

【个人信息】

- 姓名：王禹曦
- 学号：2011920
- 专业：计算机科学与技术

一、实现原理

1. 计算预测值

- 参数 θ ：一个10*784的矩阵，是每个特征向量 \vec{x}_i 的系数矩阵
- \vec{x} ：一个m*784的矩阵，即共m的特征向量

注：

- 784为每张照片展开为1维的长度，即28*28
 - 训练集中，m=60000，测试集中m=10000
- 计算过程：

$$\hat{y} = \text{softmax}(\theta \cdot x^T)$$

- 其中 `softmax()` 函数：

$$y_j = \frac{e^j}{\sum_{i=1}^{10} e^i}$$

2. 使用交叉熵函数计算损失值

- 公式如下：

$$Loss = -\frac{1}{m} \sum_{j=1}^m \sum_{i=1}^{10} y_i \log(\hat{y}_i)$$

3. 梯度下降、更新参数

- 损失函数对参数求导：

$$grad = \frac{1}{m} \sum_{i=1}^m (\hat{y} - y) \times x_i$$

- 更新参数

$$\theta = \theta - \alpha \times grad$$

二、代码细节

1. `softmax_regression()`：
-

```
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

    # theta:k,n      x: m,n      y:k,m

    m, n = x.shape

    for i in range(iters):

        # 1.计算预测值
        y_hat = softmax(np.matmul(theta, x.T))

        # 2.计算损失值
        loss = -1/m * np.sum(y*np.log(y_hat))
        print(f'Iter{i+1}Loss:', loss)

        # 3.梯度下降、更新参数
        theta -= alpha/m * np.matmul((y_hat - y), x)

    return theta
```

2. softmax():

```
def softmax(x):
    # X:(10,m)
    return np.exp(x) / np.sum(np.exp(x), axis=0)    # 对每列求和 axis=0
```

3. cal_accuracy():

```
def cal_accuracy(y_pred, y):
    # TODO: Compute the accuracy among the test set and store it in acc

    # 将y展平成1维向量
    y = y.flatten()
    # 正确个数
    right = np.sum(y_pred == y)
    # 总个数
    total = y_pred.shape[0]
    acc = right/total

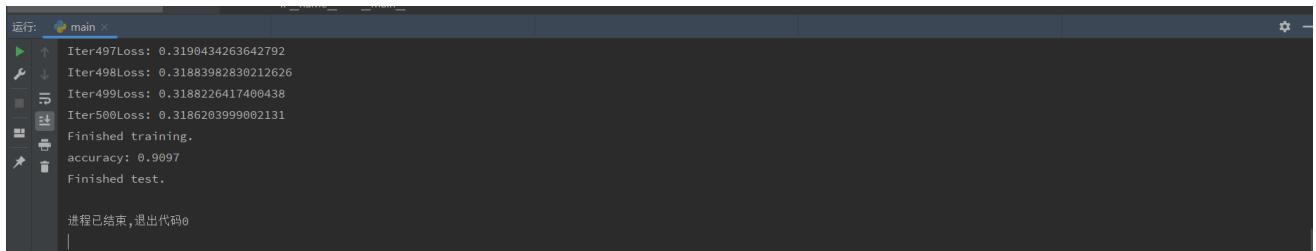
    return acc
```

4. 设置随机数种子

使每次生成的参数为相同值

```
np.random.seed(21)
```

三、实验结果



```
运行: main
↑ Iter497Loss: 0.3190434263642792
↓ Iter498Loss: 0.31883982830212626
Iter499Loss: 0.3188226417400438
Iter500Loss: 0.3186203999002131
Finished training.
accuracy: 0.9097
Finished test.

进程已结束, 退出代码0
```

经过数十次调整参数，得到较好值如上

- $\alpha = 1.0$
- $iter = 500$
- $loss = 0.3186$
- $acc = 90.97\%$

四、实验结果分析

1. 学习率

本人学习率从0.1尝试至1.5，发现

- 学习率较小时，前期与后期的收敛速度均很缓慢
- 学习率较大时，如 $\alpha > 1$ 时，会出现震荡情况，即
 - Loss时大时小，在最小值附近震荡
- 通过观察， $\alpha = 1$ 时，效果最佳

2. 迭代次数

显然，在合理学习率情况下，迭代次数越大，最后loss越小

但由于模型本身以及参数已收敛的原因，增大迭代次数，也不会得到显著的变化

如：

迭代次数从500 \rightarrow 1000，准确率只从90.97%增大到91.27%