# 第8章 神经网络技术及其在智能传感器系统中的应用

本章内容：

* 人工神经网络基础知识
* BP神经网络
* 神经网络技术在智能传感器系统中的应用
* BP神经网络的Python实现

在上一章中，主要介绍了多元回归法在传感器温度补偿中的应用及其Python实现。本章将重点介绍人工神经网络技术及其在传感器测量数据拟合中的应用，包括人工神经网络的基础知识、BP神经网络等内容。同时，还将介绍BP神经网络的Python实现方法。通过对这些技术的学习，可以更好地理解和掌握如何利用神经网络实现智能传感器系统，并在实际应用中更好地运用这些技术。

# §8.1概述

人工神经网络（Artificial Neuron Networks, ANN）是一个大规模的非线性自适应系统，由大量的处理单元构成。这种网络基于现代神经生理科学的研究成果。其设计目标是模仿人脑处理和记忆信息的方式，从而创造出一种具有类似人脑信息处理能力的新型“机器”。Konhonen曾为神经网络提出一个定义，他将其描述为一个由简单（通常是自适应）单元和层级组织构成的大规模并行系统。神经网络的目标是模仿生物神经系统处理实际问题的方法。尽管神经网络体现了人脑的信息处理和记忆机制，但它并不是对神经系统的实际描述，而是对其的简化、抽象和模拟。

神经网络的研究可以追溯到19世纪40年代。1943年，心理学家麦克洛奇（W．McCulloch）和数理逻辑学家皮兹（W.Pizz）在《数学生物物理公报（Bulletin of Mathematical Biophysics）》上发表了关于神经网络的数学模型。该模型后来一直被称为M-P神经网络模型。他们总结了神经元的基本生理特征，提出了神经元的数学描述和网络结构方法，从此开启了神经网络的研究。到19世纪70年代，虽然由于人工智能、专家系统的发展，使得神经网络的发展一度出现低潮，但神经网络的研究并没有因此停滞不前，80年代神经网络又重新兴起。目前，神经网络在传感器信息处理、信号处理、自动控制、知识处理、运输和通信等领域得到了广泛应用。

# §8.2 神经网络基础知识

## 8.2.1 神经网络结构

如果将大量的基本神经元通过一定的拓扑结构组织起来，构成群体并行分布式处理的计算结构，那么这种结构就是人工神经网络。

根据神经元之间的不同拓扑连接结构，可将神经网络分为两大类：分层网络和相互连接型网络。

### 8.2.1.1 分层网络结构

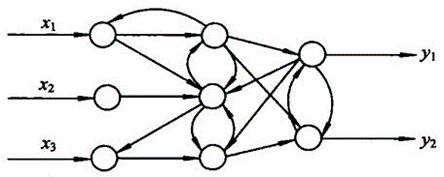
分层网络将一个神经网络模型中的所有神经元按功能分成若干层，通常有输入层、隐层（中间层）和输出层，各层按顺序连接，如图8-1所示。

  
图8-1 分层网络功能层次

输入层作为与外部激励的交互接口，接收外部输入并通过各输入神经单元将其传递到相应的隐层神经元。隐层（中间层，可能有多层）是网络内部处理单元的工作区域。不同模型在处理功能上的差异主要体现在对中间层的处理方式。输出层作为与外部显示设备或执行机构的交互接口，将网络的计算结果输出。同层之间神经元互不相连，相邻层神经元之间的连接强度由连接权值表示。为隐层第个神经元与输入层第个神经元之间的连接权值；为输出层第个神经元与隐层第个神经元之间的连接权值。

### 8.2.1.2 相互连接型网络结构

所谓相互连接型网络，指的是网络中任意两个单元之间是可达的，即任意两个单元之间存在连接路径，如图8-2所示。

  
图8-2 相互连接型网络

在该网络结构中，对于给定的某一输入模式，由某一初始网络参数出发，网络会在一段时间内动态地调整其输出模式。最终，网络可能会稳定在某一输出模式，或者陷入周期性振荡。因此，相互连接型网络可以认为是一种非线性系统。

## 8.2.2 神经元模型

在人脑中，神经细胞之间的信息传递并非是一个神经细胞把自己所接收到的信息直接传给另一个细胞，而是对所接收到的信息进行一定的处理，将处理过的信息传递给另一个神经细胞。作为对人脑神经细胞的一种模拟，人工神经元也必须具有一定的信息处理能力。

神经元是人工神经网络的基本单元，图8-3所示为一个简单的人工神经元模型。

  
图8-3 神经元模型

在图8-3中，，，⋯，，⋯，表示神经元（序号）的个输入；表示该神经元（序号为）与前层第个神经元的连接权值；为偏置值，又称阈值；为神经元的输出，神经元的输入与输出关系的一般数学表达式如下：

|  |  |  |
| --- | --- | --- |
|  |  | （8.1） |

其中：为该神经元（序号l）的总输入；为表示神经元输入输出关系的函数，称为作用函数、响应函数或传递函数。当时，称为无偏置/无阈值神经元；当时，只有当接收到的输入达到一定强度时，神经元才会被激活，称为有偏置/阈值神经元；当时，为单输入神经元；当时，称为多输入或矢量输入神经元，此时连接权，，⋯，，⋯，组成一矢量。

通过式（8.1）所表示的神经元的输入输出关系，还可以看出神经元的两个基本特性：输出是所有输入综合作用的结果；神经元具有可塑性，即其输出可以通过调整连接权值来改变。

## 8.2.3 神经元激活函数

激活函数在神经网络中起着至关重要的作用。它们的主要目的是引入非线性因素，使得神经网络能够学习并逼近复杂的非线性关系。激活函数能够实现对输入数据的非线性变换，从而提高网络的表达能力和泛化能力。激活函数的种类有很多，常用的有线型激活函数（identity）、对数型激活函数（logistic）、双正切型激活函数（tanh）和修正线型激活函数（relu）。

### 8.2.3.1 线型激活函数（identity）

线性函数是最简单的激活函数，没有非线性映射能力，只能用于建立目标变量和特征之间的线性关系。这种激活函数的数学表达式为：

|  |  |  |
| --- | --- | --- |
|  |  | （8.2） |

其函数曲线如图8-4所示。

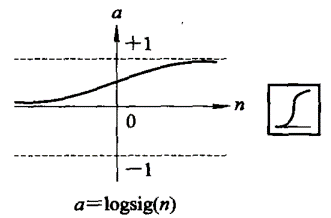
  
图8-4 线性激活函数曲线图

### 8.2.3.2对数型激活函数（logistic）

对数型激活函数，也被称为逻辑斯蒂函数，是一种常用的非线性激活函数。它可以将任何实数映射到0和1之间，因此常常被用于处理二分类问题。这种激活函数的数学表达式为：

|  |  |  |
| --- | --- | --- |
|  |  | （8.3） |

其函数曲线如图8-5所示。

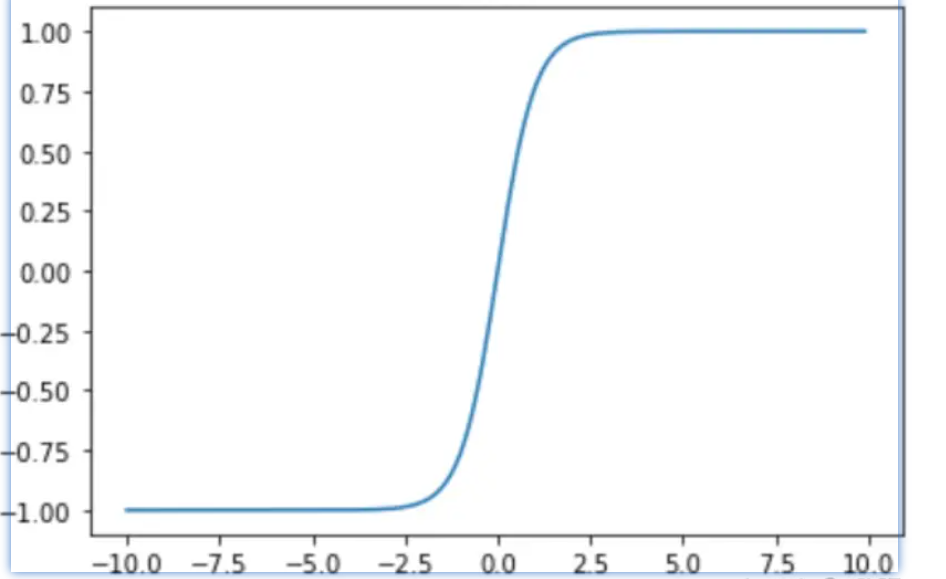
  
图8-5 对数型激活函数曲线图

### 8.2.3.3 双正切型激活函数（tanh）

双正切型激活函数（tanh）是一种常用的非线性激活函数。这种激活函数能将输入的实数值映射到一个有界的范围，即-1和1之间，从而可以缓解梯度爆炸问题。同时，其导数可以用函数自身的值来表示，这在计算反向传播的梯度时非常有用。这种激活函数的数学表达式为：

|  |  |  |
| --- | --- | --- |
|  |  | （8.4） |

其函数曲线如图8-6所示。

  
图8-6 双正切型激活函数曲线图

### 8.2.3.4修正线性激活函数（relu函数）

修正线性激活函数（ReLU）是一种常用的激活函数，具有非线性映射能力，可以用于捕捉和建立目标变量和特征之间的非线性关系。这种激活函数的数学表达式为：

|  |  |  |
| --- | --- | --- |
|  |  | （8.5） |

其函数曲线如图8-7所示。

  
图8-7 修正线性激活函数曲线图

选择激活函数并没有固定的规则，而是通常会根据具体的应用场景来决定。线性激活函数因其简单性，常被用于神经网络的输出层，用于建立目标变量和特征之间的线性关系。对数型和双正切型激活函数因其S型的非线性特性，常被用于多层神经网络的隐藏层，用于捕捉和建立目标变量和特征之间的非线性关系。修正线性激活函数因其在正区间的线性特性和负区间的零输出，在处理非线性问题时具有优势，也常被用于多层神经网络的隐藏层。总的来说，选择哪种激活函数取决于问题的性质和数据的特征。根据具体任务选择合适的网络结构、参数和优化算法，可以提高网络的性能和泛化能力。

神经元在大脑中的连接方式是多样的。根据这些连接方式的不同，可以构建各种类型的人工神经网络模型。目前，常用且成熟的网络可以从多个角度进行分类。如果从网络结构的角度进行分类，神经网络可以分为前馈神经网络和反馈神经网络。代表性的网络模型包括感知器、多层映射BP、RBF、双向联想记忆（BAM）和Hopfield模型等，它们从不同的角度描述和模拟了生物神经系统的不同层次。利用这些网络模型，我们可以实现函数逼近、数据采集聚类、模式分类、优化计算等功能。因此，神经网络在人工智能、自动控制、机器人、统计学等领域的信息处理中得到了广泛的应用。。

前馈神经网络主要包括感知器神经网络、BP神经网络和RBF神经网络等。本章将重点介绍BP神经网络。

# §8.3 BP神经网络

## 8.3.1 BP神经网络概述

BP神经网络因其权值采用反向传播（Back Propagation）的学习算法而得名。Rumelhart、McClelland和他们的同事于1982年成立了一个PDP小组，研究并行分布信息处理方法，探索人类认知的微结构。经过三年的努力，他们于1985年发展了BP神经网络学习算法。BP网络是一种多层前馈神经网络，其神经元的激活函数通常是非线性激活函数。因此，其可以实现从输入到输出的任意的非线性映射。在确定了BP网络的结构后，利用输入输出样本集对其进行训练。也就是说，对网络的权值和偏置值（bias）进行学习和调整，以使网络实现给定的输入输出映射关系。经过训练的BP网络，对于不是样本集中的输入，也能给出合适的输出，这种性质称为泛化（generalization）功能。因此，BP神经网络具有拉格朗日（Lagrange）插值法、牛顿（Newton）插值法等类似的插值功能。不过，拉格朗日插值法和牛顿插值法只能用于二维空间的曲线插值，而BP神经网络可实现多维空间的曲面插值。

## 8.3.2 BP神经网络的网络结构

BP神经网络通常由一个或多个隐藏层组成。在实际应用中，最常见的是三层BP神经网络。图8-8和图8-9展示了一个简单的三层BP神经网络的网络模型。

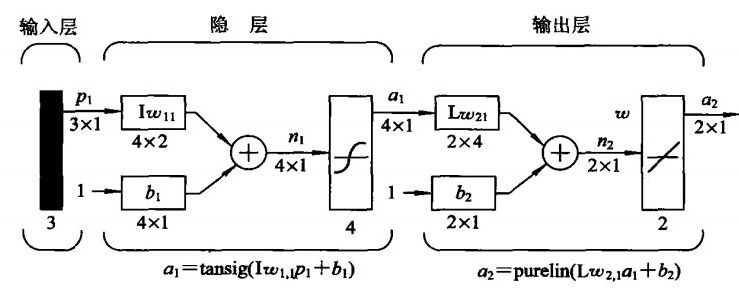
  
图8-8 BP神经网络的网络模型

  
图8-9 BP神经网络的模型示意图

在图8-8和图8-9所示的BP神经网络中，输入层包含（）个节点，隐层包含（）个节点，输出层有（）个节点。连接权值表示输入层第个节点与隐层的第个节点的连接权值，共有个权值。表示隐层第个节点与输出层第个节点的连接权值，共有个连接权值。

## 8.3.3 BP神经网络的神经元模型

BP神经网络一般由多层神经元构成，所以它的神经元激活函数可以有多种类型，其神经元激活函数的选用需要视具体情况而定。由于BP神经网络是通过误差反向传播来实现的，因此，BP神经网络中神经元的激活函数必须是连续可微的，且不能选用限幅函数。对于输出范围比较小的网络，可以将其所有的神经元激活函数全部选为对数型激活函数；如果网络的输出范围比较大，则一般把隐含层神经元激活函数选为对数型激活函数，而把输出层神经元激活函数选为纯线性函数。理论上，这样选择神经元激活函数可以以任意精度逼近任何平滑函数。对于图8-8所示BP网络模型的神经激活函数选用情况如下：

1. 输入层神经元激活函数

输入层神经元激活函数选为纯线性激活函数，故节点的输出为：

|  |  |  |
| --- | --- | --- |
|  |  | （8.6） |

式中：—第个节点的输入。

2. 隐层神经元激活函数

隐层神经元激活函数选用对数型激活函数，故节点的输出为：

|  |  |  |
| --- | --- | --- |
|  |  | （8.7） |

其中节点的总输入为：

|  |  |  |
| --- | --- | --- |
|  |  | （8.8） |

3. 输出层神经元激活函数选对数型激活函数。节点的输出：

|  |  |  |
| --- | --- | --- |
|  |  | （8.9） |

其中节点的总输入：

|  |  |  |
| --- | --- | --- |
|  |  | （8.10） |

## 8.3.4 BP神经网络的学习算法

当权值（个）、 （个）与阈值 （个）、（个）随机赋予初始值、确定分组输入，，，后，根据式（8.6）、式（8.7）、式（8.8）、式（8.9）和式（8.10）进行计算，就可得出输出层节点的输出与期望输出存在误差，输出层个节点的总误差取为

|  |  |  |
| --- | --- | --- |
|  |  | （8.11） |

网络学习也称为网络训练，就是通过反复的计算，求取，根据的大小调整网络参数，最终使得误差足够小。网络权值参数的修正数学表达式求取所遵循的规则称为学习规则，其基本思想是：使权值沿误差函数的负梯度方向改变，即：

|  |  |  |
| --- | --- | --- |
|  |  | （8.12） |
|  |  | （8.13） |

式中：—迭代次数；

—学习因子，又称步长。

按照误差反向传播算法，分别求取输出层训练误差和隐层训练误差，最后得出权值修正公式。

1. 输出层训练误差

输出层训练误差为：

|  |  |  |
| --- | --- | --- |
|  |  | （8.14） |

根据误差定义式（8.11）及式（8.9），可求得：

|  |  |  |
| --- | --- | --- |
|  |  | （8.15） |

根据式（8.9）和（8.10）可得：

|  |  |  |
| --- | --- | --- |
|  |  | （8.16） |

将式（8.15）、（8.16）代入式（8.14），得：

|  |  |  |
| --- | --- | --- |
|  |  | （8.17） |

2. 隐层训练误差

隐层训练误差为：

|  |  |  |
| --- | --- | --- |
|  |  | （8.18） |

根据式（8.7）可得：

|  |  |  |
| --- | --- | --- |
|  |  | （8.19） |

根据式（8.14）、（8.10）可得：

|  |  |  |
| --- | --- | --- |
|  |  | （8.20） |

将式（8.19）、（8.20）代入式（8.18），得：

|  |  |  |
| --- | --- | --- |
|  |  | （8.21） |

3. 权值修正公式

（1）的修正公式

将式（8.12）变换为：

|  |  |  |
| --- | --- | --- |
|  |  | （8.22） |

其中：；

则有：

|  |  |  |
| --- | --- | --- |
|  |  | （8.23） |

（2）的修正公式

将式（8.13）变换为

|  |  |  |
| --- | --- | --- |
|  |  | （8.24） |

其中：；

则有：

|  |  |  |
| --- | --- | --- |
|  |  | （8.25） |

引入势态因子，最后修正公式变为：

|  |  |  |
| --- | --- | --- |
|  |  | （8.26） |

4. 学习流程

BP网络的一个样本的学习流程如图8-10所示。

  
图8-10 BP网络训练过程及算法流程

（1）网络初始化，随机设定连接权值 ，阈值，学习因子，势态因子；

（2）向具有上述初始值的神经网络提供输入学习样本和序号；

（3）计算隐层输出值；

（4）计算输出层的输出；

（5）计算输出层和隐层训练误差；

（6）判断均方误差是否满足给定允许偏差。当满足时，结束训练；否则，进入下一步；

（7）修正权值，并转向（3）进行下一轮训练。

## 8.3.5 BP神经网络的优化器

在BP神经网络的训练过程中，优化器起着至关重要的作用。优化器的主要任务是调整网络的权值和偏置，以最小化网络输出和期望输出之间的误差。在实践中，常用的优化器有Lbfgs、Sgd和Adam。

LBFGS（Limited-memory Broyden-Fletcher-Goldfarb-Shanno）是一种迭代优化方法，特别适用于大规模参数优化问题。LBFGS采用了一种有限内存方法，能够处理大规模的优化问题，同时保持了BFGS方法的超线性收敛性能。

SGD（Stochastic Gradient Descent）是常见的优化器之一，其特点是在每次更新时只使用一个训练样本。这种方法可以加快训练速度，但可能导致更新过程变得嘈杂和不稳定。

Adam优化器融合了动量和自适应学习率的概念，通常能够提供良好的训练性能。它通过计算梯度的一阶矩估计和二阶矩估计来调整每个参数的学习率。

在实际应用中，选择优化器主要取决于具体的问题和数据。有些问题可能会发现SGD表现最好，而其他问题可能会发现Adam或LBFGS更有效。在选择优化器时，可以考虑进行多轮实验，比较不同优化器的效果，以选择最适合当前问题的优化器。同时，也需要注意，优化器的选择和其他超参数（如学习率、批大小等）的设置是相互影响的，需要综合考虑。在神经网络训练过程中，合理的优化器选择和参数设置，可以大大提高训练效率和模型性能。

# §8.4 基于BP神经网络实现涡流传感器测量数据的拟合

智能传感器系统所要研究的内容很多，除了上一章研究的温度补偿，这一节将对其中的测量数据拟合问题进行介绍。具体步骤如下。

## 8.4.1 标定实验

本节拟选用的涡流传感器的量程为0~1.2mm，将其等分为30份，每份增量0.0414mm。在测试设备和环境满足要求下，从0mm起点开始，每次增加0.0414mm的位移，直至达到1.2mm满量程。依次施加不同位移到传感器，使用网络分析仪测量其输出频率。记录每个位移值及对应的输出频率，如表8-1所示。

表8-1涡流传感器位移-频率测量数据

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 位移 d/mm | 频率 f/Hz | 位移 d/mm | 频率 f/Hz | 位移 d/mm | 频率 f/Hz |
| 0 | 382.016 | 0.44 | 389.589 | 0.88 | 393.309 |
| 0.04 | 382.908 | 0.48 | 389.988 | 0.92 | 393.413 |
| 0.08 | 383.752 | 0.52 | 390.469 | 0.96 | 393.503 |
| 0.12 | 384.595 | 0.56 | 390.857 | 1 | 393.594 |
| 0.16 | 385.388 | 0.6 | 391.308 | 1.04 | 393.696 |
| 0.2 | 386.137 | 0.64 | 391.736 | 1.08 | 393.783 |
| 0.24 | 386.814 | 0.68 | 392.099 | 1.12 | 393.864 |
| 0.28 | 387.48 | 0.72 | 392.389 | 1.16 | 393.958 |
| 0.32 | 388.059 | 0.76 | 392.658 | 1.2 | 394.051 |
| 0.36 | 388.651 | 0.8 | 392.954 |  |  |
| 0.4 | 389.095 | 0.84 | 393.153 |  |  |

## 8.4.2 样本集划分

为了提高模型的泛化能力，通常将样本数据集划分为训练集和测试集，分别用于模型的训练和测试评估。本节使用的标定数据量较少，仅有31个样本。考虑到数据量较小，采用了较为保守的划分比例，即从量程范围内随机选取80%的数据作为训练集，另外20%作为测试集。这样可以确保测试集中有足够的样本数量来评估模型的泛化性能。具体来说，从标定数据的量程范围内随机选取24个样本作为训练集，剩余的7个样本作为测试集。在样本量较少的情况下，这种划分方式是较为标准和合理的。

## 8.4.3 样本数据归一化

归一化处理后的数据位于-1～＋1或者0～1之间，这有助于实现更好的数据融合。。数据归一化的公式有多种，如：

|  |  |  |
| --- | --- | --- |
|  |  | （8.27） |

或

|  |  |  |
| --- | --- | --- |
|  |  | （8.28） |

式中：—归一化后的样本数据；

—归一化前的样本数据；

—X所在行的最小值；

—X所在行的最大值。

本例采用的是第一种归一化公式。

## 8.4.4 BP神经网络的设计与实现

设计实现BP神经网络的流程图如图8-11所示。

  
图8-11 设计实现BP神经网络的流程图

根据上述流程图，在Python环境下，可以利用Scikit-learn库建立BP神经网络模型，并进行训练与预测。详细的内容请参见8.5节。

## 8.4.5 训练校验

### 8.4.5.1 条件

输入层节点数为1，隐层节点数为50，输出层节点数为1，最大迭代次数为10000，激活函数为tanh，优化器为lbfgs，初始学习率为0.01。这些参数的选择都是为了使模型能够更好地学习和预测。

### 8.4.5.2 训练与测试

首先，运行编写的程序。程序会自动加载样本数据，并将其划分为样本集和测试集。接着，对数据进行归一化处理，然后持续进行10000次训练与校验。训练结束后，测试样本的网络输出如表8-2所示。通过与期望输出的对比，可以看到实际输出结果与期望输出结果的误差很小。其中，训练样本的均方差为，测试样本的均方差为。

表8-2 测试样本期望输出与实际输出对照表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 期望输出f/Hz | 383.752 | 393.958 | 390.469 | 389.095 | 393.783 | 393.594 | 393.309 |
| 实际输出f/Hz | 383.771 | 393.954 | 390.447 | 389.118 | 393.805 | 393.625 | 393.267 |
| 绝对偏差f/Hz | -0.019 | 0.004 | 0.022 | -0.023 | -0.022 | -0.031 | 0.042 |

### 8.4.5.3 提取模型参数

首先，在安装有相关环境的电脑上，利用scikit-learn库中的MLPRegressor类训练BP神经网络。训练完成后，通过MLPRegressor类的coefs\_属性和intercepts\_属性提取模型参数。然后，利用这些提取的模型参数，设计并开发一个可以在微控制器中运行的预测函数。最后，将这个预测函数部署到微控制器中。这样，就可以在微控制器平台上实现涡流传感器测量数据的拟合。

## 8.4.6 拟合效果评价

### 8.4.6.1 拟合后涡流传感器的静态特性

将8-1中的所有样本进行拟合，拟合后的实际输出频率如表8-3所示

表8-3 拟合后涡流传感器的输出频率

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 位移 d/mm | 频率 f/Hz | 位移 d/mm | 频率 f/Hz | 位移 d/mm | 频率 f/Hz |
| 0 | 382.015 | 0.44 | 389.585 | 0.88 | 393.267 |
| 0.04 | 382.907 | 0.48 | 390.024 | 0.92 | 393.401 |
| 0.08 | 383.771 | 0.52 | 390.447 | 0.96 | 393.519 |
| 0.12 | 384.601 | 0.56 | 390.868 | 1 | 393.625 |
| 0.16 | 385.390 | 0.6 | 391.291 | 1.04 | 393.719 |
| 0.2 | 386.133 | 0.64 | 391.708 | 1.08 | 393.805 |
| 0.24 | 386.827 | 0.68 | 392.095 | 1.12 | 393.883 |
| 0.28 | 387.472 | 0.72 | 392.428 | 1.16 | 393.954 |
| 0.32 | 388.067 | 0.76 | 392.703 | 1.2 | 394.019 |
| 0.36 | 388.614 | 0.8 | 392.926 |  |  |
| 0.4 | 389.118 | 0.84 | 393.153 |  |  |

### 8.4.6.2 拟合误差计算

在众多衡量拟合精度的指标中，最常用的是均方误差。将表8-3中的数据代入式（7.18）所示的均方误差公式可得，均方误差仅有，能过达到精度要求。

# §8.5 BP神经网络的Python实现

根据前面的介绍，利用Python实现BP神经网络的流程如下。

## 8.5.1 安装需要的第三方库

Python之所以功能强大，正是因为它拥有众多的第三方库。在本章中，将使用numpy库和scikit-learn库来实现BP神经网络。由于这两个库的安装方法已经在前面的章节中介绍过，这里就不再赘述。但是，请确保您已经成功安装了它们。

## 8.5.2 引用第三方库

在使用第三方库之前，我们需要在程序中导入它们。为了实现BP神经网络，我们需要添加以下的导入语句：

import numpy as np

from sklearn.neural\_network import MLPRegressor

from sklearn.metrics import mean\_squared\_error

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import MinMaxScaler

## 8.5.3 输入样本数据

样本数据的输入，可以使用numpy库创建数据数组，示例代码如下：

X = np.array([]) #在[]内填入样本特征值，元素间用逗号分隔

y = np.array([]) #在[]内填入样本标签值，元素间用逗号分隔

## 8.5.4 划分训练集和测试集

为了将样本数据集分为训练集和测试集，可以使用scikit-learn库中model\_selection子模块提供的train\_test\_split()函数，该函数的定义如下：

X\_train, X\_test, y\_train, y\_test = train\_test\_split(train\_data, train\_target, test\_size, random\_state, shuffle)

输入参数：

train\_data：待划分的样本特征数据，可以是列表、数组、稀疏矩阵或数据框；

train\_target：待划分的样本标签数据，可以是列表、数组或数据框；

test\_size：测试集占总样本的比例或数量，可以是浮点数、整数或None，默认为None，表示自动设置为0.25；

random\_state：随机状态，可以是整数或随机数生成器实例，默认为None，表示每次分割都是随机的；

shuffle：是否打乱样本顺序，可以是布尔值，默认为True，表示打乱。

输出参数：

X\_train：训练集的特征数据；

X\_test：测试集的特征数据；

y\_train：训练集的标签数据；

y\_test：测试集的标签数据。

利用该函数，可以使用下面的语句实现训练集和测试集的划分：

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0)

## 8.5.5 归一化数据

归一化数据是一种数据预处理方法。它的目的是将数据的数值范围统一到一个较小的区间，通常是[0,1]或[-1,1]。这样可以减少数据的差异性和异常值的影响，提高模型的稳定性和收敛速度。

scikit-learn库中preprocessing子模块提供了MinMaxScaler类，用于实现数据的归一化处理。该类的语法如下：

scaler = MinMaxScaler(feature\_range=(0, 1), copy=True)

该类的参数说明如下：

feature\_range：指定归一化后的数值范围，可以是元组类型，默认为(0, 1)，表示收敛到[0,1]区间，也可以取其他范围值；

copy：指定是否对原数据进行拷贝操作，可以是布尔值类型，默认为True，表示对原数据组拷贝操作，这样变换后不会影响原数据；

该类的主要方法说明如下：

fit(X)：根据输入数据X计算归一化所需的最小值和最大值；

transform(X)：根据已经计算出的最小值和最大值对输入数据X进行归一化变换；

fit\_transform(X)：结合fit和transform两个方法，先计算最小值和最大值，再进行归一化变换

inverse\_transform(X)：根据已经计算出的最小值和最大值对输入数据X进行反向变换，还原为原始数据。

使用该类，可以利用如下的示例代码实现样本数据的归一化：

scaler = MinMaxScaler()

X\_train = scaler.fit\_transform(X\_train.reshape(-1,1))

X\_test = scaler.transform(X\_test.reshape(-1,1))

## 8.5.6 创建并训练BP神经网络模型

scikit-learn库中neural\_network子模块提供了MLPRegressor类，用于实现BP神经网络的回归预测。该类的语法如下：

model=MLPRegressor(hidden\_layer\_sizes=(50,), activation='tanh', solver='lbfgs', max\_iter=10000, learning\_rate\_init=0.01)

该类的参数说明如下：

hidden\_layer\_sizes：指定隐藏层的大小，可以是元组类型，默认为(100,)，表示只有一个隐藏层，有100个神经元；

activation：指定激活函数的类型，可以是字符串类型，默认为'relu'，表示使用修正线性单元函数，也可以取其他值，如'tanh'表示使用双曲正切函数；

solver：指定优化器的类型，可以是字符串类型，默认为'adam'，表示使用一种基于随机梯度的优化器，也可以取其他值，如'lbfgs'表示使用一种拟牛顿方法；

max\_iter：指定最大迭代次数，可以是整数类型，默认为200，表示最多进行200次迭代；

learning\_rate\_init：指定初始学习率，可以是浮点数类型，默认为0.001，表示初始时每次更新权重和偏置的步长为0.001；

该类的主要方法说明如下：

fit(X, y)：根据输入数据X和标签数据y训练BP神经网络模型；

predict(X)：根据输入数据X预测输出数据y；

score(X, y)：根据输入数据X和标签数据y评估BP神经网络模型的性能；

使用该类的一个示例代码如下：

model=MLPRegressor(hidden\_layer\_sizes=(50,), activation='tanh', solver='lbfgs', max\_iter=10000, learning\_rate\_init=0.01)

model.fit(X\_train,y\_train)

## 8.5.7 预测并评估模型性能

预测并评估模型性能是机器学习中的一个重要步骤，它可以帮助我们了解模型的泛化能力和优化方向。预测和评估模型性能的一个示例代码如下：

#使用训练好的模型model，根据测试数据X\_test预测输出数据y\_pred

y\_pred = model.predict(X\_test)

#使用mean\_squared\_error函数，计算均方误差，并赋值给变量mse

mse = mean\_squared\_error(y\_test,y\_pred)

**例8-2** 利用8.4节的样本数据，编写Python程序实现涡流传感器测量数据的拟合。

代码如下：

# 导入必要的库

import numpy as np

from sklearn.neural\_network import MLPRegressor

from sklearn.metrics import mean\_squared\_error

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import MinMaxScaler

# 输入样本数据

X = np.array([0, 0.04, 0.08, 0.12, 0.16, 0.2, 0.24, 0.28, 0.32, 0.36, 0.4, 0.44, 0.48, 0.52, 0.56, 0.6, 0.64, 0.68, 0.72, 0.76, 0.8, 0.84, 0.88, 0.92, 0.96, 1, 1.04, 1.08, 1.12, 1.16, 1.2])

# 输出样本数据

y=np.array([382.016, 382.908, 383.752, 384.595, 385.388, 386.137, 386.814, 387.48, 388.059, 388.651, 389.095, 389.589, 389.988, 390.469, 390.857, 391.308, 391.736, 392.099, 392.389, 392.658, 392.954, 393.153, 393.309, 393.413, 393.503, 393.594, 393.696, 393.783, 393.864, 393.958 , 394.051])

# 划分训练集和测试集

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0)

# 归一化数据

scaler = MinMaxScaler()

X\_train = scaler.fit\_transform(X\_train.reshape(-1,1))

X\_test = scaler.transform(X\_test.reshape(-1,1))

# 创建并训练BP神经网络模型

model = MLPRegressor(hidden\_layer\_sizes=(50,), activation='tanh', solver='lbfgs', max\_iter=10000, learning\_rate\_init=0.01)

model.fit(X\_train,y\_train)

# 预测并评估模型性能

y\_pred = model.predict(X\_test)

mse = mean\_squared\_error(y\_test,y\_pred)

print('均方误差:', mse)

执行上面的代码，可以得到以下结果：

|  |
| --- |
| 均方误差: 0.0006585669133012904 |

这结果与上一节手工计算的结果完全一致。这证实了我们的Python程序是正确的。

# 习题8

1. 神经网络按照神经元之间的连接方式可以分为哪两类？各自的特点是什么？

2. 神经元模型的基本组成部分有哪些？它们之间的关系是什么？

3. 神经元的两个基本特性是什么？它们的意义是什么？

4. 激活函数的主要作用是什么？常用的激活函数有哪些？

5. BP神经网络的网络结构通常由哪些层组成？每层的节点数分别是多少？

6. 请简述BP神经网络的学习流程。

7. BP神经网络的优化器有哪些？它们的主要特点和优缺点分别是什么？

8. 什么是样本集划分？它的目的是什么？

9. 请简述利用Python实现BP神经网络的流程。