Project 3 Writeup - Behavioral Cloning

The goals / steps of this project are the following:

- Use the simulator to collect data of good driving behavior
- Build, a convolution neural network in Keras that predicts steering angles from images
- Train and validate the model with a training and validation set
- Test that the model successfully drives around track one without leaving the road
- Summarize the results with a written report

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Files Submitted & Code Quality

1. Submission includes all required files and can be used to run the simulator in autonomous mode

My project includes the following files:

- model.py containing the script to create and train the model
- drive.py for driving the car in autonomous mode
- model.h5 containing a trained convolutional neural network that drives the car
- output_video.mp4 which contains the video of the car driving in autonomous mode
- writeup_report.md summarizing the results

2. Submission includes functional code using the Udacity provided simulator and my drive.py file, the car can be driven autonomously around the track by executing

```
python drive.py model.h5
```

3. Submission code is usable and readable

The model.py file contains the code for training and saving the convolution neural network. The file shows the pipeline I used for training and validating the model, and it contains comments to explain how the code works.

Model Architecture and Training Strategy

1. An appropriate model architecture has been employed

Before training the model, I normalized the data using a Keras lambda layer. For my model architecture, I used the NVIDIA architecture. My models consists of 5 convolutional layers, followed by 5 fully connected layers. In between my layers I incorporated max pooling and dropout. The model also includes RELU layers to introduce nonlinearity.

2. Attempts to reduce overfitting in the model

In my model, I added dropout layers to reduce overfitting. I also made sure to train and validate my model on different data sets to ensure there was no overfitting. I also used the model to drive the car, and based on it's maneuvers, I was able to determine if the model overfit.

3. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually. I did however include a learning rate parameter of .0001. Based on my findings from project 2, I realized this was an appropriate learning rate for training a model.

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used a combination of center lane driving, recovering from the left and right sides of the road, driving both directions on the track, and driving back to the center starting from the sides.

For details about how I created the training data, see the next section.

<u>Model Architecture and Training Strategy</u>

1. Solution Design Approach

The overall strategy for deriving a model architecture was to create the best validation percentage possible.

My first step was to preprocess the data. Preprocessing is a very important aspect of training a network. You must ensure that the model does not overfit the data. I used two main methods to preprocess. First I split up my data into a test set and a validation set. Secondly, I used a lambda layer which randomized and shuffled my data.

There were other methods I could have used in order to preprocess the data. I could have included images from the other two camera angles. I also could have augmented the data by flipping it, rotating it, or changing the color space. For the sake of time, I just used the two above data preprocessing methods.

After preprocessing, I used a convolutional neural network model similar to the Nvidia model. I thought this model might be appropriate because of how powerful it has been proven to be. If it works for some of the leading tech companies in the world, it should work for my project. See below for the exact specifications of the neural network I created.

Once my data was preprocessed and I had a network in place, the last step was to gather enough data from the simulation to put into the model. To collect the data, I drove around the track 2 times clockwise and 2 times counter clockwise. I also did 2 total laps of driving from off the road back to the center. This added up to a total of 6 laps. If I wanted to improve the model, I could gather even more data. I could also drive on the more difficult track. This would help the model to generalize better, thus creating a more robust model.

After all this work, the vehicle is able to drive autonomously around the track without leaving the road.

2. Final Model Architecture

The final model architecture consisted of a convolution neural network with the following layers:

1. Normalization
2. Convolution Layer (5x5x24) + RELU activation
3. Max Pooling Layer (pool size of 2x2 and strides of 1x1)
4. Convolution Layer (5x5x36) + RELU activation
5. Max Pooling Layer (pool size of 2x2 and strides of 1x1)
6. Convolution Layer (5x5x48) + RELU activation
7. Max Pooling Layer (pool size of 2x2 and strides of 1x1)
8. Convolution Layer (3x3x64) + RELU activation
9. Convolution Layer (3x3x64) + RELU activation
10. Flatten Layer
11. Dropout Layer w/ 0.5 keep
12. Dense Layer (100)
13. Dropout Layer w/ 0.5 keep
14. Dense Layer (100)
15. Dense Layer (50)
16. Dropout Layer w/ 0.5 keep
17. Dense Layer (10)
18. Dropout Layer w/ 0.5 keep
19. Dense Layer (1)

3. Creation of the Training Set & Training Process

To capture good driving behavior, I first recorded two laps on track one using center lane driving going counter clockwise. Then I drove two laps counter clockwise. Next I recorder 1 lap recovering from the sides of the road going clockwise. I did the same thing going counter clockwise. I had 6 laps of recorder data.

After the collection process, I had 15,000+ data points. I then pre processed this data by adding a lambda layer to shuffled the data. I also randomly shuffled the data set and put 20% of the data into a validation set.

I used this training data for training the model. The validation set helped determine if the model was over or under fitting. The ideal number of epochs was 8 as evidenced by the drop in accuracy after this number of epochs. I also used an adam optimizer so that manually training the learning rate wasn't necessary.

Thank you for checking out my project. Hope you learned something about behavioral cloning!