SelfDrivingCar

February 12, 2019

```
In [1]: import numpy as np
        import pandas as pd
        import os
        import matplotlib.pyplot as plt
        import seaborn as sns
        import imageio
        import scipy.misc
        from tqdm import tqdm
        import warnings
        from tensorflow.core.protobuf import saver_pb2
        from tensorflow.python.client import device_lib

In [2]: warnings.filterwarnings('ignore', category=FutureWarning)

In [3]: !gpustat

instance-gpu  Mon Feb 11 16:12:23 2019
[0] Tesla P100-PCIE-16GB | 34'C,   0 % |    82 / 16280 MB | root(82M)


In [15]: !cat ./dataset2/data.txt | wc -l

45406


In [16]: !head -n 1 ./dataset2/data.txt

0.jpg 0.000000


In [17]: steer_df = pd.read_csv('./dataset2/data.txt', sep=' ',
                                names=['image_file', 'steer_degree'],
                                engine='python')

In [18]: steer_df['image_file'] = './dataset2/' + steer_df.image_file

In [96]: steer_df.shape

Out[96]: (45406, 3)
```

```
In [19]: steer_df.head()
```

```
Out[19]:          image_file  steer_degree
         0  ./dataset2/0.jpg           0.0
         1  ./dataset2/1.jpg           0.0
         2  ./dataset2/2.jpg           0.0
         3  ./dataset2/3.jpg           0.0
         4  ./dataset2/4.jpg           0.0
```

Convert degrees to Radians

```
In [21]: steer_df['steer_rad'] = steer_df.steer_degree * 0.01745
```

```
In [22]: steer_df.steer_rad.unique()
```

```
Out[22]: array([ 0.       ,  0.204165 ,  0.1813055, ..., -1.4781895, -1.372617 ,
                -0.883319 ])
```

```
In [23]: steer_df.shape
```

```
Out[23]: (45406, 3)
```

```
In [27]: train = steer_df.iloc[0:31785, :]  # More than 31785 for epoch calculations
         test = steer_df.iloc[31785:, :]
```

```
In [28]: train.shape
```

```
Out[28]: (31785, 3)
```

```
In [29]: test.shape
```

```
Out[29]: (13621, 3)
```

0.0.1 Convert image into feature matrix

```
In [30]: # def _convert_image(image_path, strip, resize, normalize):
         #     img_matrix = imageio.imread(image_path)
         #     if strip is not None:
         #         img_matrix = img_matrix[-strip:]
         #     if resize is not None:
         #         img_matrix = scipy.misc.imresize(img_matrix, resize)
         #     if not normalize:
         #         return img_matrix
         #     return img_matrix / 255.0
```

```
In [106]: # train_batch_pointer = 0
          # test_batch_pointer = 0
```

In [133]:
```
# def get_batch_data(size, subset='train', **kwargs):
#     global train_batch_pointer
#     global test_batch_pointer
#     strip = kwargs.get('strip')
#     resize = kwargs.get('resize')
#     normalize = kwargs.get('normalize', True)
# #     train_pointer = kwargs.get('train_pointer', -1)
# #     test_pointer = kwargs.get('test_pointer', -1)

#     fetcher = kwargs.get('fetcher', 'sequential')
#     if fetcher not in ['sequential', 'randomized']:
#         raise NotImplementedError('Fetcher type is not implemented')

#     x = []

#     if subset == 'train':
#         if fetcher == 'randomized':
#             train_index = np.random.randint(low=0, high=train.shape[0], size=size)
#         else:
# #             if train_batch_pointer == - 1 or train_batch_pointer > train.shape[0]:
# #                 raise IndexError('Train pointer value is less/more than it should
#             train_index = np.arange(train_batch_pointer, train_batch_pointer + size)
# #         print(train_pointer, train_pointer+size)
#         images = train.image_file.values[train_index]
#         y = train.steer_rad.values[train_index]
#         train_batch_pointer += size

#     elif subset == 'test':
#         if fetcher == 'randomized':
#             train_index = np.random.randint(low=0, high=test.shape[0], size=size)
#         else:
# #             if test_batch_pointer == -1 or test_batch_pointer > test.shape[0]:
# #                 raise IndexError('Test pointer value is less/more than it should b
#             test_index = np.arange(test_batch_pointer, test_batch_pointer + size)
#         images = test.image_file.values[test_index]
#         y = test.steer_rad.values[test_index]
#         test_batch_pointer += size
#     for image in images:
#         x.append(_convert_image(image, strip, resize, normalize))

#     return np.array(x), y.reshape(-1, 1)
import random
xs = []
ys = []

train_batch_pointer = 0
val_batch_pointer = 0
#read data.txt
```

```
with open("./dataset2/data.txt") as f:
    for line in f:
        xs.append("./dataset2/" + line.split()[0])
        #the paper by Nvidia uses the inverse of the turning radius,
        #but steering wheel angle is proportional to the inverse of turning radius
        #so the steering wheel angle in radians is used as the output
        ys.append(float(line.split()[1]) * scipy.pi / 180)

#get number of images
num_images = len(xs)

#shuffle list of images
c = list(zip(xs, ys))
random.shuffle(c)
xs, ys = zip(*c)

train_xs = xs[:int(len(xs) * 0.8)]
train_ys = ys[:int(len(xs) * 0.8)]

val_xs = xs[-int(len(xs) * 0.2):]
val_ys = ys[-int(len(xs) * 0.2):]

num_train_images = len(train_xs)
num_val_images = len(val_xs)


def LoadTrainBatch(batch_size):
    global train_batch_pointer
    x_out = []
    y_out = []
    for i in range(0, batch_size):
        x_out.append(scipy.misc.imresize(
            scipy.misc.imread(train_xs[(
                train_batch_pointer + i) % num_train_images])[-150:], [66, 200]) / 255
        y_out.append([train_ys[(train_batch_pointer + i) % num_train_images]])
    train_batch_pointer += batch_size
    return x_out, y_out

def LoadValBatch(batch_size):
    global val_batch_pointer
    x_out = []
    y_out = []
    for i in range(0, batch_size):
        x_out.append(scipy.misc.imresize(
            scipy.misc.imread(
                val_xs[(val_batch_pointer + i) % num_val_images])[-150:], [66, 200]) /
        y_out.append([val_ys[(val_batch_pointer + i) % num_val_images]])
    val_batch_pointer += batch_size
```

```
        return x_out, y_out
```

0.0.2 Build the Conv Net

```
In [134]: import tensorflow as tf
```

```
In [135]: def get_weights(shape, name, init='glorot_uniform'):
              """
              Get weights and initialize them
              for the each layer.
              """
              if init == 'glorot_uniform':
                  i = tf.glorot_uniform_initializer(seed=42)
              elif init == 'glorot_normal':
                  i = tf.glorot_normal_initializer(seed=42)
              elif init == 'xavier_uniform':
                  i = tf.contrib.layers.xavier_initializer(seed=42)
              elif init == 'xavier_normal':
                  i = tf.contrib.layers.xavier_initializer(seed=42, uniform=False)
              elif init == 'he_uniform':
                  i = tf.keras.initializers.he_uniform(seed=42)
              elif init == 'he_normal':
                  i = tf.keras.initializers.he_normal(seed=42)
              elif init == 'raw':
                  if len(shape) > 1:
                      initial = tf.truncated_normal(shape, stddev=0.1)
                      return tf.Variable(initial)
                  else:
                      initial = tf.constant(0.1, shape=shape)
                      return tf.Variable(initial)
          #         i = tf.initializers.random_normal(seed=42)

              return tf.get_variable(
                  name=name,
                  shape=shape,
                  dtype=tf.float32,
                  initializer=i,)

          def flatten(X, size):
              return tf.reshape(X, [-1, size])


          def Dense(X, size, init, name, activation):
              w = get_weights(shape=size, name='W_' + name, init=init)
              b = get_weights(shape=[size[-1]], name='b_' + name, init=init)

              dense = tf.matmul(X, w) + b
              print(name, size, size[-1])
```

```
        ## Applying activation

        if activation == 'relu':
            h_fc = tf.nn.relu(dense)
        elif activation == 'sigmoid':
            h_fc = tf.nn.sigmoid(dense)
        elif activation == 'leaky_relu':
            h_fc = tf.nn.leaky_relu(dense)
        elif activation == 'tanh':
            h_fc = tf.nn.tanh(dense)
        elif activation == 'atan':
            h_fc = tf.atan(dense)

    #     if dropout >= 0.0 and dropout < 1.0:
    #         return tf.nn.dropout(h_fc, keep_prob=dropout)
        return h_fc

    def Conv2d(X, size, stride, init, name, padding, activation):
        """
        Get a conv layer on X for weight W and bias b
        with stride and padding
        """
        print(name, size, size[-1])
        w = get_weights(shape=size, name='W_' + name, init=init)
        b = get_weights(shape=[size[-1]], name='b_' + name, init=init)

        conv = tf.nn.conv2d(X, w, strides=[1, stride, stride, 1],
                            padding=padding) + b

        ## Applying activation

        if activation == 'relu':
            h_conv = tf.nn.relu(conv)
        elif activation == 'sigmoid':
            h_conv = tf.nn.sigmoid(conv)
        elif activation == 'leaky_relu':
            h_conv = tf.nn.leaky_relu(conv)

        return h_conv
```

In [136]:
```
# get_weights((5, 5, 3, 24), name='conv2d_1').shape
```

In [137]:
```
# get_weights((24), name='bias_conv2d_1').shape
```

In [138]:
```
def get_available_gpus():
    local_device_protos = device_lib.list_local_devices()
    return [x.name for x in local_device_protos if x.device_type == 'GPU']
```

0.0.3 Network 1

```
In [139]: tf.reset_default_graph() # Reset any existing computation graphs

In [140]: x = tf.placeholder(name='input', dtype=tf.float32, shape=[None, 66, 200, 3])
          y_ = tf.placeholder(name='output', dtype=tf.float32, shape=[None, 1])
          keep_prob = tf.placeholder(tf.float32)

In [141]: x_image = x

In [142]: # first convolutional layer
          h_conv1 = Conv2d(x_image, (5, 5, 3, 24), 2,
                           init='raw', name='conv2d_1',
                           padding='VALID', activation='relu')

          # second convolutional layer
          h_conv2 = Conv2d(h_conv1, (5, 5, 24, 36), 2,
                           init='raw', name='conv2d_2',
                           padding='VALID', activation='relu')

          # third convolutional layer
          h_conv3 = Conv2d(h_conv2, (5, 5, 36, 48), 2,
                           init='raw', name='conv2d_3',
                           padding='VALID', activation='relu')

          # fourth convolutional layer
          h_conv4 = Conv2d(h_conv3, (3, 3, 48, 64), 1,
                           init='raw', name='conv2d_4',
                           padding='VALID', activation='relu')

          # fifth convolutional layer
          h_conv5 = Conv2d(h_conv4, (3, 3, 64, 64), 1,
                           init='raw', name='conv2d_5',
                           padding='VALID', activation='relu')

          # Flatten layer
          h_conv5_flatten = flatten(h_conv5, size=1152)

          # Dense layer 1
          h_fc1 = Dense(h_conv5_flatten, (1152, 1164), name='dense1',
                        init='raw',
                       activation='relu')
          # Dropout 1
          h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

          # Dense Layer 2
          h_fc2 = Dense(h_fc1_drop, (1164, 100), name='dense2',
                        init='raw',
                       activation='relu')
```

```
            # Dropout 2
            h_fc2_drop = tf.nn.dropout(h_fc2, keep_prob)

            # Dense Layer 3
            h_fc3 = Dense(h_fc2_drop, (100, 50), name='dense3',
                          init='raw',
                         activation='relu')
            # Dropout 3
            h_fc3_drop = tf.nn.dropout(h_fc3, keep_prob)

            # Dense Layer 4
            h_fc4 = Dense(h_fc3_drop, (50, 10), name='dense4',
                          init='raw',
                         activation='relu')

            # Dropout 4
            h_fc4_drop = tf.nn.dropout(h_fc4, keep_prob)

            # Output
            Y = Dense(h_fc4_drop, (10, 1), name='output',
                      init='raw',
                      activation='atan')
            y = tf.multiply(Y, 2)
```

```
conv2d_1 (5, 5, 3, 24) 24
conv2d_2 (5, 5, 24, 36) 36
conv2d_3 (5, 5, 36, 48) 48
conv2d_4 (3, 3, 48, 64) 64
conv2d_5 (3, 3, 64, 64) 64
dense1 (1152, 1164) 1164
dense2 (1164, 100) 100
dense3 (100, 50) 50
dense4 (50, 10) 10
output (10, 1) 1
```

0.0.4 Train the network

```
In [143]: LOGDIR = './save/'

In [144]: get_available_gpus()

Out[144]: ['/device:GPU:0']

In [ ]: with tf.Session() as sess:
            L2NormConst = 0.001
            train_vars = tf.trainable_variables()
            loss = tf.reduce_mean(tf.square(tf.subtract(y_, y))) + tf.add_n(
                [tf.nn.l2_loss(v) for v in train_vars]) * L2NormConst
```

```
train_step = tf.train.AdamOptimizer(1e-4).minimize(loss)
sess.run(tf.global_variables_initializer())

# create a summary to monitor cost tensor
tf.summary.scalar(name="RMSE", tensor=loss)
# merge all summaries into a single op
merged_summary_op = tf.summary.merge_all()
saver = tf.train.Saver(write_version = saver_pb2.SaverDef.V1)

logs_path = './logs'
summary_writer = tf.summary.FileWriter(logdir=logs_path, graph=tf.get_default_graph(
epochs = 30
batch_size = 100
with tf.device('/gpu:0'):
    for epoch in range(epochs):
        for i in range(int(num_images/batch_size)):
            xs, ys = LoadTrainBatch(batch_size)
            train_step.run(feed_dict={x: xs, y_: ys, keep_prob: 0.8})
            if i % 10 == 0:
                xs, ys = LoadValBatch(batch_size)
                loss_value = loss.eval(feed_dict={x: xs, y_: ys, keep_prob: 1.0})
                print("Epoch: %d, Step: %d, Loss: %g" % (epoch, epoch * batch_size +

            # write logs at every iteration
            summary = merged_summary_op.eval(feed_dict={x:xs, y_: ys, keep_prob: 1.0
            summary_writer.add_summary(summary, epoch * num_images/batch_size + i)

            if i % batch_size == 0:
                if not os.path.exists(LOGDIR):
                    os.makedirs(LOGDIR)
                checkpoint_path = os.path.join(LOGDIR, "model.ckpt")
                filename = saver.save(sess, checkpoint_path)
        print("Model saved in file: %s" % filename)
print("Run the command line:\n" \
          "--> tensorboard --logdir=./logs " \
          "\nThen open http://0.0.0.0:6006/ into your web browser")
```

Last 5 steps of the last epoch

```
Epoch: 29, Step: 3310, Loss: 0.146305
Epoch: 29, Step: 3320, Loss: 0.137446
Epoch: 29, Step: 3330, Loss: 0.132673
Epoch: 29, Step: 3340, Loss: 0.130797
Epoch: 29, Step: 3350, Loss: 0.443287
```

0.0.5 Network 2

```
In [154]: tf.reset_default_graph() # Reset any existing computation graphs
```

```
In [155]: x = tf.placeholder(name='input', dtype=tf.float32, shape=[None, 66, 200, 3])
          y_ = tf.placeholder(name='output', dtype=tf.float32, shape=[None, 1])
          keep_prob = tf.placeholder(tf.float32)

In [156]: x_image = x

In [157]: # first convolutional layer
          h_conv1 = Conv2d(x_image, (5, 5, 3, 24), 2,
                           init='raw', name='conv2d_1',
                           padding='VALID', activation='relu')

          # second convolutional layer
          h_conv2 = Conv2d(h_conv1, (5, 5, 24, 36), 2,
                           init='raw', name='conv2d_2',
                           padding='VALID', activation='relu')

          # third convolutional layer
          h_conv3 = Conv2d(h_conv2, (5, 5, 36, 48), 2,
                           init='raw', name='conv2d_3',
                           padding='VALID', activation='relu')

          # fourth convolutional layer
          h_conv4 = Conv2d(h_conv3, (3, 3, 48, 64), 1,
                           init='raw', name='conv2d_4',
                           padding='VALID', activation='relu')

          # fifth convolutional layer
          h_conv5 = Conv2d(h_conv4, (3, 3, 64, 64), 1,
                           init='raw', name='conv2d_5',
                           padding='VALID', activation='relu')

          # Flatten layer
          h_conv5_flatten = flatten(h_conv5, size=1152)

          # Dense layer 1
          h_fc1 = Dense(h_conv5_flatten, (1152, 1164), name='dense1',
                        init='raw',
                       activation='relu')
          # Dropout 1
          h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

          # Dense Layer 2
          h_fc2 = Dense(h_fc1_drop, (1164, 100), name='dense2',
                        init='raw',
                       activation='relu')
          # Dropout 2
          h_fc2_drop = tf.nn.dropout(h_fc2, keep_prob)
```

```
            # Dense Layer 3
            h_fc3 = Dense(h_fc2_drop, (100, 50), name='dense3',
                          init='raw',
                         activation='relu')
            # Dropout 3
            h_fc3_drop = tf.nn.dropout(h_fc3, keep_prob)

            # Dense Layer 4
            h_fc4 = Dense(h_fc3_drop, (50, 10), name='dense4',
                          init='raw',
                         activation='relu')

            # Dropout 4
            h_fc4_drop = tf.nn.dropout(h_fc4, keep_prob)

            # Output
            W_fc5 = get_weights([10, 1], name='W_fc5', init='raw')
            b_fc5 = get_weights([1], name='b_fc5', init='raw')
            y = tf.matmul(h_fc4_drop, W_fc5) + b_fc5
```

```
conv2d_1 (5, 5, 3, 24) 24
conv2d_2 (5, 5, 24, 36) 36
conv2d_3 (5, 5, 36, 48) 48
conv2d_4 (3, 3, 48, 64) 64
conv2d_5 (3, 3, 64, 64) 64
dense1 (1152, 1164) 1164
dense2 (1164, 100) 100
dense3 (100, 50) 50
dense4 (50, 10) 10
```

0.0.6 Train the network

```
In [158]: LOGDIR = './save/'

In [159]: get_available_gpus()

Out[159]: ['/device:GPU:0']

In [ ]: with tf.Session() as sess:
            L2NormConst = 0.001
            train_vars = tf.trainable_variables()
            loss = tf.reduce_mean(tf.square(tf.subtract(y_, y))) + tf.add_n(
                [tf.nn.l2_loss(v) for v in train_vars]) * L2NormConst
            train_step = tf.train.AdamOptimizer(1e-3).minimize(loss)
            sess.run(tf.global_variables_initializer())

            # create a summary to monitor cost tensor
            tf.summary.scalar(name="RMSE", tensor=loss)
```

```
# merge all summaries into a single op
merged_summary_op = tf.summary.merge_all()
saver = tf.train.Saver(write_version = saver_pb2.SaverDef.V1)

logs_path = './logs'
summary_writer = tf.summary.FileWriter(logdir=logs_path, graph=tf.get_default_graph(
epochs = 30
batch_size = 100
with tf.device('/gpu:0'):
    for epoch in range(epochs):
        for i in range(int(num_images/batch_size)):
            xs, ys = LoadTrainBatch(batch_size)
            train_step.run(feed_dict={x: xs, y_: ys, keep_prob: 0.5})
            if i % 10 == 0:
                xs, ys = LoadValBatch(batch_size)
                loss_value = loss.eval(feed_dict={x: xs, y_: ys, keep_prob: 1.0})
                print("Epoch: %d, Step: %d, Loss: %g" % (epoch, epoch * batch_size +

            # write logs at every iteration
            summary = merged_summary_op.eval(feed_dict={x:xs, y_: ys, keep_prob: 1.0
            summary_writer.add_summary(summary, epoch * num_images/batch_size + i)

            if i % batch_size == 0:
                if not os.path.exists(LOGDIR):
                    os.makedirs(LOGDIR)
                checkpoint_path = os.path.join(LOGDIR, "model2.ckpt")
                filename = saver.save(sess, checkpoint_path)
        print("Model saved in file: %s" % filename)
print("Run the command line:\n" \
          "--> tensorboard --logdir=./logs " \
          "\nThen open http://0.0.0.0:6006/ into your web browser")
```

Last 5 steps of last epoch

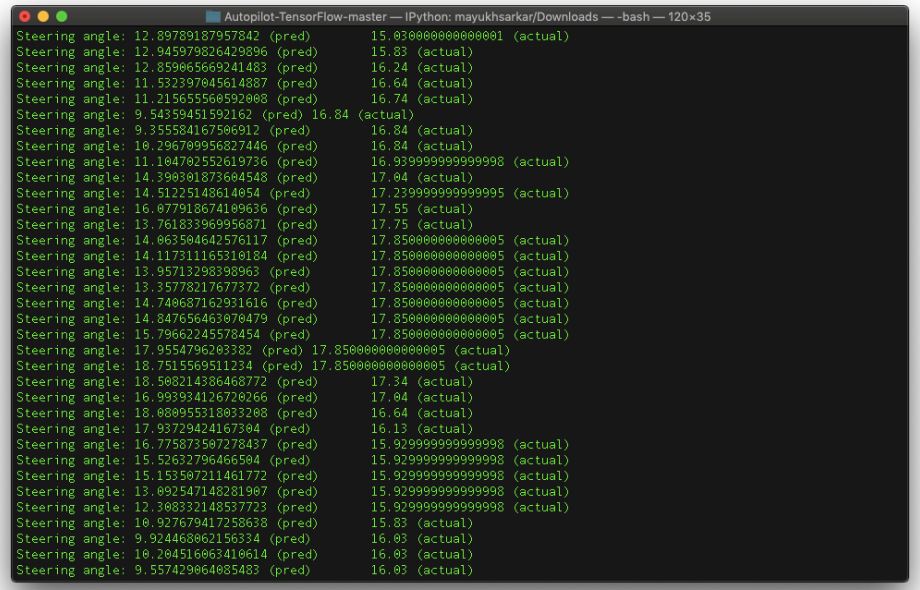
```
Epoch: 29, Step: 3310, Loss: 0.0919582
Epoch: 29, Step: 3320, Loss: 0.136387
Epoch: 29, Step: 3330, Loss: 0.193896
Epoch: 29, Step: 3340, Loss: 0.134995
Epoch: 29, Step: 3350, Loss: 0.126513
```

For video link: https://youtu.be/zhbPPFHbLDE

0.0.7 Saved Models

```
In [8]: !ls -l save
```

```
total 50360
-rw-rw-r-- 1 mayukhpay mayukhpay       79 Feb 11 23:40 checkpoint
-rw-rw-r-- 1 mayukhpay mayukhpay 31920357 Feb 11 23:40 model2.ckpt
```



Output

```
-rw-rw-r-- 1 mayukhpay mayukhpay   317497 Feb 11 23:40 model2.ckpt.meta
-rw-rw-r-- 1 mayukhpay mayukhpay 19152108 Feb 11 21:13 model.ckpt
-rw-rw-r-- 1 mayukhpay mayukhpay   166715 Feb 11 21:13 model.ckpt.meta
```

0.0.8 Observations

- The initial model works better
- The updated model is performing well but not as good as the initial one.
- I hardly think not doing `atan` at output was the main reason.
- Because now we are directly taking values instead of `atan` and square of it, the loss on paper is better thatn the previous one because I am getting more granular control over the output instead of a smoothed out value.
- I guess the lower performacne is due to the `learning rate` and `dropout` rate changes.

Below you can see the sample output.