



计算智能导论实验报告

|  |  |
| --- | --- |
| 名 称： | 计算智能导论 |
| 学生姓名： |  |
| 学 号： | 2022905226 |
| 专 业： | 人工智能 |
| 班 级： | 2022240401 |
| 指导教师： |  |
| 时 间： | 2024.10.1 |

[**一、实验目的** 3](#_Toc179203668)

[**二、实验内容** 3](#_Toc179203669)

[**三、实验原理** 3](#_Toc179203670)

[**3.1** **人工神经网络与BP神经网络** 3](#_Toc179203671)

[**3.2** **BP神经网络拟合正弦函数** 5](#_Toc179203672)

[**3.2.1** **BP神经网络结构** 5](#_Toc179203673)

[**3.2.2** **激活函数** 6](#_Toc179203674)

[**3.2.3** **损失函数** 6](#_Toc179203675)

[**3.2.4** **优化算法** 7](#_Toc179203676)

[**3.2.4.1** **为什么实际算法中需要优化器？** 7](#_Toc179203677)

[**3.2.4.2** **Adam优化器** 7](#_Toc179203678)

[**3.2.5** **白噪声** 8](#_Toc179203679)

[**3.2.6** **模型评估** 8](#_Toc179203680)

[**3.2.7** **可视化** 9](#_Toc179203681)

[**3.3** **BP神经网络公路运力预测** 9](#_Toc179203682)

[**3.3.1** **网络结构** 9](#_Toc179203683)

[**3.3.2** **激活函数、损失函数与优化算法** 9](#_Toc179203684)

[**3.3.3** **数据预处理** 9](#_Toc179203685)

[**3.3.4** **模型评估** 9](#_Toc179203686)

[**3.3.5** **过拟合防护** 10](#_Toc179203687)

[**四、实验步骤** 10](#_Toc179203688)

[**4.1** **BP神经网络拟合正弦函数** 10](#_Toc179203689)

[**4.2** **BP神经网络公路运力预测** 11](#_Toc179203690)

[**五、实验结果与分析** 13](#_Toc179203691)

[**5.1** **BP神经网络拟合正弦函数** 13](#_Toc179203692)

[**5.2** **BP神经网络公路运力预测** 13](#_Toc179203693)

[**5.2.1** **客运量模型** 14](#_Toc179203694)

[**5.2.2** **货运量模型** 14](#_Toc179203695)

[**5.2.3** **预测结果** 14](#_Toc179203696)

[**5.2.4** **拟合图和误差分析图示** 14](#_Toc179203697)

[**六、实验程序代码（部分关键代码）** 15](#_Toc179203698)

[**6.1** **BP神经网络拟合正弦函数** 15](#_Toc179203699)

[**6.2** **BP神经网络公路运力预测** 17](#_Toc179203700)

# **一、实验目的**

掌握人工神经网络的特性和应用范围，能够用人工神经网络解决 实际预测问题，能够根据应用需求，设计与实现人工神经网络，观测 神经网络的训练过程和输出。

# **二、实验内容**

1、训练 BP 神经网络，使其能够拟合某一附加有白噪声的正弦 样本数据。最后将带有白噪声的正弦样本数据点、标准正弦曲线以及 拟合后的正弦曲线进行对比。

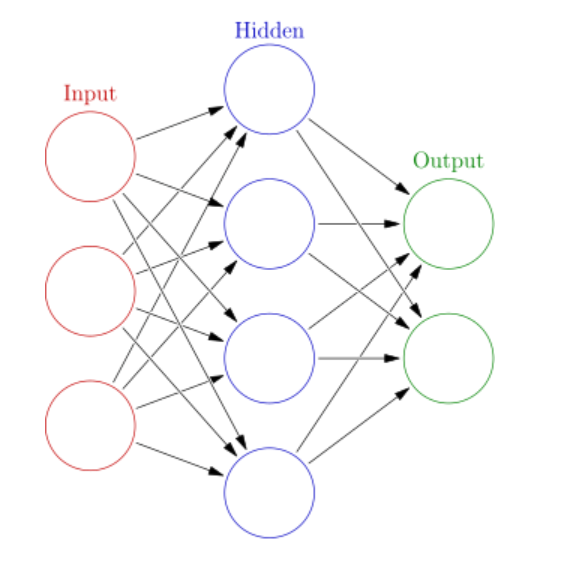
2、下表所示为某地区公路运力的历史统计数据表

（1）请根据该表，建立相应的 BP 神经网络预测模型，拟合历 年公路客运量曲线和历年公路货运量曲线。

（2）根据给出的 2010 年和 2011 年的数据，预测相应的公路客 运量和货运量。

# **三、实验原理**

## **3.1 人工神经网络与BP神经网络**

**人工神经网络（Artificial Neural Network，ANN**是一种受生物神经网络启发的计算模型。它由大量相互连接的人工神经元组成，可以处理复杂的非线性关系。一般的神经网络结构如下：

**图1：一般人工神经网络结构图**

由输入层、隐藏层和输出层组成，每层由多个神经元组成，神经元之间通过权重连接。 输入特征向量经过隐藏层处理后，通过激活函数在输出层输出结果。每层的神经元通过权重连接，每个神经元的输出由输入特征向量和权重决定。

**BP（反向传播）神经网络**是一种多层前馈神经网络，通过反向传播算法进行训练。核心思想是通过**梯度下降法最小化网络的损失函数**。

BP神经网络的一般训练过程：

* **前向传播**：输入数据从输入层经过隐藏层，最后到达输出层。特征向量经过每个神经元时，通过权重和偏置计算得到输出，计算公式遵循：
* 其中， 是神经元j的输出， 是激活函数， 是连接权重， 是输入， 是偏置。
* **计算损失**：将网络输出与目标值比较，计算损失。损失函数有多种，比如使用均方误差（MSE）遵循：
* 其中， 是样本数， 是真实值， 是预测值。
* **反向传播**：计算**损失函数对各层权重和偏置的梯度**。使用**链式法则**可以计算每层的误差项 ：
* 对于输出层：
* 对于隐藏层：
* 其中， 是第 层第 个神经元的加权输入， 是激活后的输出。
* **参数更新**：根据计算得到的梯度，使用优化算法更新网络参数。一般使用梯度下降：
* 其中， 是学习率。根据上式可以看出，更新的原则简单来讲就是**新的权重等于旧的权重减去学习率乘以损失函数对权重的偏导数**。由此也可以得出，损失函数对权重的偏导数越大，则更新的权重越小。学习率越大，则更新的权重越大。

经过多次迭代，直到损失函数收敛到最小值，BP神经网络训练完成。

## **3.2 BP神经网络拟合正弦函数**

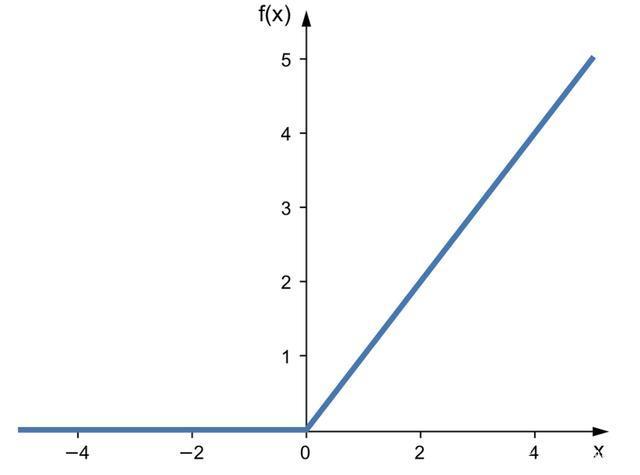
### **3.2.1 BP神经网络结构**

在拟合正弦函数任务中使用的BP神经网络结构： 对于正弦函数，输入层、输出层都只有1个节点。为了拟合更加精确，设置了3个隐藏层，每层分别有20、20、10个节点，使用ReLU作为激活函数。

* 输入层：1个节点（x值）
* 隐藏层1：20个节点，使用ReLU激活函数
* 隐藏层2：20个节点，使用ReLU激活函数
* 隐藏层3：10个节点，使用ReLU激活函数
* 输出层：1个节点（预测的y值）

### **3.2.2 激活函数**

使用ReLU（Rectified Linear Unit）作为激活函数。ReLU函数的数学表达式：

函数图像:

**图2：ReLU函数图像**

ReLU函数的优点是计算简单，能够有效缓解梯度消失问题，加速网络收敛。

### **3.2.3 损失函数**

使用均方误差（MSE）作为损失函数。对于预测值和真实值，MSE的计算公式为：

其中为样本数量。

### **3.2.4 优化算法**

在BP神经网络中，我们的目标是最小化损失函数，使得网络的预测结果尽可能接近真实值。因此需要不断调整网络的权重和偏置。这个调整过程是通过优化算法来完成的。

#### **3.2.4.1 为什么实际算法中需要优化器？**

1. **自动调整学习率**：简单的梯度下降法使用固定的学习率，可能导致收敛速度慢或不稳定。优化器可以自动调整学习率，使训练更加高效。
2. **处理稀疏梯度**：在某些情况下，梯度可能非常稀疏（大部分为零），优化器可以更好地处理这种情况。
3. **避免局部最小值**：优化器可以帮助网络跳出局部最小值，找到更好的全局最小值。

#### **3.2.4.2 Adam优化器**

Adam（Adaptive Moment Estimation）是一种常用的优化算法，本实验中主要使用Adam优化器。 Adam优化器的主要作用就是根据损失函数对各个参数的梯度，动态调整各个参数的学习率。具体是通过计算梯度的指数移动平均来实现的。

Adam优化器的更新规则：

参数解释：

- ：梯度的指数移动平均（一阶矩）

- ：梯度平方的指数移动平均（二阶矩）

- ：当前梯度

- , ：衰减率（通常 , ）

- ：学习率（通常初始值设为0.001）

- ：小常数，防止除零错误（通常设为）

- ：当前迭代次数

公式中的 代表网络的参数。包括**网络中的所有权重（weights）和偏置（biases）**。下标 表示当前的时间步或迭代次数。在Adam优化器的更新规则中， 是我们要优化的目标。

训练神经网络实际上就是不断调整这些参数（），使网络的输出越来越接近真实值。训练完成后，这些参数包含了网络从训练数据中学到的“知识”。

在Adam优化器的更新规则中：

**新的参数值（）等于旧的参数值（）减去一个更新量，这个更新量由学习率（）和经过修正的一阶矩估计（）与二阶矩估计（）计算得出**。

上述公式描述了如何根据计算出的梯度信息来更新网络的参数，使得损失函数不断减小，网络性能不断提升。

Adam优化器在工作过程中会记住之前看到的梯度（）和梯度的平方（）。就像是在学习过程中保持一定的”惯性”。之后根据历史梯度的大小，自动调整每个参数的学习步长。如果某个参数的梯度一直很小，Adam会增大它的学习步长；反之亦然。

在训练初期，由于缺乏历史信息，Adam会对估计值进行修正（ 和 ），以避免初始估计的偏差。最后根据修正后的估计值来更新网络参数。更新的方向由 决定，而步长则由 调整。

通过上述方式，Adam可以为每个参数自动选择合适的学习率，使得网络训练更加高效和稳定。就像是一个聪明的学习者，能够根据过去的经验和当前的情况，灵活地调整自己的学习策略。

### **3.2.5 白噪声**

为了模拟实际情况中的测量误差，在标准正弦函数上添加了高斯白噪声。白噪声的数学表达式为：

其中表示均值为0，方差为的高斯分布。在实验中，设置。

### **3.2.6 模型评估**

使用以下指标评估模型的最终效果：

* 均方误差（MSE）：衡量预测值与真实值之间的平均平方差。
* 均方根误差（RMSE）：MSE的平方根，与因变量的尺度一致。
* 平均绝对误差（MAE）：预测值与真实值之间的平均绝对差。

### **3.2.7 可视化**

通过可视化来直观地比较带有白噪声的正弦样本数据点、标准正弦曲线以及BP神经网络拟合后的曲线。直观地理解网络的拟合效果，并观察网络是否成功地学习了underlying的正弦函数，同时对噪声具有一定的鲁棒性。

## **3.3 BP神经网络公路运力预测**

在公路运力预测任务中，使用BP神经网络来建立**多输入单输出的回归模型**。模型捕捉输入特征（人数、机动车数量、公路面积）与输出（客运量或货运量）之间的非线性关系。

### **3.3.1 网络结构**

设计一个前馈神经网络：

* 输入层：3个节点，对应三个输入特征
* 隐藏层1：16个节点，使用ReLU激活函数
* 隐藏层2：8个节点，使用ReLU激活函数
* 输出层：1个节点，对应预测值（客运量或货运量）

### **3.3.2 激活函数、损失函数与优化算法**

激活函数仍然采用**ReLU函数**。 使用**均方误差（MSE）作为损失函数**：

其中 是真实值， 是预测值， 是样本数量。

优化器仍使用**Adam**（Adaptive Moment Estimation）优化器，动态更新参数的学习率。

### **3.3.3 数据预处理**

为了提高模型的训练效果和泛化能力，对输入数据进行了归一化处理。使用**MinMaxScaler**将所有特征缩放到[0, 1]区间：

加速梯度下降过程并提高模型的数值稳定性。

### **3.3.4 模型评估**

使用多个指标来评估模型性能：

1. 平均绝对误差（MAE）：
2. 均方误差（MSE）：
3. 均方根误差（RMSE）：
4. 决定系数（R²）：

其中， 是 的平均值。

为了更好地理解模型性能，计算归一化的误差指标，将误差除以目标变量的平均值：

归一化指标可以将相对误差百分比表示，使得不同尺度的预测任务（如客运量和货运量）可以直接比较。

### **3.3.5 过拟合防护**

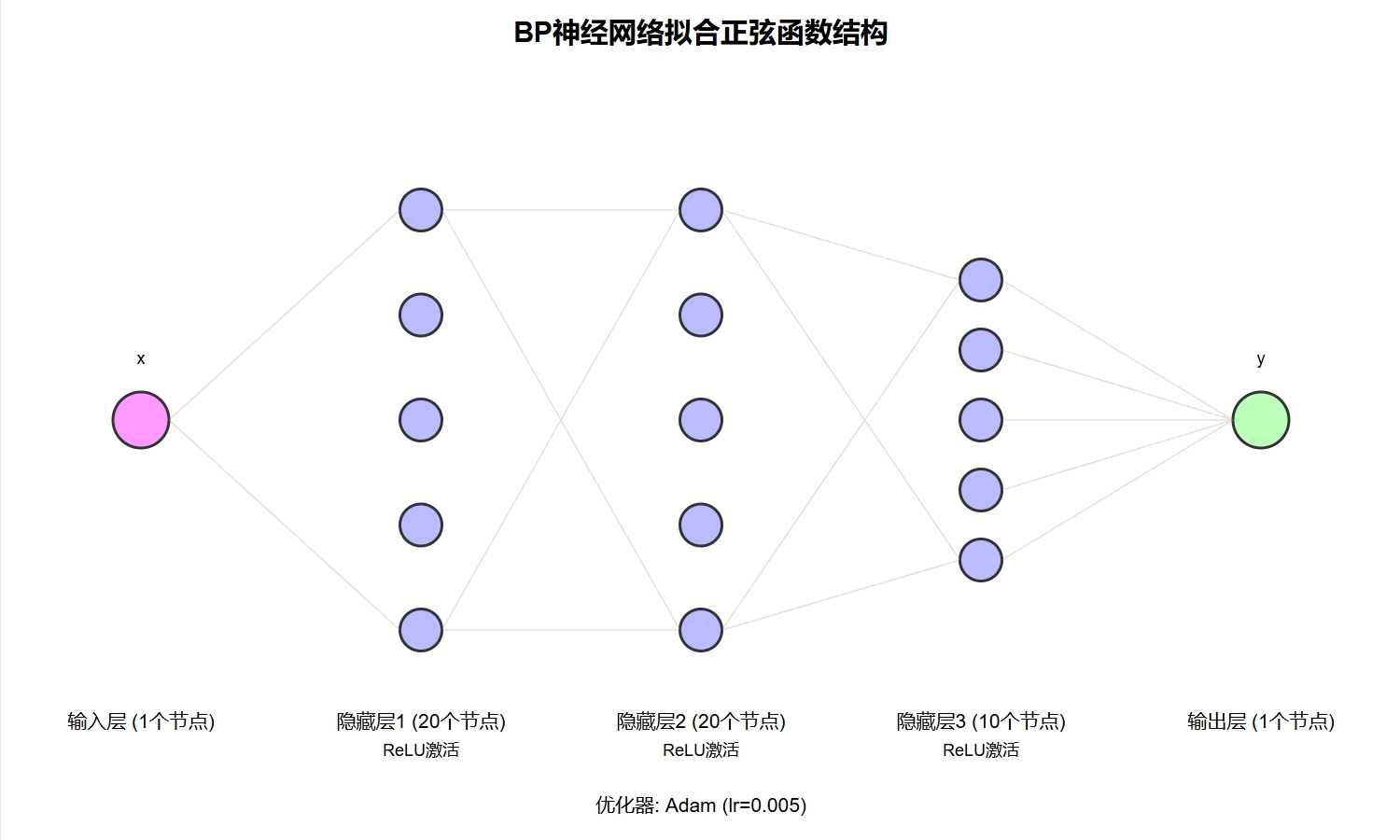
为了防止过拟合，采用以下策略：

1. **Dropout**：在第一个隐藏层后添加dropout层，**随机丢弃10%的神经元**，增强模型的泛化能力。
2. L2正则化：在Adam优化器中设置weight\_decay参数为1e-5，对大的权重值进行惩罚。

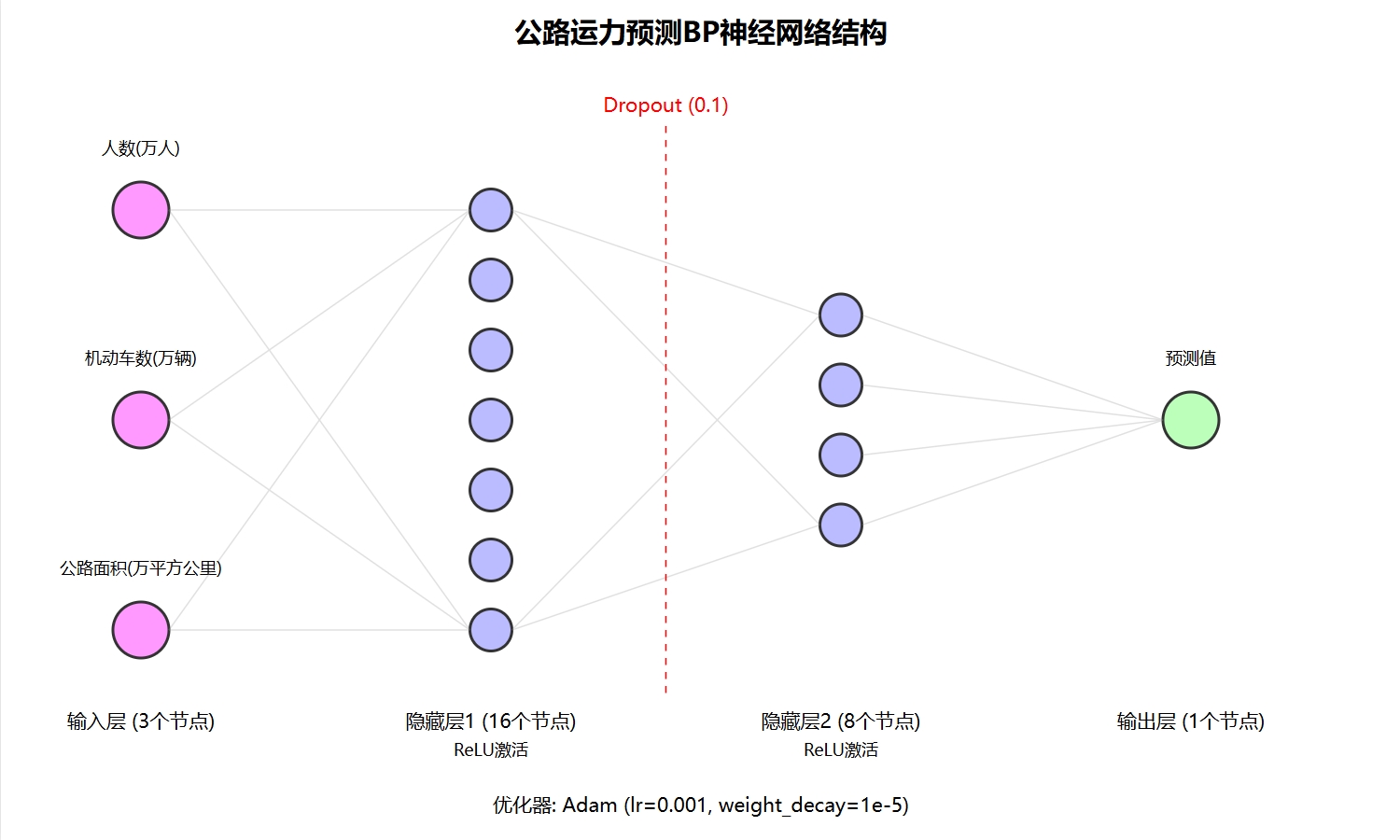
通过上述步骤减少模型对训练数据的过度依赖，提高其在未见数据上的表现。设计的BP神经网络模型能够有效地学习公路运力数据中的复杂模式，并在客运量和货运量预测任务上取得良好的性能。

# **四、实验步骤**

## **4.1 BP神经网络拟合正弦函数**

1. 数据准备
   * 使用 generate\_noisy\_sine\_data() 函数生成带有白噪声的正弦数据。
   * 白噪声中的参数设置为0.1。
   * 生成 200 个点，x 范围为 0 到 2π。
2. 定义神经网络模型
   * 使用 PyTorch 定义 BPNeuralNetwork 类，包含三个隐藏层。
   * 网络结构：输入层(1节点) -> 隐藏层1(20节点) -> 隐藏层2(20节点) -> 隐藏层3(10节点) -> 输出层(1节点)。
   * 
   * **图3：正弦拟合神经网络结构**
   * 使用 ReLU 激活函数。
3. 训练模型
   * 定义损失函数（MSE）和优化器（Adam）。
   * 在训练循环中进行前向传播、计算损失、反向传播和参数更新。
   * 训练 2000 个 epoch，每 100 个 epoch 打印一次损失。
4. 评估模型
   * 使用训练好的模型在测试数据上进行预测。
   * 计算 MSE、RMSE 和平均误差。
5. 可视化结果
   * 绘制原始数据点、标准正弦曲线和模型拟合曲线。
   * 使用 matplotlib 创建图表，展示拟合效果。

## **4.2 BP神经网络公路运力预测**

1. 数据准备和预处理
   * 从**CSV文件**加载公路运力历史数据。
   * 检查并**处理缺失值**，确保数据完整性。
   * 选择**输入特征（人数、机动车数、公路面积**）和**目标变量（客运量、货运量）**。
   * 使用MinMaxScaler对数据进行**归一化处理**，将所有特征缩放到[0, 1]区间，提高模型训练效率和数值稳定性。
2. 定义神经网络模型
   * 设计三层前馈神经网络：输入层(3节点) -> 隐藏层1(16节点) -> 隐藏层2(8节点) -> 输出层(1节点)。
   * 
   * **图4：公路运力预测神经网络结构图**
   * 使用**ReLU激活函数**。
   * 添加**Dropout层（丢弃率0.1）防止过拟合**，提高模型泛化能力。
3. 模型训练
   * 创建客运量和货运量两个独立模型实例。
   * 使用均方误差（MSE）作为损失函数。
   * 采用**Adam优化器**，设置学习率为0.001，**权重衰减为1e-5以进行L2正则化**。
   * 训练2000个epoch，每100个epoch打印一次损失值，监控训练进度。
4. 模型评估
   * 在训练数据上进行预测，并将结果转换回原始尺度。
   * 计算并输出多个评估指标：**MAE、MSE、RMSE、R²，以及它们的归一化版本**。
   * 绘制拟合曲线和误差分析图，直观展示模型性能。
5. 预测未来数据
   * 使用训练好的模型预测2010年和2011年的客运量和货运量。

# **五、实验结果与分析**

## **5.1 BP神经网络拟合正弦函数**

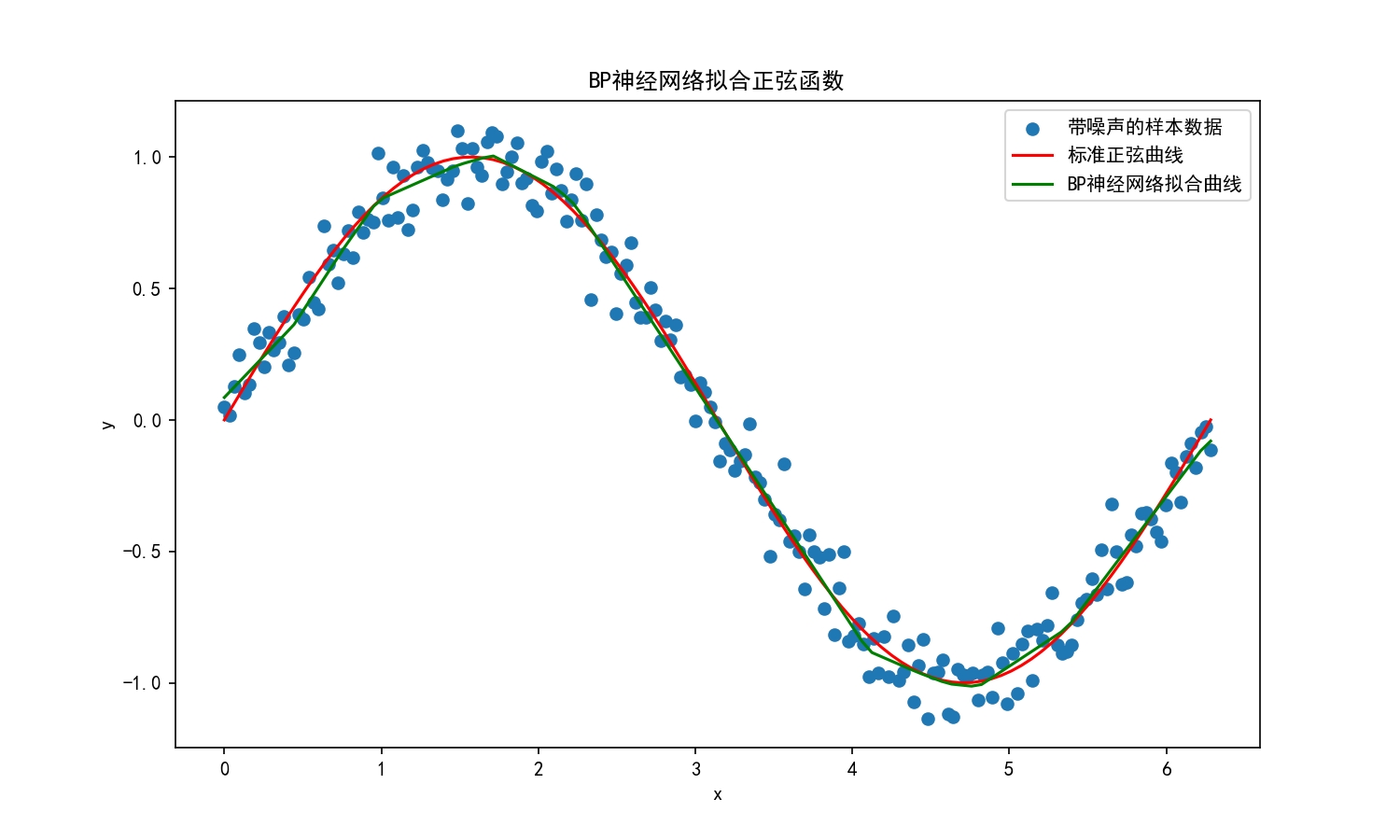
采取了均方误差（MSE）、均方根误差（RMSE）和平均绝对误差（MAE）来评估正弦拟合模型的性能。

误差参数分析表：

| 参数 | 数值 |
| --- | --- |
| MSE | 0.0079 |
| RMSE | 0.0886 |
| MAE | 0.0227 |

其中，MSE是均方误差，RMSE是均方根误差，MAE是平均绝对误差。三个指标的数值越小，说明模型的拟合效果越好。

可视化结果：



**图5：BP神经网络拟合正弦函数**

## **5.2 BP神经网络公路运力预测**

本实验中建立了两个模型，一是客运量模型，而是货运量模型，对两个模型分别进行误差分析和评估。 评估参数采用均方误差（MSE）、均方根误差（RMSE）、平均绝对误差（MAE）和决定系数（R²）。 且提到的误差参数都是经过归一化处理之后的。

### **5.2.1 客运量模型**

误差参数分析表：

| 参数 | 数值 |
| --- | --- |
| MSE | 0.0037 |
| RMSE | 0.0609 |
| MAE | 0.0438 |
| R² | 0.9886 |

### **5.2.2 货运量模型**

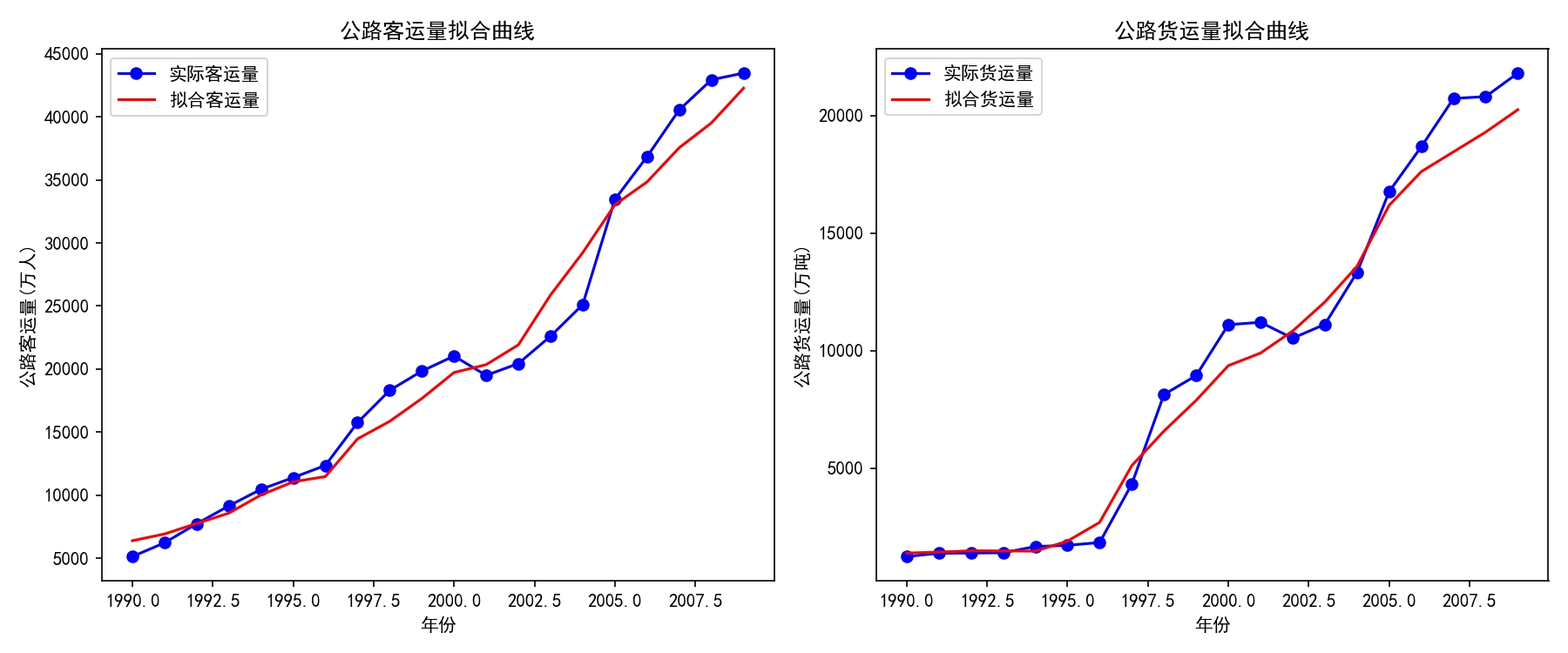
误差参数分析表：

| 参数 | 数值 |
| --- | --- |
| MSE | 0.0136 |
| RMSE | 0.1167 |
| MAE | 0.0940 |
| R² | 0.9769 |

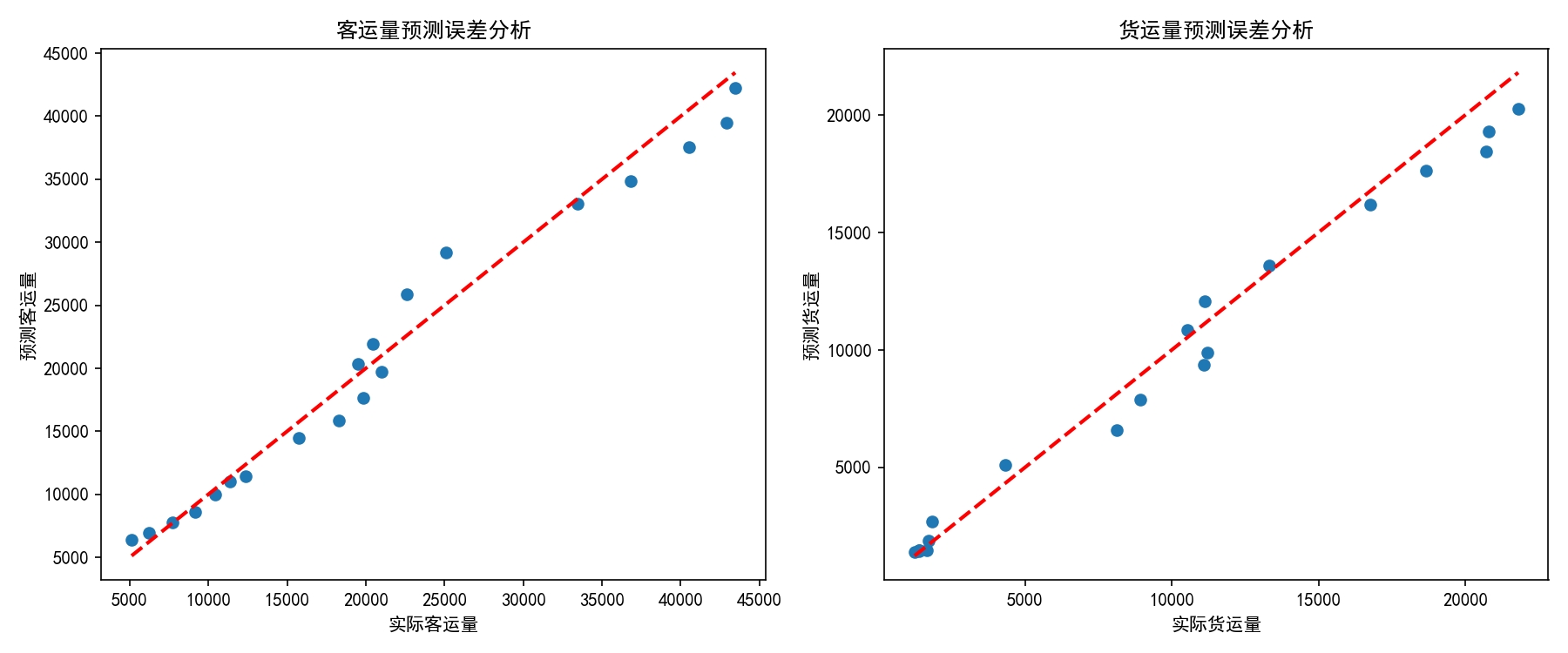
### **5.2.3 预测结果**

| 年份 | 预测公路客运量（万人） | 预测公路货运量（万吨） |
| --- | --- | --- |
| 2010 | 57841.29 | 25722.29 |
| 2011 | 60407.23 | 26845.66 |

### **5.2.4 拟合图和误差分析图示**



**图6：公路运力拟合曲线**



**图7：公路运力数据误差分析**

上述误差分析图是一种散点图，是一个**预测值 vs 实际值图**，其中X轴代表实际值（真实的客运量或货运量），Y轴代表预测值（模型预测的客运量或货运量），每个点代表一个数据样本。如果模型预测完美，所有的点都会落在一条45度的对角线上（图中的红色虚线），这条对角线表示预测值完全等于实际值。

实际情况下，**点越接近这条对角线，说明预测越准确**。位于对角线上方的点表示模型对这些样本的预测值偏高，位于对角线下方的点表示模型对这些样本的预测值偏低。点的分布越集中在对角线附近，说明模型的整体预测效果越好。

# **六、实验程序代码（部分关键代码）**

## **6.1 BP神经网络拟合正弦函数**

# 生成带有白噪声的正弦数据 默认100个点  
def generate\_noisy\_sine\_data(num\_points=200):  
 x = np.linspace(0, 2\*np.pi, num\_points)   
 y = np.sin(x) + np.random.normal(0, 0.1, num\_points)   
 return x, y  
   
# 定义BP神经网络模型  
class BPNeuralNetwork(nn.Module):  
 def \_\_init\_\_(self):  
 super(BPNeuralNetwork, self).\_\_init\_\_()  
 self.hidden1 = nn.Linear(1, 20) # 隐藏层1   
 self.hidden2 = nn.Linear(20, 20) # 隐藏层2

self.hidden3 = nn.Linear(20, 10) # 隐藏层3   
 self.output = nn.Linear(10, 1) # 输出层   
 # 前向传播  
 def forward(self, x):   
 x = torch.relu(self.hidden1(x))   
 x = torch.relu(self.hidden2(x))   
 x = torch.relu(self.hidden3(x))   
 x = self.output(x)   
 return x  
   
# 训练模型 默认设定训练epoch为2000，学习率为0.005  
def train\_model(model, x, y, num\_epochs=2000, learning\_rate=0.005):   
 criterion = nn.MSELoss() # 使用均方误差作为损失函数 MSE  
 optimizer = optim.Adam(model.parameters(), lr=learning\_rate)   
   
 for epoch in range(num\_epochs):  
 inputs = torch.FloatTensor(x).view(-1, 1)   
 targets = torch.FloatTensor(y).view(-1, 1)   
   
 optimizer.zero\_grad()   
 outputs = model(inputs) # 前向传播  
 loss = criterion(outputs, targets) # 计算损失  
 loss.backward() # 反向传播  
 optimizer.step() # 更新参数 这里的更新参数其实就是更新权重和偏置  
 if (epoch + 1) % 100 == 0:  
 print(f'Epoch [{epoch+1}/{num\_epochs}], Loss: {loss.item():.4f}')  
   
# 计算均方误差（MSE）和均方根误差（RMSE）  
def evaluate\_model(model, x, y):  
 model.eval() # 设置模型为评估模式  
 with torch.no\_grad():  
 inputs = torch.FloatTensor(x).view(-1, 1)  
 targets = torch.FloatTensor(y).view(-1, 1)  
 outputs = model(inputs)  
 mse = nn.MSELoss()(outputs, targets).item()  
 rmse = np.sqrt(mse)  
 return mse, rmse  
def calculate\_mean\_error(y\_true, y\_pred):  
 return np.mean(np.abs(y\_true - y\_pred))  
   
# 主函数  
def main():  
 plt.rcParams['font.sans-serif'] = ['SimHei']   
 plt.rcParams['axes.unicode\_minus'] = False   
   
 # 生成数据   
 x, y = generate\_noisy\_sine\_data()  
   
 # 创建和训练模型  
 model = BPNeuralNetwork()  
 train\_model(model, x, y)  
   
 # 使用训练好的模型进行预测  
 with torch.no\_grad():  
 x\_test = torch.FloatTensor(np.linspace(0, 2\*np.pi, 100)).view(-1, 1)  
 y\_pred = model(x\_test).numpy().flatten()  
   
 # 评估模型  
 mse, rmse = evaluate\_model(model, x, y)  
 print(f'MSE: {mse:.4f}, RMSE: {rmse:.4f}')  
 y\_true = np.sin(np.linspace(0, 2\*np.pi, 100))  
 mean\_error = calculate\_mean\_error(y\_true, y\_pred)  
 print(f'平均误差: {mean\_error:.4f}')  
   
 # 绘制结果   
 ·····

## **6.2 BP神经网络公路运力预测**

# 定义模型  
class Net(nn.Module):  
 def \_\_init\_\_(self):  
 super(Net, self).\_\_init\_\_()  
 self.fc1 = nn.Linear(3, 16) # 隐藏层1 输入3 输出16  
 self.fc2 = nn.Linear(16, 8) # 隐藏层2 输入16 输出8  
 self.fc3 = nn.Linear(8, 1) # 输出层 输入8 输出1  
 self.relu = nn.ReLU() # 激活函数  
 self.dropout = nn.Dropout(0.1) # 添加Dropout层  
   
 def forward(self, x):  
 x = self.relu(self.fc1(x))  
 x = self.dropout(x)  
 x = self.relu(self.fc2(x))  
 x = self.fc3(x)  
 return x  
   
# 定义评估函数  
def evaluate\_model(y\_true, y\_pred):  
 mae = mean\_absolute\_error(y\_true, y\_pred)  
 mse = mean\_squared\_error(y\_true, y\_pred)  
 rmse = np.sqrt(mse)  
 r2 = r2\_score(y\_true, y\_pred)  
   
 # 计算归一化误差  
 y\_mean = np.mean(y\_true)  
 normalized\_mae = mae / y\_mean \* 100 # 转换为百分比  
 normalized\_mse = mse / (y\_mean \*\* 2) \* 100 # 转换为百分比  
 normalized\_rmse = rmse / y\_mean \* 100 # 转换为百分比  
 return mae, mse, rmse, r2, normalized\_mae, normalized\_mse, normalized\_rmse  
   
data = pd.read\_csv('highway\_transport\_data.csv', encoding='gbk')  
   
data = data.dropna() # 处理缺失值  
   
# 准备输入特征和目标变量  
X = data[['人数(万人)', '机动车数(万辆)', '公路面积(万平方公里)']].values  
y\_passengers = data['公路客运量(万人)'].values  
y\_freight = data['公路货运量(万吨)'].values  
   
# 数据归一化  
scaler\_X = MinMaxScaler()  
scaler\_y\_passengers = MinMaxScaler()  
scaler\_y\_freight = MinMaxScaler()  
X\_scaled = scaler\_X.fit\_transform(X)  
y\_passengers\_scaled = scaler\_y\_passengers.fit\_transform(y\_passengers.reshape(-1, 1))  
y\_freight\_scaled = scaler\_y\_freight.fit\_transform(y\_freight.reshape(-1, 1))  
   
X\_tensor = torch.FloatTensor(X\_scaled)  
y\_passengers\_tensor = torch.FloatTensor(y\_passengers\_scaled)  
y\_freight\_tensor = torch.FloatTensor(y\_freight\_scaled)  
   
model\_passengers = Net()  
model\_freight = Net()  
   
# 定义损失函数和优化器  
criterion = nn.MSELoss()  
optimizer\_passengers = optim.Adam(model\_passengers.parameters(), lr=0.001, weight\_decay=1e-5)  
optimizer\_freight = optim.Adam(model\_freight.parameters(), lr=0.001, weight\_decay=1e-5)  
   
# 训练模型  
num\_epochs = 2000  
for epoch in range(num\_epochs):  
 # 客运量模型  
 outputs\_passengers = model\_passengers(X\_tensor)  
 loss\_passengers = criterion(outputs\_passengers, y\_passengers\_tensor)  
 optimizer\_passengers.zero\_grad()  
 loss\_passengers.backward()  
 optimizer\_passengers.step()  
   
 # 货运量模型  
 outputs\_freight = model\_freight(X\_tensor)  
 loss\_freight = criterion(outputs\_freight, y\_freight\_tensor)  
 optimizer\_freight.zero\_grad()  
 loss\_freight.backward()  
 optimizer\_freight.step()  
   
 if (epoch + 1) % 100 == 0:  
 print(f'Epoch [{epoch+1}/{num\_epochs}], Loss Passengers: {loss\_passengers.item():.4f}, Loss Freight: {loss\_freight.item():.4f}')  
   
# 在训练数据上进行预测  
model\_passengers.eval()  
model\_freight.eval()  
with torch.no\_grad():  
 y\_passengers\_pred = model\_passengers(X\_tensor).numpy()  
 y\_freight\_pred = model\_freight(X\_tensor).numpy()  
   
y\_passengers\_pred = scaler\_y\_passengers.inverse\_transform(y\_passengers\_pred)  
y\_freight\_pred = scaler\_y\_freight.inverse\_transform(y\_freight\_pred)  
   
# 评估模型  
mae\_passengers, mse\_passengers, rmse\_passengers, r2\_passengers, norm\_mae\_passengers, norm\_mse\_passengers, norm\_rmse\_passengers = evaluate\_model(y\_passengers, y\_passengers\_pred)  
mae\_freight, mse\_freight, rmse\_freight, r2\_freight, norm\_mae\_freight, norm\_mse\_freight, norm\_rmse\_freight = evaluate\_model(y\_freight, y\_freight\_pred)  
   
print("\n客运量模型评估:")  
print(f"平均绝对误差 (MAE): {mae\_passengers:.2f} 万人 (归一化: {norm\_mae\_passengers:.2f}%)")  
print(f"均方误差 (MSE): {mse\_passengers:.2f} (归一化: {norm\_mse\_passengers:.2f}%)")  
print(f"均方根误差 (RMSE): {rmse\_passengers:.2f} 万人 (归一化: {norm\_rmse\_passengers:.2f}%)")  
print(f"决定系数 (R²): {r2\_passengers:.4f}")  
   
print("\n货运量模型评估:")  
print(f"平均绝对误差 (MAE): {mae\_freight:.2f} 万吨 (归一化: {norm\_mae\_freight:.2f}%)")  
print(f"均方误差 (MSE): {mse\_freight:.2f} (归一化: {norm\_mse\_freight:.2f}%)")  
print(f"均方根误差 (RMSE): {rmse\_freight:.2f} 万吨 (归一化: {norm\_rmse\_freight:.2f}%)")  
print(f"决定系数 (R²): {r2\_freight:.4f}")  
   
# 拟合曲线和误差分析可视化

**········**

# 预测2010年和2011年的客运量和货运量  
prediction\_data = np.array([[73.39, 3.9, 0.98], [75.55, 4.1, 1.02]])  
prediction\_data\_scaled = scaler\_X.transform(prediction\_data)  
prediction\_tensor = torch.FloatTensor(prediction\_data\_scaled)  
   
with torch.no\_grad():  
 predicted\_passengers = scaler\_y\_passengers.inverse\_transform(model\_passengers(prediction\_tensor).numpy())  
 predicted\_freight = scaler\_y\_freight.inverse\_transform(model\_freight(prediction\_tensor).numpy())  
   
print("预测结果：")  
print("2010年预测公路客运量：{:.2f} 万人".format(predicted\_passengers[0][0]))  
print("2010年预测公路货运量：{:.2f} 万吨".format(predicted\_freight[0][0]))  
print("2011年预测公路客运量：{:.2f} 万人".format(predicted\_passengers[1][0]))  
print("2011年预测公路货运量：{:.2f} 万吨".format(predicted\_freight[1][0]))