Tasks

- 了解back-propagation (BP) 原理
  - 公式推導 →(重點：chain-rule)
  - Error →softmax → sigmod/relu → … →input
- 更改MNIST for ML Beginners → MLP/DNN
  - 最佳化→ 正確率大於99%
  - Training data + Noises (optinal)
- GPU (optional)
  - NVIDIA CUDA
  - Tensorflow with GPU support
  - 速度改善多少？

Chain Rule

Chain Rule.

Case 1.

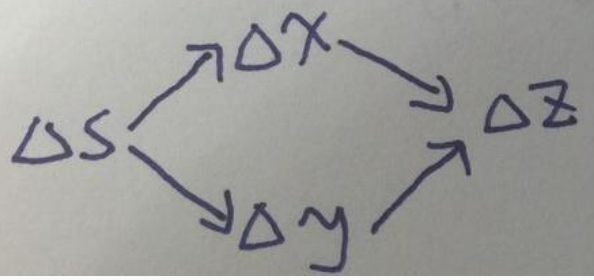
$y = g(x)$ $z = h(x)$.

$\Delta x \to \Delta y \to \Delta z$

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}.$$

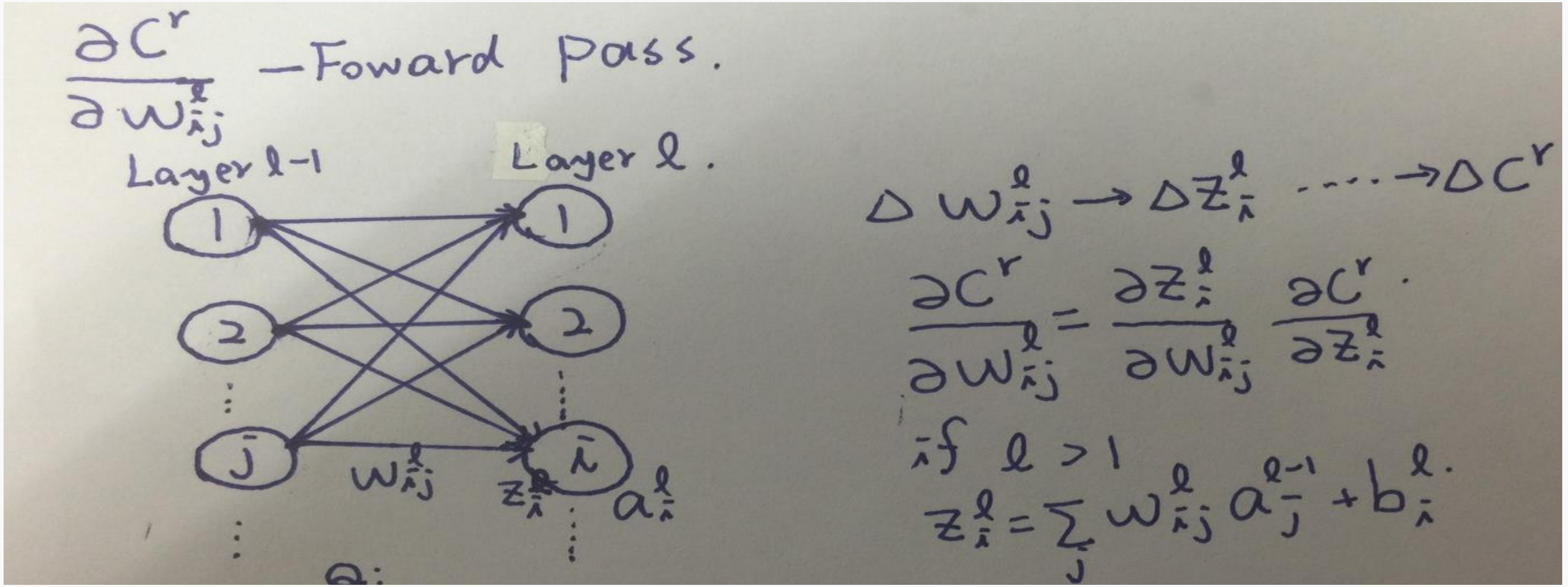
Case 2.

$x = g(s)$ $y = h(s)$ $z = k(x, y)$

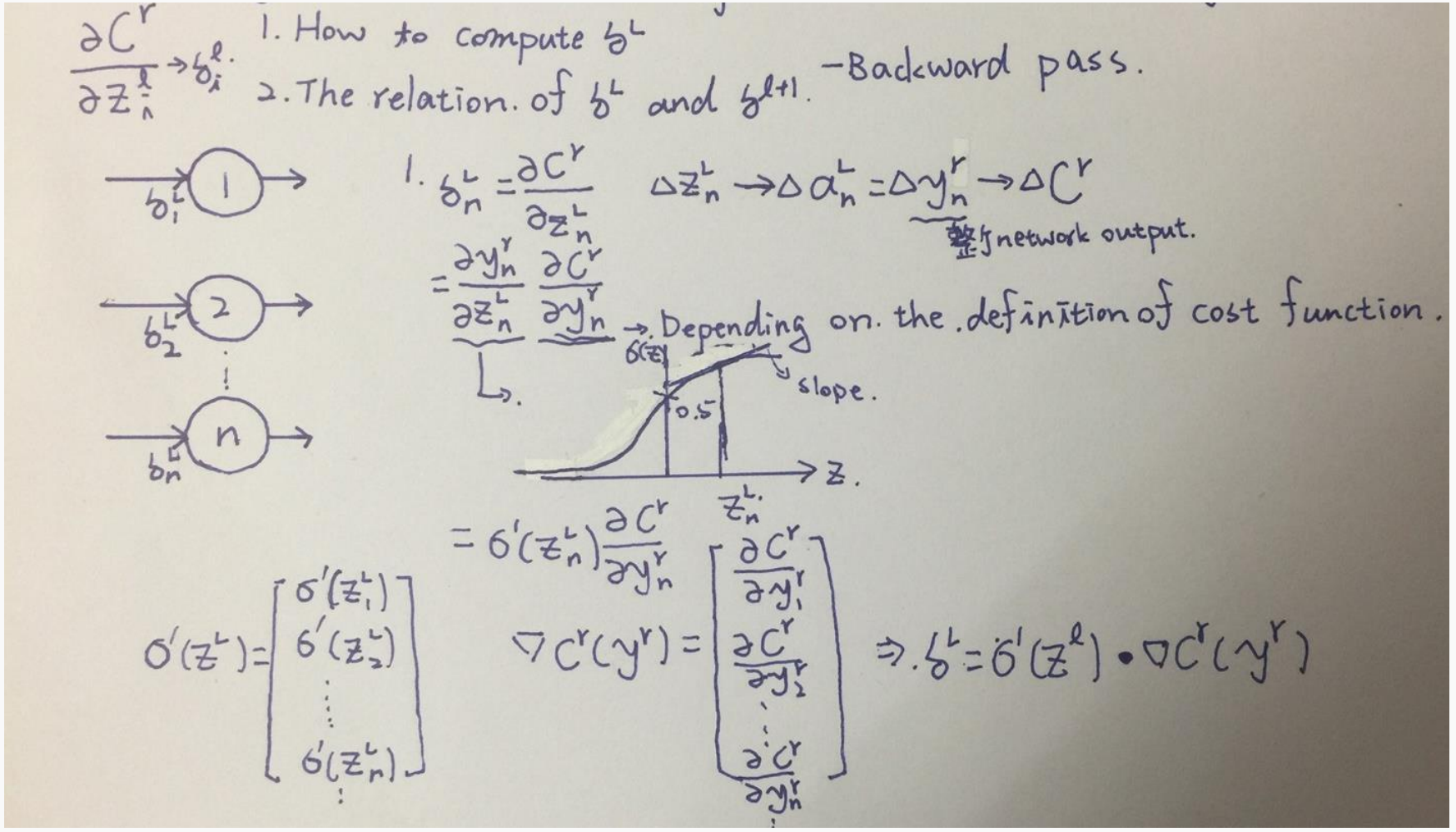


$$\frac{\partial z}{\partial s} = \frac{\partial z}{\partial x}\frac{\partial x}{\partial s} + \frac{\partial z}{\partial y}\frac{\partial y}{\partial s}.$$

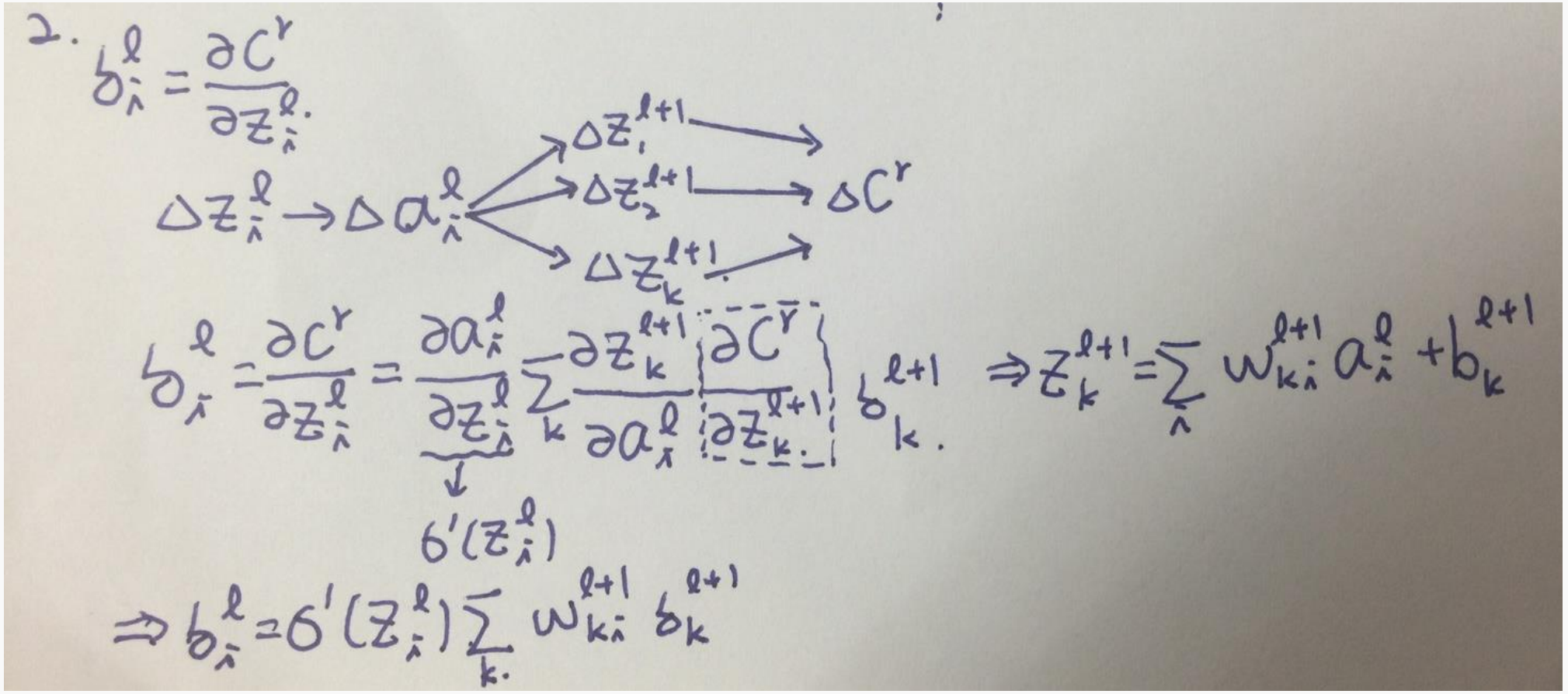
Forward Pass



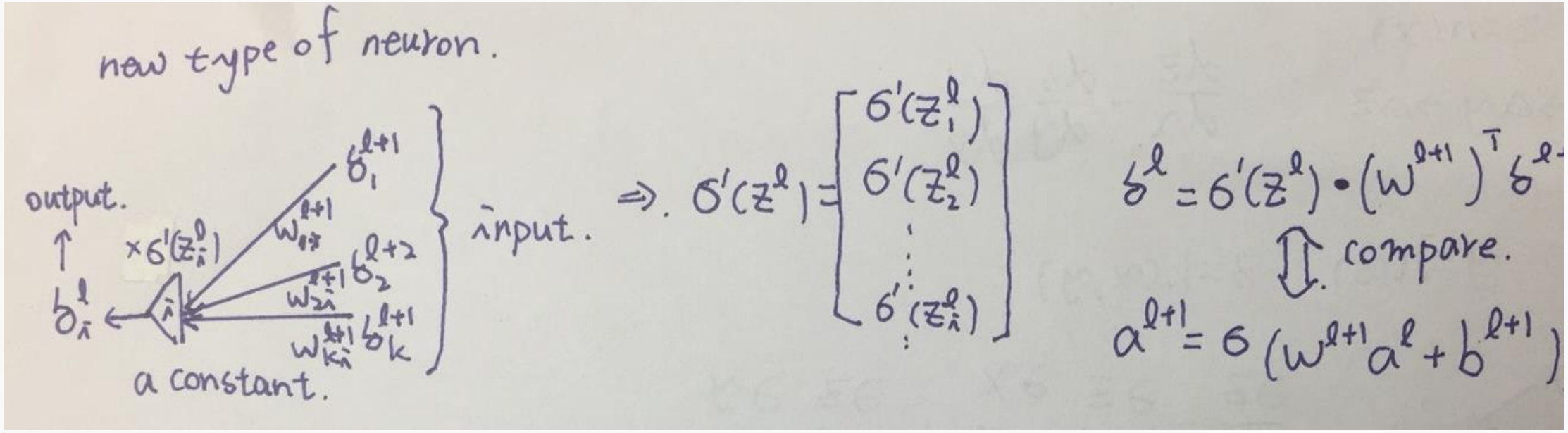
Backward Pass(1)



Backward Pass(2)



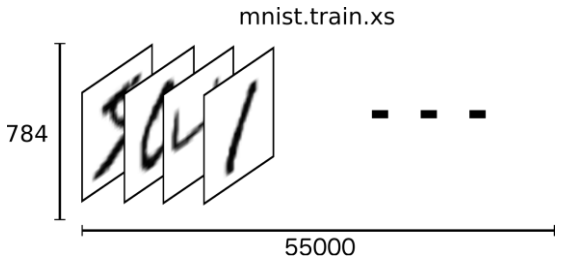
Backward Pass(3)

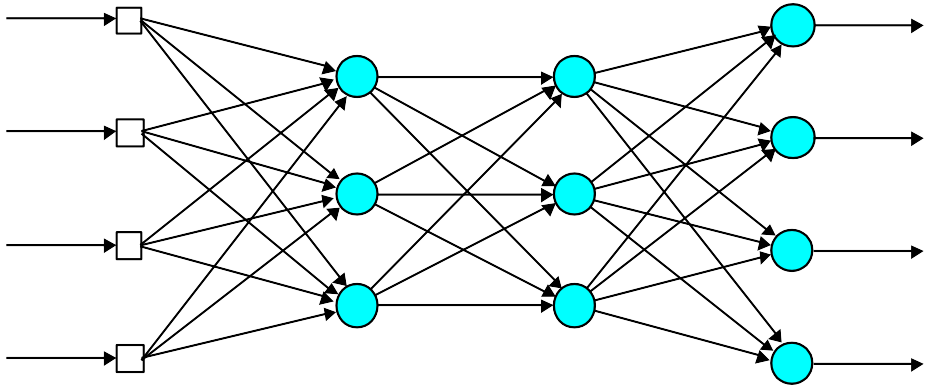


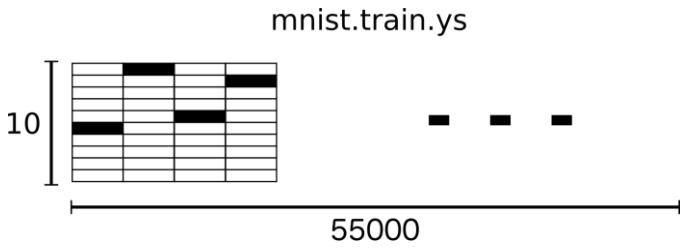
DNN 訓練流程

- 初始化權重
- 利用目前的權重計算輸出結果
- 計算輸出結果和目標結果的誤差
- 調整權重
- **LOOP STEP2-5**直到收斂

Tensorflow Tutorial







Hidden layer with ReLU&ReLU6

Without Hidden Layer

Performance:0.917

jupyter Untitled Last Checkpoint: Last Monday at 10:06 PM (autosaved)

File Edit View Insert Cell Kernel Widgets Help Pythor

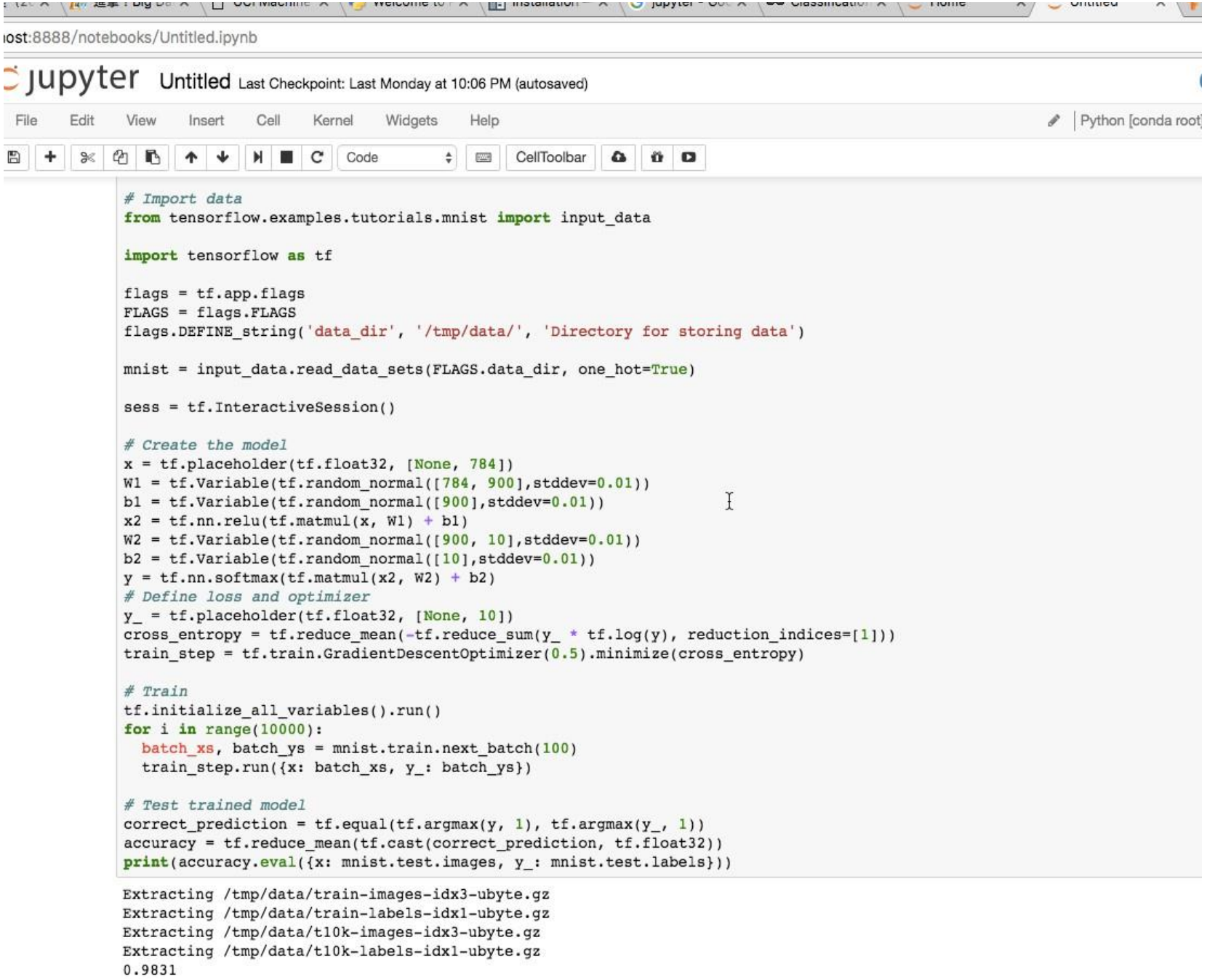
Code CellToolbar

```
# Import data
from tensorflow.examples.tutorials.mnist import input_data

import tensorflow as tf

flags = tf.app.flags
FLAGS = flags.FLAGS
flags.DEFINE_string('data_dir', '/tmp/data/', 'Directory for storing data')

mnist = input_data.read_data_sets(FLAGS.data_dir, one_hot=True)

sess = tf.InteractiveSession()

# Create the model
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.nn.softmax(tf.matmul(x, W) + b)

# Define loss and optimizer
y_ = tf.placeholder(tf.float32, [None, 10])
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

# Train
tf.initialize_all_variables().run()
for i in range(1000):
  batch_xs, batch_ys = mnist.train.next_batch(100)
  train_step.run({x: batch_xs, y_: batch_ys})

# Test trained model
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print(accuracy.eval({x: mnist.test.images, y_: mnist.test.labels}))
```

```
Extracting /tmp/data/train-images-idx3-ubyte.gz
Extracting /tmp/data/train-labels-idx1-ubyte.gz
Extracting /tmp/data/t10k-images-idx3-ubyte.gz
Extracting /tmp/data/t10k-labels-idx1-ubyte.gz
0.917
```

One Hidden Layer

Performance:0.9831



One Hidden Layer (With ReLU6)

Performance:0.9817

```
# Create the model
x = tf.placeholder(tf.float32, [None, 784])
W1 = tf.Variable(tf.random_normal([784, 441],stddev=0.01))
b1 = tf.Variable(tf.random_normal([441],stddev=0.01))
x2 = tf.nn.relu6(tf.matmul(x, W1) + b1)
W2 = tf.Variable(tf.random_normal([441, 10],stddev=0.01))
b2 = tf.Variable(tf.random_normal([10],stddev=0.01))
y = tf.nn.softmax(tf.matmul(x2, W2) + b2)

# Define loss and optimizer
y_ = tf.placeholder(tf.float32, [None, 10])
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

# Train
tf.initialize_all_variables().run()
for i in range(10000):
  batch_xs, batch_ys = mnist.train.next_batch(100)
  train_step.run({x: batch_xs, y_: batch_ys})

# Test trained model
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print(accuracy.eval({x: mnist.test.images, y_: mnist.test.labels}))
```

```
Extracting /tmp/data/train-images-idx3-ubyte.gz
Extracting /tmp/data/train-labels-idx1-ubyte.gz
Extracting /tmp/data/t10k-images-idx3-ubyte.gz
Extracting /tmp/data/t10k-labels-idx1-ubyte.gz
0.9817
```

Two Hidden Layer (With ReLU6)

Performance:**0.985**

```python
import tensorflow as tf

flags = tf.app.flags
FLAGS = flags.FLAGS
flags.DEFINE_string('data_dir', '/tmp/data/', 'Directory for storing data')

mnist = input_data.read_data_sets(FLAGS.data_dir, one_hot=True)

sess = tf.InteractiveSession()

# Create the model
x = tf.placeholder(tf.float32, [None, 784])
W1 = tf.Variable(tf.random_normal([784, 625],stddev=0.01))
b1 = tf.Variable(tf.random_normal([625],stddev=0.01))
x2 = tf.nn.relu6(tf.matmul(x, W1) + b1)
W2 = tf.Variable(tf.random_normal([625, 441],stddev=0.01))
b2 = tf.Variable(tf.random_normal([441],stddev=0.01))
x3 = tf.nn.relu6(tf.matmul(x2, W2) + b2)
W3 = tf.Variable(tf.random_normal([441, 10],stddev=0.01))
b3 = tf.Variable(tf.random_normal([10],stddev=0.01))
y = tf.nn.softmax(tf.matmul(x3, W3) + b3)
# Define loss and optimizer
y_ = tf.placeholder(tf.float32, [None, 10])
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

# Train
tf.initialize_all_variables().run()
for i in range(10000):
  batch_xs, batch_ys = mnist.train.next_batch(100)
  train_step.run({x: batch_xs, y_: batch_ys})

# Test trained model
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print(accuracy.eval({x: mnist.test.images, y_: mnist.test.labels}))
```

```
Extracting /tmp/data/train-images-idx3-ubyte.gz
Extracting /tmp/data/train-labels-idx1-ubyte.gz
Extracting /tmp/data/t10k-images-idx3-ubyte.gz
Extracting /tmp/data/t10k-labels-idx1-ubyte.gz
0.985
```

Three Hidden Layer (With ReLU6)

Performance:0.9847

```
# Create the model
x = tf.placeholder(tf.float32, [None, 784])
W1 = tf.Variable(tf.random_normal([784, 612],stddev=0.01))
b1 = tf.Variable(tf.random_normal([612],stddev=0.01))
x2 = tf.nn.relu6(tf.matmul(x, W1) + b1)
W2 = tf.Variable(tf.random_normal([612, 466],stddev=0.01))
b2 = tf.Variable(tf.random_normal([466],stddev=0.01))
x3 = tf.nn.relu6(tf.matmul(x2, W2) + b2)
W3 = tf.Variable(tf.random_normal([466, 211],stddev=0.01))
b3 = tf.Variable(tf.random_normal([211],stddev=0.01))
x4 = tf.nn.relu6(tf.matmul(x3, W3) + b3)
W4 = tf.Variable(tf.random_normal([211, 10],stddev=0.01))
b4 = tf.Variable(tf.random_normal([10],stddev=0.01))
y = tf.nn.softmax(tf.matmul(x4, W4) + b4)

# Define loss and optimizer
y_ = tf.placeholder(tf.float32, [None, 10])
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

# Train
tf.initialize_all_variables().run()
for i in range(10000):
  batch_xs, batch_ys = mnist.train.next_batch(100)
  train_step.run({x: batch_xs, y_: batch_ys})

# Test trained model
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print(accuracy.eval({x: mnist.test.images, y_: mnist.test.labels}))
```

```
Extracting /tmp/data/train-images-idx3-ubyte.gz
Extracting /tmp/data/train-labels-idx1-ubyte.gz
Extracting /tmp/data/t10k-images-idx3-ubyte.gz
Extracting /tmp/data/t10k-labels-idx1-ubyte.gz
0.9847
```

Four Hidden Layer (With ReLU6) 10000 epoch

Performance:0.982

```
# Create the model
x = tf.placeholder(tf.float32, [None, 784])
W1 = tf.Variable(tf.random_normal([784, 629],stddev=0.01))
b1 = tf.Variable(tf.random_normal([629],stddev=0.01))
x2 = tf.nn.relu6(tf.matmul(x, W1) + b1)
W2 = tf.Variable(tf.random_normal([629, 474],stddev=0.01))
b2 = tf.Variable(tf.random_normal([474],stddev=0.01))
x3 = tf.nn.relu6(tf.matmul(x2, W2) + b2)
W3 = tf.Variable(tf.random_normal([474, 319],stddev=0.01))
b3 = tf.Variable(tf.random_normal([319],stddev=0.01))
x4 = tf.nn.relu6(tf.matmul(x3, W3) + b3)
W4 = tf.Variable(tf.random_normal([319, 164],stddev=0.01))
b4 = tf.Variable(tf.random_normal([164],stddev=0.01))
x5 = tf.nn.relu6(tf.matmul(x4, W4) + b4)
W5 = tf.Variable(tf.random_normal([164, 10],stddev=0.01))
b5 = tf.Variable(tf.random_normal([10],stddev=0.01))
y = tf.nn.softmax(tf.matmul(x5, W5) + b5)

# Define loss and optimizer
y_ = tf.placeholder(tf.float32, [None, 10])
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

# Train
tf.initialize_all_variables().run()
for i in range(10000):
  batch_xs, batch_ys = mnist.train.next_batch(100)
  train_step.run({x: batch_xs, y_: batch_ys})

# Test trained model
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print(accuracy.eval({x: mnist.test.images, y_: mnist.test.labels}))
```

```
Extracting /tmp/data/train-images-idx3-ubyte.gz
Extracting /tmp/data/train-labels-idx1-ubyte.gz
Extracting /tmp/data/t10k-images-idx3-ubyte.gz
Extracting /tmp/data/t10k-labels-idx1-ubyte.gz
0.9824
```

Four Hidden Layer (With ReLU6) 5000 epoch

Performance:0.9772

```
# Create the model
x = tf.placeholder(tf.float32, [None, 784])
W1 = tf.Variable(tf.random_normal([784, 629],stddev=0.01))
b1 = tf.Variable(tf.random_normal([629],stddev=0.01))
x2 = tf.nn.relu6(tf.matmul(x, W1) + b1)
W2 = tf.Variable(tf.random_normal([629, 474],stddev=0.01))
b2 = tf.Variable(tf.random_normal([474],stddev=0.01))
x3 = tf.nn.relu6(tf.matmul(x2, W2) + b2)
W3 = tf.Variable(tf.random_normal([474, 319],stddev=0.01))
b3 = tf.Variable(tf.random_normal([319],stddev=0.01))
x4 = tf.nn.relu6(tf.matmul(x3, W3) + b3)
W4 = tf.Variable(tf.random_normal([319, 164],stddev=0.01))
b4 = tf.Variable(tf.random_normal([164],stddev=0.01))
x5 = tf.nn.relu6(tf.matmul(x4, W4) + b4)
W5 = tf.Variable(tf.random_normal([164, 10],stddev=0.01))
b5 = tf.Variable(tf.random_normal([10],stddev=0.01))
y = tf.nn.softmax(tf.matmul(x5, W5) + b5)

# Define loss and optimizer
y_ = tf.placeholder(tf.float32, [None, 10])
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

# Train
tf.initialize_all_variables().run()
for i in range(5000):
  batch_xs, batch_ys = mnist.train.next_batch(100)
  train_step.run({x: batch_xs, y_: batch_ys})

# Test trained model
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print(accuracy.eval({x: mnist.test.images, y_: mnist.test.labels}))
```

```
Extracting /tmp/data/train-images-idx3-ubyte.gz
Extracting /tmp/data/train-labels-idx1-ubyte.gz
Extracting /tmp/data/t10k-images-idx3-ubyte.gz
Extracting /tmp/data/t10k-labels-idx1-ubyte.gz
0.9772
```

Hidden layer with Sigmoid

Two Hidden Layer

Performance:0.9653

```python
# Create the model
x = tf.placeholder(tf.float32, [None, 784])
W1 = tf.Variable(tf.random_normal([784, 516],stddev=0.01))
b1 = tf.Variable(tf.random_normal([516],stddev=0.01))
x2 = tf.nn.sigmoid(tf.matmul(x, W1) + b1)
W2 = tf.Variable(tf.random_normal([516, 258],stddev=0.01))
b2 = tf.Variable(tf.random_normal([258],stddev=0.01))
x3 = tf.nn.sigmoid(tf.matmul(x2, W2) + b2)
W3 = tf.Variable(tf.random_normal([258, 10],stddev=0.01))
b3 = tf.Variable(tf.random_normal([10],stddev=0.01))
y = tf.nn.softmax(tf.matmul(x3, W3) + b3)

# Define loss and optimizer
y_ = tf.placeholder(tf.float32, [None, 10])
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

# Train
tf.initialize_all_variables().run()
for i in range(10000):
  batch_xs, batch_ys = mnist.train.next_batch(100)
  train_step.run({x: batch_xs, y_: batch_ys})

# Test trained model
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print(accuracy.eval({x: mnist.test.images, y_: mnist.test.labels}))
```

```
Extracting /tmp/data/train-images-idx3-ubyte.gz
Extracting /tmp/data/train-labels-idx1-ubyte.gz
Extracting /tmp/data/t10k-images-idx3-ubyte.gz
Extracting /tmp/data/t10k-labels-idx1-ubyte.gz
0.9653
```

Three Hidden Layer

Performance:0.9432

```
# Create the model
x = tf.placeholder(tf.float32, [None, 784])
W1 = tf.Variable(tf.random_normal([784, 580],stddev=0.01))
b1 = tf.Variable(tf.random_normal([580],stddev=0.01))
x2 = tf.nn.sigmoid(tf.matmul(x, W1) + b1)
W2 = tf.Variable(tf.random_normal([580, 386],stddev=0.01))
b2 = tf.Variable(tf.random_normal([386],stddev=0.01))
x3 = tf.nn.sigmoid(tf.matmul(x2, W2) + b2)
W3 = tf.Variable(tf.random_normal([386, 192],stddev=0.01))
b3 = tf.Variable(tf.random_normal([192],stddev=0.01))
x4 = tf.nn.sigmoid(tf.matmul(x3, W3) + b3)
W4 = tf.Variable(tf.random_normal([192, 10],stddev=0.01))
b4 = tf.Variable(tf.random_normal([10],stddev=0.01))
y = tf.nn.softmax(tf.matmul(x4, W4) + b4)

# Define loss and optimizer
y_ = tf.placeholder(tf.float32, [None, 10])
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

# Train
tf.initialize_all_variables().run()
for i in range(10000):
  batch_xs, batch_ys = mnist.train.next_batch(100)
  train_step.run({x: batch_xs, y_: batch_ys})

# Test trained model
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print(accuracy.eval({x: mnist.test.images, y_: mnist.test.labels}))
```

```
Extracting /tmp/data/train-images-idx3-ubyte.gz
Extracting /tmp/data/train-labels-idx1-ubyte.gz
Extracting /tmp/data/t10k-images-idx3-ubyte.gz
Extracting /tmp/data/t10k-labels-idx1-ubyte.gz
0.9432
```

Hidden layer with Dropout

Two Hidden Layer
(With ReLU6)

Performance:0.9472

```
# Create the model
x = tf.placeholder(tf.float32, [None, 784])
W1 = tf.Variable(tf.random_normal([784, 516],stddev=0.01))
b1 = tf.Variable(tf.random_normal([516],stddev=0.01))
x2 = tf.nn.dropout(tf.nn.relu6(tf.matmul(x, W1) + b1), 0.33)
W2 = tf.Variable(tf.random_normal([516, 258],stddev=0.01))
b2 = tf.Variable(tf.random_normal([258],stddev=0.01))
x3 = tf.nn.dropout(tf.nn.relu6(tf.matmul(x2, W2) + b2), 0.33)
W3 = tf.Variable(tf.random_normal([258, 10],stddev=0.01))
b3 = tf.Variable(tf.random_normal([10],stddev=0.01))
y = tf.nn.softmax(tf.matmul(x3, W3) + b3)

# Define loss and optimizer
y_ = tf.placeholder(tf.float32, [None, 10])
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))
train_step = tf.train.GradientDescentOptimizer(0.03).minimize(cross_entropy)

# Train
tf.initialize_all_variables().run()
for i in range(10000):
  batch_xs, batch_ys = mnist.train.next_batch(100)
  train_step.run({x: batch_xs, y_: batch_ys})

# Test trained model
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print(accuracy.eval({x: mnist.test.images, y_: mnist.test.labels}))
```

```
Extracting /tmp/data/train-images-idx3-ubyte.gz
Extracting /tmp/data/train-labels-idx1-ubyte.gz
Extracting /tmp/data/t10k-images-idx3-ubyte.gz
Extracting /tmp/data/t10k-labels-idx1-ubyte.gz
0.9472
```

Result

Model	Result
Without Hidden Layer	0.917
One Hidden Layer	0.9831
One Hidden Layer(With ReLU6)	0.9817
Two Hidden Layer(With ReLU6)	0.985
Three Hidden Layer(With ReLU6)	0.9847
Four Hidden Layer(With ReLU6)10000 epoch	0.982
Four Hidden Layer(With ReLU6)5000 epoch	0.9772
Two Hidden Layer(Sigmoid)	0.9653
Three Hidden Layer(Sigmoid)	0.9432
Two Hidden Layer(Dropout with ReLU6)	0.9472

Thanks