Exercise1 Softmax Regression

郭坤昌 2012522 计算机科学与技术

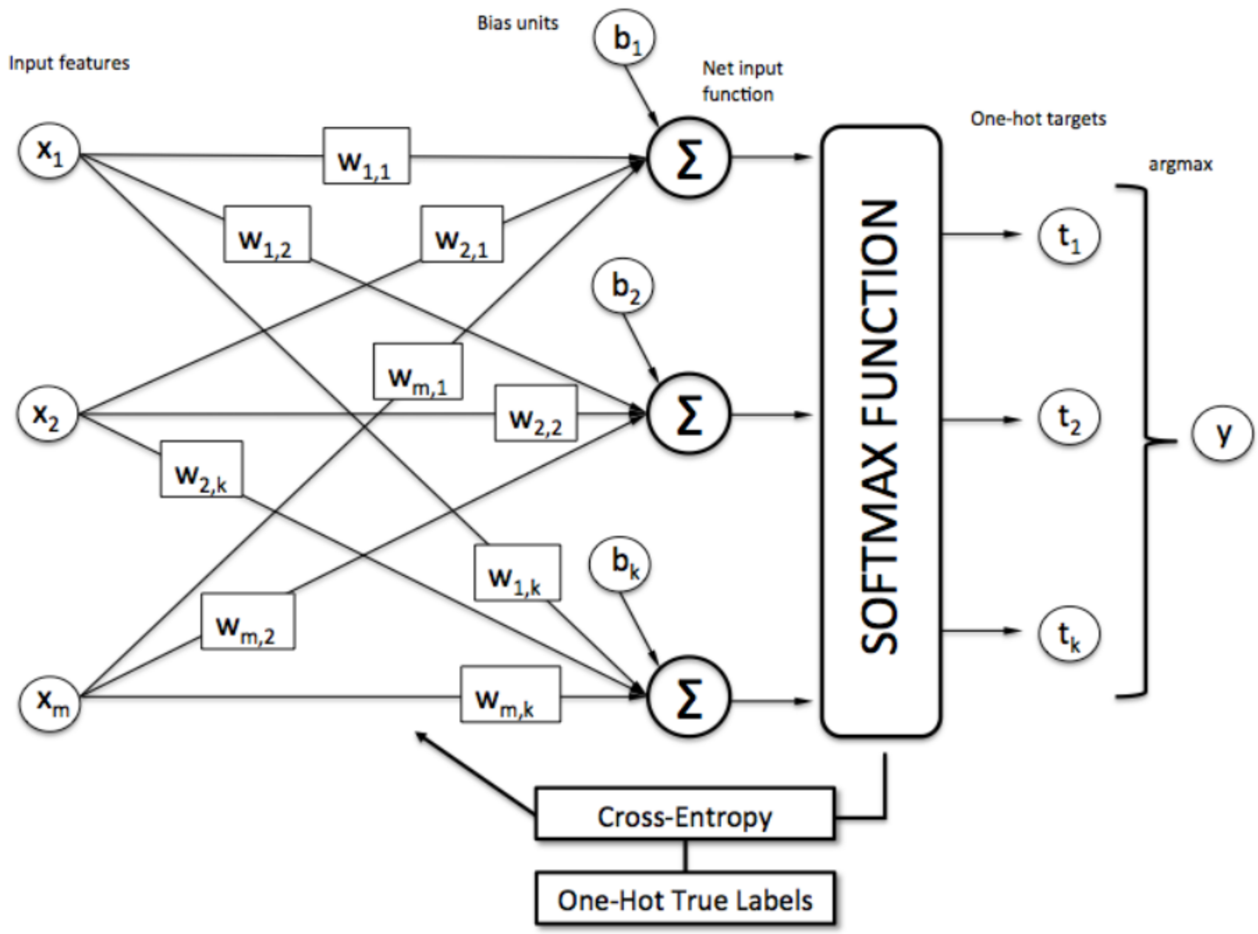
要求

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。

1. 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 (θ, x, y)，将它存储在变量 f 中，并计算梯度 ∇_θ J (θ, 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)
2. 在 evaluate.py文件中实现cal_accuracy()函数，输出分类器在测试集上的准确率

实验原理

用于多分类的`softmax regression`用于估计输入的x_i属于每一类的概率，原理图如下。对于输入特征，使用矩阵θ计算得分，再通过`softmax`函数激活，将得分归一到区间$[0, 1]$，即获得分类概率。



使用`softmax`函数激活后，得到x_i在每个类别上的分类概率。

$$h_\theta(x_i) = \begin{bmatrix} p(y_i = 1|x_i;\theta) \\ p(y_i = 2|x_i;\theta) \\ \vdots \\ p(y_i = k|x_i;\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{k} e^{\theta_j^T x_i}} \begin{bmatrix} e^{\theta_1^T x_i} \\ e^{\theta_2^T x_i} \\ \vdots \\ e^{\theta_k^T x_i} \end{bmatrix}$$

特别地，x_i被分类为j的概率为

$$p(y_i = j|x_i;\theta) = \frac{e^{\theta_j^T x_i}}{\sum_{l=1}^{k} e^{\theta_l^T x_i}}$$

`softmax`回归的代价函数为:

$$L(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{j=1}^{k} 1\{y_i = j\}\log\frac{e^{\theta_j^T x_i}}{\sum_{l=1}^{k} e^{\theta_l^T x_i}}\right]$$

其梯度求解为:

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

为了防止过拟合，引入正则项，此时代价函数和梯度为：

$$
L(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{j=1}^{k} 1\left\{y_i = j\right\}\log\frac{e^{\theta_j^T x_i}}{\sum_{l=1}^{k} e^{\theta_l^T x_i}}\right] + \lambda\sum_{i=1}^{k}\sum_{j=1}^{n}\theta_{ij}^2
$$

$$
\frac{\partial L(\theta)}{\partial \theta_j} = -\frac{1}{m}\left[\sum_{i=1}^{m} x_i\left(1\left\{y_i = j\right\} - p\left(y_i = j \mid x_i;\theta\right)\right)\right] + \lambda\theta_j
$$

实验过程

实验环境

python 3.8 with numpy 1.23.4

参数设置

```python
mnist_dir = "mnist_data/"   # dir of mnist dataset
train_data_dir = "train-images-idx3-ubyte"
train_label_dir = "train-labels-idx1-ubyte"
test_data_dir = "t10k-images-idx3-ubyte"
test_label_dir = "t10k-labels-idx1-ubyte"
k = 10  # classes
iters = 2000    # iterations to learn
alpha = 0.3 # learning rate
lam=0.005   # coefficient of regulation
```

数据集加载

在main函数中调用了加载数据集函数，从`mnist_dir`下读取，因此创建文件夹`mnist_data`，并放入数据文件。在`load_data`中进一步调用了`load_mnist`，而由于开始传入的文件名其实是对应文件夹的名称，为了简便，修改`load_mnist`函数为从该数据集对应文件夹下读取唯一的数据，修改部分代码如下，其余未改变部分用省略号略过。

```python
def load_mnist(file_dir, is_images='True'):
    bin_file_name=os.listdir(file_dir)[0]   # get the only data name from current directory
    bin_file=open(os.path.join(file_dir, bin_file_name), 'rb')
    # bin_file = open(file_dir, 'rb')    # old method to read the directory
    ……
```

读取成功，并能观察到对应数据的格式

```
In [12]: train_images, train_labels, test_images, test_labels = load_data(mnist_dir,
train_data_dir, train_label_dir, test_data_dir, test_label_dir)
Loading MNIST data from files...
Load images from mnist_data/train-images-idx3-ubyte, number: 60000, data shape: (60000, 784)
Load images from mnist_data/train-labels-idx1-ubyte, number: 60000, data shape: (60000, 1)
Load images from mnist_data/t10k-images-idx3-ubyte, number: 10000, data shape: (10000, 784)
Load images from mnist_data/t10k-labels-idx1-ubyte, number: 10000, data shape: (10000, 1)
```

`softmax`函数

`softmax`函数用以计算分类概率。输入应当是规模为$k \times m$的得分矩阵，通过对每一列（x_i在每一类上的得分）进行`softmax`归一化得到x_i在每一类上的分类概率。

```python
def softmax(z):
    z-=np.max(z)    # in case of overflow
    softmax=(np.exp(z) / np.sum(np.exp(z), axis=0))    # normalize by sum of each column
    return softmax
```

简单验证如下:

```
array_test
```
[27] 0.3s

```
array([[7, 4, 1],
       [2, 3, 4],
       [5, 2, 9],
       [6, 1, 7]])
```

原矩阵

```
softmax(array_test)
```
[28] 0.4s

使用softmax在各列上归一化

```
array([[6.62272414e-01, 6.43914260e-01, 2.93645024e-04],
       [4.46235642e-03, 2.36882818e-01, 5.89801797e-03],
       [8.96288247e-02, 8.71443187e-02, 8.75343479e-01],
       [2.43636405e-01, 3.20586033e-02, 1.18464858e-01]])
```

各列总和为1

```
softmax(array_test).argmax(axis=0)
```
[29] 0.4s

各列上分类最高的概率，如上框

```
array([0, 0, 2])
```

梯度求解

由之前计算的公式，引入正则项，梯度求解如下。为了观察训练过程，这里也定义了对应的代价函数。f和g分别对应要求的代价函数和梯度

```
def get_gradient(x, y, theta, lam):
    m = x.shape[0]
    score = np.dot(theta, x.T)
    softmax_score = softmax(score)
    f=-np.sum(y*np.log(softmax_score))/m+lam*np.sum(theta*theta)/2
    g = np.dot(softmax_score-y,x)/m+lam*theta
    return f, g
```

softmax回归

迭代，并显示每一次迭代过程中代价函数的变化。α对应为学习率。

```
def softmax_regression(theta, x, y, iters, alpha, lam):
    losses=[]
    for iter in range(iters):
        f, g=get_gradient(x,y,theta,lam)
        theta-=alpha*g
        losses.append(f)
    plt.plot(losses)
    plt.show()
    return theta
```

预测和准确率计算

由于θ为$k \times n$矩阵，输入为$m \times n$矩阵，通过矩阵相乘计算测试图片在各类别上的分类概率。

```
def predict(test_images, theta):
    # scores = np.dot(test_images, theta.T)
    # preds = np.argmax(scores, axis=1)
    scores = np.dot(theta, test_images.T)
    preds = np.argmax(scores, axis=0)
    return preds
```

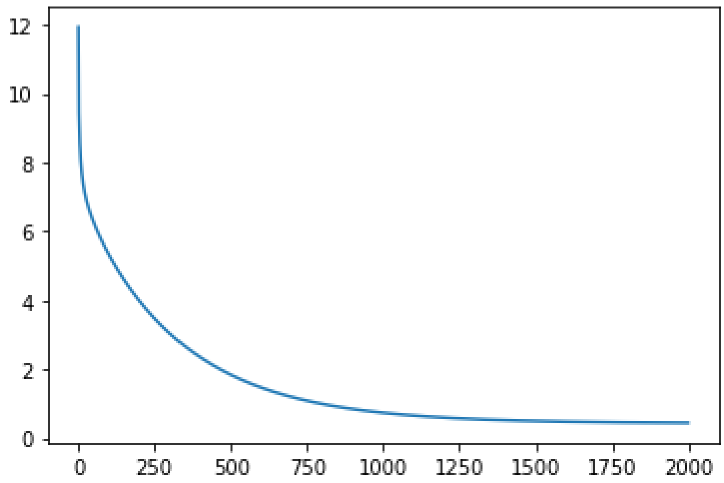
通过`for`循环直接统计预测正确的数量，并除以总数得到准确率。

```
def cal_accuracy(y_pred, y):
    acc=0
    for i in range(y.shape[0]):
        if y_pred[i] == y[i]:
            acc += 1
    return acc/y.shape[0]
```

实验结果和分析

在迭代次数为2000，学习率为0.3，正则项系数为0.005时进行实验。

损失函数曲线如下。1250轮时已基本不再下降，θ中的每个分类器通过不断训练最终趋近于$\vec{0}$



预测准确率为0.9122

accuracy	float	1	0.9122
alpha	float	1	0.3
iters	int	1	2000
k	int	1	10
lam	float	1	0.005

在调整超参数过程中，由如下结论：

1. 学习率不宜过高，也不宜过低。过高使得代价函数下降时震荡明显，不易找到最低点。过低则受正则项影响较大，下降困难。
2. 正则项一定程序防止过拟合，能提高模型分类效果，但其系数不宜过大，否则下降困难。