Exercise1-Softmax Regression

学院：网络空间安全学院 学号：2111673 专业：物联网工程 姓名：岳志鑫

一、问题分析

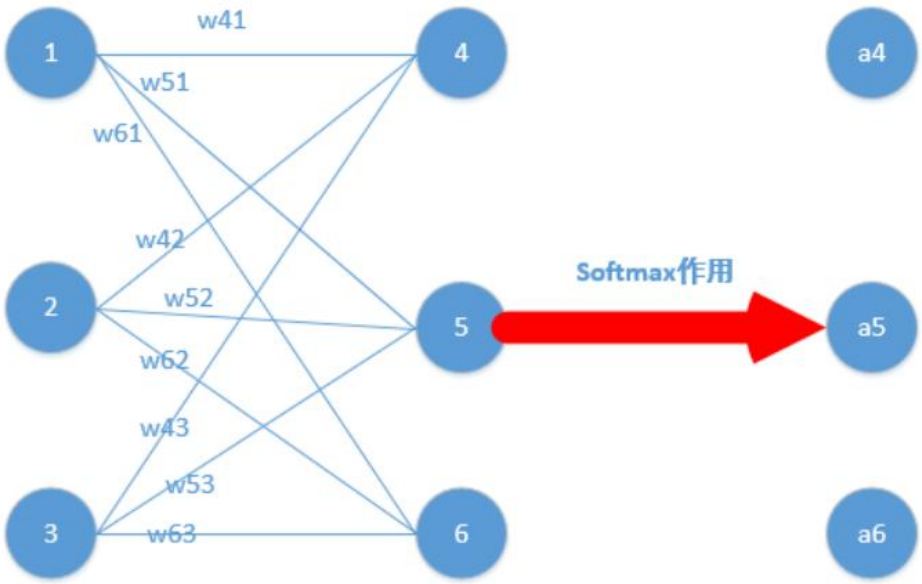
在这个练习中，需要训练一个分类器来完成对 MNIST 数据集中 0-9 10 个手写数字的分类。你的任务是在 softmax_regression.py 文件中实现 softmax 目标函数，计算损失值 J(θ, x, y)，将它存储在变量 f 中，并计算梯度 $\nabla_\theta J(\theta(x, y)$ 将它存储在变量 g 中。初始代码会将 θ 的形状定义为一个 k×n 的矩阵 (K=10 个类别)。此外，你需要完成函数 cal_accuracy()，输出分类器在测试集上的准确率。

程序中涉及到的超参可以自行设置，尽可能得到较高的分类准确率。

二、实验原理

1.Softmax Regression

Softmax 回归（Softmax Regression），也称为多类逻辑回归（Multinomial Logistic Regression），是一种用于多分类问题的分类算法，本质是把全连接层的输出序列变成一个概率序列。



Softmax 运算可以保持向量中每个元素的相对大小，并将其转换为总和为 1 的概率分布。大致步骤如下

(1)对每一个元素应用指数化操作，得到一个新向量 e

(2)计算指数化得分的总和 sum

(3)对于每个类别的指数化得分除以 sum

(4)这样就得到了概率分布的向量。

Softmax函数的公式

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Softmax 的分子将指数函数应用于向量的每个元素，对于最高的输入值返回最高的输出值。因为它的范围是(0，∞)，所以任何负数也变成正数

Softmax 分母中的求和是通过确保函数的和为 1 来标准化每个元素，创建一个概率分布。所有的指数元素加在一起，所以当每个指数元素除以这个和时，它将是它的一个百分比。

2.交叉熵

为了训练模型，我们需要定义一个 loss 来描述模型对问题的分类精度。loss 越小，代表模型的分类结果与真实值的偏差越小。在多分类问题中常使用交差熵作为损失函数，交叉熵定义如下，其中 y 代表真实值，$\bar{y}$ 代表预测值，n 代表需要区分的类别数，如这里就是 n=10

$$H_y(\tilde{y}) = -\sum_{i}^{n} y log\tilde{y}$$

对同一个问题，用 P、Q 同时描述，用 KL 散度来衡量两个分布的差异：

$$D_{KL}(p||q) = \sum_{i}^{n} p(x_i) log(\frac{p(x_i)}{q(x_i)})$$

在机器学习中，P 往往用来表示真实的样本分布。Q 用来表示模型所预测的分布。当然 Q 越接近 P 越好，这样就说明预测值十分逼近真实值。也就是上式的 D_{KL} 的值越小越好。

对公式 $D_{KL}(\mathrm{p} \| q)$ 进行变形：

$$D_{KL}(p||q) = \sum_{i}^{n} p(x_i) log(p(x_i)) - \sum_{i}^{n} p(x_i) log(q(x_i))$$

$$= -H(p(x)) + [-\sum_{i}^{n} p(x_i) log(q(x_i))]$$

等式的前一部分恰巧就是 p 的熵，等式的后一部分，就是交叉熵：

$$H(p,q) = -\sum_{i}^{n} p(x_i) log(q(x_i))$$

在机器学习中，我们需要评估 label 和 predicts 之间的差距，使用 KL 散度刚刚好，即 ，由于 KL 散度中的前一部分−H(y)不变，故在优化过程中，只需要关注交叉熵就可以了。所以一般在机器学习中直接用用交叉熵做 loss，评估模型。

3.梯度

有了交叉熵的损失函数之后，我们就要找到一个θ矩阵，使得 loss 函数在训练样本空间上最小。常用的方法就是梯度下降算法。

损失函数：

$$loss = -y^T log(y_o) - (1-y)^T log(1-y_o)$$

我们对这个损失求导数来进行梯度下降：

$\frac{\partial loss}{\partial y_o} = -y \oslash y_o + (1-y) \oslash (1-y_o)$ 。 $\frac{\partial loss}{\partial y_o}$ 可以进一步拆解：

$$\begin{aligned}\frac{\partial loss}{\partial y_o} &= \frac{\partial(-y^T log(y_o))}{\partial log(y_o)} \cdot \frac{\partial log(y_o)}{\partial y_o} - \frac{\partial((1-y)^T log(1-y_o))}{\partial log(1-y_o)} \cdot \frac{\partial log(1-y_o)}{\partial y_o} \\ &= -y \cdot diag(y_o)^{-1} - (1-y) \cdot -diag(1-y_o)^{-1} \\ &= -y \oslash y_o + (1-y) \oslash (1-y_o)\end{aligned}$$

梯度更新：

$$\theta = \alpha - \nabla_\theta J(\theta)$$

三、实现方式

1.在 softmax_regression.py 文件中实现 softmax_regression() 函数

传入函数的参数为 theta,x,y,iters,alpha

(1)theta 代表一个服从“0~1”均匀分布的 k 行 n 列矩阵。随机样本取值范围是[0,1)

(2)x 是和样本训练集 train_images 同样大小（m 行 n 列）的矩阵

(3)y 是一个大小为 m 行 k 列，除了主对角线元素为 1 其他都为 0 的矩阵。

(4)iters 代表迭代次数，初始化为 500

(5)alpha 代表学习率，初始化为 0.5

```
def softmax_regression(theta, x, y, iters, alpha):
    # TODO: Do the softmax regression by computing the gradient and
    # the objective function value of every iteration and update the theta
    m, n = x.shape  # 获取输入数据 x 的行数和列数，其中 m 是样本数，n 是特征数。

    for i in range(iters):
        chengji = np.dot(x, theta.T)  # 计算输入特征和参数的乘积之和
        fenzi = np.exp(chengji)  # 计算 softmax 函数的分子部分，对线性组合进行指数化
        # 计算 softmax 函数的分母，对分子在第二个轴上进行求和，保持维度。
```

```
        fenmu = np.sum(fenzi, axis=1, keepdims=True)
        calculate = fenzi / fenmu  # 计算 softmax 的每个类别的概率
        # 计算梯度
        gradient = -np.dot((y.T - calculate).T, x) / m
        # 使用梯度下降更新参数 theta
        theta -= alpha * gradient
        # 计算目标函数值（交叉熵损失）
        loss = -np.sum(np.log(calculate) * y.T) / m
        # 每次迭代时打印损失值
        print(f'第 {i + 1}/{iters} 次迭代，损失值: {loss}')
    return theta
```

定义了损失函数 loss，梯度矩阵 gradient，计算后使用梯度下降更新 theta，重复进行使得损失值越来越小，训练学习的正确率就会越来越高。

2.在 evaluate.py 文件中实现 cal_accuracy()函数

```
def cal_accuracy(y_pred, y):  # y_pred 是模型对测试集的预测结果，y 是测试集的真实标签
    # TODO: Compute the accuracy among the test set and store it in acc
    # 计算准确率
    correct_predictions = np.sum(y_pred == y)  # 预测正确数
    total_samples = len(y)  # 总样本数
    acc = correct_predictions / total_samples
    # 调试输出
    print(f"正确预测数量：{correct_predictions}")
    print(f"总样本数量：{total_samples}")
    return acc
```

预测的函数为 predict，目的是使用给定的参数 theta 对输入的测试图像数据进行分类预测，并返回预测的类别标签

```
def predict(test_images, theta):
    scores = np.dot(test_images, theta.T)# 执行了矩阵乘法运算，将测试图像数据 test_images 与参数 theta 相乘。这个操作得到了每个测试样本在各个类别上的得分值
    preds = np.argmax(scores, axis=1)#使用 np.argmax 函数找到每个测试样本得分最高的类别引
    return preds
```

3.train.py 文件中的 train 函数，调用 softmax_regression 对其进行训练

```
def train(train_images, train_labels, k, iters=5, alpha=0.5):#迭代次数，学习率
    m, n = train_images.shape#获取训练图像的形状，其中 m 是样本数，n 是特征数。
    # data processing
    x, y = data_convert(train_images, train_labels, m, k)  # x:[m,n], y:[1,m]
    # Initialize theta.  Use a matrix where each column corresponds to a class,
    # and each row is a classifier coefficient for that class.
    theta = np.random.rand(k, n)  # [k,n]，初始化模型参数 theta，使用随机值初始化
    # do the softmax regression
    theta = softmax_regression(theta, x, y, iters, alpha)#传入参数进行训练学习
    return theta
```

4.在 main.py 中增加一些调试输出，同时需要对预测的结果进行格式转换，否则会出现标签和预测结果不匹配的问题导致正确率异常

```
# 使用 reshape 转换为二维数组
y_predict_2d = y_predict.reshape(-1, 1)
# 计算正确率
accuracy = cal_accuracy(y_predict_2d, test_labels)
print("Finished test. ")
# 调试输出
print("预测结果：", y_predict_2d)
print("实际标签：", test_labels)
print(f"测试集上的准确率：{accuracy*100:.2f}%")
```

四、实验结果

1.学习率 α 为 0.5 时，准确率达到 90.52%

```
第 495/500 次迭代，损失值：0.3535043564752254
第 496/500 次迭代，损失值：0.35337765342370814
第 497/500 次迭代，损失值：0.3532513102354668
第 498/500 次迭代，损失值：0.35312532515621126
第 499/500 次迭代，损失值：0.35299969644370865
第 500/500 次迭代，损失值：0.35287442236767563
Finished training.
正确预测数量：9052
总样本数量：10000
Finished test.
预测结果： [[7]
 [2]
 [1]
 ...
 [4]
 [5]
 [6]]
实际标签： [[7]
 [2]
 [1]
 ...
 [4]
 [5]
 [6]]
测试集上的准确率：90.52%
```

2.学习率 α 为 0.6 时，准确率达到 90.99%

```
第 496/500 次迭代，损失值：0.34151669950822316
第 497/500 次迭代，损失值：0.34139590427928135
第 498/500 次迭代，损失值：0.341275456515949
第 499/500 次迭代，损失值：0.3411553545449449
第 500/500 次迭代，损失值：0.34103559670434036
Finished training.
正确预测数量：9099
总样本数量：10000
Finished test.
预测结果： [[7]
 [2]
 [1]
 ...
 [4]
 [5]
 [6]]
实际标签： [[7]
 [2]
 [1]
 ...
 [4]
 [5]
 [6]]
测试集上的准确率：90.99%
```

3.学习率 α 为 0.9 时，准确率达到 91.03%

```
第 499/500 次迭代，损失值：0.32025143366582454
第 500/500 次迭代，损失值：0.3201496583763326
Finished training.
正确预测数量：9103
总样本数量：10000
Finished test.
预测结果： [[7]
 [2]
 [1]
 ...
 [4]
 [5]
 [6]]
实际标签： [[7]
 [2]
 [1]
 ...
 [4]
 [5]
 [6]]
测试集上的准确率：91.03%
```

4.学习率 α 为 1.0 时，准确率达到 91.28%

```
第 499/500 次迭代，损失值：0.3202693240508438
第 500/500 次迭代，损失值：0.32006576850581797
Finished training.
正确预测数量：9128
总样本数量：10000
Finished test.
预测结果： [[7]
 [2]
 [1]
 ...
 [4]
 [5]
 [6]]
实际标签： [[7]
 [2]
 [1]
 ...
 [4]
 [5]
 [6]]
测试集上的准确率：91.28%
```

当学习率大于 1 时发现损失值出现跳动，应该是学习率过大导致的过拟合，所以最高准确率就是学习率为 1 时的 91.28%，但也许可以增加训练迭代次数提升准确率。

五、心得体会

本次实验练习了通过 softmax 回归解决多分类问题中的识别十个手写数字，对回归算法的逻辑和知识有了更深入的了解，明白了回归算法中损失函数、梯度下降，交叉熵等概念。

对于此次实验也有一些有待改进的地方，例如迭代次数过多导致的每次调试时间耗费过长、后来发现可以减少迭代次数来先调试学习率或者减少样本数量都可以加快效率，节约时间。