Exercise1-Softmax Regression

> 1911533
>
> 朱昱函

实验要求

- 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。
- 初始代码可以在课堂提供的压缩包的 ex1/ 中找到。
- MNIST数据集可在数据集下载地址中下载，共有4个.gz文件，解压后放在初始代码的 ex1/ 目录下。
- 初始代码中已经包含以下内容：
  - 在 main.py文件中加载训练和测试数据集
  - 在 main.py文件中调用训练和测试相关函数
  - 在 train.py文件中调用 softmax_regression() 函数
- 当训练完成，分类器会输出10个类别的测试准确率。
- 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 $J(\theta, x, y)$，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。
- 一开始你可以使用 for循环 来得到正确的梯度 (最好尽早使用调试策略来进行梯度检查)。之后你会发现用 for循环 实现的代码计算速度太慢了。当你得到正确的梯度之后，尽可能尝试对你的代码进行优化，再用完整的数据集来跑实验。
- 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。
- 实验环境：硬件环境：CPU。软件环境：Python 3.7，Numpy。
- 作业以压缩包方式提交，压缩包命名以本次作业为例，命名为*"第一次作业-学号-姓名"*。

原理分析与关键代码说明

softmax函数

由于y有k(本实验中k=10)个类别，故变成一个k维的向量，来表示类别估计的概率值, 并归一化（概率和为1）

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

```
def softmax(x):
    [m,k]= x.shape
    p = np.zeros([m,k])
    for i in range(m):
        p[i,:]=np.exp(x[i,:])/np.sum(np.exp(x[i,:]))
    return p
```

损失函数

将logistic回归的损失函数运用的0-1规划思想（2-dimension）扩展为k-dimension

以下为**logistic**函数的损失函数计算

$$\mathrm{J}(\Phi(\mathrm{z}), \mathrm{y}; \mathrm{w}) = \begin{cases} -\log(\Phi(\mathrm{z})) & 若\mathrm{y}=1 \\ -\log(1-\Phi(\mathrm{z})) & 若\mathrm{y}=0 \end{cases}$$

$$\mathrm{J}(\mathrm{w}) = \log(\mathrm{L}(\mathrm{w})) = \sum_{\mathrm{i}=1}^{\mathrm{n}} -\mathrm{y}^{(\mathrm{i})} \log\left(\phi\left(\mathrm{z}^{(\mathrm{i})}\right)\right) - \left(1-\mathrm{y}^{(\mathrm{i})}\right) \log\left(1-\phi\left(\mathrm{z}^{(\mathrm{i})}\right)\right)$$

扩展到k维，得到softmax损失函数为

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

```
def cal_loss(theta, x,y):  #x(m*n)   y (m*k)   theta (k*n)
    [k,m]=y.shape
    theta = np.matrix(theta)
    sum = 0
    p = softmax(np.dot(x,theta.T))
    p = p.T.reshape([k * m,1])
    y = y.reshape([k * m,1])
    temp_p=np.mat(np.log(p))
    loss = -1/m*np.dot(y.T,temp_p)
    return loss
```

利用批量梯度下降更新 θ

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

梯度为

$$\nabla_{\theta_j} J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[x^{(i)}\left(1\{y^{(i)}=j\} - p(y^{(i)}=j|x^{(i)};\theta)\right)\right]$$

```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
    [m] = y[0].shape
    for i in range(iters):
        p = softmax(np.dot(x,theta.T));
        grad = (-1/m *  np.dot(x.T ,(y.T-p)) ).T
        theta = theta - alpha * grad
        loss = cal_loss(theta,x,y)
        if i % 10 == 0:
            print("train iters: ",i)
            print(" loss: ", loss)
            print("\n")
    return theta
```

计算准确率

将测试得到的实际矩阵的分类结果和预测进行比较，每相同一次count+1，最后count和总数相除即为准确率

```python
def cal_accuracy(y_pred, y):
    num = 0
    # TODO: Compute the accuracy among the test set and store it in acc
    for i in range(len(y)):
        if (y_pred[i]==y[i]):
            num += 1
    acc = num/len(y)
    print("accuracy: ",acc)
    return acc
```

运行结果

- iters = 1000
- alpha = 0.75

```
Finished training.
accuracy:  0.9125
Finished test.
```

结果分析

- 实现了较好的分类效果
- 迭代次数与准确率成正相关