Tensorflow Tutorial

Junru Shao
Shanghai Jiao Tong University

July / 18 / 2016

Deep Learning Frameworks

- Caffe
- Torch
- Theano
- Mxnet
- Tensorflow

* It seems that Caffe has no logo

Basic Concepts

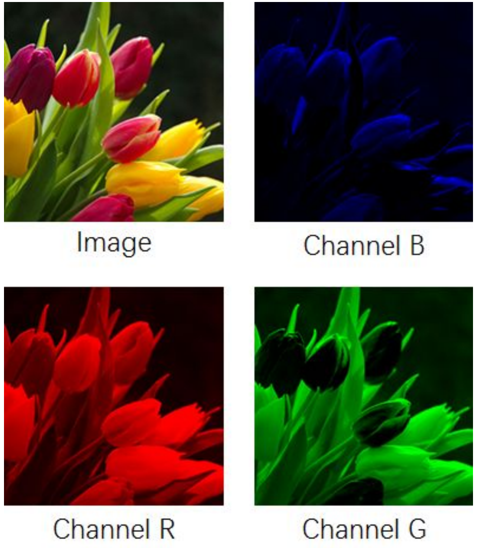
Basic Concepts: Data as Tensor

- Tensor
  - formally: maps from vector spaces to the real numbers
  - here: n-dimensional array

E.g. Tensor in Computer Vision

a single image: 3-d tensor

- [width, height, channels]



batch of images: 4-d tensor

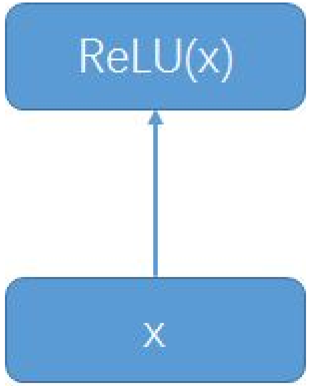
- [batch, width, height, channels]

CONT'D

- Define a tensor variable
  - tf.get_variable(name, shape, dtype, initializer)
  - e.g.
    ```
    W = tf.get_variable(
            weight, shape = [1, 2],
            dtype = tf.float32,
            initializer = tf.random_normal_initializer()
    )
    ```
- Define a tensor constant
  - tf.constant(value, dtype)

Basic Concepts: Computation as Graph

- Graph
  - node: tensor
  - edge: operation
- Operators
  - add / sub / mul / div
  - convolution
  - ReLU
- Just construct graph, no calculation is done



Basic Concepts: Launching Graph in a Session

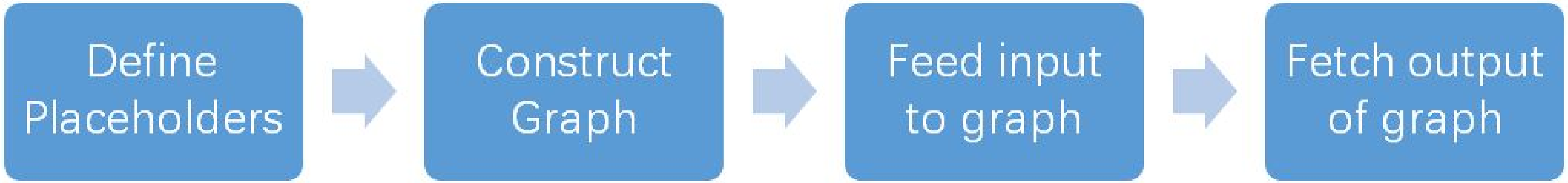
- Session
  - connect to C++ backend for efficient computation
- Calculation is done inside a specific session

Basic Concepts: Input and Output

- Input: placeholder
  - input = tf.placeholder(tf.float32, shape = (5, 10))
- Output
  - result_list = session.run(

```
    [tensors_you_want],
    feed_dict = {input : value}
)
```

Typical Basic Workflow



E.g.

```
import tensorflow as tf

# create a new session
with tf.Session() as sess:
    # create a placeholder for the 1 x 2 tensor
    matrix1 = tf.placeholder(tf.float32, shape = [1, 2])
    # create a 2 x 1 tensor
    matrix2 = tf.constant([[2.],[2.]])
    # add a hyperedge in graph, NO CALCULATION HERE
    product = tf.matmul(matrix1, matrix2)
    # do calculation
    print sess.run(product, feed_dict = {matrix1 : [[3., 3.]]})
```

```
[[ 12.]]
```

Automatic Differentiation and Optimizers

Automatic Differentiation

Core function:

```
tf.gradients(ys, xs)
```

- Given two lists of tensors, compute d(ys[i]) / d(xs[i])
- All optimizers are based on the function

Optimizers

- Various optimization algorithms
  - Sochastic gradient descent
  - Momentum
  - Adadelta
  - Adagrad
  - Adam
  - Follow-the-regularized-leader
  - RMSProp

CONT'D

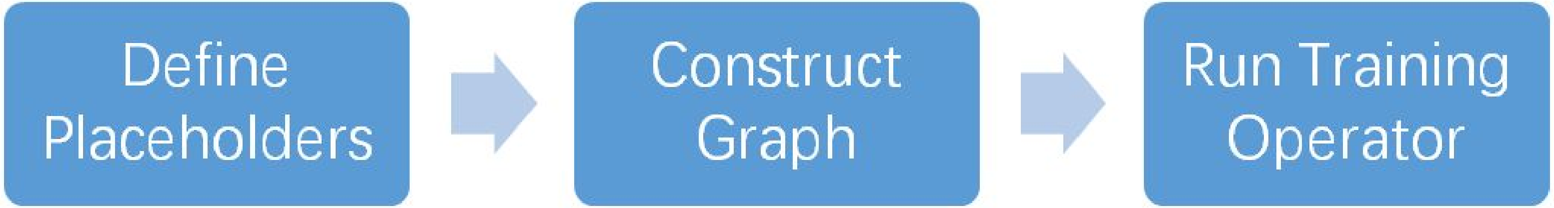
Usage:

```
opt = GradientDescentOptimizer(learning_rate = 0.1)  # create a new optimizer
train_op = opt.minimize(cost)                        # create an training operator
```

Evaluate the operator to train the model:

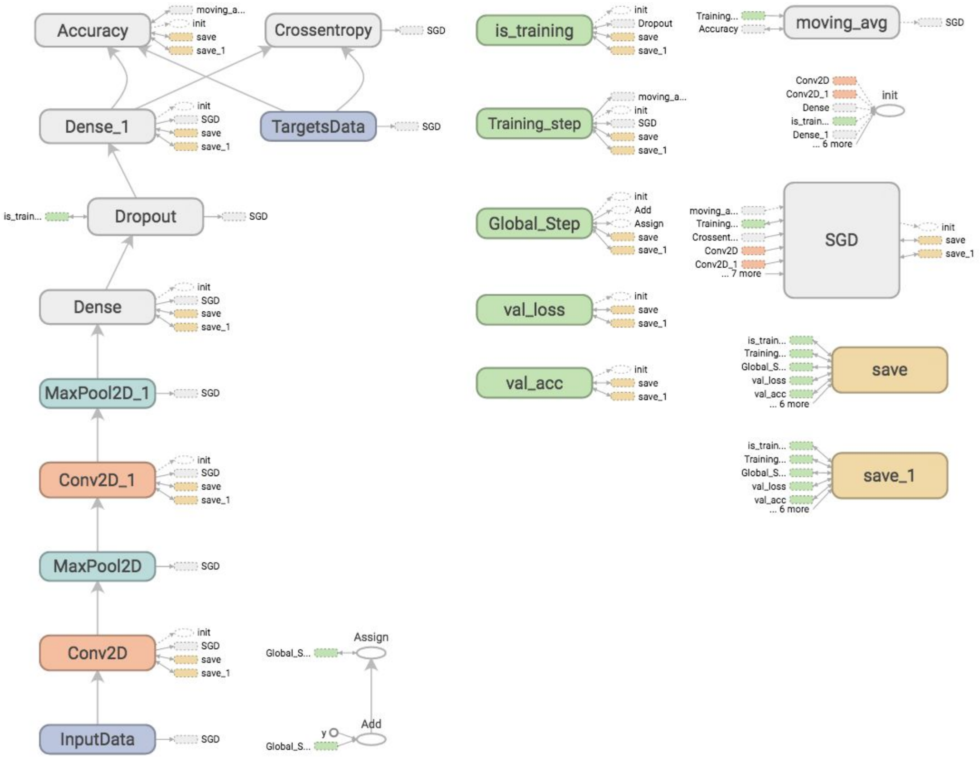
```
sess.run(train_op, feed_dict = <list of inputs>)     # a single iteration
```

Typical Workflow in Training Neural Networks

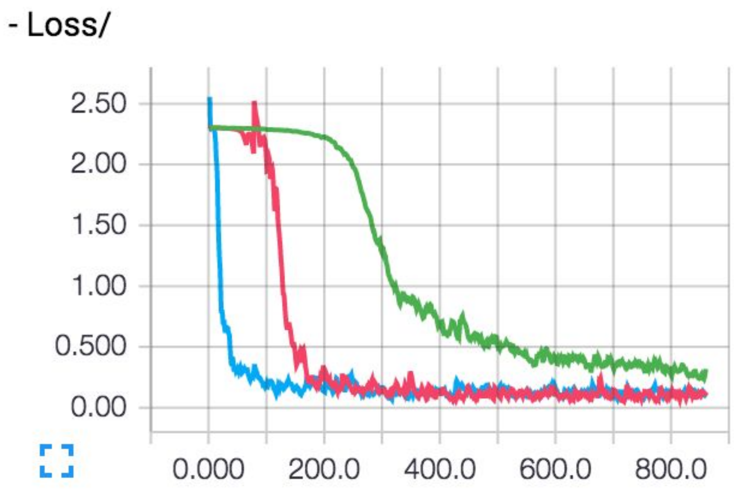


Advanced Topics

Tensorboard for Convenient Visualization

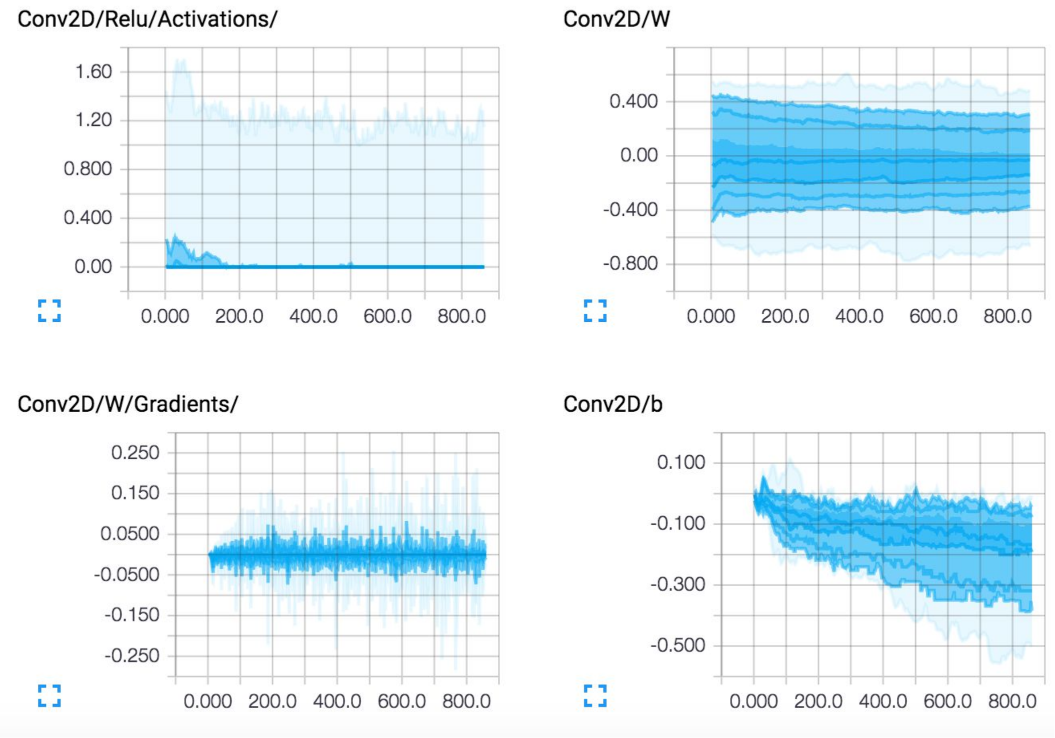


CONT'D


- Loss/
2.50
2.00
1.50
1.00
0.500
0.00
0.000
200.0
400.0
600.0
800.0


- Accuracy/
0.900
0.700
0.500
0.300
0.100
0.000
200.0
400.0
600.0
800.0

CONT’D


Conv2D/Relu/Activations/
1.60
1.20
0.800
0.400
0.00
0.000 200.0 400.0 600.0 800.0
Conv2D/W
0.400
0.00
-0.400
-0.800
0.000 200.0 400.0 600.0 800.0
Conv2D/W/Gradients/
0.250
0.150
0.0500
-0.0500
-0.150
-0.250
0.000 200.0 400.0 600.0 800.0
Conv2D/b
0.100
-0.100
-0.300
-0.500
0.000 200.0 400.0 600.0 800.0

Model Persistence

tf.train.Saver

Usage:

- See the official document for details

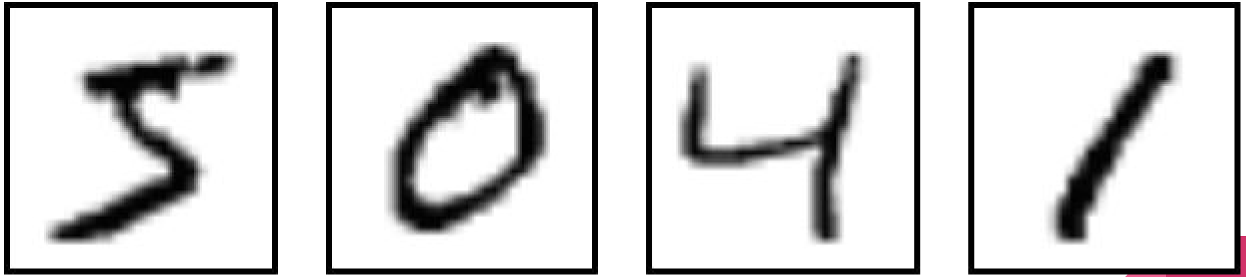
High Level API for fast prototyping

- tf.learn (a.k.a. skflow)
- Simple neural network in one line
- Under development now
- See the official document for details

Handwritten Digit Recognition
An End-to-end Example

The MNIST Dataset

- Dataset of images of handwritten digits
- Given a 28 × 28 image, predict which digit it is



Step 0: Download Dataset

- Ignore because irrelevant to our topic

- Training set: $55,000 \times 784$
- Validation set: $5,000 \times 784$
- Test set: $10,000 \times 784$

CONT'D

- Visualize input data

```
# visualize data
def display(img):
    one_image = img.reshape(IMAGE_WIDTH, IMAGE_HEIGHT)
    plt.axis('off')
    plt.imshow(one_image, cmap = cm.binary)

display(mnist.test._images[0])
```

Step 1: Define Customized Layers

```
def fc(input, output_dim, stddev = 0.1, bias = 0.1):
    input_dim = input.get_shape().as_list()[1]
    w = tf.Variable(tf.truncated_normal([input_dim, output_dim], stddev = stddev))
    b = tf.Variable(tf.constant(bias, shape = [output_dim]))
    net = tf.matmul(input, w) + b
    net = tf.nn.relu(net)
    return net
```

Using high level API of tf.learn is more convenient but less flexible

Step 2: Construct the Network

```
# define placeholders
x = tf.placeholder(tf.float32, shape = [None, IMAGE_SIZE])
y_ = tf.placeholder(tf.int64, shape = [None])
keep_prob = tf.placeholder(tf.float32)
```

```
# define network
net = tf.reshape(x, [-1, IMAGE_WIDTH, IMAGE_HEIGHT, 1])
conv1 = net = conv(net, width = 5, height = 5, channels = 32)
pool1 = net = pool(net)
conv2 = net = conv(net, width = 5, height = 5, channels = 64)
pool2 = net = pool(net)
flat = net = flatten(net)
fc1 = net = fc(net, 512)
drop = net = tf.nn.dropout(net, keep_prob)
fc2 = net = fc(net, 10)
logits = fc2

loss = tf.nn.sparse_softmax_cross_entropy_with_logits(logits, y_)
accuracy = tf.reduce_mean(tf.cast(tf.equal(tf.argmax(logits, 1), y_), tf.float32))

train_op = tf.train.AdamOptimizer(1e-4).minimize(loss)
```

Step 3: Training

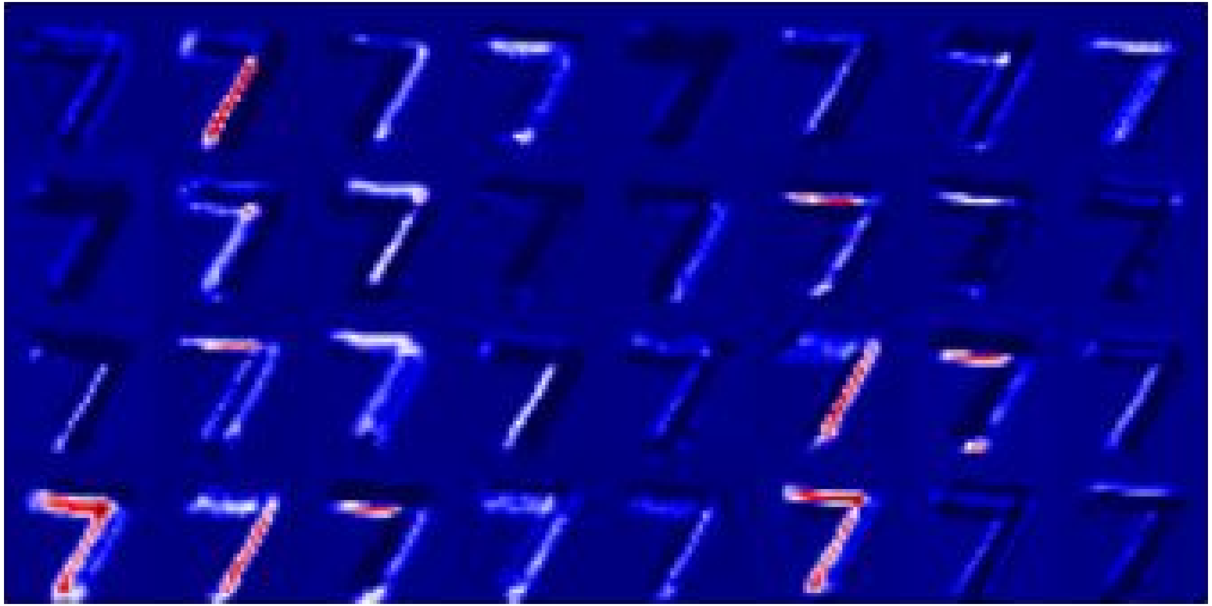
```
# start a session
sess = tf.InteractiveSession()

# initialize variables
sess.run(tf.initialize_all_variables())

for step in xrange(1, 1 + int(NUM_EPOCHS * mnist.train.num_examples // BATCH_SIZE)):
    batch_xs, batch_ys = mnist.train.next_batch(BATCH_SIZE)
    # one single train step
    sess.run(train_op, {x: batch_xs, y_: batch_ys, keep_prob: 0.5})
```

* 99.37% accuracy in test set

More: Visualize Filters



Appendix

A. Resources

- Tensorflow official API
  - https://www.tensorflow.org/api_docs/python/index.html
- Tensorflow official tutorial
  - https://www.tensorflow.org/tutorials/index.html
- My complete code of this example (in Jupyter Notebook)
  - https://nbviewer.jupyter.org/github/yzgysjr/TFTutorial/blob/master/mnist.ipynb
- My email
  - yz_sjr@sjtu.edu.cn

B. References

[1] Abadi, Martın, et al. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems." arXiv preprint arXiv:1603.04467 (2016).

[2] Hu. "Brief Torch Tutorial." In class presentation (7/18/2016).

[3] LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998): 2278-2324.

[4] Kliavin. "Tensorflow Deep NN." Kaggle Forum. Retrieved on 7/17/2016. https://www.kaggle.com/kakauandme/digit-recognizer/tensorflow-deep-nn.

Thank you for listening !
Questions are welcome !