南開大學
Nankai University

计算机学院和网络空间安全学院

机器学习作业一实验报告

基于 softmax 实现的手写体识别

姓名：魏伯繁

学号：2011395

专业：信息安全

2022 年 11 月 4 日

目录

1 实验目的

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 的 10 个手写数字的分类，公开数据集可以在网站中找到，具体的实验框架也已经给出，需要根据代码框架理解训练流程并填充必要的代码完成对于手写体的识别工作。

2 实验要求

你的任务是在 softmax_regression.py 文件中实现 softmax_regression() 函数，计算每一次迭代的损失值 J (, x, y)，将它存储在变量 f 中，并计算梯度 J (, x, y)，将它存储在变量 g 中。初始代码会将 的形状定义为一个 k×n 的矩阵 (K=10 个类别)。

在本次实验中，需要我们根据课上对于 softmax 逻辑回归模型的理解自行编写梯度下降算法更新判别矩阵，不能使用 python 编写好的库函数。

3 理论知识

3.1 softmax 回归

softmax 回归其实可以理解为扩展的逻辑回归，是逻辑回归在多分类问题上的推广，Softmax 回归跟线性回归一样将输入特征与权重做线性叠加。与线性回归的一个主要不同在于，Softmax 回归的输出值个数等于标签里的类别数。也就是说，当机器学习结束后，并不会马上告诉你我的选择是哪一个，而是将所有的可能都罗列出一个值（即图中的 o1、o2、o3），等待进行下一步的计算。

$$
\begin{aligned}
o_1 &= x_1 w_{11} + x_2 w_{21} + x_3 w_{31} + x_4 w_{41} + b_1, \\
o_2 &= x_1 w_{12} + x_2 w_{22} + x_3 w_{32} + x_4 w_{42} + b_2, \\
o_3 &= x_1 w_{13} + x_2 w_{23} + x_3 w_{33} + x_4 w_{43} + b_3.
\end{aligned}
$$

图 3.1: softmax 输出计算

下一步的计算才是 softmax 真正的核心，使用 softmax 函数将多个神经元的输出重新映射到（0,1）的区间内，也就是说不管前面神经元返回的值是多少，我都能重新对它们进行映射，而这个映射后的值具有相当良好的分类性质，由于映射后的值在（0,1）之间取值，所以可以把这个值理解为不同类别的概率，并且不同类别的概率和相加还一定等于 1

$$
\hat{y}_1, \hat{y}_2, \hat{y}_3 = \mathrm{softmax}(o_1, o_2, o_3),
$$

图 3.2: softmax 转换

$$
\hat{y}_1 = \frac{\exp(o_1)}{\sum_{i=1}^{3} \exp(o_i)}, \quad \hat{y}_2 = \frac{\exp(o_2)}{\sum_{i=1}^{3} \exp(o_i)}, \quad \hat{y}_3 = \frac{\exp(o_3)}{\sum_{i=1}^{3} \exp(o_i)}.
$$

图 3.3: softmax 计算公式

3.2 损失计算

在 softmax 回归中，使用交叉熵损失函数来衡量模型与 ground true 的差异

$$\mathrm{L}(\boldsymbol{W},\boldsymbol{b})=-\frac{1}{\mathrm{N}}\sum_{\mathrm{n}=1}^{\mathrm{N}}\sum_{\mathrm{c}=1}^{\mathrm{C}}\mathrm{y}_{\mathrm{c}}^{(\mathrm{n})}\log\hat{\mathrm{y}}_{\mathrm{c}}^{(\mathrm{n})}=-\frac{1}{\mathrm{N}}\sum_{\mathrm{n}=1}^{\mathrm{N}}(\mathrm{y}^{(\mathrm{n})})^{\mathrm{T}}\log\hat{\mathrm{y}}^{(\mathrm{n})}$$

图 3.4: 交叉熵损失函数

3.3 梯度下降

为了最小化损失函数，让模型快速收敛，我们使用梯度下降的方式找到损失函数下降最快的方式进行模型调整。

$$\begin{aligned}
\frac{\partial L(\theta)}{\partial\theta_j} &= -\frac{1}{m}\frac{\partial}{\partial\theta_j}\left[\sum_{i=1}^{m}\sum_{j=1}^{k}1\left\{y_i=j\right\}\log\frac{e^{\theta_j^T x_i}}{\sum_{l=1}^{k}e^{\theta_l^T x_i}}\right] \\
&= -\frac{1}{m}\frac{\partial}{\partial\theta_j}\left[\sum_{i=1}^{m}\sum_{j=1}^{k}1\left\{y_i=j\right\}\left(\theta_j^T x_i-\log\sum_{l=1}^{k}e^{\theta_l^T x_i}\right)\right] \\
&= -\frac{1}{m}\left[\sum_{i=1}^{m}1\left\{y_i=j\right\}\left(x_i-\sum_{j=1}^{k}\frac{e^{\theta_j^T x_i}\cdot x_i}{\sum_{l=1}^{k}e^{\theta_l^T x_i}}\right)\right] \\
&= -\frac{1}{m}\left[\sum_{i=1}^{m}x_i 1\left\{y_i=j\right\}\left(1-\sum_{j=1}^{k}\frac{e^{\theta_j^T x_i}}{\sum_{l=1}^{k}e^{\theta_l^T x_i}}\right)\right] \\
&= -\frac{1}{m}\left[\sum_{i=1}^{m}x_i\left(1\left\{y_i=j\right\}-\sum_{j=1}^{k}1\left\{y_i=j\right\}\frac{e^{\theta_j^T x_i}}{\sum_{l=1}^{k}e^{\theta_l^T x_i}}\right)\right] \\
&= -\frac{1}{m}\left[\sum_{i=1}^{m}x_i\left(1\left\{y_i=j\right\}-\frac{e^{\theta_j^T x_i}}{\sum_{l=1}^{k}e^{\theta_l^T x_i}}\right)\right] \\
&= -\frac{1}{m}\left[\sum_{i=1}^{m}x_i\left(1\left\{y_i=j\right\}-p\left(y_i=j|x_i;\theta\right)\right)\right]
\end{aligned}$$

图 3.5: 梯度下降

为了在 python 语言中简单的描述梯度下降，将 3.5 最后的计算结果表示为矩阵形式，这样就可以直接使用函数传递的参数进行梯度下降算法的编写。而在此之前，我们还需要了解一个为了防止模型过拟合而需要添加的变量–正则项。

3.4 正则项

为了防止模型过拟合，我们需要添加正则项来辅助模型的构建，即结构误差函数，也称为正则项，正则化参数的同时，最小化经验误差函数，最小化经验误差是为了极大程度的拟合训练数据，正则化参数是为了防止过分的拟合训练数据，因此对系数进行一定的惩罚。

L1 范式是比较常见的正则项范式：L1 正则化之所以可以防止过拟合，是因为 L1 范数就是各个参数的绝对值相加得到的，参数值大小和模型复杂度是成正比的。因此如果拟合出一个复杂的模型（即出现了过拟合），其 L1 范数就大，这样 L1 正则化惩罚就高，整体损失函数就没有收敛，

$$\lambda\sum_{j=1}^{n}||\theta_j||$$

图 3.6: L1 范式

到此为止，我们终于可以综合前面的结果写出 python 程序进行梯度下降，其形式化表达可以写成:

$$\frac{\partial L(\theta)}{\partial \theta} = -\frac{1}{m}(y-P)^T X + \lambda\theta$$

图 3.7: 梯度下降

根据课上的知识以及查阅资料，我们对 softmax 整体的流程以及实现方法已经有了清晰的认识，下面我们将进行 python 代码的编写

4 python 实现

4.1 梯度下降计算

梯度下降计算的 python 实现是本次实验的核心，我们在这里贴出完整的实验代码，具体的计算含义已经使用注释标注出，可以对照查看理论知识的内容即可清晰：

```
1
2  # coding=utf-8
3   from concurrent.futures import thread
4   from random import shuffle
5   import numpy as np
6
7
8   def softmax_regression(theta, x, y, iters, alpha):
9       # TODO: Do the softmax regression by computing the gradient and
10      # the objective function value of every iteration and update the theta
11      m = len(x)
12      f = list()
13      lam=0.001   # 正则项
14      data=x.T   # 矩阵转置
15      print(theta.shape)
16
```

```
    for i in range(iters):
        out=np.dot(theta,data) # 做一次矩阵乘法

        output_exp = np.exp(out)
        exp_sum = output_exp.sum(axis=0) # 对矩阵做 e 的运算
        y_hat = (output_exp / exp_sum)

        batch_size = y.shape[1]
        y_hatlog = np.log(y_hat) # 把 e 取下来
        l_sum = 0.0
        for i in range(batch_size):
            l_sum += np.dot(y_hatlog[:, i].T, y[:, i])
        train_loss = -(1.0 / batch_size) * l_sum      # 计算损失函数

        print(train_loss)     # 输出损失函数
        f.append(train_loss)

        batch_size = y.shape[1]
        g=-(1.0 / batch_size) * np.dot((y - y_hat), data.T)+lam*theta    # 计算梯度
        theta = theta - alpha * g     # 梯度下降

    return theta

    }
```

4.2 准确度计算

同样，在训练完成后我们需要在测试集上进行计算，计算方式为标签的对比，我们只需要进行统计，将一致的个数记录下来即可：

4.3 代码

```

    def cal_accuracy(y_pred, y):
    # TODO: Compute the accuracy among the test set and store it in acc
    correct = 0.0 # 有多少预测正确
    for i in range(len(y)):
        if y_pred[i] == int(y[i]): # 如果预测正确
            correct += 1
    acc = correct / len(y)
    return acc
```

10

5 实验总结

5.1 实验效果

经过参数调整以及测试，识别的正确度可以达到 91.5%

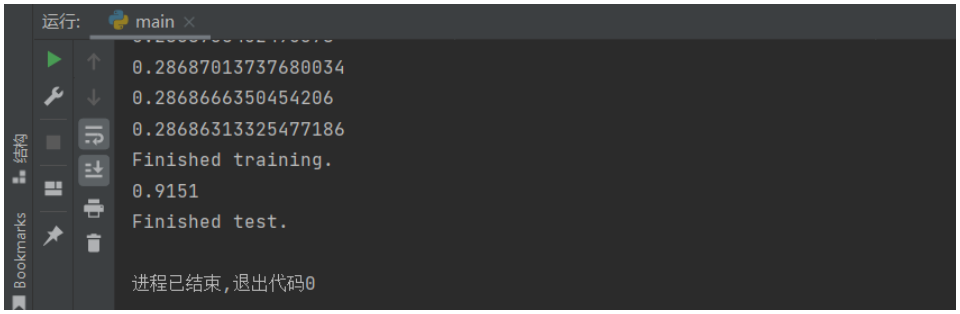

图 5.8: 正确率

5.2 结果分析

在本次实验中，我们使用 matolib 对所得到的数据进行绘图，可以清楚的看到不管是从损失值的大小还是从预测准确度大小的角度来说，softmax 回归都在相当程度上优秀的完成了手写体识别分类的任务

为了绘制损失图的变化曲线，我们每次将算得的损失值压入一个 list 中并借助 matolib 进行绘制

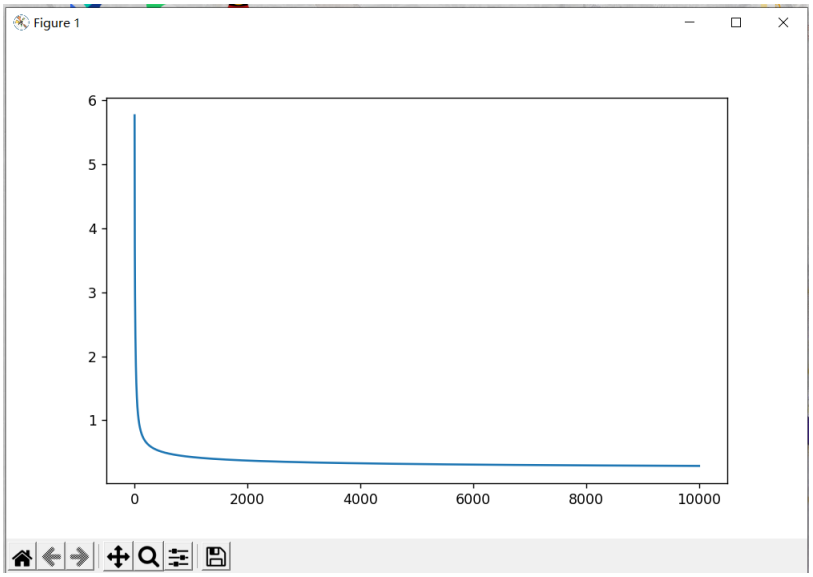

图 5.9: 损失图

同样的，为了绘制正确率变化曲线，我们将每次得到的矩阵都应用于一次手写体识别并观察结果。

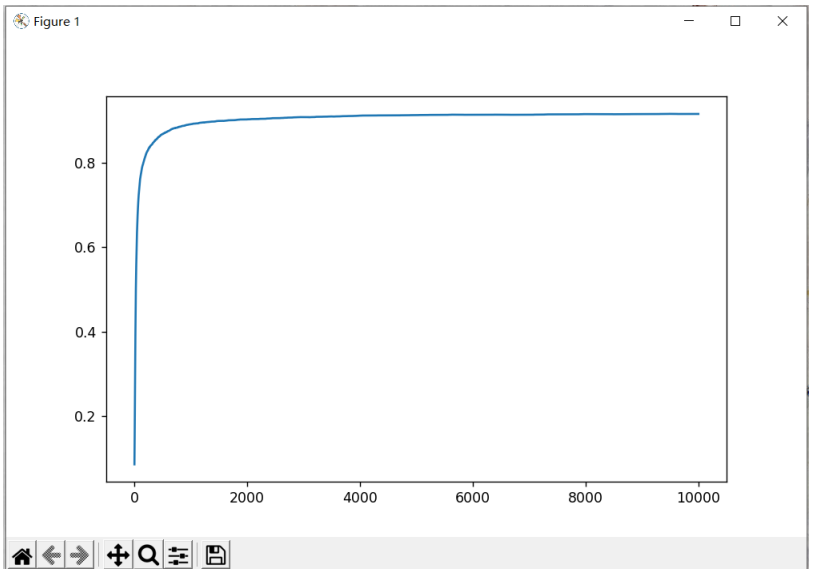

图 5.10: 正确率图

在实验分析中，如果在训练集上的损失函数下降，在测试集上的正确率就会提高，不会产生过拟合的现象，这是因为我们添加了正则项，所以可以在可能的时间范围内尽可能调大迭代次数以获得更好的实验效果。

当然，本模型相对简单，未来的改进方向可以是将数据集打散并分批次训练，这样可以充分保证模型训练的随机性以达到更好的训练效果，

5.3 softmax 分析

5.3.1 优势分析

softmax 回归在多分类模型算法中具有很多很良好的性质，比如说相同输出特征情况，soft max 更容易达到终极目标 one-hot 形式，或者说，softmax 降低了训练难度，使得多分类问题更容易收敛。

5.3.2 改进方向

同样 Softmax 鼓励真实目标类别输出比其他类别要大，但并不要求大很多。这对于日常生活中的元素分类来说是比较贴合的，不同的物体可能相差不大，那么在相对小的区别内区分多种不同的种类就变得困难，softmax 函数则可以解决这个问题。

当然，softmax 回归也存在一些问题，比如说到目前为止，仅仅使用 softmax 作为训练模型的算法已经越来越少了，现在越来越多的思路是将 softmax 函数与其他结合，构建 CNN 网络，最后使用 softmax 函数进行映射训练效果会大幅度提升。