TensorFlow TUTORIAL

Le Anh Tu

Faculty of Electrical & Electronics Engineering,
Department of Electronics & Telecommunications Engineering,
Ho Chi Minh City University of Technology, Vietnam

July, 2017

Outline

- Introduction
- Basic Usage
- Build Model
- Other Utilities

WHAT IS TENSORFLOW ?

- Tensorflow is a deep learning library recently open-sourced by Google.
- But what does it actually do?
  - Tensorflow provides primitives for defining functions on tensors and automatically computing their derivatives

Outline

Introduction

Basic Usage

Build Model

Other Utilities

BASIC USAGE

example 1:

- a = tf.constant(2)
- b = tf.constant(3)
- c = a + b

example 2:

- x = tf.placeholder(tf.int16)
- y = tf.placeholder(tf.int16)
- z = tf.add(x,y)

example 3:

input = tf.placeholder(tf.float32)

weight = tf.constant([[2.],[2.]])

bias = tf.constant(3.0)

output = tf.add(tf.matmul(input,weight), bias)

LOAD DATA & VARIABLES INITIALIZER

- from tensorflow.examples.tutorials.mnist import input_data
- mnist = input_data.read_data_sets("mnist_data/", one_hot = true)

- learning_rate = 0.001
- training_epochs = 15
- batch_size = 100
- display_step = 1

- n_input = 784 #shape:28x28
- n_classes = 10 #0-9 digits
- x = tf.placeholder('float', [none, n_inputs])
- y = tf.placeholder('float', [none, n_classes])

Outline

- Introduction
- Basic Usage
- Build Model
- Other Utilities

NEURAL NETWORK

- n_hidden_1 = 256
- n_hidden_2 = 256

- def neural_network(x, weights, biases):
  - layer_1 = tf.add(tf.matmul(x , weights['h1']), biases['b1'])
  - layer_1 = tf.nn.relu(layer_1)
  - layer_2 = tf.add(tf.matmul(layer_1, weights['h2']), biases['b2'])
  - layer_2 = tf.nn.relu(layer_2)
  - out_layer = tf.matmul(layer_2, weights['out']) + biases['out']
  - return out_layer

- weights = {
  - 'h1': tf.variable(tf.random_normal([n_input, n_hidden_1])),
  - 'h2': tf.variable(tf.random_normal([n_hidden_1, n_hidden_2])),
  - 'out': tf.variable(tf.random_normal([n_hidden_2, n_classes]))
- }
- biases = {
  - 'b1': tf.variable(tf.random_normal([n_hidden_1])),
  - 'b2': tf.variable(tf.random_normal([n_hidden_2])),
  - 'out': tf.variable(tf.random_normal([n_classes]))
- }

- y_pred = neural_network(x, weights, biases)
- cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=y_pred, labels=y))
- optimizer = tf.train.adamoptimizer(learning_rate=learning_rate).minimize(cost)

NEURAL NETWORK

- with tf.session() as sess:
  - sess.run(tf.global_variables_initializer())
  - #training loop
  - for epoch in range(training_epochs):
    - avg_cost = 0.
    - total_batch = int(mnist.train.num_examples / batch_size)
    - #loop over all batches:
    - for i in range(total_batch):
      - batch_x, batch_y = mnist.train.next_batch(batch_size)
      - _, c = sess.run([optimizer, cost], feed_dict={x: batch_x, y: batch_y})
      - #comput average loss
      - avg_cost += c / total_batch
    - if epoch % display_step == 0:
      - print('epoch:', '%04d' %(epoch+1)m 'cost = ', \ '{:.9f}'.format(avg_cost))
  - print('optimization finished')

- #test mode
- correct_prediction = tf.equal(tf.argmax(y_pred, 1), tf.argmax(y, 1))
- accuracy = tf.reduce_mean(tf.cast(correct_prediction, 'float'))
- print('accuracy:', accuracy.eval({x: mnist.test..images, y: mnist.test.labels}))

```
Epoch: 0001 cost= 183.146648452
Epoch: 0002 cost= 43.430120397
Epoch: 0003 cost= 26.733731529
Epoch: 0004 cost= 18.591381240
Epoch: 0005 cost= 13.488394500
Epoch: 0006 cost= 9.907654405
Epoch: 0007 cost= 7.318386168
Epoch: 0008 cost= 5.641316663
Epoch: 0009 cost= 4.140372237
Epoch: 0010 cost= 3.029535837
Epoch: 0011 cost= 2.293382505
Epoch: 0012 cost= 1.781092959
Epoch: 0013 cost= 1.289214330
Epoch: 0014 cost= 1.023641920
Epoch: 0015 cost= 0.802592557
Optimization Finished!
Accuracy: 0.949
```

CONVOLUTIONAL NEURAL NETWORK

- training_inters = 200000
- dropout = 0.75

- def conv2d(x,w,b,strides=1):
  - x = tf.nn.conv2d(x,w, strides = [1,strides,strides,1], padding = 'same')
  - x = tf.nn.bias_add(x,b)
  - return tf.nn.relu(x)

- def maxpool2d(x, k=2):
  - return tf.nn.max_pool(x, ksize = [1,k,k,1], strides = [1,k,k,1], padding = 'same']

- def conv_net(x,weights, biases, dropout):
  - x = tf.reshape(x, shape = [-1,28,28,1])
  - conv1 = conv2d(x, weights['wc1'], biases['bc1'])
  - conv1 = maxpool2d(conv1, k=2)
  - conv2 = conv2d(conv1 , weights['wc2'], biases['bc2'])
  - conv2 = maxpool2d(conv2, k=2)
  - fc1 = tf.reshape(conv2, [-1, weights['wd1'].get_shape().as_list()[0]])
  - fc1 = tf.add(tf.matmul(fc1, weights['wf1']), biases['bf1'])
  - fc1 = tf.nn.relu(fc1)
  - fc1 = tf.nn.dropout(fc1, dropout)
  - out = tf.add(tf.matmul(fc1, weights['out']), biases['out'])
  - return out

CONVOLUTIONAL NEURAL NETWORK

- weights = {
  - ‘wc1’: tf.Variable(tf.random_normal([5,5,1,32])),
  - ‘wc2’: tf.Variable(tf.random_normal([5,5,32,64])),
  - ‘wf1’: tf.Variable(tf.random_normal([7*7*64, 1024])),
  - ‘out’: tf.Variable(tf.random_normal([1024, n_classes]))
- }

- biases = {
  - ‘bc1’: tf.Variable(tf.random_normal([32])),
  - ‘bc2’: tf.Variable(tf.random_normal([64])),
  - ‘bf1’: tf.Variable(tf.random_normal([1024])),
  - ‘out’: tf.Variable(tf.random_normal([n_classes]))
- }

```
Iter 167680, Minibatch Loss= 51.544586, Training Accuracy= 0.97656
Iter 168960, Minibatch Loss= 16.598854, Training Accuracy= 0.99219
Iter 170240, Minibatch Loss= 102.296074, Training Accuracy= 0.97656
Iter 171520, Minibatch Loss= 126.573250, Training Accuracy= 0.97656
Iter 172800, Minibatch Loss= 50.975868, Training Accuracy= 0.98438
Iter 174080, Minibatch Loss= 13.026268, Training Accuracy= 0.99219
Iter 175360, Minibatch Loss= 61.759331, Training Accuracy= 0.98438
Iter 176640, Minibatch Loss= 104.688667, Training Accuracy= 0.97656
Iter 177920, Minibatch Loss= 117.068787, Training Accuracy= 0.96875
Iter 179200, Minibatch Loss= 68.032127, Training Accuracy= 0.97656
Iter 180480, Minibatch Loss= 55.089828, Training Accuracy= 0.99219
Iter 181760, Minibatch Loss= 12.846817, Training Accuracy= 0.98438
Iter 183040, Minibatch Loss= 19.284180, Training Accuracy= 0.98438
Iter 184320, Minibatch Loss= 415.337708, Training Accuracy= 0.95312
Iter 185600, Minibatch Loss= 0.000000, Training Accuracy= 1.00000
Iter 186880, Minibatch Loss= 133.618011, Training Accuracy= 0.96875
Iter 188160, Minibatch Loss= 314.425293, Training Accuracy= 0.96875
Iter 189440, Minibatch Loss= 97.414436, Training Accuracy= 0.97656
Iter 190720, Minibatch Loss= 22.250843, Training Accuracy= 0.98438
Iter 192000, Minibatch Loss= 24.891357, Training Accuracy= 0.99219
Iter 193280, Minibatch Loss= 223.365280, Training Accuracy= 0.96094
Iter 194560, Minibatch Loss= 85.471329, Training Accuracy= 0.98438
Iter 195840, Minibatch Loss= 35.877510, Training Accuracy= 0.99219
Iter 197120, Minibatch Loss= 127.821899, Training Accuracy= 0.97656
Iter 198400, Minibatch Loss= 20.495216, Training Accuracy= 0.99219
Iter 199680, Minibatch Loss= 194.098282, Training Accuracy= 0.97656
Optimization Finished!
Testing Accuracy: 0.976563
```

RECURRENT NEURAL NETWORK

- from tensorflow.contrib import rnn
- n_input = 28 #img shape 28x28
- n_steps = 28 #timesteps
- n_hidden = 128 #hidden layer num of features
- n_classes = 10 #0-9 digits

- x = tf.placeholder(‘float’, [none, n_steps, n_input])
- y = tf.placeholder(‘float’, [none, n_classes])

- weights = {
  - ‘out’: tf.variable(tf.random_normal([n_hidden, n_classes]))
- }
- biases = {
  - ‘out’: tf.variable(tf.random_normal([n_classes]))
- }

- def rnn(x, weights, biases):
  - x = tf.unstack(x, n_steps, 1)
  - lstm_cell = rnn.basiclstmcell(n_hidden, forget_bias = 1.0)
  - outputs , states = rnn.static_rnn(lstm_cell, x, dtype(tf.float32)
  - return tf.matmul(outputs[-1], weights[‘out’]) + biases[‘out’]
- y_pred = rnn(x, weights, biases)

```
Iter 81920, Minibatch Loss= 0.064821, Training Accuracy= 0.97656
Iter 83200, Minibatch Loss= 0.114697, Training Accuracy= 0.97656
Iter 84480, Minibatch Loss= 0.115060, Training Accuracy= 0.96875
Iter 85760, Minibatch Loss= 0.162514, Training Accuracy= 0.95312
Iter 87040, Minibatch Loss= 0.173857, Training Accuracy= 0.95312
Iter 88320, Minibatch Loss= 0.107287, Training Accuracy= 0.96094
Iter 89600, Minibatch Loss= 0.121720, Training Accuracy= 0.95312
Iter 90880, Minibatch Loss= 0.112574, Training Accuracy= 0.97656
Iter 92160, Minibatch Loss= 0.138351, Training Accuracy= 0.96094
Iter 93440, Minibatch Loss= 0.209882, Training Accuracy= 0.93750
Iter 94720, Minibatch Loss= 0.088629, Training Accuracy= 0.98438
Iter 96000, Minibatch Loss= 0.089171, Training Accuracy= 0.96094
Iter 97280, Minibatch Loss= 0.161869, Training Accuracy= 0.94531
Iter 98560, Minibatch Loss= 0.026504, Training Accuracy= 1.00000
Iter 99840, Minibatch Loss= 0.047803, Training Accuracy= 0.98438
Optimization Finished!
Testing Accuracy: 0.953125
```

Outline

- Introduction
- Basic Usage
- Build Model
- Other Utilities

SAVE AND RESTORE MODEL

- model_path = './model/model.ckpt'

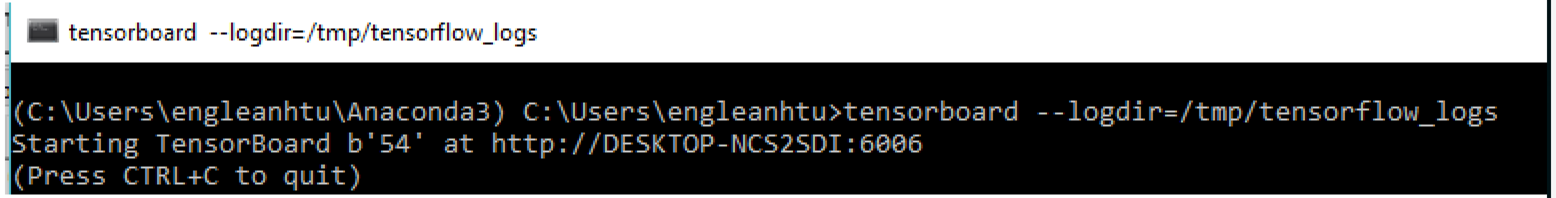
- #Build your model below
- ...
- #Put here before running session
- saver = tf.train.Saver()

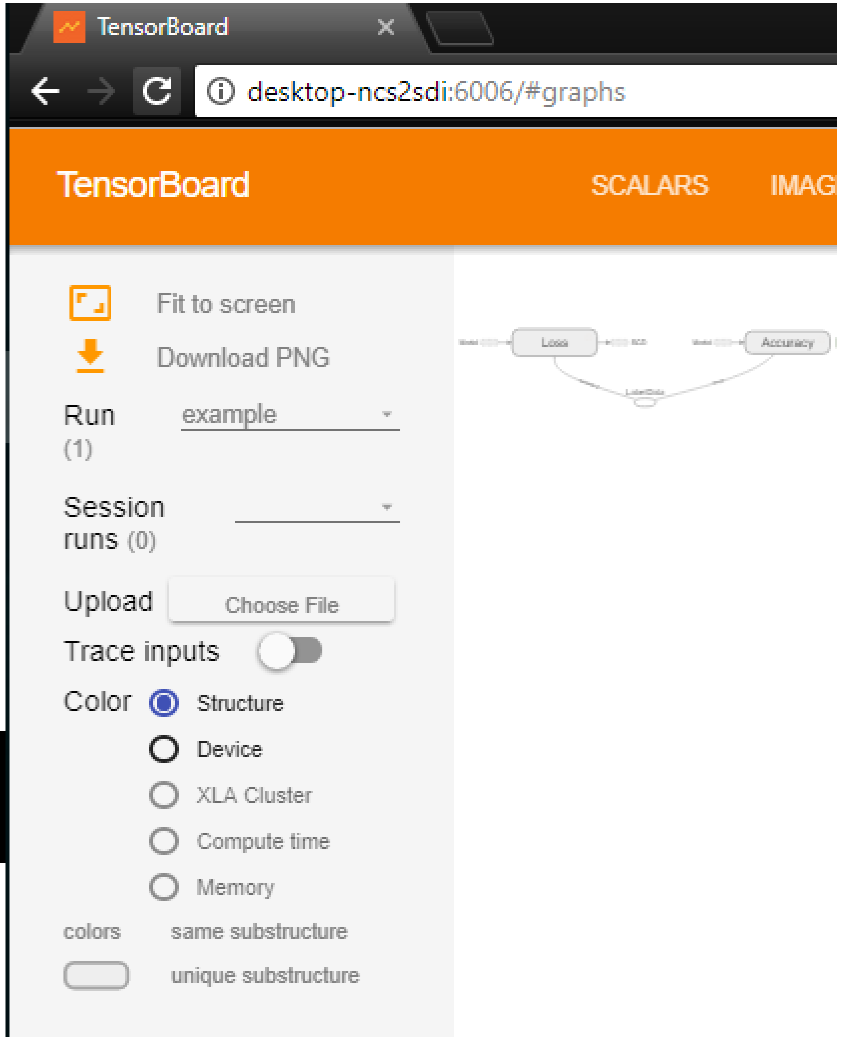
- #If you want to save your trained model
- save_path = saver.save(sess,model_path)

- #If you want to load your saved model to reuse
- load_path = saver.restore(sess,model_path)

TENSORBOARD

- See more at
- https://www.tensorflow.org/get_started/summaries_and_tensorboard





OPERATION ON MULTI-GPU

- # "/cpu:0":The CPU of your machine.
- # "/gpu:0":The first GPU of your machine
- # "/gpu:1":The second GPU of your machine

- #Processing Units Logs
- Log_device_placement = True

- #Using single GPU to compute
- With tf.device('/gpu:0'):
  - ...

- #Using CPU to compute
- With tf.device('/cpu:0'):
  - …

- #Using multi GPU to compute
- With tf.device('/gpu:0'):
  - …
- With tf.device('/gpu:1'):
  - …
- With tf.device('/cpu:0'):
  - …

THANK YOU

E-mail: eng.leanhtu@gmail.com
Code: github.com/engleanhtu/tensorflowtutorial

Suggestions Questions