BEHAVIOURAL CLONING FOR SELF DRIVING CAR

DEEP NEURAL NETWORK ARCHITECTURE

The network that I used was exactly the same as the NVIDIA behavioural cloning architecture. I used Keras to implement this model. This model is a tested model by NVIDIA behavioural cloning team. Besides this model is derived from Lenet in a way. Before trying this model I tried Lenet. But for this particular training data set Lenet had a huge validation loss. So I went ahead with NVIDIA model which gave a good validation accuracy.

1) The input image was of dimensions (160,320,3)
2) The previous input was passed through a lambda layer to normalize and mean center the images in the range 0 to 0.5
3) The previous lambda layer passed the output into a cropping layer. This layer cropped the image height by 50 pixels in top and 30 pixels in bottom. No cropping of the image was done widthwise. The output of this layer is an image of size (80, 320, 3)
4) The previous cropped image was passed into a convolution layer. This layer used a filter of size (5,5,3) and strides (2,2) and 24 output filters. The output of this layer was of the dimensions (38, 158, 24)
5) The output of the previous layer was passed through relu activation unit
6) The output of previous convolution layer was passed through another convolution layer. The filter used was of dimensions (5,5,24). The number of output filters was 36. The strides were (2,2). The output of this layer was of the dimensions (17, 77, 36).
7) The output of the previous layer was passed through relu activation unit
8) The output of the previous layer was passed through another convolution layer. The filter used was of dimensions (5,5,36). The number of output filters was 48. The strides were (2,2). The output of this layer was of the dimensions (7, 37, 48).
9) The output of the previous layer was passed through relu activation unit.
10) The output of the previous layer was passed through another convolution layer. The filter was of the dimensions (3,3,48). The number of output filters was 64. The strides were (1,1). The output of this layer was of the dimensions (5, 35, 64)
11) The output of the previous layer was passed through relu activation unit.
12) The output of the previous layer was passed through another convolution layer. The filter was of the dimensions (3,3,64). The number of output filters were 54. The strides were (1,1). The output of this layer was of the dimensions (3, 33, 54)
13) The output of the previous layer was passed through relu activation unit
14) The output of the previous layer was flattened to 5346 featuress
15) The 5346 features of the previous layer were passed through a full layer of 100 perceptrons.
16) The output of the previous layer was passed through a full layer of 50 perceptrons.
17) The output of the previous layer was passed through a full layer of 10 perceptrons.
18) The output of the previous layer was passes through a single perceptron. This perceptron outputs the steering angle.

MODEL COMPILATION

Mean squared error was used as the metric to optimize the weights and biases. The optimizer used was adam optimizer. Adam stands for Adaptive Moment optimizer. This is a more optimized stochastic gradient descent. This is a combination of gradient descent with momentum algorithm and RMS prop algorithm.

The gradient descent with momentum algorithm is used to accelerate gradient descent algorithm by taking into consideration the exponentially weighted average of the gradients.

Using averages makes the algorithm converge towards the minima faster.

MODEL TRAINING

The batch size used was 128 and 10 epochs. The data set was divided into 80% training data and 20% validation data before starting with training. In all total images that were used for training were close to 15000. We trained the model over both the tracks two times each. We used recovery data to train the model to recover when the car went astray. I drove the car at very low speeds so that I could illustrate correct steering angles to the model. I did not use the left mirror and right mirror images.

We used a generator to generate the input data. Before each epoch we shuffled the training data. The generator returned a flipped (left-right) subset of each batch of data after each normal batch of data.

We also used a callback. The EarlyStopping callback. This callback was called to stop training when the validation loss remained below 0.01 for three consecutive epochs.

MODEL PERSISTENCE

We saved the model to model.h5

IMAGE DATASET