实验二 BP网络的训练与测试

文荟俨 19S103256

1. 实验目的

1.编程实现3-3-4型BP神经网络算法，熟练掌握前向传播和反向传播学习算法，生成4类给定数据进行训练和测试，用训练好的网络对5个样本做预测，并与理论计算的结果作对比。

2.探究BP算法中学习因子算法收敛趋势、收敛速度之间的关系。

1. 实验计划

1.生成数据

采用np.random.multivariate\_normal (mean, cov, size=None, check\_valid=None, tol=None)方法，它用于生成多元正态分布矩阵。其中mean和cov为必要的传参而size，check\_valid以及tol为可选参数。参数的具体说明如下：

(1)mean：mean是多维分布的均值，维度为1；

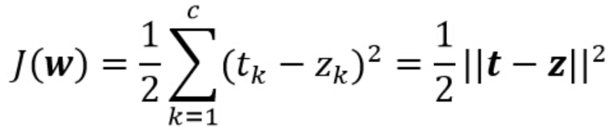
(2)cov：协方差矩阵（协方差基本概念戳这里），注意：协方差矩阵必须是对称的且需为半正定矩阵；

(3)size：指定生成的正态分布矩阵的维度（例：若size=(1, 1, 2)，则输出的矩阵的shape即形状为 1X1X2XN（N为mean的长度））。

(4)check\_valid：这个参数用于决定当cov即协方差矩阵不是半正定矩阵时程序的处理方式，它一共有三个值：warn，raise以及ignore。当使用warn作为传入的参数时，如果cov不是半正定的程序会输出警告但仍旧会得到结果；当使用raise作为传入的参数时，如果cov不是半正定的程序会报错且不会计算出结果；当使用ignore时忽略这个问题即无论cov是否为半正定的都会计算出结果。

2.网络结构定义，前向传播

网络采用3-3-4型BP网，即输入层为3，隐藏层为3，输出层为4，含偏置项，隐藏层激活函数采用ReLu，输出层采用softmax，loss函数J(W)采用均方差，如下所示：



网络结构如图1所示。

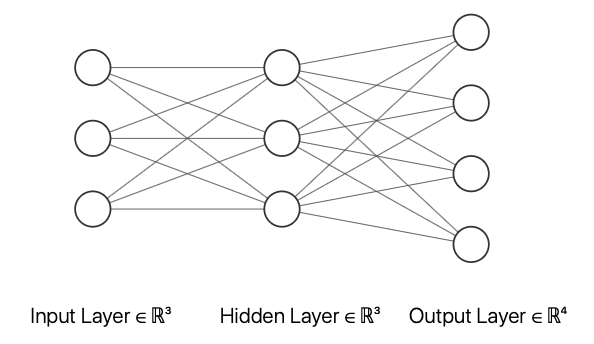
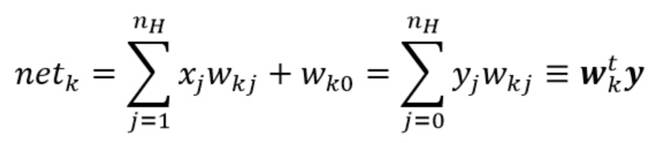


图1 BP神经网络结构图(偏置项未展示)

前向传播的规则如下:



经过上式计算出输出后，经过激活函数得到如下结果：

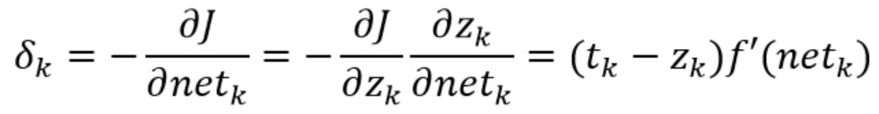


3.反向传播

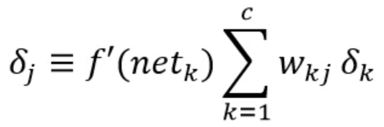
该阶段遵循如下规则:



其中，单元K的敏感度定义为:



隐单元的敏感度定义为:



4.预测

最后,经过softmax之后的输出,求其最大值所在的索引,将它作为预测的标签。规则如下:

np.argmax(softmax())

1. 实验过程

该章节对代码的核心部分作简要说明。均方差的定义如下：

def mean\_square(output, y):  
 return np.sum((y - output) \*\* 2) / 2.0

softmax的定义如下：

def softmax(x):

exps = np.exp(x - np.max(x))

return exps / np.sum(exps)

ReLu函数的定义如下:

def ReLuFunc(x):

x = (np.abs(x) + x) / 2.0

return x

前向传播的定义如下:

# Hidden layer

for j in range(0, 3):

for i in range(0, 3):

net1[j] = net1[j] + w1[j][i] \* x[i]

net1[j] += bias1[j]

fnet1 = ReLuFunc(net1)

# Output layer

for k in range(0, 4):

for j in range(0, 3):

net2[k] = net2[k] + w2[k][j] \* fnet1[j]

net2[k] += bias2[k]

output = softmax(net2)

反向传播的定义如下:

# Back

det\_k = np.zeros(4)

for k in range(0, 4):

det\_k[k] = (y[k] - output[k]) \* (output[k] \* (1 - output[k]))

det\_j = np.zeros(3)

for j in range(0, 3):

sum = 0

for k in range(0, 4):

sum = sum + w2[k][j] \* det\_k[k]

if net1[j] >= 0:

det\_j[j] = sum

else:

det\_j[j] = 0

# Update w and b

for j in range(0, 3):

for i in range(0, 3):

w1[j][i] = w1[j][i] + learning\_rate \* \

det\_j[j] \* x[i]

bias1[j] = bias1[j] + learning\_rate \* det\_j[j]

for k in range(0, 4):

for j in range(0, 3):

w2[k][j] = w2[k][j] + learning\_rate \* \

det\_k[k] \* fnet1[j]

bias2[k] = bias2[k] + learning\_rate \* det\_k[k]

准确率计算定义如下:

if np.argmax(output) == y:

acc += 1

1. 实验结果分析

1.数据生成

初始数据的分布如图2所示。

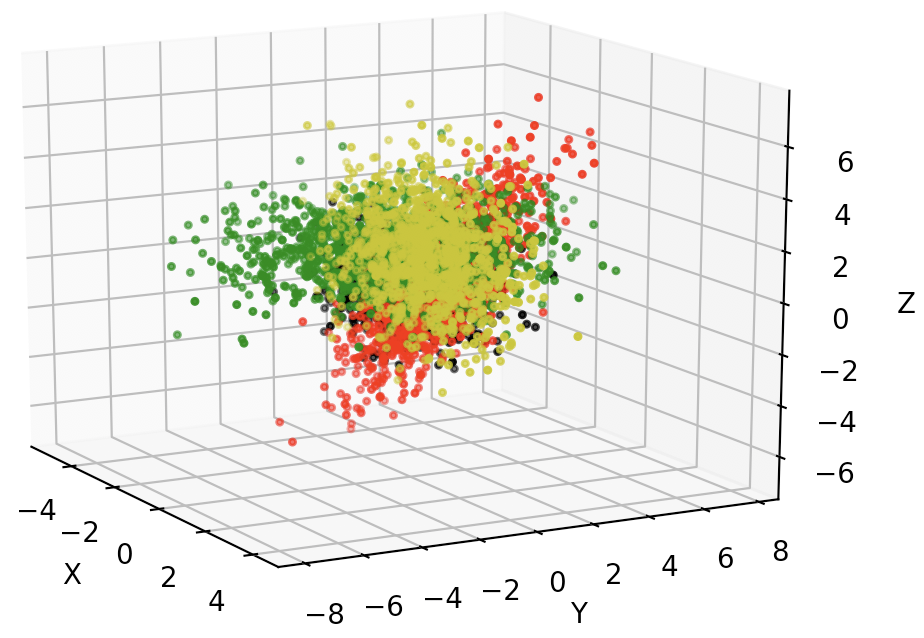


图2 初始样本分布图

2.网络训练及测试集预测

学习率设为0.01，每隔2轮降低为原来的0.97倍，每轮遍历所有的训练样本，得到结果图3所示。其中，最浅色的虚线是测试集400个样本的准确率随训练轮数的变化，最后稳定在51%左右。

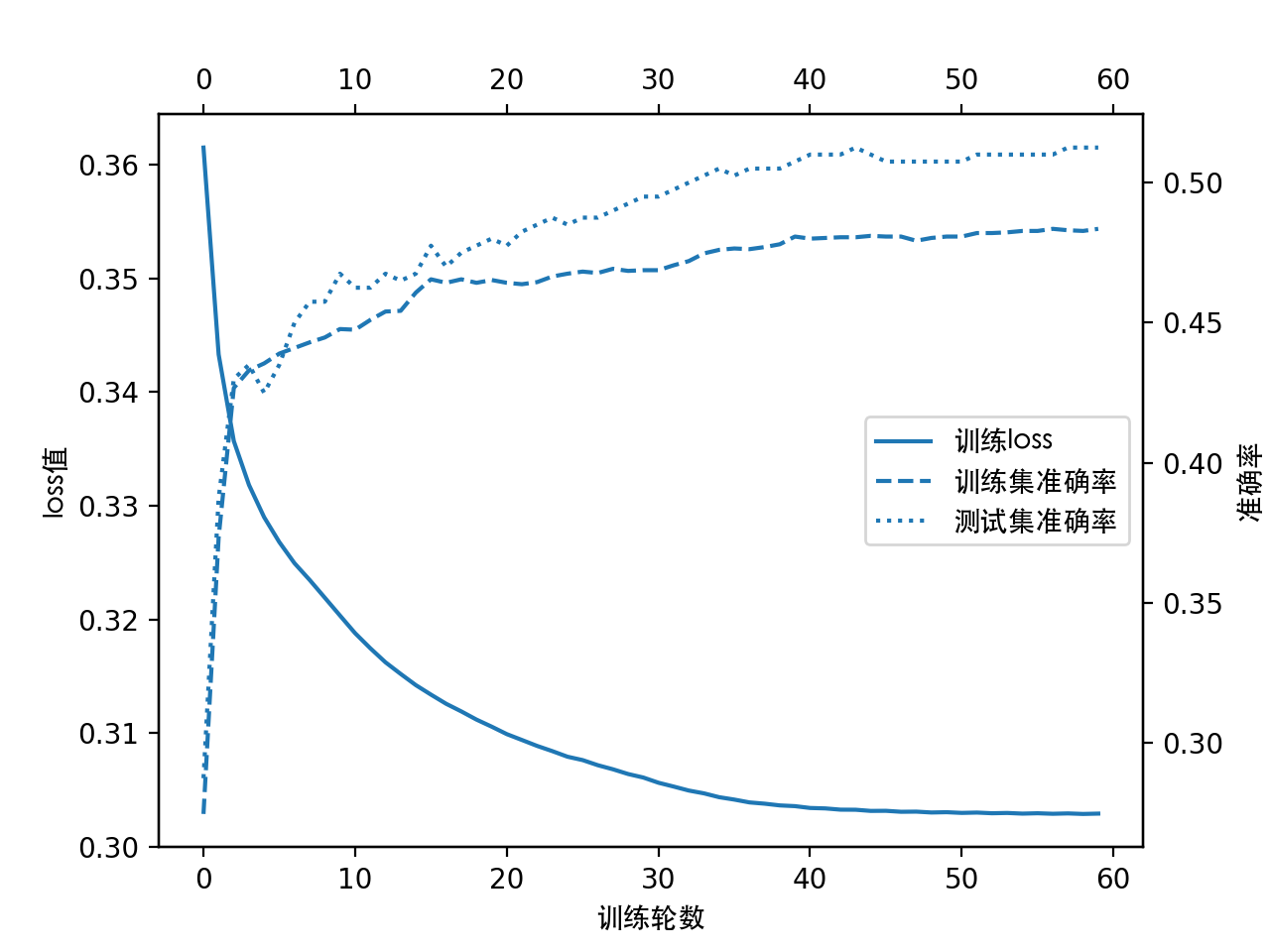
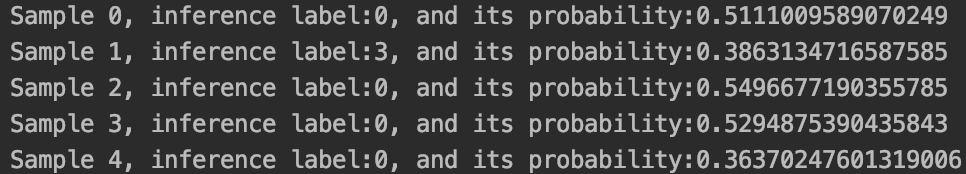


图3 训练过程各指标曲线图

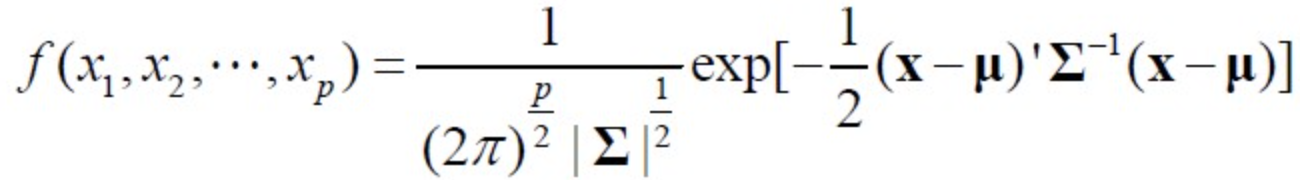
3.给定样本预测

(1)训练网络预测



(2)理论计算

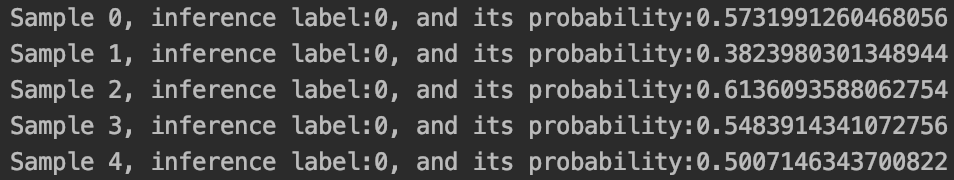
该部分采用贝叶斯函数进行估计，先验概率为0.25，似然函数采用多维高斯分布概率密度函数，其公式定义如下：



具体计算的函数定义如下：

u = np.asarray([[0, 0, 0],  
 [0, 1, 0],  
 [-1, 0, 1],  
 [0, 0.5, 1]])  
sigma = np.asarray([[[1, 0, 0], [0, 1, 0], [0, 0, 1]],  
 [[1, 0, 1], [0, 2, 2], [1, 2, 5]],  
 [[2, 0, 0], [0, 6, 0], [0, 0, 1]],  
 [[2, 0, 0], [0, 1, 0], [0, 0, 3]]])  
p = np.zeros(4)  
for i in range(0, 4):  
 det = np.linalg.det(sigma[i])  
 inv = np.linalg.inv(sigma[i])  
 xu\_ = np.transpose(x - u[i])  
 xu = np.transpose(xu\_)  
 first = np.dot(xu\_, inv)  
 second = np.dot(first, xu)  
 d = len(x)  
 p[i] = 1.0/((2\*np.pi)\*\*(d/2.0)\*np.sqrt(det))\*np.exp(-second/2.0)  
prob = max(p)/np.sum(p)

得到结果如下：



对比网络预测结果如表1所示：

表1 BP网络预测结果与理论计算比较

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 样本数据 | 先验概率 | 理论后验概率(最大值) | 理论结果(最大值对应类别) | 预测后验概率 | 预测结果 |
| (0,0,0) | 0.25 | 0.57 | 0 | 0.51 | 0 |
| (-1,0,1) | 0.25 | 0.38 | 0 | 0.39 | 3 |
| (0.5,-0.5,0) | 0.25 | 0.61 | 0 | 0.55 | 0 |
| (-1,0,0) | 0.25 | 0.55 | 0 | 0.53 | 0 |
| (0,0,-1) | 0.25 | 0.50 | 0 | 0.36 | 0 |

分析：对5个样本的预测，BP网计算的结果是是03000，而理论计算的结果是00000，只有第2个样本的结果不一致，其他均相符。