TensorFlow tutorials

TA: 張家仁(Jia-Ren Chang)

followwar@gmail.com

EC546 BSPLAB

Deep Learning Framework

- C++
  - Caffe
- Python
  - TensorFlow, Theano, Keras, MXNet, pyTorch
- Lua
  - Torch7
- Matlab
  - MatConvNet

Tensor Flow

- Tensor: n-dimensional arrays
  - Vector: 1-D tensor
  - Matrix: 2-D tensor

- Deep learning processes are flows of tensors

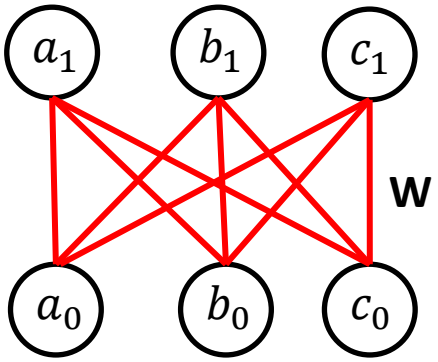
TensorFlow

- Graph: A TensorFlow computation, represented as a dataflow graph
- Op.: data operation
- Session: A class for running TensorFlow operations
- Tensor: data, n-dim arrays
- Variable: A variable maintains state in the graph across calls to run().
- Feed: lets you inject data into any Tensor in a computation graph.
- Fetch: lets you get data out of any Tensor in a computation graph.

TensorFlow

1. Building Graph: create operations
2. Define Variable
3. Initialize Variable
4. Open session, and run Graph

A simple ReLU network



$$a_1 = \mathrm{ReLU}(a_0 w_{a,a} + b_0 w_{b,a} + c_0 w_{c,a})$$

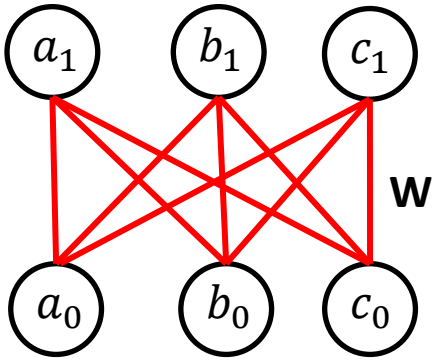
$$b_1 = \mathrm{ReLU}(a_0 w_{a,b} + b_0 w_{b,b} + c_0 w_{c,b})$$

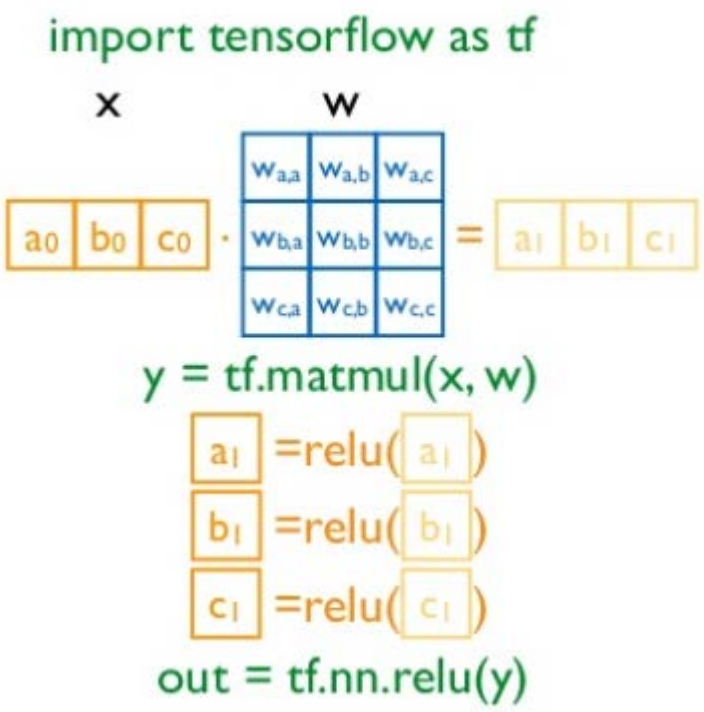
$$c_1 = \mathrm{ReLU}(a_0 w_{a,c} + b_0 w_{b,c} + c_0 w_{c,c})$$

where ReLU(x) = max(x,0)

Matrix operation

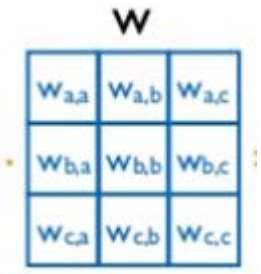
Create operation





Define Variable

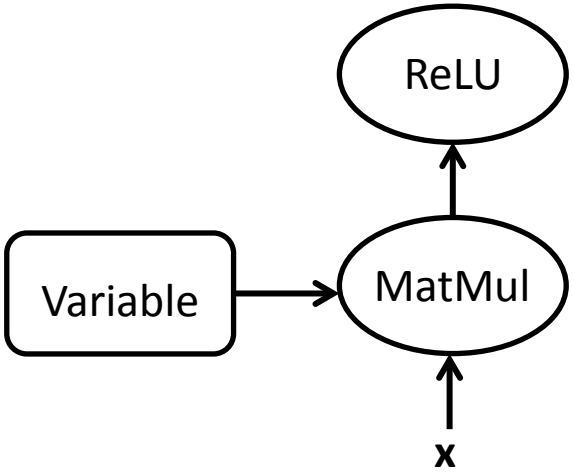
- Variable(<initial-value>,name=<optional-name>)




```
import tensorflow as tf
w = tf.Variable(tf.random_normal([3,3]), name='w')
y = tf.matmul(x,w)
out = tf.nn.relu(y)
```

Define Graph

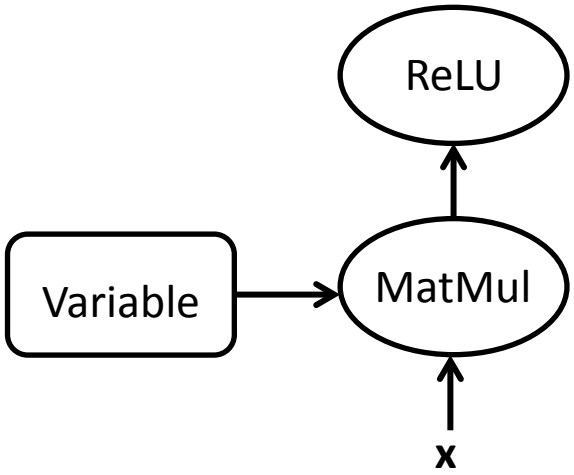
- Code defines a data flow graph




```
import tensorflow as tf
w = tf.Variable(tf.random_normal([3,3]), name='w')
y = tf.matmul(x,w)
out = tf.nn.relu(y)
```

Session

- Session: manage resource for graph execution

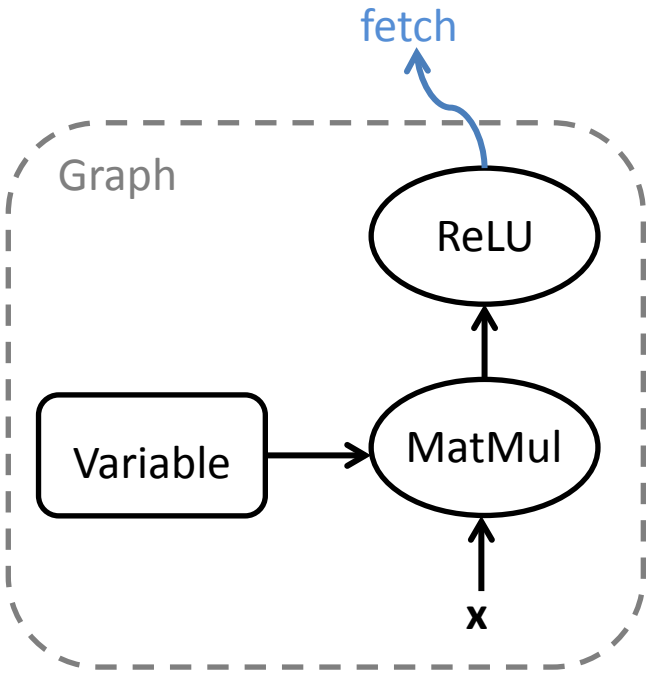



```
import tensorflow as tf
sess = tf.Session()
w = tf.Variable(tf.random_normal([3,3]), name='w')
y = tf.matmul(x,w)
out = tf.nn.relu(y)
result = sess.run(out)
```

```
Sess = tf.InteractiveSession()
For interactive mode
```

Fetch

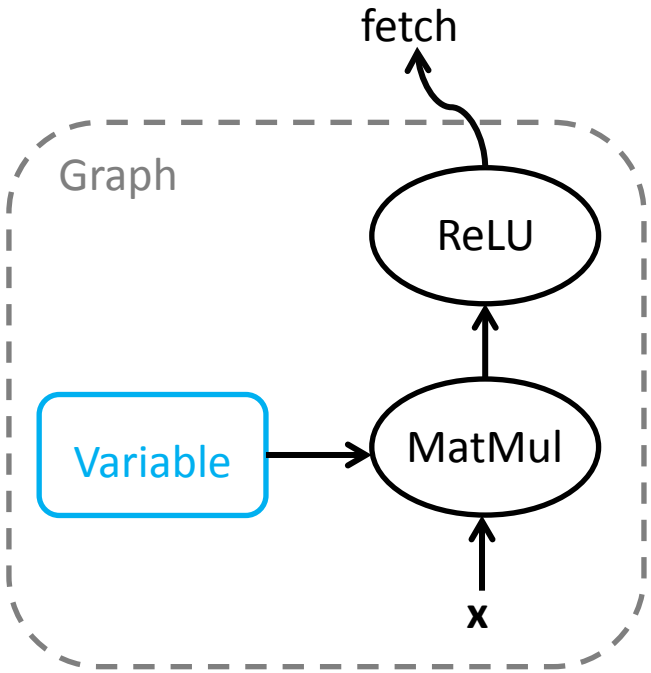
- Retrieve content from a node




```
import tensorflow as tf
sess = tf.Session()
w = tf.Variable(tf.random_normal([3,3]), name='w')
y = tf.matmul(x,w)
out = tf.nn.relu(y)
print sess.run(out)
```

Initialize Variable

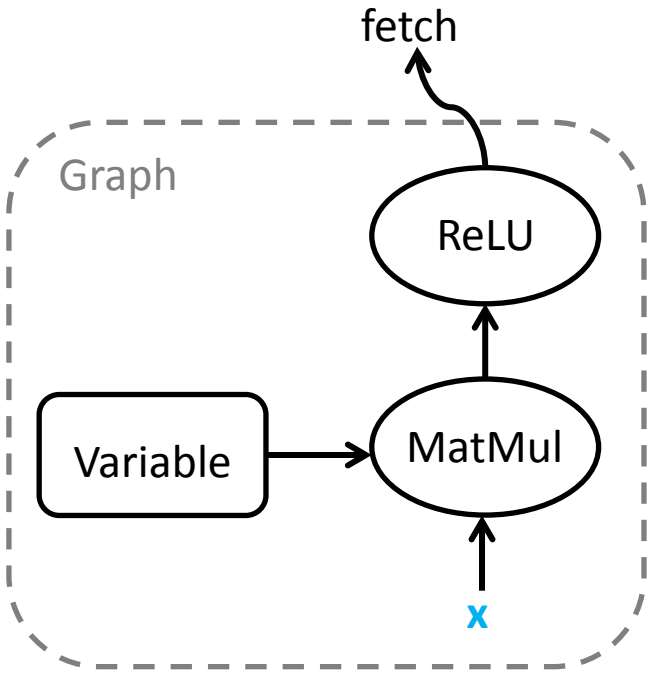
- Variable is an empty node
  - Fill in the content of a variable node




```
import tensorflow as tf
sess = tf.Session()
w = tf.Variable(tf.random_normal([3,3]), name='w')
y = tf.matmul(x,w)
out = tf.nn.relu(y)
 sess.run(tf.global_variables_initializer())
print sess.run(out)
```

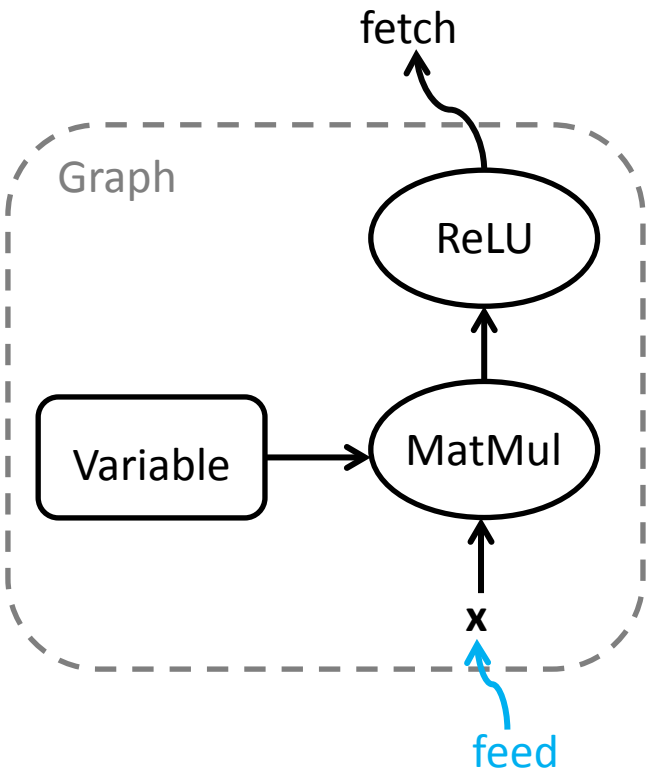
Placeholder

- placeholder(<data type>,shape=<optional-shape>,name=<optional-name>)




```
import tensorflow as tf
sess = tf.Session()
x = tf.placeholder("float",[1,3])
w = tf.Variable(tf.random_normal([3,3]), name='w')
y = tf.matmul(x,w)
out = tf.nn.relu(y)
sess.run(tf.global_variables_initializer())
print sess.run(out)
```

Feed



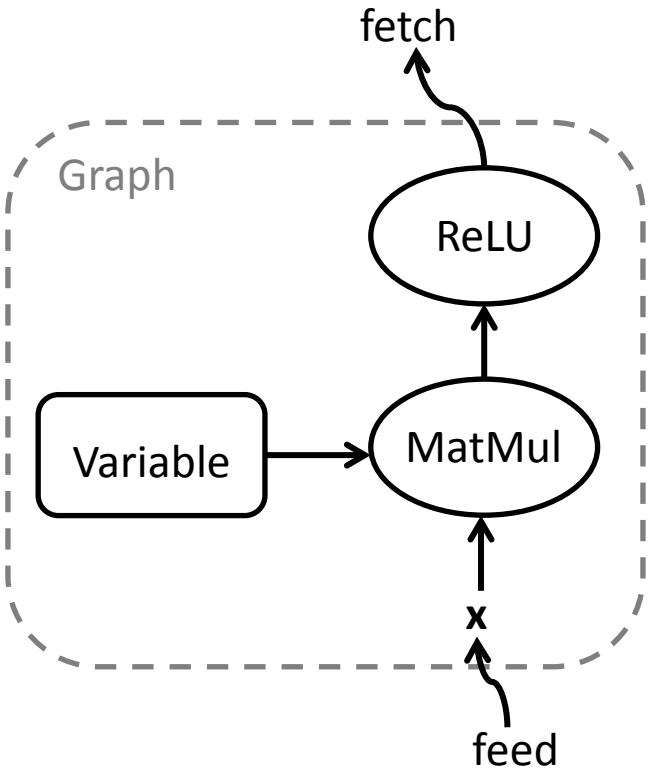

```
import numpy as np
import tensorflow as tf
sess = tf.Session()
x = tf.placeholder("float",[1,3])
w = tf.Variable(tf.random_normal([3,3]), name='w')
y = tf.matmul(x,w)
out = tf.nn.relu(y)
sess.run(tf.global_variables_initializer())
print sess.run(out, feed_dict={x:np.array([[1,2,3]])})
```

Session management

- Need to release resources after use

  `sess.close()`

- Common usage

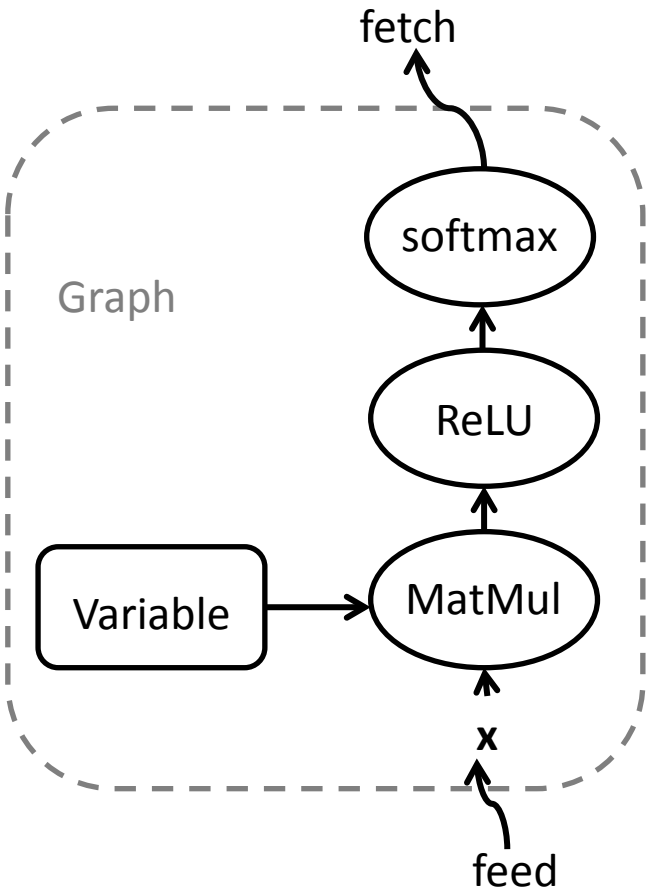



```
import numpy as np
import tensorflow as tf

with tf.Session() as sess:
  x = tf.placeholder("float",[1,3])
  w = tf.Variable(tf.random_normal([3,3]), name='w')
  y = tf.matmul(x,w)
  out = tf.nn.relu(y)
 sess.run(tf.global_variables_initializer())
  print sess.run(out, feed_dict={x:np.array([[1,2,3]])})
```

Prediction

- Softmax $\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$ for $j = 1, \ldots, K$.




```
import numpy as np
import tensorflow as tf

with tf.Session() as sess:
  x = tf.placeholder("float",[1,3])
  w = tf.Variable(tf.random_normal([3,3]), name='w')
  y = tf.matmul(x,w)
  out = tf.nn.relu(y)
  softmax = tf.nn.softmax(out)
  sess.run(tf.global_variables_initializer())
  print sess.run(softmax, feed_dict={x:np.array([[1,2,3]])})
```

Loss function

- Define loss function
  - Cross entropy

$$\mathcal{L}(X,Y) = -\frac{1}{n}\sum_{i=1}^{n} y^{(i)} \ln a(x^{(i)}) + \left(1-y^{(i)}\right) \ln\left(1-a(x^{(i)})\right)$$

```
import numpy as np
import tensorflow as tf

with tf.Session() as sess:
 x = tf.placeholder("float",[1,3])
 w = tf.Variable(tf.random_normal([3,3]), name='w')
 y = tf.matmul(x,w)
 out = tf.nn.relu(y)
 labels = tf.placeholder("float",[1,3])
 answer = np.array([[0,1,0]])
 cross_entropy = tf.nn.softmax_cross_entropy_with_logits(logits=out, labels=labels)

 sess.run(tf.global_variables_initializer())
  sess.run(softmax , feed_dict={x:np.array([[1,2,3]]), labels:answer})
```

Optimization

- Gradient descent
  - GradientDescentOptimizer(learning rate)

```
import numpy as np
import tensorflow as tf

with tf.Session() as sess:
 x = tf.placeholder("float",[1,3])
 w = tf.Variable(tf.random_normal([3,3]), name='w')
 y = tf.matmul(x,w)
 out = tf.nn.relu(y)
 labels = tf.placeholder("float",[1,3])
 answer = np.array([[0,1,0]])
 cross_entropy = tf.nn.softmax_cross_entropy_with_logits(out, labels, name='xen')
 sess.run(tf.global_variables_initializer())
  train_op = tf.train.GradientDescentOptimizer(0.1).minimize(cross_entropy)
  sess.run(train_op , feed_dict={x:np.array([[1,2,3]]), labels:answer})
```

Iterative update

```
import numpy as np
import tensorflow as tf
x = tf.placeholder('float',[1,3])
w = tf.Variable(tf.random_normal([3,3]), name='w')
y = tf.matmul(x,w)
out = tf.nn.relu(y)
labels = tf.placeholder('float',[1,3])
answer = np.array([[0,1,0]])

cross_entropy = tf.nn.softmax_cross_entropy_with_logits(logits=out, labels=labels)
train_op = tf.train.GradientDescentOptimizer(0.1).minimize(cross_entropy)

with tf.Session() as sess:
 sess.run(tf.global_variables_initializer())
 for step in range(10):
  sess.run(train_op , feed_dict={x:np.array([[1,2,3]]), labels:answer})
```

convolution

- Conv2d(input, filter, strides, padding, use_cudnn_on_gpu=None, name=None)

MNIST

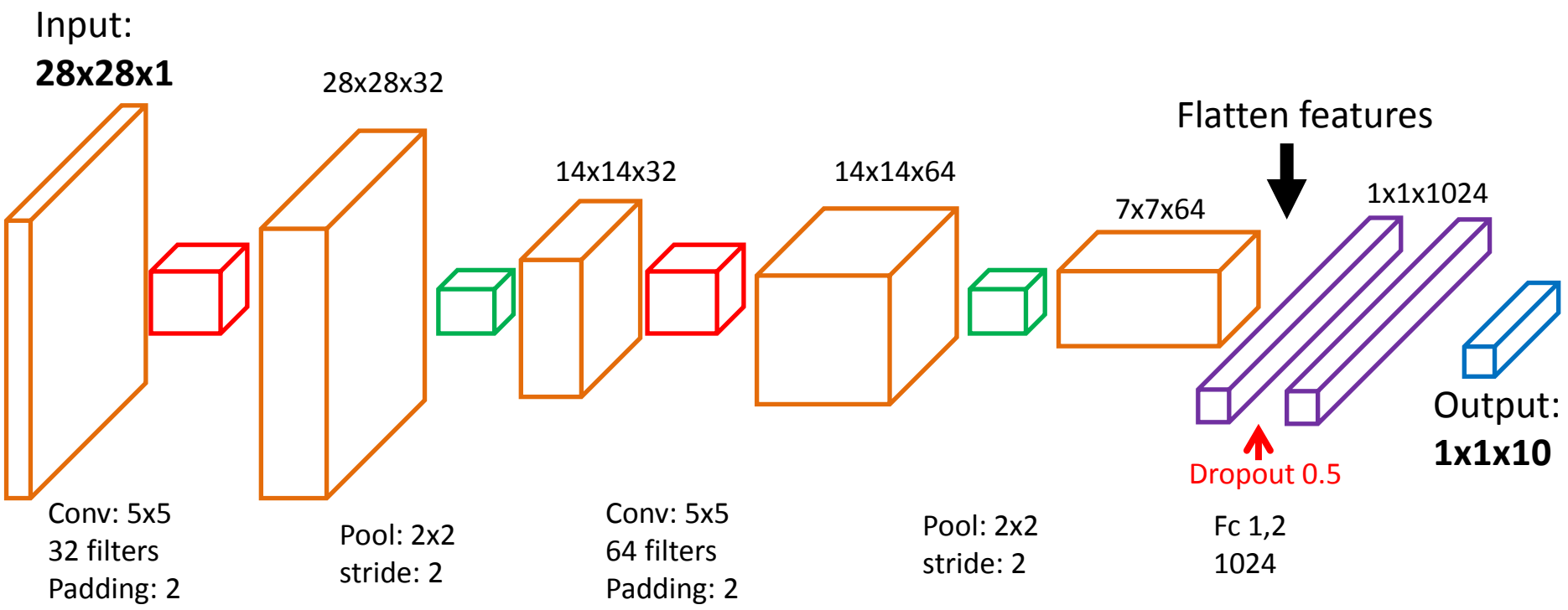
- 60000 28×28 grey images for training, 10000 for testing

First CNN

https://www.tensorflow.org/versions/master/tutorials/mnist/pros/#deep-mnist-for-experts

- Build a Multilayer Convolutional Network
- 2 convolutional layers
- 2 fully connected layers

First CNN


Input:
28x28x1
28x28x32
14x14x32
14x14x64
7x7x64
Flatten features
1x1x1024
Output:
1x1x10
Dropout 0.5
Conv: 5x5
32 filters
Padding: 2
Pool: 2x2
stride: 2
Conv: 5x5
64 filters
Padding: 2
Pool: 2x2
stride: 2
Fc 1,2
1024

First CNN

- How many parameters in this model?
- 1st conv: 5x5x1x32 + 32 (bias) = 832
- 2nd conv: 5x5x32x64 + 64 = 51,200
- 3rd fc: 7x7x64x1024 + 1024 = 3,212,288
- 4th fc: 1x1x1024x10 +10 = 10,250

- Total: 3,274,570

Git the example code

- Execute command
- *git clone https://github.com/JiaRenChang/DLcourse_NCTU.git*

Run this example

cd DLcourse_NCTU

python CNN_mnist.py

CNN code

- Import functions and data

```
from __future__ import print_function
import tensorflow as tf

#use others' data loader
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets('MNIST_data', one_hot=True)
```

CNN code

- Define functions

```
12 # define functions
13 def weight_variable(shape):
14     initial = tf.truncated_normal(shape, stddev=0.1)
15     return tf.Variable(initial)
16 
17 def bias_variable(shape):
18     initial = tf.constant(0.1, shape=shape)
19     return tf.Variable(initial)
20 
21 def conv2d(x, W):
22     return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')
23 
24 def max_pool_2x2(x):
25     return tf.nn.max_pool(x, ksize=[1,2,2,1], strides=[1,2,2,1], padding='SAME')
```

CNN code

- Define placeholder

```
27 #define placeholder
28 x = tf.placeholder(tf.float32, [None, 784])
29 y_ = tf.placeholder(tf.float32, [None, 10])
30 keep_prob = tf.placeholder(tf.float32)#Dropout rate
31 x_image = tf.reshape(x, [-1, 28, 28, 1]) # resize back to 28 x 28
```

ConvNet

```
34 ## conv1 layer ##
35 W_conv1 = weight_variable([5,5, 1,32]) # patch 5x5, in size 1, out size 32
36 b_conv1 = bias_variable([32])
37 h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1) # output size 28x28x32
38 h_pool1 = max_pool_2x2(h_conv1)                                    # output size 14x14x32
39
40 ## conv2 layer ##
41 W_conv2 = weight_variable([5,5, 32, 64]) # patch 5x5, in size 32, out size 64
42 b_conv2 = bias_variable([64])
43 h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2) # output size 14x14x64
44 h_pool2 = max_pool_2x2(h_conv2)     # output size 7x7x64
45
46 ## fc1 layer ##
47 W_fc1 = weight_variable([7*7*64, 1024])
48 b_fc1 = bias_variable([1024])
49 # [n_samples, 7, 7, 64] ->> [n_samples, 7*7*64]
50 h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
51 h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)
52
53 ## dropout ##
54 h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
55
56 ## fc2 layer ##
57 W_fc2 = weight_variable([1024, 10])
58 b_fc2 = bias_variable([10])
59 y_fc = tf.matmul(h_fc1_drop, W_fc2) + b_fc2
```

Learning rate

```
62 ## the learning rate
63 # learning rate 0.1 for first 80 epoch (469 iteration for 1 epoch)
64 # decay learning rate to 0.01 at 81th epoch
65 # decay learning rate to 0.01 at 121th epoch
66 global_step = tf.Variable(0, trainable=False)
67 boundaries = [37520, 56280]
68 values = [0.1, 0.01, 0.001]
69 learning_rate = tf.train.piecewise_constant(global_step, boundaries, values)
```

Loss function

```
73 # the loss function
74 cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y_fc))
75
76 # optimizer SGD
77 train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(cross_entropy)
78
79 # prediction
80 correct_prediction = tf.equal(tf.argmax(y_fc,1), tf.argmax(y_,1))
81 accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

Start session

```
83 # start session
84 sess = tf.InteractiveSession()
85 # initize variable
86 sess.run(tf.global_variables_initializer())
87
88 for i in range(76916): # 164 epoch for all (496*164)
89   batch = mnist.train.next_batch(128)
90   ## test every 100 step
91   if i%100 == 0:
92     print("step %d, Test accuracy %g"%(i, (accuracy.eval(feed_dict={
93         x:mnist.test.images,y_: mnist.test.labels, keep_prob: 1.0}))))
94   ## trianing
95   train_step.run(feed_dict={x: batch[0], y_: batch[1], keep_prob: 0.5})
96
97 ##final testing
98 print("Final test accuracy %g"%accuracy.eval(feed_dict={
99     x: mnist.test.images, y_: mnist.test.labels, keep_prob: 1.0}))
100
```

It’s should be ~99.28% test accuracy