基于门控循环单元（GRU）的海洋浪高时间序列预测模型

3230104001 陈芊屿

1 代码文件结构

```
wave_prediction/
├── checkpoints/                    # 模型检查点存储目录
│   ├── epoch_10.pth                # 第 10 轮训练的模型权重等
│   ├── epoch_20.pth
│   ├── epoch_30.pth
│   ├── ...
│   ├── epoch_100.pth
│   └── best_checkpoint.pth         # 测试集表现最佳的模型副本
│
├── figures/                        # 训练结果可视化图表目录
│   ├── loss_curve.png              # 训练损失曲线
│   ├── prediction_comparison.png # 预测值与真实值曲线
│
├── logs/
│   ├── run_20251225_095241/      # 附 2025 年 12 月 25 日 09:52:41 的训练日志
│   │   ├── train.log
│   └── ...
│
├── backbone.py                     # GRU 神经网络架构定义
├── data_process.py                 # 数据加载与预处理
├── model.py                        # 模型封装类（训练、测试）
├── train.py                        # 主训练脚本
├── utils.py                        # 工具函数集合
│
├── ocean_buoy_data_june_2023.csv  # 原始数据集
│
└── requirements.txt                # Python 依赖包列表
```

具体文件见附件。附件中提供了 2025 年 12 月 25 日 09:52:41 的训练日志，文件中预存的检查点也为该次训练中得到的权重。

注意：如果直接运行 train.py，会覆盖原先的检查点。

2 数据集概况

文件名称：ocean_buoy_data_june_2023.csv
时间范围：2023 年 6 月整月，每小时数据
空间范围：10 个浮标，分布在 20-25°N、120-125°E 海域
数据量：约 7200 行记录(10 浮标×30 天×24 小时)

Timestamp：观测时间（每小时）

buoy_id：浮标编号（B01 - B10）
latitude：纬度（20 - 25°N）
longitude：经度（120 - 125°E）
wind_speed：风速（0 - 25 米/秒）
sea_level_pressure：海平面气压（995 - 1020 百帕）
significant_wave_height：有效浪高（0.2 - 6 米）

3 建模策略

鉴于原始数据属于时间跨度短（仅一个月）、多个不同浮标站点、具备强物理相关性的时序数据，采取以下策略：

3.1 数据选择与混合训练

单个浮标的数据量仅为 720 条，不足以训练深度神经网络，极易导致过拟合。因此，采取混合训练的策略，将 10 个浮标的数据视为独立但同分布的样本进行联合训练，因此总样本量扩充至 7200 条，从而让模型学习通用的“海浪物理机制”而非单一站点的噪声。

3.2 模型架构选择

尽管用于训练的数据量扩充至 7200 条，数据集规模依然比较小。因此，为防止出现过拟合现象，采用参数数量较少的门控循环单元（Gated Recurrent Unit, GRU）作为模型的架构。作者认为，对于 7200 条的数据，采用 1 层的 GRU，既能够降低模型的复杂度，也足以实现理想的预测效果。

4 数据预处理与特征工程

4.1 数据预处理

为了适应深度学习模型的输入要求，进行了以下处理：

首先对数据进行归一化。由于风速、气压和浪高的量纲差异巨大，使用 MinMaxScaler 将所有数据缩放到 [0,1][0,1]区间。这有助于梯度下降算法更快收敛，防止大数值特征主导模型权重。

对于输入数据采用滑动窗口将时间序列转化为监督学习样本。本次实验中选取窗口长度为 12。

4.2 特征工程

对于原始的数据文件，可以直接提取出六个特征：时间、经度、纬度、风速、海平面气压、过去的有效浪高。

而原始数据文件中的时间以小时数为单位，并采取 24 小时制，但数字上的突变会误导模型，因此采取周期编码方式，通过正余弦函数对时间进行编码，解决时间不连续问题。因此时间相关的特征变为二维，总体输入特征维度为七维：时间正弦分量（hour_sin）、时间余弦分量（hour_cos）、归一化经度（longitude）、归一化纬度（latitude）、归一化风速（wind_speed）、归一化海平面气压（sea_level_pressure）、归一化有效浪高（significant_wave_height）。

5 数据集划分与超参数设置

5.1 数据集划分

由于原始数据是时间序列，因此不同时刻数据前后具有强相关性，基于时间顺序切分数据集。实验中训练集与测试集比例设定为 5:1（选取前 25 天作为训练集，后 5 天作为测试集），由于总体数据体量小，暂时不设定验证集。

5.2 超参数设置

超参数	设置值	选择依据
batch_size	64	平衡训练速度与梯度稳定性
hidden_size	64	捕捉该数据量的非线性特征，避免过拟合
window_size	12	观察过去 12 个时间步（12 小时）的数据
num_layers	1	GRU 层数，单层对于本任务足够
learning_rate	0.001	AdamW 优化器的常用初始学习率
weight_decay	0.01	标准权重衰减率，防止过拟合
n_epochs	100	训练总轮次，100 次足以完成训练

6 模型实现与训练流程

本次实验采用 PyTorch 的标准范式，并结合了自定义 Dataset 类进行封装。

5.1 数据处理模块

首先通过 panda 库读取 csv 文件。出于前文所述的原因，对原始数据集进行划分、归一化、并针对时间列进行周期编码实现时间嵌入。为避免数据泄露，在数据归一化过程中使用训练集的最值对全部数据集进行统一的归一化。

构建自定义 Dataset 类。由于现有数据包含十个浮标的数据，全部混合在一起会导致模型学得不真实的数据规律，为确保数据在时间、空间维度上的连续性，需要对数据进行分组后再分别进行滑窗操作。对于每一个滑窗切片，统一堆叠到 feature_list 中，将连续的时间序列数据转化为可以学习的监督数据样本。随后将所有滑窗样本及其对应的 target 转为张量。最终将处理后的数据送入 Dataloader 中。核心数据库类 WaveDataset 定义如下：

```
class WaveDataset(Dataset):
    def __init__(self, df: pd.DataFrame, input_cols: list, target_col: str, window_size=12):
        super().__init__()
        self.window_size = window_size

        features_list = []
        target_list = []

        for buoy_id, group in df.groupby("buoy_id"):
            data = group[input_cols].values
```

```
            target_idx = input_cols.index(target_col)
            for i in range(len(data) - window_size):
                x_window = data[i : i + window_size, :]
                y_label = data[i + window_size, target_idx]

                features_list.append(x_window)
                target_list.append(y_label)

        self.features = torch.from_numpy(np.array(features_list)).float()
        self.target = torch.from_numpy(np.array(target_list).reshape(-1, 1)).float()
    def __len__(self):
        return len(self.features)
    def __getitem__(self, idx):
        return self.features[idx], self.target[idx]
```

5.2 骨干网络架构

考虑到原始数据的体量，模型架构不能过于复杂，复杂的模型看似具备更强的学习能力，但参数量太大，在较小的数据集上反而容易导致过拟合。因此整个模型只使用一层 GRU 连接一层全连接层得到输出（dropout 层可选，但在本实验中默认为 0）。

由于 Pytorch 中 GRU 网络默认输出每一个时间步的预测值，而对于本实验中的任务，只需要序列中最后一个时间步的信息，因此在 forward 函数中需要对输出进行切片。

对于骨干网络，使用 Pytorch 的典型架构。出于篇幅原因，仅展示自定义 GRU 类核心代码如下：

```
class GRU(nn.Module):
    def __init__(
        self,
        input_size,
        hidden_size,
        output_size,
        num_layers=1,
        gru_dropout=0.0,
        fc_dropout=0.0
    ):
        super().__init__()
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.num_layers =num_layers

        self.rnn = nn.GRU(
            input_size=input_size,
            hidden_size=hidden_size,
            num_layers=num_layers,
```

```
            batch_first=True,
            dropout=gru_dropout if num_layers > 1 else 0.0
        )
        self.dropout = nn.Dropout(p=fc_dropout)
        self.fc_out = nn.Linear(
            in_features=hidden_size,
            out_features=output_size
        )
        self.init_params()
```

5.3 模型封装类

在 Model 类中，由于没有设置验证集，因此在训练过程中采用“定时快照+后验选择”的策略：每训练 10 个 epochs 对当前模型的权重进行保存，训练完 100 个 epochs 后，对所存储的十个模型进行评估，防止过拟合；同时，每 25 个 epochs 调用学习率调度器降低学习率。

训练结束后，自动执行_select_best_checkpoint 方法，对于每个权重，在测试集上进行推理，得到归一化的预测值之后进行反归一化，与真实值对比并计算 RMSE 作为权重评估指标，选择 RMSE 最小的一个权重作为最佳模型权重，输出 best_checkpoint.pth。根据 Pytorch 规范，定义核心训练方法如下：

```
    def train_model(
        self,
        dataloader: DataLoader,
        n_epochs=100,
        save_dir='./checkpoints',
        save_interval=10
    ):
        for epoch in range(n_epochs):
            self._train_one_epoch(
                log=self.log_train,
                model=self.model,
                dataloader=dataloader['train_loader'],
                optimizer=self.optimizer,
                criterion=self.criterion
            )
            loss = self.log_train['loss']
            self.train_losses.append(loss)
            self.lr_scheduler.step()

            if (epoch + 1) % save_interval == 0:
                self.save_checkpoint(epoch=epoch, save_dir=save_dir)
                print(f"Checkpoint saved at epoch {epoch+1}, loss: {loss:.6f}")

        self.logger.info("----------Training finished----------\n")
        return
```

```
    def _train_one_epoch(self,
                         log: Dict,
                         model: nn.Module,
                         dataloader: DataLoader,
                         optimizer: Optimizer,
                         criterion: Callable):
        model = model.train()
        losses = []
        for features, targets in tqdm(dataloader):

            features = features.to(self.device)
            targets = targets.to(self.device)

            optimizer.zero_grad()

            outputs = model(features)

            loss: Tensor = criterion(outputs, targets)

            loss.backward()

            optimizer.step()

            losses.append(loss.item())

        loss = np.mean(losses)
        log['loss'] = loss
        return
```

7 结果评估与可视化

同理，最终加载最佳权重并在测试集上进行推理，反归一化后采取多个评估指标并在控制台返回结果，结果包括：MSE，RMSE，MAE，R^2 Score。其中一次实验结果如下：

```
Test Results:
MSE: 0.0016
RMSE: 0.0405
MAE: 0.0297
R2 Score: 0.9689
```

根据指标可知，单层 GRU 已经足以对原始数据进行预测。同时利用 matplotlib 对训练结果进行可视化，得到训练过程中的 loss 曲线和最佳模型在测试集上预测的表现，如图 1, 2 所示。

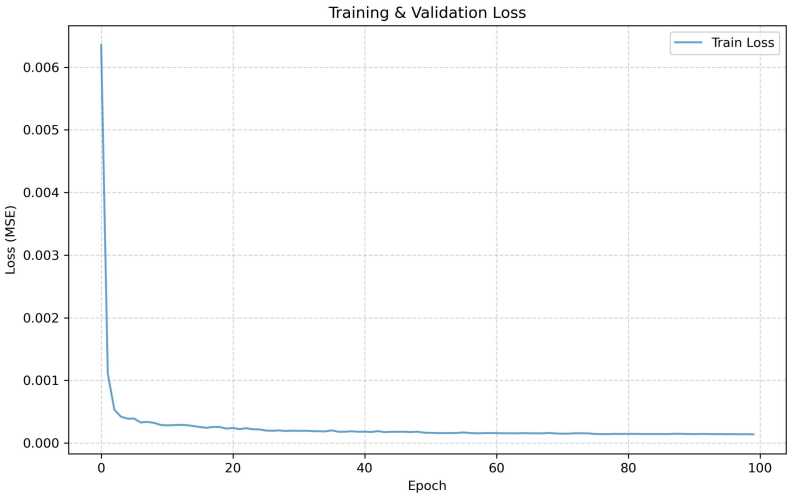

图 1 训练损失曲线

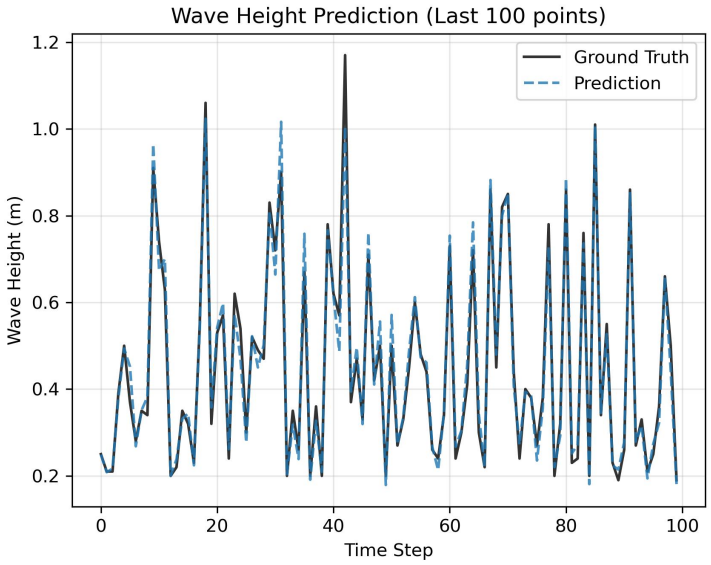

图 2 最佳模型浪高预测(最后 100 个点)

根据可视化结果可知，loss 在训练过程中非常小，同时模型在测试集上同样表现良好，预测值与真实值基本符合，因此本模型能够较好地进行浪高预测。

8 总结

本实验构建了一个完整的端到端浪高 GRU 预测系统，经过实验验证，模型具有良好的泛化能力。最终的测试结果验证了该方案在短期海浪高度预测中的有效性，为海洋环境监测提供了有效的预测方法。