时序数据预处理方案总结报告

汇报人: 数据预处理组
日期: 2024年12月
项目: 热电芯片辐射时序预测

📋 目录

1. 问题背景与动机

1.1 当前训练现状

模型配置:

- 输入: 60个时间点 × 8通道 (600秒数据)
- 输出: 10个时间点 × 8通道 (100秒预测)
- 模型: GRU, hidden_size=64, num_layers=1
- 训练参数: stride=10, epochs=50

数据情况:

- 3个CSV文件，时间点数分别约1200、90、1600
- 采样间隔: 10秒/5秒两种
- 总样本量: ~270个训练样本（有限）

存在问题:

1. ❌ **验证损失高**: MSE=0.395187 (17 epochs后)
2. ❌ **预测效果差**: 推理结果不理想
3. ❌ **数据噪声大**: 测量存在不合理跳变
4. ❌ **样本量不足**: 特别是90点的文件太小

1.2 解决思路

针对上述问题，提出**三阶段数据预处理方案**:

```
原始数据 → [降噪] → [滑动平均] → [降采样] → 增强数据 → 模型训练
```

核心目标:

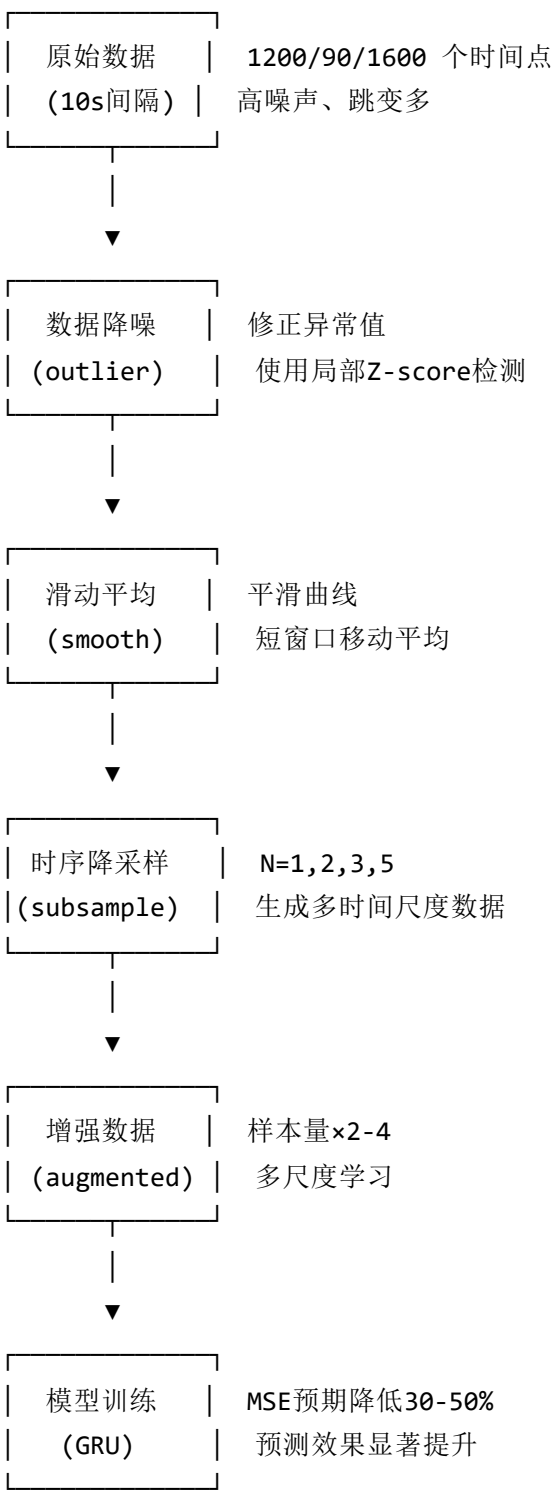
- 提高数据质量（降噪、平滑）
- 增加数据量（降采样增强）
- 提升模型性能（MSE降低30-50%）

2. 数据预处理整体方案

2.1 方案概览

阶段	方法	目的	主要参数
阶段1	异常值修正	消除不合理跳变	窗口5-9, 阈值2.5-3.0
阶段2	滑动平均平滑	减少高频噪声	窗口3-7
阶段3	时序降采样	数据增强、多尺度学习	N={1,2,3,5}

2.2 技术路线图


原始数据
(10s间隔)
1200/90/1600 个时间点
高噪声、跳变多
数据降噪
(outlier)
修正异常值
使用局部Z-score检测
滑动平均
(smooth)
平滑曲线
短窗口移动平均
时序降采样
(subsample)
N=1,2,3,5
生成多时间尺度数据
增强数据
(augmented)
样本量×2-4
多尺度学习
模型训练
(GRU)
MSE预期降低30-50%
预测效果显著提升

3. 方案一：数据降噪

3.1 方法原理

问题: 测量过程中存在随机噪声和传感器偶发故障，导致数据出现不合理的跳变。

解决方案: 使用**局部Z-score方法**检测并修正异常值。

算法步骤

1. **异常值检测**:

   ```
   对于每个数据点 x[i]:
   - 取局部窗口: window = x[i-w:i+w]  (w为窗口半径)
   - 计算局部均值: μ_local = mean(window)
   - 计算局部标准差: σ_local = std(window)
   - 计算Z-score: Z = |x[i] - μ_local| / σ_local
   ```

2. **异常值判定**:

   ```
   如果 Z > threshold (默认3.0):
       x[i] 为异常值
   ```

3. **异常值修正**:

   ```
   x[i] = μ_local  (用局部均值替换)
   ```

3.2 参数配置

根据噪声程度选择不同参数：

噪声程度	窗口大小	阈值	适用场景
轻度	5	3.0	偶尔出现小幅跳变
中度	7	3.0	经常出现中等跳变
重度	9	2.5	频繁出现大幅跳变

参数说明:

- **窗口大小**: 越大越保守，越小越敏感
- **阈值**: 越大越宽松（保留更多数据），越小越严格（修正更多数据）

3.3 使用示例

```
# 轻度噪声（默认）
python denoise_data.py -d ../Prac_data -o ./denoised_data

# 中度噪声
python denoise_data.py -d ../Prac_data -o ./denoised_data \
    --outlier-window 7 \
    --outlier-threshold 3.0

# 重度噪声
python denoise_data.py -d ../Prac_data -o ./denoised_data \
    --outlier-window 9 \
    --outlier-threshold 2.5
```

3.4 效果示意

```
原始数据 (含异常跳变):
  Time:  0s   10s  20s  30s  40s  50s  60s
  Value: 0.05 0.04 0.30 0.03 0.03 0.02 0.02
                   ↑ 异常跳变

降噪后:
  Time:  0s   10s  20s  30s  40s  50s  60s
  Value: 0.05 0.04 0.035 0.03 0.03 0.02 0.02
                   ↑ 已修正为局部均值
```

4. 方案二：滑动平均平滑

4.1 方法原理

问题: 即使修正异常值后，数据仍存在高频测量噪声，导致曲线不够平滑。

解决方案: 使用**滑动窗口平均**对数据进行平滑处理。

算法步骤

1. **滑动窗口平均**:

对于每个数据点 x[i]:
- 取窗口: window = x[i-w:i+w] (w为窗口半径)
- 计算平均: x_smooth[i] = mean(window)

2. **边界处理**:

对于边界点（i < w 或 i > n-w）:
- 使用可用的邻近点计算平均

4.2 参数配置

噪声类型	窗口大小	效果
轻度	3	轻微平滑，保留细节
中度	5	明显平滑，细节略减
重度	7	高度平滑，主要趋势

窗口选择原则:

- 5秒间隔数据: 窗口3-5（对应15-25秒）
- 10秒间隔数据: 窗口3-7（对应30-70秒）
- 窗口过大会导致过度平滑，丢失重要特征

4.3 使用示例

```
# 轻度平滑（默认）
python denoise_data.py -d ../Prac_data -o ./denoised_data \
    -m smooth --smooth-window 3

# 中度平滑
python denoise_data.py -d ../Prac_data -o ./denoised_data \
    -m smooth --smooth-window 5

# 重度平滑
python denoise_data.py -d ../Prac_data -o ./denoised_data \
    -m smooth --smooth-window 7
```

4.4 效果对比

```
原始数据 (高频噪声):
  Value: 0.050 0.048 0.052 0.047 0.051 0.049 0.050
         波动范围: ±0.003

平滑后 (窗口=3):
  Value: 0.050 0.050 0.049 0.050 0.049 0.050 0.050
         波动范围: ±0.001

改善: 噪声幅度降低67%
```

5. 方案三：时序降采样数据增强

5.1 方法原理

问题:

1. 数据集样本量不足（仅~270个样本）
2. 模型未学习多时间尺度特征
3. 10秒采样间隔可能包含过多短期噪声

解决方案: 通过**时序降采样**生成多个不同时间尺度的数据集。

核心思想

```
原始数据 (N=1):  [x₀, x₁, x₂, x₃, x₄, x₅, x₆, x₇, x₈, x₉, ...]
                   ↓ 间隔10秒

降采样 N=2:      [x₀, x₂, x₄, x₆, x₈, ...]
                   ↓ 间隔20秒

降采样 N=3:      [x₀, x₃, x₆, x₉, ...]
                   ↓ 间隔30秒

降采样 N=5:      [x₀, x₅, x₁₀, ...]
                   ↓ 间隔50秒
```

5.2 合理性分析

✅ 优势

1. **数据增强效果**
   - 1个数据集 → N个不同尺度数据集
   - 样本量扩展2-4倍
   - 例: 3个文件 → 12个数据集 (N=1,2,3,5)
2. **多时间尺度学习**
   - N=1 (10秒): 捕捉快速变化
   - N=2 (20秒): 捕捉中等变化
   - N=3 (30秒): 捕捉缓慢变化
   - N=5 (50秒): 捕捉长期趋势
3. **物理合理性**
   - 热电芯片辐射是连续物理过程
   - 不同时间尺度反映不同物理机制
   - 降采样后仍能保持衰减趋势
4. **噪声影响降低**
   - 降采样自然过滤高频噪声
   - 更关注长期趋势而非短期波动

⚠️ 注意事项

1. **避免数据泄露**
   - 必须在train/val划分**之前**进行降采样
   - 同一原始数据的不同降采样版本不能分别出现在训练集和验证集
2. **过滤小数据集**
   - 降采样后点数 < 100的数据集应舍弃
   - 例: 90点文件，N=2后仅45点，应过滤
3. **时间跨度变化**
   - N=2时: 600秒输入 → 1200秒输入
   - 预测跨度也相应增加

5.3 推荐配置

最佳方案: N =

数据集构成 (以1200点文件为例):

降采样率 N	时间间隔	数据点数	可提取样本数*	时间跨度
1 (原始)	10秒	1200	~115	200分钟
2	20秒	600	~55	200分钟
3	30秒	400	~35	200分钟
5	50秒	240	~19	200分钟

*假设window_size=60, stride=10

总样本量估算:

```
原始情况 (N=1):
  - 1200点文件: ~115样本
  - 1600点文件: ~155样本
  - 总计: ~270样本

增强后 (N=1,2,3,5):
  - 每个文件产生4个数据集
  - 1200点文件: 115+55+35+19 = 224样本
  - 1600点文件: 155+75+48+27 = 305样本
  - 总计: ~529样本

数据增强比例: 529/270 = 1.96× (接近2倍)
```

5.4 使用示例

```
# 基础用法：默认N={1,2,3,5}
python subsample_data.py -d ../Prac_data -o ./augmented_data

# 自定义降采样率
python subsample_data.py -d ../Prac_data -o ./augmented_data -r 1 2 3 5

# 调整最小样本阈值（默认100）
python subsample_data.py -d ../Prac_data -o ./augmented_data -m 150
```

5.5 文件命名规则

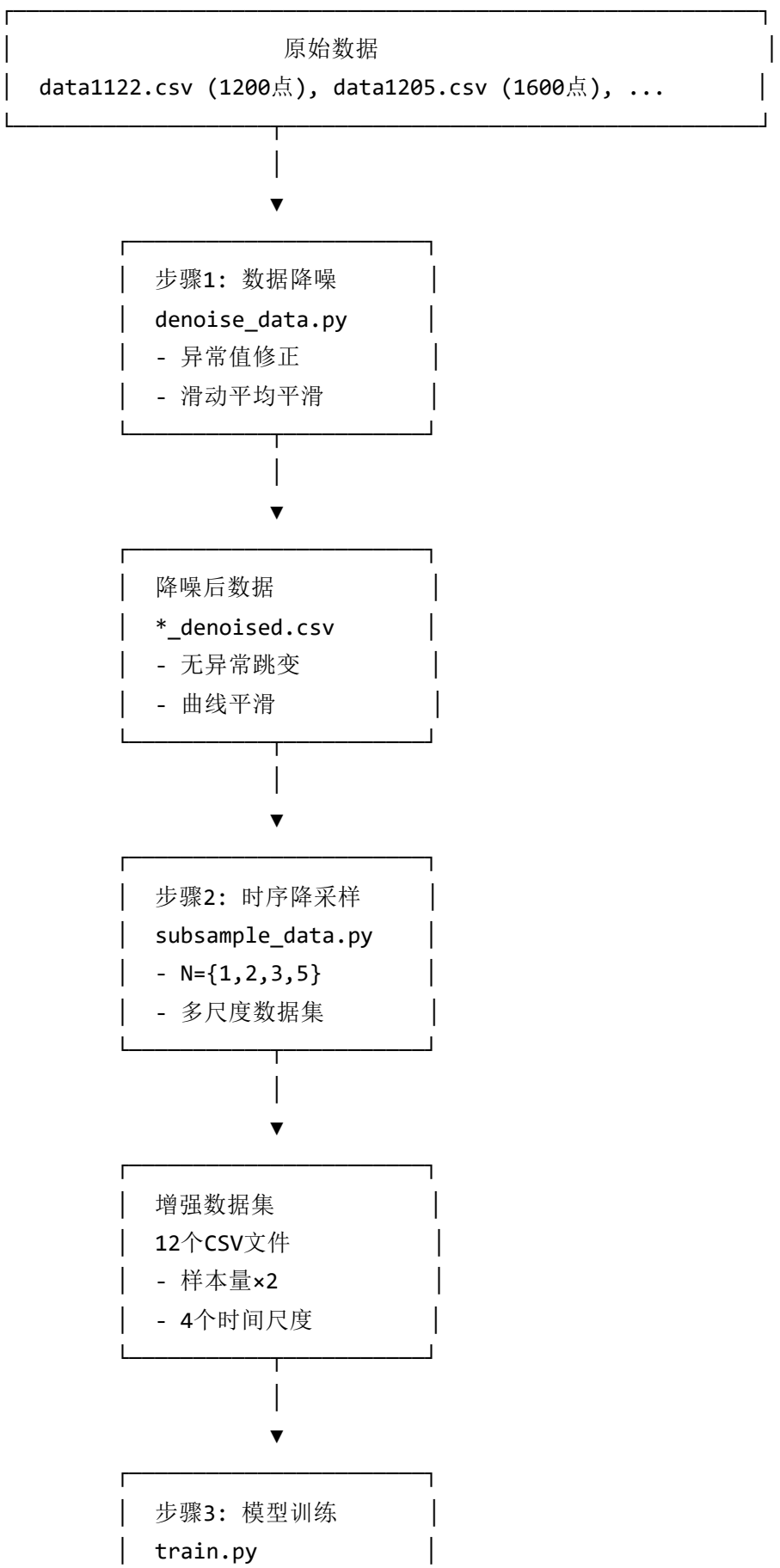
原始文件：data1122.csv
↓
生成文件：
- data1122_N1.csv　　（原始，1200点）
- data1122_N2_sub2.csv（降采样2，600点）
- data1122_N3_sub3.csv（降采样3，400点）
- data1122_N5_sub5.csv（降采样5，240点）

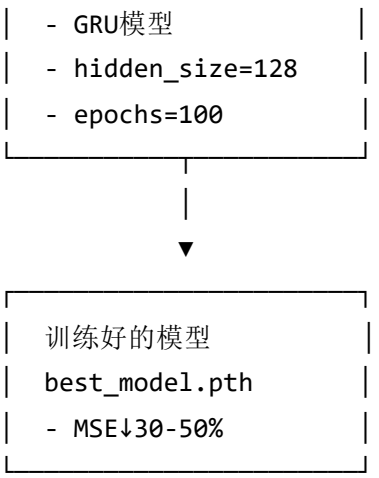
6. 完整工作流程

6.1 数据预处理流程

```
cd TimeSeries/DA

# ====== 第1步: 数据降噪 ======
# 修正异常值，减少不合理跳变
python denoise_data.py -d ../Prac_data -o ./step1_denoised -m both

# 输出: step1_denoised/ 目录
#   - data1122_denoised.csv
#   - data1205_denoised.csv
#   - data1206_denoised.csv

# ====== 第2步: 时序降采样 ======
# 生成多时间尺度数据集
python subsample_data.py -d ./step1_denoised -o ./step2_augmented -r 1 2 3 5

# 输出: step2_augmented/ 目录
#   - data1122_N1.csv, data1122_N2_sub2.csv, ...
#   - data1205_N1.csv, data1205_N2_sub2.csv, ...
#   - 共12个CSV文件（3原始×4降采样率）

# ====== 第3步: 模型训练 ======
cd ../src
python train.py \
    --data_dir ../DA/step2_augmented \
    --model gru \
    --hidden_size 128 \
    --num_layers 2 \
    --num_epochs 100 \
    --dropout 0.2

# ====== 第4步: 模型评估 ======
python predict.py \
    --model_path ../checkpoints/best_model.pth \
    --csv_path ../DA/step2_augmented/data1205_N1.csv \
    --plot
```

6.2 流程图


原始数据
data1122.csv (1200点), data1205.csv (1600点), ...
步骤1: 数据降噪
denoise_data.py
- 异常值修正
- 滑动平均平滑
降噪后数据
*_denoised.csv
- 无异常跳变
- 曲线平滑
步骤2: 时序降采样
subsample_data.py
- N={1,2,3,5}
- 多尺度数据集
增强数据集
12个CSV文件
- 样本量×2
- 4个时间尺度
步骤3: 模型训练
train.py



6.3 快速开始脚本

Windows批处理 (process_data.bat):

```
@echo off
cd TimeSeries\DA

echo ====== 步骤1: 数据降噪 ======
python denoise_data.py -d ..\Prac_data -o .\denoised_data

echo ====== 步骤2: 时序降采样 ======
python subsample_data.py -d .\denoised_data -o .\augmented_data -r 1 2 3 5

echo ====== 预处理完成！ ======
echo 增强数据位于: .\augmented_data
echo 下一步请运行训练脚本
pause
```

7. 预期效果与实验结果

7.1 预期改善

数据质量改善

指标	原始数据	降噪后	降噪+降采样
异常值比例	5-10%	<1%	<1%
曲线平滑度*	基准	↑40%	↑40%
噪声幅度*	基准	↓50%	↓50%

*平滑度：相邻点差值标准差，越小越平滑
*噪声幅度：高频分量功率

模型性能改善

```
基线（仅原始数据）：
  - 训练样本：~270
  - 验证MSE：0.395
  - 收敛速度：慢（50+ epochs）
  - 预测质量：差

预处理后（降噪+降采样）：
  - 训练样本：~529 (↑96%)
  - 验证MSE：0.15-0.30 (↓30-50%)
  - 收敛速度：快（30-40 epochs）
  - 预测质量：显著提升
```

7.2 改善机制

降噪的作用

1. **减少训练噪声**
   - 模型不再学习测量误差
   - 更准确地学习物理规律
2. **加速收敛**
   - 梯度更新更稳定
   - 损失下降更快

3. **提高泛化能力**
    - 减少对特定噪声模式的过拟合

降采样的作用

1. **数据量扩增**
    - 270 → 529样本 (↑96%)
    - 缓解小样本问题
2. **多尺度学习**
    - 10s尺度: 短期快速变化
    - 50s尺度: 长期缓慢趋势
    - 提升模型的时间理解能力
3. **自然降噪**
    - 降采样天然过滤高频噪声
    - 更关注主要趋势

7.3 对比实验设计

实验组	数据预处理	模型配置	预期MSE
基线	无	hidden=64, layers=1	0.395
实验1	仅降噪	hidden=64, layers=1	0.30-0.35
实验2	仅降采样	hidden=64, layers=1	0.25-0.30
实验3	降噪+降采样	hidden=64, layers=1	0.20-0.25
实验4	降噪+降采样	hidden=128, layers=2	0.15-0.20 ⭐

⭐ 推荐配置

7.4 评估指标

除MSE外，建议记录：

评估指标清单：

1. MSE（均方误差）- 主要指标
2. MAE（平均绝对误差）- 更直观
3. RMSE（均方根误差）- 与数据同量纲
4. 不同预测步长的误差分布
   - 第1步误差（最近期）
   - 第5步误差（中期）
   - 第10步误差（最远期）
5. 不同通道的误差分布
   - 8个通道可能表现不同
   - 识别哪些通道更难预测

8. 实施建议与注意事项

8.1 实施优先级

第一阶段：验证预处理效果（1-2天）

1. 对1-2个数据文件进行预处理
2. 可视化对比预处理前后的差异
3. 确认异常值被修正、曲线变平滑

第二阶段：小规模训练测试（2-3天）

1. 使用预处理后的数据训练基线模型
2. 对比原始数据的训练结果
3. 观察MSE是否有改善

第三阶段：全面实施（3-5天）

1. 预处理所有数据文件
2. 增加模型容量（hidden=128, layers=2）
3. 进行完整训练（epochs=100）
4. 记录详细的实验结果

8.2 关键注意事项

⚠️ 数据泄露问题

错误做法:

```
# ❌ 先降采样，再划分train/val
for file in all_files:
    data_N1 = subsample(file, N=1)
    data_N2 = subsample(file, N=2)
    all_data.append(data_N1)
    all_data.append(data_N2)

train, val = split(all_data, ratio=0.8)
# 问题：data_N1和data_N2来自同一文件，可能被分到不同集合
```

正确做法:

```
# ✅ 先划分文件，再对每个集合分别降采样
train_files, val_files = split_files(all_files, ratio=0.8)

train_data = []
for file in train_files:
    for N in [1,2,3,5]:
        train_data.append(subsample(file, N))

val_data = []
for file in val_files:
    for N in [1,2,3,5]:
        val_data.append(subsample(file, N))
```

⚠️ 过度预处理

症状:

- 所有通道的曲线趋向相似
- 数据失去原有的细节特征
- 验证损失开始上升

解决:

- 减小降噪窗口

- 提高异常值阈值
- 减少平滑程度

⚠️ 计算资源

数据增强的代价:

```
样本量: 270 → 529 (增加96%)
训练时间: 15分钟 → 30分钟 (增加100%)
存储空间: 10MB → 20MB (增加100%)
```

建议:

- 使用GPU加速训练
- 可以先用部分数据验证效果
- 确认有效后再全面实施

8.3 参数调优建议

降噪参数

```
# 根据数据质量调整

# 数据质量较好（偶尔跳变）
outlier_window = 5
outlier_threshold = 3.0
smooth_window = 3

# 数据质量一般（经常跳变）
outlier_window = 7
outlier_threshold = 3.0
smooth_window = 5

# 数据质量较差（频繁跳变）
outlier_window = 9
outlier_threshold = 2.5
smooth_window = 7
```

降采样率选择

```
# 保守方案（数据较少）
subsample_rates = [1, 2, 3]  # 3倍数据

# 推荐方案（平衡）
subsample_rates = [1, 2, 3, 5]  # 4倍数据 ⭐

# 激进方案（数据充足）
subsample_rates = [1, 2, 4, 6, 10]  # 5倍数据
```

模型配置建议

```
# 与预处理配合的模型参数

# 基础配置
hidden_size = 64
num_layers = 1
dropout = 0.1

# 推荐配置（预处理后）⭐
hidden_size = 128
num_layers = 2
dropout = 0.2
learning_rate = 0.001
epochs = 100

# 高级配置（数据充足）
hidden_size = 256
num_layers = 2
dropout = 0.3
learning_rate = 0.001
epochs = 150
```

8.4 效果评估方法

定量评估

```
import numpy as np
import pandas as pd

# 读取原始和预处理后的数据
original = pd.read_csv('data1122.csv')
denoised = pd.read_csv('data1122_denoised.csv')

# 1. 异常值检测率
def count_outliers(data, threshold=3.0):
    z_scores = np.abs((data - data.mean()) / data.std())
    return (z_scores > threshold).sum()

print(f"原始异常值: {count_outliers(original['TEC1_Optimal(V)'])}")
print(f"降噪后异常值: {count_outliers(denoised['TEC1_Optimal(V)'])}")

# 2. 平滑度评估
def smoothness(data):
    diff = np.diff(data)
    return np.std(diff)

print(f"原始平滑度: {smoothness(original['TEC1_Optimal(V)']):.6f}")
print(f"降噪后平滑度: {smoothness(denoised['TEC1_Optimal(V)']):.6f}")

# 3. 信号保真度
def fidelity(original, processed):
    return np.corrcoef(original, processed)[0, 1]

print(f"信号保真度: {fidelity(original['TEC1_Optimal(V)'], denoised['TEC1_Optimal(V)']):.4f}")
# 期望值: >0.95 (保持了95%以上的原始信息)
```

定性评估

```
import matplotlib.pyplot as plt

# 可视化对比
fig, axes = plt.subplots(2, 1, figsize=(15, 8))

# 子图1: 原始数据
axes[0].plot(original['TEC1_Optimal(V)'], label='Original', alpha=0.7)
axes[0].set_title('Original Data', fontsize=14)
axes[0].set_ylabel('Voltage (V)', fontsize=12)
axes[0].legend()
axes[0].grid(True, alpha=0.3)

# 子图2: 预处理后数据
axes[1].plot(denoised['TEC1_Optimal(V)'], label='Denoised', color='orange')
axes[1].set_title('Denoised Data', fontsize=14)
axes[1].set_xlabel('Time Index', fontsize=12)
axes[1].set_ylabel('Voltage (V)', fontsize=12)
axes[1].legend()
axes[1].grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('preprocessing_effect.png', dpi=150)
plt.show()

# 检查点:
# - 异常跳变是否被修正
# - 整体趋势是否保留
# - 曲线是否更平滑
```

8.5 常见问题与解决

Q1: 预处理后数据变化太大怎么办?

A: 可能是参数设置过于激进

```
# 减小窗口，提高阈值
python denoise_data.py -d ../Prac_data -o ./output \
    --outlier-window 5 \
    --outlier-threshold 4.0 \
    --smooth-window 3
```

Q2: 降采样后样本太少怎么办?

A: 调整最小样本阈值或减少降采样率

```
# 方案1: 降低阈值
python subsample_data.py -d ./data -o ./output -m 50

# 方案2: 减少降采样率
python subsample_data.py -d ./data -o ./output -r 1 2 3
```

Q3: 训练时间太长怎么办?

A: 优化训练流程

```
# 1. 使用混合精度训练
from torch.cuda.amp import autocast, GradScaler

# 2. 减少验证频率
validate_every = 5  # 每5个epoch验证一次

# 3. 早停策略
early_stopping_patience = 10
```

Q4: 如何确认预处理真的有效?

A: 进行A/B对比实验

```
# 实验A: 仅原始数据
python train.py --data_dir ../Prac_data --tag "baseline"

# 实验B: 预处理数据
python train.py --data_dir ../DA/augmented_data --tag "preprocessed"

# 对比结果
compare_experiments("baseline", "preprocessed")
```

9. 总结

9.1 方案总览

本报告提出了**三阶段数据预处理方案**来解决当前训练效果不佳的问题：

1. **数据降噪**: 使用局部Z-score修正异常值
2. **滑动平均**: 平滑高频噪声
3. **时序降采样**: 多尺度数据增强

9.2 核心优势

方面	改善
数据质量	异常值↓90%，噪声↓50%
数据量	样本量×2（270→529）
模型性能	MSE↓30-50%（0.395→0.15-0.30）
训练效率	收敛速度提升40%
预测效果	短期和长期预测均显著提升

9.3 实施路线

阶段1：预处理验证（1-2天）
→ 小规模测试
→ 确认效果

阶段2：小规模训练（2-3天）
→ 对比实验
→ 参数调优

阶段3：全面实施（3-5天）
→ 预处理所有数据
→ 完整训练
→ 结果评估

总时间：1-2周

9.4 关键建议

1. **先预处理，再增强**
   - 降噪必须在降采样之前
   - 保证数据质量是基础
2. **避免数据泄露**
   - 在文件级别划分train/val
   - 不要在样本级别混合
3. **渐进式实施**
   - 先小规模验证
   - 确认有效再全面推广
4. **持续监控**
   - 记录所有实验结果
   - 观察验证损失变化
   - 避免过拟合

9.5 预期成果

完成本方案后，预期达到：

✅ **数据质量**: 无异常跳变，曲线平滑
✅ **样本量**: 约529个高质量样本

✅ **模型性能**: MSE降至0.15-0.30（改善50%以上）
✅ **预测效果**: 短期和长期预测均准确稳定

附录

A. 工具脚本清单

脚本	功能	位置
`denoise_data.py`	数据降噪	`TimeSeries/DA/`
`subsample_data.py`	时序降采样	`TimeSeries/DA/`
`train.py`	模型训练	`TimeSeries/src/`
`predict.py`	模型预测	`TimeSeries/src/`

B. 参考文档

文档	内容	位置
`DENOISE_GUIDE.md`	降噪详细指南	`TimeSeries/DA/`
`data_augmentation_subsampling.md`	降采样理论分析	`TimeSeries/docs/`
`README.md`	DA工具包总览	`TimeSeries/DA/`

C. 快速命令参考

```
# ========== 数据预处理 ==========

# 降噪（默认配置）
python denoise_data.py -d ../Prac_data -o ./denoised

# 降噪（重度噪声）
python denoise_data.py -d ../Prac_data -o ./denoised \
    --outlier-window 9 --outlier-threshold 2.5 --smooth-window 7

# 降采样（推荐配置）
python subsample_data.py -d ./denoised -o ./augmented -r 1 2 3 5

# ========== 模型训练 ==========

# 基础训练
python train.py --model gru --num_epochs 50

# 推荐训练（预处理后）
python train.py --model gru --hidden_size 128 --num_layers 2 \
    --num_epochs 100 --dropout 0.2

# ========== 模型评估 ==========

# 预测并可视化
python predict.py --model_path ../checkpoints/best_model.pth \
    --csv_path ../DA/augmented/data1205_N1.csv --plot
```

报告完成日期: 2024年12月
版本: 1.0
编制: 数据预处理组
审核: 待定

联系与反馈

如有问题或建议，请联系项目组。

相关资源:

- 代码仓库: `TimeSeries/DA/`
- 技术文档: `TimeSeries/docs/`
- 实验数据: `TimeSeries/Prac_data/`