

多分类

请使用jupyter完成相关代码的运行。若未安装该依赖，直接阅读该 pdf 文件同样可以获取练习结果。

算法步骤

多分类问题 —— BP算法步骤

(交叉熵代价函数 + Softmax激活函数)

- S1: 初始化权值系数
- S2: 提取一个训练样本 $\{x, d\}$, 计算输出向量与期望输出之差
以及输出节点的delta向量

$$\begin{aligned} e &= d - y \\ \delta &= e \end{aligned}$$

- S3: 向后传播输出误差, 计算上一层节点的delta向量

$$e^{(k)} = W^T \delta \quad \delta^{(k)} = \varphi'(\nu^{(k)}) e^{(k)}$$

- S4: 重复S3至抵达第一隐含层
- S5: 调整各层权值系数
- S6: 重复S2-5遍历每个训练样本
- S7: 重复S2-6直到输出误差符合预期



准备工作

导入依赖。

```
import numpy as np
import matplotlib.pyplot as plt
```

Softmax 函数

g 代表一个常用的逻辑函数 (logistic function) 为S形函数 (Softmax function) , 公式为: $\frac{e^{v_i}}{\sum\{e^{\{v_j\}}\}}$

```
def softmax(z):
    """
    softmax 函数
    Args:
        z (m, n): 输入
    Returns:
        g (m, n): softmax 函数输出
    """
    d = np.exp(z)
    # 注意 d sum时的axis
    return d / d.sum(axis = 1).reshape(-1, 1)
```

Softmax函数的梯度的函数

```
def softmax_gradient(z):
    return np.multiply(softmax(z), (1 - softmax(z)))
```

前向传播函数

单隐层网络，注意偏置 $b = 0$ 。

```
def forward_propagate(X, theta1, theta2):
    ...
    前向传播函数

    Args:
        X (m, n): m个样本, n个特征
        theta1 (t, n): t个神经元
        theta2 (k, t): k个输出

    Returns:
        a1 (m, n)
        z2 (m, t)
        a2 (m, t)
        z3 (m, k)
        h (m, k)
    ...

    m = X.shape[0]

    # 激活项a, 线性输出z
    a1 = X
    z2 = a1 * theta1.T
    a2 = softmax(z2)
    z3 = a2 * theta2.T
    h = softmax(z3)

    return a1, z2, a2, z3, h
```

代价函数

交叉熵代价函数。

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m \sum_{k=1}^K \left[-y_k^{(i)} \log((h_\theta(x^{(i)}))_k) - (1 - y_k^{(i)}) \log(1 - (h_\theta(x^{(i)}))_k) \right],$$

```
def cost_fcn(params, input_size, hidden_size, num_labels, X, y):  
    '''
```

交叉熵代价函数

Args:

params (hidden_size * input_size + num_labels * hidden_size,) : 参数
input_size (int) : 输入的特征数

hidden_size (int) : 隐藏层的神经元数

num_labels (int) : 输出层神经元数 / 类别数

X (m, n) : m个样本, n个特征

y (m, k) : 真实值, k个类别

Returns:

cost (int) : 代价

'''

m = X.shape[0]

X = np.matrix(X)

Y = np.matrix(Y)

```
# reshape the parameter array into parameter matrices for each layer
```

```

theta1 = np.matrix(np.reshape(params[:hidden_size * input_size], (hidden_size, input_size)))
theta2 = np.matrix(np.reshape(params[hidden_size * input_size:], (num_labels, hidden_size)))

# run the feed-forward pass
a1, z2, a2, z3, h = forward_propagate(X, theta1, theta2)

# compute the cost
J = np.multiply(-y, np.log(h)) - np.multiply((1 - y), np.log(1 - h))
cost = J.sum() / m

return cost

```

BP算法

交叉熵代价函数 + Softmax。

```

def backprop(params, input_size, hidden_size, num_labels, X, Y):
    ...

```

执行反向传播并返回代价和梯度

Args:

`params` (`hidden_size * input_size + num_labels * hidden_size,`) : 参数
`input_size` (`int`) : 输入的特征数
`hidden_size` (`int`) : 隐藏层的神经元数
`num_labels` (`int`) : 输出层神经元数 / 类别数
`X` (`m, n`) : `m`个样本, `n`个特征
`y` (`m, k`) : 真实值, `k`个类别

Returns:

```
    cost (int) : 代价  
    grad (hidden_size * input_size + num_labels * hidden_size, ) : 梯度  
    ...
```

```
m = X.shape[0]
```

```
# 将参数数组重构为每一层的参数矩阵
```

```
theta1 = np.matrix(np.reshape(params[:hidden_size * input_size], (hidden_size, input_size)))  
theta2 = np.matrix(np.reshape(params[hidden_size * input_size:], (num_labels, hidden_size)))
```

```
# 前向传播
```

```
a1, z2, a2, z3, h = forward_propagate(X, theta1, theta2)
```

```
# 计算损失
```

```
J = np.multiply(-y, np.log(h)) - np.multiply((1 - y), np.log(1 - h))  
cost = J.sum() / m
```

```
# 误差反向传播
```

```
error3 = h - y  
delta3 = error3  
error2 = delta3 @ theta2  
delta2 = np.multiply(error2, softmax_gradient(z2))  
# 计算梯度  
grad2 = delta3.T @ a2 / m  
grad1 = delta2.T @ a1 / m
```

```
# 将梯度矩阵分解成单个数组  
grad = np.concatenate((np.ravel(grad1), np.ravel(grad2)))
```

```
return cost, grad
```

训练函数。

```
def train(X, y, params, input_size, hidden_size, num_labels, alpha, iters, step = -1):  
    '''  
  
    Args:  
        X (m, n): m个样本, n个特征  
        y (m, k): 真实值, k个类别  
        params (hidden_size * input_size + num_labels * hidden_size, ): 参数  
        input_size (int): 输入的特征数  
        hidden_size (int): 隐藏层的神经元数  
        num_labels (int): 输出层神经元数/类别数  
        alpha (int): 学习率  
        iters (int): 最大迭代次数  
        step (int): 每组样本的数量  
  
    Returns:  
        g (1, n): 参数最终值  
        cost(iters, 1): 代价函数历史值  
    '''  
  
    m = X.shape[0]  
    g = params.copy()  
    cost = np.matrix(np.zeros((iters, 1)))
```

```
# 默认认为批量
if step == -1:
    step = m

for i in range(iters):
    for j in range(0, m, step):
        xs = X[j:j+step, :]
        ys = y[j:j+step]

        # 计算梯度和损失
        J, grad = backprop(g, input_size, hidden_size, num_labels, x, y)

        g = g - alpha * grad
        cost[i, 0] = J

    return g, cost
```

作业9

设计多分类问题的网络结构。

输入层节点数: 25 隐层节点数: 50 输出层节点数: 5

```
# 初始化设置
```

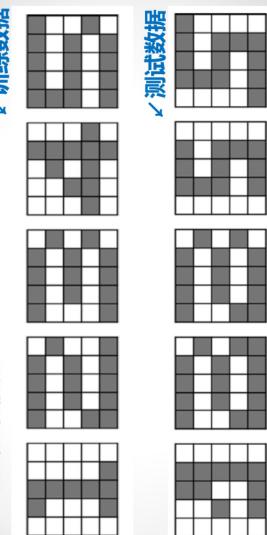
```
input_size = 25  
hidden_size = 50  
num_labels = 5  
alpha = 0.9  
epoch = 2000
```

作业10

用训练数据训练网络，用测试数据测试训练结果(注:运行多次观察结果是否变化，思考原因)

多分类问题实例

■ 测试泛化能力



↙训练数据

↙测试数据

```
# 数据  
X = np.array([[0, 1, 1, 0, 0],  
              [0, 0, 1, 0, 0],  
              [0, 0, 1, 0, 0],  
              [0, 1, 1, 0, 0],  
              [[1, 1, 1, 1, 0],
```

```
testdata = np.array([[ [0, 0, 0, 1],  
[0, 1, 1, 0],  
[1, 0, 0, 0],  
[1, 1, 1, 1] ],  
  
[[1, 1, 1, 0],  
[0, 0, 1, 1],  
[0, 1, 1, 0],  
[0, 0, 0, 1],  
[1, 1, 1, 0],  
  
[[0, 0, 1, 0],  
[0, 1, 1, 0],  
[0, 1, 1, 0],  
[1, 1, 1, 1],  
[0, 0, 1, 0],  
  
[[1, 1, 1, 1],  
[1, 0, 0, 0],  
[1, 1, 1, 0],  
[0, 0, 0, 1],  
[1, 1, 1, 0]])
```

```
[0, 1, 1, 1, 0],  
[[1, 1, 1, 1, 0],  
 [0, 0, 0, 1],  
 [0, 1, 1, 0],  
 [1, 0, 0, 1],  
 [1, 1, 1, 1],  
  
[[1, 1, 1, 1, 0],  
 [0, 0, 0, 1],  
 [0, 1, 1, 0],  
 [1, 0, 0, 1],  
 [1, 1, 1, 0],  
  
[[0, 1, 1, 1, 0],  
 [0, 1, 0, 0, 0],  
 [0, 1, 1, 1, 0],  
 [0, 0, 0, 1, 0],  
 [0, 1, 1, 1, 0],  
  
[[0, 1, 1, 1, 1],  
 [0, 1, 0, 0, 0],  
 [0, 1, 1, 1, 0],  
 [0, 0, 0, 1, 0],  
 [0, 1, 1, 1, 0]])
```

Y = np.eye(5)

```

X = X.transpose(0, 2, 1).reshape(5, 25)
m = X.shape[0]
X = np.matrix(X)
Y = np.matrix(Y)

# 随机初始化完整网络参数大小的参数数组
params = np.random.random(size = hidden_size * input_size + num_labels * hidden_size) * 2 - 1

# 训练
theta, cost = train(X, Y, params, input_size, hidden_size, num_labels, alpha, epoch)

# 观察输出
testdata = testdata.transpose(0, 2, 1).reshape(5, 25)
testdata = np.matrix(testdata)

ytest = np.eye(5)
ytest[3,3] = 0
ytest[3,4] = 1

theta1 = np.reshape(theta[:hidden_size * input_size], (hidden_size, input_size))
theta2 = np.reshape(theta[hidden_size * input_size:], (num_labels, hidden_size))

print(f'''训练数据网络预测值: \n{forward_propagate(X, theta1, theta2)[-1].argmax(axis=1)+1}
训练数据损失: {cost[-1]}''')

测试数据网络预测值: \n{forward_propagate(testdata, theta1, theta2)[-1].argmax(axis=1)+1}
测试数据损失: {costFcn(theta1, input_size, hidden_size, num_labels, testdata, ytest)}'''')
# 绘制曲线

```

```
fig, ax = plt.subplots()
t = np.linspace(1, epoch, epoch)
ax.plot(t, m * cost) # 创建t的取值范围
# 作误差曲线

ax.set_xlabel('epoch')
ax.set_ylabel('error')
ax.set_title('epoch-error curve')

plt.show()
```

训练数据网络预测值：

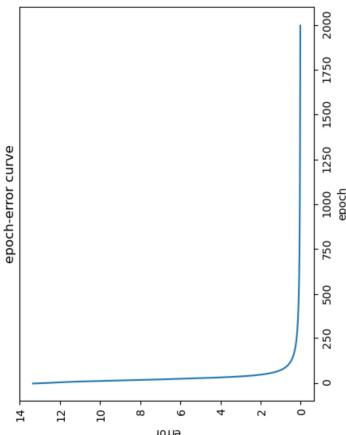
```
[1]
[2]
[3]
[4]
[5]
```

训练数据损失： [[0.00454072]]

测试数据网络预测值：

```
[1]
[2]
[3]
[3]
[5]
```

测试数据损失： 3.8129727838821252



为便于观察，使用`argmax`函数展现测试数据的预测结果，多次测试结果多次出现变化。

原因：需要更加丰富的特征(5×5 过少)和更大的训练集。

加上偏置。

作业11

尝试构造其它测试数据测试网络。

```
testdata = np.array([[0, 0, 0, 0, 0],  
[0, 0, 1, 0, 0],  
[0, 0, 1, 0, 0],  
[0, 0, 1, 0, 0],  
[1, 1, 1, 1, 1],  
[[0, 0, 1, 1, 1],
```

```
[[0, 0, 1, 1, 1],
```

```
[0, 0, 0, 0, 1],  
[1, 1, 1, 1, 1],  
[1, 0, 0, 0, 0],  
[1, 1, 0, 0, 0],  
,
```

```
[[1, 1, 1, 1, 1],  
 [0, 0, 0, 0, 1],  
 [1, 1, 1, 1, 1],  
 [0, 0, 0, 0, 1],  
 [1, 1, 1, 1, 1],  
,
```

```
[[0, 0, 0, 1, 0],  
 [0, 0, 1, 1, 0],  
 [0, 1, 1, 1, 0],  
 [1, 1, 1, 1, 1],  
 [0, 0, 1, 0, 1],  
,
```

```
[[1, 1, 1, 1, 1],  
 [1, 0, 0, 0, 0],  
 [1, 1, 1, 1, 1],  
 [0, 0, 0, 1, 1],  
 [1, 1, 1, 1, 1]])
```

```
testdata = testdata.transpose(0, 2, 1).reshape(5, 25)  
testdata = np.matrix(testdata)  
ytest = np.eye(5)  
print('测试数据网络预测值: \n', forward_propagate(testdata, theta1, theta2)[-1].argmax(axis=1) + 1)
```

```
测试数据损失: {cost_fcn(theta, input_size, hidden_size, num_labels, testdata, ytest) }''' )
```

测试数据网络预测值:

```
[ [1]  
[2]  
[5]  
[4]  
[5] ]
```

测试数据损失: 0.4228558371104397