Behavioral Cloning Project

Background:

The purpose of the project is to develop a behavioral cloning network by training on a driving video in order to compute steering angle. The process is as follows:

- use the simulator to collect video of what good driving behavior.
- Build a convolution neural network in Keras that predicts steering angles from the video images
- Train and validate the model with a training and validation set
- Test that the model successfully drives around the track once without leaving the road.
- Summarizes the findings

Rubric Outline

- Submitted files: model.py, drive.py, model.h5 and write-up and video.mp4
- The model.py is well written and functional. It clearly shows what the convolutional network is shaped and the reasons for various decisions.
- Model Architecture: I chose Nvidia's model after testing others. I feel that Nvidia is able to learn quickly (less than 5 epochs) without much drop-out required for addressing overfitting.
- Train vs validation split. I chose a 80% training vs 20% validation split. The simulator in automated driving served the purpose of test set.
- Overfitting issues: I was getting a training set MSE of 0.038 vs. validation set MSE of 0.107. So, there was some overfitting. I adjusted the learning rate and also added dropout. I found that additional video of good driving reduced the overfitting along with a smaller learning rate to address the initial underfitting.
- The training data was sampled by the driving simulator. The training and the validation sets were shuffled from the entire set for true random sampling.

Model

The model chose is the Nvidia model. The following is the code representing the model in Keras API. The cropping of the image to cut out the sky was an important factor. Also, the

```
model = Sequential()

# normalize image values between -.5 : .5
model.add(Lambda(lambda x: x / 255.0 - .5, input_shape=(in_row, in_col,
in_depth)))

## horizon: 70, front of car: 25
model.add(Cropping2D(cropping=((70, 25), (0, 0)), input_shape=(in_row, in_col,
in_depth)))

# valid border mode should get rid of a couple each way, whereas same keeps
# Use relu (non-linear activation function), not mentioned in Nvidia paper but
a standard
model.add(Conv2D(24, kernel_size=(5, 5), strides=(2, 2), padding='valid',
activation='relu'))
model.add(Conv2D(36, kernel_size=(5, 5), strides=(2, 2), padding='valid',
activation='relu'))
model.add(Conv2D(48, kernel_size=(5, 5), strides=(2, 2), padding='valid',
activation='relu'))
model.add(Conv2D(64, kernel_size=(3, 3), strides=(1, 1), padding='valid',
activation='relu'))
model.add(Conv2D(64, kernel_size=(3, 3), strides=(1, 1), padding='valid',
activation='relu'))
model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dense(100, activation='relu'))
model.add(Dense(50, activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dense(1, activation='tanh'))

# compile with normal adam optimizer (loss .001) and return
adam = Adam(lr=lr)
model.compile(loss='mse', optimizer=adam)
```

Simulation

The car in the simulator is able to navigate the course correctly. The challenges faced were in dealing with the dirt turn-off areas. Additional samples were taken in order to compensate for these special areas of the course.

Conclusion

Behavioral cloning is a powerful tool in the machine learning practitioner's toolbox. The value is with developing a system that can learn by being provided examples of the behavior. This becomes viable with high performance computing resources along with low demanding network model. The challenges of development were in getting the best data (cropped, good examples) compared to getting the model exactly right.