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	15M2	专业（方向）	互联网
学号	15352218	姓名	林燕娜

一、 实验题目

利用 BP 神经网络实现回归

二、 实验内容

1. 算法原理

➢ **BP 神经网络：**

在之前，我们学到的几种算法里面，很多种都是线性不可分的。可比说 PLA，而 BP 神经网络，通过隐藏层的设计，使得达到多元可分的情况。

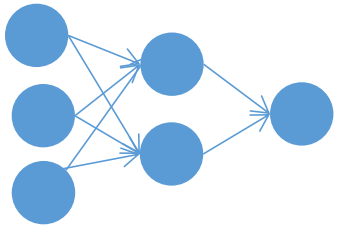
神经网络分为两个过程，一个是正向传递过程，一个是反向传递过程。

正向传递过程：经过一定的计算，可以得到最后的预测结果。

反向传递过程：根据预测到的最终结果与实际结果的误差，反作用于输入层的系数 W1 和隐藏层的系数 W2，进行一定的更新。

对于返向传递过程：

在本次实验中，利用的是 matlab 工具，鉴于之前 ta 课上说的利用 for 循环和直接用矩阵运算的时间成本差距非常大。所以自行进行了公式的推导，将最后的矩阵运算更新 W 公式算了出来。



假设：

->原来训练集的标签的 train_label（矩阵大小是 K*1）

->输入层的数据为 train(矩阵大小 K*M，其中 K 是样本数量，M 是特征属性个数)

->隐藏层的节点高数为 N

->输入层到隐藏层的系数矩阵 W1（矩阵大小为 M*N）

->隐藏层的输入是 $hidden\_input = train * W1$（矩阵大小是 K*N）

->隐藏层的输出是 $hidden\_output = \frac{1}{1+e^{-hidden\_input}} = 1./(1+e^{-hidden\_input})$（矩阵大小是 K*N）。

->隐藏层到输出层的系数矩阵为 W2（矩阵大小是 N*1）

->预测出来的是 $y\_predict\_label = hidden\_output * W2$（矩阵大小是 K*1）

则，误差为：$E = \frac{1}{2}(y\_predict - train\_label).^2$；

则对于隐藏层的系数 W2 求偏导：

$$\frac{\partial E}{\partial W2} = \frac{\partial E}{\partial y\_predict}\frac{\partial y\_predict}{\partial W2} = (hidden\_output')*(y\_predict - train\_label)$$

这样子的话，求出来的倒数是所有样本加起来的倒数，则 W2 的更新公式就是：

$$W2 = W2 - \eta\frac{\partial E}{\partial W2} = W2 - \eta*(hidden\_output')*(y\_predict - train\_label)/K$$

而对于输入层的系数 W1，求偏导：

$$\frac{\partial E}{\partial W1} = \frac{\partial E}{\partial y\_predict}\frac{\partial y\_predict}{\partial hidden\_output}\frac{\partial hidden\_output}{\partial hidden\_input}\frac{\partial hidden\_input}{\partial W1}$$
$$= (y\_predict - train\_label)*W2*hidden\_output*(1-hidden\_output)*train$$

而考虑到矩阵运算存在着维度上的问题，其导数在矩阵运算中应该为：

$$\frac{\partial E}{\partial W1} = (train')*\{(y\_predict - train\_label)*(W2').*[hidden\_output.*(1-hidden\_output)]\}$$

同样，算出来的导数是基于所有样本的导数和，则 W1 的更新公式为：

$W1 = W1 - \eta\frac{\partial E}{\partial W1}/K$。

因为 W1 公式比较复杂，可以一步步的思考，每一步代表什么意义。

$[hidden\_output.*(1-hidden\_output)]$ 是一个 K*N 的矩阵，其 ij 元素代表的是：第 i 个样本的第 j 个隐藏层节点的输出值和 1-输出值相乘。

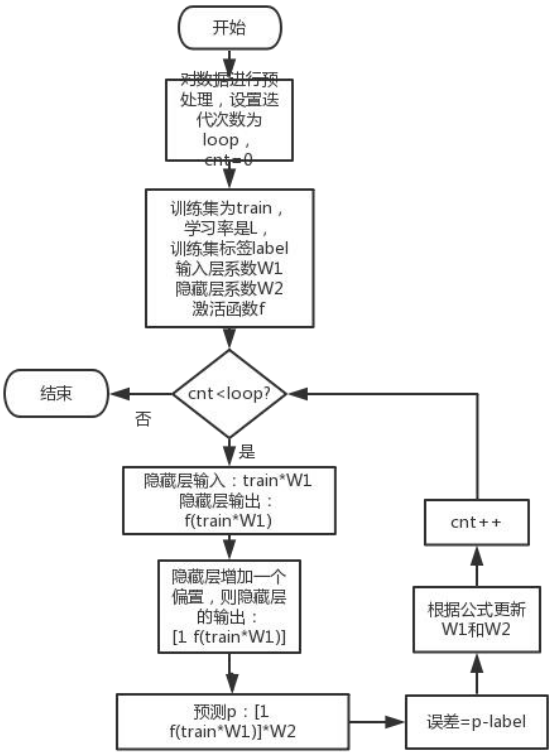
$(y\_predict - train\_label)*(W2')$ 是一个 K*N 的矩阵，其 ij 元素代表的是：第 i 个样本的第 j 个隐藏层节点对样本 i 的预测误差造成的影响，W2 对应的 j 元素就是其权重。

$\{(y\_predict - train\_label)*(W2').*[hidden\_output.*(1-hidden\_output)]\}$ 是

一个 K*N 的矩阵，其 ij 元素代表的是：第 i 个样本的第 j 个隐藏层节点输入的对最终误差的影响。

则最后与 train 的相乘，是一个 M*N 的矩阵，其 ij 元素代表的是：所有样本的对 Wij 的导数的一个求和。

2. 伪代码/流程图



3. 关键代码（带注释）

- **动态更新 W:**

->迭代 loop 次

->前向传输:

->根据当前的 W1，算出每个隐藏节点的输入，然后再利用激活函数，算出每个隐藏层节点的输出。

->根据 W2 算出最终的结果。

->后向传输:

->根据算出的最终结果，对 W2 和 W1 求导，然后用梯度下降法的公式更新 W1 和 W2。

```
for k=1:loop;
    %forward
    hidden_input = train*W1;
    hidden_output1 = 1.0 ./ (1.0+exp(-hidden_input));
    %加偏置
    one_hidden = ones(part_size,1);
    hidden_output = [one_hidden hidden_output1];
    %算出最终的预测结果，与原来的作比较算出误差。
    y_predict = hidden_output*W2;
    error_output = (y_predict - train_lable);
    E_train(k) = 0.5 * norm(error_output,2)* norm(error_output,2)/part_size;
    %backward，根据公式更新W1和W2
    W2_tmp = W2(2:Middle+1,:);%去掉偏置项
    W1 = W1 - learning*(train')*((error_output*W2_tmp').*(hidden_output1.*(1-hidden_output1)))/part_size;
    W2 = W2 - learning*(hidden_output)'*(error_output)/part_size;
```

4. 创新点&优化（如果有）

- **数据预处理：**

在本次实验中，可以看到原始数据是比较杂乱的，很多并没有实际的参考价值。所以进行了数据的预处理。

将 instant 的 id、年份 year、日期、weekday 等无用或者重复信息忽视掉。

对 season、weather、hr 等属性进行一个 onehot 处理。

具体：

对于 season，一共有四个季度，用三个属性就可以代表着四个季度；

对于 weather，用一个属性来代表，1、2 代表天气好，3、4 代表天气不好；

对于 hr，分为上班时间和下班时间；

最终整体属性的数据都在 0—1 之间。

- **自己推导 E 对 W 的公式，以一个矩阵运算代替 for 完成更新：**

鉴于之前 ta 课上说的 matlab 利用 for 循环和直接用矩阵运算的时间成本差距非常大。所以并没有选择一个个样本进行运算，而是一次性计算所有的样本，利用矩阵运算达到佳和的效果。

具体原理见实验原理。

- **将 sigmoid 函数换成 tanh 函数：**

简单的在计算隐藏层的输出的是有，将公式换成对应的 tanh 的公式，在最后反向传播的时候，将

$$\frac{\partial hidden\_output}{\partial hidden\_input} = hidden\_output * (1 - hidden\_output)$$

换成：

$$\frac{\partial hidden\_output}{\partial hidden\_input} = 1 - \tanh(hidden\_output)^2$$

就可以了。

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）

数据集的分法：

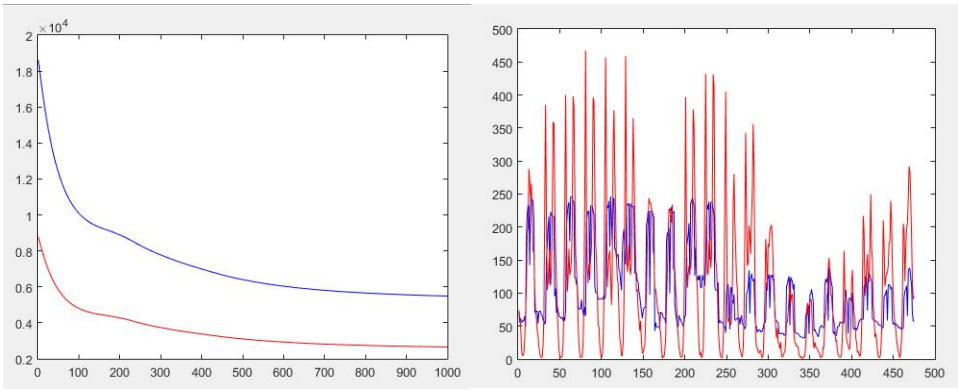
利用随机数给每个样本生成一个 0-1 之间的数字，根据随机数排序，取出最后的 20%作为验证集，前面的 80%作为训练集。

结果展示：

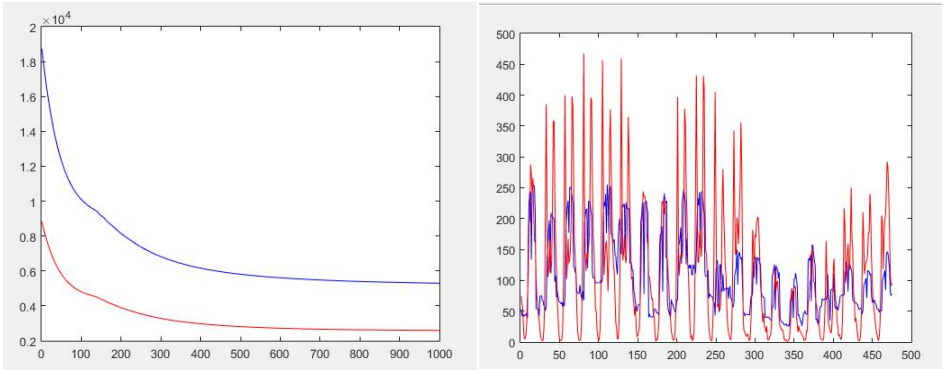
- **用 sigmoid 函数，迭代 1000 次的结果：**

（左图的蓝色线为训练集的 loss 曲线，红色为验证集合的 loss 曲线）

（右图的蓝色线为预测结果，红色为原来的真实结果。）



- **用 tanh 激活函数，迭代 1000 次结果：**



- **小数据集验证**

循环次数 1，三个输入节点，2 个隐藏节点，1 个输入节点，激活函数为 sigmoid，学习率等于 1。

训练集和测试集如下：

x1	x2	x3	lable
0	0.7	0.3	10
0.8	0.2	0.1	?

取输入层的系数为：$W1=\begin{bmatrix}1 & 0\\ 1 & 3\\ 2 & 3\end{bmatrix}$，隐藏层系数为：$W2=\begin{bmatrix}0\\ 4\\ 2\end{bmatrix}$

则，输入层的输入：$hidden\_input = train * W1 = \begin{bmatrix}0.5 & 0.7 & 0.3\end{bmatrix}\begin{bmatrix}1 & 0\\ 1 & 3\\ 2 & 3\end{bmatrix} = \begin{bmatrix}1.8 & 3\end{bmatrix}$

输入层的输出：$hidden\_output = 1.0./(1+e^{-hidden\_input}) = \begin{bmatrix}0.858 & 0.9526\end{bmatrix}$

增加一个偏置，则 $hidden\_output1 = \begin{bmatrix}1 & 0.858 & 0.9526\end{bmatrix}$

则最后预测为：y_predict = 5.3377.

W1 的更新：

$$\frac{\partial E}{\partial W1} = \begin{bmatrix}0.5\\ 0.7\\ 0.3\end{bmatrix} * ((5.3377-10) * \begin{bmatrix}4 & 2\end{bmatrix} .* (\begin{bmatrix}0.858 & 0.9526\end{bmatrix} .* (1-\begin{bmatrix}0.858 & 0.9526\end{bmatrix})))$$

$$= \begin{bmatrix}-1.1362 & -0.2105\\ -1.5907 & -0.2948\\ -0.6817 & -0.1263\end{bmatrix}$$

$$W1 = \begin{bmatrix}1 & 0\\ 1 & 3\\ 2 & 3\end{bmatrix} - 1 * \begin{bmatrix}-1.1362 & -0.2105\\ -1.5907 & -0.2948\\ -0.6817 & -0.1263\end{bmatrix} = \begin{bmatrix}2.1362 & 0.2105\\ 2.5907 & 3.2948\\ 2.6817 & 3.1263\end{bmatrix}$$

同理。可得到，W2：

$$W2 = \begin{bmatrix}0\\ 4\\ 2\end{bmatrix} - 1 * \begin{bmatrix}1\\ 0.858\\ 0.9526\end{bmatrix} * (5.3372 - 10) = \begin{bmatrix}4.6628\\ 8\\ 6.442\end{bmatrix}$$

则根据 W1，W2，可以判断测试集为：16.94.

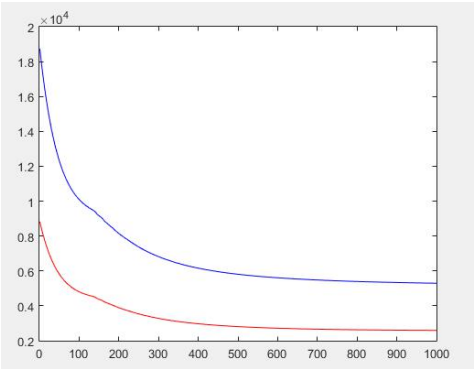
程序跑出的结果：

```
W1 =

    2.1351    0.2106
    2.5891    3.2949
    2.6810    3.1264

>> W2

W2 =

    4.6623
    8.0009
    6.4411

>> last_y_predict

last_y_predict =

   16.9333
```

验证完毕，结果正确。

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）

- Loss 函数：



蓝色线为训练集的 loss 曲线，红色为验证集合的 loss 曲线。从图中可以看出，随着迭代次数的增加，loss 会逐渐下降。这是因为随着迭代次数的增加，输入层的系数和输出层的系数不断的更新，划分的曲线（BP 神经网络就是一堆线去将一个空间进行划分）也越来越拟合我们的数据，所以 loss 整体呈现一个下降的趋势。

四、 思考题

- **尝试说明下其他激活函数的优缺点。**

ReLU 函数：

该激活函数的公式是：$f(x)=\max(0,x)$. 而其导数的计算公式为：

$$f'(x)=\begin{cases}1 & x>0\\ 0 & x<0\end{cases}$$

优点：

1）正向传播时，计算简单、速度快；
2）梯度不饱和，在反向传播过程中，就不会出现梯度大于 1 或者小于 1 的累乘，导致梯度消失或者梯度爆炸。
3）收敛速度相对于 sigmoid、tanh 函数更快。

缺点：

当步长过大、输入为负数的时候，导数会变成 0，这回引起该节点之后都不会更新。这样子，也就是 RELU 的噪声鲁棒性非常差。稍微有一个噪声，就可能导致一些点坏死。

- **有什么方法可以实现传递过程中不激活所有节点？**

1）可以经过一些数据的预处理，筛选掉一些属性、特征，就可以不用激活这些节点。
2）给每个节点设置一个阈值，如果值没有达到阈值，就设置该节点的 value=0，就能达到不激活这个节点的目的。

- **梯度消失和梯度爆炸是什么？可以怎么解决？**

梯度消失（gradient vanishing）：

在本次实验过程中，我们用到的是 sigmoid、tanh 函数作为激活函数。两个函数都能将负无穷到正无穷映射到 0/-1 到 1 之间的函数，并且其导数分别是：f′(x)=f(x)(1-f(x))和 1-tanh(x)^2，都是小于 1 的数字。在反向传播的过程中会对函数链式求偏导然后相乘。如果隐藏层次过多，乘的次数过多，出现了 0 的情况，导致比较前面的隐藏层无法更新。这就是梯度消失。

解决方法：

因为这是因为 sigmoid、tanh 函数的特性导致的，所以解决方法就是换一种激活函数，可比说 ReLU 激活函数的函数范围 0 到正无穷，并且，其梯度要么 0 要么 1，不会随着 x 的增大或减小消失。

梯度爆炸(gradient explosion)：

同样的，如果是激活函数大于 1，然后层次过多，多次相乘累积可能就太大，使得最后的梯度爆炸了。

解决方法：

1）设置梯度剪切阈值，一旦梯度超过那个值，就限制其大小，就设置为阈值。
2）也可以和梯度消失一样，换一个 ReLU 激活函数。