华南理工大学

《深度学习与神经网络》课程实验报告

实验题目：　　第一次作业　　

姓名：　何宇航　　学号：201830170110

班级：18计科(2)班　　组别：

<table>
<tr><td>实验概述</td></tr>
<tr><td>【实验目的及要求】
一、　Numpy 基本操作（提交所有代码截图及运行结果）
二、　Tensorflow 练习（提交每个练习的实现步骤描述以及下面要求提交的结果）
【实验环境】
操作系统：Windows XP</td></tr>
<tr><td>实验内容</td></tr>
<tr><td>小结</td></tr>
<tr><td>本次实验利用了 tesorflow 与 numpy 的 python 库成功地实现了实验要求。Nump 的相关操作已经在大二的选修课学过了，所以这次的 numpy 练习顺利完成，在运用 tensorflow 构建神经网络的时候遇到了一些阻碍，根据老师上课的知识，我知道构建一个网络的步骤，但是老师并没有教授 tensorflow 库的使用方法，所以我通过阅读开发文档，结合老师教的相关知识，一步步地用 tensorflow 实现了神经网络的分类和回归任务，最后在 MNIST 数据集上的训练效果大概是 0.92 准确率。</td></tr>
<tr><td>指导教师评语及成绩</td></tr>
<tr><td>评语：
成绩：　　指导教师签名：
批阅日期：</td></tr>
</table>

隐藏代码

In [ ]:

```
#@title
%tensorflow_version 1.x
import tensorflow as tf
hello = tf.constant('Hello, Tensorflow')
sess = tf.Session()
print(sess.run(hello))
```

```
b'Hello, Tensorflow'
```

1、导入 numpy 库
2、建立一个一维数组 a，初始化为[4,5,6]
(1) 输出 a 的类型（type）
(2) 输出 a 的各维度大小（shape）
(3) 输出 a 的第一个元素\

In [ ]:

```
#@title
import numpy as np
a=np.array([4,5,6])
print(type(a))
print(a.shape)
print(a[0])
```

```
<class 'numpy.ndarray'>
(3,)
4
```

3、建立一个二维数组 b，初始化为[[4,5,6],[1,2,3]]
(1) 输出 b 的各维度大小（shape）
(2) 输出b[0,0],b[0,1],b[1,1]这三个元素

In [ ]:

```
b=np.array([[4,5,6],[1,2,3]])
print(b.shape)
print(b[0,0],b[0,1],b[1,1])
```

```
(2, 3)
4 5 2
```

4、建立矩阵
(1) 建立一个大小为3 × 3的全 0 矩阵 c
(2) 建立一个大小为4 × 5的全 1 矩阵 d
(3) 建立一个大小为4 × 4的单位矩阵 e

In [ ]:

```
c=np.zeros((3,3))
d=np.ones((4,5))
e=np.eye(4)
print(c);print(d);print(e)
```

```
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
[[1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]]
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
```

5、建立一个数组 f，初始化为[0,1,2,3,4,5,6,7,8,9,10,11]（arange）
(1) 输出 f 以及 f 的各维度大小

In [ ]:

```
f=np.arange(12)
f.shape
```

Out[ ]:

```
(12,)
```

(2) 将 f 的 shape 改为3 × 4（reshape）

In [ ]:

```
f=f.reshape(3,4)
```

(3) 输出 f 以及 f 的各维度大小

In [ ]:

```
print(f)
f.shape
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

Out[ ]:

```
(3, 4)
```

(4) 输出 f 第二行（f[1,:]）

In [ ]:

```
f[1,:]
```

Out[ ]:

```
array([4, 5, 6, 7])
```

(5) 输出 f 最后两列（f[:,2:]）

In [ ]:

```
f[:,2:]
```

Out[ ]:

```
array([[ 2,  3],
       [ 6,  7],
       [10, 11]])
```

(6) 输出 f 第三行最后一个元素（使用-1 表示最后一个元素）

In [ ]:

```
f[2,-1]
```

Out[ ]:

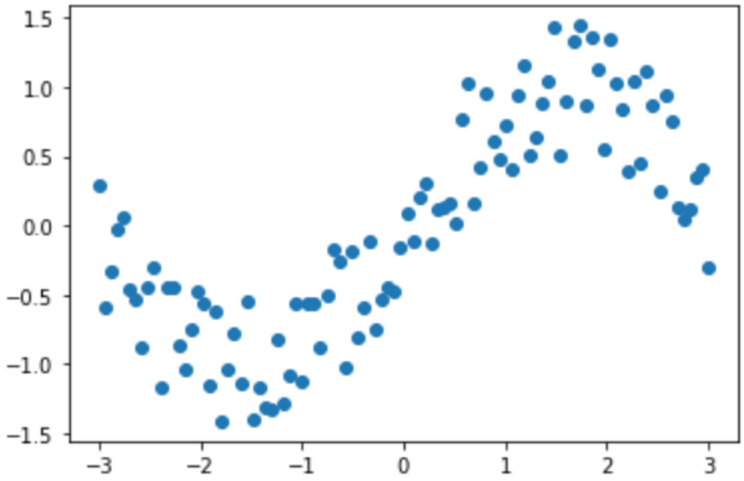
```
11
```

二、Tensorflow 练习（提交每个练习的实现步骤描述以及下面要求提交的结果）

1、线性回归
(1) 生成训练数据

In [ ]:

```
#@title
num_observations=100
x=np.linspace(-3,3,num_observations)
y=np.sin(x)+np.random.uniform(-0.5,0.5,num_observations)
import matplotlib.pyplot as plt
plt.scatter(x,y)
plt.show()
```



(2)使用 tensorflow 实现线性回归模型，训练参数w和b。

In [ ]:

```
#@title
n=len(x)

X = tf.placeholder("float")
Y = tf.placeholder("float")

W = tf.Variable(np.random.randn(), name = "W")
b = tf.Variable(np.random.randn(), name = "b")

learning_rate = 0.01
training_epochs = 1000

#@title
# 初始y_pred X*W
y_pred = tf.add(X*W, b)


# Loss函数
cost = tf.reduce_sum(tf.pow(y_pred-Y, 2)) / (2 * n)

# 梯度下降
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

# 初始化
init = tf.global_variables_initializer()
print('(3) 输出参数w、b和损失。(提交运行结果) ')
# 训练
with tf.Session() as sess:

    # 初始化
    sess.run(init)

    # epoch训练
    for epoch in range(training_epochs):

        # 随机梯度下降，一个一个feed
        for (_x, _y) in zip(x, y):
            sess.run(optimizer, feed_dict = {X : _x, Y : _y})

        # 显示w b
        if (epoch + 1) % 50 == 0:
            c = sess.run(cost, feed_dict = {X : x, Y : y})
            print("Epoch", (epoch + 1), ": cost =", c, "W =", sess.run(W), "b =", sess.
run(b))

    training_cost = sess.run(cost, feed_dict ={X: x, Y: y})
    weight = sess.run(W)
    bias = sess.run(b)
```

```
WARNING:tensorflow:From /tensorflow-1.15.2/python3.6/tensorflow_core/pytho
n/ops/math_grad.py:1375: where (from tensorflow.python.ops.array_ops) is d
eprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
(3) 输出参数w、b和损失。(提交运行结果)
Epoch 50 : cost = 0.5143953 W = 0.5935373 b = -0.7645738
Epoch 100 : cost = 0.23997615 W = 0.38526684 b = -0.47551134
Epoch 150 : cost = 0.17217466 W = 0.33997333 b = -0.30036998
Epoch 200 : cost = 0.14876172 W = 0.33003342 b = -0.19418705
Epoch 250 : cost = 0.14019178 W = 0.32779777 b = -0.12979871
Epoch 300 : cost = 0.13702145 W = 0.32726234 b = -0.0907505
Epoch 350 : cost = 0.13584204 W = 0.32711577 b = -0.06706932
Epoch 400 : cost = 0.13540003 W = 0.3270646 b = -0.052707434
Epoch 450 : cost = 0.13523243 W = 0.32704246 b = -0.043997485
Epoch 500 : cost = 0.13516775 W = 0.3270312 b = -0.038715076
Epoch 550 : cost = 0.13514212 W = 0.32702398 b = -0.035511367
Epoch 600 : cost = 0.13513155 W = 0.3270197 b = -0.033568557
Epoch 650 : cost = 0.135127 W = 0.32701772 b = -0.032390304
Epoch 700 : cost = 0.1351249 W = 0.32701614 b = -0.03167566
Epoch 750 : cost = 0.1351239 W = 0.32701507 b = -0.031242307
Epoch 800 : cost = 0.13512339 W = 0.32701486 b = -0.0309795
Epoch 850 : cost = 0.13512309 W = 0.32701334 b = -0.030820057
Epoch 900 : cost = 0.13512293 W = 0.32701632 b = -0.030723426
Epoch 950 : cost = 0.13512284 W = 0.32701364 b = -0.030664798
Epoch 1000 : cost = 0.13512278 W = 0.32701126 b = -0.030629147
```

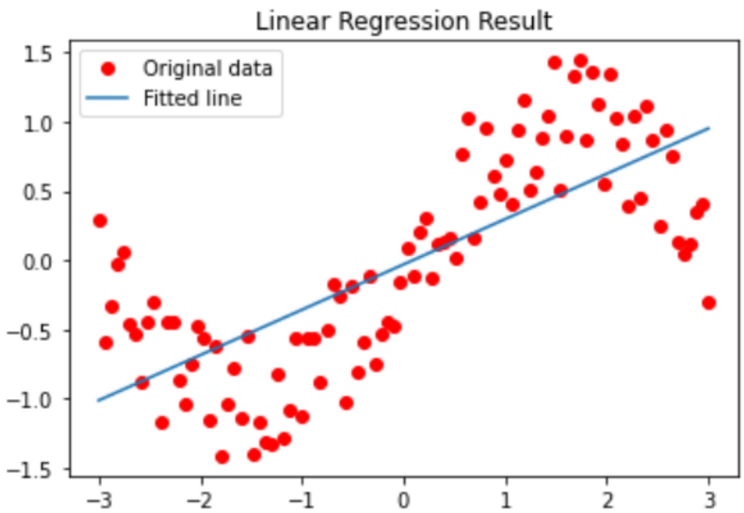
(3) 输出参数w、b和损失。(提交运行结果)
(4) 画出预测回归曲线以及训练数据散点图，对比回归曲线和散点图并分析原因。 (提交图片及分析)

In [ ]:

```
#@title
# 预测值
predictions = weight * x + bias
print("Training cost =", training_cost, "Weight =", weight, "bias =", bias, '\n')

# 画图
plt.plot(x, y, 'ro', label ='Original data')
plt.plot(x, predictions, label ='Fitted line')
plt.title('Linear Regression Result')
plt.legend()
plt.show()
```

```
Training cost = 0.13512278 Weight = 0.32701126 bias = -0.030629147
```



答： 从回归曲线和散点图来看，拟合效果不好，其原因在于散点数据本身不是由一个线性函数得到的，散点数据符合非线性关系

2、线性回归（使用多项式函数对原始数据进行变换）
(1) 生成训练数据，数据同上
(2) 使用 tensorflow 实现线性回归模型，这里我们假设y是x的 3 次多项式

In [ ]:

```
#@title
num_observations=100
x=np.linspace(-3,3,num_observations)
y=np.sin(x)+np.random.uniform(-0.5,0.5,num_observations)

n=len(x)

X = tf.placeholder("float")
Y = tf.placeholder("float")

W1 = tf.Variable(np.random.randn(), name = "W1")
W2 = tf.Variable(np.random.randn(), name = "W2")
W3 = tf.Variable(np.random.randn(), name = "W3")
b = tf.Variable(np.random.randn(), name = "b")

learning_rate = 0.01
training_epochs = 1000

# 初始y_pred
y_pred = tf.add(W1*X+W2*X**2+W3*X*X**2, b)


# Loss函数
cost = tf.reduce_sum(tf.pow(y_pred-Y, 2)) / (2 * n)

# 梯度下降
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

# 初始化
init = tf.global_variables_initializer()
print('(3) 输出参数w、b和损失。 (提交运行结果) ')
# 训练
with tf.Session() as sess:

    # 初始化
    sess.run(init)

    # epoch训练
    for epoch in range(training_epochs):

        # 随机梯度下降，一个一个feed
        for (_x, _y) in zip(x, y):
            sess.run(optimizer, feed_dict = {X : _x, Y : _y})

        # 显示w b
        if (epoch + 1) % 50 == 0:
            c = sess.run(cost, feed_dict = {X : x, Y : y})
            print("Epoch", (epoch + 1), ": cost =", c, "W =", sess.run([W1,W2,W3]), "b
 =", sess.run(b))

    training_cost = sess.run(cost, feed_dict ={X: x, Y: y})
    weight = sess.run([W1,W2,W3])
    bias = sess.run(b)
```

（3）输出参数w、b和损失。（提交运行结果）

```
Epoch 50 : cost = 0.080133505 W = [0.8694852, 0.09120809, -0.09722608] b =
-0.5061119
Epoch 100 : cost = 0.066621765 W = [0.8607483, 0.07598354, -0.09534014] b
= -0.42715374
Epoch 150 : cost = 0.05782079 W = [0.85421735, 0.06394225, -0.09392096] b
= -0.36338887
Epoch 200 : cost = 0.052071217 W = [0.84935343, 0.054233544, -0.09283787]
b = -0.3119296
Epoch 250 : cost = 0.048311036 W = [0.845748, 0.046406224, -0.09201265] b
= -0.2704062
Epoch 300 : cost = 0.04584833 W = [0.8430924, 0.040095642, -0.091385566] b
= -0.23689915
Epoch 350 : cost = 0.04423272 W = [0.8411463, 0.035007823, -0.09090966] b
= -0.2098622
Epoch 400 : cost = 0.04317076 W = [0.83973324, 0.03090614, -0.09054963] b
= -0.18804795
Epoch 450 : cost = 0.042471003 W = [0.8387156, 0.027599437, -0.090277895]
b = -0.17044759
Epoch 500 : cost = 0.042008586 W = [0.83799183, 0.024933731, -0.090073496]
b = -0.15624769
Epoch 550 : cost = 0.04170195 W = [0.83748287, 0.022784727, -0.08992007] b
= -0.14479145
Epoch 600 : cost = 0.041497763 W = [0.8371315, 0.0210522, -0.08980534] b =
-0.1355486
Epoch 650 : cost = 0.04136116 W = [0.8368937, 0.019655533, -0.089719795] b
= -0.12809238
Epoch 700 : cost = 0.041269273 W = [0.83674145, 0.018529829, -0.08965682]
b = -0.12207809
Epoch 750 : cost = 0.04120704 W = [0.836645, 0.01762225, -0.089610055] b =
-0.11722613
Epoch 800 : cost = 0.04116459 W = [0.8365884, 0.016890716, -0.089575574] b
= -0.113312766
Epoch 850 : cost = 0.041135382 W = [0.836561, 0.016300987, -0.08955053] b
= -0.110155925
Epoch 900 : cost = 0.041115113 W = [0.83655745, 0.015825659, -0.08953319]
b = -0.10760962
Epoch 950 : cost = 0.0411009 W = [0.83656275, 0.015442596, -0.08952047] b
= -0.105556086
Epoch 1000 : cost = 0.041090827 W = [0.8365793, 0.015133788, -0.08951206]
b = -0.10389968
```

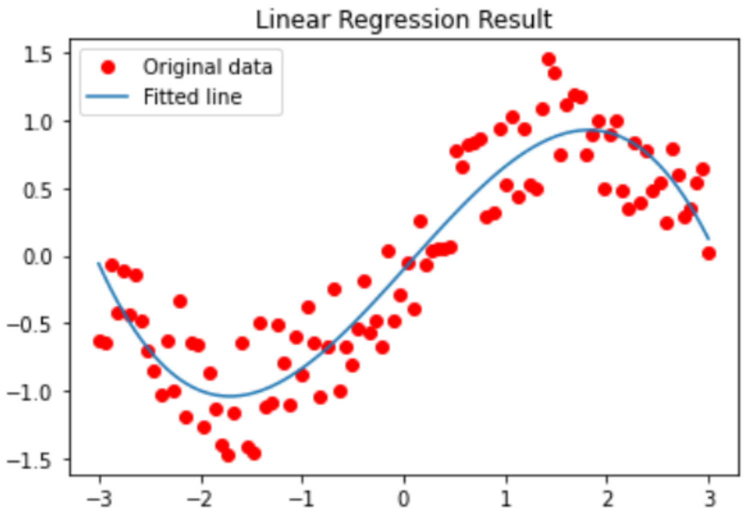
(3) 输出参数w、b和损失。（提交运行结果）\ (4)画出预测回归曲线以及训练数据散点图，对比并分析原因

In [ ]:

```
#@title
# 预测值
predictions = weight[0]*x+weight[1]*x**2+weight[2]*x*x**2 + bias
print("Training cost =", training_cost, "Weight =", weight, "bias =", bias, '\n')

# 画图
plt.plot(x, y, 'ro', label ='Original data')
plt.plot(x, predictions, label ='Fitted line')
plt.title('Linear Regression Result')
plt.legend()
plt.show()
```

```
Training cost = 0.041090827 Weight = [0.8365793, 0.015133788, -0.08951206]
bias = -0.10389968
```



答： 从回归曲线和散点图来看，拟合效果挺好，其原因在于散点数据符合非线性关系，用二次函数可以拟合出较好的效果

3、Softmax 分类
(1) 获取 MNIST 数据集，每张图片像素为28 × 28
(2)模型框架为softmax,以下是训练过程

In [ ]:

```
#@title
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

learning_rate = 0.5
training_epochs = 2000

# X是一个Placeholder ,这个值后续再放入让TF计算，这里是一个784维，但是训练数量不确定的（用None表示）的浮点值
X = tf.placeholder("float", [None,784 ])
Y = tf.placeholder("float", [None, 10])
# 设置对应的权值和偏置的表示，Variable代表一个变量，会随着程序的生命周期做一个改变
# 需要给一个初始的值，这里都全部表示为0
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))

y_pred = tf.nn.softmax(tf.matmul(X, W) + b)

#交叉熵去衡量 reduce_sum 累加
cross_entropy = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(y_pred), reduction_indices=[1
]))
#训练的步骤，告诉tf，用梯度下降法去优化，学习率是0.5，目的是最小化交叉熵
train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(cross_entropy)
# 到目前为止，我们已经定义完了所有的步骤，下面就需要初始化这个训练步骤了，首先初始化所有变量（之前定义的变量）
init = tf.initialize_all_variables()


sess=tf.Session()

# 初始化
sess.run(init)

#记录
cost_list=[]
accur_list=[]
best_cost=[]
best_accur=[]
correct_prediction = tf.equal(tf.argmax(Y,1), tf.argmax(y_pred,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
temp_bc=1;temp_ba=0
# epoch训练
for epoch in range(training_epochs):


    # 随机梯度下降，一个一个feed
    batch_xs, batch_ys = mnist.train.next_batch(100,shuffle=True)
    sess.run(train_step, feed_dict = {X : batch_xs, Y : batch_ys})
    c = sess.run(cross_entropy, feed_dict = {X : batch_xs, Y : batch_ys})
    a = sess.run(accuracy, feed_dict={X: mnist.test.images, Y: mnist.test.labels})
    cost_list.append(c)
    accur_list.append(a)
    if (temp_bc>c):
      best_cost.append(c)
      temp_bc=c
    else:
      best_cost.append(temp_bc)
    if (temp_ba<a):
      best_accur.append(a)
```

```
        temp_ba=a
    else:
        best_accur.append(temp_ba)
    # 显示w b
    if (epoch + 1) % 50 == 0:
        c = sess.run(cross_entropy, feed_dict = {X : batch_xs, Y : batch_ys})
        print("Epoch", (epoch + 1), ": cost =", c, "b =", sess.run(b))


print(sess.run(accuracy, feed_dict={X: mnist.test.images, Y: mnist.test.labels}))
```

```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
Epoch 50 : cost = 0.488661 b = [-0.08365311  0.17071919 -0.01936617 -0.039
8109   0.04756831  0.12266015
 -0.03735828  0.10729032 -0.23253372 -0.03551574]
Epoch 100 : cost = 0.28145513 b = [-0.11440752  0.23287612 -0.03404478 -0.
09278446  0.05192118  0.25853932
 -0.03208613  0.14459965 -0.38564333 -0.02896999]
Epoch 150 : cost = 0.4099089 b = [-0.13395491  0.2536639  -0.01877126 -0.1
0980898  0.07138553  0.32393235
 -0.03772432  0.21619423 -0.4951012  -0.06981513]
Epoch 200 : cost = 0.37972486 b = [-0.14302328  0.286344   -0.01859528 -0.
11127625  0.06263638  0.41599742
 -0.05539409  0.25708807 -0.6107534  -0.08302315]
Epoch 250 : cost = 0.26081622 b = [-0.15754078  0.2982701  -0.00481044 -0.
13929795  0.05569964  0.5119739
 -0.05050829  0.26923695 -0.69256836 -0.09045447]
Epoch 300 : cost = 0.3585502 b = [-0.18628873  0.31043202  0.0024429  -0.1
5927848  0.06799024  0.60696745
 -0.04855032  0.318889   -0.77167046 -0.14093305]
Epoch 350 : cost = 0.31131023 b = [-0.19237146  0.30520523  0.00364579 -0.
1941037   0.05370741  0.6732599
 -0.05486214  0.3706587  -0.82410055 -0.14103861]
Epoch 400 : cost = 0.23199815 b = [-0.22584778  0.30957863  0.04228301 -0.
17169496  0.05705787  0.7282641
 -0.06800772  0.38669467 -0.9155795  -0.14274773]
Epoch 450 : cost = 0.23570156 b = [-0.22694992  0.31017455  0.05393755 -0.
18725136  0.0555296   0.79694295
 -0.06658499  0.39234045 -0.9710894  -0.15704903]
Epoch 500 : cost = 0.16716383 b = [-0.2540482   0.3228966   0.0273614  -0.
1796646   0.0709409   0.88400394
 -0.05751431  0.39317328 -1.0362443  -0.17090437]
Epoch 550 : cost = 0.39077964 b = [-0.2849169   0.33173046  0.04024429 -0.
19251645  0.06142029  0.91891205
 -0.05885805  0.43331265 -1.0710317  -0.17829618]
Epoch 600 : cost = 0.14842016 b = [-0.29922882  0.3392674   0.05171284 -0.
19243403  0.05549396  0.94685644
 -0.08443061  0.46439105 -1.1038734  -0.17775455]
Epoch 650 : cost = 0.17474252 b = [-0.30483067  0.33808962  0.07057053 -0.
20372397  0.03843868  1.0016346
 -0.08192864  0.49856994 -1.1542704  -0.20254964]
Epoch 700 : cost = 0.18353167 b = [-0.33413532  0.35213998  0.09920752 -0.
2109599   0.03008954  1.0269748
 -0.08386698  0.48921704 -1.1895704  -0.17909671]
Epoch 750 : cost = 0.23695326 b = [-0.31150895  0.3381859   0.09752315 -0.
2069099   0.01319031  1.0791265
 -0.08264916  0.51780033 -1.2356678  -0.2090905 ]
Epoch 800 : cost = 0.2639675 b = [-0.30839658  0.33418274  0.06366304 -0.2
1682994  0.01048653  1.1191083
 -0.08949833  0.5658282  -1.2604716  -0.21807247]
Epoch 850 : cost = 0.29129654 b = [-0.33361033  0.3542038   0.08155959 -0.
23540735  0.02012694  1.1596742
 -0.09241668  0.5860593  -1.3003083  -0.23988079]
Epoch 900 : cost = 0.3436376 b = [-0.32888743  0.34512475  0.08422786 -0.2
379457   0.01247232  1.1974183
 -0.12152442  0.602313   -1.3302871  -0.22291182]
Epoch 950 : cost = 0.210371 b = [-0.361407    0.36511153  0.09824184 -0.25
541097  0.01519365  1.2493272
 -0.12838635  0.59371823 -1.3467643  -0.22962369]
```

```
Epoch 1000 : cost = 0.23031451 b = [-0.38475788  0.34498206  0.09988663 -
0.26995122  0.04080839  1.3154526
 -0.12319298  0.6278574  -1.4136434  -0.23744151]
Epoch 1050 : cost = 0.14097853 b = [-0.39332268  0.36191761  0.10647235 -
0.25369605  0.00885555  1.3602668
 -0.12005255  0.6373072  -1.4534397  -0.25430852]
Epoch 1100 : cost = 0.27251866 b = [-0.40173915  0.38285276  0.08664355 -
0.24258211  0.02357661  1.3817452
 -0.10047836  0.63263893 -1.500327   -0.26233068]
Epoch 1150 : cost = 0.18916047 b = [-0.4255953   0.38300732  0.11428412 -
0.2574125   0.01280986  1.3961093
 -0.12134478  0.65459305 -1.5100858  -0.2463657 ]
Epoch 1200 : cost = 0.16208476 b = [-0.42332143  0.3651082   0.08720616 -
0.25720593  0.00631885  1.4238651
 -0.11832412  0.6878039  -1.5437615  -0.22768956]
Epoch 1250 : cost = 0.2912988 b = [-0.42742306  0.37978256  0.12939985 -0.
25554436  0.00850067  1.4343749
 -0.13042475  0.6834646  -1.5643682  -0.25776237]
Epoch 1300 : cost = 0.30018324 b = [-0.42094752  0.38439944  0.12330632 -
0.27948052  0.02680703  1.4510151
 -0.15773612  0.7187701  -1.5704187  -0.27571547]
Epoch 1350 : cost = 0.1829716 b = [-0.44669062  0.39920452  0.11078816 -0.
27647886  0.02112874  1.5179212
 -0.16937396  0.7230758  -1.5872433  -0.29233202]
Epoch 1400 : cost = 0.14309293 b = [-0.45513368  0.39844885  0.14383586 -
0.2836951   0.01472179  1.5478451
 -0.16800202  0.73476547 -1.6275946  -0.30519196]
Epoch 1450 : cost = 0.20628296 b = [-0.46998513  0.3775659   0.15388933 -
0.2857216  -0.00563685  1.5722858
 -0.16036299  0.77526003 -1.6482998  -0.3089945 ]
Epoch 1500 : cost = 0.18379861 b = [-0.47949043  0.3930295   0.15196593 -
0.2829428   0.00591085  1.6003625
 -0.17448677  0.7563745  -1.676146   -0.29457724]
Epoch 1550 : cost = 0.19577932 b = [-0.48844457  0.40325516  0.12549321 -
0.31182563  0.0026698   1.6144122
 -0.14811446  0.7739901  -1.683871   -0.28756395]
Epoch 1600 : cost = 0.2769577 b = [-0.49768943  0.41168645  0.13910463 -0.
3350251  -0.00564481  1.6588088
 -0.15119526  0.7783409  -1.700356   -0.29802886]
Epoch 1650 : cost = 0.16987345 b = [-0.5064674   0.43602026  0.13344248 -
0.34140706 -0.00963341  1.6708522
 -0.15752529  0.802249   -1.7167922  -0.31073818]
Epoch 1700 : cost = 0.24087252 b = [-0.5076963   0.41346487  0.14675175 -
0.35349214 -0.01295409  1.6807464
 -0.1475813   0.83791876 -1.7313204  -0.32583684]
Epoch 1750 : cost = 0.2963069 b = [-0.51999116  0.42618465  0.17074671 -0.
33403686  0.00472093  1.6620561
 -0.14525907  0.8344774  -1.771195   -0.3277025 ]
Epoch 1800 : cost = 0.19326149 b = [-0.5049628   0.41405058  0.15726803 -
0.34736732 -0.01262577  1.6707066
 -0.1578789   0.84048873 -1.7677625  -0.29191586]
Epoch 1850 : cost = 0.22758035 b = [-0.5156319   0.41969594  0.18213469 -
0.3654692  -0.01618248  1.6911098
 -0.14899084  0.84484893 -1.791369   -0.30014467]
Epoch 1900 : cost = 0.16427647 b = [-0.5295494   0.41734284  0.18127552 -
0.3652741  -0.03062802  1.7340747
 -0.15532357  0.86009395 -1.8075061  -0.30450433]
Epoch 1950 : cost = 0.28043708 b = [-0.54871255  0.41585115  0.17616776 -
0.35973105 -0.01214821  1.7534658
 -0.16390587  0.8598036  -1.8031969  -0.31759185]
Epoch 2000 : cost = 0.14477982 b = [-0.5601682   0.42155898  0.18561807 -
```

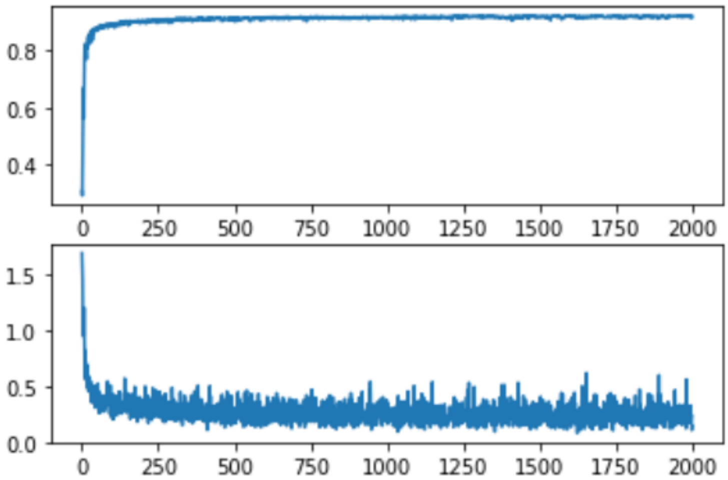
```
0.36926314 -0.01576897  1.7741855
 -0.17249899  0.863118   -1.8020524  -0.32472673]
0.9214
```

(3) 画出训练和测试过程的准确率随迭代次数变化图，画出训练和测试过程的损 失随迭代次数变化图。（提交最终分类精度、分类损失以及两张变化图）

准确率变化与损失变化

In [ ]:

```
#@title
fig,ax=plt.subplots(2,1)
ax[0].plot(range(training_epochs) ,accur_list)
ax[1].plot(range(training_epochs),cost_list)
plt.show()
```



准确率变化与损失变化 （Global Best)

In [ ]:

```
#@title
fig,ax=plt.subplots(2,1)
ax[0].plot(best_accur,'r-')
ax[1].plot(best_cost,'b-')
```

Out[ ]:

```
[<matplotlib.lines.Line2D at 0x7f26f2087668>]
```