


中国研究生创新实践系列大赛

“华为杯”第十八届中国研究生

数学建模竞赛

学　　校	中南大学
参赛队号	21105330243
队员姓名	1.邹江枫
	2.杨金娜
	3.王辉

中国研究生创新实践系列大赛

“华为杯”第十八届中国研究生

数学建模竞赛

题　目　信号干扰下的超宽带（UWB）精确定位问题

摘　　要

超宽带 UWB 技术因厘米级的定位精度、信号穿透能力强、设备成本功耗低等优点，被广泛应用于军事及民用等领域，但如何有效抑制信号干扰和场景切换的影响，仍是实现 UWB 精准定位的关键目标。本文通过对 UWB 实测数据进行深入的挖掘，建立有无干扰样本的分类模型和拟合模型，对定位误差具有一定的校正效果。

针对问题一，本文首先采用**正则表达式**遍历匹配出关键数据，其次判断同一时间戳下测得的 4 行数据是否完整，若缺失则将此时间戳的所有锚点数据剔除，然后根据同一时间戳下的测距值与校验值删除完全重复的样本，再采用**拉依达准则**（3σ 准则）去除误差较大的异常值，最后得到的处理数据用于之后的建模研究。

针对问题二，由于实测距离与靶点坐标之间具有较强的非线性关系和噪声干扰，采用一般的线性模型不能完全准确的预测靶点的坐标。为此我们基于**卡尔曼滤波**和**BP 神经网络**建立 8 输入 4 输出的精准定位模型。首先，利用实际场景 1 的坐标信息，建立锚点距离与靶点坐标之间的转换公式，并对 4 个锚点实测距离序列进行卡尔曼平滑，以滤除样本数据中的噪声影响；接着，基于真实靶点坐标反推出 4 个推演锚点距离，得到锚点实测与推演间的距离差值；然后，为了模型能够具有较好的去噪和拟合能力，将原有的实测数据与新计算出的距离差值进行特征组合，输出为四个锚点到靶点的预测距离，整个过程相当于**实测数据的校正**，再基于预测距离得到靶点的坐标。

针对问题三，定位模型须满足不同的应用场景，因此为验证任务 2 在实验场景 1 下构建的模型效果，我们将任务 2 建立的正常模型与异常模型分别用于实验场景 2 下 10 组数据的预测，最终得到靶点的精确 3 维坐标（见表 4-3 和 4-4）。

针对问题四，在应用任务 2 所建立的正常模型和异常模型之前，须对实测数据是否受到信号干扰进行判断，才能选择正确的模型进行精准定位。为此我们采用了基于 BP 神经网络的分类模型，模型输入仍采用**实测距离与推演距离的差值**，用来反映正常数据与异常数据的差别，以更好的区分正负样本。模型的输出为预测概率，当大于

阈值 0.5 时判断为正常数据，反之则为异常数据。最终分类模型的测试准确率达到了 94.43%，验证了模型在区分样本有无受干扰上的有效性。

针对问题五，我们首先对数据进行预处理操作，去除掉其中重复、相似、异常的样本数据，再基于清洗后的数据采用任务 4 的分类模型，对数据进行正确分类，判断测得的数据是否收到信号干扰。基于分类结果，再采用任务 2 训练好的相对应的定位模型，预测动态靶点的坐标并绘制轨迹曲线。同时预处理中的卡尔曼平滑操作通过滤除靶点运动过程中的噪声，可以更好地实现正常/异常样本切换时的精准定位。

关键词：拉依达准则；BP 神经网络；回归；分类预测；动态轨迹

目录

1 问题重述

1.1 问题背景

相较于 WiFi、ZigBee 等传统无线通信技术，超宽带(Ultra Wideband, UWB)技术因实时测距精度高、抗多径干扰能力强、设备成本功耗低等优点，逐渐成为室内精确定位系统的研究主流[1,2]。其中，基于飞行时间（Time of Flight, TOF）的双向测距算法是最常用的 UWB 定位方法之一，基本原理是通过测量收发器之间的信号传播时间，再乘以光速来实现厘米级的定位精度。但由于实际室内环境的复杂多变，UWB 通信信号极易受到干扰而导致测距误差，严重时甚至无法实现室内定位，因此如何有效抑制信号干扰的影响，并实现超宽带（UWB）精确定位是目前的主要研究热点[3]。

UWB 锚点与 Tag 靶点间的测距值是 TOF 定位算法的重要依据，但由于环境噪声、信号干扰等因素的影响，4 个测距值往往存在着不同程度的误差，相应计算出的 Tag 靶点范围也就会不准，故必须对其进行校正处理，以满足精确室内定位的要求。因此基于实际场景的实测数据，若能准确识别信号有无受干扰，并建立相应的精确定位模型，校正实际测距误差，将是室内精确定位的又一次革命性进展。

1.2 问题重述

针对上述研究背景与现状，题目提供了不同实验场景下，在有信号干扰和无信号干扰 2 种情况下不同锚点到不同靶点的数据文件，以及靶点的动态轨迹等相关数据，主要需利用数据挖掘技术研究和完成以下几个任务：

任务 1 数据预处理

实测场景 1 提供了 648 个数据文件，可根据正常、异常数据分成两类，每类数据包括 324 个文件，分别代表 4 个锚点到 324 个不同靶点的数据。每个文件又包括多行数据，分别代表四个锚点在不同时刻到此靶点（Tag）的距离信息。需对这些数据文件中的异常、缺失、相同或相似数据进行预处理，并重点列出以下 4 个数据文件：

“正常数据”文件夹中：　　24.正常.txt、　　109.正常.txt

“异常数据”文件夹中：　　1.异常.txt、　　100.异常.txt

任务 2 定位模型

根据任务 1 处理后的正常样本和异常样本，分别建立能充分反映实验场景信息的 2 个定位模型，使模型在有无干扰的情况下，均能准确拟合 Tag（3 维坐标）的精确位置。模型效果在附件 2 中提供的前 5 组（信号无干扰）数据和后 5 组（信号有干扰）数据上进行测试，并选择合适的评价指标验证模型的精度。

任务 3 不同场景应用

任务 2 的训练数据仅采集于实验场景 1，但其建立的定位模型要求能够在不同的

实际场景中使用，因此本题需将在实验场景 1 下建立的 2 个定位模型迁移到实验场景 2，实现附件 2 中 10 组数据（前 5 组无干扰、后 5 组有干扰）的三维坐标精准定位。

任务 4 分类模型

任务 2 建立的定位模型是在已知信号有无干扰的情况下建立的，但在 UWB 采集数据时不知道是否存在信号干扰，因此需先利用任务 1 获得的数据，建立分类模型，使模型能够正确区分采集到的数据是否存在干扰，再选择合适的评价指标评估分类模型的精度，并用模型对附件 4 的信号进行正确分类。

任务 5 运动轨迹定位

运动轨迹定位是 UWB 重要应用之一，任务 2 得到的定位模型是单个静态点的定位模型，没有考虑靶点的自身运动规律。本题要求同时考虑信号是否受干扰和动态靶点这 2 个因素建立模型，来对靶点的运动轨迹进行精准定位和图形绘制，测试数据采用附件 5 中采集到的动态靶点数据。

2 基本假设与符号说明

2.1 基本假设

（1）假设题目所采集正常样本可以正确反映相应靶点的定位信息；

（2）假设同一 Tag 坐标信息对应的多个实际测距数据中的噪声服从高斯分布和马尔科夫假设；

（3）假设预处理后的结果是符合后续建模要求的，且训练好的定位模型能够同时很好的应对有无干扰和不同实验场景的情形。

2.2 符号说明

表 2-1 符号说明

符号	说明
T_i	第 i 个时间戳（附件 1）
m_i	锚点 i 的测距值变量
m	x_i 的算术平均值
n_i	第 i 个测量值的剩余误差
σ	标准误差
$\mathbf{x}$	系统状态
$\mathbf{u}$	系统输入
$\mathbf{y}$	观测输出
$\mathbf{w}$	过程噪声
$\mathbf{v}$	观测噪声
$\mathbf{\Phi}$	状态转移矩阵
$\mathbf{H}$	观测矩阵
x_i	神经网络的 n 维输入向量
y_i	神经网络的拟合 Tag 标签
W	权重矩阵 1
V	权重矩阵 2
b^1	偏置矢量 1
b^2	偏置矢量 2

3 数据预处理

3.1 问题分析

对于问题一，附件 1 给出了在多个 Tag 坐标位置处采集得到的 648 组记录文件，每组文件包含了在同一 Tag 坐标点采集到的多个数据，其中有干扰和无干扰情况下各 324 组。由于实际测量过程中的误差，本题须对原始记录进行数据提取及预处理操作，并删除掉其中的一些“无用”样本，主要考虑异常、缺失、相同或相似 4 种情况，再经过卡尔曼滤波用于之后的建模研究。总体的数据处理流程如下图 3-1 所示。

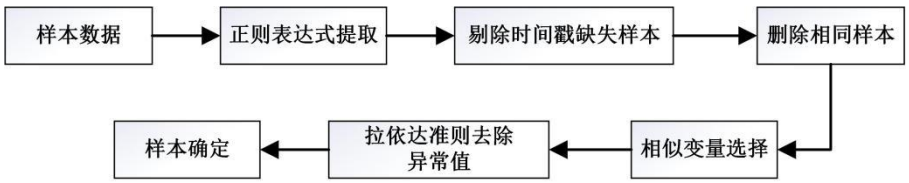


图 3-1 问题一思路流程图

总体处理步骤分为 5 步，首先基于原始数据格式，采用正则表达式抓取其中的可用数据，按照每 4 行保存成一个样本的保存要求得到原始数据集 1；再判断同一时间戳是否采集到了完整的 4 个锚点测距值，剔除掉其中有缺失和完全重复的样本，得到数据集 2；之后分析测量值与校验值等变量的相关性，实现关键特征变量的选取，删除不必要的变量列；然后采用拉依达准则（3 σ 准则）去除误差较大的异常值，得到数据集 3；最后以二维表形式展示最终的数据预处理结果。

3.2 数据处理

附件 1 为实际场景下测得的数据，主要包括无干扰的正常数据、有干扰的异常数据以及对应的 Tag 坐标信息。其中 Tag 信息为标准的结构化数据格式，不需要再做额外的处理，具体格式如表 3-1 所示；

表 3-1　三维 Tag 坐标数据信息格式

Tag 编号	坐标点 x	坐标点 y	坐标点 z
1	50	50	88
2	50	100	88
3	50	150	88
...	...	...	...
324	450	450	200

有无干扰下的正常/异常样本数据则由结构化数字和非结构化文本共同构成，因此需要进一步的数据提取和处理。其中须重点列出的第 24 个 Tag 的正常样本数据和第 1 个 Tag 的异常样本数据格式具体如表 3-2 所示。T 表示时间戳，即数据采集的时刻；PR 表示 4 个锚点的 ID 标号；4 列数据分别对应该锚点的测距值(mm)：测距值的校验值：数据序列号：数据编号；每 4 行为 1 组，代表了 UWB 采集到的一组完整数据。

表 3-2　24.正常.txt 和 1.异常.txt 原始文件数据

24.正常 txt	1.异常 txt
T:094843811:RR:0:0:3280:3280:127:15743	T:090531088:RR:0:0:1280:1280:229:3301
T:094843811:RR:0:1:4660:4660:127:15743	T:090531088:RR:0:1:4550:4550:229:3301
T:094843811:RR:0:2:2600:2600:127:15743	T:090531088:RR:0:2:4550:4550:229:3301
T:094843811:RR:0:3:3910:3910:127:15743	T:090531088:RR:0:3:6300:6300:229:3301
T:094844019:RR:0:0:3280:3280:128:15744	T:094844019:RR:0:0:3280:3280:128:15744
T:094844019:RR:0:1:4670:4670:128:15744	T:094844019:RR:0:1:4670:4670:128:15744
T:094844019:RR:0:2:2600:2600:128:15744	T:094844019:RR:0:2:2600:2600:128:15744
T:094844019:RR:0:3:3920:3920:128:15744	T:094844019:RR:0:3:3920:3920:128:15744
...	...

3.3.1　正则表达式提取数据

附件 1 中 648 组样本数据均为如表 3-2 所示的非结构化数据格式，故需设计一种文本匹配模式以抓取出其中关键的结构化数据，再用于之后数据的处理和建模。本题采用正则表达式来得到时间戳、锚点标号、对应测距值、对应校验值这 4 列关键数据，针对每个 txt 文本进行遍历（正常/异常样本分开处理），若存在以下格式：

$$\text{Compile} = \text{r'T:(\w+):RR:0:(\w+):(\w+):(\w+)'} \tag{3-1}$$

则抽取出相应位置的数值形成一条结构化数据，之后将相同时间戳的 4 个锚点测距值和校验值合并为一个样本数据，再与 Tag 标签信息进行整合，形成初始的正常样本和异常样本。其中 Compile 表示数据匹配模板，\w+表示匹配单词字符及数字。

3.3.2　缺失数据的处理

由于同一时间戳的 4 个锚点数据才为一组完整的数据，此部分数据处理工作需结合正则表达式提取出来的时间戳标识进行。首先分别遍历正常样本和异常样本 txt 文件，利用正则表达式匹配出关键数据，再统计样本中每个时间戳的频次，若存在：

$$Counter(T_i) < 4 \tag{3-2}$$

则认为该时间戳数据缺失，但由于实际测量得到的数值本身存在着不同程度的误差，再进行补全将可能引入更加复杂的误差，且数据样本足够用来之后的建模工作，故我们不考虑这类缺失数据的补全，直接将该时间戳的数据剔除。

3.3.3　相同数据的剔除

缺失数据处理完后，将同一时间戳的四个锚点数据整合成一个样本，即每个样本对应 1 个时间戳和 8 个距离数据，再进行相同数据的剔除。首先因测距值和校验值完全相同，我们将 4 列校验值全部删除，只将 4 列测距值作为模型输入；再进行每行样本的遍历比较，剔除掉相同的样本数据；最后处理得到的测距数据再与对应的 Tag 坐标信息进行整合形成数据集 2。

3.3.4　异常数据的剔除

针对正常样本和异常样本中可能存在的异常值，本文选用拉依达准则（又称 3σ 准则）对数据集进行检验。假设测量得到 4 个测距值 m_1, m_2, m_3, m_4 为等精度测量，可计算出其算术平均值 m 以及剩余误差 $n_b = m_i - m(i = 1,2,\cdots,4)$，进而求出标准误差：

$$\sigma = \left[\frac{1}{n-1}\sum_{i=1}^{n} n_i^2\right]^{1/2} = \left\{\left[\sum_{i=1}^{n} m_i^2 - \left(\sum_{i=1}^{n} m_i\right)^2 / n\right] / (n-1)\right\}^{1/2} \tag{3-3}$$

若某个测量值 m_b 的剩余误差 $n_b\,(1 <= b <= 4)$，满足 $|n_b| = |m_b - m| > 3\sigma$ 那么就可认为该行样本是含有粗大误差的异常值，应予剔除。例如正常样本中第 24 个 Tag 位置以及第 109 个 Tag 位置数据的剔除情况如下图 3-2、3-3 所示。

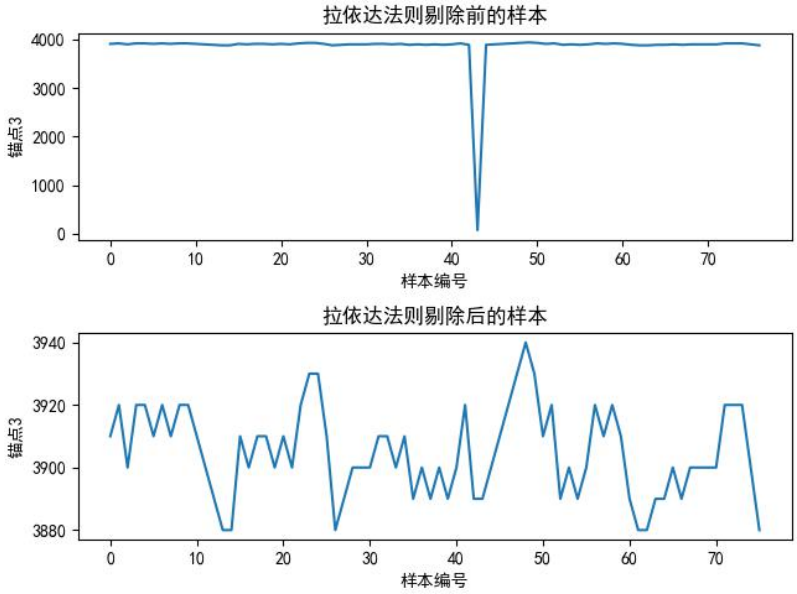


图 3-2　24.正常.txt 文件锚点 3 剔除前后的数据对比

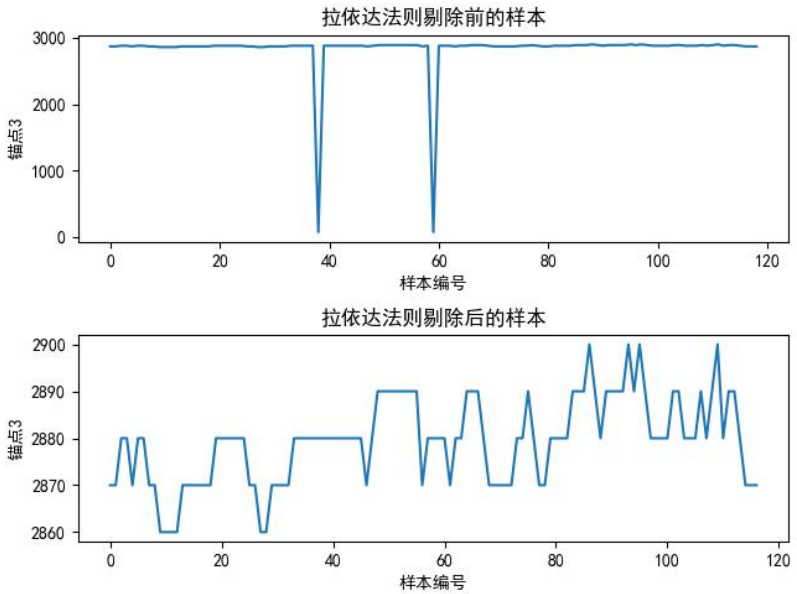


图 3-3　109.正常.txt 文件锚点 3 剔除前后的数据对比

3.4 样本确定

至此，“无用”数据（缺失、相同、相似、异常）的预处理过程完毕。按照上述过程的处理结果如表 3-3 所示，具体数据结果见上传的附件 1-“第一问数据处理”。

表 3-3　无用数据的处理结果

操作类别	24.正常.txt	109.正常.txt	1.异常.txt	100.异常.txt
缺失数据	38	87	78	83
相同数据	52	113	159	66
异常数据	1	2	1	3
总计剔除	91	202	238	152
剩余个数	77	119	177	196

其中 24.正常.txt 文件共检测出缺失数据 38 个、相同数据 52 个、异常数据 1 个。109.正常.txt 文件中共检测出缺失数据 87 个、相同数据 113 个、异常数据 2 个。1.异常.txt 文件中共检测出缺失数据 78 个、相同数据 159 个、异常数据 1 个。100.异常.txt 文件中共检测出缺失数据 83 个、相同数据 66 个、异常数据 3 个。剔除以后的样本数据再用于之后的模型建立。

4 基于卡尔曼滤波和 BP 神经网络的精准定位模型

4.1 问题分析

问题二要求基于上述预处理后的数据和 4 个实测距离，分别针对正常和异常样本建立 2 个定位模型，再采用附件 2 中的 10 组实测数据进行模型验证。从数据构成考虑，无论是正常样本还是异常样本，Tag 在同一坐标点上都保留了采集到的多组数据，并存在着不同程度的误差，若直接进行数据拟合将会引入噪声，而卡尔曼滤波技术能基于历史数据有效实现数据去噪与平滑，动态修正 UWB 测距误差，进而获得较高的预测效果。因此针对问题二，我们首先采用卡尔曼滤波技术对 4 个测距值进行数据平滑和去噪，再进行数据归一化处理，进而建立 BP 神经模型用于定位数据的拟合预测。

整体解决思路如图 4-1 所示，样本数据包括测量数据和真实坐标，根据真实坐标可以得到真实距离，我们将真实距离当作 BP 网络模型的标签。理论上一组距离数据对应一个三维坐标，根据测量距离我们能够得到靶点理论坐标，再根据距离推导公式得到锚点到靶点的理论距离。理论距离与测量距离之间是存在误差的，我们可以得到理论距离与测量距离的差值，再结合实际的测量距离，可以得到一组输入向量，输入维度为 8。我们随机的将样本数据根据 9：1 的比例分为训练集和测试集，为了在模型训练过程中使模型快速收敛，我们选择了均方差作为损失函数，在模型训练过程中，通过反向传播逐渐降低模型的损失，使得预测输出不断接近期望输出。在模型训练完成后，我们通过对测试集进行预测，并采用一定的评价指标对模型进行评价。

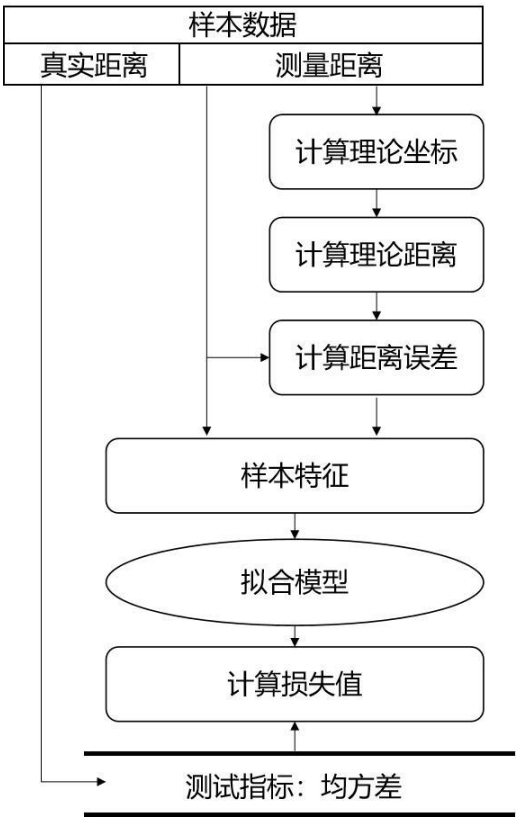


图 4-1　问题 2 思路流程图

4.2 基于卡尔曼滤波的时间序列平滑

4.2.1　卡尔曼滤波

卡尔曼理论[4]假设处理信号满足下述状态转换方程和观测方程，其中 $\mathbf{x}$ 表示系统状态（隐变量），$\mathbf{u}$ 为系统输入，$\mathbf{y}$ 表示观测输出，$\mathbf{w}$ 表征过程噪声，$\mathbf{v}$ 为观测噪声。

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}) + \mathbf{w}_k \tag{4-1}$$

$$\mathbf{y}_k = h(\mathbf{x}_k) + \mathbf{v}_k \tag{4-2}$$

噪声数据假定服从均值为 0、方差固定的白噪声，概率分布如下式所示。

$$\mathbf{w}_k \sim \mathrm{N}(0, R) \tag{4-3}$$

$$\mathbf{v}_k \sim \mathrm{N}(0, Q) \tag{4-4}$$

基于上述模型，卡尔曼滤波算法流程主要分为预测和更新两步。首先对于每个时刻计算状态估计和协方差矩阵，进而得到卡尔曼增益对模型估计和观测两部分进行加权更新，具体计算过程如下式所示：

$$\hat{\mathbf{x}}_k = \mathbf{\Phi}_{k-1}\hat{\mathbf{x}}_{k-1} + \mathbf{B}_{k-1}\mathbf{u}_{k-1} \tag{4-5}$$

$$\hat{\mathbf{P}}_k = \mathbf{\Phi}_{k-1}\hat{\mathbf{P}}_{k-1}\mathbf{\Phi}_{k-1}^T + \mathbf{Q}_{k-1} \tag{4-6}$$

$$K_k = \hat{\mathbf{P}}_k \mathbf{H}_k{}^T (\mathbf{H}_k \hat{\mathbf{P}}_k \mathbf{H}_k{}^T + \mathbf{R}_k)^{-1} \tag{4-7}$$

$$\hat{\mathbf{x}}_k^* = \hat{\mathbf{x}}_k + K_k(\mathbf{y}_k - \mathbf{H}_k\hat{\mathbf{x}}_k) \tag{4-8}$$

$$\mathbf{P}_k^* = \mathbf{P}_k - K_k \mathbf{H}_k \mathbf{P}_k \tag{4-9}$$

其中 $\mathbf{\Phi}$ 是状态转移矩阵，$\mathbf{H}$ 是观测矩阵，当系统模型非线性时，矩阵具有时变特性，可通过扩展卡尔曼滤波（EKF）等模型线性近似非线性，实现状态估计与观测。

4.2.2　卡尔曼平滑

卡尔曼滤波算法通过历史数据来预测未来的数据，适用于在线数据的实时处理。而在离线数据处理中，为充分运用所有的观测数据，可采用卡尔曼平滑技术后向传递更新，以进一步提升预测精度[5]。

$$\mathbf{C}_k = \hat{\mathbf{P}}_k \mathbf{\Phi}_k \hat{\mathbf{P}}_{k+1}^{-1} \tag{4-10}$$

$$\hat{\mathbf{x}}_k^s = \hat{\mathbf{x}}_k^* + \mathbf{C}_k (\hat{\mathbf{x}}_{k+1}^s - \hat{\mathbf{x}}_{k+1}) \tag{4-11}$$

$$\mathbf{P}_k^s = \mathbf{P}_k^* + \mathbf{C}_k (\hat{\mathbf{P}}_{k+1}^s - \hat{\mathbf{P}}_{k+1}) \mathbf{C}_k^T \tag{4-12}$$

算法首先利用卡尔曼滤波方程前向传递后的状态估计和协方差矩阵，作为后向传递的输入，进而得到整个时间序列的平滑估计值。卡尔曼平滑效果如图 4-2 所示，误差带为高斯假设下 90%的置信区间。

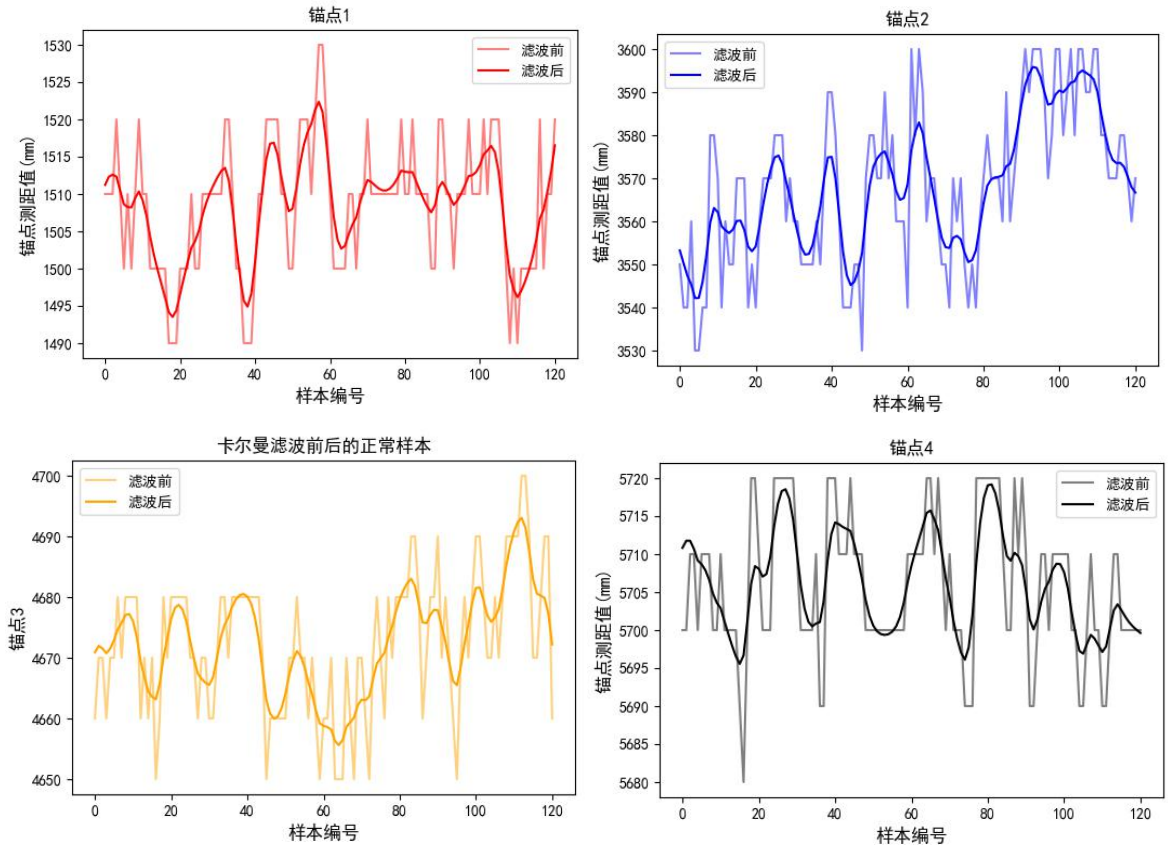


图 4-2　卡尔曼滤波平滑前后的正常样本（4 个锚点数据）

可以看出，滤波前的数据随测量时间的变化而波动剧烈，这是由于环境噪声等干扰所导致的，直接用来拟合模型有所欠妥。因此本文采用卡尔曼平滑来对同一时间戳下的实测距离进行处理，既保留原有序列时间模式，也能提升模型拟合的预测效果。

4.3　BP 神经网络基本原理

BP 神经网络是一种多层前馈神经网络，通过前向传播来计算网络误差，然后通过误差反向传播来更新网络参数。网络训练过程中采用了监督学习的方法。下面以三层（输出层，隐含层，输出层）神经网络介绍 BP 网络的基本原理。

如图所示为三层 BP 神经网络的拓扑结构：

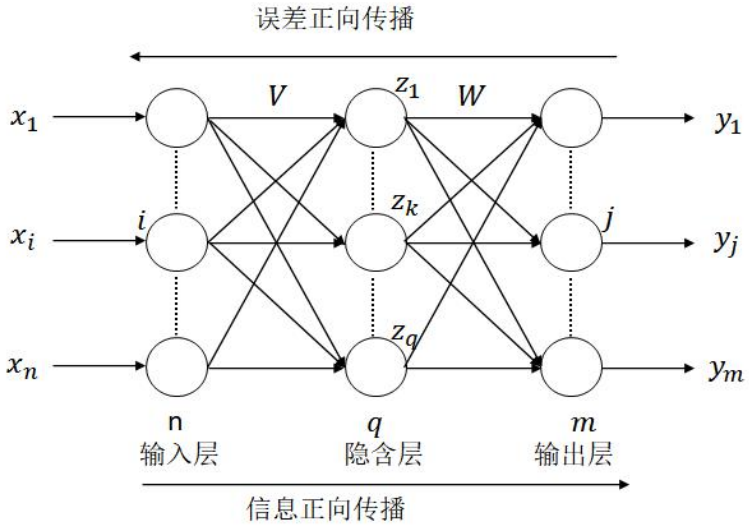


图 4-3　三层 BP 神经网络的模型结构

输入$[x_1, x_2, \cdots, x_n]$为 n 维向量，网络包含两层权值矩阵 V 和 W 和偏置矢量b^1和b^2，m 个神经元的输出组成了 m 维输出向量$[y_1, y_2, \cdots, y_n]$。

$$y = Wf(Vx + b^1) + b^2 \tag{4-13}$$

其中，隐层和权值矩阵和偏置向量的具体形式如下：

$$V = \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1n} \\ v_{21} & v_{22} & \cdots & v_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ v_{q1} & v_{q2} & \cdots & v_{qn} \end{bmatrix} \quad b^1 = \begin{bmatrix} b_1^1 \\ b_2^1 \\ \vdots \\ b_q^1 \end{bmatrix} \tag{4-14}$$

$$W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1q} \\ w_{21} & w_{22} & \cdots & w_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mq} \end{bmatrix} \quad b^2 = \begin{bmatrix} b_1^2 \\ b_2^2 \\ \vdots \\ b_m^2 \end{bmatrix} \tag{4-15}$$

BP 神经网络训练过程主要包括前向传播与误差修正两方面。在前向传播过程中，输入信息通过中间的隐含层处理后传输至输出层。每层神经元的状态仅由前一层神经元的输出决定，传输至输出层与期望输出进行比较，当与期望输出存在较大的误差时，将误差反向传播，沿路径返回采用梯度下降算法更新隐含层和输出层的权重和偏置向量，参数更新完成之后进行下一轮的传播过程，开始新的网络训练，直至预测输出与期望输出的误差达到最小。根据上述过程，BP 算法步骤可概括为下述流程框图：

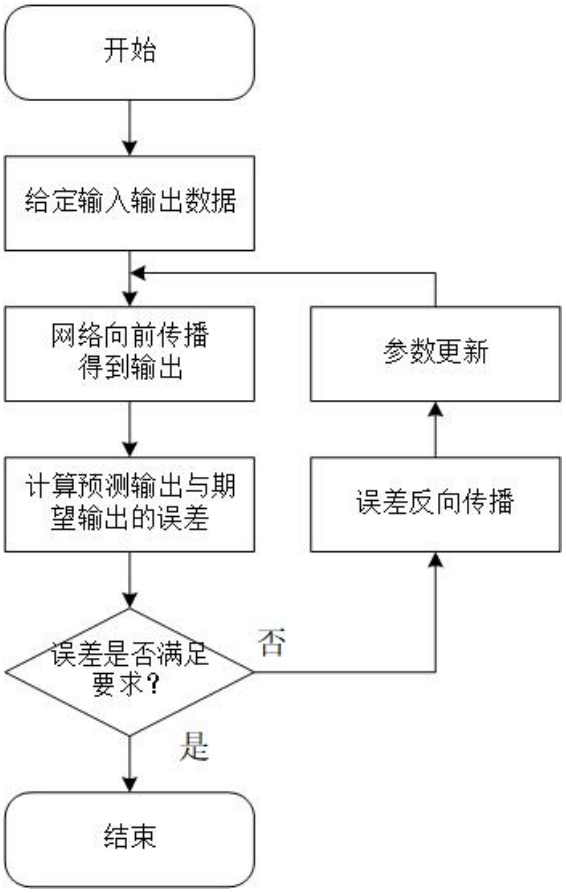


图 4-4　BP 算法的流程框图

从上图可以看出 BP 神经网络实际上是输入到输出的映射，当用模型来解决拟合预测问题时，输出为 4 个锚点到靶点的实际测距值。

4.4 基于实际场景信息和 BP 神经网络的精准定位模型构建

平滑以后的数据再用于上述 BP 神经网络模型的训练，实际场景信息则主要体现在实测距离与 Tag 坐标的转换公式上。整个模型采用 8 输入 4 输出的网络结构，总共由 7 层 BP 神经网络构成，用来拟合实测距离与真实坐标之间的强非线性关系，进而实现正常样本和异常样本的精准定位。

4.4.1　数据整理

此步骤需对任务 1 预处理后的数据进行二次调整，将前文保留出来的 4 个实测距离分别进行卡尔曼滤波和归一化处理，再将每一行作为 1 个输入样本。但利用 BP 神经网络通过输入四个距离，再直接拟合靶点的预测坐标，预测结果可能是不准确的，主要原因在于同一 Tag 对应的不同组实测数据中往往包含了许多测量噪声，且测距值与 Tag 坐标间的关系满足实际场景下的定位公式。因此为了使神经网络的预测结果尽可能准确，我们首先根据已知靶点 Tag 的位置坐标反推出对应的锚点到靶点的 4 个真实距离，再将这 4 个真实距离作为 BP 神经网络的标签。其中网络输出的预测距离相当于校正后的实测距离，当预测输出与真实的距离的误差越小，就证明模型的效果越

好。训练集和测试集按照 9：1 的比例进行划分。

4.4.2　网络结构层设计

（1）输入输出层

除了将实测距离作为输入，我们还根据实测距离求解坐标，再根据坐标求解四个新的距离。因为实测距离存在误差，实际情况下是不能确定一个实际存在的靶点，但是根据公式推导抵消坐标的二次项，可以求解得到一个靶点坐标，但是这个坐标与真实的靶点坐标存在误差，我们再根据存在误差的靶点坐标反推锚点到靶点的距离，得到与实测距离的差值，从而将误差量化。我们将实测距离与误差距离一起输入网络模型。输出为校正后的距离。故输入神经元的数目 n=8，输出神经元的个数 m=4。

（2）隐藏层

在神经网络的设计过程中，隐含层的层数与每层的神经元的数目十分重要。隐层神经元数目过多，会增大网络的计算量，甚至会产生过拟合；隐层神经元数目过少，则会影响网络的性能，达不到好的分类效果。经过反复调试，确定使用了一层输入，一层输出，中间层为 6 层隐藏层，神经网络结构如下图所示，上面的数字代表了当前层的神经元的个数：

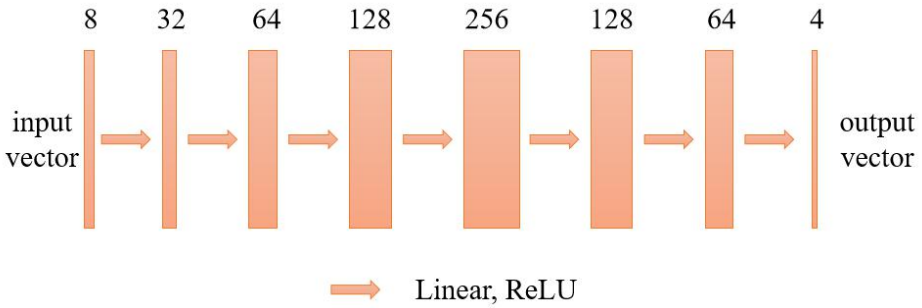


图 4-5　正常样本定位模型的网络结构层设计

4.4.3　模型实现

模型框架选在 pytorch 框架下实现，基于 BP 神经网络训练得到的定位模型的输入神经元数目为 8，中间隐藏层数目为 6，隐层神经元的数目先增大，再逐渐减少，最后输出层的神经元数目为 4。最大迭代次数为 200 次，　采用 Adam 优化器，初始学习率设置为 10e-4，损失在 5 次迭代次数不下降学习率下降到当前学习率的 1/2。

4.4 正常/异常定位模型的求解与验证

本小节将对正常样本和异常样本定位模型的预测结果进行可视化展示和分析，首先通过将数据测试文件导入到训练好的 2 个定位模型中，再根据预测的距离求出对应靶点的坐标，得到的最终预测效果分别如下图 4-6、4-7 所示。

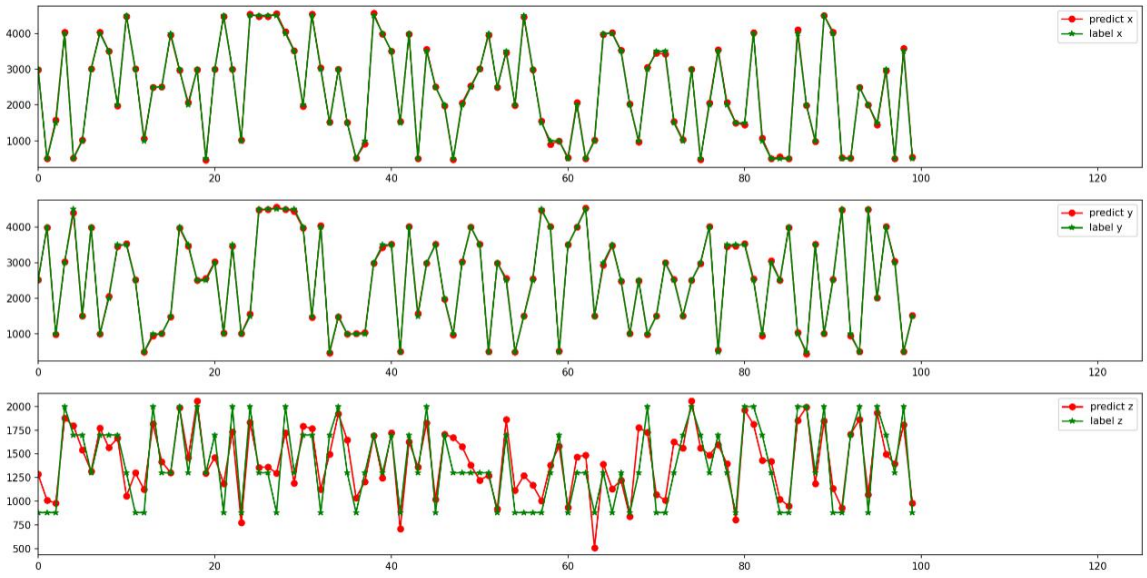


图 4-6　正常样本定位模型的 3 维 Tag 坐标预测效果

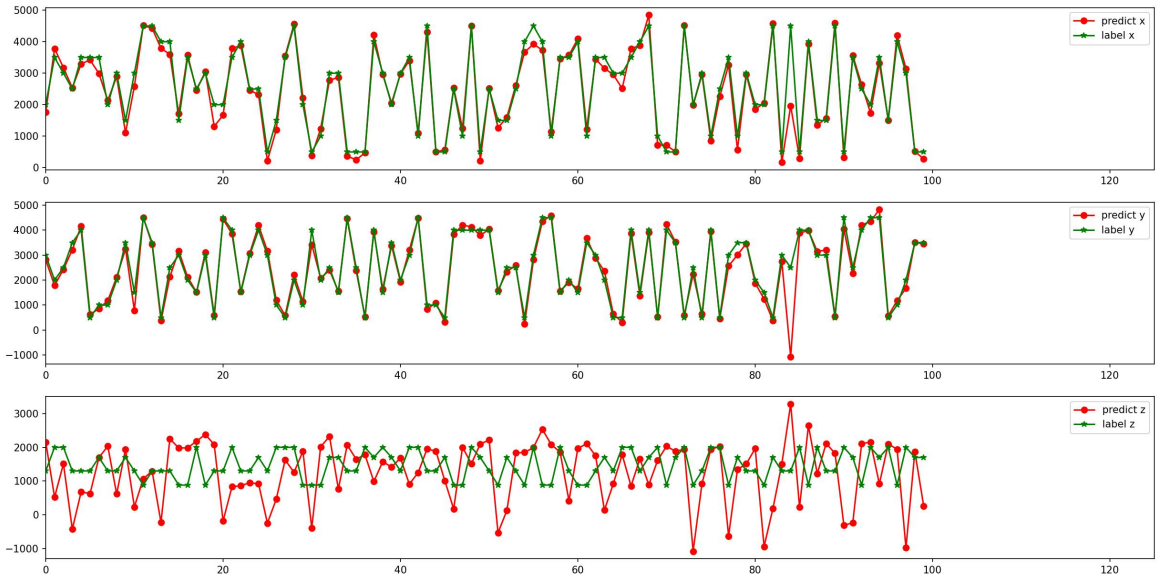


图 4-7　异常样本定位模型的 3 维 Tag 坐标预测效果

上图共展示了 100 个靶点的 x，y，z 坐标的预测结果与真实结果的拟合程度。横坐标为样本编号，纵坐标为 Tag 坐标值，其中红色线条表示预测的坐标值，绿色线条表示真的坐标值，单位取毫米。从图中可以明显的看出，多层 BP 神经网络定位模型在 x 和 y 坐标的预测上达到了较高的精度，尽管 z 坐标的预测值存在一定的偏差，但是结果也在可接受的范围内。

定位精度采用每维 MAE 的均值和方差进行衡量，分别对测试集的三维坐标（x，y，z）和二维（x，y）和一维 x、y、z 的精度进行了计算，模型精度结果如下表所示。

表 4-1　5 组无干扰样本的定位结果

维度	精度	
	MSE 均值	MSE 方差
（x，y，z）	21.87	12.80
（x，y）	5.50	2.80
x	2.92	2.16
y	2.58	1.96
z	16.37	12.25

表 4-2　5 组有干扰样本的定位精度

维度	精度	
	MSE 均值	MSE 方差
（x，y，z）	56.42	26.80
（x，y）	20.15	14.04
x	10.13	7.82
y	10.02	8.00
z	36.27	22.21

将 5 组无干扰和 5 组有干扰数据分别输入到正常定位模型和异常定位模型中，最终的 Tag 坐标求解结果如表 4-3 和 4-4 所示。

表 4-3　5 组无干扰样本的定位结果

编号	锚点 ID				预测 Tag 坐标
	锚点 1	锚点 2	锚点 3	锚点 4	(x,y,z)
1	4220	2580	3730	1450	[1189.5, 656.3, 1099.3]
2	4500	1940	4420	1460	[3192.5, 1729.2, 1078.9]
3	3550	2510	3410	2140	[2764.7, 1190.6, 1204.9]
4	3300	3130	2900	2790	[2474.7, 1035.4, 1825.9]
5	720	4520	3050	5380	[1498.8, 2520.0, 1946.2]

表 4-4　5 组有干扰样本的定位结果

编号	锚点 ID				预测 Tag 坐标		
	锚点 1	锚点 2	锚点 3	锚点 4	x	y	z
1	5100	2220	4970	800	[2106.8, 706.5, 1672.3]		
2	2900	3210	3140	2890	[4186.2, 1790.4, 1339.2]		
3	2380	3530	2320	3760	[1760.7, 1237.5, 1697.2]		
4	2150	3220	3140	3640	[3648.4, 1943.1, 1327.7]		
5	1620	3950	2580	4440	[4521.8, 2152.8, 1639.0]		

5 不同实验场景下的迁移定位应用

5.1 问题分析

在基于实验场景 1 建立的 2 个定位模型基础上，问题 3 希望其也能很好地满足在实验场景 2 下的定位精度。具体过程主要是针对附件 3 给出的在实验场景 2 下采集到的 10 组数据（前 5 组无干扰、后 5 组有干扰），再采用问题二所建立的 2 个定位模型进行测试，进而实现 3 维 Tag 坐标的精准定位。由于问题二将 4 个测距值与 3 个 Tag 坐标的拟合转化为了 4 个测距值与由 Tag 反推回来的 4 个真实距离间的拟合，即神经网络主要用来实现距离的校正功能，再融入实验场景 1 的坐标信息，进行 Tag 坐标的精准定位，因此也能较好的应用在实验场景 2 中。

5.2 模型验证

将 5 组无干扰和 5 组有干扰数据分别输入到正常定位模型和异常定位模型中，最终的 Tag 坐标求解结果如表 5-1 和 5-2 所示。

表 5-1　5 组无干扰样本的定位结果

编号	锚点 ID				预测 Tag 坐标
	锚点 1	锚点 2	锚点 3	锚点 4	(x,y,z)
1	4220	2580	3730	1450	[3664.5, 2175.3, 1212.8]
2	4500	1940	4420	1460	[4109.5, 1685.9, 1349.2]
3	3550	2510	3410	2140	[3178.0, 1714.1, 1221.2]
4	3300	3130	2900	2790	[2587.1, 1878.9, 1482.4]
5	720	4520	3050	5380	[737.8, -45.7, 1354.3]

表 5-2　5 组有干扰样本的定位结果

编号	锚点 ID				预测 Tag 坐标
	锚点 1	锚点 2	锚点 3	锚点 4	(x,y,z)
1	5100	2220	4970	800	[4237.8, 2348.6, 1514.6]
2	2900	3210	3140	2890	[2463.8, 1579.3, 1350.8]
3	2380	3530	2320	3760	[1757.4, 1414.4, 1185.8]
4	2150	3220	3140	3640	[2087.4, 858.7, 1370.6]
5	1620	3950	2580	4440	[1172.3, 786.7, 1259.2]

6 基于多层 BP 神经网络的分类模型

6.1 问题分析

任务四要求基于上述预处理后的数据样本，建立正确的分类模型，在不知道信号是否存在干扰的情况下，能进行准确分类，判断信号是否存在干扰。直接观察采集到的距离数据不能直接看出是否是在信号受到干扰的情况下采集的。由于存在四个锚点，每个锚点都可能会受到信号干扰，变量之间存在高度非线性关系。BP 神经网络是一种典型的反向传播神经网络，通过训练不断修正网络权值和阈值使误差沿负梯度方向下降，具有很强的非线性映射能力，非常适合解决一些非线性问题，可用于函数逼近、分类等任务。另外网络拓扑结构简单，具有较高的计算精度。在此任务中，我们从 648 组数据文件中抽取 90%的数据用于 BP 神经网络的训练，剩下 10%的数据对模型精度进行验证以评价模型的合理性。问题四的思路流程如图所示。

6.2 信号干扰分类模型建立

6.2.1 数据整理

考虑到异常数据在同一时刻只有一个锚点受到影响，无法直接根据正常数据和异常数据的测距值准确的判断信号是否存在干扰，此处需要对附件一预处理过的 648 个数据文件进行处理。由于附件一提供了 324 个靶点的实际物理坐标，四个锚点的物理坐标也是已知的，我们可以根据三维空间通过欧式距离计算公式得到靶点到四个锚点的真实距离。信号在受到干扰的情况下得到的实测距离应该是存在较大误差的，所以正常测距值和异常测距值与真实距离的误差应该存在很大差异，我们考虑将实测值与真实距离的差值作为输入。考虑到用 BP 神经网络做二分类模型，我们的输出值为单一输出，正常数据的标签为 1，异常数据的标签为 0。

6.2.2 网络结构层设计

（1）输入输出层：将前文提取出来的四个距离差值作为输入，输出为单一输出，故输入神经元的数目 n=4，输出神经元的个数 m=1。

（2）隐层：在神经网络的设计过程中，隐含层的层数与每层的神经元的数目十分重要。隐层神经元数目过多，会增大网络的计算量，甚至会产生过拟合；隐层神经元数目过少，则会影响网络的性能，达不到好的分类效果。经过反复调试，确定使用了一层输入，一层输出，中间层为 6 层隐藏层，神经网络结构如下图所示，上面的数字代表了当前层的神经元的个数：

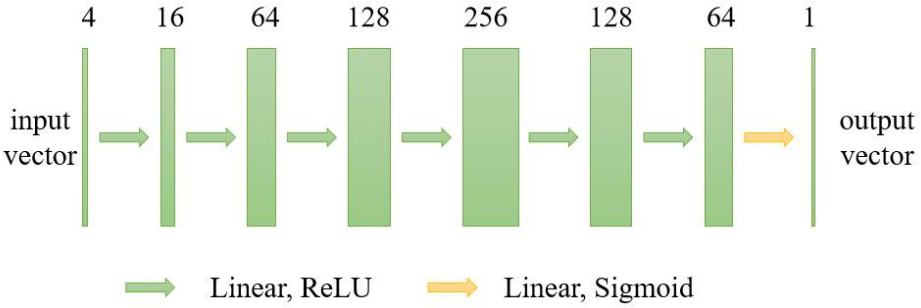


图 6-1　有无干扰分类模型的网络结构层设计

6.2.3　激活函数的选取

激活函数的一个重要作用是加入非线性因素，解决线性模型所不能解决的问题，如果输出层之前不加激活函数，模型就只能解决线性可分的问题，无法解决非线性问题。目前深度学习常用的激活函数有两种 Sigmoid 激活和 Tanh 激活。Sigmoid 激活是使用范围最广的一类激活函数，它在物理上最接近神经元的输出，输出范围在(0,1)之间，可以被表示成概率。此处我们训练的是分类模型，我们选择了 Sigmoid 激活函数输出一个概率，选定阈值为 0.5，概率大于 0.5 的数据认为是正常数据，不受信号干扰，概率小于 0.5 的数据认为是异常数据，在测距过程中收到了信号干扰。Sigmoid 激活函数的定义如下：

$$f(x)=\frac{1}{1+e^{-x}} \tag{6-1}$$

6.2.4　模型实现

最终我们训练的分类模型是基于 BP 神经网络，输入神经元的数目为 4，中间隐层有 6 层，隐层神经元的数目先增大，再逐渐减少，最后输出层的激活函数采用 Sigmoid 激活，输出一个概率值。模型是在 pytorch 框架下实现的，最大迭代次数为 100 次， 采用 Adam 优化器，初始学习率设置为 5×10e-5，损失在 5 次迭代次数不下降学习率下降到当前学习率的 1/2。模型训练过程中的损失变化情况如下：

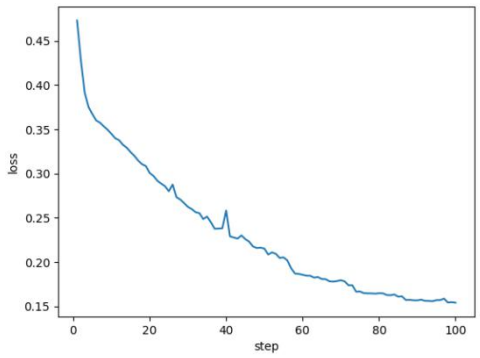


图 6-2　模型训练过程中的损失变化情况

6.2.5 分类模型评价指标

对于二分类问题，我们将结果分成四类分别为 TP（实际数据无干扰，预测数据

无干扰），FP（实际数据有干扰，预测数据无干扰），TN（实际数据有干扰，预测数据有干扰），FN（实际数据无干扰，预测数据有干扰）。根据得到的 TP，FP，TN，FN 的数据，我们采用 ROC 曲线直观评价分类模型。曲线的横坐标为假阳性率（FPR），纵坐标为真阳性率（TPR）。其中：

$$FPR = \frac{FP}{(TN+FP)} \tag{6-2}$$

$$TPR = \frac{TP}{TP+FN} \tag{6-3}$$

6.3 模型验证

6.3.1 信号干扰分类模型验证

将测试集导入训练好的信号干扰分类模型，得到对测试集的分类结果，我们已知测试集的正确标签，得出对应的 TP、FP、TN、FN 的数目，并绘制出混淆矩阵和 ROC 曲线，结果如图 6-3 和 6-4 所示：

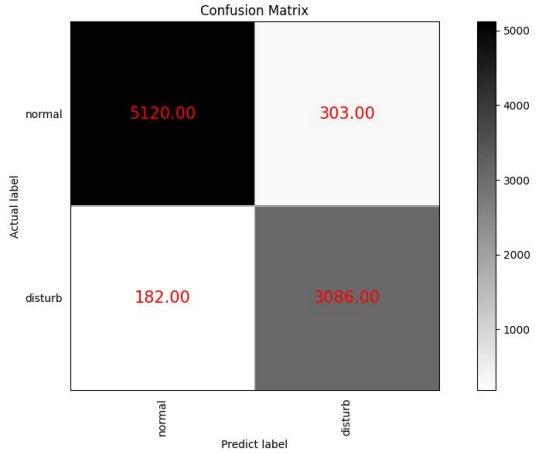


图 6-3　有无干扰分类模型的混淆矩阵

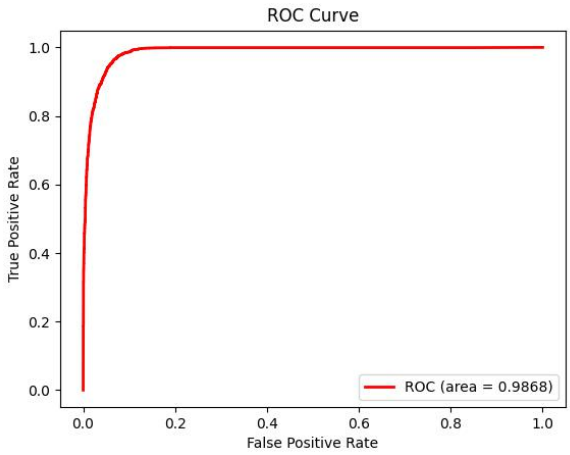


图 6-4　有无干扰分类模型的 ROC 曲线

可以看出分类模型的效果在测试集上的效果十分良好，同时再将分类模型应用到附件 4 的数据中，10 个预测数据的分类分别为正常数据、异常、异常、异常、异常、正常、正常、异常、异常、正常。

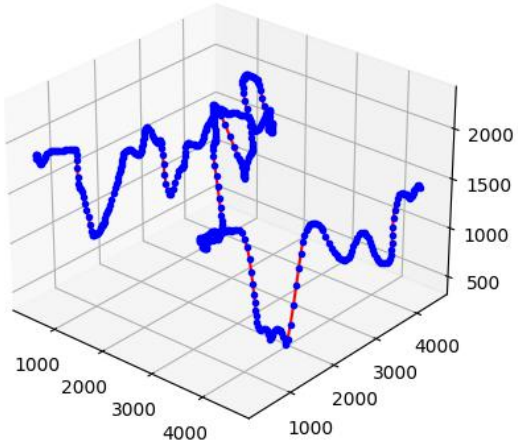
7 动态轨迹定位

7.1 问题分析

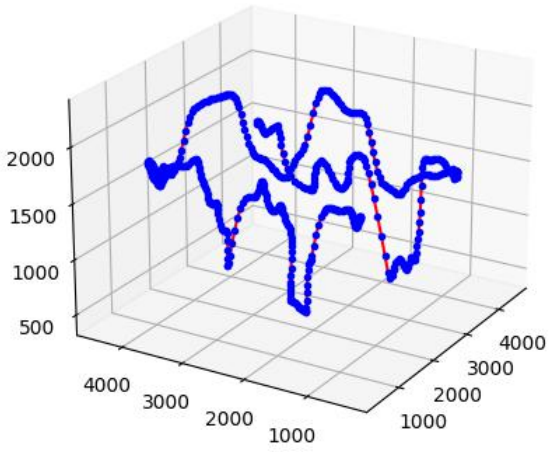
任务 5 是动态靶点的运动轨迹可视化，需挖掘出其中的靶点运动规律，再结合之前构建好的静态定位模型进行动态靶点的精准定位。具体操作为：首先基于附件 5 给出的一组连续采集到的动态靶点数据，我们采用任务 1 提出的预处理方法进行数据的提取和清洗，然后采用任务 4 得到的分类模型对数据进行 2 分类，将其分为正常无干扰数据和异常有干扰数据，再按照样本是否受干扰选择任务 2 中相应构建好的定位模型，预测动态靶点的坐标并绘制轨迹曲线。其中预处理中的卡尔曼平滑操作通过滤除靶点运动过程中的噪声，可以更好地实现正常/异常样本切换时的精准定位。

7.2 动态轨迹可视化

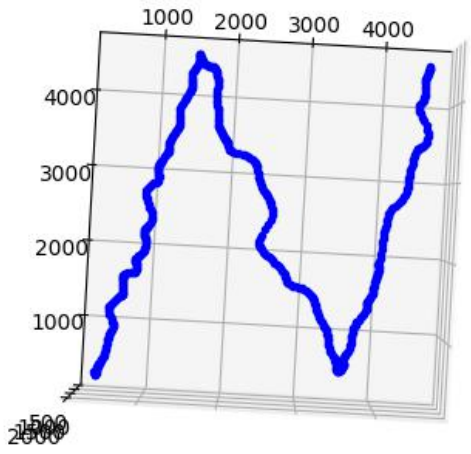
不同视图的动态靶点运动轨迹如图 7-1 所示，可以看出，采集到的多组数据对应的靶点三维坐标也在不断变化。



（a）视图 1



（b）视图 2



（c）视图 3

图 5-1　动态靶点运动轨迹可视化

8 模型评价

8.1 模型的优点

（1）任务 2 的多层 BP 定位模型充分考虑了各变量之间的非线性关系，通过预测靶点到锚点的校正距离的方法间接预测靶点的坐标，比通过模型直接预测靶点坐标能达到较高的预测精度。

（2）任务 4 的分类模型通过输入实测距离与校验距离的差值，充分考虑了正常数据与异常数据的差别，达到了较好的分类效果。

（3）定位模型与分类模型均采用 BP 神经网络。算法速度快，精度高。

（4）模型具有较好的泛化能力。给到一个新的实测数据，当不确定是否存在信号干扰的情况下，可以先根据分类模型对数据进行分类，再选取正常定位模型或异常定位模型预测靶点坐标，对真实场景中靶点坐标的确定具有实际应用价值。

8.2 模型的缺点

（1）数据预处理过程中，假定所有数据服从正态分布，实际场景中数据可能不遵循正态分布原则。

（2）定位模型在 x，y 坐标的预测上精度较高，在 z 坐标的预测上还存在一定的误差，在实际应用中还有很大提升的空间。

（3）分析问题时发现异常样本中干扰锚点的出现位置不同，可能会对神经网络的拟合效果造成影响，综合考虑干扰锚点的识别和 BP 神经网络和非线性拟合效果，结果会更加科学和准确。

参考文献

[1] 王伟. "基于卡尔曼滤波和加权 LM 法的井下精确定位算法." 工矿自动化 45.11 (2019): 5-9.

[2] Promwong S, Thongkam J. Evaluation of weighted impulse radio for ultra-wideband localization[J]. Wireless Personal Communications, 2020, 115(4): 2695-2704.

[3] Wang C, Ning Y, Li X, et al. A Robust Unscented Kalman Filter applied to Ultra-wideband Positioning[J]. International Journal of Image and Data Fusion, 2020, 11(4): 308-330.

[4] Staal O M , Saelid S , Fougner A L , et al. Kalman smoothing for objective and automatic preprocessing of glucose data[J]. IEEE Journal of Biomedical & Health Informatics, 2018:1-1.

[5] H. E. Rauch, C. Striebel, and F. Tung, “Maximum likelihood estimates of linear dynamic systems,” AIAA journal, vol. 3, no. 8, pp. 1445‐1450, 1965.

附录 1

程序代码：

1、数据处理

```
import os
import re
import numpy as np
import pandas as pd
from os import listdir
from collections import Counter
def missing_data_processing(content,j):
    a = []
    time = re.compile(r'T:(\w+):RR').findall(content)
    b = list(Counter(time))
    for i in b:
        if (Counter(time)[i]== 4):
            a.append(i)
    if (int(j)==109):
        print(len(a))
    return a

def three_sigmal(data):
    n = 3    # n*sigma
    print(data)
    ymean = np.mean(data, axis=0)   # 均值
    ystd = np.std(data, axis=0)     # 标准差
    threshold1 = ymean - n * ystd
    threshold2 = ymean + n * ystd
    outlier = []   # 将异常值保存
    final = []     # 删除掉异常值后的数据
    for i in range(0, len(data)):
        #print(np.abs(data[i]-ymean))
        if (np.abs(data[i]-ymean) > n * ystd).any() :     #
            outlier.append(data[i])
        else:
            final.append(data[i])
    return outlier

normal_data_path = 'D:/竞赛/数学建模/2021 年 E 题/附件 1：UWB 数据集/正常数据'
abnormal_data_path = 'D:/竞赛/数学建模/2021 年 E 题/附件 1：UWB 数据集/异常数据'
label_path = 'D:/竞赛/数学建模/2021 年 E 题/附件 1：UWB 数据集/Tag 坐标信息.txt'
filename1 = listdir(normal_data_path)
filename2 = listdir(abnormal_data_path)
```

```
filename1.sort(key=lambda x:int(x.split('.')[0]))
filename2.sort(key=lambda x:int(x.split('.')[0]))
# 整合标签数据
label = []
with open(label_path, encoding='utf-8') as f:
    line = f.readline()  # 以行的形式进行读取文件
    while line:
        a = line.split()
        b = a[1:4]  # 这是选取需要读取的位数
        label.append(b) # 将其添加在列表之中
        line = f.readline()
    f.close()
## 整合正常、异常数据
normal_df = extract_data(normal_data_path,filename1,label)
abnormal_df = extract_data(abnormal_data_path,filename2,label)
## 删除重复数据
x1 = normal_df.drop_duplicates()
x2 = abnormal_df.drop_duplicates()
print(abnormal_df['measure1'][abnormal_df['filename']==str(100)])
print(x2['measure1'][x2['filename']==str(100)])
```

2、定位模型

```
import torch
from torch import nn as nn

class CL_Model(nn.Module):
    def __init__(self):
        super(CL_Model, self).__init__()
        self.Liner = nn.Sequential(nn.Linear(8, 32),
                                   nn.ReLU(inplace=True),
                                   nn.Linear(32, 64),
                                   nn.ReLU(inplace=True),
                                   nn.Linear(64, 128),
                                   nn.ReLU(inplace=True),
                                   nn.Linear(128, 256),
                                   nn.ReLU(inplace=True),
                                   nn.Linear(256, 128),
                                   nn.ReLU(inplace=True),
                                   nn.Linear(128, 64),
                                   nn.ReLU(inplace=True),
                                   nn.Linear(64, 4))

    def forward(self, x):
```

```
            x = self.Liner(x)
            return x

log_file = 'log/%s.txt' % TAG
sys.stdout = Logger(log_file, sys.stdout)
sys.stderr = Logger(log_file, sys.stderr)    # redirect std err, if necessary
config = Config()
model = CL_Model()
model.cuda()
optimer = optim.Adam(model.parameters(), lr=config.lr)
schedule = optim.lr_scheduler.ReduceLROnPlateau(optimer, mode='max',
factor=config.factor, patience=config.patient, verbose=True)
LOSS = nn.BCELoss()
train_loader = get_loader(mode='train', batch_size=config.batch_size)
test_loader = get_loader(mode='test', batch_size=config.batch_size)
best_acc = 0
best_epoch = 0
for epoch in range(config.epoch):
    train_loss = train(model, train_loader, optimer, LOSS, epoch, config)
    test_acc = test(model, test_loader, epoch, config)
    save_model(model, epoch, test_acc, best_acc, schedule, optimer)
    if test_acc > best_acc:
        best_epoch = epoch
        best_acc = test_acc
    print('{} Epoch [{:03d}/{:03d}], train_loss:{:.4f}, test_acc:{:.4f},
best_epoch:{:03d}, best_acc:{:.4f}'.
          format(datetime.now(), epoch, config.epoch, train_loss, test_acc,
best_epoch, best_acc))
    schedule.step(test_acc)
```

3、分类模型

```
import torch
from torch import nn as nn

class CL_Model(nn.Module):
    def __init__(self):
        super(CL_Model, self).__init__()
        self.Liner = nn.Sequential(nn.Linear(4, 16),
                                   nn.ReLU(inplace=True),
                                   nn.Linear(16, 64),
                                   nn.ReLU(inplace=True),
                                   nn.Linear(64, 128),
                                   nn.ReLU(inplace=True),
                                   nn.Linear(128, 256),
```

```
                    nn.ReLU(inplace=True),
                    nn.Linear(256, 128),
                    nn.ReLU(inplace=True),
                    nn.Linear(128, 64),
                    nn.ReLU(inplace=True),
                    nn.Linear(64, 1),
                    nn.Sigmoid())

    def forward(self, x):
        x = self.Liner(x)
        return x
```