**实 验 报 告**

**学 院：** 计算机科学与技术学院

**班 级：** 计非211

**学 号：** 202131990522

**姓 名：** 嵇纹垲

**课 程：** 人工智能基础

**指导教师：** 周昌军

**完成时间：** 2024 **年** 4 **月** 22 **日**

**浙 江 师 范 大 学 制**

**实验四：神经网络****实现MINST手写字符识别**

**实验题目：**

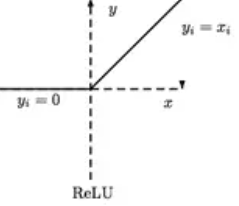
用神经网络解决MINST手写字符识别的问题

**问题分析：**

本次实验与上一次实验一样，都是要实现对数据的分类。但是与上一次实验不同的地方在于，本次要求的分类数目更多，并且数据类型更加复杂，是一个三维矩阵，并且数据量也大了很多。

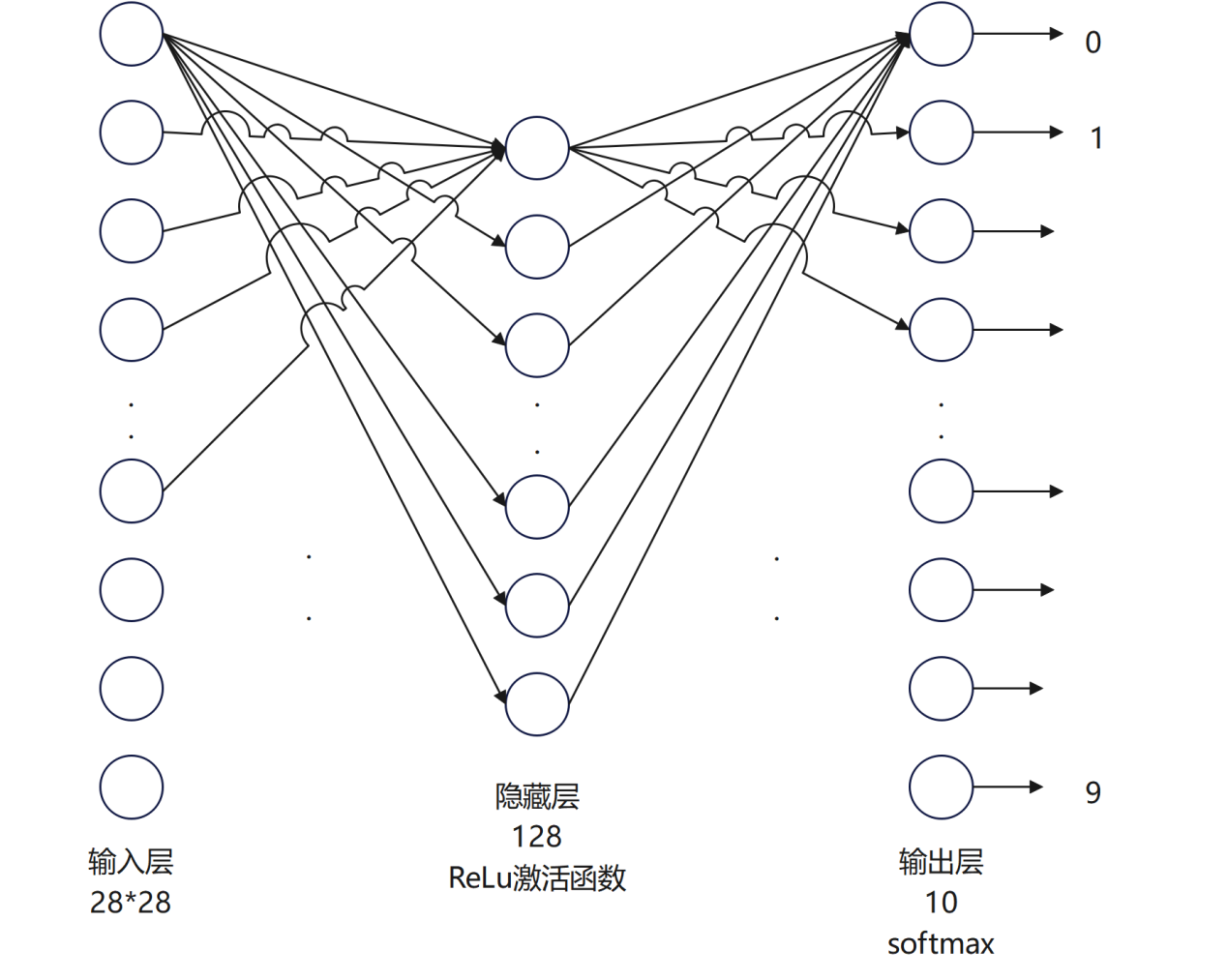
为此，我们需要在原本的简单的输入层->输出层之间新添加一个隐藏层。这是因为在原本在层与层之间是简单的线性函数，很难用纯线性的函数来实现将数据分为10类之多，由此引入隐藏层并且将隐藏层到输出层的函数改成非线性的函数，这样神经网络在学习的过程中就能更好地拟合更加复杂的函数，从而更好地实现分类。

本次实验采用ReLu函数作为隐藏层的激活函数



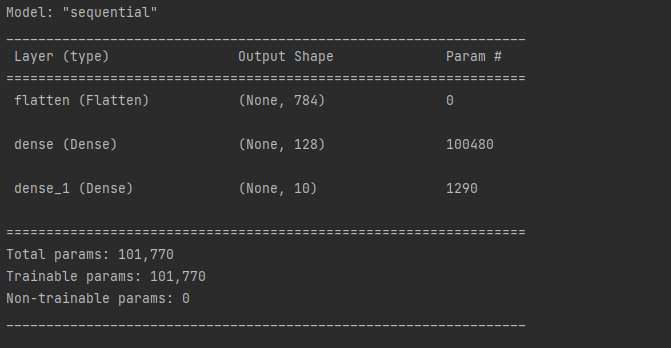
它是一个分段函数,不难发现其导函数的形式非常简单，这可以通过简单的取max操作来替代求导操作，从而简化运算，帮助加速训练过程。

另外注意到每一个输入数据是一个28\*28的灰度图片，所以输入是一个二维矩阵，我们需要将其扁平化为一个长度为28\*28的一位矩阵，作为输入层。由此，神经网络的基本架构就做好了。



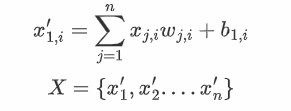
对于神经网络的节点权值的调整，我们还是采用梯度下降法。

神经网络的结构展示如下：

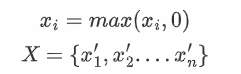


**算法设计：**

**输入层到隐藏层：**



首先是基本的权值转换。



然后用ReLu函数来进行更新。由此我们得到了隐藏层的每一个节点的输入权值

**隐藏层到输出层：**

同样是先进行权值转换，然后用softmax函数来计算划归为每一个分类的概率，最后取高来得到数据的最终分类

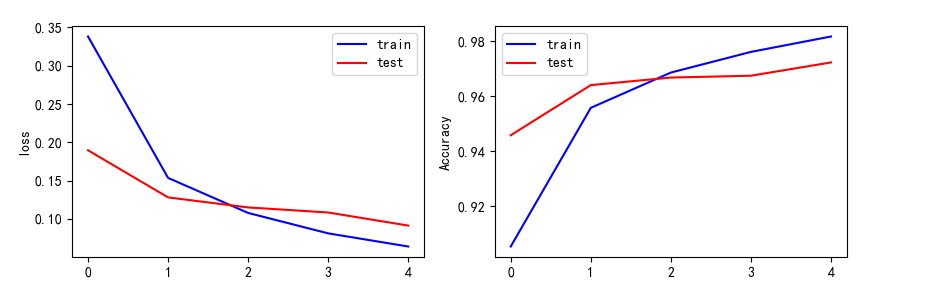
**权值更新：**

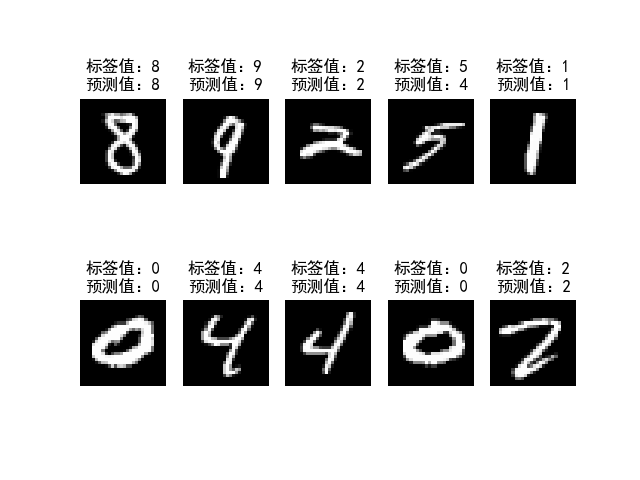
每次训练完之后用对应函数来计算训练的误差，然后将误差反向转播至神经网络的每一层，并且用ReLu函数来沿着梯度的逆方向进行权值的更新。最终完成对神经网络的训练。

**运算结果：**

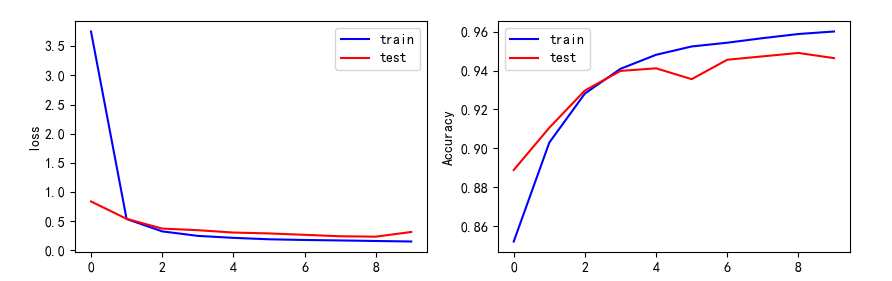
以下展示训练5/100/500次的损失函数和分类准确率的变化

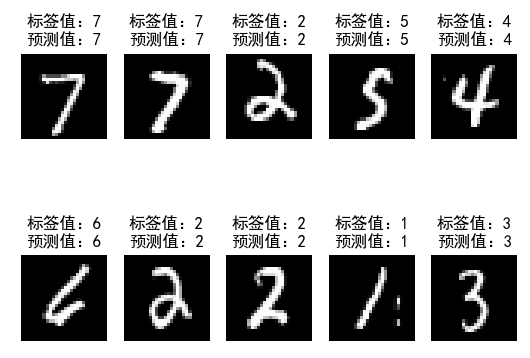
训练5次的结果



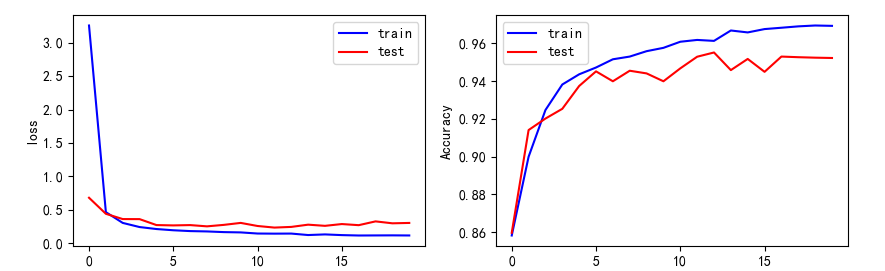


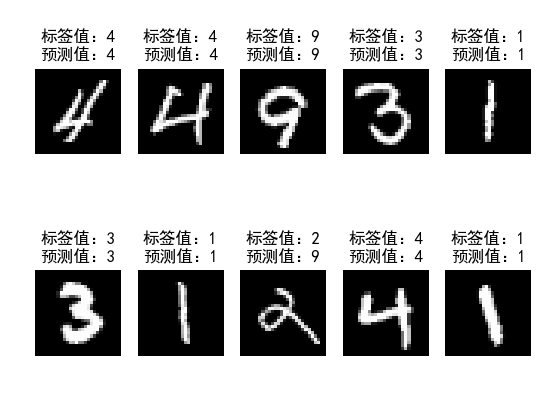
训练100次的结果





训练500次的结果





可以看到，随着预测次数的增加，损失函数确实呈现了下降的趋势，而相对应的，分类的准确率也在不断上升。但是当预测次数上升到一定程度的时候，两者都不再呈现明显的变化。

**实验总结：**

本次实验利用神经网络实现MINST手写字符识别，采用了梯度下降算法对网络参数进行优化，并且引入ReLU激活函数，它为模型引入了非线性特性，提高了模型的表达能力。同时，ReLU函数的稀疏性有助于减少过拟合，提高模型的泛化能力。实验中，我通过调整网络结构、优化算法等手段，实现了较高的识别准确率，验证了神经网络在手写字符识别领域的有效性。

**代码：**

#60000条训练数据和10000条测试数据，28x28像素的灰度图像

#隐含层激活函数：ReLU函数

#输出层激活函数：softmax函数（实现多分类）

#损失函数：稀疏交叉熵损失函数

#输入层有784个节点，隐含层有128个神经元，输出层有10个节点

import tensorflow as tf

import matplotlib.pyplot as plt

import numpy as np

import os

os.environ['TF\_CPP\_MIN\_LOG\_LEVEL'] = '2'# 忽略警告

os.environ["KMP\_DUPLICATE\_LIB\_OK"]="TRUE"

import time

print('--------------')

nowtime = time.strftime('%Y-%m-%d %H:%M:%S')

print(nowtime)

#显示中文

plt.rcParams['font.sans-serif'] = ['SimHei']

#加载数据

mnist = tf.keras.datasets.mnist

(train\_x,train\_y),(test\_x,test\_y) = mnist.load\_data()

# print(type(train\_x))

print('\n train\_x:%s, train\_y:%s, test\_x:%s, test\_y:%s'%(train\_x.shape,train\_y.shape,test\_x.shape,test\_y.shape))

# #数据预处理

# X\_train,X\_test = tf.cast(train\_x/255.0,tf.float32),tf.cast(test\_x/255.0,tf.float32) # 网上用了归一化，但我感觉没必要

X\_train,X\_test = tf.cast(train\_x,tf.float32),tf.cast(test\_x,tf.float32) #归一化

y\_train,y\_test = tf.cast(train\_y,tf.int16),tf.cast(test\_y,tf.int16)

# #建立模型

model = tf.keras.Sequential()

model.add(tf.keras.layers.Flatten(input\_shape=(28,28))) #添加Flatten层说明输入数据的形状

model.add(tf.keras.layers.Dense(128,activation='relu')) #添加隐含层，为全连接层，128个节点，relu激活函数

model.add(tf.keras.layers.Dense(10,activation='softmax')) #添加输出层，为全连接层，10个节点，softmax激活函数

print('\n',model.summary()) #查看网络结构和参数信息

# 配置模型训练方法

# adam算法参数采用keras默认的公开参数，损失函数采用稀疏交叉熵损失函数，准确率采用稀疏分类准确率函数

model.compile(optimizer='adam',loss='sparse\_categorical\_crossentropy',metrics=['sparse\_categorical\_accuracy'])

# #训练模型

# #批量训练大小为64，迭代5次，测试集比例0.2（48000条训练集数据，12000条测试集数据）

print('--------------')

nowtime = time.strftime('%Y-%m-%d %H:%M:%S')

print('训练前时刻：'+str(nowtime))

#

history = model.fit(X\_train,y\_train,batch\_size=64,epochs=20,validation\_split=0.2)

#

print('--------------')

nowtime = time.strftime('%Y-%m-%d %H:%M:%S')

print('训练后时刻：'+str(nowtime))

#评估模型

model.evaluate(X\_test,y\_test,verbose=2) #每次迭代输出一条记录，来评价该模型是否有比较好的泛化能力

#

# #保存模型参数

model.save\_weights('./mnist\_weights.h5')

# #保存整个模型

model.save('mnist\_weights.h5')

#结果可视化

print(history.history)

loss = history.history['loss'] #训练集损失

val\_loss = history.history['val\_loss'] #测试集损失

acc = history.history['sparse\_categorical\_accuracy'] #训练集准确率

val\_acc = history.history['val\_sparse\_categorical\_accuracy'] #测试集准确率

plt.figure(figsize=(10,3))

plt.subplot(121)

plt.plot(loss,color='b',label='train')

plt.plot(val\_loss,color='r',label='test')

plt.ylabel('loss')

plt.legend()

plt.subplot(122)

plt.plot(acc,color='b',label='train')

plt.plot(val\_acc,color='r',label='test')

plt.ylabel('Accuracy')

plt.legend()

#暂停5秒关闭画布，否则画布一直打开的同时，会持续占用GPU内存

#根据需要自行选择

#plt.ion() #打开交互式操作模式

#plt.show()

#plt.pause(5)

#plt.close()

#使用模型

plt.figure()

for i in range(10):

num = np.random.randint(1,10000)

plt.subplot(2,5,i+1)

plt.axis('off')

plt.imshow(test\_x[num],cmap='gray')

demo = tf.reshape(X\_test[num],(1,28,28))

y\_pred = np.argmax(model.predict(demo))

plt.title('标签值：'+str(test\_y[num])+'\n预测值：'+str(y\_pred))

#y\_pred = np.argmax(model.predict(X\_test[0:5]),axis=1)

#print('X\_test[0:5]: %s'%(X\_test[0:5].shape))

#print('y\_pred: %s'%(y\_pred))

#plt.ion() #打开交互式操作模式

plt.show()

#plt.pause(5)

#plt.close()