中山大学数据科学与计算机学院
移动信息工程专业-人工智能
本科生实验报告

（2017-2018 学年秋季学期）

课程名称：**Artificial Intelligence**

教学班级	周四下午 7-8 节	专业（方向）	互联网
学号	15352010	姓名	蔡烨

一、 实验题目

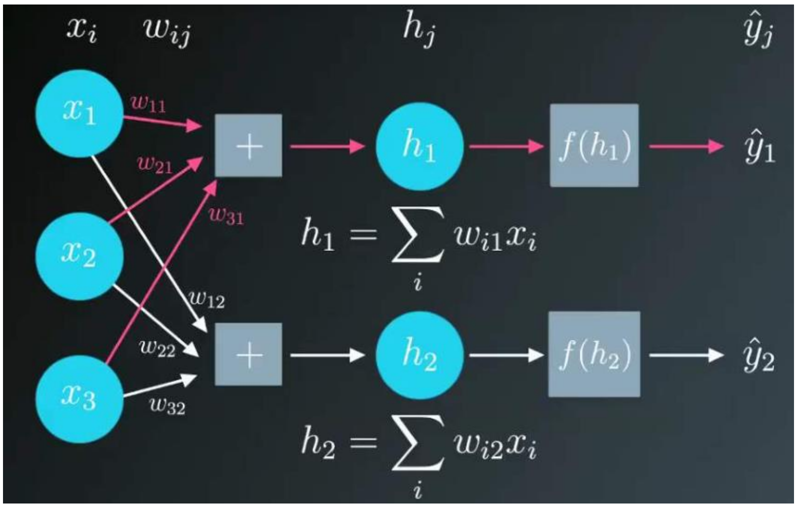
反向传播神经网络

二、 实验内容

1. 算法原理

BPNN：反向传播神经网络（以三层神经网络为例）

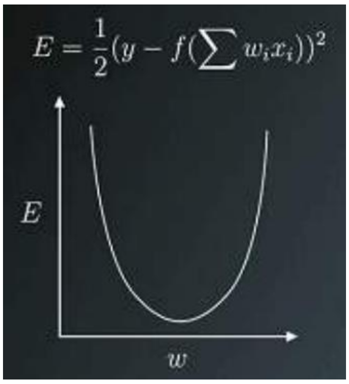
- 前向传输



• 输入层的每个节点x_i代表数据样本一个属性，对其加权求和后，作为隐藏层的输入 h。
• 将 h 作用到隐藏层中的激活函数 f(h)上，得到隐藏层的输出。
•隐藏层每个节点的输出 f(h)都作为输出层的输入，同样经历激活函数后（此处为 f(x)=x），得到输出值 y'

- 反向传播

经过神经网络得到的输出值 y'，跟真正的值 y 之间会存在误差$E = \frac{1}{2}(y - y')^2$，其中$y' = f(\sum w_i x_i)$.以 w 为自变量时，函数 E 是一个开口向上的抛物线。显然，E 越小，则神经网络的输出与真实值之间的误差越小，因此，通过对 w 进行梯度下降调整，来达到较小误差。



w_i的调整公式为：$w_i = w_i + \Delta w_i$，为了让 E 最小，w 的变化方向需是在梯度上，因此 $\Delta w_i \propto -\frac{\partial E}{\partial w_i}$，所以令$\Delta w_i = -\mu\frac{\partial E}{\partial w_i}$。

令$y' = f(h)$，$h = \sum w_i x_i$，通过链式求导来求$\frac{\partial E}{\partial w_i}$

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2}(y-y')^2 = -(y-y')\frac{\partial y'}{\partial w_i} = -(y-y')\frac{\partial f(h)}{\partial w_i} = -(y-y')f'(h)\frac{\partial h}{\partial w_i}$$
$$= -(y-y')f'(h)\frac{\partial}{\partial w_i}\sum w_i x_i = -(y-y')f'(h)x_i$$

令$\delta = (y-y')f'(h)$，所以 $w_i = w_i + \Delta w_i = w_i - \mu\frac{\partial E}{\partial w_i} = w_i + \mu(y-y')f'(h)x_i = w_i + \mu\delta x_i$

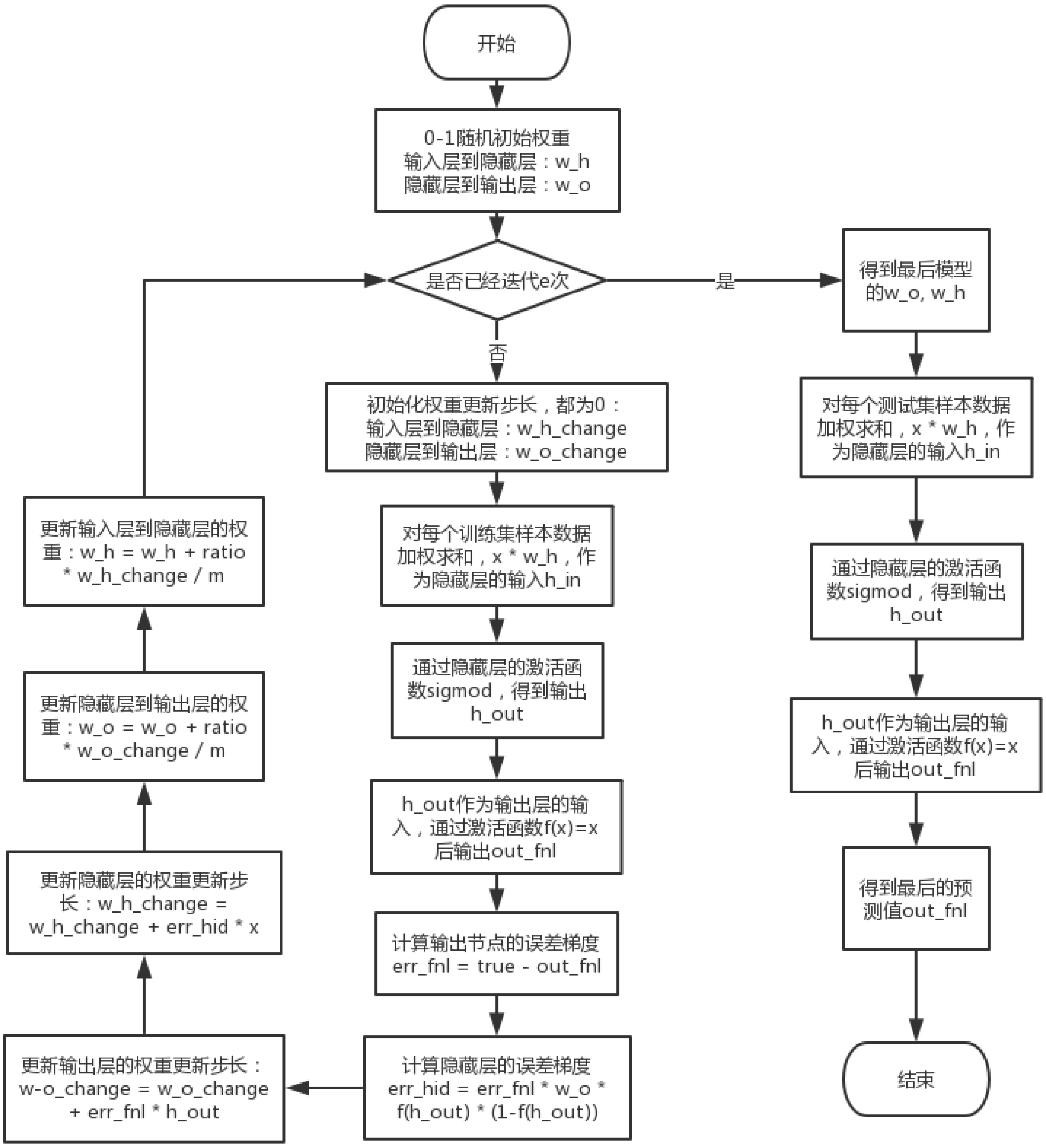
• 由上述公式，若第 k+1 层的误差为δ^{k+1}，则第 k 层节点 j 的误差即为 h+1 层误差乘以两层间的权重向量 w：$\delta_j^k = \sum \delta^{k+1} w f'(h_j)$。这就是反向传播的关键步骤。
• 输出层的激活函数 f(x)=x，因此输出层的误差：$\delta = (y-y')$

•隐藏层的激活函数$f(x) = \mathrm{sigmod}(x) = \frac{1}{1+e^{-x}}$，$f'(x) = f(x) * (1 - f(x))$，因此隐藏层的误差：

$\delta = (y-y')f(h)(1-f(h))$
• 权重步长更新：$\Delta w_{ij} = \Delta w_{ij} + \delta_j^k x_i$
• 权重的更新：$w_{ij} = w_{ij} + \mu\Delta w_{ij}/m$，其中$\mu$是学习率，m 是数据点的数量。

迭代多次，使得 E 尽可能小，构造的神经网络模型尽可能准确，从而可以用来预测其他数据。

2. 伪代码


开始
0-1随机初始权重
输入层到隐藏层：w_h
隐藏层到输出层：w_o
是否已经迭代e次
是
否
得到最后模型的w_o, w_h
初始化权重更新步长，都为0：
输入层到隐藏层：w_h_change
隐藏层到输出层：w_o_change
对每个训练集样本数据加权求和，x * w_h，作为隐藏层的输入h_in
通过隐藏层的激活函数sigmod，得到输出h_out
h_out作为输出层的输入，通过激活函数f(x)=x后输出out_fnl
计算输出节点的误差梯度 err_fnl = true - out_fnl
计算隐藏层的误差梯度 err_hid = err_fnl * w_o * f(h_out) * (1-f(h_out))
更新输出层的权重更新步长：w-o_change = w_o_change + err_fnl * h_out
更新隐藏层的权重更新步长：w_h_change = w_h_change + err_hid * x
更新隐藏层到输出层的权重：w_o = w_o + ratio * w_o_change / m
更新输入层到隐藏层的权重：w_h = w_h + ratio * w_h_change / m
对每个测试集样本数据加权求和，x * w_h，作为隐藏层的输入h_in
通过隐藏层的激活函数sigmod，得到输出h_out
h_out作为输出层的输入，通过激活函数f(x)=x后输出out_fnl
得到最后的预测值out_fnl
结束

3. 关键代码截图（带注释）

```python
# 计算每一列属性值与结果的相关性
def relevant(file):
    data_ori, cnt_ori = readData(file)
    data = data_ori
    cnt = cnt_ori
    y_avg = sum(cnt) / len(cnt)  # 求y的平均值
    cnt = cnt - y_avg
    m, n = np.shape(data)
    corr = []
    var_y = np.dot(cnt.T, cnt) / m
    for i in range(n):
        x_avg = sum(data[:, i]) / len(data[:, i])  # 求x的平均值
        data[:, i] = data[:, i] - x_avg
        var_x = np.dot(data[:, i].T, data[:, i]) / m  # 方差
        cov = np.dot(data[:, i].T, cnt) / m  # 协方差
        temp = cov / (np.sqrt(var_x) * np.sqrt(var_y))  # 相关系数
        corr.append(temp[0, 0])
    index = []
    for i in range(len(corr)):
        if(abs(corr[i]) >= 0.2):
            index.append(i)
    return index
```

```python
# 数据预处理
def processData(file_train, file):
    data_ori, cnt = readData(file)
    index = relevant(file_train)
    # 剔除相关性太低的列
    m, n = np.shape(data_ori)
    data = np.mat(np.ones((m, 1 + len(index))))  # 添加截距项
    for i in range(len(index)):
        for j in range(m):
            data[j, 1 + i] = data_ori[j, index[i]]
    # min-max归一化
    m, n = np.shape(data)
    for i in range(n):
        min_ = min(data[:, i])
        max_ = max(data[:, i])
        dif = max_ - min_
        if(dif != 0):
            for j in range(m):
                data[j, i] = (data[j, i] - min_) / dif
    return data, cnt
```

```python
def train(file):
    data, cnt = processData(file, file)
    m, n = np.shape(data)
    # 初始化权重，0到1之间的随机数
    w_o = np.mat(np.random.random(size=(1, n * 2)))
    w_h = np.mat(np.random.random(size=(n * 2, n)))
    ratio = 0.001  # 学习率
    for i in range(1000):
        # 初始化权重更新步长
        w_o_change = np.mat(np.zeros((1, n * 2)))
        w_h_change = np.mat(np.zeros((n * 2, n)))
        # 计算最后的输出
        h_in = np.dot(data, w_h.T)
        h_out = sigmod(h_in)
        out_fnl = np.dot(h_out, w_o.T)
        # 计算误差梯度
        err_fnl = cnt - out_fnl
        err_hid = np.multiply(err_fnl * w_o, np.multiply(h_out, (1 - h_out)))
        # 更新权重更新步长
        w_o_change = w_o_change + err_fnl.T * h_out
        w_h_change = w_h_change + err_hid.T * data
        # 更新权重
        w_o = w_o + ratio * w_o_change / m
        w_h = w_h + ratio * w_h_change / m

    return w_o, w_h
```

4. 创新点&优化（如果有）

数据预处理：

- 根据相关性大小进行筛选

$$\rho_{XY} = \frac{Cov(X,Y)}{\sqrt{D(X)}\sqrt{D(Y)}}$$

根据相关性系数公式，求出每一列（每个属性）与结果列直接的相关性，剔除掉相关性太低的属性列。从而选出真正能够影响到结果的属性。

- 归一化：min-max 离差标准法

对每一列数据进行归一化，减少因为某一列数据值过大，即使权重很小也依然能对结果产生过大影响的情况。Min-max 是对原始数据的线性变化，将数据映射到[0,1]：

$$x* = \frac{x - min}{max - min}$$

三、 实验结果及分析

1. 实验结果展示示例（可图可表可文字，尽量可视化）

小数据集验证：

X=[1,0,1],　label 为 1，权重为 0 到 1 的随机数，学习率 0.001：

```
数据：
[[ 1.  0.  1.]]
标签：
[[ 1.]]
输入层到隐藏层的权重w_h：
[[ 0.62654702  0.66981198  0.63127019]
 [ 0.1763892   0.32460732  0.04360528]]
隐藏层到输出层的权重w_o：
[[ 0.43466372  0.42396215]]
```

手动计算：

x1=1, x2=0, x3=1, y=1

w14=0.62654702, w24=0.66981198, w34=0.63127019

w15=0.1763892, w25=0.32460732, w35=0.04360528

w46=0.43466372, w56=0.42396215

前向：

隐藏层节点的输入：

h_in4 = x1*w14 + x2*w24 + x3*w34 = 1.25781721

h_in5 = x1*w15 + x2*w25 + x3*w35 = 0.21999448

隐藏层节点的输出：

h_out4 = sigmod(h_in4) = 0.778650123791

h_out5= sigmod(h_in5) = 0.554777871671

输出层的输出=输入：

out = in = h_out4*w46 + h_out5*w56 = 0.5736557786315177

保留八位数后 out=0.57365578

与图片中的“预测值”符合

反向：

输出层的误差梯度：

err = out – label = -0.42634422

隐藏层的误差梯度：

err_hid4 = err * w46 * h_out4 * (1-h_out4) = -0.03194004

err_hid5 = err * w56 * h_out5 * (1-h_out5) = -0.04464608

权重更新步长：

Δw46 = 0 + err * h_out4 = -0.33197298

Δw56 = 0 + err * h_out5 = -0.23652634

Δw14 = 0 + err_hid4 * x1 = -0.03194004

Δw24 = 0 + err_hid4 * x2 = 0

Δw34 = 0 + err_hid4 * x3 = -0.03194004

Δw15 = 0 + err_hid5 * x1 = -0.04464608

Δw25 = 0 + err_hid5 * x2 = 0

Δw35 = 0 + err_hid5 * x3 = -0.04464608

更新权重：

w46 = w46 + 0.001 * Δw46/ 2 = 0.43449773

w56 = w56 + 0.001 * Δw56 / 2 = 0.42384389

w14 = w14 + 0.001 * Δw14 / 2 = 0.62653104

w24 = w24 + 0.001 * Δw24 / 2 = 0.66981198

w34 = w34 + 0.001 * Δw34 / 2 = 0.63125422

w15 = w15 + 0.001 * Δw15 / 2 = 0.17636688

w25 = w25 + 0.001 * Δw25 / 2 = 0.32460732

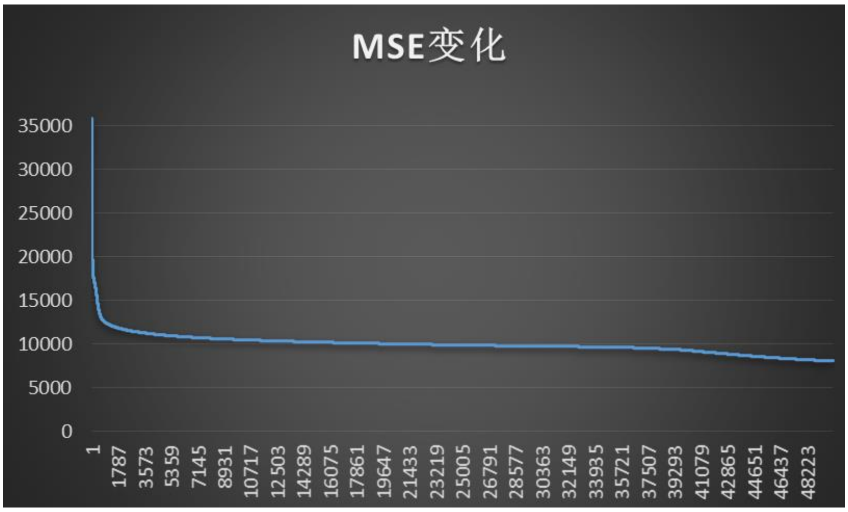
w35 = w35 + 0.001 * Δw35 / 2 = 0.04358296

上述结果，因为存在尾数问题，适当的四舍五入后，与下图中结果一致：

```
数据：
[[ 1.  0.  1.]]
标签：
[[ 1.]]
输入层到隐藏层的权重w_h：
[[ 0.62654702  0.66981198  0.63127019]
 [ 0.1763892   0.32460732  0.04360528]]
隐藏层到输出层的权重w_o：
[[ 0.43466372  0.42396215]]
迭代1次之后的w_h:
[[ 0.62657896  0.66981198  0.63130213]
 [ 0.17643384  0.32460732  0.04364993]]
迭代1次之后的w_o:
[[ 0.43499569  0.42419868]]
预测值：
[[ 0.57365578]]
```

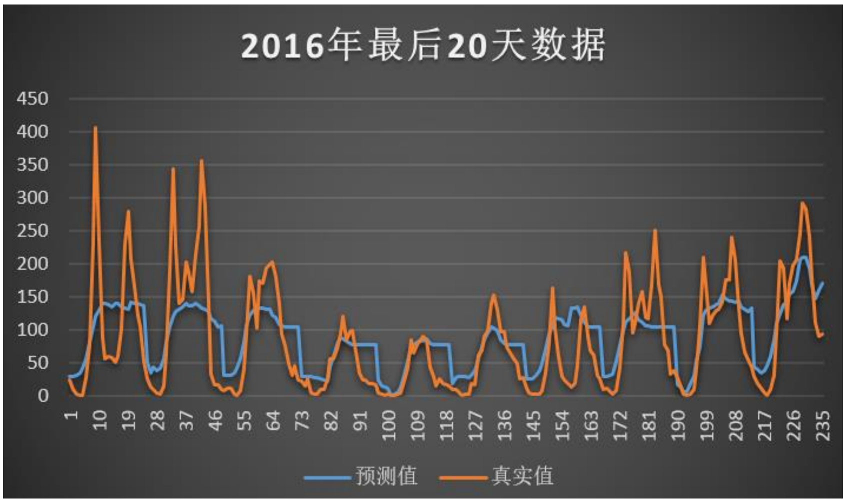
2. 评测指标展示即分析（如果实验题目有特殊要求，否则使用准确率）

学习率=0.001，权重初始化为 0 到 1 的随机数，权重更新步长初始化为 0，剔除了相关性小于 20%的属性列，对数据进行了归一化，隐藏层节点个数为特征数两倍的情况下：



每次迭代记录下的 train_lose 和 validate_loss：

最后 20 天预测结果和真实结果的对比图：



3. 思考题

（1）尝试说明下其他激活函数的优缺点。

- Tanh 函数：
  优点：Tanh 函数把输入值映射到（-1，1）的范围，因此基本上，它是 0 均值的，而 sigmod 因为不是 0 均值的，会导致后面的神经元的输入为非 0 均值的信号，可能会对梯度产生影响，因此这一点上，Tanh 函数比较有优势。
  缺点：存在梯度饱和问题，当时输入非常大或者非常小的时候，神经元的梯度会接近于 0.
- 近似生物神经的激活函数 Softplus、ReLu:
  • softplus 函数的 logistic-sigmod 函数原函数 softplus（x）=log(1+e^x)，可以看做强制非负校正函数的平滑版本
  • 强制非负校正函数 ReLu（x）=max（0，max），收敛速度比 sigmod/tanh 快很多，不需要指数运算，计算复杂度不会那么高，只需要一个阈值就可以得到激活值。但是在 x<0 时梯度为 0，这样就导致负的梯度在这个 ReLu 被置零，而且这个神经元有可能再也不会被任何数据激活，导致神经元“坏死”。

（2）有什么方法可以实现传递过程中不激活所有节点？

答：以概率 P 舍弃部分神经元，其他神经元以概率 q=1-p 被保留，被舍弃的神经元的输出设置为 0，这样就不会对下层产生影响。

（3）梯度消失和梯度爆炸是什么？可以怎么解决？

答：

• 梯度消失：当神经网络的层数非常深的时候，最后一层产生的偏差越来越小，最终会变成 0，因为是反向传播，所以可能会导致层数比较浅的权重没有得到更新。另外一个原因是，使用 sigmod 函数将数据映射到 0 和 1 之间，而这个函数的求导 f’(x)=f(x)(1-f(x))，两个 0 到 1 之间的数据相乘会得到一个更小的数，导致更新值越来越小，最终变成 0，从而导致层数比较浅的权重没有得到更新。

• 梯度爆炸：由于初始化权重过大，前面层次的变化会比后面层次的变化更快，导致权重越来越大。

因此，梯度消失和梯度爆炸的产生，都是因为浅层次的梯度来自于其后面层上项的乘积（实际上就是因为链式求导）。因此，解决可以有：使用 ReLu 激活函数来替代 sigmod 函数，或者初始化权重时不要太大，在合理范围内。

|---------------- 如有优化，重复 1，2 步的展示，分析优化后结果 -------------|

PS：可以自己设计报告模板，但是内容必须包括上述的几个部分，不需要写实验感想