Behavioural Cloning

Udacity Student

Udacity
Self-driving car Nanodegree Program
Project 3

Contents

1 Introduction **1**

2 Project structure **1**

3 Dataset characteristics **2**
3.1 Preprocessing . . . 3
3.2 Augmentation . . . 3

4 Model **4**
4.1 Architecture . . . 4
4.2 Training procedure . . . 4

5 Results **5**
5.1 Autonomous drive . . . 5

6 Conclusion **6**

1 Introduction

This document serves as the project writeup for the Udacity Self-driving car Nanodegree Program, term 1, project 3, behavioural cloning. The project code, as well as the present document can be retrieved at https://github.com/kacl780tr/carnd_project_03.

The goal of this project is to develop and train a neural net architecture to drive a simulated car around a track.

2 Project structure

It should be noted that for this project the data handling code has been gathered into a companion file **drive_data.py**, which is imported into the main model training file **model.py**, rather than implementing everything directly within the training file. This allows the training code file itself to be maintained a relatively clean state.

Note also that in order for the trained network to function correctly on my machine, modifications were made to the **drive.py** file to adjust the Keras TensorFlow backend.

3 Dataset characteristics

The simulator used for this project has two operational modes

Training mode This mode allows the car to be "driven" via mouse and keyboard, while various data features are recorded for use in training the neural network. The data recorded are

- image from a center camera
- image from a left-hand camera
- image from a right-hand camera
- steering angle $\alpha \in [-1, 1]$
- throttle position
- brake position
- vehicle speed

Autonomous mode This mode allows the trained network to receive the current simulated camera views, and feeds the generated steering angle back to the simulator

Although brake and throttle are recorded in addition to the steering angle, for the purposes of this project only the steering angle was generated by the trained neural network, with the speed being automatically controlled with a provided feedback loop.

This setup allows essentially unlimited amounts of training data to be generated. There are two tracks available, and data can be generated from either. All training data generated were gathered using the mouse as the steering control, since, as suggested in the project materials, this resulted in smoother steering angles.

One data set was generated from each track. For both data sets

- The shape of a camera image is (160, 320, 3) *(height, width, channel)*,
- The steering angle α is a floating point value in the range $[-1.0, 1.0]$ corresponding to an actual steering angle $\theta = \alpha 25^\circ$,

then for the track specific data sets we had

Track 1 data

- The training set contains 56,347 examples
- The size of the validation set is 18,782
- The size of test set is 18,783

Track 2 data

- The training set contains 37,479 examples
- The size of the validation set is 12,493
- The size of test set is 12,494

Examples of the images from the center, left-, and right-hand cameras of the simulator for track 1 are shown in figures 1, 2, and 3, respectively.

Figure 1: Example of center camera image

Figure 2: Example of left-hand camera image

3.1 Preprocessing

Data normalization is well-known technique in machine learning. It makes the scales of the various dimensions much more equal, and thus makes the task of optimization less dependent on specific directions or starting points.

For our purposes, we have chosen to normalize the image data such that all values lie in the range (-1, +1), which makes the image symmetric about zero. The images are provided as unsigned byte arrays, and must be converted to floating point arrays to avoid overflow issues with the normalization.

Also, in order to allow the neural network to focus more on relevant details in the images, each image was cropped before being input to the neural network. Cropping consisted of removing the upper 55 pixels, as well as the lower 25 pixels, leaving a normalized image of size (80, 320, 3) as input to the neural network proper. The lower crop results in the removal of the vehicle hood from the input, which is entirely useless as a control feature. The upper crop, on the other hand, serves to remove many distractions such as trees.

In the implementation here, the preprocessing steps are applied as the initial layers of the neural network. This allows the images from the simulator in autonomous mode to be fed directly into the neural network prediction function, rather than preprocessed separately beforehand.

3.2 Augmentation

For supervised learning problems such as the one here, more data is almost always better. It was therefore decided to apply data augmentation techniques as a means of increasing the amount of data available, in order to

- Increase the variety of features available to the network.
- Decrease the probability of overfitting.

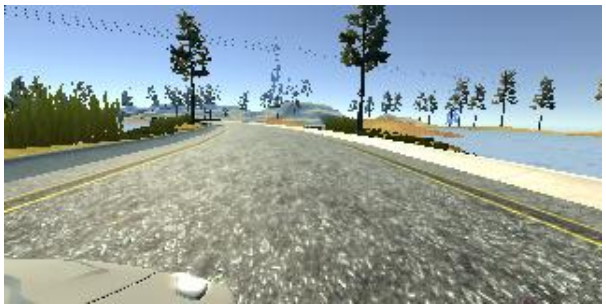

Figure 3: Example of right-hand camera image

The augmentation technique applied here is that of left-right reversal. Images from all three cameras are reversed in the horizontal plane, accompanied with a change of sign in the associated steering angle. In this way a sequence of images recorded from driving around the track counter-clockwise can be made to look as though the track was driven in the clockwise direction. This effectively doubles the size of the data available, without the inconvenience of actually having to drive the track in the clockwise direction.

Also, the steering angles for left- and right-hand images were biased from the recorded values, in order to provide corrective feedback to the neural network. The bias value $\delta = 0.1$ was found to work quite well for both tracks.

4 Model

4.1 Architecture

The final architecture was chosen from amongst several alternatives. These alternatives included

- a multi-layer perceptron architecture (no convolutions),
- a network based on the LeNet architecture,
- two different networks based on the VGG architecture,
- two different networks based on the NVIDIA architecture as described in the project materials.

The final architecture was chosen because it provided satisfactory performance on both track 1 and track 2 as well as being more compact than some alternatives considered. A detailed description of the architecture is provided in table 1.

The loss measure used for gradient descent is the mean-squared error on the steering angles output by the network.

4.2 Training procedure

The model was trained using the Adam optimizer, with all parameters except the learning rate left at their default values. Learning rate decay was employed such that the learning rate was reduced by a factor of 0.75 each time a sequence of 3 epochs was encountered without a reduction in the validation loss. This was implemented using the Keras **ReduceLROnPlateau** callback facility.

To facilitate collection of results, the Keras **ModelCheckpoint** callback was used to save the model each time a new minimum was achieved in the validation loss. Insufficient progress toward a new global minimum validation loss resulted in early termination of the run, through the use of the Keras **EarlyStopping** callback.

Layer	In	Out	Notes
Input	-	$160 \cdot 320 \cdot 3$	Placeholder for RGB image
Cropping	$160 \cdot 320 \cdot 3$	$80 \cdot 320 \cdot 3$	Keras Cropping2D layer
Normalization	$80 \cdot 320 \cdot 3$	$80 \cdot 320 \cdot 3$	Keras Lambda layer
Convolution $(5 \cdot 5)$	$80 \cdot 320 \cdot 3$	$38 \cdot 158 \cdot 24$	Stride 2, padding = valid
ReLU activation	$38 \cdot 158 \cdot 24$	$38 \cdot 158 \cdot 24$	-
Convolution $(5 \cdot 5)$	$38 \cdot 158 \cdot 24$	$17 \cdot 77 \cdot 36$	Stride 2, padding = valid
ReLU activation	$17 \cdot 77 \cdot 36$	$17 \cdot 77 \cdot 36$	-
Convolution $(5 \cdot 5)$	$17 \cdot 77 \cdot 36$	$7 \cdot 37 \cdot 48$	Stride 2, padding = valid
Convolution $(3 \cdot 3)$	$7 \cdot 37 \cdot 48$	$5 \cdot 35 \cdot 64$	Stride 1, padding = valid
Convolution $(3 \cdot 3)$	$5 \cdot 35 \cdot 64$	$3 \cdot 33 \cdot 64$	Stride 1, padding = valid
Flatten	$3 \cdot 33 \cdot 64$	$1 \cdot 6336$	-
Fully connected	$1 \cdot 6336$	$1 \cdot 100$	-
ReLU activation	$1 \cdot 100$	$1 \cdot 100$	-
Fully connected	$1 \cdot 100$	$1 \cdot 50$	-
ReLU activation	$1 \cdot 50$	$1 \cdot 50$	-
Fully connected	$1 \cdot 50$	$1 \cdot 1$	- Final output steering angle

Table 1: Model layer architecture

The primary means of preventing overfitting was simply the provision of adequate amounts of data, given that data acquisition was effectively unlimited, and was easily augmented as described in section 3.2. In fact, the amount of data was probably well in excess of that required, and resulted in training taking longer than necessary.

It was not found necessary to use the generator approach, as the machine which was used for this project has a substantial complement of memory.

5 Results

At completion of training, the model presented the following results:

- training loss $\lessapprox 0.0042$
- validation loss of 0.0038
- testing loss of 0.0034

These results provide evidence that the model is fitting well, without overfitting, and is capable of generalizing to unseen data in a reasonable manner.

5.1 Autonomous drive

As can be seen from the files **video.mp4** and **video_2.mp4** the model does a satisfactory job of navigating multiple laps of both tracks 1 and 2.

This is good evidence that the model has not simply "memorized" either track, but is, in fact, reacting to features identified in the images.

While several architectures were found to be able to navigate track 1, the number which were successful on both tracks were only three. The chosen architecture was notable in being able to complete several laps of both tracks.

6 Conclusion

In this project is a very small example of what is sometimes called an "end-to-end", or deep-learning approach to autonomous vehicle control. Rather than explicitly programming a set of specialized functions to recognize lane lines, and to integrate other sensor data, one simply allows a deep network to decide for itself what is, and what is not, important in the sensor data streams.

It does, however, suggest that this approach holds promise. It would be interesting to take a reinforcement learning approach to such an environment.