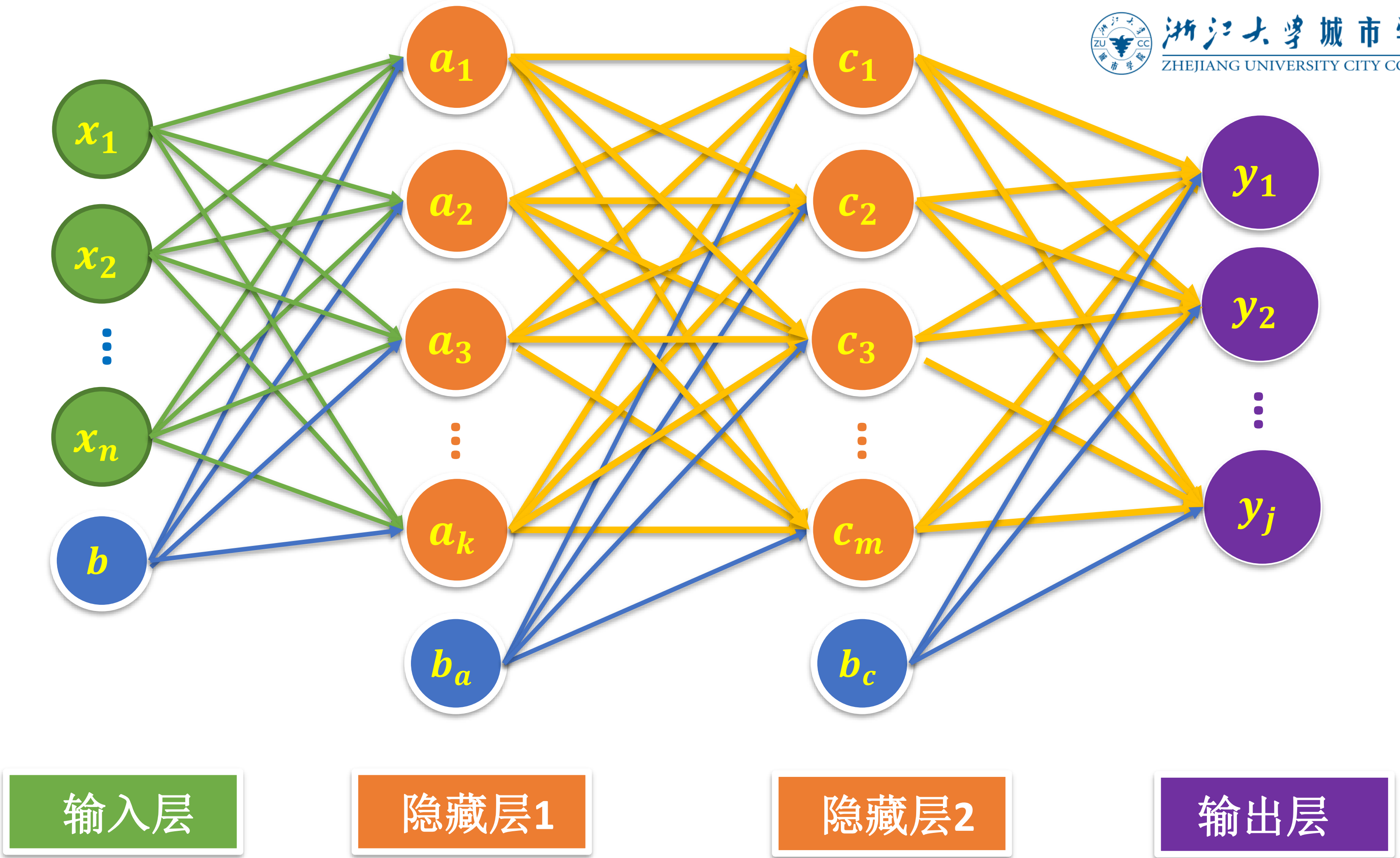
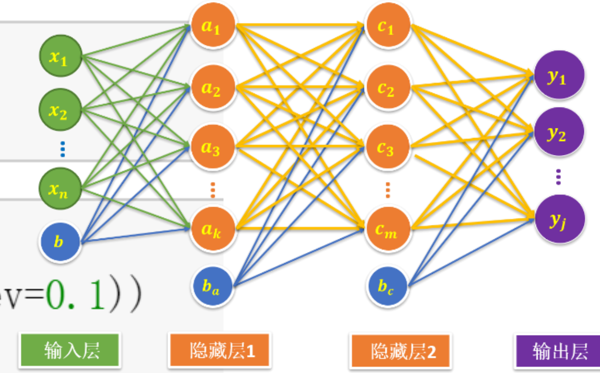
再多一点，多层网络建模实现


x_1
x_2
x_n
b
a_1
a_2
a_3
a_k
b_a
c_1
c_2
c_3
c_m
b_c
y_1
y_2
y_j
输入层
隐藏层1
隐藏层2
输出层

浙江大学城市学院 ZHEJIANG UNIVERSITY CITY COLLEGE

构建模型

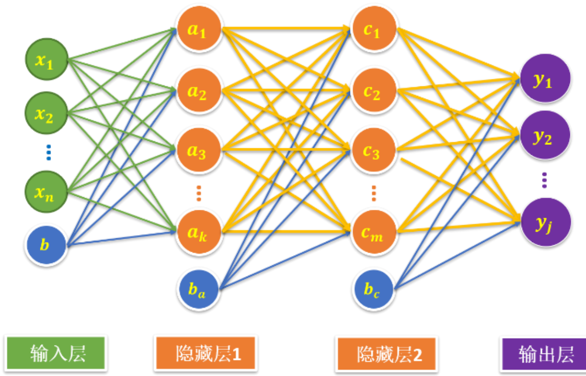
```
H1_NN = 256    # 第1隐藏层神经元为 256 个
H2_NN = 64     # 第2隐藏层神经元为 64 个
```




```
# 输入层 - 第1隐藏层参数和偏置项
W1 = tf.Variable(tf.truncated_normal([784, H1_NN], stddev=0.1))
b1 = tf.Variable(tf.zeros([H1_NN]))

# 第1隐藏层 - 第2隐藏层参数和偏置项
W2 = tf.Variable(tf.truncated_normal([H1_NN,H2_NN], stddev=0.1))
b2 = tf.Variable(tf.zeros([H2_NN]))

# 第2隐藏层 - 输出层参数和偏置项
W3 = tf.Variable(tf.truncated_normal([H2_NN,10], stddev=0.1))
b3 = tf.Variable(tf.zeros([10]))
```

构建模型

```
# 计算第1隐藏层结果
Y1 = tf.nn.relu(tf.matmul(x, W1) + b1)

# 计算第2隐藏层结果
Y2 = tf.nn.relu(tf.matmul(Y1, W2) + b2)

# 计算输出结果
forward = tf.matmul(Y2, W3) + b3
pred = tf.nn.softmax(forward)
```



训练结果

```
Train Epoch: 01 Loss= 0.155673772  Accuracy= 0.9586
Train Epoch: 02 Loss= 0.171394646  Accuracy= 0.9540
Train Epoch: 03 Loss= 0.143150017  Accuracy= 0.9606
Train Epoch: 04 Loss= 0.148644924  Accuracy= 0.9638
Train Epoch: 05 Loss= 0.126665384  Accuracy= 0.9678
Train Epoch: 06 Loss= 0.155957386  Accuracy= 0.9650
Train Epoch: 07 Loss= 0.177521363  Accuracy= 0.9594
Train Epoch: 08 Loss= 0.147528306  Accuracy= 0.9702

Train Epoch: 36 Loss= 0.371294141  Accuracy= 0.9628
Train Epoch: 37 Loss= 0.345369697  Accuracy= 0.9672
Train Epoch: 38 Loss= 0.337678194  Accuracy= 0.9728
Train Epoch: 39 Loss= 0.281204611  Accuracy= 0.9724
Train Epoch: 40 Loss= 0.313496113  Accuracy= 0.9734
Train Finished takes: 100.00
```

评估模型

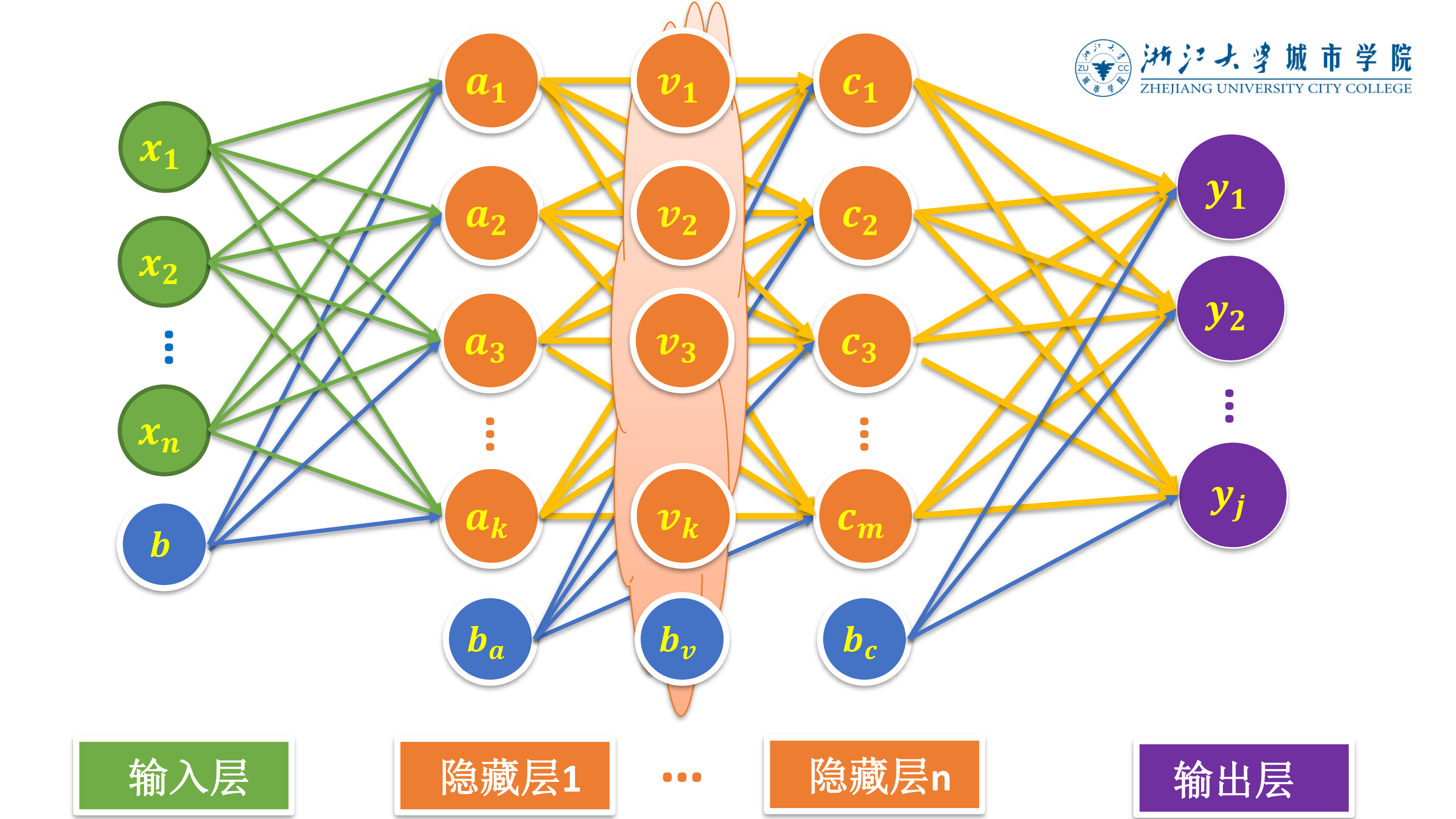
评估模型

```
accu_test =  sess.run(accuracy,
                      feed_dict={x: mnist.test.images,  y: mnist.test.labels})

print("Test Accuracy:",accu_test)
```

```
Test Accuracy: 0.969
```

多层效果不一定就比单层网络效果好！

什么，还要再多一点？



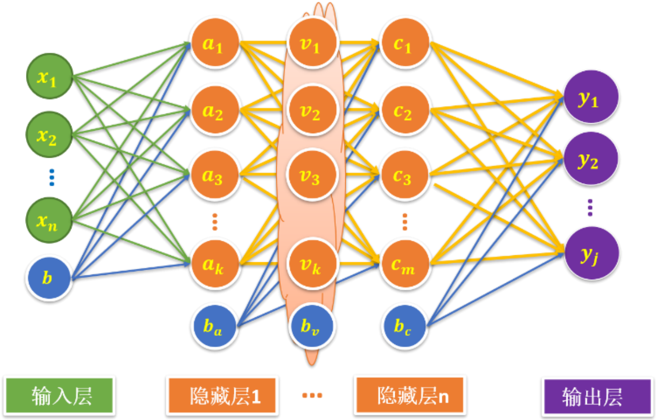
浙江大学城市学院
ZHEJIANG UNIVERSITY CITY COLLEGE
x_1
x_2
x_n
b
a_1
a_2
a_3
a_k
b_a
v_1
v_2
v_3
v_k
b_v
c_1
c_2
c_3
c_m
b_c
y_1
y_2
y_j
输入层
隐藏层1
…
隐藏层n
输出层

构建模型

构建模型

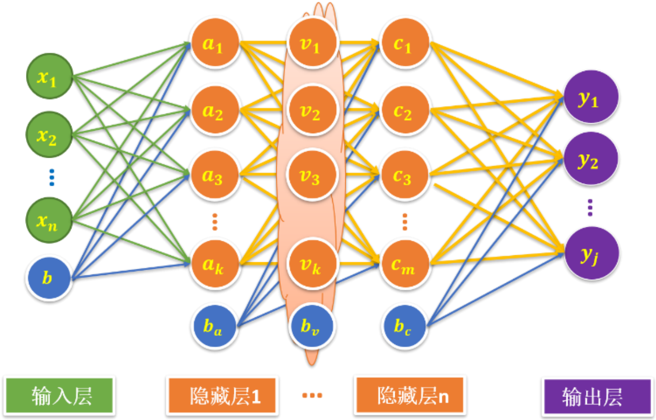
```
H1_NN = 256      # 第1隐藏层神经元为 256 个
H2_NN = 64       # 第2隐藏层神经元为 64 个
H3_NN = 32       # 第3隐藏层神经元为 32 个
```

```
# 输入层 - 第1隐藏层参数和偏置项
W1 = tf.Variable(tf.truncated_normal([784, H1_NN], stddev=0.1))
b1 = tf.Variable(tf.zeros([H1_NN]))

# 第1隐藏层 - 第2隐藏层参数和偏置项
W2 = tf.Variable(tf.truncated_normal([H1_NN,H2_NN], stddev=0.1))
b2 = tf.Variable(tf.zeros([H2_NN]))

# 第2隐藏层 - 第3隐藏层参数和偏置项
W3 = tf.Variable(tf.truncated_normal([H2_NN, H3_NN], stddev=0.1))
b3 = tf.Variable(tf.zeros([H3_NN]))

# 第3隐藏层 - 输出层参数和偏置项
W4 = tf.Variable(tf.truncated_normal([H3_NN,10], stddev=0.1))
b4 = tf.Variable(tf.zeros([10]))
```



构建模型

```
# 计算第1隐藏层结果
Y1 = tf.nn.relu(tf.matmul(x, W1) + b1)

# 计算第2隐藏层结果
Y2 = tf.nn.relu(tf.matmul(Y1, W2) + b2)

# 计算第3隐藏层结果
Y3 = tf.nn.relu(tf.matmul(Y2, W3) + b3)

# 计算输出结果
forward = tf.matmul(Y3, W4) + b4
pred = tf.nn.softmax(forward)
```



训练结果

```
Train Epoch: 01 Loss= 0.144908100  Accuracy= 0.9624
Train Epoch: 02 Loss= 0.150267839  Accuracy= 0.9638
Train Epoch: 03 Loss= 0.124543823  Accuracy= 0.9696
Train Epoch: 04 Loss= 0.151412591  Accuracy= 0.9656
Train Epoch: 05 Loss= 0.169072613  Accuracy= 0.9660
Train Epoch: 06 Loss= 0.138989180  Accuracy= 0.9726
Train Epoch: 07 Loss= 0.141304657  Accuracy= 0.9718
Train Epoch: 08 Loss= 0.153570235  Accuracy= 0.9668

Train Epoch: 35 Loss= 0.286709249  Accuracy= 0.9696
Train Epoch: 36 Loss= 0.312437803  Accuracy= 0.9758
Train Epoch: 37 Loss= 0.240455911  Accuracy= 0.9736
Train Epoch: 38 Loss= 0.268984914  Accuracy= 0.9714
Train Epoch: 39 Loss= 0.206968024  Accuracy= 0.9736
Train Epoch: 40 Loss= 0.210474610  Accuracy= 0.9740
Train Finished takes: 104.00
```

评估模型

评估模型

```
accu_test =  sess.run(accuracy,
                      feed_dict={x: mnist.test.images, y: mnist.test.labels})

print("Test Accuracy:",accu_test)
```

```
Test Accuracy: 0.9744
```

试着修改超参数，看看准确率的变换

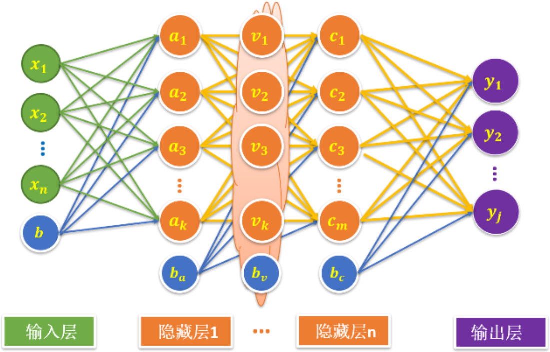
重构建模过程

构建模型

构建模型

```
H1_NN = 256       # 第1隐藏层神经元为 256 个
H2_NN = 64        # 第2隐藏层神经元为 64 个
H3_NN = 32        # 第3隐藏层神经元为 32 个
```

```
# 输入层 - 第1隐藏层参数和偏置项
W1 = tf.Variable(tf.truncated_normal([784, H1_NN], stddev=0.1))
b1 = tf.Variable(tf.zeros([H1_NN]))

# 第1隐藏层 - 第2隐藏层参数和偏置项
W2 = tf.Variable(tf.truncated_normal([H1_NN,H2_NN], stddev=0.1))
b2 = tf.Variable(tf.zeros([H2_NN]))

# 第2隐藏层 - 第3隐藏层参数和偏置项
W3 = tf.Variable(tf.truncated_normal([H2_NN, H3_NN], stddev=0.1))
b3 = tf.Variable(tf.zeros([H3_NN]))

# 第3隐藏层 - 输出层参数和偏置项
W4 = tf.Variable(tf.truncated_normal([H3_NN,10], stddev=0.1))
b4 = tf.Variable(tf.zeros([10]))
```




```
# 计算第1隐藏层结果
Y1 = tf.nn.relu(tf.matmul(x, W1) + b1)

# 计算第2隐藏层结果
Y2 = tf.nn.relu(tf.matmul(Y1, W2) + b2)

# 计算第3隐藏层结果
Y3 = tf.nn.relu(tf.matmul(Y2, W3) + b3)

# 计算输出结果
forward = tf.matmul(Y3, W4) + b4
pred = tf.nn.softmax(forward)
```

定义全连接层函数

```
# 定义全连接层函数

def fcn_layer(inputs,            # 输入数据
              input_dim,         # 输入神经元数量
              output_dim,        # 输出神经元数量
              activation=None):  # 激活函数

    W = tf.Variable(tf.truncated_normal([input_dim, output_dim], stddev=0.1))
                                    # 以截断正态分布的随机数初始化W
    b = tf.Variable(tf.zeros([output_dim]))
                                    # 以0初始化b

    XWb = tf.matmul(inputs, W) + b  # 建立表达式: inputs * W + b

    if activation is None: # 默认不使用激活函数
        outputs = XWb
    else:                  # 若传入激活函数，则用其对输出结果进行变换
        outputs = activation(XWb)

    return outputs
```

构建模型

单隐层模型

构建输入层

```
x = tf.placeholder(tf.float32, [None, 784], name="X")
```

构建隐藏层

```
# 隐藏层包含256个神经元
h1=fcn_layer(inputs=x,
             input_dim=784,
             output_dim=256,
             activation=tf.nn.relu)
```

构建输出层

```
forward=fcn_layer(inputs=h1,
                  input_dim=256,
                  output_dim=10,
                  activation=None)

pred = tf.nn.softmax(forward)
```

构建模型

双隐层模型

构建输入层

```
x = tf.placeholder(tf.float32, [None, 784], name="X")
```

```
H1_NN = 256    # 第1隐藏层神经元为 256 个
H2_NN = 64     # 第2隐藏层神经元为 64 个
```

构建隐藏层1

```
h1=fcn_layer(inputs=x,
             input_dim=784,
             output_dim=H1_NN,
             activation=tf.nn.relu)
```

构建隐藏层2

```
h2=fcn_layer(inputs=h1,
             input_dim=H1_NN,
             output_dim=H2_NN,
             activation=tf.nn.relu)
```

构建输出层

```
forward=fcn_layer(inputs=h2,
                  input_dim=H2_NN,
                  output_dim=10,
                  activation=None)

pred = tf.nn.softmax(forward)
```

构建模型

三隐层模型

浙江大学城市学院
ZHEJIANG UNIVERSITY CITY COLLEGE

构建输入层

```
x = tf.placeholder(tf.float32, [None, 784], name="X")
```

```
H1_NN = 256    # 第1隐藏层神经元为 256 个
H2_NN = 64     # 第2隐藏层神经元为 64 个
H3_NN = 32     # 第3隐藏层神经元为 32 个
```

构建隐藏层1

```
h1=fcn_layer(inputs=x,
             input_dim=784,
             output_dim=H1_NN,
             activation=tf.nn.relu)
```

构建隐藏层2

```
h2=fcn_layer(inputs=h1,
             input_dim=H1_NN,
             output_dim=H2_NN,
             activation=tf.nn.relu)
```

构建隐藏层3

```
h3=fcn_layer(inputs=h2,
             input_dim=H2_NN,
             output_dim=H3_NN,
             activation=tf.nn.relu)
```

构建输出层

```
forward=fcn_layer(inputs=h3,
                  input_dim=H3_NN,
                  output_dim=10,
                  activation=None)

pred = tf.nn.softmax(forward)
```

训练模型的保存

初始化参数和文件目录

```
# 存储模型的粒度
save_step=5

# 创建保存模型文件的目录
import os
ckpt_dir = "./ckpt_dir/"
if not os.path.exists(ckpt_dir):
    os.makedirs(ckpt_dir)
```

训练模型

训练并存储模型

```
# 声明完所有变量后，调用tf.train.Saver
saver = tf.train.Saver()
```

训练模型

```
    if (epoch+1) % display_step == 0:
        print("Train Epoch:", '%02d' % (epoch+1),
              "Loss=", "{:.9f}".format(loss)," Accuracy=","{:.4f}".format(acc))

    if (epoch+1) % save_step == 0:
            saver.save(sess, os.path.join(ckpt_dir,
                                          'mnist_h256_model_{:06d}.ckpt'.format(epoch+1)))#存储模型
            print('mnist_h256_model_{:06d}.ckpt saved'.format(epoch+1))

saver.save(sess, os.path.join(ckpt_dir, 'mnist_h256_model.ckpt'))
print("Model saved!")

# 显示运行总时间
duration =time()-startTime
print("Train Finished takes:","{:.2f}".format(duration))
```

修改后的训练过程

```python
# 声明完所有变量后，调用tf.train.Saver
saver = tf.train.Saver()

# 记录训练开始时间
from time import time
startTime=time()

sess = tf.Session()
sess.run(tf.global_variables_initializer())

for epoch in range(train_epochs):
    for batch in range(total_batch):
        xs, ys = mnist.train.next_batch(batch_size)# 读取批次数据
        sess.run(optimizer,feed_dict={x: xs,y: ys}) # 执行批次训练

    #total_batch个批次训练完成后，使用验证数据计算误差与准确率
    loss,acc = sess.run([loss_function,accuracy],
                        feed_dict={x: mnist.validation.images,
                                   y: mnist.validation.labels})

    if (epoch+1) % display_step == 0:
        print("Train Epoch:", '%02d' % (epoch+1),
            "Loss=", "{:.9f}".format(loss)," Accuracy=","{:.4f}".format(acc))

    if (epoch+1) % save_step == 0:
            saver.save(sess, os.path.join(ckpt_dir,
                                   'mnist_h256_model_{:06d}.ckpt'.format(epoch+1)))#存储模型
            print('mnist_h256_model_{:06d}.ckpt saved'.format(epoch+1))

saver.save(sess, os.path.join(ckpt_dir, 'mnist_h256_model.ckpt'))
print("Model saved!")

# 显示运行总时间
duration =time()-startTime
print("Train Finished takes:","{:.2f}".format(duration))
```

训练模型

des › TF_ZUCC_6_MNIST_H256 › ckpt_dir

名称	修改日期	类型	大小
checkpoint	2018/11/8 19:59	文件	1 KB
mnist_h256_model.ckpt.data-00000-of-000	2018/11/8 19:59	DATA-00000-OF-0...	2,386 KB
mnist_h256_model.ckpt.index	2018/11/8 19:59	INDEX 文件	1 KB
mnist_h256_model.ckpt.meta	2018/11/8 19:59	META 文件	40 KB
mnist_h256_model_000040.ckpt.data-0000...	2018/11/8 19:59	DATA-00000-OF-0...	2,386 KB
mnist_h256_model_000040.ckpt.index	2018/11/8 19:59	INDEX 文件	1 KB
mnist_h256_model_000040.ckpt.meta	2018/11/8 19:59	META 文件	40 KB
mnist_h256_model_000035.ckpt.data-0000...	2018/11/8 19:59	DATA-00000-OF-0...	2,386 KB
mnist_h256_model_000035.ckpt.index	2018/11/8 19:59	INDEX 文件	1 KB
mnist_h256_model_000035.ckpt.meta	2018/11/8 19:59	META 文件	40 KB
mnist_h256_model_000030.ckpt.data-0000...	2018/11/8 19:59	DATA-00000-OF-0...	2,386 KB
mnist_h256_model_000030.ckpt.index	2018/11/8 19:59	INDEX 文件	1 KB
mnist_h256_model_000030.ckpt.meta	2018/11/8 19:59	META 文件	40 KB
mnist_h256_model_000025.ckpt.data-0000...	2018/11/8 19:59	DATA-00000-OF-0...	2,386 KB
mnist_h256_model_000025.ckpt.index	2018/11/8 19:59	INDEX 文件	1 KB
mnist_h256_model_000025.ckpt.meta	2018/11/8 19:59	META 文件	

缺省最多保留最近5次的

训练模型的还原与应用

定义相同结构的模型

构建输入层

```
x = tf.placeholder(tf.float32, [None, 784], name="X")
```

构建隐藏层

```
# 隐藏层包含256个神经元
h1=fcn_layer(inputs=x,
             input_dim=784,
             output_dim=256,
             activation=tf.nn.relu)
```

构建输出层

```
forward=fcn_layer(inputs=h1,
                  input_dim=256,
                  output_dim=10,
                  activation=None)

pred = tf.nn.softmax(forward)
```

设置模型文件的存放目录

设置目录

```
# 必须指定为模型文件的存放目录
ckpt_dir = "./ckpt_dir/"
```

des › TF_ZUCC_6_MNIST_H256 › ckpt_dir

名称	修改日期	类型	大小
checkpoint	2018/11/8 19:59	文件	1 KB
mnist_h256_model.ckpt.data-00000-of-000	2018/11/8 19:59	DATA-00000-OF-0...	2,386 KB
mnist_h256_model.ckpt.index	2018/11/8 19:59	INDEX 文件	1 KB
mnist_h256_model.ckpt.meta	2018/11/8 19:59	META 文件	40 KB
mnist_h256_model_000040.ckpt.data-0000...	2018/11/8 19:59	DATA-00000-OF-0...	2,386 KB
mnist_h256_model_000040.ckpt.index	2018/11/8 19:59	INDEX 文件	1 KB
mnist_h256_model_000040.ckpt.meta	2018/11/8 19:59	META 文件	40 KB
mnist_h256_model_000035.ckpt.data-0000...	2018/11/8 19:59	DATA-00000-OF-0...	2,386 KB
mnist_h256_model_000035.ckpt.index	2018/11/8 19:59	INDEX 文件	1 KB
mnist_h256_model_000035.ckpt.meta	2018/11/8 19:59	META 文件	40 KB
mnist_h256_model_000030.ckpt.data-0000...	2018/11/8 19:59	DATA-00000-OF-0...	2,386 KB
mnist_h256_model_000030.ckpt.index	2018/11/8 19:59	INDEX 文件	1 KB
mnist_h256_model_000030.ckpt.meta	2018/11/8 19:59	META 文件	40 KB
mnist_h256_model_000025.ckpt.data-0000...	2018/11/8 19:59	DATA-00000-OF-0...	2,386 KB
mnist_h256_model_000025.ckpt.index	2018/11/8 19:59	INDEX 文件	1 KB
mnist_h256_model_000025.ckpt.meta	2018/11/8 19:59	META 文件	40 KB

缺省最多保留最近5份

读取还原模型

读取模型

```
# 创建saver
saver = tf.train.Saver()

sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)

ckpt = tf.train.get_checkpoint_state(ckpt_dir)

if ckpt and ckpt.model_checkpoint_path:
    saver.restore(sess, ckpt.model_checkpoint_path)# 从已保存的模型中读取参数
    print("Restore model from "+ckpt.model_checkpoint_path)
```

输出还原模型的准确率

输出模型准确率

```
print ("Accuracy:", accuracy.eval(session=sess,
                                  feed_dict={x: mnist.test.images, y: mnist.test.labels}))
```

Accuracy: 0.9755