机器学习第一次作业——神经网络BP算法的回归与分类

统计81 于越 *

April 28, 2021

Abstract

神经网络一词，在不同学科中的定义是有出入的，而在机器学习中，一般采用最为广泛的一种理解，即“神经网络是由具有适应性的简单单元组成的广泛并行互联的网络，其组织能够模拟生物神经系统对真实世界物体所做出的交互反应”。作为神经网络中最经典的算法之一，BP算法被广泛应用于神经网络尤其是多层前馈神经网络的训练，而研究者主要通过训练完成的神经网络进行数据集的回归、拟合与分类等工作。在本文中，我主要给出了神经网络BP算法的具体的两种推导形式，并且构建了一个较为简易的单层的神经网络进行BP算法的实例演示，主要根据给出的男性大学生的身高体重数据计算身体质量指数（BMI）的公式，并用神经网络对于这一实例进行了数据的训练与测试，通过比较拟合结果与真实值之间产生的误差研究神经网络BP算法的优势。

1 关于激活函数（Sigmoid函数）：

神经网络中最为基本的单元是神经元。神经网络主要由输入层、输出层和隐层组成，当前层的神经元需要接受来自上一层的神经元传递而来的信号，再通过神经元中的“激活函数”来产生当前层神经元的输出。最典型的神经元激活函数便是Sigmoid函数：

$$f(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

Sigmoid函数在0的邻域外增长较为平缓，在0附近的变化趋于陡峭，且其具有较好的导数性质：

$f'(x) = f(x)[1 - f(x)]$ 这有利于后续BP算法的推导以及神经网络深度的增加。

*学号：2183210516

2 关于具体的“学习”：

神经网络主要需要的是三类数据集：用来给神经网络训练和学习的数据集，用于验证的数据集以及用于测试的数据集。关于训练数据集，我们需要动用这一数据集来调整神经元之间的“连接权”以及每个神经元功能的阈值，即神经网络“所学内容”蕴含在连接权与阈值之中。我们主要通过“隐层”来传递连接权与阈值，以达到不断训练、迭代的目的。

3 关于BP算法：

BP算法，即误差逆传播算法（BackPropagation），是神经网络中的经典算法，主要通过双向推导与验证的方式，正向通过连接权进行计算，反向通过正向所得的计算值与实际值进行误差比较的方式不断修改神经网络中的连接权以减小这一误差。当这一误差通过多次训练小于某一阈值时，即完成BP算法，可以给出最后的数据拟合。下面给出BP算法的推导过程。不妨假设这一神经网络的结构是多层的。在这里有两种推导方式，一种是不计入激活函数单单对于$z_j = \sum_i w_{ij}x_i$的推导（激活函数σ内部的线性加和部分，未经过激活函数Sigmoid函数的处理），另一种是计入了激活函数对于$a_j = \sigma(\sum_i w_{ij}x_i)$的推导：首先，不妨设$f_w(x) = \sigma(\sum_i w_i x_i)$，为了便于后文中误差函数的书写。在这里，我们主要采用均方误差的形式：$E(D) = \sum_n (f_w(x_n) - y_n)^2$. 先给出最后一层的对于$z_j$的推导：

$$\frac{\partial E}{\partial w_i} = \sum_n 2[f_w(x_n) - y_n]\frac{\partial f_w(x_n)}{\partial z}\frac{\partial z}{\partial w_i} = \sum_n 2[f_w(x_n) - y_n]\sigma(z)[1-\sigma(z)]x_i \tag{2}$$

再从后向前依次进行推导。若将通过神经网络的计算函数写出如下形式：

$$f_w(x) = \sigma(\ldots\sigma(\sum_j w_{jk}\sigma(\sum_i w_{ij}a_i))) \tag{3}$$

(将a_i看作内部已经通过运算得到的值).则有

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial z_j}\frac{\partial z_j}{\partial w_{ij}} = \frac{\partial E}{\partial z_j}a_i \tag{4}$$

而

$$\frac{\partial E}{\partial z_j} = \sum_k \frac{\partial E}{\partial z_k}\frac{\partial z_k}{\partial z_j} = \sum_k \frac{\partial E}{\partial z_k}\frac{\partial z_k}{\partial a_j}\frac{\partial a_j}{\partial z_j} = \sum_k \frac{\partial E}{\partial z_k}w_{jk}\sigma(z_j)[1-\sigma(z_j)] \tag{5}$$

后续的偏导数也类似于上述推导层层推进即可。下面再给出关于$a_j = \sigma(\sum_i w_{ij}x_i)$的相关推导：函数先给出最后一层的对于$a_j$的推导，结果与上面的推导类似：

$$\frac{\partial E}{\partial w_i} = \sum_n 2[f_w(x_n) - y_n]\frac{\partial f_w(x_n)}{\partial a_j}\frac{\partial a_j}{\partial z_i}\frac{\partial z_i}{\partial w_i} = \sum_n 2[f_w(x_n) - y_n]\sigma(z_i)[1-\sigma(z_i)]x_i \tag{6}$$

仍旧仿照上述过程从后向前进行推导：若将通过神经网络的计算函数写出如下形式：

$$f_w(x) = \sigma(\dots\sigma(\sum_j w_{jk}\sigma(\sum_i w_{ij}a_i))) \tag{7}$$

则有

$$\frac{\partial E}{\partial a_j} = \sum_k \frac{\partial E}{\partial a_k}\frac{\partial a_k}{\partial a_j} = \sum_k \frac{\partial E}{\partial a_k}\frac{\partial a_k}{\partial z_j}\frac{\partial z_j}{\partial a_j} = \sum_k \frac{\partial E}{\partial z_k}w_{jk} = \sum_k \frac{\partial E}{\partial a_k}w_{jk}\sigma(z_k)[1-\sigma(z_k)] \tag{8}$$

而若我们再在上述基础上再添加一个隐层，即得到：

$$f_w(x) = \sigma(\dots\sigma(\sum_l w_{kl}\sigma(\sum_j w_{jk}\sigma(\sum_i w_{ij}a_i)))) \tag{9}$$

则

$$\frac{\partial E}{\partial a_j} = \sum_l \frac{\partial E}{\partial a_l}\frac{\partial a_l}{\partial a_k}\frac{\partial a_k}{\partial a_j} = \sum_l \frac{\partial E}{\partial a_l}\frac{\partial a_l}{\partial a_k}\frac{\partial a_k}{\partial z_k}\frac{\partial z_k}{\partial a_j} = \sum_l \frac{\partial E}{\partial a_l}w_{kl}\sigma(z_k)[1-\sigma(z_k)]w_{jk} \tag{10}$$

最后在代入下式即可：

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial a_j}\frac{\partial a_j}{\partial w_{ij}} = \frac{\partial E}{\partial a_j}\sigma(zj)[1-sigma(z_j)]x_i \tag{11}$$

BP算法需要被学习率$\eta \in (0,1)$所控制，而学习率可以通过控制训练次数，学习速率和训练最小误差以实现。学习率过大或过小都是不合适的。详细算法步骤伪代码如下:

Algorithm 1 神经网络BP算法

Input: 训练集和测试集 $D = (x_k, y_k)_{k=1}^m$.学习率，包含训练次数epoch，学习速率lr，训练目标最小误差goal

Initialize:

对数据归一化处理

1: **while** 当前样本输出与当前参数误差≥设定好的误差阈值 **do**

2: 计算输出层神经元和因曾神经元的梯度项，并更新连接权和阈值

3: **end while**

4: **Output:** 连接权与阈值确定的神经网络

4 神经网络BP算法对于男大学生身体质量指数的回归拟合

4.1 结果呈现

BMI，即身体质量指数，是用来衡量一个人身体状况和身材是否匀称的重要指标。在这里，我根据20组男大学生的身高与体重数据，计算出他们的BMI指数，再通过神

经网络BP算法进行拟合，以测试BP算法的效果。这20组男大学生身高分布在(1.5,2)(单位：米)之间，体重在(50,100)(单位：千克)之间。BMI计算公式如下：

$$BMI = \frac{weight}{height^2} \tag{12}$$

首先，对于这20组数据，我们直接用全部的20组数据进行训练，再进行测试。在这里，我构建的是一个较为简单的单隐层的神经网络，由输入层，一层隐层和输出层构成。一层隐层上我赋予了10个隐藏节点进行连接权的计算。首先，对于身高和体重数据，二者之间数量级相差过大，如果直接处理容易因为量纲导致拟合不准确，故我们对数据进行归一化处理，将数据依大小分布在(0,1)之间。关于学习率方面，我设定了训练次数为50000次，训练速率为0.035，训练目标的最小误差为$6.5 * 10^{-3}$.在这里，为了保证拟合的准确性，我还引入了一个噪声强度的修正项，以防止过拟合的状况出现。接着，我们要初始化各层之间的连接权值以及算法训练目标的阈值。用一个能量函数的形式对误差(我选用的是误差的平方和形式)进行计算，进而与设定的训练目标最小误差进行比较。如果误差大于训练的目标误差，就利用之前推导的函数负梯度下降原理进行连接权值与阈值的更新。(更新过程中两层的连接权见下图)达到训练目标后，这一网络便可投入测试与应用了。值得一提的是，在这里我隐层和输出层之间的激活函数采用了常函数而不是Sigmoid函数，具体的解释可以见下文4.2。这里给出了对于这20组数据的测试拟合结果(见图一)，可以发现拟合结果还是很不错的。与此同时，我还给出了误差平方和在训练神经网络过程中的变化图。对于这50000次训练，我只画出了前100次训练的误差数据。发现前10次训练的误差还是比较大的，但是10次之后的误差就趋于平稳，且几乎为零。这说明经过多次训练，这一神经网络对于这些实验数据的拟合性能较为不错。(代码见附录)

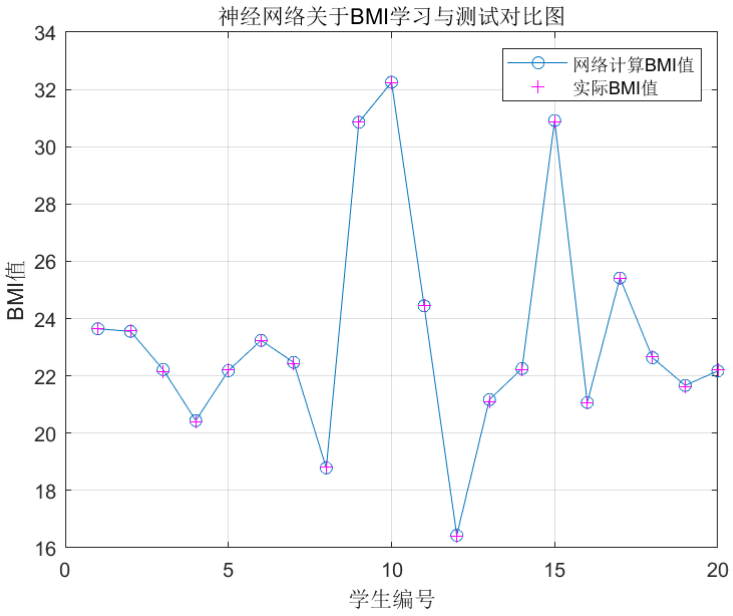

Figure 1: result of BP algorithm

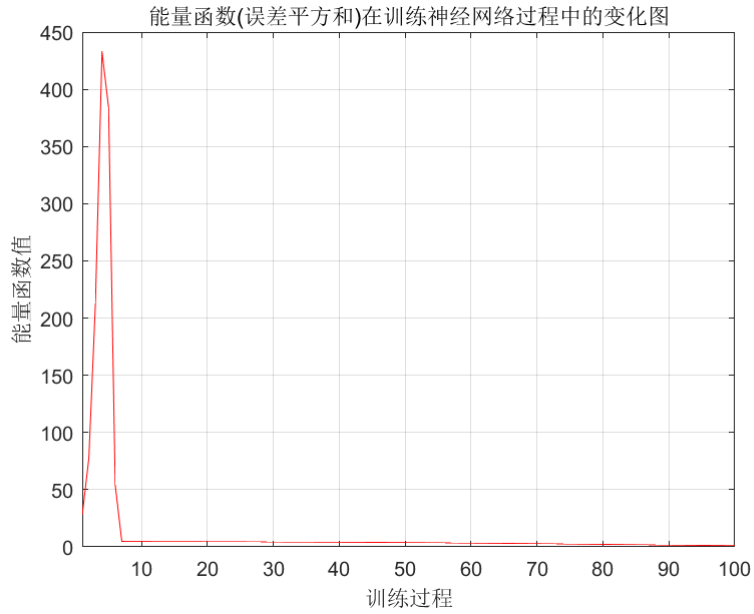

Figure 2: error of BP algorithm train

1	2
0.1023	0.1444
-0.0617	0.1444
0.0162	0.0720
-0.0426	0.3541
-0.0120	0.0878
0.0178	-0.0427
0.1056	0.2925
-0.0760	0.0955
0.3473	0.0250
0.3725	0.1019

Figure 3: 连接权W1的更新值

1	2	3	4	5	6	7	8	9	10
-0.4398	-0.3772	-0.2140	-0.1689	-0.1806	-0.2336	-0.1987	-0.3345	-0.1613	-0.3647

Figure 4: 连接权W2的更新值

1	2
0.0398	-0.0306
0.0079	-0.0061
-0.1074	0.0897
-0.1210	0.1166
-0.1116	0.0898
-0.0635	0.0498
-0.0858	0.0685
-0.0244	0.0191
-0.1162	0.1174
0.0028	-0.0022

Figure 5: 连接权W1的更新值(偏导数)

1	2	3	4	5	6	7	8	9	10
-10.3222	-10.3337	-12.5299	-12.4205	-11.5565	-10.2587	-10.6343	-10.9326	-12.3899	-10.2635

Figure 6: 连接权W2的更新值(偏导数)

4.2 关于隐层与输出层之间激活函数选择的讨论

值得一提的是，在这个简单的三层神经网络中，从隐层到输出层的神经网络激活函数我并没有选择一个Sigmoid函数，而是单单应用了一个常函数去进行激活。理论上按照前文的推导，激活函数确实应该使用Sigmoid函数的形式，但是在这个例子中若隐层到输出层采用Sigmoid函数，拟合的结果会相当糟糕(见下图)。深入思考会导致这一结果的原因，我认为问题出现在前面的"归一化与反归一化"的问题上。因为在这里我对于身高、体重以及BMI这些数据的归一化采用的是matlab自带的premnmx归一化与postnmnx反归一化函数(详情可以参见附录代码)，也就是

$$result = 2 * \frac{m - min}{max - min} - 1; \tag{13}$$

这样的归一化方法主要是一个线性的形式(可以类比数理统计中的均匀分布)，而Sigmoid函数本身带有指数的形式在其中，如果最后一层的激活函数仍旧采用Sigmoid函数的形式，反归一化函数没有提供对应的类似logistic回归函数的对数形式，就会导致测试数据的拟合结果十分糟糕。可以看到，在最后一层激活函数采用了Sigmoid函数的情形下，前面的8个拟合值近乎相等，这应该是受到了Sigmoid激活函数的影响，使得他们的数量级近乎等同。故这里的最后一层激活函数我采用了一个简单的常函数，就是为了与归一化、反归一化的形式对应起来。

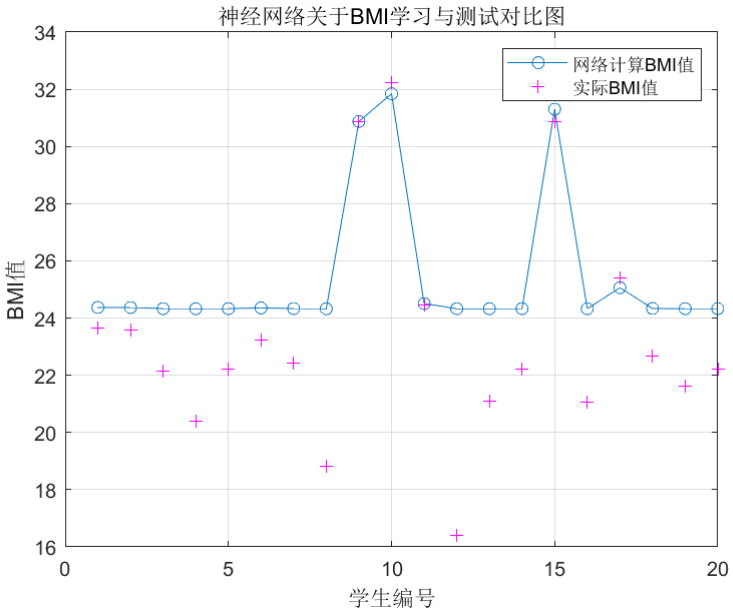

Figure 7: result of BP algorithm(最后一层激活函数采用Sigmoid函数)

5 分类问题——logit函数用于分类

5.1 理论推导

logit函数回归，又称对数几率回归，其思想和理论基础在于，原先的分类判别函数是一个阶跃函数，不连续，故难以进行将函数求逆的操作

$$y = g^{-1}(w^{\mathrm{T}}x+b) \tag{14}$$

因此，我们引入sigmoid函数$y = \frac{1}{1+e^{-z}}$作为原先的单位阶跃函数的替代，将原先的z值变化为(0,1)上的连续可微的函数。故我们有

$$y = \frac{1}{1+e^{-(w^{\mathrm{T}}x+b)}} \tag{15}$$

上式可以变换为

$$ln\frac{y}{1-y} = w^{\mathrm{T}}x+b \tag{16}$$

此时的$\frac{y}{1-y}$即可理解为几率。而$ln\frac{y}{1-y}$也就成为了对数几率，即logit。

事实上，将y视为后验概率估计$p(y=1|x)$，则有

$$ln\frac{p(y=1|x)}{p(y=0|x)} = w^{\mathrm{T}}x+b \tag{17}$$

因此显然有

$$p(y=1|x) = \frac{e^{w^{\mathrm{T}}x+b}}{1+e^{w^{\mathrm{T}}x+b}} \qquad p(y=0|x) = \frac{1}{1+e^{w^{\mathrm{T}}x+b}} \tag{18}$$

故采取极大似然估计，可得对数似然为

$$l(w,b)=\sum_{i=1}^{m} lnp(y_i|x_i;w,b) \tag{19}$$

若将参数空间(w,b)即为β，我们采用乘法公式的理论，可以将似然项写为

$$p(y_i|x_i;w,b)=y_i p_1(\hat{x_i};\beta)+(1-y_i)p_0(\hat{x_i};\beta) \tag{20}$$

故对数似然可以写为

$$l(\beta)=\sum_{i=1}^{m}(-y_i\beta^{\mathrm{T}}\hat{x_i}+ln(1+e^{\beta^{\mathrm{T}}\hat{x_i}})) \tag{21}$$

故接下来最小化这里的$l(\beta)$以求解β即可。可以采用牛顿法或梯度下降法等方法得出迭代式。

5.2 具体实验

在这里，我自行生成了横纵坐标均在(0,1)区间内的300个点，其中240个点作为训练集，60个点作为测试集进行实验。对于训练集获得的损失函数与分类结果如下图所示：

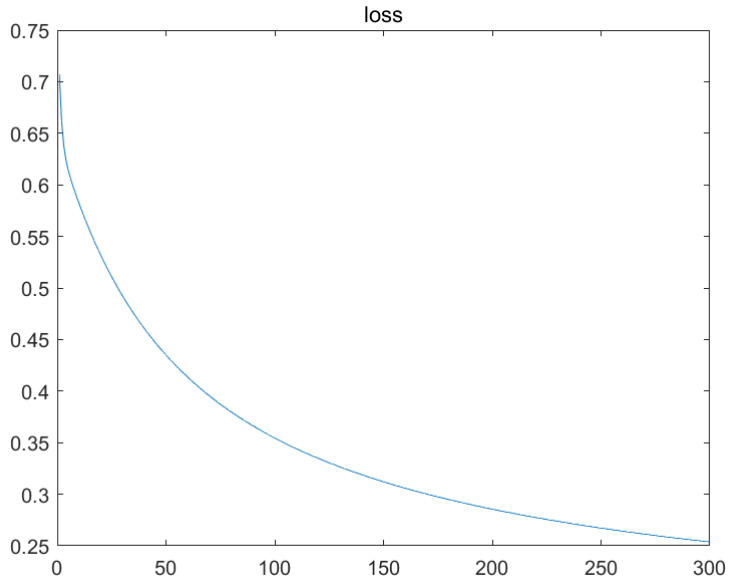

Figure 8: logit回归应用于分类时的损失函数

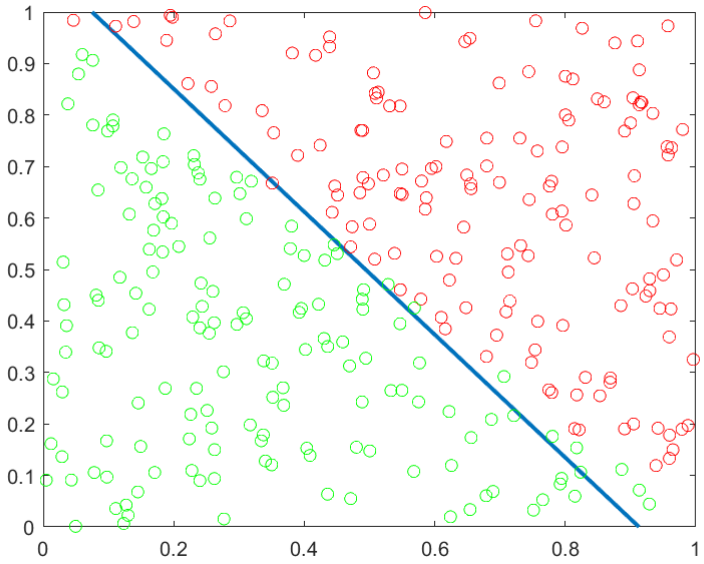

Figure 9: logit回归应用于训练集分类结果

可以看到此时的分类效果还是不错的，除了极少数的点产生了分类错误情况。我们再应用于测试集，判断训练的准确率如下所示：

```
ans =

    0.9833
```

Figure 10: logit回归应用于测试集的准确率

准确率达到了98.3%，结果还是很不错的。

6 有关BP算法的一些思考：类比多元数据分析线性回归理论

神经网络BP算法与传统的多元数据分析中的线性回归理论都是用来对于源数据进行参数估计与数据拟合的方法，二者之间存在着诸多的异同点，揭示了变量间的内在规律，可以应用于预测与控制等问题。

6.1 线性回归模型可以看成是神经网络的简化

在线性回归模型中，某个变量Y往往与另外一些变量$X_1,X_2,\ldots,X_{p-1}$之间存在一定的相关关系，但事实上这种关系并不是明确的，只能说是变量$X_1,X_2,\ldots,X_{p-1}$取值部分

决定了Y的取值。我们不妨将变量X_1,X_2,...,X_{p-1}这些看成是神经网络的输入端，将Y看成是神经网络的输出端，而线性回归模型的输出结果往往是

$$Y = \beta_0 + \beta 1 X_1 + \ldots + \beta_{p-1} X_{p-1} + \epsilon \tag{22}$$

的形式，我们可以将这一形式看成是经过数据训练得出的拟合函数的形式。BP算法中有着许多的连接权，我们可以把上面线性回归模型估计式中的每一个参数β类比为连接权，即通过这一"连接权"联系每一个变量进而得出我们想要的结果。而上述这些操作我们需要一定的数据支撑，将这些数据给看成是所谓的用来进行训练的训练数据。与此同时，在机器学习的学习机结果中存在着防止过拟合出现的误差噪声项，我们也可以将其与线性回归模型中的误差项ϵ给联系起来。这样，除了函数形式上线性回归模型更加局限于线性之外，别的方面都可以相互联系起来。

6.2 线性回归模型矩阵形式的参数估计与BP算法的连接权

线性回归模型本质上是一个参数估计的方法，主要是将数据写成矩阵形式，再利用高等代数与矩阵分析的相关理论简化计算，得出相应线性方程参数、常数项以及误差项的估计。而BP算法主要是通过偏导数的链式法则理论结合学习率不断进行连接权的更新以减小误差，从而达到拟合的效果。这两个看似没有关系，但在我看来还是有着一些联系的。首先，两种方法都是要在一定的限制下使得误差项达到最小，这在算法上是具有联通性的，而且二者用以减小误差的方法也有类似之处。首先是误差函数上，BP算法的学习机应用了$E(D) = \sum_n (f_w(x_n) - y_n)^2$形式的能量函数作为误差函数，也就是类似于最小二乘均方误差的形式，而线性回归模型对于回归参数系数β的最小二乘估计也是选择了误差平方和

$$S(\beta) = \sum_{i=1}^{n} \epsilon_i^2 = \epsilon^T \epsilon = (Y - X\beta)^T (Y - X\beta) = \sum_{i=1}^{n} (y_i - \sum_{j=0}^{p-1} \beta_j x_{ij})^2 \tag{23}$$

的形式以使误差达到最小。于是，在线性回归模型中也类似地有求偏导的操作：

$$\frac{\partial(S(\beta))}{\partial \beta_k} = -2 \sum_{i=1}^{n} (y_i - \sum_{j=0}^{p-1} \beta j x_{ij}) x_{ik} = 0, k = 0, 1, \ldots, p-1 \tag{24}$$

即

$$\sum_{i=1}^{n} y_i x_{ik} = \sum_{i=1}^{n} \sum_{j=0}^{p-1} \beta_j x_{ij} x_{ik} = \sum_{j=0}^{p-1} (\sum_{i=1}^{n} x_{ij} x_{ik}) \beta_j \tag{25}$$

也就转化成了矩阵方程$X^T X \beta = X^T Y$.因此偏导链式法则的方法用在了线性回归模型参数估计中最重要的转化矩阵方程的一步，二者的联系体现在了这里。

在上面也提到了神经网络BP算法中的连接权类似于线性回归模型中的回归系数，因此上面的求偏导的步骤类比于神经网络中最后一层隐层到达输出层的操作。而在转化为矩阵方程的过程中，我们要最小化误差项的过程也很大程度地体现在了系数β上，因此最小化误差的过程与BP算法中更新连接权的过程也有一定相似之处。

6.3 神经网络“回归”线性

神经网络可以进行线性的回归拟合，但所能做的工作是远不限于线性的回归拟合的。而相比之下线性回归模型的工作模式就较为单一了，这也导致了线性回归模型很大程度上所存在的局限性。换句话说，我们可以把线性回归模型想象成“低维”的方法，而神经网络是相对“高维”的方法。但是正如数学分析和概率分布中有着能将高维“退化”成低维的方法，我认为在神经网络中也可以有类似的操作。因为神经网络BP算法中我们应用到了激活函数Sigmoid函数，这是对于网络连接权的独特处理。因此，借鉴这种模式，我们对于各种各样的数据，我们也可以做类似的“中心化”亦或是“正则化”的操作，并且结合次序统计量，将数据的分布尽可能向线性靠拢。在这样的基础上，我们就可以对这些数据进行线性操作，最后再将数据还原，得到拟合的结果。当然这还仅仅是我个人的想法，能否真正实现还有待考证。

但神经网络与回归分析的联系应该还远不止于此。神经网络还可以用来进行目标数据的分类，这其中就有着基于对数函数的logit方法(对数几率回归)。这与回归分析中的logistic回归模型有着许多的联系，就例如二者模型在公式上都有着下述的形式：

$$ln(\frac{E(Y)}{1-E(Y)}) = \beta_0 + \sum_{j=1}^{p-1} \beta_j X_j \tag{26}$$

进而

$$E(Y) = \frac{exp(\beta_0 + \sum_{j=1}^{p-1} \beta_j X_j)}{1 + exp(\beta_0 + \sum_{j=1}^{p-1} \beta_j X_j)} \tag{27}$$

同样地，这两个方法也都有着迭代与极大似然的思想进行深入。在神经网络的logit方法中，主要是将结果转化为Sigmoid函数这样的点概率分布再进行极大似然，而logistic回归模型则是主要利用Fisher信息矩阵进行参数的极大似然估计。神经网络的logit模型运用于分类问题时会使用迭代，而logistic回归模型中也用到了Newton-Raphson迭代解法(类似于数值分析newton法的推广)。故二者之间确实有联系可言，还有许多仍等待着去挖掘。

7 总结

神经网络BP算法，采用正向计算连接权与逆向误差逆传导的方式，训练出达到训练目标的神经网络，应用于训练数据后对于数据的回归与分类。随着机器学习的不断发展，延伸出了深度学习中的卷积神经网络，更易于捕捉数据的特征进行拟合，大大减少了需要训练的参数数目。总之，神经网络仍有着极大的研究潜力等待我们去发掘。

A 相关代码

```
numberOfSample = 20; %输入样本数量
%取测试样本数量等于输入训练集()样本数量，因为输入样本
    （训练集）容量较少，否则一般必须用新鲜数据进行测试

numberOfTestSample = 20;
numberOfHiddenNeure =10;
inputDimension = 2;
outputDimension = 1;


%准备好训练集

numberOfHeight=[1.72 1.76 1.97 1.84 1.67 1.82
 1.90 1.77 1.59 1.68 1.74 1.78 1.86 1.91
  1.64 1.81 1.73 1.68 1.85 1.71];
numberOfWeight=[70 73 86 69 62 77 81 59
78 91 74 52 73 81 83 69 76 64 74 65];
BMI=[];
for i=1:20
BMI(i)=numberOfWeight(i)/(numberOfHeight(i)).^2;
end

input=[numberOfHeight;numberOfWeight];%目标输入矩
    阵
output=BMI;%目标输出矩阵

%对训练集中的输入数据矩阵和目标数据矩阵进行归一化处理
[sampleInput, minp, maxp, tmp, mint, maxt]
= premnmx(input, output);

%噪声强度
noiseIntensity = 0.01;
%利用正态分布产生噪声
noise = noiseIntensity * randn
(outputDimension, numberOfSample);
%给样本输出矩阵添加噪声，防止网络过度拟合tmp
```

```
sampleOutput = tmp + noise;

%取测试样本输入输出()与输入样本相同，因为输入样本（训练
    集）容量较少，否则一般必须用全新数据进行测试，以确保
    神经网络应用的广泛性



testSampleInput = sampleInput;
testSampleOutput = sampleOutput;

%最大训练次数
maxEpochs = 50000;

%网络的学习速率
learningRate = 0.035;

%训练网络所要达到的目标误差
error0 = 0.65*10^(-3);

%初始化输入层与隐含层之间的权值为防止开始权值过大(难以调
    整，故先行乘以系数，
0.5%维数W110*2)
W1 = 0.5 * rand(numberOfHiddenNeure
, inputDimension) ;
%初始化输入层与隐含层之间的阈值，维数B110*1
B1 = 0.5 * rand(numberOfHiddenNeure, 1) ;
%初始化输出层与隐含层之间的权值，维数W21*10
W2 = 0.5 * rand(outputDimension
, numberOfHiddenNeure) ;
%初始化输出层与隐含层之间的阈值，维数B21*1
B2 = 0.5 * rand(outputDimension, 1) ;

%保存能量函数误差平方和()的历史记录
errorHistory = [];

for i = 1:maxEpochs
%隐含层输出
```

```
hiddenOutput = logsig(W1 * sampleInput +
repmat(B1, 1, numberOfSample));
%输出层输出这里默认取了最后一层的激活函数为常数，(不然效
    果较为糟糕
)
networkOutput = W2 * hiddenOutput
+ repmat(B2, 1, numberOfSample);
%实际输出与网络输出之差
error = sampleOutput - networkOutput;
%计算能量函数误差平方和()
E = sumsqr(error);
errorHistory = [errorHistory E];

if E < error0
break;
end

%以下依据能量函数的负梯度下降原理对权值和阈值进行调整
delta2 = error;%维数为1*20
delta1 = W2' * delta2.*hiddenOutput.*(1 -
    hiddenOutput);
%这里的激活函数采用常函数而非(函数，维
    数Sigmoiddelta110*20)
dW2 = delta2 * hiddenOutput';
dB2 = delta2 * ones(numberOfSample, 1);

dW1 = delta1 * sampleInput';
dB1 = delta1 * ones(numberOfSample, 1);
W2 = W2 + learningRate * dW2;
B2 = B2 + learningRate * dB2;
W1 = W1 + learningRate * dW1;
B1 = B1 + learningRate * dB1;
end
%下面对已经训练好的网络进行仿真()测试
%对测试样本进行处理
testHiddenOutput = logsig(W1 * testSampleInput
+ repmat(B1, 1, numberOfTestSample));
```

```
testNetworkOutput = W2 * testHiddenOutput
+ repmat(B2,1, numberOfTestSample);
%还原网络输出层的结果反归一化()
a = postmnmx(testNetworkOutput, mint, maxt);
%绘制测试样本神经网络输出和实际样本输出的对比图
(figure(1))
```

```
t = 1:20;
%测试样本网络输出值BMI
a1 = a(1,:);
figure(1);
plot(t,a1,'o-',t,BMI,'m+');
legend('网络计算值BMI','实际值BMI');
xlabel('学生编号');ylabel('值BMI');
title('神经网络关于学习与测试对比图BMI');
grid on;
%观察能量函数误差平方和()在训练神经网络过程中的变化情况
```

```
figure(3);
n = length(errorHistory);
t3 = 1:n;
plot(t3, errorHistory, 'r-');
%为了更加清楚地观察出能量函数值的变化情况，这里我只绘制
    前次的训练情况
100
xlim([1 100]);
xlabel('训练过程');
ylabel('能量函数值');
title('能量函数误差平方和()在训练神经网络过程中的变化
    图');
grid on;对数几率回归应用于分类：
```

```
logit
```

```
clear
clc

%% 数据准备
%二分类 随机生成数据。个数据 300 每个数据个特征 2
data=1*rand(300,2);
label=zeros(300,1);
%label(sqrt(data(:,1).^2+data(:,2).^2)<8)=1;
label((data(:,2)+data(:,1)>1))=1;
%在上加常数特征项; data
data=[data,ones(size(data,1),1)];

%打乱顺序
randIndex = randperm(size(data,1));
data_new=data(randIndex,:);
label_new=label(randIndex,:);

%训练80% 测试 20%
k=0.8*size(data,1);
X1=data_new(1:k,:);
Y1=label_new(1:k,:);
X2=data_new(k+1:end,:);
Y2=label_new(k+1:end,:);

[m1,n1] = size(X1);
[m2,n2] = size(X2);
Features=size(data,2); %特征个数
%% 开始训练
%设定学习率为0.01
delta=1;
lamda=0.2; %正则项系数

theta1=rand(1,Features);
%theta1=[.5,.5];
%训练模型%

%梯度下降算法求解（每次都是对全部的数据进行训练）theta
```

```
num = 300; %最大迭代次数
L=[];
while(num)
dt=zeros(1,Features);
loss=0;
for i=1:m1
xx=X1(i,1:Features);
yy=Y1(i,1);
h=1/(1+exp(-(theta1 * xx')));
dt=dt+(h-yy) * xx;
loss=loss+ yy*log(h)+(1-yy)*log(1-h);
end
loss=-loss/m1;
L=[L,loss];

theta2=theta1 - delta*dt/m1 - lamda*theta1/m1;
theta1=theta2;
num = num - 1;

if loss<0.01
break;
end
end
figure(1);
plot(L)
title('loss')

figure(2);
x=0:0.1:10;
y=(-theta1(1)*x-theta1(3))/theta1(2);
plot(x,y,'linewidth',2)
hold on
plot(data(label==1,1),data(label==1,2),'ro')
hold on
plot(data(label==0,1),data(label==0,2),'go')
axis([0 1 0 1])
```

```
%测试数据并计算准确率
acc=0;
for i=1:m2
xx=X2(i,1:Features)';
yy=Y2(i);
finil=1/(1+exp(-theta2 * xx));
if finil>0.5 && yy==1
acc=acc+1;
end
if finil<=0.5 && yy==0
acc=acc+1;
end
end
acc/m2
```