南　开　大　学

网 络 空 间 安 全 学 院

机器学习第一次实验报告

Softmax 回归

许健

学号：2013018

专业：信息安全

指导教师：谢晋

2022 年 11 月 3 日

一、 Softmax 回归原理

有关 Softmax 回归的详细知识课上已经讲授，在此不作赘述。仅列出 Softmax 回归中较为关键的公式，也是后面程序编写的基础，程序所做的事情也不过是用语言把数学公式表达出来。

（一） Softmax 回归的决策函数

Softmax 回归对样本 i 分类的矢量计算表达式为

$$o^{(i)} = x^{(i)} W^T + b$$

$$\hat{y}^{(i)} = softmax(o^{(i)}) = \frac{exp(o_i)}{\sum_{i=1}^{k} exp(o_i)}$$

Softmax 回归的决策函数可以表示为

$$\hat{y} = argmax \ \ \hat{y}^{(i)}$$

（二） 交叉熵损失函数

采用交叉熵损失函数，Softmax 回归模型的损失函数为

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

其中 $\hat{y}^n = Softmax(x^n W^T + b)$ 为样本 $x^{(n)}$ 在每个类别的后验概率。

当每个样本的真实标签 $y^{(n)}$ 只有一个标签，即是一个 C 维 one-hot 向量：若样本属于 i 类，则只有第 i 维是 1，其余都是 0。所以

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

（三） 梯度

损失函数 $L(W)$ 关于 W 的梯度为

$$\frac{\partial \mathrm{L(W)}}{\partial \mathrm{W}} = -\frac{1}{\mathrm{N}}\sum_{\mathrm{n}=1}^{\mathrm{N}} \mathrm{x(n)}\left(\mathrm{y}^{(\mathrm{n})} - \hat{\mathrm{y}}^{(\mathrm{n})}\right)^{\mathrm{T}}$$

采用梯度下降法，Softmax 回归的训练过程为:

1）随机初始化 W_0

2) 迭代更新:

$$\mathrm{W_{t+1}} = \mathrm{W_t} + \alpha\left(\frac{1}{\mathrm{N}}\sum_{\mathrm{n}=1}^{\mathrm{N}} \mathrm{x(n)}\left(\mathrm{y}^{(\mathrm{n})} - \hat{\mathrm{y}}^{(\mathrm{n})}\right)^{\mathrm{T}}\right)$$

二、 程序编写

(一) Softmax_regression 函数

$softmax(scores)$ 使用 Python 的广播机制，计算 Softmax 函数。在 $softmax\_regression$ 中将调用该函数。

softmax 函数

```python
def softmax(scores):
    sum_exp = np.sum(np.exp(scores), axis=1, keepdims=True)
    softmax = np.exp(scores)/sum_exp
    return softmax
```

$softmax\_regression$ 主要过程包括:

1. 计算 $y^{(i)} = x^{(i)}W^{(T)}$
2. 计算损失 loss
3. 计算损失函数的导数
4. 在训练中增加正则化项，防止过拟合，设置 lam=0.01
5. 更新权重矩阵

程序中使用的公式均为对数学公式的翻译，涉及矩阵运算、求导的相关知识，请自行补充。主要的代码虽然只有十来行，但是需要对 Softmax 回归的整体理解，后续还包括对模型参数的调整，以得到最优的结果。

softmax_regression

```python
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
    # x:[m,n], y:[k,m], theta[k,n]
    n_samples, n_features = x.shape
    n_classes = y.shape[0]
    lam = 0.01 #正则项
    for i in range(iters):
        scores = np.dot(x, theta.T)
        probs = softmax(scores)
        loss = - (1.0 / n_samples) * np.sum(y.T * np.log(probs))
        print("第",i,"次loss:%f" % (loss))
        dw = -(1.0 / n_samples) * np.dot((y - probs.T), x) + lam * theta
        # 更新权重矩阵
        theta = theta - alpha * dw

    return theta
```

（二） cal_accuracy

将训练得到的 y_yred 与真实样本标签进行比对，返回模型训练精确度。

cal_accuracy

```python
def cal_accuracy(y_pred, y):
    # TODO: Compute the accuracy among the test set and store it in acc
    acc = np.sum(y_pred == y.reshape(y_pred.shape[0])) / y_pred.shape[0]
    return acc
```

三、 实验结果分析

实验中使用的 iters 为 500，学习率 alpha 为 0.5，样本类别 k 为 10，正则化项 lam 为 0.01。实验环境为 CPU，Python3.6，Numpy 库。最后实验结果如图1所示，实验精度达到 0.905。

```
第 497 次loss:0.383789
第 498 次loss:0.383782
第 499 次loss:0.383774
Finished training.
Finished test.
0.905
```

图 1: 实验结果

实验中选择的迭代次数、学习率都较为合理，我尝试使用不同的组合，模型的精确度上限基本就是 0.905。

在本次实验中我们尝试增加了正则化项，但是对模型的精度并没有什么改变，因为我们的模型只有一层隐藏层，复杂度较低，相对来说不会出现过拟合现象。

四、 探究：梯度检查

使用调试策略进行梯度检查我并不是特别了解，查阅资料后知道，反向传播算法很难调试得到正确结果，尤其是当实现程序存在很多难于发现的 bug 时。梯度检查是一种对求导结果进行数值检验的方法，该方法可以验证求导代码是否正确。回顾求导公式的数学定义：

$$\frac{d}{d\theta}J = \lim_{\epsilon\to 0}\frac{J(\theta+\epsilon)-J(\theta-\epsilon)}{2\epsilon}$$

由此我们可得梯度校验的数值校验公式：

$$g(\theta) \approx \frac{J(\theta+\epsilon)-J(\theta-\epsilon)}{2\epsilon}$$

也就是使用导数的定义与求导公式得到的结果比对，如果误差在合理范围内则说明我们的求导公式是正确的。

实际应用中我们常将 ϵ 设为一个很小的常量，比如 10^{-4} 数量级，我们不会将它设得太小，那将导致数值舍入误差。在假定 $\epsilon = 10^{-4}$ 的情况下，通常会发现左右两端至少有四位有效数字是一致的（或者说精度至少在 0.0001 一级）。

由于本实验的求导公式确信没有问题，所以我并没有加上，仅以一段代码来演示 sigmoid 函数使用梯度检查的效果：

梯度检查

```
import numpy as np

def sigmoid(z):
    return 1./(1+np.exp(-z))
def sigmoid_prime(z):
    return sigmoid(z)*(1-sigmoid(z))
def check_gradient(f, x0, epsilon):
    return (f(x0+epsilon) - f(x0-epsilon))/2/epsilon

if __name__ == '__main__':
    x0 = np.array([1, 2, 3])
    epsilon = 1e-4
    print(sigmoid_prime(x0))
            # [ 0.19661193  0.10499359  0.04517666]
    print(check_gradient(sigmoid, x0, epsilon))
            # [ 0.19661193  0.10499359  0.04517666]
```