2022-6-21

BP_FittingTool 原理手册

目录

一、测试案例介绍

1 背景介绍

煤炭发电厂的输出废气中含有大量的重金属颗粒及粉尘，需要对废气进行处理后才能正常排放。MnO_2 作为一种常用的去除废气中苯离子的催化剂，每立方的废气中，MnO_2 的含量会影响其催化时间，其关系如下表所示。请设计一个神经网络实现对 MnO_2 的含量及其催化时间的拟合。

2 题目要求

1. 给出网络结构，给出网络建立方法、初始化方法和训练方法，给出每层神经网络的参数结构（权值、偏置的维度），并给出详细的程序流程框图；
2. 在同一张图上给出实际的催化时间曲线和神经网络训练后拟合的近似曲线；
3. 给出你对 MnO2 含量为 4.6~5.6g 下催化时间的预测值，将表中空白内容填写完整（保留 3 位小数）；

3 实验数据

表 1.1 MnO_2 含量及催化时间

MnO_2（g）	1	1.2	1.4	1.6	1.8	2
催化时间（h）	1.5000	0.5376	-0.0429	-0.3569	-0.4920	-0.5125
MnO_2（g）	2.2	2.4	2.6	2.8	3.0	3.2
催化时间（h）	-0.4649	-0.3814	-0.2840	-0.1871	0.1000	-0.0287
MnO_2（g）	3.4	3.6	3.8	4.0	4.2	4.4
催化时间（h）	0.0227	0.0515	0.0564	0.0375	-0.0020	-0.0547
MnO_2（g）	4.6	4.8	5	5.2	5.4	5.6
催化时间（h）						

4 问题分析

该问题属于对已知数据集的函数拟合及预测问题。可以使用 BP（Backpropagation 算法实现，下一章将介绍该方法。

二、BP 算法原理

1 多层网络结构

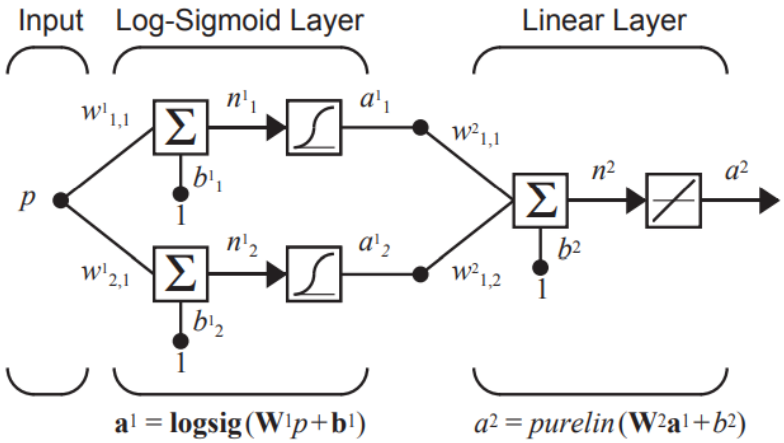


图 2.1 多层网络结果示意图

2 算法计算流程

2.1 正向传播

确定好网络结构之后，输入训练集数据进行正向传播。以图 2.1 的网络为例，正向传播结果为：

$$\begin{aligned}\mathbf{a}^1 &= \log sig\left(\mathbf{W}^1\mathbf{p} + \mathbf{b}^1\right)\\ \mathbf{a}^2 &= purelin\left(\mathbf{W}^2\mathbf{a}^1 + \mathbf{b}^2\right)\end{aligned} \tag{2.1}$$

推广到一般多层网络，正向传播的结果为：

$$\mathbf{a}^{m+1} = \mathbf{f}^{m+1}\left(\mathbf{W}^{m+1}\mathbf{a}^m + \mathbf{b}^{m+1}\right) \text{ for } m = 0,1,\ldots,M-1 \tag{2.2}$$

其中 $\mathbf{a}^0 = \mathbf{p}$ 为网络输入， $\mathbf{a} = \mathbf{a}^M$ 为输出层结果。

得到网络正向传播结果后，将其与目标输出（来自有监督学习的训练集）对比计算误差，为性能指数计算及反向传播做准备。

2.2 性能指数

在有监督学习中，网络输出与目标输出的偏差为误差。定义性能指数为误差的函数，通过迭代更新网络参数使性能指数不断减小，直至满足收敛要求，以达到网络设计目的。

这里使用近似均方误差定义性能指数，即使用第 k 次迭代的平方误差代替均方误差，具体公式如下：

$$\hat{F}(\mathbf{x}) = (\mathbf{t}(k) - \mathbf{a}(k))^T(\mathbf{t}(k) - \mathbf{a}(k)) = \mathbf{e}^T(k)\mathbf{e}(k) \tag{2.3}$$

明确性能指数后，可由式 2.4、2.5 使用近似最速下降法进行网络属性的迭代更新。

$$w_{i,j}^m(k+1)=w_{i,j}^m(k)-\alpha\frac{\partial\hat{F}}{\partial w_{i,j}^m} \tag{2.4}$$

$$b_i^m(k+1)=b_i^m(k)-\alpha\frac{\partial\hat{F}}{\partial b_i^m} \tag{2.5}$$

上式中 α 为学习率，其中偏导数项可由复合函数链式法则计算，2.3 节将介绍该计算方法的原理，2.4 节将介绍该方法在网络反向传播中的应用。

2.3 链式法则

复合函数的求导法则如下：

$$\frac{df(n(w))}{dw}=\frac{df(n)}{dn}\times\frac{dn(w)}{dw} \tag{2.6}$$

使用上式方法计算式 2.4、2.5 中的偏导数项得到：

$$\frac{\partial\hat{F}}{\partial w_{i,j}^m}=\frac{\partial\hat{F}}{\partial n_i^m}\times\frac{\partial n_i^m}{\partial w_{i,j}^m} \tag{2.7}$$

$$\frac{\partial\hat{F}}{\partial b_i^m}=\frac{\partial\hat{F}}{\partial n_i^m}\times\frac{\partial n_i^m}{\partial b_i^m} \tag{2.8}$$

其中 $n_i^m=\sum_{j=1}^{s^{m-1}}w_{i,j}^m a_j^{m-1}+b_i^m$ 为第 m 层的净输入，由此可将上述公式的第二项简化为：

$$\frac{\partial n_i^m}{\partial w_{i,j}^m}=a_j^{m-1},\frac{\partial n_i^m}{\partial b_i^m}=1 \tag{2.9}$$

为了方便分析，设 $s_i^m\equiv\frac{\partial\hat{F}}{\partial n_i^m}$ 为敏感度系数。将敏感度系数代入式 2.4、2.5 中得到：

$$\frac{\partial\hat{F}}{\partial w_{i,j}^m}=s_i^m a_j^{m-1} \tag{2.10}$$

$$\frac{\partial\hat{F}}{\partial b_i^m}=s_i^m \tag{2.11}$$

结合式 2.7-2.11 最终得到近似最速下降法的表达式的矩阵形式为：

$$\mathbf{W}^m(k+1)=\mathbf{W}^m(k)-\alpha\mathbf{s}^m\left(\mathbf{a}^{m-1}\right)^T \tag{2.12}$$

$$\mathbf{b}^m(k+1)=\mathbf{b}^m(k)-\alpha\mathbf{s}^m \tag{2.13}$$

其中 $\mathbf{s}^m\equiv\frac{\partial\hat{F}}{\partial\mathbf{n}^m}=\left[\frac{\partial\hat{F}}{\partial n_1^m}\quad\frac{\partial\hat{F}}{\partial n_2^m}\quad\cdots\quad\frac{\partial\hat{F}}{\partial n_{s^m}^m}\right]^T$。

2.4 敏感度反向传播

经过上面的讨论我们已经得到了网络信息更新的表达式，现在剩下的问题是如何计算敏感度系数，使网络能够解决实际问题。

首先用 Jacobian 矩阵表示网络 m 层与 m+1 层的传递关系：

$$\frac{\partial \mathbf{n}^{m+1}}{\partial \mathbf{n}^{m}} \equiv \begin{bmatrix} \frac{\partial n_1^{m+1}}{\partial n_1^m} & \frac{\partial n_1^{m+1}}{\partial n_2^m} & \cdots & \frac{\partial n_1^{m+1}}{\partial n_{s^m}^m} \\ \frac{\partial n_2^{m+1}}{\partial n_1^m} & \frac{\partial n_2^{m+1}}{\partial n_2^m} & \cdots & \frac{\partial n_2^{m+1}}{\partial n_{s^m}^m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial n_{s^{m+1}}^{m+1}}{\partial n_1^m} & \frac{\partial n_{s^{m+1}}^{m+1}}{\partial n_2^m} & \cdots & \frac{\partial n_{S^{m+1}}^{m+1}}{\partial n_{s^m}^m} \end{bmatrix} \tag{2.14}$$

考虑上式矩阵中的 i, j 元素：

$$\begin{aligned} \frac{\partial n_i^{m+1}}{\partial n_j^m} &= \frac{\partial\left(\sum_{i=1}^{s^m} w_{i,l}^{m+1} a_l^m + b_i^{m+1}\right)}{\partial n_j^m} = w_{i,j}^{m+1} \frac{\partial a_j^m}{\partial n_j^m} \\ &= w_{i,j}^{m+1} \frac{\partial f^m\left(n_j^m\right)}{\partial n_j^m} = w_{i,j}^{m+1} \dot{f}^m\left(n_j^m\right) \end{aligned} \tag{2.15}$$

其中 $\dot{f}^m\left(n_j^m\right) = \frac{\partial f^m\left(n_j^m\right)}{\partial n_j^m}$，这样可以将式 2.14 的 Jacobian 矩阵改写为：

$$\frac{\partial \mathbf{n}^{m+1}}{\partial \mathbf{n}^{m}} = \mathbf{W}^{m+1} \dot{\mathbf{F}}^m\left(\mathbf{n}^m\right) \tag{2.16}$$

其中：

$$\dot{\mathbf{F}}^m\left(\mathbf{n}^m\right) = \begin{bmatrix} \dot{f}^m\left(n_1^m\right) & 0 & \cdots & 0 \\ 0 & \dot{f}^m\left(n_2^m\right) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \dot{f}^m\left(n_{s^m}^m\right) \end{bmatrix} \tag{2.17}$$

至此，可以使用链式法则写出敏感度系数传递关系的矩阵形式为：

$$\begin{aligned} \mathbf{s}^m &= \frac{\partial \hat{F}}{\partial \mathbf{n}^m} = \left(\frac{\partial \mathbf{n}^{m+1}}{\partial \mathbf{n}^m}\right)^T \frac{\partial \hat{F}}{\partial \mathbf{n}^{m+1}} = \dot{\mathbf{F}}^m\left(\mathbf{n}^m\right)\left(\mathbf{W}^{m+1}\right)^T \frac{\partial \hat{F}}{\partial \mathbf{n}^{m+1}} \\ &= \dot{\mathbf{F}}^m\left(\mathbf{n}^m\right)\left(\mathbf{W}^{m+1}\right)^T \mathbf{s}^{m+1} \end{aligned} \tag{2.18}$$

从上式可以看到，反向传播将敏感度信息从输出层逆着数据流的方向传递。最后要做的是从输出层 s^M 出发，推到完整的反向传播公式，具体过程如下。

$$s_i^M = \frac{\partial \hat{F}}{\partial n_i^M} = \frac{\partial (\mathbf{t}-\mathbf{a})^T (\mathbf{t}-\mathbf{a})}{\partial n_i^M} = \frac{\partial \sum_{i=1}^{S^M} \left(t_j - a_j\right)^2}{\partial n_i^M} = -2\left(t_i - a_i\right) \frac{\partial a_i}{\partial n_i^M} \tag{2.19}$$

由于：

$$\frac{\partial a_i}{\partial n_i^M}=\frac{\partial a_i^M}{\partial n_i^M}=\frac{\partial f^M\left(n_i^M\right)}{\partial n_i^M}=\dot{f}^M\left(n_i^M\right) \tag{2.20}$$

式 2.19 可改写成矩阵形式为：

$$\mathbf{s}^M=-2\dot{\mathbf{F}}^M\left(\mathbf{n}^M\right)(\mathbf{t}-\mathbf{a}) \tag{2.21}$$

2.5 网络更新

总结反向传播算法的流程。

✧ 正向传播得到网络输出：

$$\mathbf{a}^0=\mathbf{p} \tag{2.22}$$

$$\mathbf{a}^{m+1}=\mathbf{f}^{m+1}\left(\mathbf{W}^{m+1}\mathbf{a}^m+\mathbf{b}^{m+1}\right)\ \text{for}\ m=0,1,\ldots,M-1 \tag{2.23}$$

$$\mathbf{a}=\mathbf{a}^M \tag{2.24}$$

✧ 敏感度反向传播：

$$\mathbf{s}^M=-2\dot{\mathbf{F}}^M\left(\mathbf{n}^M\right)(\mathbf{t}-\mathbf{a}) \tag{2.25}$$

$$\mathbf{s}^m=\dot{\mathbf{F}}^m\left(\mathbf{n}^m\right)\left(\mathbf{W}^{m+1}\right)^T\mathbf{s}^{m+1},\ \text{for}\ m=M-1,\ldots,2,1 \tag{2.26}$$

✧ 近似最速下降法更新网络的权值及偏置：

$$\mathbf{W}^m(k+1)=\mathbf{W}^m(k)-\alpha\mathbf{s}^m\left(\mathbf{a}^{m-1}\right)^T \tag{2.27}$$

$$\mathbf{b}^m(k+1)=\mathbf{b}^m(k)-\alpha\mathbf{s}^m \tag{2.28}$$

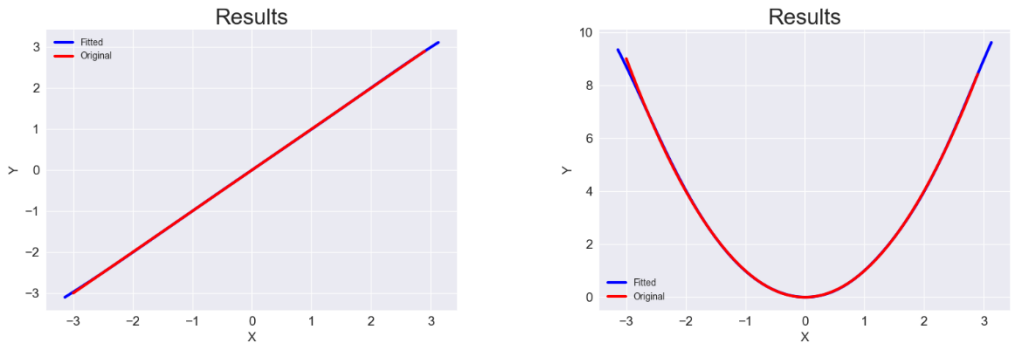
3 程序验证

本节为算法及程序的验证部分，用以检验算法的合理性及自编程序的正确性。选用两类函数（幂函数、三角函数）拟合验证。

网络结构为 1-6-1 网络，隐藏层有 6 个神经元，传递函数为 log-sigmoid；输出层有 1 个神经元，传递函数为 linear；初始的权值及偏置设为-1 到 1 的随机值；学习率设为 0.01。

3.1 幂函数验证

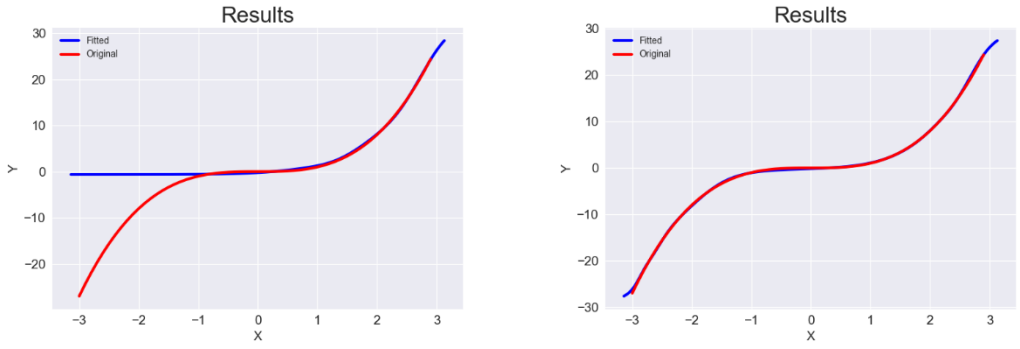
首先验证幂函数的拟合效果，选用指数 1（线性）、2、3 测试。其中线性函数和二次函数的结果如图 2.2 所示。



(a) 线性函数　　(b) 二次函数

图 2.2 幂函数拟合结果

可以看到 6 个神经元的双层网络可以较好的拟合前两阶的幂函数。但对于三次幂，该网络的拟合效果较差，如图 2.3(a)，可以看到在负半轴的拟合趋势错误。这里更改网络结构，采用三层网络，每个隐藏层设 8 个神经元（1-8-8-1），其余设置不变，得到的结果如图 2.3(b)所示，可以看到拟合结果较好。但在训练集外的拟合结果趋势并不正确，这可能是激活函数本身性质或训练轮数不足导致的。

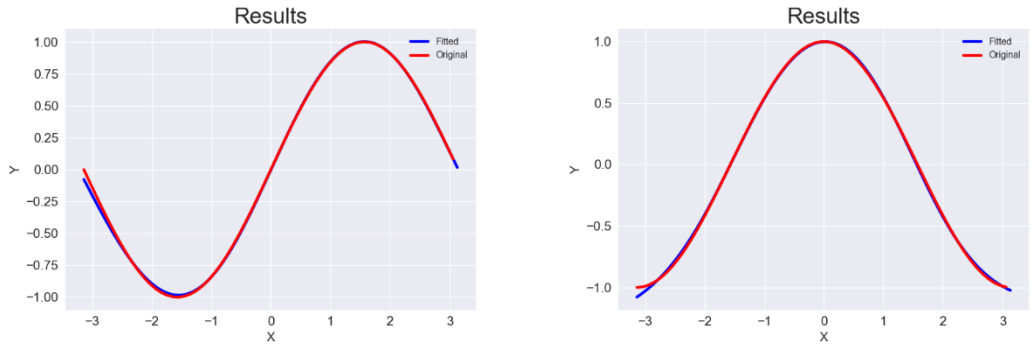


(a) 原网络拟合结果　　(b) 新网络拟合结果

图 2.3 三次幂拟合结果对比

3.2 三角函数验证

三角函数的拟合结果如图 2.4 所示。可以看到 6 神经元的双层网络能够较好的拟合三角函数。但不能忽视的是，在定义域边缘的点，网络拟合结果较差。



(a) sin(x)拟合结果　　(b) cos(x)拟合结果

图 2.4 三角函数拟合结果

三、催化时间预测

1 计算流程图

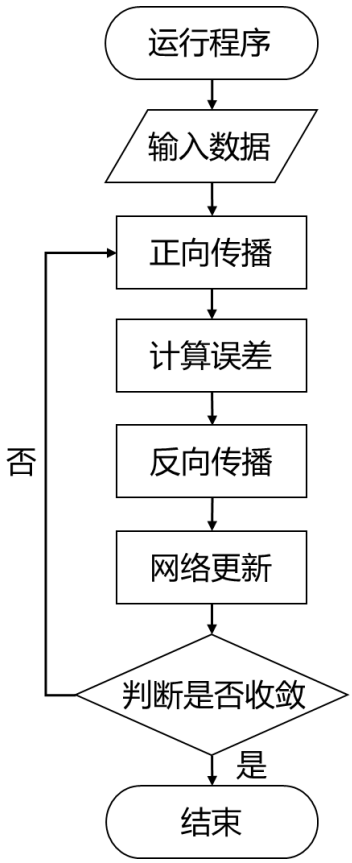


图 3.1 程序计算流程图

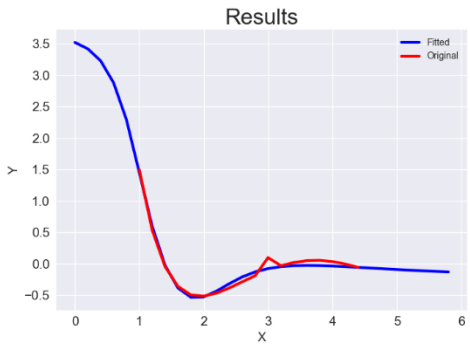
2 网络构建

取 MnO_2 含量为横坐标，催化时间为纵坐标绘制函数，采用“1-8-8-1”型网络（三层网络，每个隐藏层 8 个神经元）对所给数据进行拟合。使用不同的迭代轮数重复计算，找到拟合结果较好的情况。

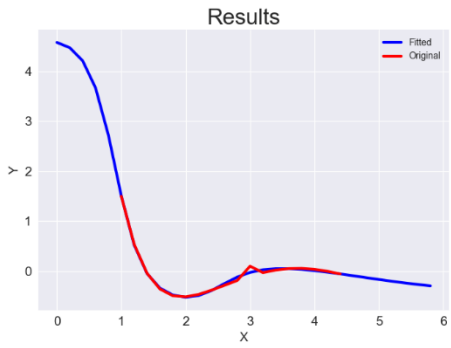
网络各层的权值及偏置设置为-1 到 1 的随机值；网络学习率设为 0.05；隐藏层激活函数为 log-sigmoid；输出层传递函数为 linear。网络各层的详细信息可见程序执行后生成的“outfile.txt”文件。

3 计算结果

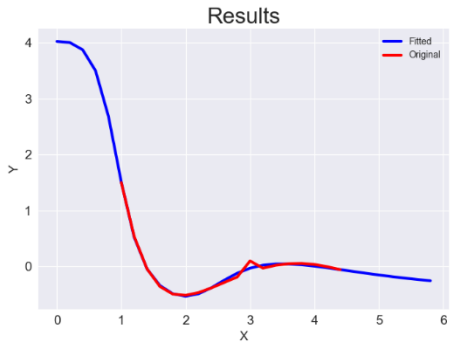
“1-8-8-1”型为三层网络，隐藏层为两层。测试了 epoch=2000:10000:2000（min=2000，max=10000，step=2000）的几种情况，由于网络随机初始化不能保证两次计算的结果完全相同，这里选择多次测试结果中几个典型的拟合结果展示，详见图 3.2。



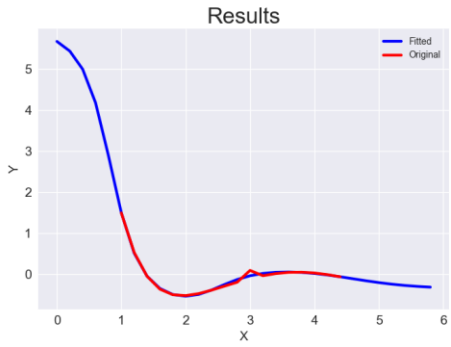
(a) epoch=2000



(b) epoch=4000



(c) epoch=8000



(d) epoch=10000

图 3.2 1-8-8-1 网络测试结果对比

对比图 3.2 的各个结果，最终选择 epoch=10000 的结果，并将其整理到表 3.1。

表 3.1 催化时间预测

MnO_2(g)	**4.6**	**4.8**	**5.0**	**5.2**	**5.4**	**5.6**
催化时间(h)	-0.103	-0.152	-0.196	-0.234	-0.265	-0.289

4 总结与讨论

最后对本文的内容进行简单的总结及不足之处的讨论：

✧ 总结：

1：分析了题目的问题，给出了使用 BP 算法拟合的解决方案；

2：简要介绍的 BP 算法的原理；

3：使用自编 BP 算法程序对题目问题进行求解，给出结果。

✧ 不足及改进：

1：使用近似最速下降法更新网络，网络训练速度较慢。以后可以考虑使用启发式方法或者数值优化方法进行改进；

2：每次执行程序网络信息都会随机设置，导致训练结果出现差异。考虑添加网络参数读写功能；

3：使用 S 形激活函数，使部分无法通过 S 形函数线性组合来拟合的函数的拟合效果较差，特别是在边界位置的导数不连续。考虑改进激活函数拓展程序的拟合能力。

附件、Python 程序

见项目源码。