**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 | **15M1/1518** | 专业（方向） | **移动互联网** |
| 学号 |  | 姓名 | **Jw** |

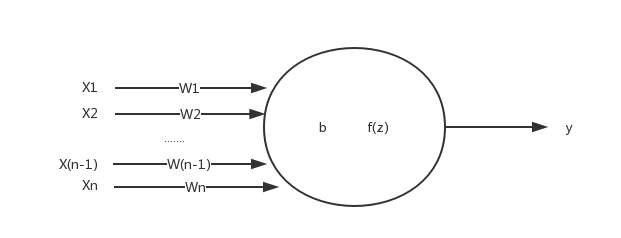
# 实验题目

**BP神经网络(误差反向传播神经网络) ------ Back propagation nerual network**

# 实验内容

1. 算法原理

**神经元模型：**学习神经网络首先要知道网络的组成元素----神经元。神经网络是由许多神经元组成的网络系统，一个神经元的组成如下图：



它包括：

1.输入信号Xi，这些信号代表来自环境的数据或其他神经元的激活

2.一组实值权重Wi，这些权重的值代表连接强度

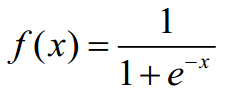
3.一个阈值b

4.一个激活函数f（z）

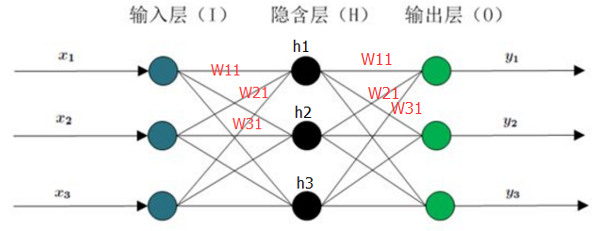
5.输出y

这其中：

在实际应用中，激活函数f通常采用Sigmoid函数，将数据映射到(0，1)范围：



**BP神经网络：**BP神经网络是一种三层或三层以上的神经网络模型。它的训练算法采用的是误差逆向传播算法(err Back Propagation)。网络模型拓扑结构包括三层，分别是输入层（input layer）、隐藏层（hidden layer）和 输出层（output layer），其中输入层和输出层都只有一层点，但是**隐藏层可以是一层也可以是有很多层**。一个例子如下（注意输入层与隐含层之间的权重w 和 隐含层与输出层之间的w 是不一样的）：



BP神经网络的每一次迭代包括两个部分、前向传播输入以及反向传播误差。也就是先从左向右先将输入传递到输出层产生结果，然后计算输出层结果与真实结果的误差，再将误差逆向传播去修改权重。重复上述过程直到输出误差收敛或者迭代到了一定的次数。

**前向传播输入部分：**

1. 给定隐藏层或输出层的单元j，单位j的净输入为

是从上一层单元i到单元j的连接权重，是上一层单元i的输出，是单元j的阈值。

2. 给定单位j的净输入，则对隐藏层单位j的输出为

对输出层单位j的输出为

**反向传播误差部分：**

我们的目标是最小化输出的误差，为了让误差以最快的速度下降，我们采用梯度下降法作为学习策略，将误差往回传然后不断去调整权重和阈值，流程如下(推导在后面):

1. 对于输出层中的单元k, 设真实输出为误差由下式计算

2. 对于隐藏层单元j的误差为:

3. 权重以及阈值更新公式:

其中 alpha是学习率，是一个(0,1)之间的浮点数，上式是所有情况，要注意的是由于输出层的输出=输入，此时隐藏层到输出层的更新公式中为1，其他情况就是激活函数的导数。

**梯度下降的推导：**

我们采用均方误差来衡量，为方便计算，将每一个输出点k误差定义为

将均方误差误差定义为

我们的目标是要让E最小，这个目标通过对权重和阈值的调整来实现。

为了调整权重，我们对E求的偏导得:

同理，为了调整阈值，我们对E求的偏导得：

\*1

然后权重更新公式就是：

上式是针对一个样本的情况，如果有多个样本，这上面两式+后面的部分为所有样本求出的值的均值。

1. 伪代码

**训练部分：**

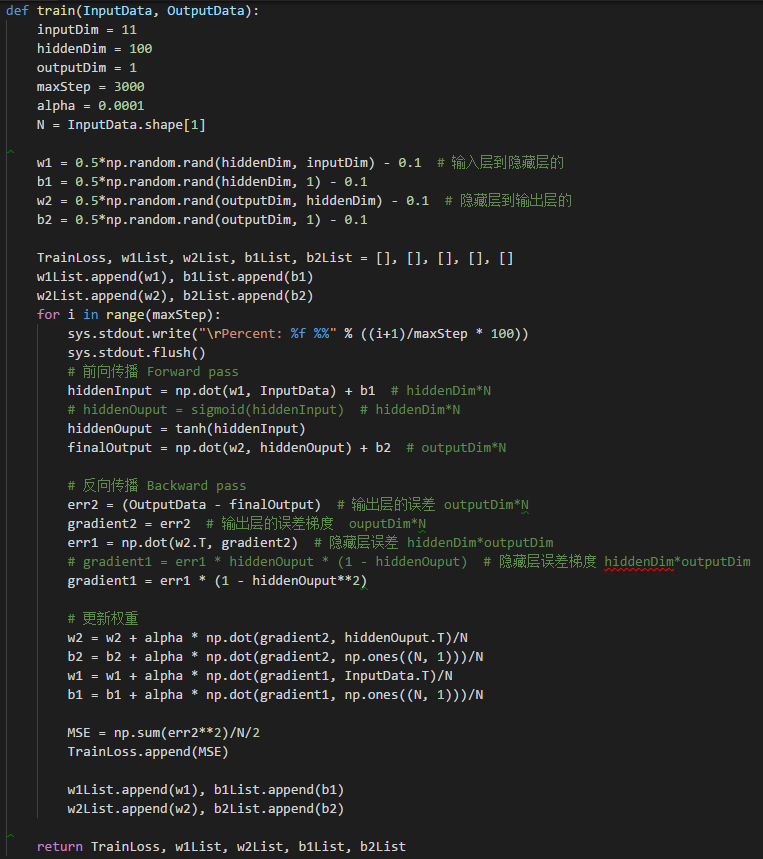
|  |
| --- |
| **Input：**训练集D={(xk, yk) | 1<=k<=n}  **Output：**权重矩阵W1、W2，阈值矩阵B1，B2  [**Initialize**](http://www.baidu.com/link?url=Y_fU4WNoXXr3KvG6_AtbUgVH7yw51MwMaUbKP9wjvoebV9tS3A71xZkvN37Qwunb-xZtpGJFH0InLNqoXVc5Qt5ibIMw_cBqSnggKFi7egu)**：**W和B全随机，设置最大迭代次数maxStep，步长  ***for t = 0,1…maxStep***  ***hiddenInput = W1\*InputData + B1***  ***finalInput = hiddenOuput = f（hiddenInput）***  ***finalOuput = W2 \* hiddenOutput + B2***  ***errK = realOuput – finalOutput***  ***errj = W2 \* errK \* f’(hiddenInput)***  ***W2 = W2 + alpha \* errK\*hiddenOutpu/Nt***  ***B2 = B2 + alpha \* errK/N***  ***W1=W1+alpha\*errj\*InputData/N***  ***B1=B1+alpha\*errj/N***  ***end for*** |

**训练部分：**

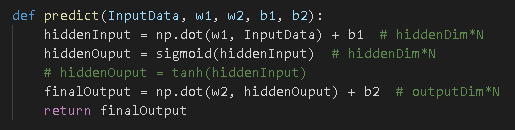
|  |
| --- |
| **Input：**测试集D={(xk, yk) | 1<=k<=n}，训练好的W1,W2，B1，B2  **Output：**预测结果finalOutput  ***hiddenInput = W1\*InputData + B1***  ***finalInput = hiddenOuput = f（hiddenInput）***  ***finalOuput = W2 \* hiddenOutput + B2*** |

1. 关键代码截图（带注释）

1.训练函数



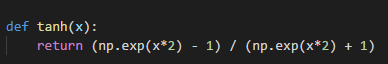
2.预测函数、输入需要预测的矩阵以及训练好的W1、W2、B1、B2，输出标签Y

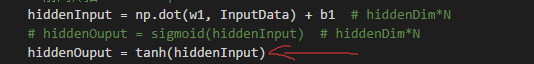


1. 创新点&优化（如果有）

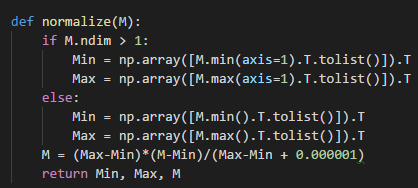
1. python向量化运算

2. 激活函数除了用了默认的sigmoid，还尝试了双曲正切tanh





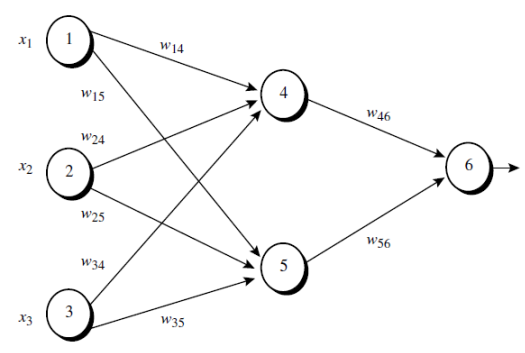
3.对数据使用了minmax 归一化



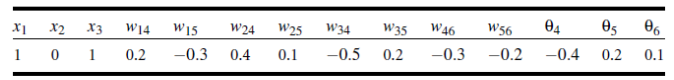
# 实验结果及分析

1. 实验结果展示示例（可图可表可文字，尽量可视化）

一个简单的小数据集：一个训练元组（X=1,0,1），标签为1



输入数据如下：



使用sigmoid作为激活函数，迭代一次，学习率为0.9。

**手动计算结果如下**：

前向传播过程：

|  |  |  |
| --- | --- | --- |
| Unit j | Net input IJ | Output Oj |
| 4 | 0.2+0-0.5-0.4 = -0.7 | 1/（1+e0.7）=0.332 |
| 5 | -0.3 + 0 + 0.2 + 0.2 = 0.1 | 1/（1+e-0.1）=0.525 |
| 6 | (-0.3)(0.332)-(0.2\*0.525)+0.1=-0.105 | -0.105 |

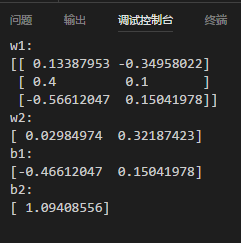
反向传播过程：

|  |  |
| --- | --- |
| Unit j | Err j |
| 6 | 1 - （- 0.105）=1.105 |
| 5 | 0.525\*（1-0.525）\*（1.105）\*（-0.2）= -0.055 |
| 4 | 0.332\*（1-0.332）\*（1.105）\*（-0.3）= -0.0735 |

权重及阈值更新（用b代替）

|  |  |
| --- | --- |
| W46 | -0.3 + 0.9 \* 1.105 \* 0.332 = 0.030 |
| W56 | -0.2 + 0.9 \* 1.105 \* 0.525 = 0.322 |
| W14 | 0.2 + 0.9 \* (-0.0735) \* 1 = 0.13385 |
| W15 | -0.3+0.9\*（-0.055）\*1 = -0.3495 |
| W24 | 0.4+0.9\*（-0.0735）\*0=0.4 |
| W25 | 0.1+0.9\*（-0.055）\*0 = -0.1 |
| W34 | -0.5+0.9\*（-0.0735）\*1= -0.56615 |
| W35 | 0.2 + 0.9\*（-0.055）\*1 = 0.1505 |
| B6 | 0.1+0.9\*1.105 = 1.0945 |
| B5 | 0.2+0.9\*（-0.055）=1.505 |
| B4 | -0.4 + 0.9\*（-0.0735）=- - 0.46615 |

**程序计算结果如下，直接看最后的权重和阈值结果**：



如图W1 的 第i行分别对应 Wi4 和 Wi5 (1<=i<=3)

W2的两个数字分别是W46 和 W56

b1的两个数字分别是b4、b5

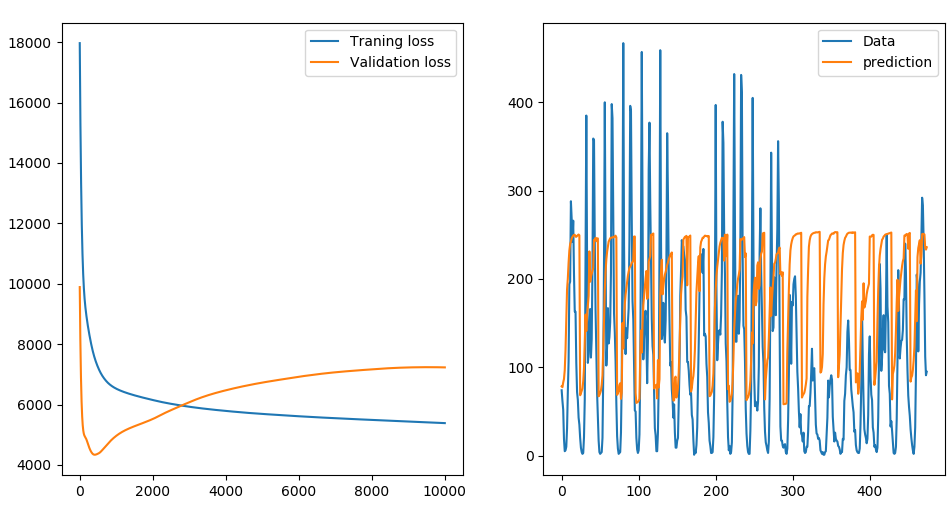
b2 的数字 是 b6

对比可知，在计算误差范围内，与手动计算结果基本一致。

**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）**

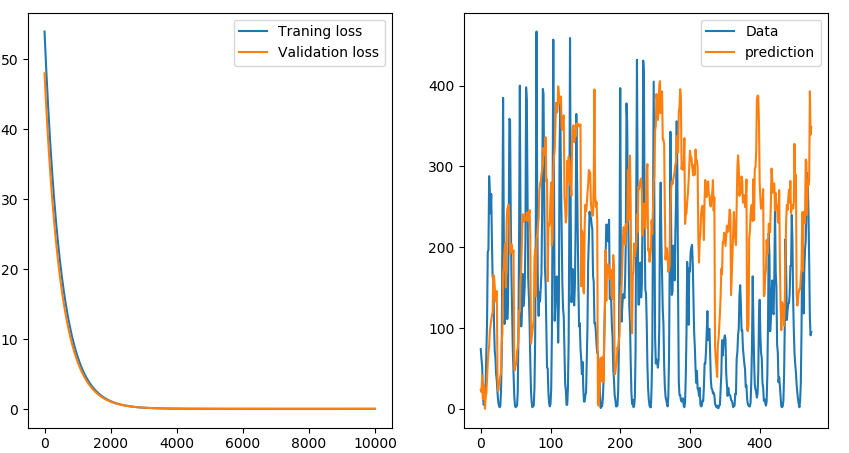
我将训练集数据的后20天即2011年12月11号到2011年12月30号的数据作为验证集（共475个样本），并且去掉了训练数据中的索引列、日期列、年列，因为个人觉得没什么用，前两个都是一个索引作用，年的话训练集都是11年也没有用。

采用sigmoid作为激活函数、没有作归一化处理、学习率固定为0.0001、100个隐藏层结点且迭代1万次的结果如下：



左边 Loss function的对比图 可以看到训练集的loss仍在下降，但是验证集的loss却已经触底反弹，说明这个时候已经过拟合了，从右边输出结果对比也可以看到预测的结果不好。造成这种现象的原因可能是学习率设置太大，数据没有归一化发散性比较强，我观察了一下12月的真实结果都比较小，而训练集有很多挺大的结果，这就很大几率造成训练集训练出来的模型在验证集上表现很差。

采用双曲正切函数tanh作为激活函数，并且使用minmax进行归一化处理，学习率固定为0.00002，100个隐藏层结点且迭代1万次的结果如下：

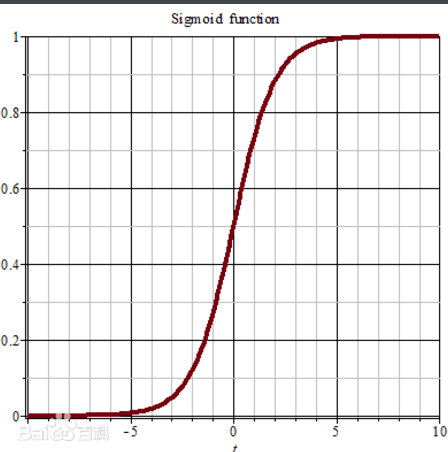


可以发现左边loss曲线都比较好，到最后都收敛了，纵坐标值比较小是因为做了归一化。右边输出结果对比也比之前好很多，在最后样本数300-400这个区间的结果差距比较大，原因也可能是因为12月真实标签数据较小，而训练集的标签数据大数比较多，模型不能很好地符合验证集。

# 思考题

**1.尝试说明下其他激活函数的优缺点**

答：1. **Sigmoid函数**：



优点：将数据映射到了（0,1）区间，数据不容易发散；

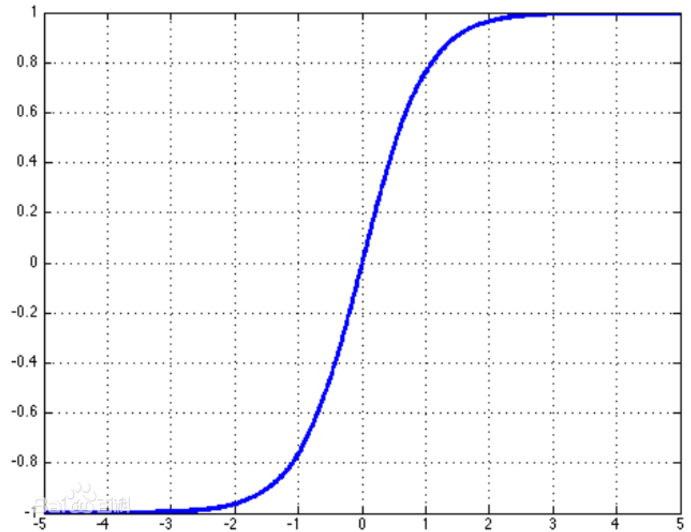
让数据的传递有非线性的形式，这样才能逼近目标函数；

是凸函数，有最优值，可导，可以采用梯度下降的优化方法

缺点：当输入过大或者过小的时候，梯度接近0，出现过饱和现象，因此在反向传播时，这个局部梯度会与整个代价函数关于该神经元输出的梯度相乘，结果也会接近为 0；

Sigmoid函数不是关于原点中心对称的，这样会导致后面网络层的输入也不是以0为均值的，后果就是假如输入数据为正，那么反向传播时W就会全为正或者全为负，就会导致梯度下降出现锯齿形波动。

2.**tanh函数：**



优点：将数据映射到（-1,1），由图可以看出它其实是sigmoid的放大版，除了拥有和sigmoid函数一样的优点外，它是以原点为中心的。

缺点：同样存在过饱和问题使得梯度消失。

3. **ReLU函数：**

F(x) = max(0,x)

优点：在梯度下降上比前两者有更快的收敛速度，因为它是线性非饱和的；

计算比前两者要简单快速；

缺点：大的梯度流过一个ReLU神经元并且更新了参数之后，这个神经元可能再也不会对任何数据有激活现象，这时这个神经元的梯度永远会是0。一般在学习率设置较大的时候会出现这种情况。

4. **Leaky-ReLU函数**：

F(x) =x(x>=0) F(x) = ax(x<0) a是很小的数

优点：保留了很小的负数梯度值，解决ReLu容易大量神经元死亡的问题。

缺点：效果不太稳定

**2.有什么方法可以实现传递过程中不激活所有节点？**

答：可以使用dropout技术，就是在每次训练的时候，随机临时删除一半神经元，这样做的好处是能提高网络的泛化能力，防止过拟合。

**3. 梯度消失和梯度爆炸是什么？可以怎么解决？**

答：我们知道在反向传播的过程中，梯度是会逐层相乘往前传播，那么假如每一层的梯度都是一个比1小的很小的数，那么经过足够多层的传播之后梯度会变成0，例如如果每一层的梯度都是0.9，那么反向传n层会有，这样权重基本就不会再更新，这就是梯度消失现象。

梯度爆炸刚好相反，由于初始权重过大，这样误差大梯度也大，回传时候前面层的变换就会比后面层的变化更快，权重会越来越大。

解决办法：1. 初始化的权重不能过大也不要过小

2. 学习率设置要恰当，可以采用动态学习率

3. 采用ReLu函数代替sigmoid或tanh

# 一些补充

1. 关于隐藏层点数的确定，隐藏层点数是自己定的，但是随意定好像不是太靠谱，一般有几条经验公式：

其中m是隐藏层节点数、n是输入层节点数、l是输出层节点数、是一个1-10之间的常数。这次实验的时候前三条试过都不太好，最后一条我也结合了一下逐步试验发调了一下，最终用了100个点。

2.学习率最好应该设为自动调整，增加BPNN的稳定性。

3.我们不能一味地追求训练集误差达到最小，这样很容易过拟合，比如我实验结果展示的sigmoid那个结果，就是这样的情况，避免过拟合一般要试出最佳训练次数、然后训练数据划分训练集和验证集，用K折验证会更稳定。

4.BP网络的可解释性不强，很多参数、网络结构得凭经验去确定。