle model did not perform nearly as well, plateauing at a training loss value of about 0.23 and a test loss value of roughly 0.25 on the validation set.

For both models, the testing/validation loss was lower than the training loss. This may be because our training model had some images and data that were “hard” to learn. For example, there were times when the rover paused for a while to pick up objects. Maybe our testing data did not have as many images when the rover was performing that task (e.g., the testing data may have had more images of the rover driving straight).

For future work, we would begin testing other approaches for the steering and throttle models. Perhaps a classification approach would give better results? Perhaps introducing stochasticity to the model, to get out of the local minima the model seems to get stuck in would give better results? There are many approaches that could be taken to improve the quality of this steering model, but it is difficult to say which one would provide the most accurate predictions at this point in time.

3. When running an inference study, it was found that steering values are predicted really quite well. When the rover should be going straight, the model almost always gets this right. However, when the rover should be turning, the model gets this correct maybe 33% of the time based on the preliminary results in the final code cell.

We believe this is because, in the total 800 images used to train the model, only a small percentage of them include the rover actually turning. This means that one of the most optimal models would almost always guess zero, which indicates that the model is getting stuck in a local minimum, as discussed earlier. For future work, we could attempt to fix this by collecting data of only the rover turning, or removing data points where the turn value is below a certain threshold.

With the throttle values, the model was almost always guessing values of 0.3-0.4. We believe this is because the throttle of the rover was almost always zero in the data collected, because when driving the car, we did not need to accelerate a lot. When the throttle wasn’t zero, it was almost always 1, because of the way the rover game works. This means that one of the most optimal models would predict 0.3-0.4 for every picture, because this would provide one the lowest losses. Again, we believe this could be fixed with more time by either introducing a stochastic approach to the model, to make sure the model doesn’t get stuck in local minima, or by tailoring the data to include more non-zero throttle values. Finally, because the throttle values are almost always either 0 or 1, a better model may use a classification approach, rather than a regression approach. Again, it is hard to say without spending more time on the model and the data.