Behavioral Cloning

Behavioral Cloning Project

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.

Model Architecture and Training Strategy

1. An appropriate model architecture has been employed

See `create_model`.

When picking my model architecture I knew I wanted at least two Conv2Ds to since my network would need to recognize spacial patterns, such as road curves and road edges, in order to drive effectively. I didn't believe the the LeNet architecture's configuration would be a perfect fit size it was used for smaller images and was designed to recognize more complex patterns. Nonetheless, in the face of positive results, I used the LeNet filter size (5x5) and kept the depths (6 and 16). I kept LeNet's MaxPooling. I also kept the RELU's to introduce nonlinearity.

Before data enters the Conv2D layers, it is normalized. Next it is cropped to remove visual features that shouldn't impact driving.

After the Con2D, I have three dense layers of width 120, 80 and 1. Between every dense layer there is dropout.

There is no activation function at the end of the model, since this isn't a classification problem.

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting. I used an 80-20 training-validation split so that I could determine when overfitting was occuring. I determined that over fitting was occuring in the very second or third epoch. As a result, I didn't train more than one epoch.

3. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually.

4. Appropriate training data

Training data was chosen to teach the vehicle the following:

- When the vehicle was too far on one side of the road
- How to drive the vehicle when vehicle was in the center of the road

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

Model Architecture and Training Strategy

1. Solution Design Approach

When picking my model architecture I knew I wanted at least two Conv2Ds to since my network would need to recognize spacial patterns, such as road curves and road edges, in order to drive effectively. I wasn't certain what depth or filter size to use. I didn't believe the the LeNet architecture's configuration would be a perfect fit size it was used for smaller images and was designed to recognize more complex patterns. Nonetheless, this architecture had proven suitable for recognizing spatial patterns in the past. So it seemed a good starting point. Given the extreme simplicitly of this assignment's problem (turn right when too close to the left, turn right when too far to the left) I was a bit concerned that my model would be too complex for the problem and cause overfitting. I took a few stabs at preventing this.

First, I tried adding dropout layers between my dense neural network layers. This reduced validation loss during training. But it caused the car to perform worst on the track. I settled on the following simple approach: only train for one epoch.

I wasn't confident that the LeNet architecture was the right choice. But at this stage, I decided that collecting good training data was the key to this assignment. So I put my concerns out of mind for now.

I briefly tried the multi-camera trick: use left, right and center cameras with adjusted input labels to teach the car to drive towards the recorded car position. However, I felt this approach taught the car to retrace its exact path instead of teaching the car to avoid the edges of the track. Plus, the cameras

were closer toghether then I wanted them to be. If I was going to use fake labels, I felt I could do something that more directly taught the model what I wanted it to learn.

I took three recordings along the first half of the track:

1. Car driving too far to the left
2. Car driving too far to the right
3. Car driving well

I assigned fake stearing angles to all images in set (1) and (2). Then I randomized all (image, label) pairs in the three sets. Testing my model on the car showed that it performed well. However, it got confused at the bridge. I believe this was because I had a relatively small number of images from the bridge. So I recorded an additional set of images from the bridge and retrained my model.

At the end of this process, the vehicle is able to drive autonomously around the track without leaving the road.

I decided to try simplying my model to remove some of the unnecessary layers. My label collection approach suggests that my model was functioning somewhat similarly to the following interpolation problem: **detect the left side and ride side, determine how far we are from each side and then perform some linear interpolation on the result**. I reasoned that I might not need the second Conv2D to detect track edges. I removed the Conv2D and the car drove 80% of the track. Perhaps I could have gotten this approach working. Instead of following this train of thought farther, I stuck with the LeNet architecture since it already worked.

2. Final Model Architecture

The final model architecture (`create_model()`) consisted of a convolution neural network with the following layers and layer sizes:

- Normalization layer
- Cropping layer
- Convolution, 5x5 filter, 2x2 stride and depth of 6
- Max pooling layer
- Convolution, 5x5 filter, 2x2 stride and depth of 16
- Max pooling layer
- Flatten to one dimension, so we can apply dense neural network layers
- Neural network layer (width of 120)
- Neural network layer (width of 84)
- Neural network layer (width of 1)

3. Creation of the Training Set & Training Process

To capture good driving behavior, I first recorded half a lap on track one using center lane driving. Here is an example image of center lane driving:

I then recorded half a track of the vehicle driving along the left side of the lane to teach the car when it must drive urgently to the right. The following consecutive images are an example of this:

Then I repeated this process on the first half of the track for the right hand size in order to get more data points.

To further augment the data sat and remove bias resulting from training on a counter-clockwise track, I flipped a copy of every single image and multiplied its label by -1 before using it to train my model.

I finally 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 1 as evidenced by the increase in validation loss in the second epoch. I used an adam optimizer so that manually training the learning rate wasn't necessary.