# 实验zero

## 实验任务

将MNIST机器学习入门中的简单代码，使用tensorboard记录训练过程并可视化显示。

1. 训练次数：1000次
2. 采样距离：50次
3. 计算图结构：输入层、隐藏层1、输出层

## 实验步骤

1. 将下列源代码存为experiment\_zero.py，将训练的日志数据写入文件夹exp\_zero\_logs：

"""

本实验在experiment.py的基础上，做了如下改动：

@使用了tensorboard

"""

import input\_data

mnist = input\_data.read\_data\_sets("MNIST\_data/", one\_hot=True)

import tensorflow as tf

# soft回归模型

# 第一层

with tf.name\_scope('input\_layer') :

x = tf.placeholder(tf.float32,[None,784]) # 60000x784的矩阵

y\_ = tf.placeholder("float",[None,10]) # 60000x10 的矩阵

with tf.name\_scope('hidden\_layer1') :

with tf.name\_scope('weight'):

W1 = tf.Variable(tf.zeros([784,10])) # 784x10的矩阵

tf.summary.histogram('hidden\_layer1/weight',W1)

with tf.name\_scope('weight'):

b1 = tf.Variable(tf.zeros([10])) # 长度为10的行矩阵

tf.summary.histogram('hidden\_layer1/weight',b1)

with tf.name\_scope('W1x1b1'):

W1x1b1 = tf.matmul(x,W1)+b1 # matmul(x,W1) 为60000x10的矩阵，每行加上偏置行向量b1

tf.summary.histogram('hidden\_layer/W1x1b1',W1x1b1)

y = tf.nn.softmax(W1x1b1) #激活函数,经softmax后，变为归一化的 60000x10 的概率矩阵

# 训练模型

# 用交叉熵作为损失函数

with tf.name\_scope('loss'):

loss = - tf.reduce\_sum(y\_ \* tf.log(y))

tf.summary.scalar('loss',loss)

with tf.name\_scope('train'):

train\_step = tf.train.GradientDescentOptimizer(0.01).minimize(loss)

# 评估模型

with tf.name\_scope('estimation'):

correct\_prediction = tf.equal(tf.argmax(y,1),tf.argmax(y\_,1))

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction,"float"))

tf.summary.scalar('accuracy',accuracy)

init = tf.initialize\_all\_variables()

sess = tf.Session()

sess.run(init)

merged = tf.summary.merge\_all() #将图形、训练过程等数据合并在一起

train\_writer = tf.summary.FileWriter('exp\_zero\_train\_logs',sess.graph) #将训练日志写入到logs文件夹下

# valid\_writer = tf.summary.FileWriter('valid\_logs',sess.graph) #将训练日志写入到logs文件夹下

for i in range(1000):

batch\_xs, batch\_ys = mnist.train.next\_batch(100)

sess.run(train\_step,feed\_dict = {x:batch\_xs, y\_ :batch\_ys})

if(i%50==0):

result = sess.run(merged,feed\_dict={x:batch\_xs,y\_:batch\_ys}) #计算需要写入的日志数据

#result1 = sess.run(accuracy,feed\_dict={x:batch\_xs,y\_:batch\_ys})

train\_writer.add\_summary(result,i) #将日志数据写入文件

#valid\_writer.add\_summary(result1,i) #将日志数据写入文件

1. 在tensorflow环境下运行源码

在anaconda prompt中输入命令python experiment\_zero

1. 使用tensorboard分析日志文件

在anaconda prompt中输入命令tensorboard –logdir==exp\_zero\_logs

## 实验结果



图 1 准确率与损失函数

从图 1中可以看出损失函数和准确率的确是逐步收敛的，精度收敛值不是很好。

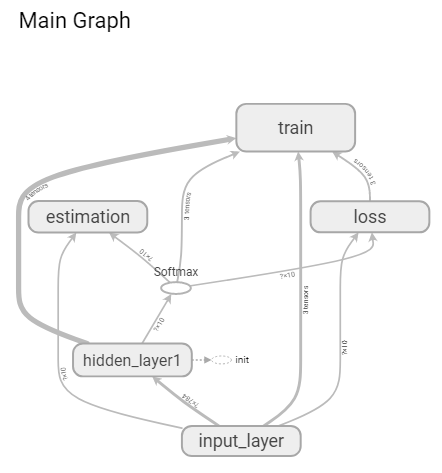


图 2 计算图

## 增加训练次数再实验

将训练次数修改为10000次，每50次记录训练数据，实验结果为：

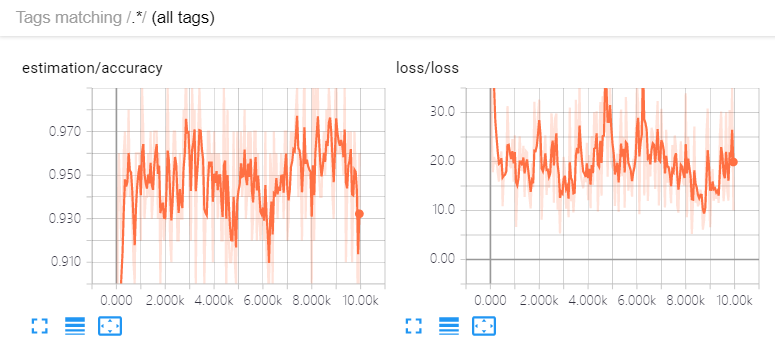


图 3

从图 3中看到，在1000次以前迅速收敛，1000次以后在0.95附近波动

将训练次数修改为10000次，每500次记录训练数据，实验结果为：



图 4

从图 4中看到同样是训练10000次，采样间隔修改为500以后，收敛趋势还是比较明显的，而且收敛值比原始实验收敛值高。

# 实验one

## 实验任务

在实验zero的基础上，增加隐藏层2

1. 训练次数：1000次
2. 采样距离：50次
3. 计算图结构：输入层、隐藏层1、隐藏层2、输出层

## 实验步骤

1. 将下列代码保存为experiment\_one.py, 将训练的日志数据写入文件夹exp\_one\_train\_logs：

"""

本实验在experiment.py的基础上，做了如下改动：

@增加了两个隐藏层

@使用了tensorboard

"""

import input\_data

mnist = input\_data.read\_data\_sets("MNIST\_data/", one\_hot=True)

import tensorflow as tf

# soft回归模型

# 第一层

with tf.name\_scope('input\_layer') :

x = tf.placeholder(tf.float32,[None,784]) # 60000x784的矩阵

y\_ = tf.placeholder("float",[None,10]) # 60000x10 的矩阵

with tf.name\_scope('hidden\_layer1') :

with tf.name\_scope('weight'):

W1 = tf.Variable(tf.zeros([784,10])) # 784x10的矩阵

tf.summary.histogram('hidden\_layer1/weight',W1)

with tf.name\_scope('weight'):

b1 = tf.Variable(tf.zeros([10])) # 长度为10的行矩阵

tf.summary.histogram('hidden\_layer1/weight',b1)

with tf.name\_scope('W1x1b1'):

W1x1b1 = tf.matmul(x,W1)+b1 # matmul(x,W1) 为60000x10的矩阵，每行加上偏置行向量b1

tf.summary.histogram('hidden\_layer/W1x1b1',W1x1b1)

x2 = tf.nn.relu(W1x1b1) #激活函数 60000x10的矩阵

with tf.name\_scope('hidden\_layer2') :

with tf.name\_scope('weight'):

W2 = tf.Variable(tf.zeros([10,784])) # 10x784的矩阵

tf.summary.histogram('hidden\_layer2/weight',W2)

with tf.name\_scope('weight'):

b2 = tf.Variable(tf.zeros([784])) # 长度为784的行矩阵

tf.summary.histogram('hidden\_layer2/weight',b2)

with tf.name\_scope('W2x2b2'):

W2x2b2 = tf.matmul(x2,W2)+b2 # matmul(x,W1) 为60000x784的矩阵，每行加上偏置行向量b2

tf.summary.histogram('hidden\_layer2/W2x2b2',W2x2b2)

x3 = tf.nn.relu(W2x2b2) #激活函数 60000x784的矩阵

with tf.name\_scope('output\_layer') :

with tf.name\_scope('weight'):

W = tf.Variable(tf.zeros([784,10])) # 784x10的矩阵

tf.summary.histogram('output\_layer/weight',W)

with tf.name\_scope('weight'):

b = tf.Variable(tf.zeros([10])) # 长度为10的行矩阵

tf.summary.histogram('output\_layer/weight',b)

with tf.name\_scope('Wxb'):

Wxb = tf.matmul(x3,W)+b # matmul(x,W1) 为60000x10的矩阵，每行加上偏置行向量b

tf.summary.histogram('output\_layer/Wxb',Wxb)

y = tf.nn.softmax(Wxb) #激活函数,经softmax后，变为归一化的 60000x10 的概率矩阵

# 训练模型

# 用交叉熵作为损失函数

with tf.name\_scope('loss'):

loss = - tf.reduce\_sum(y\_ \* tf.log(y))

tf.summary.scalar('loss',loss)

with tf.name\_scope('train'):

train\_step = tf.train.GradientDescentOptimizer(0.01).minimize(loss)

# 评估模型

with tf.name\_scope('estimation'):

correct\_prediction = tf.equal(tf.argmax(y,1),tf.argmax(y\_,1))

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction,"float"))

tf.summary.scalar('accuracy',accuracy)

init = tf.initialize\_all\_variables()

sess = tf.Session()

sess.run(init)

merged = tf.summary.merge\_all() #将图形、训练过程等数据合并在一起

train\_writer = tf.summary.FileWriter('exp\_one\_train\_logs',sess.graph) #将训练日志写入到logs文件夹下

# valid\_writer = tf.summary.FileWriter('valid\_logs',sess.graph) #将训练日志写入到logs文件夹下

for i in range(1000):

batch\_xs, batch\_ys = mnist.train.next\_batch(100)

sess.run(train\_step,feed\_dict = {x:batch\_xs, y\_ :batch\_ys})

if(i%50==0):

result = sess.run(merged,feed\_dict={x:batch\_xs,y\_:batch\_ys}) #计算需要写入的日志数据

#result1 = sess.run(accuracy,feed\_dict={x:batch\_xs,y\_:batch\_ys})

train\_writer.add\_summary(result,i) #将日志数据写入文件

#valid\_writer.add\_summary(result1,i) #将日志数据写入文件

1. 在tensorflow环境下运行源码

在anaconda prompt中输入命令python experiment\_zero

1. 使用tensorboard分析日志文件

在anaconda prompt中输入命令

tensorboard --logdir==exp\_zero\_train\_logs

## 实验结果

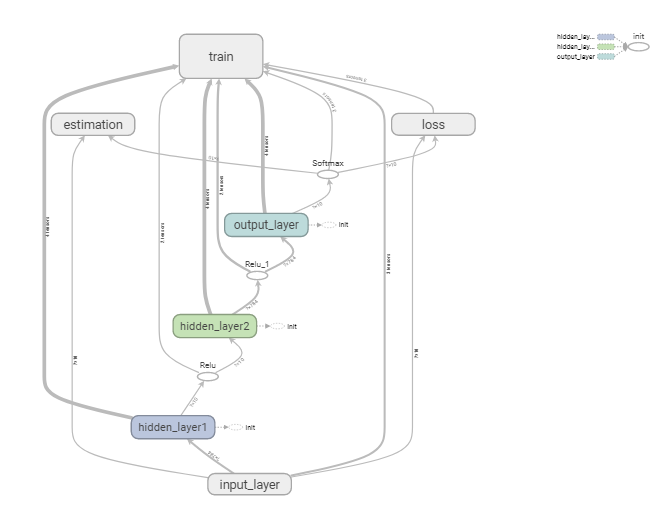


图 5 计算图

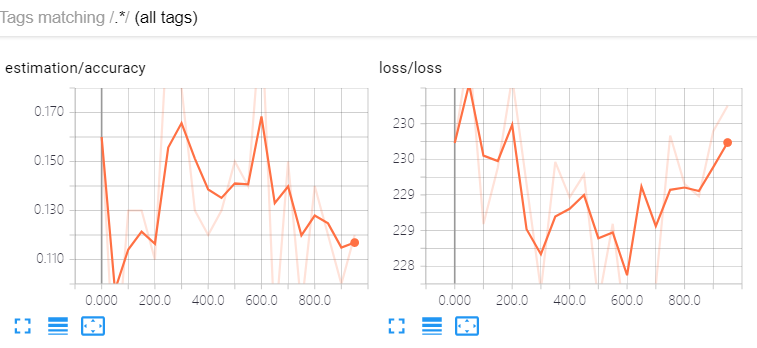


图 6 准确率与损失函数

## 增加训练次数再实验

将训练次数修改为10000次，每50次记录训练数据，实验结果为：

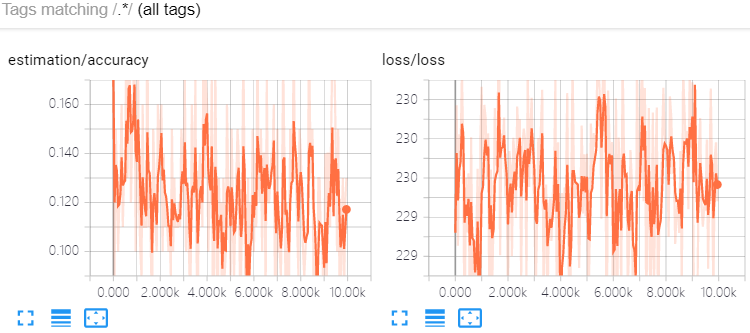


图 7

将训练次数修改为10000次，每500次记录训练数据，实验结果为：

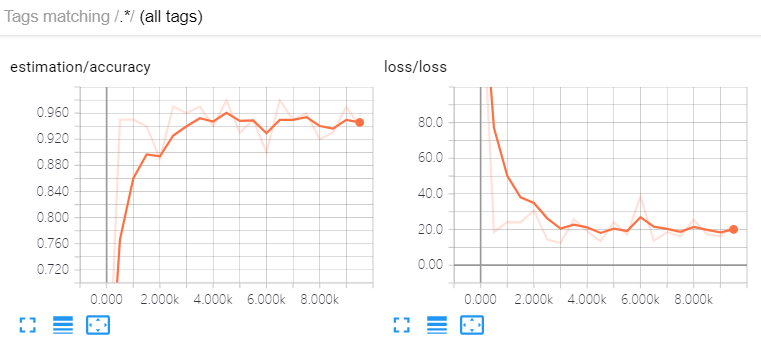


图 8

将训练次数修改为1000次，每5次记录训练数据，实验结果为：



图 9

将训练次数修改为1000次，每100次记录训练数据，实验结果为：



图 10

## 结论

模型变复杂以后，训练次数和采样距离也要跟着增大才能有充分的时间令损失函数和效率收敛。

# 实验two

## 实验任务

在实验zero的基础上增加模型的保存与载入

参考教程<http://blog.csdn.net/u014595019/article/details/53912710>



图 11

## 实验步骤

1. 将下列代码保存为experiment\_two.py, 将训练得到的模型写入文件夹“exp\_two\_model\_savepath\”

"""

本实验在experiment\_zero.py的基础上，做了如下改动：

@增加了模型保存与载入功能

"""

import input\_data

mnist = input\_data.read\_data\_sets("MNIST\_data/", one\_hot=True)

import tensorflow as tf

# soft回归模型

# 第一层

with tf.name\_scope('input\_layer') :

x = tf.placeholder(tf.float32,[None,784]) # 60000x784的矩阵

y\_ = tf.placeholder("float",[None,10]) # 60000x10 的矩阵

with tf.name\_scope('hidden\_layer1') :

with tf.name\_scope('weight'):

W1 = tf.Variable(tf.zeros([784,10])) # 784x10的矩阵

tf.summary.histogram('hidden\_layer1/weight',W1)

with tf.name\_scope('weight'):

b1 = tf.Variable(tf.zeros([10])) # 长度为10的行矩阵

tf.summary.histogram('hidden\_layer1/weight',b1)

with tf.name\_scope('W1x1b1'):

W1x1b1 = tf.matmul(x,W1)+b1 # matmul(x,W1) 为60000x10的矩阵，每行加上偏置行向量b1

tf.summary.histogram('hidden\_layer/W1x1b1',W1x1b1)

y = tf.nn.softmax(W1x1b1) #激活函数,经softmax后，变为归一化的 60000x10 的概率矩阵

# 训练模型

# 用交叉熵作为损失函数

with tf.name\_scope('loss'):

loss = - tf.reduce\_sum(y\_ \* tf.log(y))

tf.summary.scalar('loss',loss)

with tf.name\_scope('train'):

train\_step = tf.train.GradientDescentOptimizer(0.01).minimize(loss)

# 评估模型

with tf.name\_scope('estimation'):

correct\_prediction = tf.equal(tf.argmax(y,1),tf.argmax(y\_,1))

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction,"float"))

tf.summary.scalar('accuracy',accuracy)

saver = tf.train.Saver() # 生成 saver

init = tf.initialize\_all\_variables()

sess = tf.Session()

sess.run(init)

merged = tf.summary.merge\_all() #将图形、训练过程等数据合并在一起

train\_writer = tf.summary.FileWriter('exp\_zero\_train\_logs',sess.graph) #将训练日志写入到logs文件夹下

# valid\_writer = tf.summary.FileWriter('valid\_logs',sess.graph) #将训练日志写入到logs文件夹下

for i in range(10000):

batch\_xs, batch\_ys = mnist.train.next\_batch(100)

sess.run(train\_step,feed\_dict = {x:batch\_xs, y\_ :batch\_ys})

if(i%500==0):

result = sess.run(merged,feed\_dict={x:batch\_xs,y\_:batch\_ys}) #计算需要写入的日志数据

#result1 = sess.run(accuracy,feed\_dict={x:batch\_xs,y\_:batch\_ys})

train\_writer.add\_summary(result,i) #将日志数据写入文件

#valid\_writer.add\_summary(result1,i) #将日志数据写入文件

saver.save(sess,"exp\_two\_model\_savepath/")

1. 输入命令python experiment\_two执行脚本训练模型
2. 将下列代码作为测试文件experiment\_two\_check.py

"""

本实验在experiment\_two.py的基础上，做了如下改动：

@测试experiment\_two的模型保存与载入功能

"""

import input\_data

mnist = input\_data.read\_data\_sets("MNIST\_data/", one\_hot=True)

import tensorflow as tf

# soft回归模型

# 第一层

with tf.name\_scope('input\_layer') :

x = tf.placeholder(tf.float32,[None,784]) # 60000x784的矩阵

y\_ = tf.placeholder("float",[None,10]) # 60000x10 的矩阵

with tf.name\_scope('hidden\_layer1') :

with tf.name\_scope('weight'):

W1 = tf.Variable(tf.zeros([784,10])) # 784x10的矩阵

#tf.summary.histogram('hidden\_layer1/weight',W1)

with tf.name\_scope('weight'):

b1 = tf.Variable(tf.zeros([10])) # 长度为10的行矩阵

#tf.summary.histogram('hidden\_layer1/weight',b1)

with tf.name\_scope('W1x1b1'):

W1x1b1 = tf.matmul(x,W1)+b1 # matmul(x,W1) 为60000x10的矩阵，每行加上偏置行向量b1

#tf.summary.histogram('hidden\_layer/W1x1b1',W1x1b1)

y = tf.nn.softmax(W1x1b1) #激活函数,经softmax后，变为归一化的 60000x10 的概率矩阵

# 训练模型

# 用交叉熵作为损失函数

with tf.name\_scope('loss'):

loss = - tf.reduce\_sum(y\_ \* tf.log(y))

#tf.summary.scalar('loss',loss)

with tf.name\_scope('train'):

train\_step = tf.train.GradientDescentOptimizer(0.01).minimize(loss)

# 评估模型

with tf.name\_scope('estimation'):

correct\_prediction = tf.equal(tf.argmax(y,1),tf.argmax(y\_,1))

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction,"float"))

#tf.summary.scalar('accuracy',accuracy)

saver = tf.train.Saver() # 生成 saver

init = tf.initialize\_all\_variables()

sess = tf.Session()

sess.run(init)

#merged = tf.summary.merge\_all() #将图形、训练过程等数据合并在一起

saver.restore(sess,"exp\_two\_model\_savepath/")

print (sess.run(accuracy,feed\_dict = {x:mnist.test.images,y\_:mnist.test.labels}))

#for i in range(10000):

#batch\_xs, batch\_ys = mnist.train.next\_batch(100)

#sess.run(train\_step,feed\_dict = {x:batch\_xs, y\_ :batch\_ys})

#if(i%500==0):

# result = sess.run(merged,feed\_dict={x:batch\_xs,y\_:batch\_ys}) #计算需要写入的日志数据

#result1 = sess.run(accuracy,feed\_dict={x:batch\_xs,y\_:batch\_ys})

# train\_writer.add\_summary(result,i) #将日志数据写入文件

#valid\_writer.add\_summary(result1,i) #将日志数据写入文件

## 实验结果



图 12 模型文件

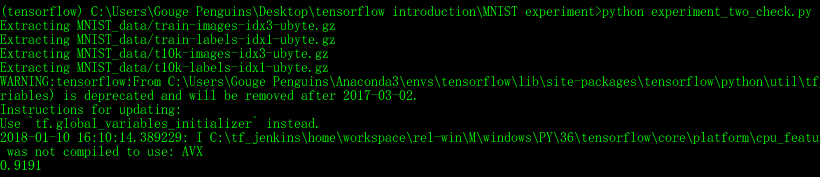


图 13 测试文件运行结果

# 实验three

## 实验任务

在实验one的基础上，变更激活函数为图 13 激活函数，并使用tensorboard查看收敛情况

* 训练次数10000
* 采样距离500

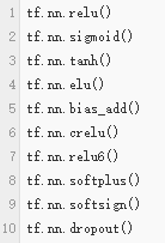


图 14 激活函数

## 实验步骤

1. 将激活函数设置为sigmoid，依照实验one中的步骤，训练模型并查看收敛情况。

源码：

"""

本实验在experiment\_one.py的基础上，做了如下改动：

@更换了激活函数为sigmoid

"""

import input\_data

mnist = input\_data.read\_data\_sets("MNIST\_data/", one\_hot=True)

import tensorflow as tf

# soft回归模型

# 第一层

with tf.name\_scope('input\_layer') :

x = tf.placeholder(tf.float32,[None,784]) # 60000x784的矩阵

y\_ = tf.placeholder("float",[None,10]) # 60000x10 的矩阵

with tf.name\_scope('hidden\_layer1') :

with tf.name\_scope('weight'):

W1 = tf.Variable(tf.zeros([784,10])) # 784x10的矩阵

tf.summary.histogram('hidden\_layer1/weight',W1)

with tf.name\_scope('weight'):

b1 = tf.Variable(tf.zeros([10])) # 长度为10的行矩阵

tf.summary.histogram('hidden\_layer1/weight',b1)

with tf.name\_scope('W1x1b1'):

W1x1b1 = tf.matmul(x,W1)+b1 # matmul(x,W1) 为60000x10的矩阵，每行加上偏置行向量b1

tf.summary.histogram('hidden\_layer/W1x1b1',W1x1b1)

x2 = tf.nn.sigmoid(W1x1b1) #激活函数 60000x10的矩阵

with tf.name\_scope('hidden\_layer2') :

with tf.name\_scope('weight'):

W2 = tf.Variable(tf.zeros([10,784])) # 10x784的矩阵

tf.summary.histogram('hidden\_layer2/weight',W2)

with tf.name\_scope('weight'):

b2 = tf.Variable(tf.zeros([784])) # 长度为784的行矩阵

tf.summary.histogram('hidden\_layer2/weight',b2)

with tf.name\_scope('W2x2b2'):

W2x2b2 = tf.matmul(x2,W2)+b2 # matmul(x,W1) 为60000x784的矩阵，每行加上偏置行向量b2

tf.summary.histogram('hidden\_layer2/W2x2b2',W2x2b2)

x3 = tf.nn.sigmoid(W2x2b2) #激活函数 60000x784的矩阵

with tf.name\_scope('output\_layer') :

with tf.name\_scope('weight'):

W = tf.Variable(tf.zeros([784,10])) # 784x10的矩阵

tf.summary.histogram('output\_layer/weight',W)

with tf.name\_scope('weight'):

b = tf.Variable(tf.zeros([10])) # 长度为10的行矩阵

tf.summary.histogram('output\_layer/weight',b)

with tf.name\_scope('Wxb'):

Wxb = tf.matmul(x3,W)+b # matmul(x,W1) 为60000x10的矩阵，每行加上偏置行向量b

tf.summary.histogram('output\_layer/Wxb',Wxb)

y = tf.nn.softmax(Wxb) #激活函数,经softmax后，变为归一化的 60000x10 的概率矩阵

# 训练模型

# 用交叉熵作为损失函数

with tf.name\_scope('loss'):

loss = - tf.reduce\_sum(y\_ \* tf.log(y))

tf.summary.scalar('loss',loss)

with tf.name\_scope('train'):

train\_step = tf.train.GradientDescentOptimizer(0.01).minimize(loss)

# 评估模型

with tf.name\_scope('estimation'):

correct\_prediction = tf.equal(tf.argmax(y,1),tf.argmax(y\_,1))

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction,"float"))

tf.summary.scalar('accuracy',accuracy)

init = tf.initialize\_all\_variables()

sess = tf.Session()

sess.run(init)

merged = tf.summary.merge\_all() #将图形、训练过程等数据合并在一起

train\_writer = tf.summary.FileWriter('exp\_three\_sigmoid\_train\_logs',sess.graph) #将训练日志写入到logs文件夹下

# valid\_writer = tf.summary.FileWriter('valid\_logs',sess.graph) #将训练日志写入到logs文件夹下

for i in range(1000):

batch\_xs, batch\_ys = mnist.train.next\_batch(100)

sess.run(train\_step,feed\_dict = {x:batch\_xs, y\_ :batch\_ys})

if(i%100==0):

result = sess.run(merged,feed\_dict={x:batch\_xs,y\_:batch\_ys}) #计算需要写入的日志数据

#result1 = sess.run(accuracy,feed\_dict={x:batch\_xs,y\_:batch\_ys})

train\_writer.add\_summary(result,i) #将日志数据写入文件

#valid\_writer.add\_summary(result1,i) #将日志数据写入文件

1. 将激活函数设置为tanh，依照实验one中的步骤，训练模型并查看收敛情况。

"""

本实验在experiment\_one.py的基础上，做了如下改动：

@更换了激活函数为tanh

"""

import input\_data

mnist = input\_data.read\_data\_sets("MNIST\_data/", one\_hot=True)

import tensorflow as tf

# soft回归模型

# 第一层

with tf.name\_scope('input\_layer') :

x = tf.placeholder(tf.float32,[None,784]) # 60000x784的矩阵

y\_ = tf.placeholder("float",[None,10]) # 60000x10 的矩阵

with tf.name\_scope('hidden\_layer1') :

with tf.name\_scope('weight'):

W1 = tf.Variable(tf.zeros([784,10])) # 784x10的矩阵

tf.summary.histogram('hidden\_layer1/weight',W1)

with tf.name\_scope('weight'):

b1 = tf.Variable(tf.zeros([10])) # 长度为10的行矩阵

tf.summary.histogram('hidden\_layer1/weight',b1)

with tf.name\_scope('W1x1b1'):

W1x1b1 = tf.matmul(x,W1)+b1 # matmul(x,W1) 为60000x10的矩阵，每行加上偏置行向量b1

tf.summary.histogram('hidden\_layer/W1x1b1',W1x1b1)

x2 = tf.nn.tanh(W1x1b1) #激活函数 60000x10的矩阵

with tf.name\_scope('hidden\_layer2') :

with tf.name\_scope('weight'):

W2 = tf.Variable(tf.zeros([10,784])) # 10x784的矩阵

tf.summary.histogram('hidden\_layer2/weight',W2)

with tf.name\_scope('weight'):

b2 = tf.Variable(tf.zeros([784])) # 长度为784的行矩阵

tf.summary.histogram('hidden\_layer2/weight',b2)

with tf.name\_scope('W2x2b2'):

W2x2b2 = tf.matmul(x2,W2)+b2 # matmul(x,W1) 为60000x784的矩阵，每行加上偏置行向量b2

tf.summary.histogram('hidden\_layer2/W2x2b2',W2x2b2)

x3 = tf.nn.tanh(W2x2b2) #激活函数 60000x784的矩阵

with tf.name\_scope('output\_layer') :

with tf.name\_scope('weight'):

W = tf.Variable(tf.zeros([784,10])) # 784x10的矩阵

tf.summary.histogram('output\_layer/weight',W)

with tf.name\_scope('weight'):

b = tf.Variable(tf.zeros([10])) # 长度为10的行矩阵

tf.summary.histogram('output\_layer/weight',b)

with tf.name\_scope('Wxb'):

Wxb = tf.matmul(x3,W)+b # matmul(x,W1) 为60000x10的矩阵，每行加上偏置行向量b

tf.summary.histogram('output\_layer/Wxb',Wxb)

y = tf.nn.softmax(Wxb) #激活函数,经softmax后，变为归一化的 60000x10 的概率矩阵

# 训练模型

# 用交叉熵作为损失函数

with tf.name\_scope('loss'):

loss = - tf.reduce\_sum(y\_ \* tf.log(y))

tf.summary.scalar('loss',loss)

with tf.name\_scope('train'):

train\_step = tf.train.GradientDescentOptimizer(0.01).minimize(loss)

# 评估模型

with tf.name\_scope('estimation'):

correct\_prediction = tf.equal(tf.argmax(y,1),tf.argmax(y\_,1))

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction,"float"))

tf.summary.scalar('accuracy',accuracy)

init = tf.initialize\_all\_variables()

sess = tf.Session()

sess.run(init)

merged = tf.summary.merge\_all() #将图形、训练过程等数据合并在一起

train\_writer = tf.summary.FileWriter('exp\_three\_tanh\_train\_logs',sess.graph) #将训练日志写入到logs文件夹下

# valid\_writer = tf.summary.FileWriter('valid\_logs',sess.graph) #将训练日志写入到logs文件夹下

for i in range(100000):

batch\_xs, batch\_ys = mnist.train.next\_batch(100)

sess.run(train\_step,feed\_dict = {x:batch\_xs, y\_ :batch\_ys})

if(i%5000==0):

result = sess.run(merged,feed\_dict={x:batch\_xs,y\_:batch\_ys}) #计算需要写入的日志数据

#result1 = sess.run(accuracy,feed\_dict={x:batch\_xs,y\_:batch\_ys})

train\_writer.add\_summary(result,i) #将日志数据写入文件

#valid\_writer.add\_summary(result1,i) #将日志数据写入文件

1. 将激活函数设置为elu，依照实验one中的步骤，训练模型并查看收敛情况。

源码：

"""

本实验在experiment\_one.py的基础上，做了如下改动：

@更换了激活函数为elu

"""

import input\_data

mnist = input\_data.read\_data\_sets("MNIST\_data/", one\_hot=True)

import tensorflow as tf

# soft回归模型

# 第一层

with tf.name\_scope('input\_layer') :

x = tf.placeholder(tf.float32,[None,784]) # 60000x784的矩阵

y\_ = tf.placeholder("float",[None,10]) # 60000x10 的矩阵

with tf.name\_scope('hidden\_layer1') :

with tf.name\_scope('weight'):

W1 = tf.Variable(tf.zeros([784,10])) # 784x10的矩阵

tf.summary.histogram('hidden\_layer1/weight',W1)

with tf.name\_scope('weight'):

b1 = tf.Variable(tf.zeros([10])) # 长度为10的行矩阵

tf.summary.histogram('hidden\_layer1/weight',b1)

with tf.name\_scope('W1x1b1'):

W1x1b1 = tf.matmul(x,W1)+b1 # matmul(x,W1) 为60000x10的矩阵，每行加上偏置行向量b1

tf.summary.histogram('hidden\_layer/W1x1b1',W1x1b1)

x2 = tf.nn.elu(W1x1b1) #激活函数 60000x10的矩阵

with tf.name\_scope('hidden\_layer2') :

with tf.name\_scope('weight'):

W2 = tf.Variable(tf.zeros([10,784])) # 10x784的矩阵

tf.summary.histogram('hidden\_layer2/weight',W2)

with tf.name\_scope('weight'):

b2 = tf.Variable(tf.zeros([784])) # 长度为784的行矩阵

tf.summary.histogram('hidden\_layer2/weight',b2)

with tf.name\_scope('W2x2b2'):

W2x2b2 = tf.matmul(x2,W2)+b2 # matmul(x,W1) 为60000x784的矩阵，每行加上偏置行向量b2

tf.summary.histogram('hidden\_layer2/W2x2b2',W2x2b2)

x3 = tf.nn.elu(W2x2b2) #激活函数 60000x784的矩阵

with tf.name\_scope('output\_layer') :

with tf.name\_scope('weight'):

W = tf.Variable(tf.zeros([784,10])) # 784x10的矩阵

tf.summary.histogram('output\_layer/weight',W)

with tf.name\_scope('weight'):

b = tf.Variable(tf.zeros([10])) # 长度为10的行矩阵

tf.summary.histogram('output\_layer/weight',b)

with tf.name\_scope('Wxb'):

Wxb = tf.matmul(x3,W)+b # matmul(x,W1) 为60000x10的矩阵，每行加上偏置行向量b

tf.summary.histogram('output\_layer/Wxb',Wxb)

y = tf.nn.softmax(Wxb) #激活函数,经softmax后，变为归一化的 60000x10 的概率矩阵

# 训练模型

# 用交叉熵作为损失函数

with tf.name\_scope('loss'):

loss = - tf.reduce\_sum(y\_ \* tf.log(y))

tf.summary.scalar('loss',loss)

with tf.name\_scope('train'):

train\_step = tf.train.GradientDescentOptimizer(0.01).minimize(loss)

# 评估模型

with tf.name\_scope('estimation'):

correct\_prediction = tf.equal(tf.argmax(y,1),tf.argmax(y\_,1))

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction,"float"))

tf.summary.scalar('accuracy',accuracy)

init = tf.initialize\_all\_variables()

sess = tf.Session()

sess.run(init)

merged = tf.summary.merge\_all() #将图形、训练过程等数据合并在一起

train\_writer = tf.summary.FileWriter('exp\_three\_elu\_train\_logs',sess.graph) #将训练日志写入到logs文件夹下

# valid\_writer = tf.summary.FileWriter('valid\_logs',sess.graph) #将训练日志写入到logs文件夹下

for i in range(100000):

batch\_xs, batch\_ys = mnist.train.next\_batch(100)

sess.run(train\_step,feed\_dict = {x:batch\_xs, y\_ :batch\_ys})

if(i%5000==0):

result = sess.run(merged,feed\_dict={x:batch\_xs,y\_:batch\_ys}) #计算需要写入的日志数据

#result1 = sess.run(accuracy,feed\_dict={x:batch\_xs,y\_:batch\_ys})

train\_writer.add\_summary(result,i) #将日志数据写入文件

#valid\_writer.add\_summary(result1,i) #将日志数据写入文件

1. 将激活函数设置为relu6，依照实验one中的步骤，训练模型并查看收敛情况。

训练次数10000，采样距离500

源码：

"""

本实验在experiment\_one.py的基础上，做了如下改动：

@更换了激活函数为relu6

"""

import input\_data

mnist = input\_data.read\_data\_sets("MNIST\_data/", one\_hot=True)

import tensorflow as tf

# soft回归模型

# 第一层

with tf.name\_scope('input\_layer') :

x = tf.placeholder(tf.float32,[None,784]) # 60000x784的矩阵

y\_ = tf.placeholder("float",[None,10]) # 60000x10 的矩阵

with tf.name\_scope('hidden\_layer1') :

with tf.name\_scope('weight'):

W1 = tf.Variable(tf.zeros([784,10])) # 784x10的矩阵

tf.summary.histogram('hidden\_layer1/weight',W1)

with tf.name\_scope('weight'):

b1 = tf.Variable(tf.zeros([10])) # 长度为10的行矩阵

tf.summary.histogram('hidden\_layer1/weight',b1)

with tf.name\_scope('W1x1b1'):

W1x1b1 = tf.matmul(x,W1)+b1 # matmul(x,W1) 为60000x10的矩阵，每行加上偏置行向量b1

tf.summary.histogram('hidden\_layer/W1x1b1',W1x1b1)

x2 = tf.nn.relu6(W1x1b1) #激活函数 60000x10的矩阵

with tf.name\_scope('hidden\_layer2') :

with tf.name\_scope('weight'):

W2 = tf.Variable(tf.zeros([10,784])) # 10x784的矩阵

tf.summary.histogram('hidden\_layer2/weight',W2)

with tf.name\_scope('weight'):

b2 = tf.Variable(tf.zeros([784])) # 长度为784的行矩阵

tf.summary.histogram('hidden\_layer2/weight',b2)

with tf.name\_scope('W2x2b2'):

W2x2b2 = tf.matmul(x2,W2)+b2 # matmul(x,W1) 为60000x784的矩阵，每行加上偏置行向量b2

tf.summary.histogram('hidden\_layer2/W2x2b2',W2x2b2)

x3 = tf.nn.relu6(W2x2b2) #激活函数 60000x784的矩阵

with tf.name\_scope('output\_layer') :

with tf.name\_scope('weight'):

W = tf.Variable(tf.zeros([784,10])) # 784x10的矩阵

tf.summary.histogram('output\_layer/weight',W)

with tf.name\_scope('weight'):

b = tf.Variable(tf.zeros([10])) # 长度为10的行矩阵

tf.summary.histogram('output\_layer/weight',b)

with tf.name\_scope('Wxb'):

Wxb = tf.matmul(x3,W)+b # matmul(x,W1) 为60000x10的矩阵，每行加上偏置行向量b

tf.summary.histogram('output\_layer/Wxb',Wxb)

y = tf.nn.softmax(Wxb) #激活函数,经softmax后，变为归一化的 60000x10 的概率矩阵

# 训练模型

# 用交叉熵作为损失函数

with tf.name\_scope('loss'):

loss = - tf.reduce\_sum(y\_ \* tf.log(y))

tf.summary.scalar('loss',loss)

with tf.name\_scope('train'):

train\_step = tf.train.GradientDescentOptimizer(0.01).minimize(loss)

# 评估模型

with tf.name\_scope('estimation'):

correct\_prediction = tf.equal(tf.argmax(y,1),tf.argmax(y\_,1))

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction,"float"))

tf.summary.scalar('accuracy',accuracy)

init = tf.initialize\_all\_variables()

sess = tf.Session()

sess.run(init)

merged = tf.summary.merge\_all() #将图形、训练过程等数据合并在一起

train\_writer = tf.summary.FileWriter('exp\_three\_relu6\_train\_logs',sess.graph) #将训练日志写入到logs文件夹下

# valid\_writer = tf.summary.FileWriter('valid\_logs',sess.graph) #将训练日志写入到logs文件夹下

for i in range(10000):

batch\_xs, batch\_ys = mnist.train.next\_batch(100)

sess.run(train\_step,feed\_dict = {x:batch\_xs, y\_ :batch\_ys})

if(i%500==0):

result = sess.run(merged,feed\_dict={x:batch\_xs,y\_:batch\_ys}) #计算需要写入的日志数据

#result1 = sess.run(accuracy,feed\_dict={x:batch\_xs,y\_:batch\_ys})

train\_writer.add\_summary(result,i) #将日志数据写入文件

#valid\_writer.add\_summary(result1,i) #将日志数据写入文件

1. 将激活函数设置为softplus，依照实验one中的步骤，训练模型并查看收敛情况。

## 实验结果

### 激活函数为sigmoid

1. 运行截图

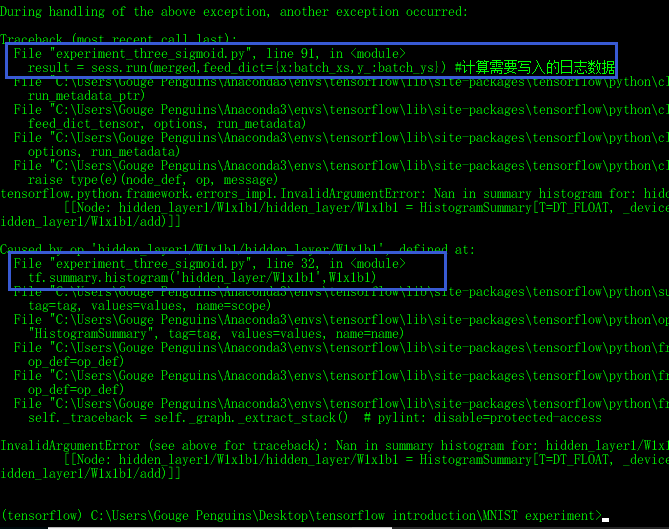


图 15



图 16



图 17

### 激活函数为tanh



图 18

准确率和损失函数都不收敛

### 激活函数为elu

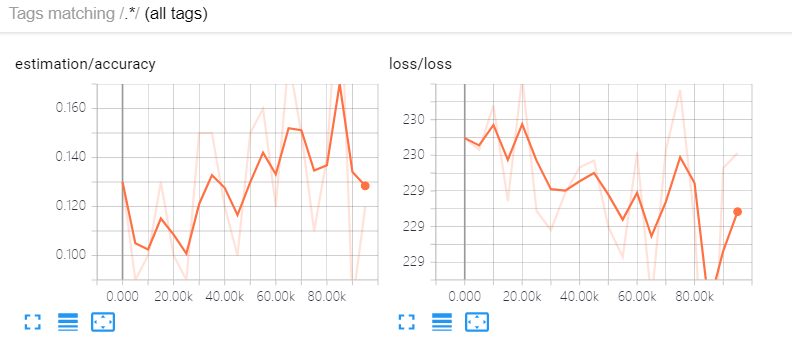


图 19

训练次数为100000，采样距离为5000，训练结果为图 16所示，从图中可见，精度和损失函数都有收敛的趋势，但是远未达到收敛的程度，应该是需要更多次的训练。

### 激活函数为relu6

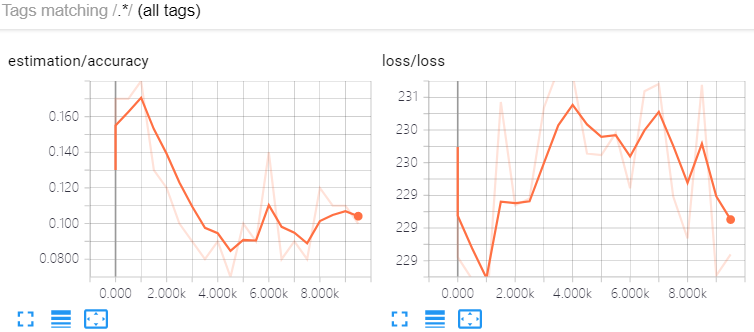


图 20

不收敛

### 激活函数为softplus



图 21

# 实验four

## 实验任务

在实验one的基础上，变更损失函数为：

loss = tf.reduce\_mean(tf.square(y\_ - y))

并使用tensorboard查看收敛情况

## 实验步骤

源码：

"""

本实验在experiment\_one.py的基础上，做了如下改动：

@变更了损失函数

"""

import input\_data

mnist = input\_data.read\_data\_sets("MNIST\_data/", one\_hot=True)

import tensorflow as tf

# soft回归模型

# 第一层

with tf.name\_scope('input\_layer') :

x = tf.placeholder(tf.float32,[None,784]) # 60000x784的矩阵

y\_ = tf.placeholder("float",[None,10]) # 60000x10 的矩阵

with tf.name\_scope('hidden\_layer1') :

with tf.name\_scope('weight'):

W1 = tf.Variable(tf.zeros([784,10])) # 784x10的矩阵

tf.summary.histogram('hidden\_layer1/weight',W1)

with tf.name\_scope('weight'):

b1 = tf.Variable(tf.zeros([10])) # 长度为10的行矩阵

tf.summary.histogram('hidden\_layer1/weight',b1)

with tf.name\_scope('W1x1b1'):

W1x1b1 = tf.matmul(x,W1)+b1 # matmul(x,W1) 为60000x10的矩阵，每行加上偏置行向量b1

tf.summary.histogram('hidden\_layer/W1x1b1',W1x1b1)

x2 = tf.nn.relu(W1x1b1) #激活函数 60000x10的矩阵

with tf.name\_scope('hidden\_layer2') :

with tf.name\_scope('weight'):

W2 = tf.Variable(tf.zeros([10,784])) # 10x784的矩阵

tf.summary.histogram('hidden\_layer2/weight',W2)

with tf.name\_scope('weight'):

b2 = tf.Variable(tf.zeros([784])) # 长度为784的行矩阵

tf.summary.histogram('hidden\_layer2/weight',b2)

with tf.name\_scope('W2x2b2'):

W2x2b2 = tf.matmul(x2,W2)+b2 # matmul(x,W1) 为60000x784的矩阵，每行加上偏置行向量b2

tf.summary.histogram('hidden\_layer2/W2x2b2',W2x2b2)

x3 = tf.nn.relu(W2x2b2) #激活函数 60000x784的矩阵

with tf.name\_scope('output\_layer') :

with tf.name\_scope('weight'):

W = tf.Variable(tf.zeros([784,10])) # 784x10的矩阵

tf.summary.histogram('output\_layer/weight',W)

with tf.name\_scope('weight'):

b = tf.Variable(tf.zeros([10])) # 长度为10的行矩阵

tf.summary.histogram('output\_layer/weight',b)

with tf.name\_scope('Wxb'):

Wxb = tf.matmul(x3,W)+b # matmul(x,W1) 为60000x10的矩阵，每行加上偏置行向量b

tf.summary.histogram('output\_layer/Wxb',Wxb)

y = tf.nn.softmax(Wxb) #经softmax后，变为归一化的 60000x10 的概率矩阵

# 训练模型

# 用交叉熵作为损失函数

with tf.name\_scope('loss'):

loss = tf.reduce\_mean(tf.square(y\_ - y))

tf.summary.scalar('loss',loss)

with tf.name\_scope('train'):

train\_step = tf.train.GradientDescentOptimizer(0.01).minimize(loss)

# 评估模型

with tf.name\_scope('estimation'):

correct\_prediction = tf.equal(tf.argmax(y,1),tf.argmax(y\_,1))

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction,"float"))

tf.summary.scalar('accuracy',accuracy)

init = tf.initialize\_all\_variables()

sess = tf.Session()

sess.run(init)

merged = tf.summary.merge\_all() #将图形、训练过程等数据合并在一起

train\_writer = tf.summary.FileWriter('exp\_four\_train\_logs',sess.graph) #将训练日志写入到logs文件夹下

# valid\_writer = tf.summary.FileWriter('valid\_logs',sess.graph) #将训练日志写入到logs文件夹下

for i in range(1000):

batch\_xs, batch\_ys = mnist.train.next\_batch(100)

sess.run(train\_step,feed\_dict = {x:batch\_xs, y\_ :batch\_ys})

if(i%100==0):

result = sess.run(merged,feed\_dict={x:batch\_xs,y\_:batch\_ys}) #计算需要写入的日志数据

#result1 = sess.run(accuracy,feed\_dict={x:batch\_xs,y\_:batch\_ys})

train\_writer.add\_summary(result,i) #将日志数据写入文件

#valid\_writer.add\_summary(result1,i) #将日志数据写入文件

## 实验结果

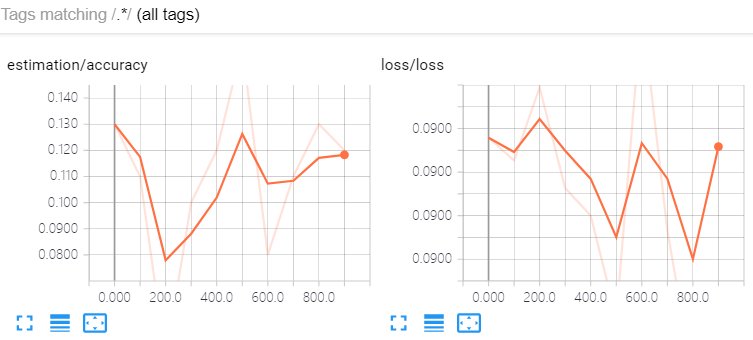


图 22

修改训练次数为10000，采样次数为100以后



图 23

修改训练次数为10000，采样次数为500以后



图 24

## 结论

损失函数选的不好，模型无论如何都糟糕

## 修改损失函数再实验1

变更损失函数为：

loss = tf.reduce\_mean(-tf.reduce\_sum(y \* tf.log(tf.clip\_by\_value(Wxb,1e-10,1.0)), reduction\_indices = [1]))

并使用tensorboard查看收敛情况



图 25

从图 25中可以看到，损失函数收敛了，而且比以前的收敛都要好，但是准确度不收敛。从而可以验证，之前的损失函数选的不好，所以不收敛，现在的损失函数收敛了，但是准确率不收敛，这个损失函数也不是很好。仔细观察发现，这个损失函数的比较对象是最后一个fully connect层激活前与激活后的特征值，与原始数据标签没有任何关系，难怪准确度不高。

## 修改损失函数再实验2

变更损失函数为：

loss = tf.reduce\_mean(-tf.reduce\_sum(y\_ \* tf.log(tf.clip\_by\_value(y,1e-10,1.0)), reduction\_indices = [1]))

并使用tensorboard查看收敛情况



图 26

变更损失函数为：

loss = tf.reduce\_mean(-tf.reduce\_sum(y \* tf.log(tf.clip\_by\_value(y\_,1e-10,1.0)), reduction\_indices = [1]))

并使用tensorboard查看收敛情况



图 27

# 实验five

## 实验任务

使用卷积神经网络重做实验zero，tensorboard可视化

## 实验步骤

## 实验结果

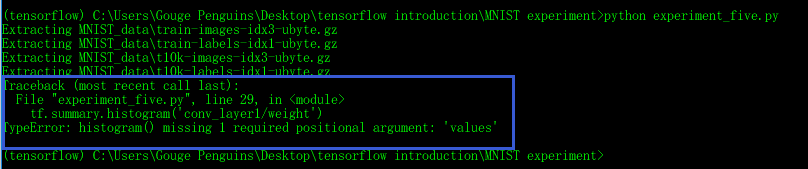


图 28

等了将近两个小时，给我来一个这么一个bug

训练一次0.5秒，10000次的训练=5000秒=1.3h

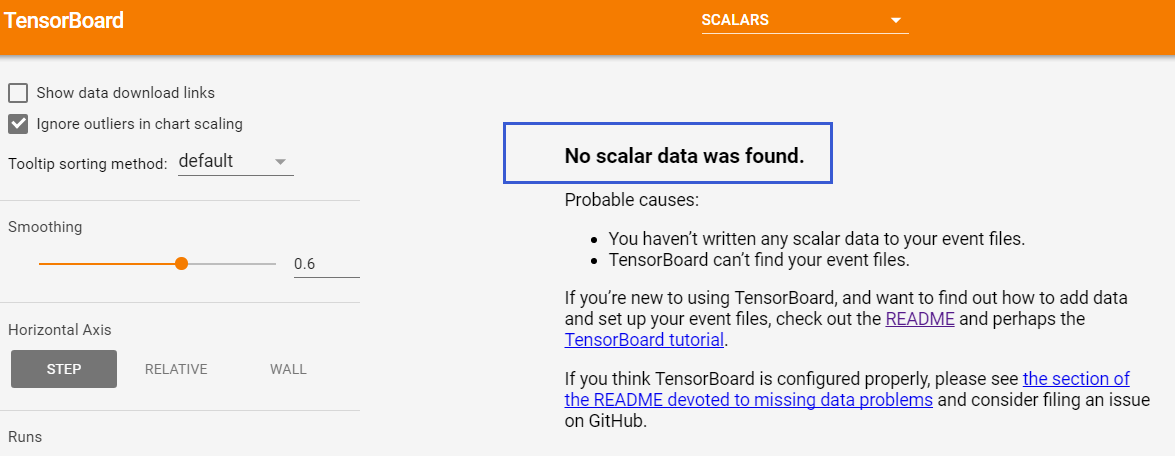


图 29

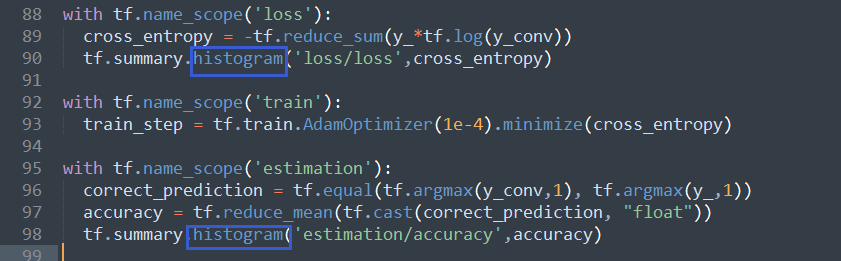


图 30

写错代码导致没有标量数据

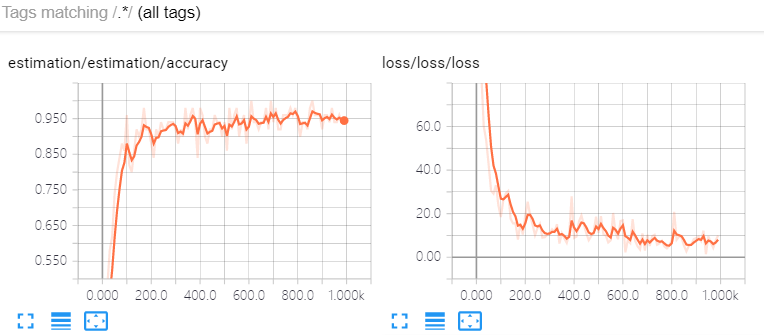


图 31

# 实验six

## 实验任务

使用卷积神经网络重做实验one。

在实验five的基础上，增加一层卷积池化

## 实验步骤

图像是28\*28像素的，原来的代码两层池化（pooling\_height=2,pooling\_width=2），结果为7\*7像素的，没法再池化了.所以，再卷积一次，不池化。源码：

import input\_data

mnist = input\_data.read\_data\_sets('MNIST\_data', one\_hot=True)

import tensorflow as tf

def weight\_variable(shape):

initial = tf.truncated\_normal(shape, stddev=0.1)

return tf.Variable(initial)

def bias\_variable(shape):

initial = tf.constant(0.1, shape=shape)

return tf.Variable(initial)

def conv2d(x, W):

return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def max\_pool\_2x2(x):

return tf.nn.max\_pool(x, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')

# 第一层:输入层

with tf.name\_scope('input\_layer') :

x = tf.placeholder("float", shape=[None, 784])

y\_ = tf.placeholder("float", shape=[None, 10])

keep\_prob = tf.placeholder("float")

# 第二层：卷积层+池化

with tf.name\_scope('conv\_layer1'):

with tf.name\_scope('weight'):

W\_conv1 = weight\_variable([5, 5, 1, 32]) # 5x5的卷积核，单通道，32个卷积核

tf.summary.histogram('conv\_layer1/weight',W\_conv1)

with tf.name\_scope('bias'):

b\_conv1 = bias\_variable([32]) #32个偏置值

#一张28x28的图像与上述卷积核卷积后的结果为32个28x28的矩阵

#每个矩阵一个偏置值，偏置值将会加到这个矩阵的每一个元素上面

tf.summary.histogram('conv\_layer1/bias',b\_conv1)

with tf.name\_scope('x\_image'):

x\_image = tf.reshape(x, [-1,28,28,1])

tf.summary.histogram('conv\_layer1/x\_image',x\_image)

with tf.name\_scope('h\_conv1'):

h\_conv1 = tf.nn.relu(conv2d(x\_image, W\_conv1) + b\_conv1)

tf.summary.histogram('conv\_layer1/h\_conv1',h\_conv1)

with tf.name\_scope('h\_pool1'):

h\_pool1 = max\_pool\_2x2(h\_conv1)

tf.summary.histogram('conv\_layer1/h\_pool1',h\_pool1)

# 第一层操作的结果 图片张数(batch) 个 28x28的矩阵，32通道

# 池化结果 图片张数(batch) 个 14x14的矩阵，32通道

# 第三层：卷积层+池化

with tf.name\_scope('conv\_layer2'):

with tf.name\_scope('weight'):

W\_conv2 = weight\_variable([5, 5, 32, 64]) # 5x5的卷积核，32通道，64个卷积核

tf.summary.histogram('conv\_layer2/weight',W\_conv2)

with tf.name\_scope('bias'):

b\_conv2 = bias\_variable([64]) # 32个偏置值

# 一张14x14 32通道的图像与上述卷积核卷积后的结果为64个14x14的矩阵

tf.summary.histogram('conv\_layer2/bias',b\_conv2)

with tf.name\_scope('h\_conv2'):

h\_conv2 = tf.nn.relu(conv2d(h\_pool1, W\_conv2) + b\_conv2)

tf.summary.histogram('conv\_layer2/h\_conv2',h\_conv2)

with tf.name\_scope('h\_pool2'):

h\_pool2 = max\_pool\_2x2(h\_conv2)

tf.summary.histogram('conv\_layer2/h\_pool2',h\_pool2)

# 第二层操作的结果 图片张数(batch) 个 14x14的矩阵，64通道

# 池化结果 图片张数(batch) 个 7x7的矩阵，64通道

# 第三层：卷积层+池化

with tf.name\_scope('conv\_layer3'):

with tf.name\_scope('weight'):

W\_conv3 = weight\_variable([5, 5, 64, 128])

tf.summary.histogram('conv\_layer2/weight',W\_conv3)

with tf.name\_scope('bias'):

b\_conv3= bias\_variable([128])

tf.summary.histogram('conv\_layer2/bias',b\_conv3)

with tf.name\_scope('h\_conv3'):

h\_conv3 = tf.nn.relu(conv2d(h\_pool2, W\_conv3) + b\_conv3)

tf.summary.histogram('conv\_layer3/h\_conv3',h\_conv3)

#with tf.name\_scope('h\_pool3'):

# h\_pool3 = max\_pool\_2x2(h\_conv3)

#tf.summary.histogram('conv\_layer3/h\_pool3',h\_pool3)

# 第三层操作的结果 图片张数(batch) 个 14x14的矩阵，128通道

# 池化结果：不做卷积 图片张数(batch) 个 7x7的矩阵，128通道

# 第四层：dropout层

with tf.name\_scope('dropout'):

with tf.name\_scope('weight'):

W\_fc1 = weight\_variable([7 \* 7 \* 128, 1024])

tf.summary.histogram('dropout/weight',W\_fc1)

with tf.name\_scope('bias'):

b\_fc1 = bias\_variable([1024])

tf.summary.histogram('dropout/bias',b\_fc1)

with tf.name\_scope('h\_pool2\_flat'):

h\_pool2\_flat = tf.reshape(h\_conv3, [-1, 7\*7\*128])

tf.summary.histogram('dropout/h\_pool2\_flat',h\_pool2\_flat)

with tf.name\_scope('h\_fc1'):

h\_fc1 = tf.nn.relu(tf.matmul(h\_pool2\_flat, W\_fc1) + b\_fc1)

tf.summary.histogram('dropout/h\_fc1',h\_fc1)

with tf.name\_scope('h\_fc1\_drop'):

h\_fc1\_drop = tf.nn.dropout(h\_fc1, keep\_prob)

tf.summary.histogram('dropout/h\_fc1\_drop',h\_fc1\_drop)

# 第五层：fully-connect层

with tf.name\_scope('fully-connect'):

with tf.name\_scope('weight'):

W\_fc2 = weight\_variable([1024, 10])

tf.summary.histogram('fully-connect/weight',W\_fc2)

with tf.name\_scope('bais'):

b\_fc2 = bias\_variable([10])

tf.summary.histogram('fully-connect/bias',b\_fc2)

with tf.name\_scope('y\_conv'):

y\_conv=tf.nn.softmax(tf.matmul(h\_fc1\_drop, W\_fc2) + b\_fc2)

tf.summary.histogram('fully-connect/y\_conv',y\_conv)

with tf.name\_scope('loss'):

cross\_entropy = -tf.reduce\_sum(y\_\*tf.log(tf.clip\_by\_value(y\_conv,1e-10,1.0)))

tf.summary.scalar('loss/loss',cross\_entropy)

with tf.name\_scope('train'):

train\_step = tf.train.AdamOptimizer(1e-4).minimize(cross\_entropy)

with tf.name\_scope('estimation'):

correct\_prediction = tf.equal(tf.argmax(y\_conv,1), tf.argmax(y\_,1))

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, "float"))

tf.summary.scalar('estimation/accuracy',accuracy)

sess = tf.InteractiveSession()

sess.run(tf.initialize\_all\_variables())

merged = tf.summary.merge\_all() #将图形、训练过程等数据合并在一起

train\_writer = tf.summary.FileWriter('exp\_six\_logs',sess.graph) #将训练日志写入到logs文件夹下

for i in range(1000):

batch = mnist.train.next\_batch(50)

train\_step.run(feed\_dict={x: batch[0], y\_: batch[1], keep\_prob: 0.5})

if i%10 == 0:

result = sess.run(merged,feed\_dict={x: batch[0], y\_: batch[1], keep\_prob: 1.0}) #计算需要写入的日志数据

train\_writer.add\_summary(result,i) #将日志数据写入文件

print(i)

## 实验结果

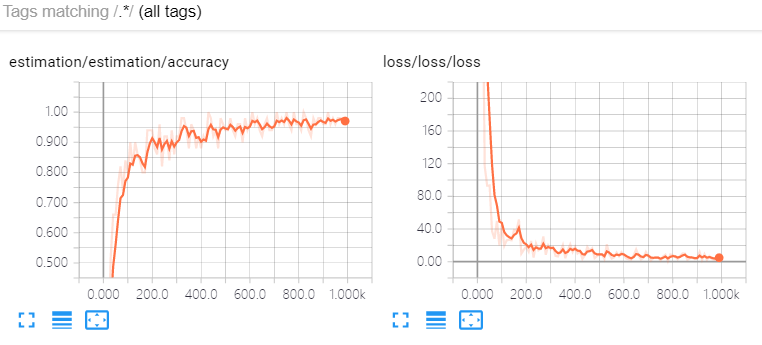


图 32

# 实验seven

## 实验任务

使用卷积神经网络重做实验two：模型的保存与导入

## 实验步骤

### 模型保存

import input\_data

mnist = input\_data.read\_data\_sets('MNIST\_data', one\_hot=True)

import tensorflow as tf

def weight\_variable(shape):

initial = tf.truncated\_normal(shape, stddev=0.1)

return tf.Variable(initial)

def bias\_variable(shape):

initial = tf.constant(0.1, shape=shape)

return tf.Variable(initial)

def conv2d(x, W):

return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def max\_pool\_2x2(x):

return tf.nn.max\_pool(x, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')

# 第一层:输入层

with tf.name\_scope('input\_layer') :

x = tf.placeholder("float", shape=[None, 784])

y\_ = tf.placeholder("float", shape=[None, 10])

keep\_prob = tf.placeholder("float")

# 第二层：卷积层+池化

with tf.name\_scope('conv\_layer1'):

with tf.name\_scope('weight'):

W\_conv1 = weight\_variable([5, 5, 1, 32]) # 5x5的卷积核，单通道，32个卷积核

tf.summary.histogram('conv\_layer1/weight',W\_conv1)

with tf.name\_scope('bias'):

b\_conv1 = bias\_variable([32]) #32个偏置值

#一张28x28的图像与上述卷积核卷积后的结果为32个28x28的矩阵

#每个矩阵一个偏置值，偏置值将会加到这个矩阵的每一个元素上面

tf.summary.histogram('conv\_layer1/bias',b\_conv1)

with tf.name\_scope('x\_image'):

x\_image = tf.reshape(x, [-1,28,28,1])

tf.summary.histogram('conv\_layer1/x\_image',x\_image)

with tf.name\_scope('h\_conv1'):

h\_conv1 = tf.nn.relu(conv2d(x\_image, W\_conv1) + b\_conv1)

tf.summary.histogram('conv\_layer1/h\_conv1',h\_conv1)

with tf.name\_scope('h\_pool1'):

h\_pool1 = max\_pool\_2x2(h\_conv1)

tf.summary.histogram('conv\_layer1/h\_pool1',h\_pool1)

# 第一层操作的结果 图片张数(batch) 个 28x28的矩阵，32通道

# 池化结果 图片张数(batch) 个 14x14的矩阵，32通道

# 第三层：卷积层+池化

with tf.name\_scope('conv\_layer2'):

with tf.name\_scope('weight'):

W\_conv2 = weight\_variable([5, 5, 32, 64]) # 5x5的卷积核，32通道，64个卷积核

tf.summary.histogram('conv\_layer2/weight',W\_conv2)

with tf.name\_scope('bias'):

b\_conv2 = bias\_variable([64]) # 32个偏置值

# 一张14x14 32通道的图像与上述卷积核卷积后的结果为64个14x14的矩阵

tf.summary.histogram('conv\_layer2/bias',b\_conv2)

with tf.name\_scope('h\_conv2'):

h\_conv2 = tf.nn.relu(conv2d(h\_pool1, W\_conv2) + b\_conv2)

tf.summary.histogram('conv\_layer2/h\_conv2',h\_conv2)

with tf.name\_scope('h\_pool2'):

h\_pool2 = max\_pool\_2x2(h\_conv2)

tf.summary.histogram('conv\_layer2/h\_pool2',h\_pool2)

# 第二层操作的结果 图片张数(batch) 个 14x14的矩阵，64通道

# 池化结果 图片张数(batch) 个 7x7的矩阵，64通道

# 第三层：卷积层+池化

with tf.name\_scope('conv\_layer3'):

with tf.name\_scope('weight'):

W\_conv3 = weight\_variable([5, 5, 64, 128])

tf.summary.histogram('conv\_layer2/weight',W\_conv3)

with tf.name\_scope('bias'):

b\_conv3= bias\_variable([128])

tf.summary.histogram('conv\_layer2/bias',b\_conv3)

with tf.name\_scope('h\_conv3'):

h\_conv3 = tf.nn.relu(conv2d(h\_pool2, W\_conv3) + b\_conv3)

tf.summary.histogram('conv\_layer3/h\_conv3',h\_conv3)

#with tf.name\_scope('h\_pool3'):

# h\_pool3 = max\_pool\_2x2(h\_conv3)

#tf.summary.histogram('conv\_layer3/h\_pool3',h\_pool3)

# 第三层操作的结果 图片张数(batch) 个 14x14的矩阵，128通道

# 池化结果：不做卷积 图片张数(batch) 个 7x7的矩阵，128通道

# 第四层：dropout层

with tf.name\_scope('dropout'):

with tf.name\_scope('weight'):

W\_fc1 = weight\_variable([7 \* 7 \* 128, 1024])

tf.summary.histogram('dropout/weight',W\_fc1)

with tf.name\_scope('bias'):

b\_fc1 = bias\_variable([1024])

tf.summary.histogram('dropout/bias',b\_fc1)

with tf.name\_scope('h\_pool2\_flat'):

h\_pool2\_flat = tf.reshape(h\_conv3, [-1, 7\*7\*128])

tf.summary.histogram('dropout/h\_pool2\_flat',h\_pool2\_flat)

with tf.name\_scope('h\_fc1'):

h\_fc1 = tf.nn.relu(tf.matmul(h\_pool2\_flat, W\_fc1) + b\_fc1)

tf.summary.histogram('dropout/h\_fc1',h\_fc1)

with tf.name\_scope('h\_fc1\_drop'):

h\_fc1\_drop = tf.nn.dropout(h\_fc1, keep\_prob)

tf.summary.histogram('dropout/h\_fc1\_drop',h\_fc1\_drop)

# 第五层：fully-connect层

with tf.name\_scope('fully-connect'):

with tf.name\_scope('weight'):

W\_fc2 = weight\_variable([1024, 10])

tf.summary.histogram('fully-connect/weight',W\_fc2)

with tf.name\_scope('bais'):

b\_fc2 = bias\_variable([10])

tf.summary.histogram('fully-connect/bias',b\_fc2)

with tf.name\_scope('y\_conv'):

y\_conv=tf.nn.softmax(tf.matmul(h\_fc1\_drop, W\_fc2) + b\_fc2)

tf.summary.histogram('fully-connect/y\_conv',y\_conv)

with tf.name\_scope('loss'):

cross\_entropy = -tf.reduce\_sum(y\_\*tf.log(tf.clip\_by\_value(y\_conv,1e-10,1.0)))

tf.summary.scalar('loss/loss',cross\_entropy)

with tf.name\_scope('train'):

train\_step = tf.train.AdamOptimizer(1e-4).minimize(cross\_entropy)

with tf.name\_scope('estimation'):

correct\_prediction = tf.equal(tf.argmax(y\_conv,1), tf.argmax(y\_,1))

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, "float"))

tf.summary.scalar('estimation/accuracy',accuracy)

sess = tf.InteractiveSession()

sess.run(tf.initialize\_all\_variables())

merged = tf.summary.merge\_all() #将图形、训练过程等数据合并在一起

train\_writer = tf.summary.FileWriter('exp\_seven\_logs',sess.graph) #将训练日志写入到logs文件夹下

for i in range(1000):

batch = mnist.train.next\_batch(50)

train\_step.run(feed\_dict={x: batch[0], y\_: batch[1], keep\_prob: 0.5})

if i%10 == 0:

result = sess.run(merged,feed\_dict={x: batch[0], y\_: batch[1], keep\_prob: 1.0}) #计算需要写入的日志数据

train\_writer.add\_summary(result,i) #将日志数据写入文件

print(i)

saver = tf.train.Saver() # 生成 saver

saver.save(sess,"exp\_seven\_model\_savepath/")

### 模型导入

import input\_data

mnist = input\_data.read\_data\_sets('MNIST\_data', one\_hot=True)

import tensorflow as tf

def weight\_variable(shape):

initial = tf.truncated\_normal(shape, stddev=0.1)

return tf.Variable(initial)

def bias\_variable(shape):

initial = tf.constant(0.1, shape=shape)

return tf.Variable(initial)

def conv2d(x, W):

return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def max\_pool\_2x2(x):

return tf.nn.max\_pool(x, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')

# 第一层:输入层

with tf.name\_scope('input\_layer') :

x = tf.placeholder("float", shape=[None, 784])

y\_ = tf.placeholder("float", shape=[None, 10])

keep\_prob = tf.placeholder("float")

"""

# 第二层：卷积层+池化

with tf.name\_scope('conv\_layer1'):

with tf.name\_scope('weight'):

W\_conv1 = weight\_variable([5, 5, 1, 32]) # 5x5的卷积核，单通道，32个卷积核

tf.summary.histogram('conv\_layer1/weight',W\_conv1)

with tf.name\_scope('bias'):

b\_conv1 = bias\_variable([32]) #32个偏置值

#一张28x28的图像与上述卷积核卷积后的结果为32个28x28的矩阵

#每个矩阵一个偏置值，偏置值将会加到这个矩阵的每一个元素上面

tf.summary.histogram('conv\_layer1/bias',b\_conv1)

with tf.name\_scope('x\_image'):

x\_image = tf.reshape(x, [-1,28,28,1])

tf.summary.histogram('conv\_layer1/x\_image',x\_image)

with tf.name\_scope('h\_conv1'):

h\_conv1 = tf.nn.relu(conv2d(x\_image, W\_conv1) + b\_conv1)

tf.summary.histogram('conv\_layer1/h\_conv1',h\_conv1)

with tf.name\_scope('h\_pool1'):

h\_pool1 = max\_pool\_2x2(h\_conv1)

tf.summary.histogram('conv\_layer1/h\_pool1',h\_pool1)

# 第一层操作的结果 图片张数(batch) 个 28x28的矩阵，32通道

# 池化结果 图片张数(batch) 个 14x14的矩阵，32通道

# 第三层：卷积层+池化

with tf.name\_scope('conv\_layer2'):

with tf.name\_scope('weight'):

W\_conv2 = weight\_variable([5, 5, 32, 64]) # 5x5的卷积核，32通道，64个卷积核

tf.summary.histogram('conv\_layer2/weight',W\_conv2)

with tf.name\_scope('bias'):

b\_conv2 = bias\_variable([64]) # 32个偏置值

# 一张14x14 32通道的图像与上述卷积核卷积后的结果为64个14x14的矩阵

tf.summary.histogram('conv\_layer2/bias',b\_conv2)

with tf.name\_scope('h\_conv2'):

h\_conv2 = tf.nn.relu(conv2d(h\_pool1, W\_conv2) + b\_conv2)

tf.summary.histogram('conv\_layer2/h\_conv2',h\_conv2)

with tf.name\_scope('h\_pool2'):

h\_pool2 = max\_pool\_2x2(h\_conv2)

tf.summary.histogram('conv\_layer2/h\_pool2',h\_pool2)

# 第二层操作的结果 图片张数(batch) 个 14x14的矩阵，64通道

# 池化结果 图片张数(batch) 个 7x7的矩阵，64通道

# 第三层：卷积层+池化

with tf.name\_scope('conv\_layer3'):

with tf.name\_scope('weight'):

W\_conv3 = weight\_variable([5, 5, 64, 128])

tf.summary.histogram('conv\_layer2/weight',W\_conv3)

with tf.name\_scope('bias'):

b\_conv3= bias\_variable([128])

tf.summary.histogram('conv\_layer2/bias',b\_conv3)

with tf.name\_scope('h\_conv3'):

h\_conv3 = tf.nn.relu(conv2d(h\_pool2, W\_conv3) + b\_conv3)

tf.summary.histogram('conv\_layer3/h\_conv3',h\_conv3)

#with tf.name\_scope('h\_pool3'):

# h\_pool3 = max\_pool\_2x2(h\_conv3)

#tf.summary.histogram('conv\_layer3/h\_pool3',h\_pool3)

# 第三层操作的结果 图片张数(batch) 个 14x14的矩阵，128通道

# 池化结果：不做卷积 图片张数(batch) 个 7x7的矩阵，128通道

# 第四层：dropout层

with tf.name\_scope('dropout'):

with tf.name\_scope('weight'):

W\_fc1 = weight\_variable([7 \* 7 \* 128, 1024])

tf.summary.histogram('dropout/weight',W\_fc1)

with tf.name\_scope('bias'):

b\_fc1 = bias\_variable([1024])

tf.summary.histogram('dropout/bias',b\_fc1)

with tf.name\_scope('h\_pool2\_flat'):

h\_pool2\_flat = tf.reshape(h\_conv3, [-1, 7\*7\*128])

tf.summary.histogram('dropout/h\_pool2\_flat',h\_pool2\_flat)

with tf.name\_scope('h\_fc1'):

h\_fc1 = tf.nn.relu(tf.matmul(h\_pool2\_flat, W\_fc1) + b\_fc1)

tf.summary.histogram('dropout/h\_fc1',h\_fc1)

with tf.name\_scope('h\_fc1\_drop'):

h\_fc1\_drop = tf.nn.dropout(h\_fc1, keep\_prob)

tf.summary.histogram('dropout/h\_fc1\_drop',h\_fc1\_drop)

# 第五层：fully-connect层

with tf.name\_scope('fully-connect'):

with tf.name\_scope('weight'):

W\_fc2 = weight\_variable([1024, 10])

tf.summary.histogram('fully-connect/weight',W\_fc2)

with tf.name\_scope('bais'):

b\_fc2 = bias\_variable([10])

tf.summary.histogram('fully-connect/bias',b\_fc2)

with tf.name\_scope('y\_conv'):

y\_conv=tf.nn.softmax(tf.matmul(h\_fc1\_drop, W\_fc2) + b\_fc2)

tf.summary.histogram('fully-connect/y\_conv',y\_conv)

with tf.name\_scope('loss'):

cross\_entropy = -tf.reduce\_sum(y\_\*tf.log(tf.clip\_by\_value(y\_conv,1e-10,1.0)))

tf.summary.scalar('loss/loss',cross\_entropy)

with tf.name\_scope('train'):

train\_step = tf.train.AdamOptimizer(1e-4).minimize(cross\_entropy)

with tf.name\_scope('estimation'):

correct\_prediction = tf.equal(tf.argmax(y\_conv,1), tf.argmax(y\_,1))

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, "float"))

tf.summary.scalar('estimation/accuracy',accuracy)

"""

sess = tf.InteractiveSession()

saver = tf.train.Saver() # 生成 saver

saver.restore(sess,"exp\_seven\_model\_savepath/")

print("start")

batch = mnist.train.next\_batch(50)

print (sess.run(accuracy,feed\_dict = {x:batch[0],y\_:batch[1],keep\_prob: 0.5}))

#print (sess.run(accuracy,feed\_dict = {x:mnist.test.images,y\_:mnist.test.labels,keep\_prob: 0.5}))

print("end")

# sess.run(tf.initialize\_all\_variables())

"""

merged = tf.summary.merge\_all() #将图形、训练过程等数据合并在一起

train\_writer = tf.summary.FileWriter('exp\_seven\_check\_logs',sess.graph) #将训练日志写入到logs文件夹下

for i in range(1000):

batch = mnist.train.next\_batch(50)

train\_step.run(feed\_dict={x: batch[0], y\_: batch[1], keep\_prob: 0.5})

if i%10 == 0:

result = sess.run(merged,feed\_dict={x: batch[0], y\_: batch[1], keep\_prob: 1.0}) #计算需要写入的日志数据

train\_writer.add\_summary(result,i) #将日志数据写入文件

print(i)

"""

### 模型导入方式2

import input\_data

mnist = input\_data.read\_data\_sets('MNIST\_data', one\_hot=True)

import tensorflow as tf

# 第一层:输入层

with tf.name\_scope('input\_layer') :

x = tf.placeholder("float", shape=[None, 784])

y\_ = tf.placeholder("float", shape=[None, 10])

keep\_prob = tf.placeholder("float")

sess = tf.InteractiveSession()

saver = tf.train.Saver() # 生成 saver

saver.restore(sess,"exp\_seven\_model\_savepath/")

print("start")

batch = mnist.train.next\_batch(50)

print (sess.run(accuracy,feed\_dict = {x:batch[0],y\_:batch[1],keep\_prob: 0.5}))

print("end")

结果：

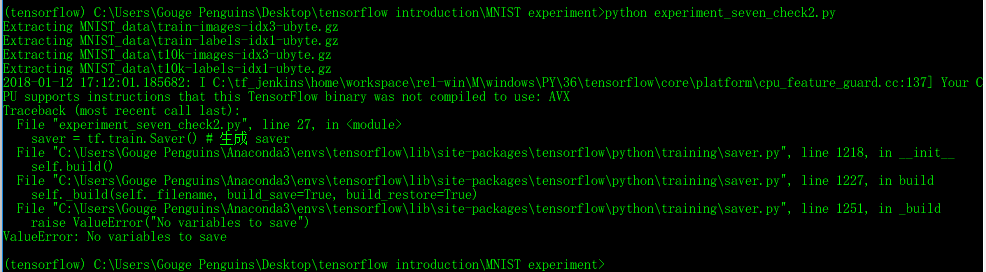


图 33

## 实验结果

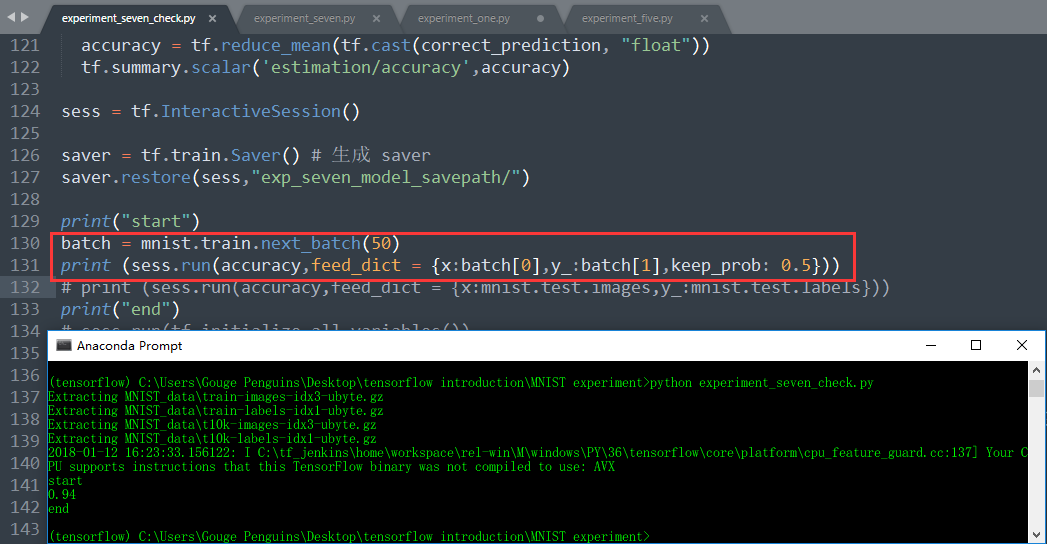


图 34

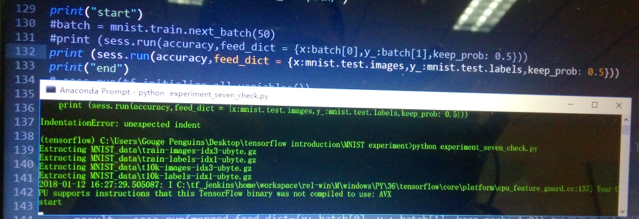


图 35

用图 33中所示的方式可以导入模型，直接运行并得到准确度，而图 34所示的方式却输出start后运行sess.run就奔溃，整个桌面都卡住，只能重新启动计算机。

## 另一种模型保存与导入

连计算图的结构信息都保存下来，调用时不用重复定义计算图结构，教程：

<http://blog.csdn.net/laolu1573/article/details/66971800>

<http://blog.csdn.net/lwplwf/article/details/62419087>

# 实验eight

## 实验任务

使用卷积神经网络重做实验three，更换激活函数

## 实验步骤

源码：

import input\_data

mnist = input\_data.read\_data\_sets('MNIST\_data', one\_hot=True)

import tensorflow as tf

def weight\_variable(shape):

initial = tf.truncated\_normal(shape, stddev=0.1)

return tf.Variable(initial)

def bias\_variable(shape):

initial = tf.constant(0.1, shape=shape)

return tf.Variable(initial)

def conv2d(x, W):

return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def max\_pool\_2x2(x):

return tf.nn.max\_pool(x, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')

# 第一层:输入层

with tf.name\_scope('input\_layer') :

x = tf.placeholder("float", shape=[None, 784])

y\_ = tf.placeholder("float", shape=[None, 10])

keep\_prob = tf.placeholder("float")

# 第二层：卷积层+池化

with tf.name\_scope('conv\_layer1'):

with tf.name\_scope('weight'):

W\_conv1 = weight\_variable([5, 5, 1, 32])

tf.summary.histogram('conv\_layer1/weight',W\_conv1)

with tf.name\_scope('bias'):

b\_conv1 = bias\_variable([32])

tf.summary.histogram('conv\_layer1/bias',b\_conv1)

with tf.name\_scope('x\_image'):

x\_image = tf.reshape(x, [-1,28,28,1])

tf.summary.histogram('conv\_layer1/x\_image',x\_image)

with tf.name\_scope('h\_conv1'):

h\_conv1 = tf.nn.elu(conv2d(x\_image, W\_conv1) + b\_conv1)

tf.summary.histogram('conv\_layer1/h\_conv1',h\_conv1)

with tf.name\_scope('h\_pool1'):

h\_pool1 = max\_pool\_2x2(h\_conv1)

tf.summary.histogram('conv\_layer1/h\_pool1',h\_pool1)

# 第三层：卷积层+池化

with tf.name\_scope('conv\_layer2'):

with tf.name\_scope('weight'):

W\_conv2 = weight\_variable([5, 5, 32, 64])

tf.summary.histogram('conv\_layer2/weight',W\_conv2)

with tf.name\_scope('bias'):

b\_conv2 = bias\_variable([64])

tf.summary.histogram('conv\_layer2/bias',b\_conv2)

with tf.name\_scope('h\_conv2'):

h\_conv2 = tf.nn.elu(conv2d(h\_pool1, W\_conv2) + b\_conv2)

tf.summary.histogram('conv\_layer2/h\_conv2',h\_conv2)

with tf.name\_scope('h\_pool2'):

h\_pool2 = max\_pool\_2x2(h\_conv2)

tf.summary.histogram('conv\_layer2/h\_pool2',h\_pool2)

# 第四层：dropout层

with tf.name\_scope('dropout'):

with tf.name\_scope('weight'):

W\_fc1 = weight\_variable([7 \* 7 \* 64, 1024])

tf.summary.histogram('dropout/weight',W\_fc1)

with tf.name\_scope('bias'):

b\_fc1 = bias\_variable([1024])

tf.summary.histogram('dropout/bias',b\_fc1)

with tf.name\_scope('h\_pool2\_flat'):

h\_pool2\_flat = tf.reshape(h\_pool2, [-1, 7\*7\*64])

tf.summary.histogram('dropout/h\_pool2\_flat',h\_pool2\_flat)

with tf.name\_scope('h\_fc1'):

h\_fc1 = tf.nn.elu(tf.matmul(h\_pool2\_flat, W\_fc1) + b\_fc1)

tf.summary.histogram('dropout/h\_fc1',h\_fc1)

with tf.name\_scope('h\_fc1\_drop'):

h\_fc1\_drop = tf.nn.dropout(h\_fc1, keep\_prob)

tf.summary.histogram('dropout/h\_fc1\_drop',h\_fc1\_drop)

# 第五层：dropout层

with tf.name\_scope('fully-connect'):

with tf.name\_scope('weight'):

W\_fc2 = weight\_variable([1024, 10])

tf.summary.histogram('fully-connect/weight',W\_fc2)

with tf.name\_scope('bais'):

b\_fc2 = bias\_variable([10])

tf.summary.histogram('fully-connect/bias',b\_fc2)

with tf.name\_scope('y\_conv'):

y\_conv=tf.nn.softmax(tf.matmul(h\_fc1\_drop, W\_fc2) + b\_fc2)

tf.summary.histogram('fully-connect/y\_conv',y\_conv)

with tf.name\_scope('loss'):

cross\_entropy = -tf.reduce\_sum(y\_\*tf.log(y\_conv))

tf.summary.scalar('loss/loss',cross\_entropy)

with tf.name\_scope('train'):

train\_step = tf.train.AdamOptimizer(1e-4).minimize(cross\_entropy)

with tf.name\_scope('estimation'):

correct\_prediction = tf.equal(tf.argmax(y\_conv,1), tf.argmax(y\_,1))

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, "float"))

tf.summary.scalar('estimation/accuracy',accuracy)

sess = tf.InteractiveSession()

sess.run(tf.initialize\_all\_variables())

merged = tf.summary.merge\_all() #将图形、训练过程等数据合并在一起

train\_writer = tf.summary.FileWriter('exp\_eight\_logs',sess.graph) #将训练日志写入到logs文件夹下

for i in range(1000):

batch = mnist.train.next\_batch(50)

train\_step.run(feed\_dict={x: batch[0], y\_: batch[1], keep\_prob: 0.5})

if i%10 == 0:

result = sess.run(merged,feed\_dict={x: batch[0], y\_: batch[1], keep\_prob: 1.0}) #计算需要写入的日志数据

train\_writer.add\_summary(result,i) #将日志数据写入文件

print(i)

## 实验结果

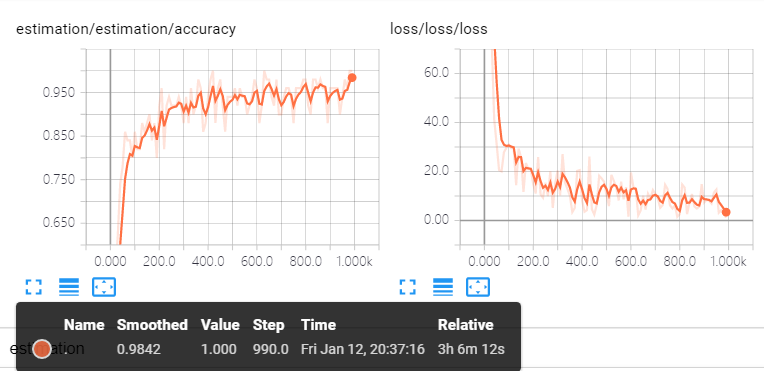


图 36 elu

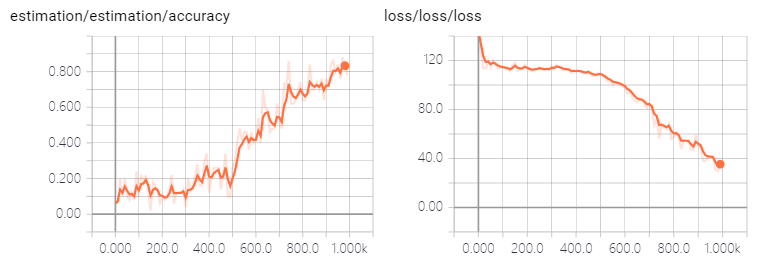


图 37 sigmoid 训练次数1000，还没有收敛

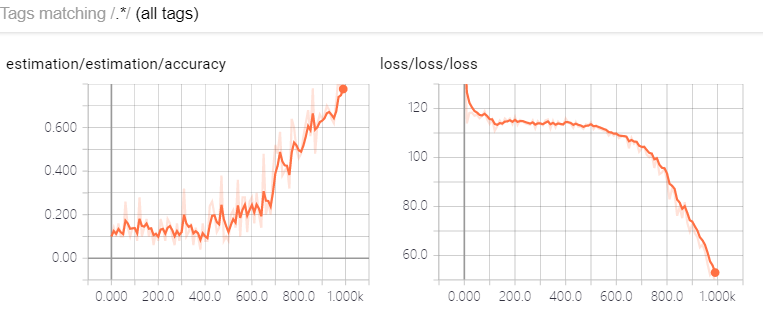


图 38 softplus 训练次数1000，还没有收敛

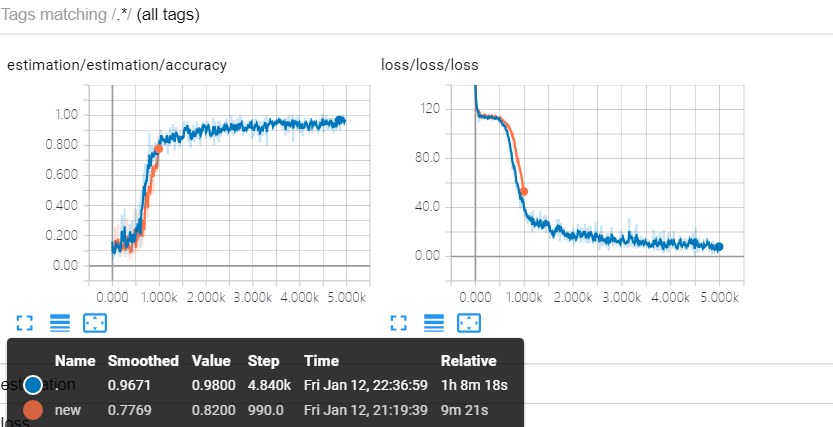


图 39 训练5000次以后就收敛了（蓝色线）

# 实验nine

## 实验任务

使用卷积神经网络重做实验four，更换损失函数为均方误差

## 实验步骤

源代码：

import input\_data

mnist = input\_data.read\_data\_sets('MNIST\_data', one\_hot=True)

import tensorflow as tf

def weight\_variable(shape):

initial = tf.truncated\_normal(shape, stddev=0.1)

return tf.Variable(initial)

def bias\_variable(shape):

initial = tf.constant(0.1, shape=shape)

return tf.Variable(initial)

def conv2d(x, W):

return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def max\_pool\_2x2(x):

return tf.nn.max\_pool(x, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')

# 第一层:输入层

with tf.name\_scope('input\_layer') :

x = tf.placeholder("float", shape=[None, 784])

y\_ = tf.placeholder("float", shape=[None, 10])

keep\_prob = tf.placeholder("float")

# 第二层：卷积层+池化

with tf.name\_scope('conv\_layer1'):

with tf.name\_scope('weight'):

W\_conv1 = weight\_variable([5, 5, 1, 32])

tf.summary.histogram('conv\_layer1/weight',W\_conv1)

with tf.name\_scope('bias'):

b\_conv1 = bias\_variable([32])

tf.summary.histogram('conv\_layer1/bias',b\_conv1)

with tf.name\_scope('x\_image'):

x\_image = tf.reshape(x, [-1,28,28,1])

tf.summary.histogram('conv\_layer1/x\_image',x\_image)

with tf.name\_scope('h\_conv1'):

h\_conv1 = tf.nn.relu(conv2d(x\_image, W\_conv1) + b\_conv1)

tf.summary.histogram('conv\_layer1/h\_conv1',h\_conv1)

with tf.name\_scope('h\_pool1'):

h\_pool1 = max\_pool\_2x2(h\_conv1)

tf.summary.histogram('conv\_layer1/h\_pool1',h\_pool1)

# 第三层：卷积层+池化

with tf.name\_scope('conv\_layer2'):

with tf.name\_scope('weight'):

W\_conv2 = weight\_variable([5, 5, 32, 64])

tf.summary.histogram('conv\_layer2/weight',W\_conv2)

with tf.name\_scope('bias'):

b\_conv2 = bias\_variable([64])

tf.summary.histogram('conv\_layer2/bias',b\_conv2)

with tf.name\_scope('h\_conv2'):

h\_conv2 = tf.nn.relu(conv2d(h\_pool1, W\_conv2) + b\_conv2)

tf.summary.histogram('conv\_layer2/h\_conv2',h\_conv2)

with tf.name\_scope('h\_pool2'):

h\_pool2 = max\_pool\_2x2(h\_conv2)

tf.summary.histogram('conv\_layer2/h\_pool2',h\_pool2)

# 第四层：dropout层

with tf.name\_scope('dropout'):

with tf.name\_scope('weight'):

W\_fc1 = weight\_variable([7 \* 7 \* 64, 1024])

tf.summary.histogram('dropout/weight',W\_fc1)

with tf.name\_scope('bias'):

b\_fc1 = bias\_variable([1024])

tf.summary.histogram('dropout/bias',b\_fc1)

with tf.name\_scope('h\_pool2\_flat'):

h\_pool2\_flat = tf.reshape(h\_pool2, [-1, 7\*7\*64])

tf.summary.histogram('dropout/h\_pool2\_flat',h\_pool2\_flat)

with tf.name\_scope('h\_fc1'):

h\_fc1 = tf.nn.relu(tf.matmul(h\_pool2\_flat, W\_fc1) + b\_fc1)

tf.summary.histogram('dropout/h\_fc1',h\_fc1)

with tf.name\_scope('h\_fc1\_drop'):

h\_fc1\_drop = tf.nn.dropout(h\_fc1, keep\_prob)

tf.summary.histogram('dropout/h\_fc1\_drop',h\_fc1\_drop)

# 第五层：dropout层

with tf.name\_scope('fully-connect'):

with tf.name\_scope('weight'):

W\_fc2 = weight\_variable([1024, 10])

tf.summary.histogram('fully-connect/weight',W\_fc2)

with tf.name\_scope('bais'):

b\_fc2 = bias\_variable([10])

tf.summary.histogram('fully-connect/bias',b\_fc2)

with tf.name\_scope('y\_conv'):

y\_conv=tf.nn.softmax(tf.matmul(h\_fc1\_drop, W\_fc2) + b\_fc2)

tf.summary.histogram('fully-connect/y\_conv',y\_conv)

with tf.name\_scope('loss'):

cross\_entropy = tf.reduce\_mean(tf.square(y\_ - y\_conv))

tf.summary.scalar('loss/loss',cross\_entropy)

with tf.name\_scope('train'):

train\_step = tf.train.AdamOptimizer(1e-4).minimize(cross\_entropy)

with tf.name\_scope('estimation'):

correct\_prediction = tf.equal(tf.argmax(y\_conv,1), tf.argmax(y\_,1))

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, "float"))

tf.summary.scalar('estimation/accuracy',accuracy)

sess = tf.InteractiveSession()

sess.run(tf.initialize\_all\_variables())

merged = tf.summary.merge\_all() #将图形、训练过程等数据合并在一起

train\_writer = tf.summary.FileWriter('exp\_nine\_logs',sess.graph) #将训练日志写入到logs文件夹下

for i in range(1000):

batch = mnist.train.next\_batch(50)

train\_step.run(feed\_dict={x: batch[0], y\_: batch[1], keep\_prob: 0.5})

if i%10 == 0:

result = sess.run(merged,feed\_dict={x: batch[0], y\_: batch[1], keep\_prob: 1.0}) #计算需要写入的日志数据

train\_writer.add\_summary(result,i) #将日志数据写入文件

print(i)

## 实验结果



图 40

## 修改损失函数再实验1

变更损失函数为：

loss = tf.reduce\_mean(-tf.reduce\_sum(y \* tf.log(tf.clip\_by\_value(Wxb,1e-10,1.0)), reduction\_indices = [1]))

并使用tensorboard查看收敛情况



图 41

# 实验ten

## 实验任务

使用卷积神经网络做人脸检测

## 实验步骤

## 实验结果

# 实验eleven

## 实验任务

使用卷积神经网络做人脸识别

## 实验步骤

## 实验结果