TensorFlow
- MNIST example -

2019 – 2020

Ando Ki, Ph.D.
adki@future-ds.com

Table of contents

- Get TensorFlow packages
- MNIST using MLP
- Project: MNIST using one hidden layer
- MNIST using CNN

Get TensorFlow packages

- This package is required to utilize MNIST dataset.
- Go to the project directory

```
$ cd $(PROJECT)/codes
```

- Get a copy of TensorFlow package

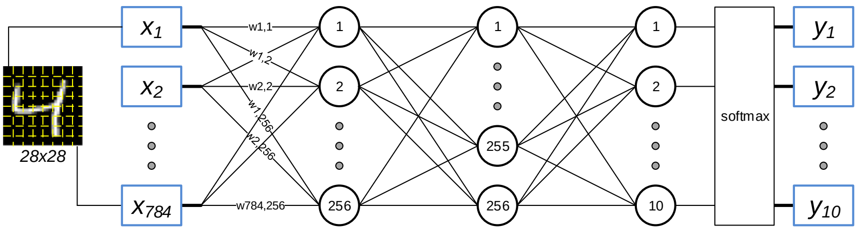
```
$ git clone https://github.com/tensorflow/tensorflow.git
```

  - ▶ or visit following site and get a copy of it
    - ➲ check directory hierarchy and its name; modify if necessary

```
https://github.com/tensorflow/tensorflow
```

MNIST using MLP

- MLP (Multi-Layer Perceptron) applies to MNIST
  - ▶ 784 (28x28) inputs of black and white ➔ converted to floating number 0.0 ~ 1.0
  - ▶ 10 outputs representing digit 0 to 9
  - ▶ Two hidden layer
  - ▶ 256 features for each hidden layer

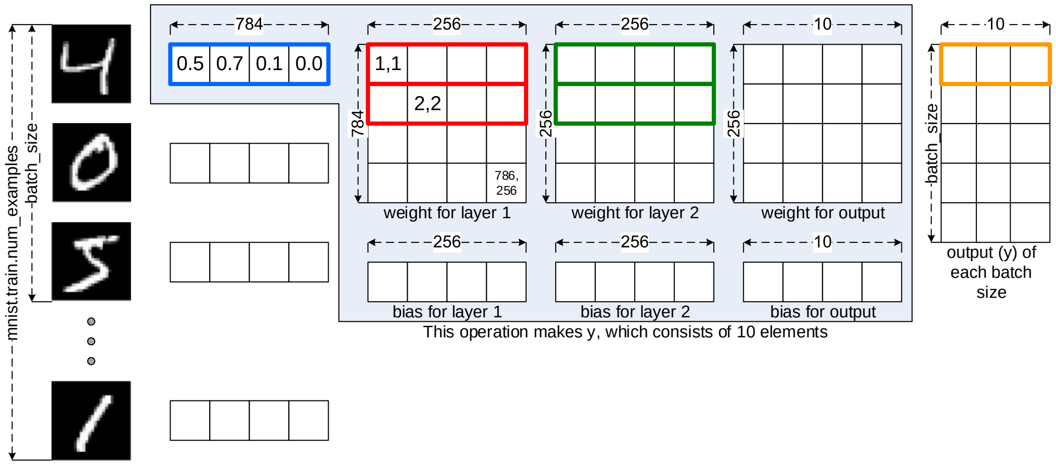


$(PROJECT)/codes/tensorflow-projects/mnist-projects/mlp

MNIST using MLP

- Total testing patterns
  - 60K images (train patterns: 55K images)
- Batch number
  - 100 ➔ group 600 batches (train batches: 550)
- Epoch number
  - The number of whole trainings (forward calculation and back propagation)

MNIST using MLP

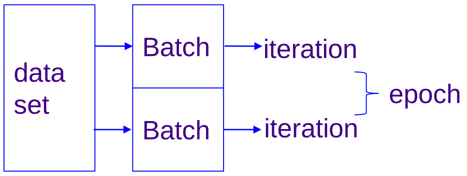


This operation makes y, which consists of 10 elements

Terminology

- Iteration
  - Forward and backward for a number of inputs (i.e., batch or minibatch)
- Batch or minibatch
  - A number of inputs (e.g., testing images) to complete an iteration
  - Batch size: the number of training examples in one forward/backward pass
    - The bigger the batch size, the more memory space needed
  - Minibatch
    - Take a small number of examples at a time, ranging from 1 to a few hundred, during one iteration
- Epoch
  - one epoch: one forward pass and one backward pass of all the training examples
  - it contains a number of iterations

- 1 epoch = (number of iterations)
- = (total training examples) /
- (batch size or minibatch size)



TensorFlow oprators

- tf.placeholder()
- tf.Variable()
- tf.global_variables_initilizer()
- tf.add()
- tf.matmul()
- tf.reduce_mean()
- tf.nn.relu()
- tf.train.AdamOptimizer()
- tf.train.GradientDescentOptimizer()
- tf.argmax()

MNIST using MLP

$(PROJECT)/codes/tensorflow-projects/mnist-projects/mlp/mnist_mlp.py

```
#-----------------------------------------------------------------------
import os
import sys
os.environ['TF_CPP_MIN_LOG_LEVEL']='2'
sys.path.append(os.path.dirname("../../../tensorflow/tensorflow"))
#-----------------------------------------------------------------------
import tensorflow as tf
import tensorflow as tf
import numpy as np
from random import randint
import matplotlib.pyplot as plt
#-----------------------------------------------------------------------
# Import MNIST data
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("../dataset", one_hot=True)
#-----------------------------------------------------------------------
# Parameters
learning_rate = 0.01
training_epochs = 1000
batch_size = 100
display_step = 10
```

Depress warnings

Path where to find module

TensorFlow module

MNIST dataset handling module

Read MNIST dataset

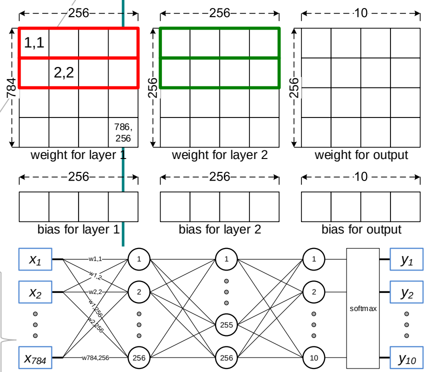
Parameters

MNIST using MLP

```
#-----------------------------------------------------------------------
# tf Graph Input
x = tf.placeholder("float", [None, 784], name="x-input")
y = tf.placeholder("float", [None, 10], name="y-output")

#-----------------------------------------------------------------------
# Model weights
W1 = tf.Variable(tf.random_normal([784,256]), name="weight1")
b1 = tf.Variable(tf.random_normal([256]), name="bias1")
W2 = tf.Variable(tf.random_normal([256,256]), name="weight2")
b2 = tf.Variable(tf.random_normal([256]), name="bias2")
W3 = tf.Variable(tf.random_normal([256,10]), name="weight-out")
b3 = tf.Variable(tf.random_normal([10]), name="bias-out")

#-----------------------------------------------------------------------
# inference -> hypothesis
layer_1 = tf.add(tf.matmul(x, W1), b1)
layer_1 = tf.nn.sigmoid(layer_1, name="layer1-sigmoid")
layer_2 = tf.add(tf.matmul(layer_1, W2), b2)
layer_2 = tf.nn.sigmoid(layer_2, name="layer2-sigmoid")
out_layer = tf.add(tf.matmul(layer_2, W3), b3)
infer = tf.nn.softmax(out_layer, name="infer")
```

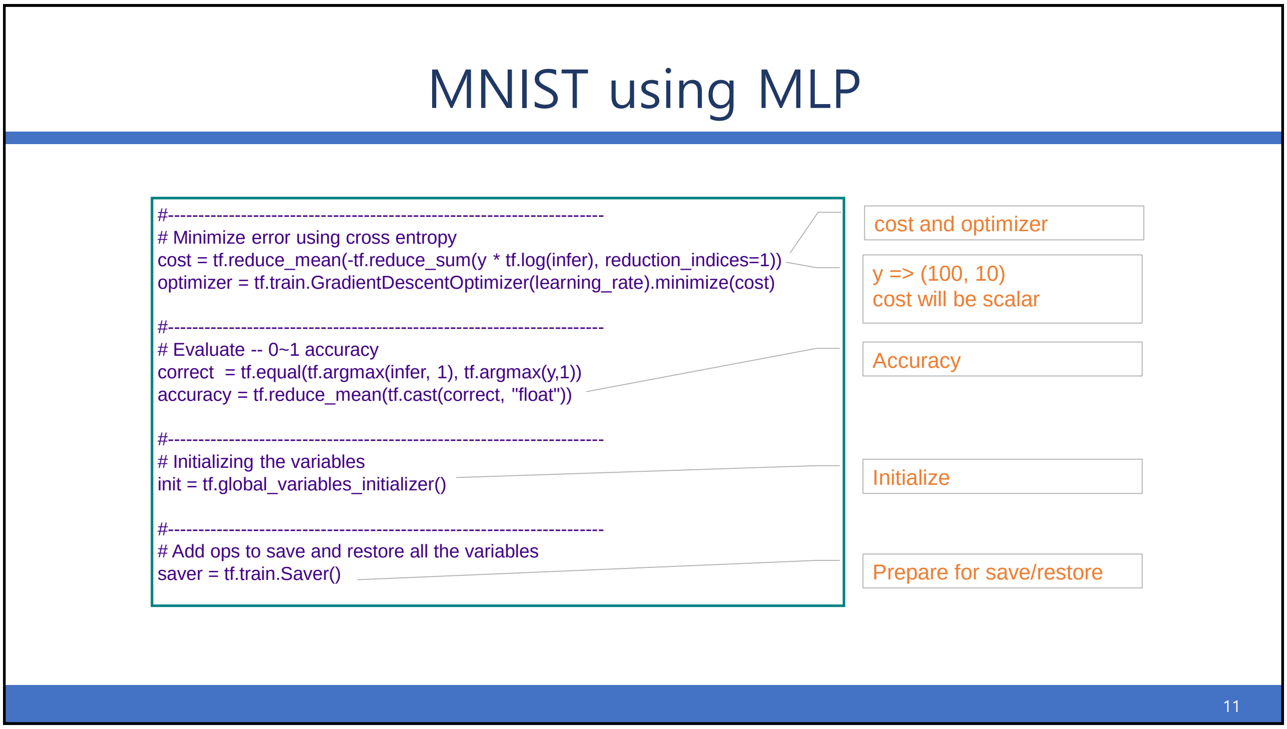
Input and output

Network parameters



Network building

Note softmax



MNIST using MLP

```
#------------------------------------------------------------------------
# Minimize error using cross entropy
cost = tf.reduce_mean(-tf.reduce_sum(y * tf.log(infer), reduction_indices=1))
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

#------------------------------------------------------------------------
# Evaluate -- 0~1 accuracy
correct  = tf.equal(tf.argmax(infer, 1), tf.argmax(y,1))
accuracy = tf.reduce_mean(tf.cast(correct, "float"))

#------------------------------------------------------------------------
# Initializing the variables
init = tf.global_variables_initializer()

#------------------------------------------------------------------------
# Add ops to save and restore all the variables
saver = tf.train.Saver()
```

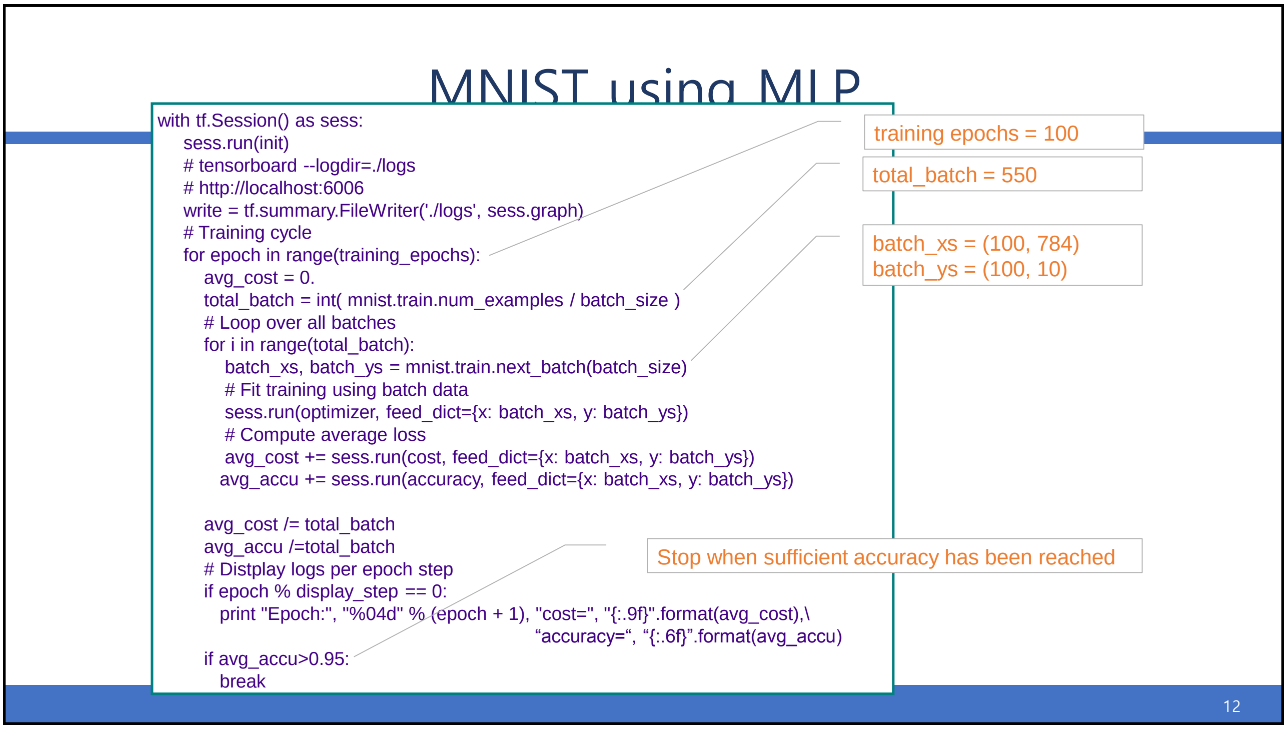
cost and optimizer

y => (100, 10)
cost will be scalar

Accuracy

Initialize

Prepare for save/restore



MNIST using MLP

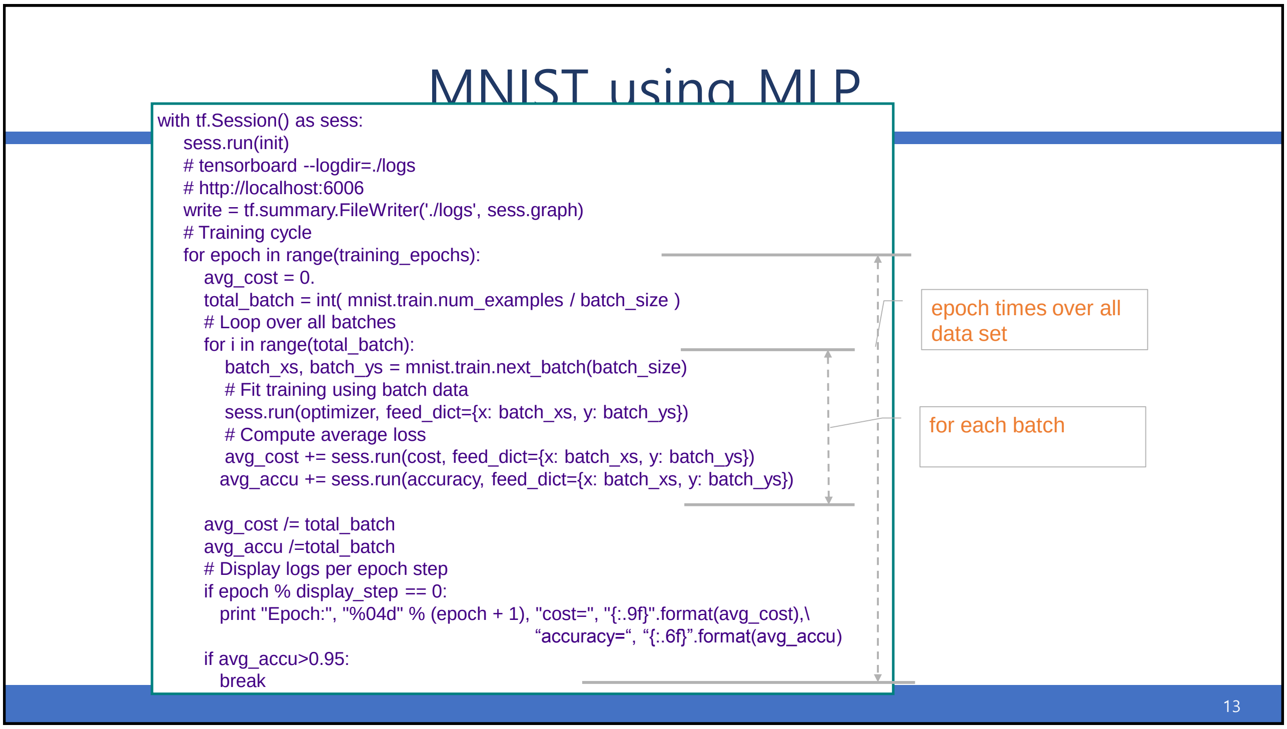
```
with tf.Session() as sess:
    sess.run(init)
    # tensorboard --logdir=./logs
    # http://localhost:6006
    write = tf.summary.FileWriter('./logs', sess.graph)
    # Training cycle
    for epoch in range(training_epochs):
       avg_cost = 0.
       total_batch = int( mnist.train.num_examples / batch_size )
       # Loop over all batches
       for i in range(total_batch):
          batch_xs, batch_ys = mnist.train.next_batch(batch_size)
          # Fit training using batch data
          sess.run(optimizer, feed_dict={x: batch_xs, y: batch_ys})
          # Compute average loss
          avg_cost += sess.run(cost, feed_dict={x: batch_xs, y: batch_ys})
         avg_accu += sess.run(accuracy, feed_dict={x: batch_xs, y: batch_ys})

       avg_cost /= total_batch
       avg_accu /=total_batch
       # Distplay logs per epoch step
       if epoch % display_step == 0:
         print "Epoch:", "%04d" % (epoch + 1), "cost=", "{:.9f}".format(avg_cost),\
                                                          "accuracy=", "{:.6f}".format(avg_accu)
       if avg_accu>0.95:
         break
```

training epochs = 100

total_batch = 550

batch_xs = (100, 784)
batch_ys = (100, 10)

Stop when sufficient accuracy has been reached

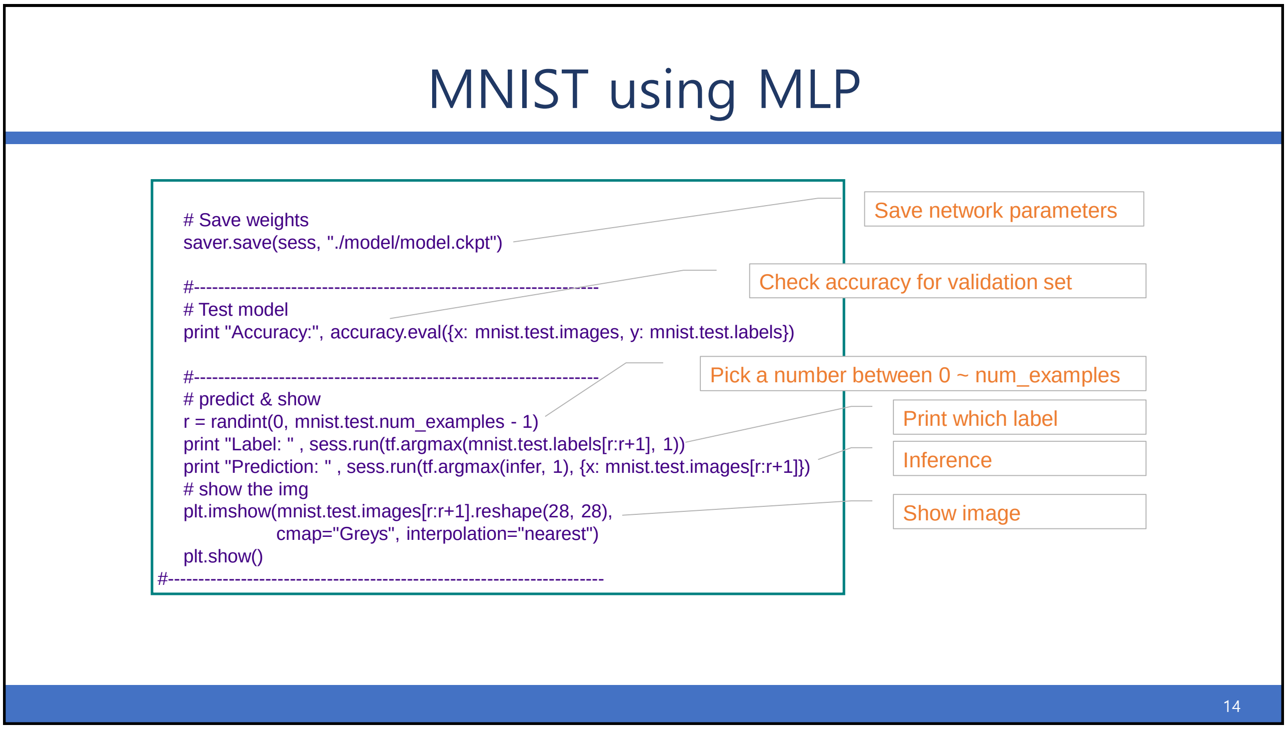


MNIST using MLP

```
with tf.Session() as sess:
    sess.run(init)
    # tensorboard --logdir=./logs
    # http://localhost:6006
    write = tf.summary.FileWriter('./logs', sess.graph)
    # Training cycle
    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_xs, batch_ys = mnist.train.next_batch(batch_size)
            # Fit training using batch data
            sess.run(optimizer, feed_dict={x: batch_xs, y: batch_ys})
            # Compute average loss
            avg_cost += sess.run(cost, feed_dict={x: batch_xs, y: batch_ys})
           avg_accu += sess.run(accuracy, feed_dict={x: batch_xs, y: batch_ys})

        avg_cost /= total_batch
        avg_accu /=total_batch
        # Display logs per epoch step
        if epoch % display_step == 0:
           print "Epoch:", "%04d" % (epoch + 1), "cost=", "{:.9f}".format(avg_cost),\
                                                       “accuracy=“, “{:.6f}”.format(avg_accu)
        if avg_accu>0.95:
           break
```

epoch times over all data set

for each batch



MNIST using MLP

```
    # Save weights
    saver.save(sess, "./model/model.ckpt")

    #----------------------------------------------------------------------
    # Test model
    print "Accuracy:", accuracy.eval({x: mnist.test.images, y: mnist.test.labels})

    #----------------------------------------------------------------------
    # predict & show
    r = randint(0, mnist.test.num_examples - 1)
    print "Label: " , sess.run(tf.argmax(mnist.test.labels[r:r+1], 1))
    print "Prediction: " , sess.run(tf.argmax(infer, 1), {x: mnist.test.images[r:r+1]})
    # show the img
    plt.imshow(mnist.test.images[r:r+1].reshape(28, 28),
                  cmap="Greys", interpolation="nearest")
    plt.show()
#-----------------------------------------------------------------------
```

Save network parameters

Check accuracy for validation set

Pick a number between 0 ~ num_examples

Print which label

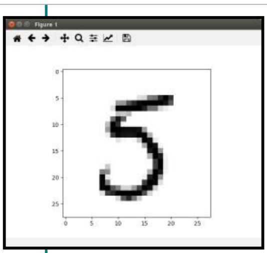
Inference

Show image

MNIST using MLP

```
(tensorflow)$ python mnist_mlp.py
Extracting ../dataset/train-images-idx3-ubyte.gz
Extracting ../dataset/train-labels-idx1-ubyte.gz
Extracting ../dataset/t10k-images-idx3-ubyte.gz
Extracting ../dataset/t10k-labels-idx1-ubyte.gz
Epoch: 0001 cost= 5.25198 accuracy= 0.30315
Epoch: 0011 cost= 0.89775 accuracy= 0.77602
Epoch: 0021 cost= 0.64263 accuracy= 0.82938
Epoch: 0031 cost= 0.52474 accuracy= 0.85504
Epoch: 0041 cost= 0.45274 accuracy= 0.87211
Epoch: 0051 cost= 0.40308 accuracy= 0.88496
Epoch: 0061 cost= 0.36560 accuracy= 0.89400
……
Epoch: 0161 cost= 0.20485 accuracy= 0.93944
Epoch: 0171 cost= 0.19650 accuracy= 0.94225
Epoch: 0181 cost= 0.18897 accuracy= 0.94424
Epoch: 0191 cost= 0.18194 accuracy= 0.94656
Epoch: 0201 cost= 0.17524 accuracy= 0.94880
Optimization finished
Accuracy: 0.9113
Label:  [5]
Prediction:  [5]
^C
(tensorflow)$ tensorboard –logdir=logs
```

When ‘matplotlib’ is missing:
$ sudo apt-get install libpng-dev
$ sudo apt-get install libfreetype6-dev
$ pip install matplotlib



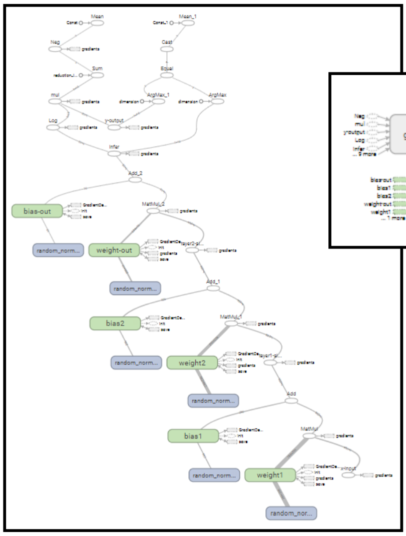
Training accuracy

Validation accuracy

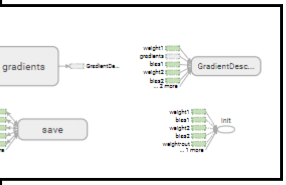
Predict correct label

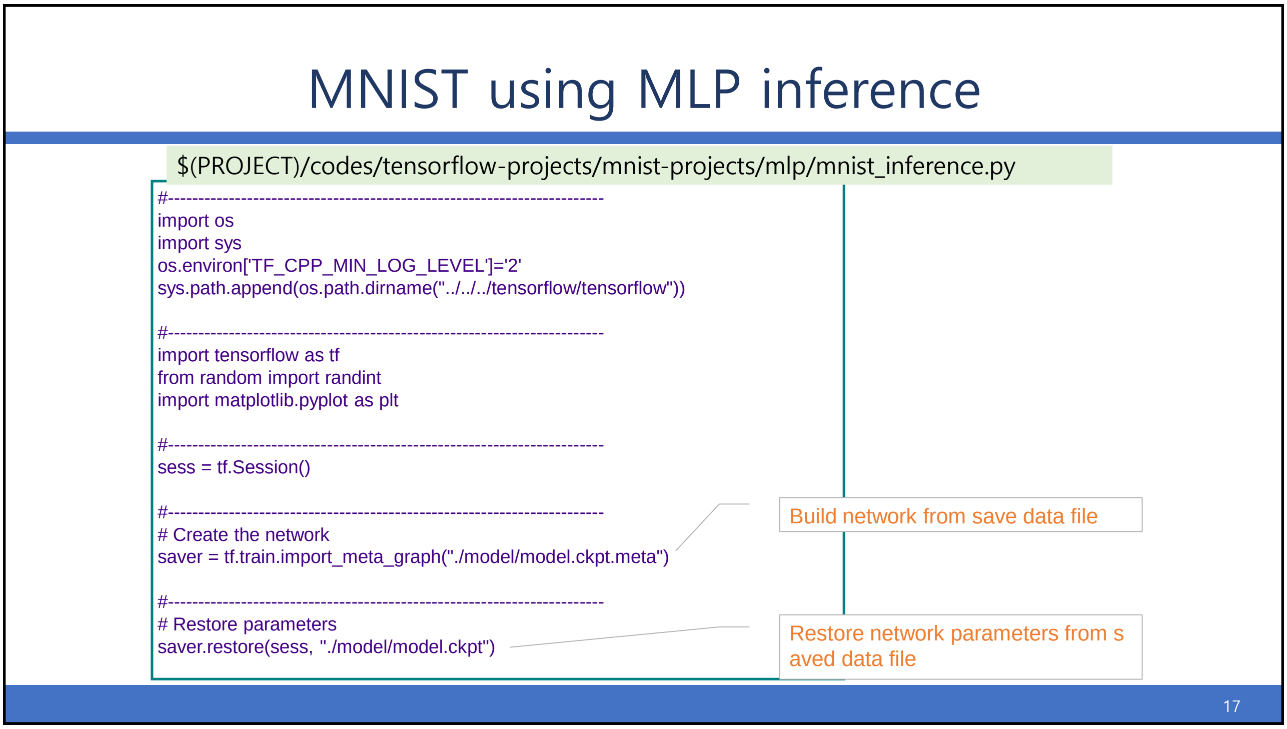
Invoke tensor board

MNIST using MLP



invoke web browser: http://localhost:6006
Then select ‘GRAPH’ menu





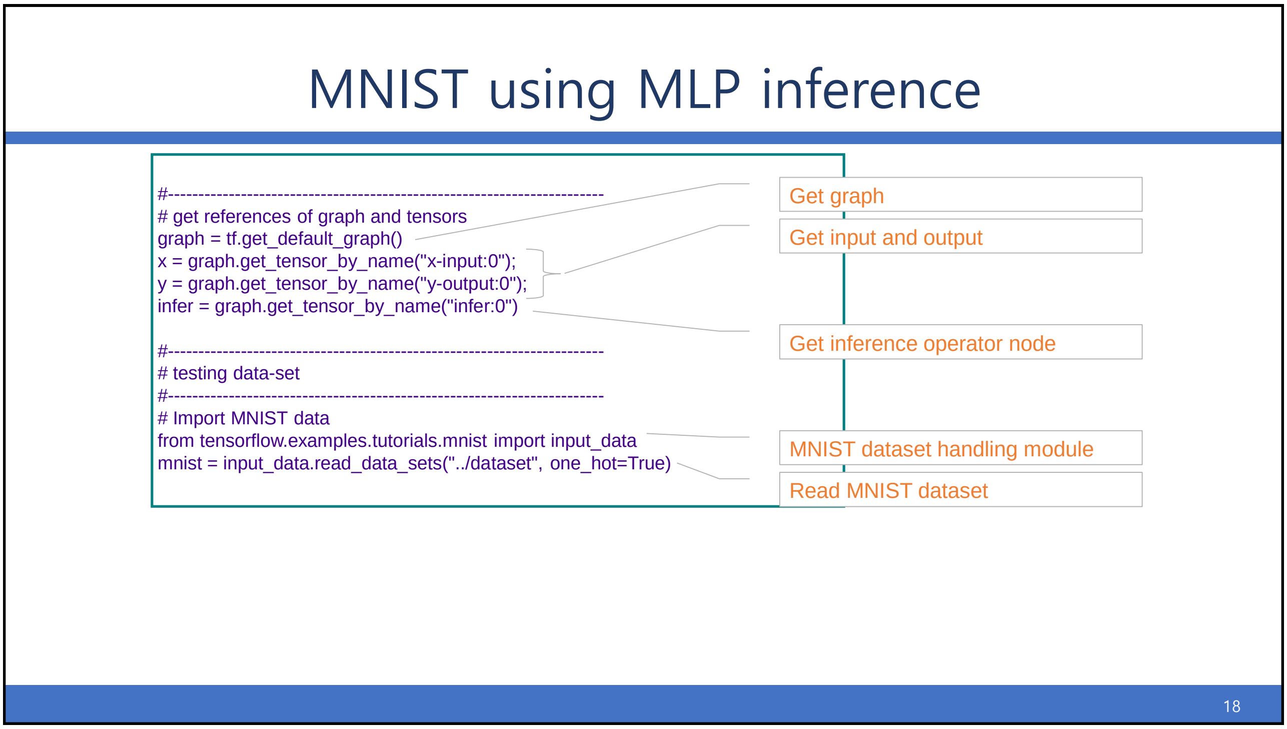
MNIST using MLP inference

$(PROJECT)/codes/tensorflow-projects/mnist-projects/mlp/mnist_inference.py

```
#------------------------------------------------------------------------
import os
import sys
os.environ['TF_CPP_MIN_LOG_LEVEL']='2'
sys.path.append(os.path.dirname("../../../tensorflow/tensorflow"))

#------------------------------------------------------------------------
import tensorflow as tf
from random import randint
import matplotlib.pyplot as plt

#------------------------------------------------------------------------
sess = tf.Session()

#------------------------------------------------------------------------
# Create the network
saver = tf.train.import_meta_graph("./model/model.ckpt.meta")

#------------------------------------------------------------------------
# Restore parameters
saver.restore(sess, "./model/model.ckpt")
```

Build network from save data file

Restore network parameters from s aved data file



MNIST using MLP inference

```
#------------------------------------------------------------------------
# get references of graph and tensors
graph = tf.get_default_graph()
x = graph.get_tensor_by_name("x-input:0");
y = graph.get_tensor_by_name("y-output:0");
infer = graph.get_tensor_by_name("infer:0")

#------------------------------------------------------------------------
# testing data-set
#------------------------------------------------------------------------
# Import MNIST data
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("../dataset", one_hot=True)
```

Get graph

Get input and output

Get inference operator node

MNIST dataset handling module

Read MNIST dataset

MNIST using MLP inference

```
#-----------------------------------------------------------------------
# predict & show
for i in range (10):
    r = randint(0, mnist.test.num_examples - 1)
    print "Label: " , sess.run(tf.argmax(mnist.test.labels[r:r+1], 1))
    print "Prediction: " , sess.run(tf.argmax(infer, 1),\
                                {x: mnist.test.images[r:r+1]})

    #-------------------------------------------------------------------
    # show the img
    plt.imshow(mnist.test.images[r:r+1].reshape(28, 28),\
               cmap="Greys", interpolation="nearest")
    plt.show()

#-----------------------------------------------------------------------
```

Test 10 patterns

Pick a number between 0 ~ num_examples

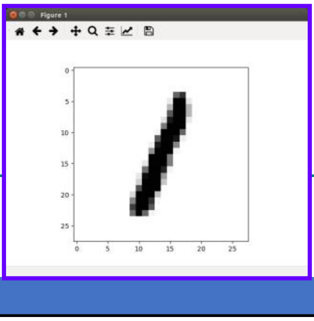
Print which label

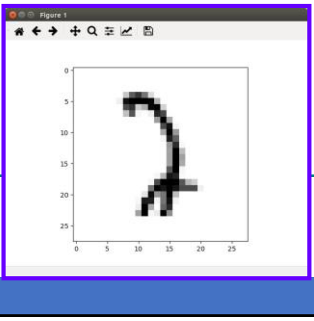
Inference

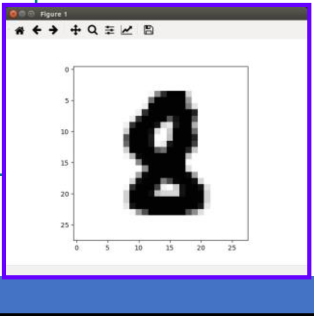
Show image

MNIST using MLP inference

```
(tensorflow)$ python mnist_inference.py
Extracting ../dataset/train-images-idx3-ubyte.gz
Extracting ../dataset/train-labels-idx1-ubyte.gz
Extracting ../dataset/t10k-images-idx3-ubyte.gz
Extracting ../dataset/t10k-labels-idx1-ubyte.gz
Label:  [1]
Prediction:  [1]
Label:  [2]
Prediction:  [2]
Label:  [8]
Prediction:  [8]
```







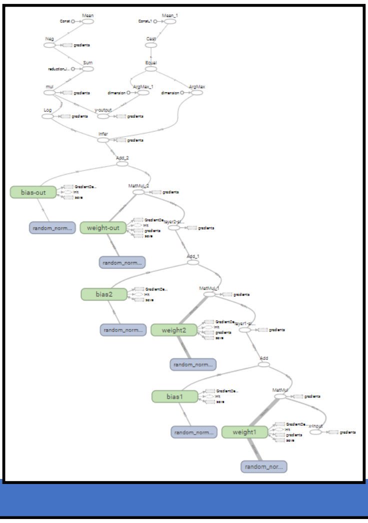
MNIST using MLP

- This example shows how to use MLP to test MNIST
  - ▶ Step 1: go to your project directory
    - ➲ [user@host] cd $(PROJECT)/codes/tensorflow-project/mnist-project/mlp
  - ▶ Step 2: see the codes
  - ▶ Step 3: run Python under virtual environment
    - ➲ (do not forget to run ‘$ source ~/tensorflow/bin/activate’)
    - ➲ [user@host] python mnist_mlp.py
    - ➲ [user@host] python mnist_inference.py

```
[user@host] cd $(PROJECT)/codes/tensorflow-project/mnist-project/mlp
[user@host] python mnist_mlp.py
[user@host] python mnist_inference.py
```

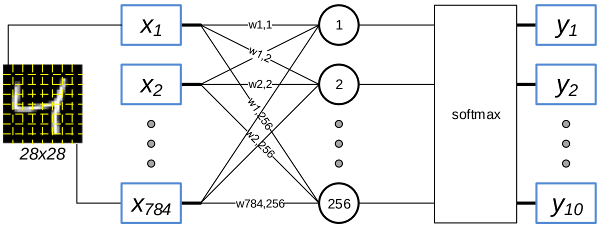
MNIST using MLP

- Make more structured style using “with tf.name.scope(“LAYER1”):”



MNIST using single hidden layer network

- Logistic regression for MNIST
  - 784 (28x28) inputs of black and white ➔ converted to floating number 0.0 ~ 1.0
  - 10 outputs representing digit 0 to 9
  - one hidden layer
  - 256 features for each hidden layer

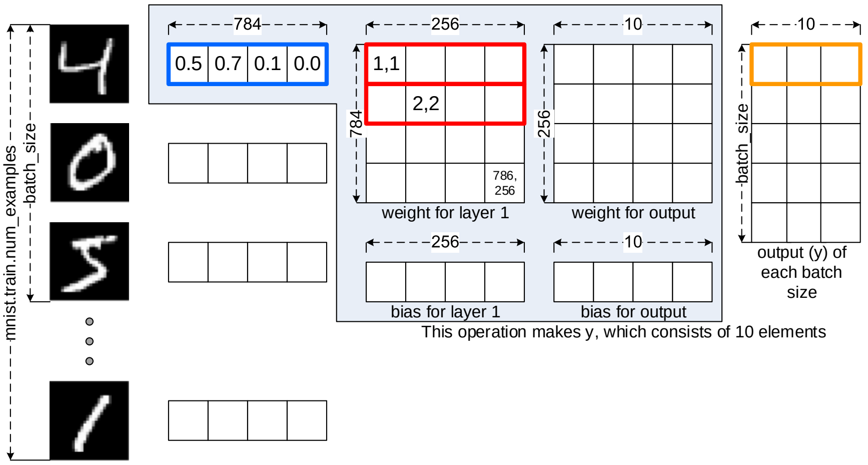


$(PROJECT)/codes/tensorflow-projects/mnist-projects/single

MNIST using single hidden layer network

- Total testing patterns
  - 60K images
- Batch number
  - 100 ➔ group 600 batches
- Epoch number
  - The number of whole trainings (forward calculation and back propagation)

MNIST using single hidden layer network



This operation makes y, which consists of 10 elements

MNIST using single hidden layer network

```
…. common parts are not shown ….
# tf Graph Input
x = tf.placeholder("float", [None, 784], name="x-input") # mnist data image of
shape 28*28
y = tf.placeholder("float", [None, 10], name="y-output") # 0-9 digits recognition =>
10 classes

#----------------------------------------------------------------------
# Model weights
W = tf.Variable(tf.zeros([784, 10]), name="weight")
b = tf.Variable(tf.zeros([10]), name="bias")

#----------------------------------------------------------------------
# inference -> hypothesis
infer = tf.nn.softmax(tf.matmul(x, W) + b, name="infer")

# Minimize error using cross entropy
cost = tf.reduce_mean(-tf.reduce_sum(y * tf.log(infer), reduction_indices=1))
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
# Evaluate -- 0~1 accuracy
correct  = tf.equal(tf.argmax(infer, 1), tf.argmax(y,1))
accuracy = tf.reduce_mean(tf.cast(correct, "float"))

# Initializing the variables
init = tf.global_variables_initializer()
```

MNIST using single hidden layer network

```
#----------------------------------------------------------------------
with tf.Session() as sess:
    sess.run(init)
    # Training cycle
    for epoch in range(training_epochs):
        avg_cost = 0.; avg_accu = 0.
        total_batch = int( mnist.train.num_examples / batch_size )
        # Loop over all batches
        for i in range(total_batch):
            batch_xs, batch_ys = mnist.train.next_batch(batch_size)
            # Fit training using batch data
            sess.run(optimizer, feed_dict={x: batch_xs, y: batch_ys})
            # Compute average loss
            avg_cost += sess.run(cost, feed_dict={x: batch_xs, y: batch_ys})
            avg_accu += sess.run(accuracy, feed_dict={x: batch_xs, y: batch_ys})
        avg_cost /= total_batch;  avg_accu /= total_batch
        # Distplay logs per epoch step
        if epoch % display_step == 0:
          print "Epoch:", "%04d" % (epoch + 1),\
                "cost=", "{:.5f}".format(avg_cost),\
                "accuracy=", "{:.5f}".format(avg_accu)
        if avg_accu>0.95:
          break
    print "Optimization finished“
```

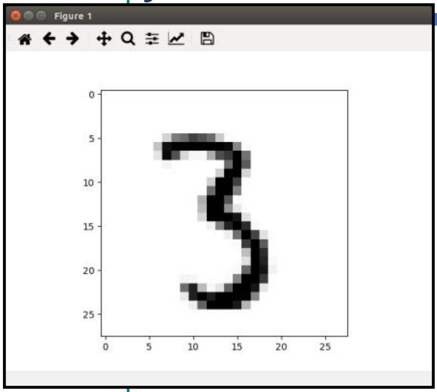
…. common parts are not shown ….

epoch times over all data set

for each batch

MNIST using single hidden layer network

```
(tensorflow)$ python mnist_single.py
Extracting ../dataset/train-images-idx3-ubyte.gz
Extracting ../dataset/train-labels-idx1-ubyte.gz
Extracting ../dataset/t10k-images-idx3-ubyte.gz
Extracting ../dataset/t10k-labels-idx1-ubyte.gz
Epoch: 0001 cost= 10.12062 accuracy= 0.12273
Epoch: 0011 cost= 1.31484 accuracy= 0.74938
Epoch: 0021 cost= 0.97869 accuracy= 0.80429
Epoch: 0031 cost= 0.83487 accuracy= 0.82842
… …
Epoch: 0071 cost= 0.61011 accuracy= 0.86305
Epoch: 0081 cost= 0.58191 accuracy= 0.86751
Epoch: 0091 cost= 0.55826 accuracy= 0.87131
... ...
Epoch: 0191 cost= 0.43436 accuracy= 0.89205
Epoch: 0201 cost= 0.42729 accuracy= 0.89313
Epoch: 0211 cost= 0.42070 accuracy= 0.89489
... ...
Epoch: 0471 cost= 0.33196 accuracy= 0.91184
Epoch: 0481 cost= 0.33010 accuracy= 0.91236
Epoch: 0491 cost= 0.32824 accuracy= 0.91265
Optimization finished
Accuracy: 0.9106
Label:  [3]
Prediction:  [3]
```



Training accuracy

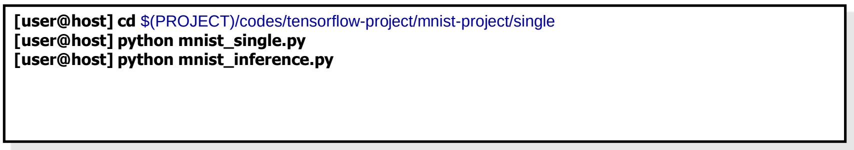
Validation accuracy

Predict correct label

MNIST using single hidden layer network

- This example shows how to use single hidden layer to test MNIST
  - Step 1: go to your project directory
    - [user@host] cd $(PROJECT)/codes/tensorflow-project/mnist-project/single
  - Step 2: see the codes
  - Step 3: run Python under virtual environment
    - (do not forget to run ‘$ source ~/tensorflow/bin/activate’)
    - [user@host] python mnist_single.py
    - [user@host] python mnist_inference.py

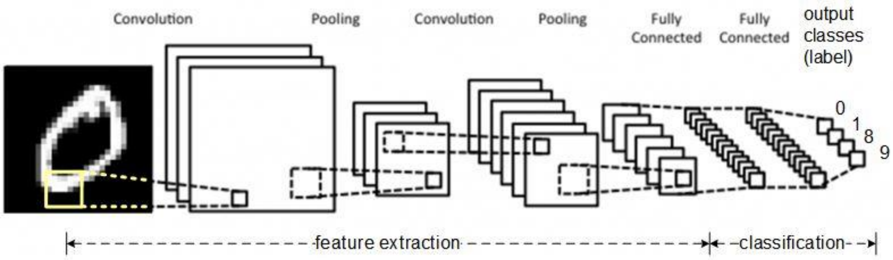
Prepare your own code in
$(PROJECT)/codes/tensorflow-projects/mnist-projects/single



```
[user@host] cd $(PROJECT)/codes/tensorflow-project/mnist-project/single
[user@host] python mnist_single.py
[user@host] python mnist_inference.py
```

MNIST using CNN

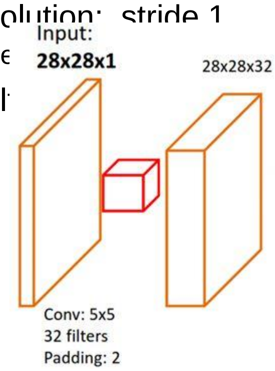
- Logistic regression for MNIST
  - 784 (28x28) inputs of black and white ➔ converted to floating number 0.0 ~ 1.0
  - 10 outputs representing digit 0 to 9
  - one hidden layer
  - 256 features for each hidden layer



$(PROJECT)/codes/tensorflow-projects/mnist-projects/cnn

MNIST using CNN: 1st convolutional layer

- Input: 28x28 pixels
- Convolution filter: 32 kernels with 5x5
- Convolution: stride 1
  - It ge ber of elements
- Resul f 28x28

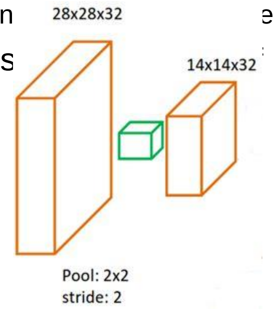


```
… …
with tf.name_scope("WEIGHT-BIAS"):
    weights = {
          # 5x5 conv, 1 input, 32 outputs
          'wc1': tf.Variable(tf.random_normal([5, 5, 1, 32]),\
                  name="wc1"),
… …

with tf.name_scope("CONV-POOL-1"):
       # Convolution Layer
       conv1 = conv2d(x, weights['wc1'], biases['bc1'])

… …

def conv2d(x, W, b, strides=1):
   with tf.name_scope("CONV"):
       # Conv2D wrapper, with bias and relu activation
       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)
```

31

MNIST using CNN: 1st pooling

- Input: 32 features with 28x28
- Max pooling filter: 5x5
- Convolution: stride 2
  - It gen elements
- Results 14x14

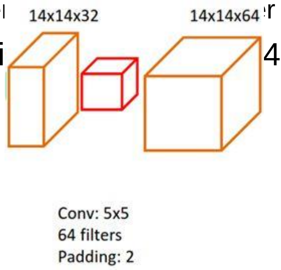


```
with tf.name_scope("CONV-POOL-1"):
      # Convolution Layer
      conv1 = conv2d(x, weights['wc1'], biases['bc1'])
      # Max Pooling (down-sampling)
      conv1 = maxpool2d(conv1, k=2)
```

32

MNIST using CNN: 2nd convolutional layer

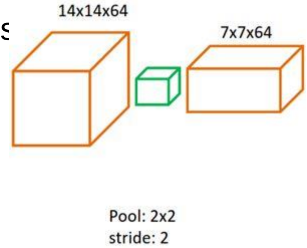
- Input: 32 features with 14x14 pixels
- Convolution filter: 64 kernels with 5x5
- Convolution: stride 1
  - It gene r of elements
- Results i 4x14



```
… …
with tf.name_scope("WEIGHT-BIAS"):
    weights = {
          # 5x5 conv, 1 input, 32 outputs
          'wc1': tf.Variable(tf.random_normal([5, 5, 1, 32]),
name="wc1"),
          # 5x5 conv, 32 inputs, 64 outputs
          'wc2': tf.Variable(tf.random_normal([5, 5, 32, 64]),
name="wc2"),
… …
 with tf.name_scope("CONV-POOL-2"):
       # Convolution Layer
       conv2 = conv2d(conv1, weights['wc2'], biases['bc2'])
       # Max Pooling (down-sampling)
       conv2 = maxpool2d(conv2, k=2)
… …
def conv2d(x, W, b, strides=1):
   with tf.name_scope("CONV"):
       # Conv2D wrapper, with bias and relu activation
       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)
```

MNIST using CNN: 2nd pooling

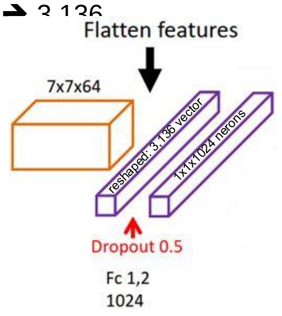
- Input: 64 features with 14x14
- Max pooling filter: 2x2
- Convolution: stride 2
  - It generates ½ number of elements
- Results 7



```
with tf.name_scope("CONV-POOL-2"):
      # Convolution Layer
      conv2 = conv2d(conv1, weights['wc2'], biases['bc2'])
      # Max Pooling (down-sampling)
      conv2 = maxpool2d(conv2, k=2)
```

MNIST using CNN: fully connected layer

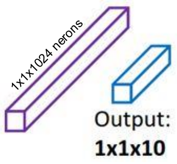
- Input: 64 features with 7x7
- Reshaping: 3-D array to 1-D vector
  - 64x7x7 ➔ 3,136
- Neurons:



```
 with tf.name_scope("FC"):
        # Fully connected layer
        # Reshape conv2 output to fit fully connected layer
input
        fc1 = tf.reshape(conv2, [-1,
weights['wd1'].get_shape().as_list()[0]])
        fc1 = tf.add(tf.matmul(fc1, weights['wd1']),
biases['bd1'])
        fc1 = tf.nn.relu(fc1)
        # Apply Dropout
        fc1 = tf.nn.dropout(fc1, dropout)
```

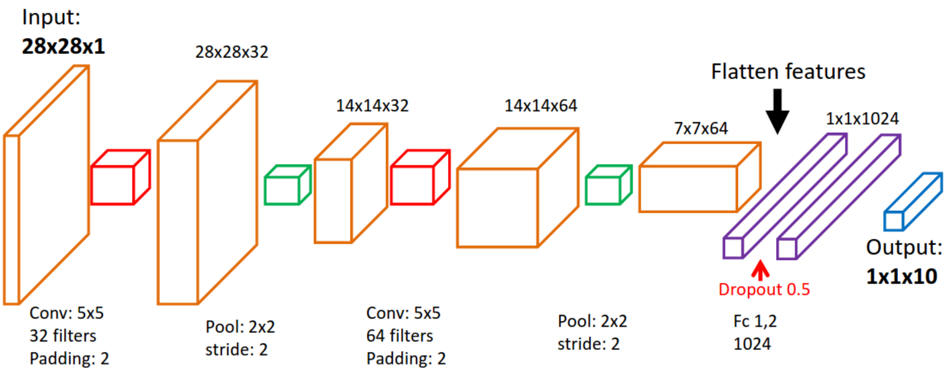
MNIST using CNN: read-out layer

- Input: 1024 neurons
- Output: 10 classes



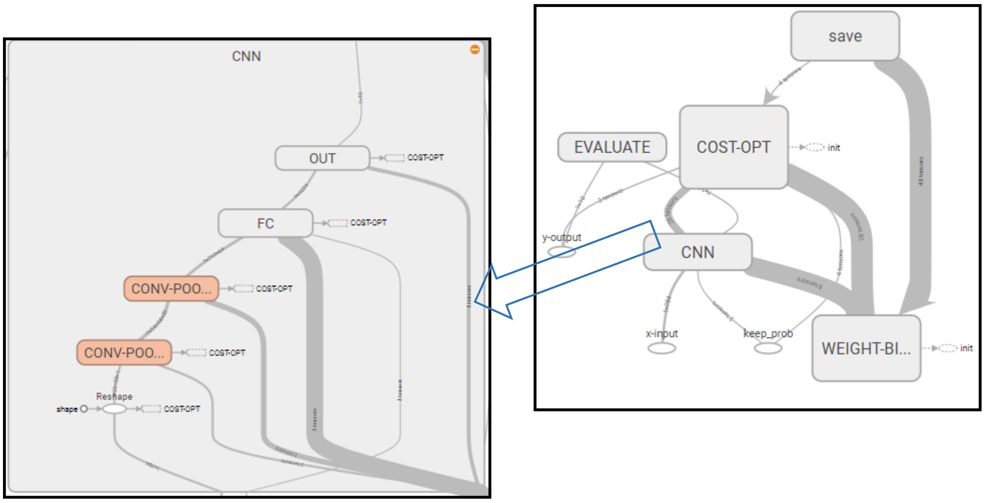
```
 with tf.name_scope("OUT"):
        # Output, class prediction
        out = tf.add(tf.matmul(fc1, weights['out']), biases['out'],
name="add-output")
```

MNIST using CNN



http://www.cnblogs.com/BigBallon/p/6701846.html

MNIST using CNN



MNIST using CNN

- This example shows how to use CNN to test MNIST
  - Step 1: go to your project directory
    - [user@host] cd $(PROJECT)/codes/tensorflow-project/mnist-project/cnn
  - Step 2: see the codes
  - Step 3: run Python under virtual environment
    - (do not forget to run '$ source ~/tensorflow/bin/activate')
    - [user@host] python mnist_cnn.py
    - [user@host] python mnist_inference.py

```
[user@host] cd $(PROJECT)/codes/tensorflow-project/mnist-project/cnn
[user@host] python mnist_cnn.py
[user@host] python mnist_inference.py
```

㈜퓨쳐디자인시스템
34051 대전광역시 유성구 문지로 193, KAIST 문지캠퍼스, F723호
(042) 864-0211~0212 / contact@future-ds.com / www.future-ds.com

Future Design Systems, Inc.
Faculty Wing F723, KAIST Munji Campus, 193 Munji-ro, Yuseong-gu, Daejeon 34051, Korea
+82-042-864-0211~0212 / contact@future-ds.com / www.future-ds.com

FUTURE
Design Systems