**中山大学计算机学院**

**人工智能**

**本科生实验报告**

**（2022学年春季学期）**

课程名称：Artificial Intelligence

|  |  |  |  |
| --- | --- | --- | --- |
| 教学班级 | **系统结构班** | 专业（方向） | **计算机科学与技术** |
| 学号 | **21307358** | 姓名 | **曾慧蕾** |

# 实验题目

* 在MNIST手写数字数据集完成图像分类任务，在测试集完成测试，计算准确率。
* 要求设计多层感知机(至少两层全连接层)，并使用激活函数，选择合适的损失函数，并手动推导参数更新公式，利用训练集完成网络训练，并在测试集上计算准确率
* 报告要求给出参数更新公式的推导，损失的可视化展示，以及学习率对准确率影响的可视化展示

# 实验内容

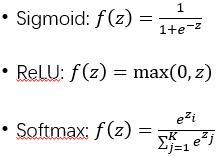
1. 算法原理

在神经网络多层感知机识别 MNIST 手写数据集中，通常使用的算法有梯度下降算法。具体来说，我们首先初始化神经网络的权重和偏置，然后使用训练集中的样本来计算输出误差。最后，我们使用梯度下降算法来更新权重和偏置，以使损失函数最小化。这个过程通常需要多次迭代，直到损失函数收敛或达到最大迭代次数。

* **激活函数：**

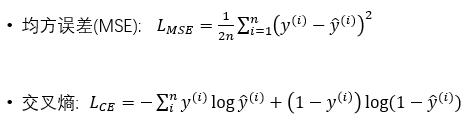
在神经网络中传递信号时，每个神经元都会对输入信号进行加权求和，并将其传递给下一层神经元。但是，如果只使用线性函数来激活神经元，那么整个神经网络就只能表示线性函数，这会大大降低神经网络的表达能力。因此，我们需要使用非线性函数来激活神经元，这些非线性函数被称为激活函数，这在多层感知机中非常重要。

激活函数的作用是将神经元的输入信号转换为输出信号，并引入非线性性质。这样，神经网络就可以表示更加复杂的函数，从而提高其表达能力。常见的激活函数包括：



* **损失函数：**

损失函数是用来衡量模型预测结果与真实结果之间的差异的函数。通常情况下，我们希望模型的预测结果尽可能地接近真实结果，因此我们需要使用损失函数来度量模型的预测误差。在分类问题中，我们通常使用交叉熵损失函数来度量模型的预测结果与真实标签之间的差异。



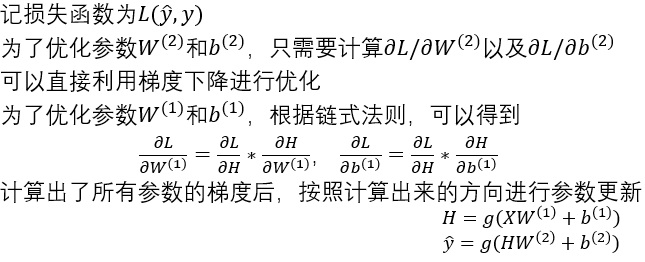
* **梯度下降算法：**

梯度下降算法是一种用于最小化损失函数的优化算法。其基本思想是沿着损失函数的梯度方向更新参数，以使损失函数最小化。具体来说，梯度下降算法分为两个阶段：计算梯度和更新参数。在计算梯度阶段，我们计算损失函数对每个参数的偏导数。在更新参数阶段，我们沿着梯度的反方向更新参数，以使损失函数最小化。

梯度下降的一般公式为：（其中𝜃为参数，为函数梯度，𝜂为学习率）

1. 手动推导参数更新

梯度下降的优化多层感知机的思路为：

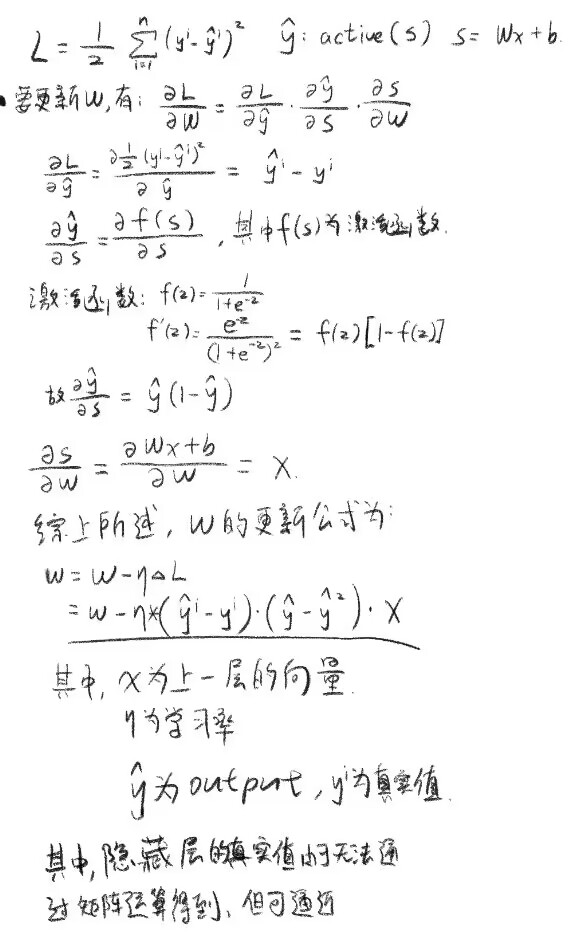


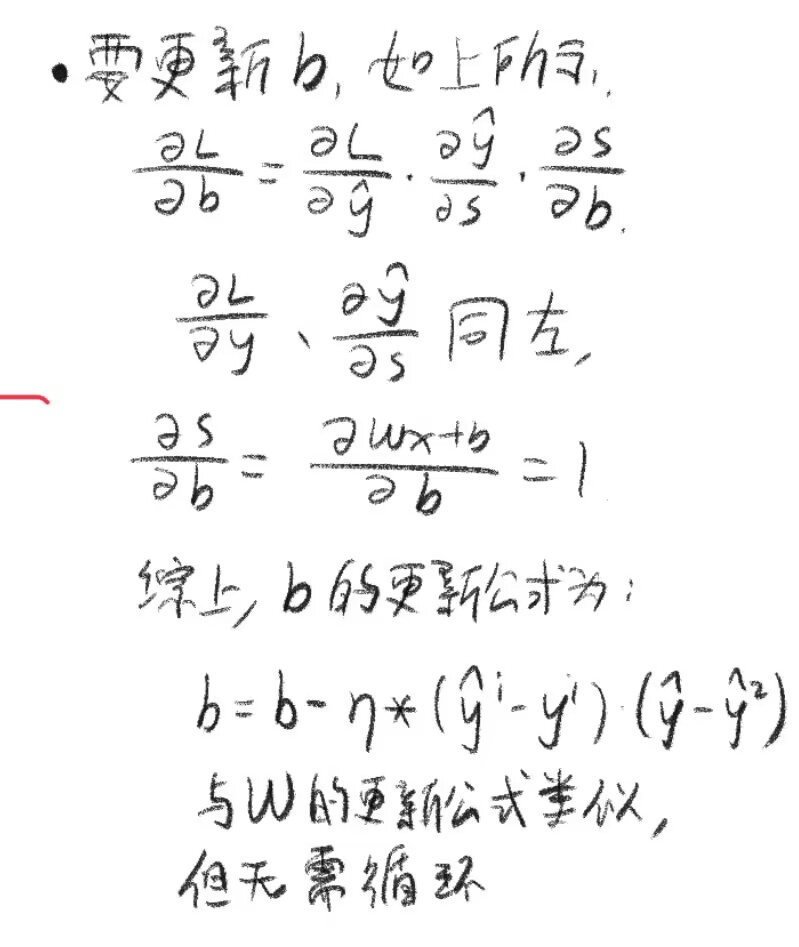
构造的神经网络有两个全连接层，包括一层隐藏层和一层输出层，因此推导公式仅需要推出一项，此后的皆可类推。

基于此，我们的手动推导参数更新公式如下所示：



首先确定各符号

此为权重矩阵的更新公式

此为偏移量的更新公式

1. 关键代码展示（带注释）

数据导入+参数初始化：

def initialization():

    # MNIST数据集中的图片和标签数据是按照大端字节序存储

    file=[  "D:\\code\\python\\file\\lab10\\minst\\train-images.idx3-ubyte", # 训练集图像 用坐标来表示 大小从0~255 长28宽28

            "D:\\code\\python\\file\\lab10\\minst\\train-labels.idx1-ubyte", # 跟图片对应的标签

            "D:\\code\\python\\file\\lab10\\minst\\t10k-images.idx3-ubyte",

            "D:\\code\\python\\file\\lab10\\minst\\t10k-labels.idx1-ubyte"]

    data\_set=[decode\_idx3\_ubyte(file[0]),decode\_idx3\_ubyte(file[2])]

    train\_label=[decode\_idx1\_ubyte(file[1])][0]

    test\_label=[decode\_idx1\_ubyte(file[3])][0]

    #将图片矩阵转换成行向量，方便计算

    for i in range(0,len(data\_set[0])): #60000

        train\_list.append(data\_set[0][i].reshape(784,)/255)

    for i in range(0,len(data\_set[1])): #10000

        test\_list.append(data\_set[1][i].reshape(784,)/255)

    # 学习率

    a1=0.3

    a2=0.3

    #权重初始化

    w1=np.random.uniform(-0.5,0.5,(784,100)) # 输入层结点数 & 隐藏层结点数

    w2=np.random.uniform(-0.5,0.5,(100,10)) # 隐藏层结点数 & 输出层结点数（0~9）

    #偏置向量初始化

    b1=[0 for \_ in range(0,100)]

    b2=[0 for \_ in range(0,10)]

激活函数：

def sigmoid(z):

    ans=[]

    for i in range(0,len(z)):

        e\_z=math.exp(-z[i])

        ans.append(1/(1+e\_z))

    return np.array(ans)

训练部分，包括多层感知机的各参数优化。更新公式为第二部分给出的公式。其中：隐藏层的损失函数通过输出层的损失函数反向传播得到近似值。

def train(a1,a2,b1,b2,w1,w2,loss):

    for i in trange(0,len(train\_list)):

        hide=np.dot(train\_list[i],w1)+b1

        hide\_ans=sigmoid(hide)

        output=np.dot(hide\_ans,w2)+b2

        output\_ans=sigmoid(output)

        real\_ans=np.array([0 for \_ in range(0,10)])

        real\_ans[int(train\_label[i])]=1

        # 更新参数

        e2=real\_ans-output\_ans

        dt\_output=output\_ans\*(1-output\_ans)\*e2 #三个同维向量相乘

        e1=np.dot(w2,e2) #e1通过e2倒推回去 回到100\*1的状态 虽然不是完全符合的值但是接近

        dt\_hide=hide\_ans\*(1-hide\_ans)\*e1

        for j in range(0,10):

            w2[:,j]+=a2\*dt\_output[j]\*hide\_ans

        for j in range(0,100):

            w1[:,j]+=a1\*dt\_hide[j]\*train\_list[i]

        b2+=a2\*dt\_output

        b1+=a1\*dt\_hide

        if i%100==0:

            loss.append(abs(np.mean(e2)))

1. 创新点&优化（如果有）

通过多次调整学习率的值，最终找到了能使正确率达到最高的一个值。实验结果将在第三部分展示。

    a1=0.05

    a2=0.05

    lr=[]

    x\_=[]

    while a1<1:

        loss=[]

        train(a1,a2,b1,b2,w1,w2,loss)

        test(b1,b2,w1,w2,lr)

        if a1==0.3:

            p1=plt.figure(1)

            x=[i for i in range(0,len(loss))]

            plt.plot(x,loss)

            plt.title("loss graph")

        x\_.append(a1)

        a1+=0.05

        a2+=0.05

    p2=plt.figure(2)

    plt.plot(x\_,lr)

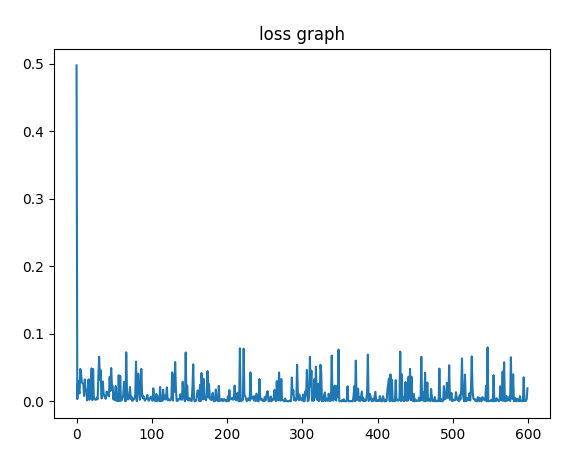
    plt.title("learn rate and correct rate")

    plt.show()

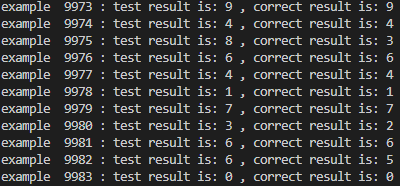
# 实验结果及分析

1. 实验结果展示示例（可图可表可文字，尽量可视化）

当学习率a1=0.3，a2=0.3时，损失函数可视化结果如下：

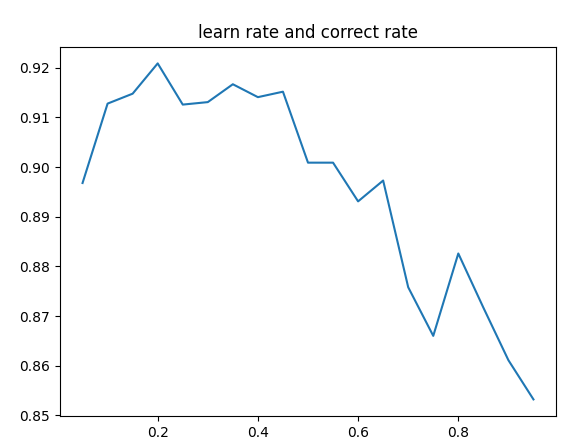


同时，部分识别结果以及最终准确率如下所示：

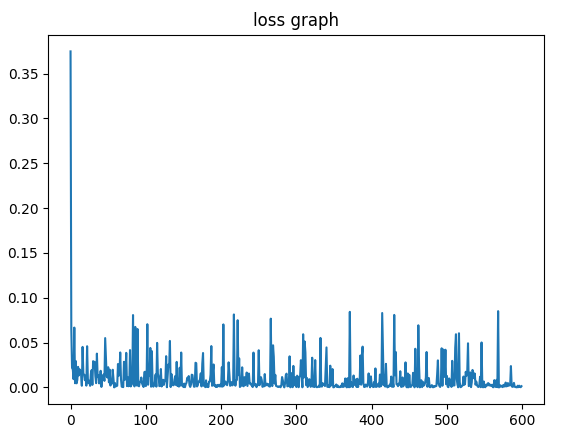




2. 评测指标展示及分析（机器学习实验必须有此项，其它可分析运行时间等）



**此为正确率受学习率影响的可视化图。每次迭代学习率会增长0.05然后重新训练神经网络，得出结果如图所示，可见当a=0.2时正确率最高。根据测试，当a=0.2时，loss曲线图和正确率如下所示：**





# 参考资料

[mnist解析代码参考：idx-ubyte文件解析\_idx3-ubyte](https://blog.csdn.net/lindorx/article/details/94639183)

[神经网络的梯度下降公式推导](https://blog.csdn.net/JasonDean/article/details/122778527)

[NumPy 教程 | 菜鸟教程 (runoob.com)](https://www.runoob.com/numpy/numpy-tutorial.html)

超算习堂-课时26-第14讲实验课/-课时25-第13讲