AI VIET NAM – COURSE 2023

Time-series Forecasting Project

(MLP, RNN, LSTM, Bi-LSTM, and XGBoost)

Ngày 8 tháng 12 năm 2023

Giới thiệu về data time-series project cho việc dự đoán nhiệt độ dầu của máy biến áp điện :

Phân phối nguồn điện là quá trình chuyển giao và phân phối năng lượng điện từ các nhà máy điện đến người sử dụng cuối cùng. Đây là một phần quan trọng trong hệ thống điện để đảm bảo cung cấp điện ổn định và đáng tin cậy cho người dân và các doanh nghiệp. Nhưng việc dự đoán nhu cầu tiếp theo của một khu vực cụ thể là khó khăn, vì nó thay đổi theo các ngày trong tuần, ngày lễ, mùa, thời tiết, nhiệt độ, v.v. Tuy nhiên, không có phương pháp hiện tại nào có thể thực hiện dự đoán dài hạn dựa trên dữ liệu thực tế với độ chính xác cao. Bất kỳ kết quả dự đoán sai lầm nào có thể gây hại cho máy biến áp điện. Do đó, hiện tại, không có một phương pháp hiệu quả để dự đoán việc sử dụng điện trong tương lai, do đó các nhà quản lý phải đưa ra dự đoán dựa trên kinh nghiệm, mà khi đó kết qua dự đoán thường cao hơn nhiều so với nhu cầu thực tế. Điều này gây lãng phí điện và giảm giá trị của thiết bị. Thông qua quá trình nghiên cứu, người ta nhận thấy rằng nhiệt độ dầu có thể phản ánh tình trạng của máy biến áp điện. Một trong những chiến lược hiệu quả nhất là dự đoán làm thế nào nhiệt độ dầu của máy biến áp điện là an toàn và tránh lãng phí không cần thiết. Do đó, để giải quyết vấn đề này, công ty Phát triển Khoa học và Công nghệ Guowang Fuda của Bắc Kinh đã xây dựng hệt thống thu thập dữ liệu trong 2 năm về nhiệt độ dầu của máy biến áp điện. Dựa trên bộ dữ liệu Electricity Transformer Dataset (ETDataset dataset), chúng ta có thể phát triển các phương pháp dự đoán nhiệt độ dầu của bộ máy áp điện và khả năng chịu tải tối đa của nó.

Bộ dữ liệu ETDataset (ETTH1, hình 1) bao gồm 7 thông tin như sau: the recorded date, high useful load (HUFL), high useless load (HULL), middle useful load (MUFL), middle useLess load (MULL), low useful load (LUFL), low useLlss load (LULL), và oil temperature (OT).

	date	HUFL	HULL	MUFL	MULL	LUFL	LULL	OT
0	2016-07-01 00:00:00	5.827	2.009	1.599	0.462	4.203	1.340	30.531000
1	2016-07-01 00:15:00	5.760	2.076	1.492	0.426	4.264	1.401	30.459999
2	2016-07-01 00:30:00	5.760	1.942	1.492	0.391	4.234	1.310	30.038000
3	2016-07-01 00:45:00	5.760	1.942	1.492	0.426	4.234	1.310	27.013000
4	2016-07-01 01:00:00	5.693	2.076	1.492	0.426	4.142	1.371	27.787001

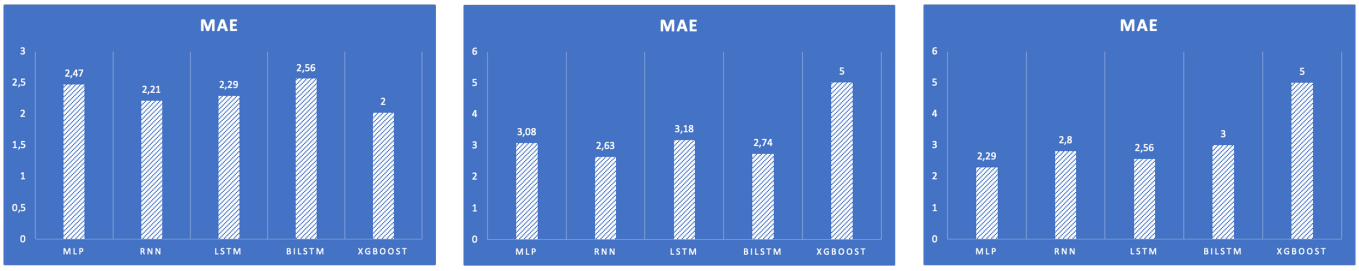
Hình 1: Một vài sample data từ tập dữ liệu ETTH1

Trong project này chúng ta sẽ sử dụng các giải thuật multilayer perceptron(MLP), recurrent neural network (RNN), long-short term memory (LSTM), Bidirectional LSTM (Bi-LSTM) và XGBoost để dữ

đoán nhiệt độ dầu của máy biến áp. ETTh1 dataset được tổ chức thành 3 trường hợp khác nhau để đánh giá để đánh giá độ chính xác của các giải thuật cho bài toán dự đoán time-series data:

- Case study 1 (Multivariate to Multivariate):
  1. Input: HUFL, HULL, MUFL, MULL, LUFL, LULL và OT
  2. Output: HUFL, HULL, MUFL, MULL, LUFL, LULL và OT
- Case study 2 (Multivariate to Univariate):
  1. Input: HUFL, HULL, MUFL, MULL, LUFL, LULL và OT
  2. Output: OT
- Case study 3 (Univariate to Univariate):
  1. Input: OT
  2. Output: OT

Hình 2 thể hiện kết quả của các giải thuật ứng với 3 case study bên trên. Để hoàn thành được project này, AIVN thừa nhận rằng người đọc đã nắm vững và biết cách sử dụng PyTorch để cài đặt các giải thuật **MLP, RNN, LSTM, Bi-LSTM và XGBoost**



Multivariate to Multivariate | Multivariate to Univariate | Univariate to Univariate

Hình 2: Kết quả dự đoán OT ứng với 3 trường hợp case study

Thực hành 1: (Tải dataset và import các thư viện cần thiết)
Hãy cài đặt đoạn chương trình sau để tải dữ liệu ETTh1 về máy tính:

```
!curl -o ETTh1.csv https://raw.githubusercontent.com/zhouhaoyi/ETDataset/main/ETT-small/ETTh1.csv
```

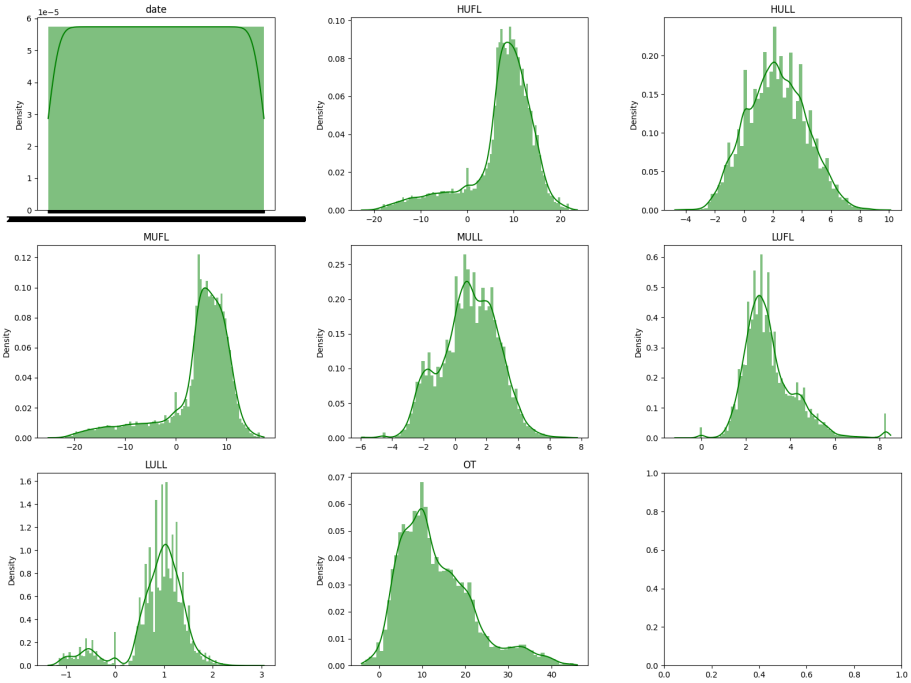
Hãy cài đặt đoạn chương trình sau để import các thư viện cần thiết:

```
import os
import time
import numpy as np
import polars as pl
from torch.utils.data import TensorDataset, DataLoader
import torch
import torch.nn as nn
import torch.nn.functional as F
import seaborn as sns
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from xgboost import XGBRegressor
```

Thực hành 2: (Data Exloration) hãy cài đặt đoạn chương trình sau để phân tích tổng quan dữ liệu của tập dataset ETTh1:

```
train_df = pl.read_csv(file_path)
columns_to_plot = train_df.columns

num_columns = 3
num_rows = int(np.ceil(len(columns_to_plot) / num_columns))
grid_layout = (num_rows, num_columns)

fig, axes = plt.subplots(*grid_layout, figsize=(16, 12))

axes = axes.flatten()

for i, column in enumerate(columns_to_plot):
    sns.histplot(train_df[column], kde=True, ax=axes[i],
                 color='green', stat="density", linewidth=0)
    axes[i].set_title(column)
plt.tight_layout()
plt.show()
```

Hình 3 thể hiện kết quả thực hiện bước data exploration phân tích data distribution of từng feature trong tập dữ liệu ETTh1.



Hình 3: Kết quả phân tích data distribution của từng feature trên tập dữ liệu ETTh1

Thực hành 3: (Data Preparation) Hãy cài đặt lại đoạn chương trình bên dưới để thực hiện bước tổ chức tập dữ liệu ETTh1 cho 3 trường hợp cần so sánh: multivariate to multivariate, multivariate to univariate, và univariate to univariate.

```
class TimeSeriesDataLoader:
    def __init__(self, file_path, input_size, label_size, offset, train_size, val_size
    , date_column=None, target_name=None, features_type='M', batch_size=64):
```

```
        if offset < label_size:
            print(f"Offset will be change from {offset} to {label_size}")
            offset = label_size

        self.input_size = input_size
        self.label_size = label_size
        self.offset = offset
        self.train_size = train_size
        self.val_size = val_size
        self.target_name = target_name
        self.features_type = features_type
        self.batch_size = batch_size

        # Load the data
        self.df = pl.read_csv(file_path)
        if date_column is not None: self.df = self.df.drop('date')

        # Determine in_variable and out_variable based on features_type
        if features_type == 'S':
            self.in_variable = 1
            self.out_variable = 1
        elif features_type == 'M':
            self.in_variable = len(self.df.columns)
            self.out_variable = len(self.df.columns)
        elif features_type == 'MS':
            self.in_variable = len(self.df.columns)
            self.out_variable = 1
        else:
            raise ValueError("Invalid features_type. Choose from 'S' for Univariate-to-Univariate, 'M' for Multivariate-to-Multivariate, 'MS' for Multivariate-to-Univariate.")

        # Preprocess the data
        self.X_train, self.y_train = self.__create_dataset(0, int(train_size * len(self.df)))
        print(f'{self.X_train.shape = }')
        print(f'{self.y_train.shape = }')
        self.X_val, self.y_val = self.__create_dataset(int(train_size * len(self.df)), int((train_size + val_size) * len(self.df)))
        print(f'{self.X_val.shape = }')
        print(f'{self.y_val.shape = }')
        self.X_test, self.y_test = self.__create_dataset(int((train_size + val_size) * len(self.df)), None)
        print(f'{self.X_test.shape = }')
        print(f'{self.y_test.shape = }')

        # Convert to PyTorch DataLoaders
        self.train_loader = self.__create_dataloader(self.X_train, self.y_train)
        self.val_loader = self.__create_dataloader(self.X_val, self.y_val)
        self.test_loader = self.__create_dataloader(self.X_test, self.y_test)

    def __create_dataset(self, start_idx, end_idx):
        if end_idx is None:
            end_idx = len(self.df) - self.label_size - self.offset

        start_idx += self.input_size + self.offset

        features = []
        labels = []

```

```
        for idx in range(start_idx, end_idx):
            feature_start_idx = idx - self.input_size - self.offset
            feature_end_idx = feature_start_idx + self.input_size

            label_start_idx = idx - 1
            label_end_idx = label_start_idx + self.label_size

            if self.features_type == 'S':
                feature = self.df.select(self.target_name)[feature_start_idx:
    feature_end_idx]
                label = self.df.select(self.target_name)[label_start_idx:label_end_idx
    ]
            elif self.features_type == 'M':
                feature = self.df[feature_start_idx:feature_end_idx]
                label = self.df[label_start_idx:label_end_idx]
            elif self.features_type == 'MS':
                feature = self.df[feature_start_idx:feature_end_idx]
                label = self.df.select(self.target_name)[label_start_idx:label_end_idx
    ]
            else:
                raise ValueError("Invalid features_type. Choose from 'S', 'M', 'MS'.")

            features.append(feature.to_numpy())
            labels.append(label.to_numpy())
            self.out_features = label.columns
            self.in_features = label.columns

        return np.array(features), np.array(labels)

    def __create_dataloader(self, X, y):
        X_tensor = torch.tensor(X, dtype=torch.float32)
        y_tensor = torch.tensor(y, dtype=torch.float32)
        dataset = TensorDataset(X_tensor, y_tensor)
        return DataLoader(dataset, batch_size=self.batch_size, shuffle=True)
```

3.1 Tổ chức dữ liệu cho trường hợp multivariate to multivariate:

```

features_type='M'
sub_dir = 'multi2multi'
os.makedirs(os.path.join(weight_dir, sub_dir), exist_ok=True)
multi2multi_loader = TimeSeriesDataLoader(file_path,
                                          input_size=input_size,
                                          label_size=label_size,
                                          offset=offset,
                                          train_size=train_size,
                                          val_size=val_size,
                                          target_name=target_name,
                                          features_type=features_type,
                                          date_column=date_column)
```

Question 1: Hãy cho biết shape của tập X_train và y_train

```
a)
self.X_train.shape = (11762, 336, 7)
self.y_train.shape = (11762, 96, 1)
b)
self.X_train.shape = (11762, 336, 7)
self.y_train.shape = (11762, 96, 7)
c)
self.X_train.shape = (11782, 336, 7)
```

```
self.y_train.shape = (11782, 96, 1)
d)
self.X_train.shape = (11792, 336, 7)
self.y_train.shape = (11792, 96, 1)
```

3.2 Tổ chức dữ liệu cho trường hợp multivariate to univariate:

```

ffeatures_type='MS'
sub_dir = 'multi2uni'
os.makedirs(os.path.join(weight_dir, sub_dir), exist_ok=True)
multi2uni_loader = TimeSeriesDataLoader(file_path,
                                        input_size=input_size,
                                        label_size=label_size,
                                        offset=offset,
                                        train_size=train_size,
                                        val_size=val_size,
                                        target_name=target_name,
                                        features_type=features_type,
                                        date_column=date_column)
```

Question 2: Hãy cho biết shape của tập X_train và y_train

```

a)
self.X_train.shape = (11762, 336, 7)
self.y_train.shape = (11762, 96, 1)
b)
self.X_train.shape = (11772, 336, 7)
self.y_train.shape = (11772, 96, 1)
c)
self.X_train.shape = (11782, 336, 7)
self.y_train.shape = (11782, 96, 1)
d)
self.X_train.shape = (11792, 336, 7)
self.y_train.shape = (11792, 96, 1)
```

3.2 Tổ chức dữ liệu cho trường hợp univariate to univariate:

```
features_type='S'
sub_dir = 'uni2uni'
os.makedirs(os.path.join(weight_dir, sub_dir), exist_ok=True)
uni2uni_loader = TimeSeriesDataLoader(file_path,
                                      input_size=input_size,
                                      label_size=label_size,
                                      offset=offset,
                                      train_size=train_size,
                                      val_size=val_size,
                                      target_name=target_name,
                                      features_type=features_type,
                                      date_column=date_column)
```

Question 3: Hãy cho biết shape của tập X_train và y_train

```

a)
self.X_train.shape = (11762, 336, 7)
self.y_train.shape = (11762, 96, 1)
b)
self.X_train.shape = (11772, 336, 7)
self.y_train.shape = (11772, 96, 1)
c)
self.X_train.shape = (11762, 336, 1)
```

```
self.y_train.shape = (11762, 96, 1)
d)
self.X_train.shape = (11792, 336, 7)
self.y_train.shape = (11792, 96, 1)
```

Thực hành 4: (Training Model Preparation) Hãy cài đặt lại đoạn chương trình bên dưới để định nghĩa các mô hình MLP, RNN, LSTM, và Bi-LSTM.

```python
import torch
import torch.nn as nn

class ModelManager:
    def __init__(self, model, train_loader, val_loader=None, lr=0.001, patience=100):
        self.model = model
        self.train_loader = train_loader
        self.val_loader = val_loader
        self.patience = patience
        self.best_loss = float('inf')
        self.counter = 0
        self.criterion = nn.L1Loss()
        self.optimizer = torch.optim.Adam(model.parameters(), lr=lr)

    def train(self, num_epochs, save_dir='.'):
        os.makedirs(save_dir, exist_ok=True)
        save_path = os.path.join(save_dir, f'best-{self.model.__class__.__name__}.pth')
        for epoch in range(num_epochs):
            start_time = time.time()
            self.model.train()  # Set the model to training mode
            total_train_loss = 0

            for inputs, targets in self.train_loader:
                # Forward pass
                outputs = self.model(inputs)
                loss = self.criterion(outputs, targets)
                total_train_loss += loss.item()

                # Backward pass and optimization
                self.optimizer.zero_grad()
                loss.backward()
                self.optimizer.step()

            avg_train_loss = total_train_loss / len(self.train_loader)

            # Validate the model
            val_loss = self.evaluate(loader=self.val_loader)

            # Check for early stopping
            if self.early_stopping(val_loss, save_path):
                print(f"Early stopping at epoch {epoch + 1}")
                return

            print(f'Epoch [{epoch + 1}/{num_epochs}], '
                  f'time: {int(time.time() - start_time)}s, '
                  f'loss: {avg_train_loss:.4f}, '
                  f'val_loss: {val_loss:.4f}')

        self.load_model(save_path)

    def evaluate(self, loader):
```

```
        self.model.eval()  # Set the model to evaluation mode
        total_loss = 0

        with torch.no_grad():
            for inputs, targets in loader:
                outputs = self.model(inputs)
                loss = self.criterion(outputs, targets)
                total_loss += loss.item()

        avg_loss = total_loss / len(loader)
        return avg_loss

    def early_stopping(self, val_loss, save_path):
        if val_loss < self.best_loss:
            self.best_loss = val_loss
            self.counter = 0
            self.save_model(save_path)
        else:
            self.counter += 1
        return self.counter >= self.patience

    def save_model(self, save_path):
        torch.save(self.model.state_dict(), save_path)
        print(f'Model saved to {save_path}')

    def load_model(self, load_path):
        self.model.load_state_dict(torch.load(load_path))
        print(f'Model loaded from {load_path}')

    def predict(self, input_data):
        self.model.eval()  # Set the model to evaluation mode

        if isinstance(input_data, DataLoader):
            # If input_data is a DataLoader, iterate through batches and concatenate
    predictions
            predictions = []
            with torch.no_grad():
                for inputs, _ in input_data:
                    outputs = self.model(inputs)
                    predictions.append(outputs)
            predictions = torch.cat(predictions, dim=0)
        else:
            # Assume input_data is a single input tensor
            with torch.no_grad():
                predictions = self.model(input_data).unsqueeze(0)

        return predictions

    def plot(self, y, yhat, feature_names=None, save_dir='.', save_plots=True,
    num_elements=None):
        if feature_names is None:
            feature_names = [f'Feature {i + 1}' for i in range(y.shape[2])]

        if num_elements is not None:
            y = y[:num_elements]
            yhat = yhat[:num_elements]

        for feature_index, feature_name in enumerate(feature_names):
            plt.figure(figsize=(10, 5))

```

```
            plt.plot(y[:, :, feature_index].flatten(), label='y', linestyle='-')
            plt.plot(yhat[:, :, feature_index].flatten(), label='y_hat', linestyle='--')

            plt.title(feature_name)
            plt.xlabel('Time Step')
            plt.ylabel('Values')
            plt.legend()

            if save_plots:
                # Create the save directory if it doesn't exist
                os.makedirs(os.path.join(save_dir, self.model.__class__.__name__), exist_ok=True)

                # Save the plot
                save_path = os.path.join(save_dir, self.model.__class__.__name__, f'{feature_name}.png')
                plt.savefig(save_path)

            plt.show()
            plt.close()  # Close the plot to avoid overlapping in saved images

class MachineLearningModelManager(ModelManager):
    def __init__(self, model, xtrain, ytrain, xval, yval):
        self.model = model
        self.xtrain = xtrain
        self.ytrain = ytrain
        self.xval = xval
        self.yval = yval

    def preprocessing(self, x):
        return x.reshape(x.shape[0], -1)

    def save_model(self, save_path):
        import pickle
        with open(save_path, 'wb') as model_file:
            pickle.dump(self.model, model_file)
        print(f'Model saved to {save_path}')

    def train(self, save_dir='.'):
        self.model.fit(self.preprocessing(self.xtrain),
                       self.preprocessing(self.ytrain),
                       eval_set=[(self.preprocessing(self.xval), self.preprocessing(self.yval))])
        save_path = os.path.join(save_dir, f'best-{self.model.__class__.__name__}.pkl')
        self.save_model(save_path=save_path)

    def predict(self, x):
        return self.model.predict(self.preprocessing(x))

    def evaluate(self, x, y):
        from sklearn.metrics import mean_absolute_error
        # print(self.preprocessing(y).shape)
        # print(self.predict(self.preprocessing(x)).shape)
        return mean_absolute_error(self.preprocessing(y), self.predict(self.preprocessing(x)))

    def plot(self, y, yhat, feature_names=None, save_dir='.', save_plots=True, num_elements=None):
```

```
        yhat = yhat.reshape(y.shape[0], y.shape[1], -1)
        super().plot(y, yhat, feature_names=feature_names, save_dir=save_dir, save_plots=save_plots, num_elements=num_elements)
```

4.1 Xây dựng training model cho giải thuật MLP

```

class MLP(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, ahead):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, output_size * ahead)  # Adjust for output sequence length
        self.ahead = ahead
        self.output_size = output_size

    def forward(self, x):
        # Flatten the input
        x = x.view(x.size(0), -1)  # Reshape from [batch, lag, features] to [batch, lag * features]
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        return x.view(-1, self.ahead, self.output_size)  # Reshape to [batch, ahead, features]
```

4.2 Xây dựng training model cho giải thuật RNN

```

class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, num_layers, ahead):
        super(RNN, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.ahead = ahead
        self.output_size = output_size

        # RNN Layer - can be replaced with nn.LSTM or nn.GRU
        self.rnn = nn.RNN(input_size, hidden_size, num_layers, batch_first=True)

        # Output layer
        self.fc = nn.Linear(hidden_size, output_size * ahead)

    def forward(self, x):
        # Initialize hidden state
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(x.device)

        # Forward propagate RNN
        out, _ = self.rnn(x, h0)

        # Get the last time step's output for each sequence
        out = out[:, -1, :]

        # Pass through the linear layer and reshape
        out = self.fc(out).view(-1, self.ahead, self.output_size)
        return out
```

4.3 Xây dựng training model cho giải thuật Bi-LSTM

```
class LSTM(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, num_layers, ahead):
        super(LSTM, self).__init__()
```

```
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.ahead = ahead
        self.output_size = output_size

        # LSTM Layer
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)

        # Output layer
        self.fc = nn.Linear(hidden_size, output_size * ahead)

    def forward(self, x):
        # Initialize hidden and cell states
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(x.device)
        c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(x.device)

        # Forward propagate LSTM
        out, _ = self.lstm(x, (h0, c0))

        # Get the last time step's output for each sequence
        out = out[:, -1, :]

        # Pass through the linear layer and reshape
        out = self.fc(out).view(-1, self.ahead, self.output_size)
        return out
```

4.4 Xây dựng training model cho giải thuật LSTM

```
class BiLSTM(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, num_layers, ahead):
        super(BiLSTM, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.ahead = ahead
        self.output_size = output_size

        # BiLSTM Layer
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True,
    bidirectional=True)

        # Output layer
        # Output size is doubled because BiLSTM has two hidden states for each layer
        self.fc = nn.Linear(hidden_size * 2, output_size * ahead)

    def forward(self, x):
        # Initialize hidden and cell states
        h0 = torch.zeros(self.num_layers * 2, x.size(0), self.hidden_size).to(x.device
    )  # 2 for bidirection
        c0 = torch.zeros(self.num_layers * 2, x.size(0), self.hidden_size).to(x.device
    )

        # Forward propagate BiLSTM
        out, _ = self.lstm(x, (h0, c0))

        # Get the last time step's output for each sequence
        out = out[:, -1, :]

        # Pass through the linear layer and reshape
        out = self.fc(out).view(-1, self.ahead, self.output_size)
        return out
```

Thực hành 5: (MLP Training and Performance Evaluation) Hãy cài đặt lại đoạn chương trình bên dưới huấn luyện và đánh giá performance của giải thuật MLP ứng với 3 case study.

5.1 Case study 1 (multivariate to multivariate):

```
MLP_multi2multi = MLP(input_size=multi2multi_loader.in_variable*input_size,
                      hidden_size=hidden_size,
                      output_size=multi2multi_loader.out_variable,
                      ahead=label_size)
MLP_multi2multi_manager = ModelManager(model=MLP_multi2multi,
                                       train_loader=multi2multi_loader.train_loader,
                                       val_loader=multi2multi_loader.val_loader,
                                       lr=learning_rate,
                                       patience=patience)
MLP_multi2multi_manager.train(num_epochs=num_epochs,
                              save_dir=os.path.join(weight_dir, sub_dir))
results.append({
    "Name": MLP_multi2multi_manager.model.__class__.__name__,
    "Type": sub_dir,
    "MAE": MLP_multi2multi_manager.evaluate(loader=multi2multi_loader.test_loader)
})
results[-1]
```

Question 4: Hãy cho biết kết quả MAE sau khi thực hiện huấn luyện mô hình MLP

```
a) 2.47
b) 2.57
c) 2.67
d) 2.77
```

5.2 Case study 2 (multivariate to univariate):

```
MLP_multi2uni = MLP(input_size=multi2uni_loader.in_variable*input_size,
                    hidden_size=hidden_size,
                    output_size=multi2uni_loader.out_variable,
                    ahead=label_size)
MLP_multi2uni_manager = ModelManager(model=MLP_multi2uni,
                                     train_loader=multi2uni_loader.train_loader,
                                     val_loader=multi2uni_loader.val_loader,
                                     lr=learning_rate,
                                     patience=patience)
MLP_multi2uni_manager.train(num_epochs=num_epochs,
                            save_dir=os.path.join(weight_dir, sub_dir))
results.append({
    "Name": MLP_multi2uni_manager.model.__class__.__name__,
    "Type": sub_dir,
    "MAE": MLP_multi2uni_manager.evaluate(loader=multi2uni_loader.test_loader)
})
results[-1]
```

Question 5: Hãy cho biết kết quả MAE sau khi thực hiện huấn luyện mô hình MLP cho case study trên

```
a) 2.47
b) 2.57
c) 2.67
d) 3.08
```

5.3 Case study 3 (univariate to univariate):

```
MLP_uni2uni = MLP(input_size=uni2uni_loader.in_variable*input_size,
```

```
                    hidden_size=hidden_size,
                    output_size=uni2uni_loader.out_variable,
                    ahead=label_size)
MLP_uni2uni_manager = ModelManager(model=MLP_uni2uni,
                                   train_loader=uni2uni_loader.train_loader,
                                   val_loader=uni2uni_loader.val_loader,
                                   lr=learning_rate,
                                   patience=patience)
MLP_uni2uni_manager.train(num_epochs=num_epochs,
                          save_dir=os.path.join(weight_dir, sub_dir))
results.append({
    "Name": MLP_uni2uni_manager.model.__class__.__name__,
    "Type": sub_dir,
    "MAE": MLP_uni2uni_manager.evaluate(loader=uni2uni_loader.test_loader)
})
results[-1]
```

Question 6: Hãy cho biết kết quả MAE sau khi thực hiện huấn luyện mô hình MLP cho case study trên

```
a) 2.47
b) 2.57
c) 2.29
d) 3.08
```

Thực hành 6: (RNN Training and Performance Evaluation) Hãy cài đặt lại đoạn chương trình bên dưới huấn luyện và đánh giá performance của giải thuật RNN ứng với 3 case study.

6.1 Case study 1 (multivariate to multivariate):

```
RNN_multi2multi = RNN(input_size=multi2multi_loader.in_variable,
                      hidden_size=hidden_size,
                      output_size=multi2multi_loader.out_variable,
                      ahead=label_size,
                      num_layers=num_layers)
RNN_multi2multi_manager = ModelManager(model=RNN_multi2multi,
                                       train_loader=multi2multi_loader.train_loader,
                                       val_loader=multi2multi_loader.val_loader,
                                       lr=learning_rate,
                                       patience=patience)
RNN_multi2multi_manager.train(num_epochs=num_epochs,
                              save_dir=os.path.join(weight_dir, sub_dir))
results.append({
    "Name": RNN_multi2multi_manager.model.__class__.__name__,
    "Type": sub_dir,
    "MAE": RNN_multi2multi_manager.evaluate(loader=multi2multi_loader.test_loader)
})
results[-1]
```

Question 7: Hãy cho biết kết quả MAE sau khi thực hiện huấn luyện mô hình RNN

```
a) 2.21
b) 2.57
c) 2.67
d) 2.77
```

6.2 Case study 2 (multivariate to univariate):

```
RNN_multi2uni = RNN(input_size=multi2uni_loader.in_variable,
                    hidden_size=hidden_size,
```

```
                    output_size=multi2uni_loader.out_variable,
                    ahead=label_size,
                    num_layers=num_layers)
RNN_multi2uni_manager = ModelManager(model=RNN_multi2uni,
                                     train_loader=multi2uni_loader.train_loader,
                                     val_loader=multi2uni_loader.val_loader,
                                     lr=learning_rate,
                                     patience=patience)
RNN_multi2uni_manager.train(num_epochs=num_epochs,
                            save_dir=os.path.join(weight_dir, sub_dir))
results.append({
    "Name": RNN_multi2uni_manager.model.__class__.__name__,
    "Type": sub_dir,
    "MAE": RNN_multi2uni_manager.evaluate(loader=multi2uni_loader.test_loader)
})
results[-1]
```

Question 8: Hãy cho biết kết quả MAE sau khi thực hiện huấn luyện mô hình RNN cho case study trên

```
a) 2.47
b) 2.57
c) 2.67
d) 2.63
```

6.3 Case study 3 (univariate to univariate):

```
RNN_uni2uni = RNN(input_size=uni2uni_loader.in_variable,
                  hidden_size=hidden_size,
                  output_size=uni2uni_loader.out_variable,
                  ahead=label_size,
                  num_layers=num_layers)
RNN_uni2uni_manager = ModelManager(model=RNN_uni2uni,
                                   train_loader=uni2uni_loader.train_loader,
                                   val_loader=uni2uni_loader.val_loader,
                                   lr=learning_rate,
                                   patience=patience)
RNN_uni2uni_manager.train(num_epochs=num_epochs,
                          save_dir=os.path.join(weight_dir, sub_dir))
results.append({
    "Name": RNN_uni2uni_manager.model.__class__.__name__,
    "Type": sub_dir,
    "MAE": RNN_uni2uni_manager.evaluate(loader=uni2uni_loader.test_loader)
})
results[-1]
```

Question 9: Hãy cho biết kết quả MAE sau khi thực hiện huấn luyện mô hình RNN cho case study trên

```
a) 2.47
b) 2.8
c) 2.29
d) 3.08
```

Thực hành 7: (LSTM Training and Performance Evaluation) Hãy cài đặt lại đoạn chương trình bên dưới huấn luyện và đánh giá performance của giải thuật LSTM ứng với 3 case study.

7.1 Case study 1 (multivariate to multivariate):

```
LSTM_multi2multi = LSTM(input_size=multi2multi_loader.in_variable,
```

```
                        hidden_size=hidden_size,
                        output_size=multi2multi_loader.out_variable,
                        ahead=label_size,
                        num_layers=num_layers)
LSTM_multi2multi_manager = ModelManager(model=LSTM_multi2multi,
                                        train_loader=multi2multi_loader.train_loader,
                                        val_loader=multi2multi_loader.val_loader,
                                        lr=learning_rate,
                                        patience=patience)
LSTM_multi2multi_manager.train(num_epochs=num_epochs,
                               save_dir=os.path.join(weight_dir, sub_dir))
results.append({
    "Name": LSTM_multi2multi_manager.model.__class__.__name__,
    "Type": sub_dir,
    "MAE": LSTM_multi2multi_manager.evaluate(loader=multi2multi_loader.test_loader)
})
results[-1]
```

Question 10: Hãy cho biết kết quả MAE sau khi thực hiện huấn luyện mô hình LSTM

```
a) 2.29
b) 2.57
c) 2.67
d) 3.18
```

7.2 Case study 2 (multivariate to univariate):

```
LSTM_multi2uni = LSTM(input_size=multi2uni_loader.in_variable,
                      hidden_size=hidden_size,
                      output_size=multi2uni_loader.out_variable,
                      ahead=label_size,
                      num_layers=num_layers)
LSTM_multi2uni_manager = ModelManager(model=LSTM_multi2uni,
                                      train_loader=multi2uni_loader.train_loader,
                                      val_loader=multi2uni_loader.val_loader,
                                      lr=learning_rate,
                                      patience=patience)
LSTM_multi2uni_manager.train(num_epochs=num_epochs,
                             save_dir=os.path.join(weight_dir, sub_dir))
results.append({
    "Name": LSTM_multi2uni_manager.model.__class__.__name__,
    "Type": sub_dir,
    "MAE": LSTM_multi2uni_manager.evaluate(loader=multi2uni_loader.test_loader)
})
results[-1]
```

Question 11: Hãy cho biết kết quả MAE sau khi thực hiện huấn luyện mô hình LSTM cho case study trên

```
a) 2.47
b) 2.57
c) 2.67
d) 3.18
```

7.3 Case study 3 (univariate to univariate):

```
LSTM_uni2uni = LSTM(input_size=uni2uni_loader.in_variable,
                    hidden_size=hidden_size,
                    output_size=uni2uni_loader.out_variable,
                    ahead=label_size,
                    num_layers=num_layers)
```

```
LSTM_uni2uni_manager = ModelManager(model=LSTM_uni2uni,
                                    train_loader=uni2uni_loader.train_loader,
                                    val_loader=uni2uni_loader.val_loader,
                                    lr=learning_rate,
                                    patience=patience)
LSTM_uni2uni_manager.train(num_epochs=num_epochs,
                           save_dir=os.path.join(weight_dir, sub_dir))
results.append({
    "Name": LSTM_uni2uni_manager.model.__class__.__name__,
    "Type": sub_dir,
    "MAE": LSTM_uni2uni_manager.evaluate(loader=uni2uni_loader.test_loader)
})
results[-1]
```

Question 12: Hãy cho biết kết quả MAE sau khi thực hiện huấn luyện mô hình LSTM cho case study trên

```
a) 2.47
b) 2.56
c) 2.29
d) 3.08
```

Thực hành 8: (Bi-LSTM Training and Performance Evaluation) Hãy cài đặt lại đoạn chương trình bên dưới huấn luyện và đánh giá performance của giải thuật Bi-LSTM ứng với 3 case study.

8.1 Case study 1 (multivariate to multivariate):

```
BiLSTM_multi2multi = BiLSTM(input_size=multi2multi_loader.in_variable,
                            hidden_size=hidden_size,
                            output_size=multi2multi_loader.out_variable,
                            ahead=label_size,
                            num_layers=num_layers)
BiLSTM_multi2multi_manager = ModelManager(model=BiLSTM_multi2multi,
                                          train_loader=multi2multi_loader.train_loader
    ,
                                          val_loader=multi2multi_loader.val_loader,
                                          lr=learning_rate,
                                          patience=patience)
BiLSTM_multi2multi_manager.train(num_epochs=num_epochs,
                                 save_dir=os.path.join(weight_dir, sub_dir))
results.append({
    "Name": BiLSTM_multi2multi_manager.model.__class__.__name__,
    "Type": sub_dir,
    "MAE": BiLSTM_multi2multi_manager.evaluate(loader=multi2multi_loader.test_loader)
})
results[-1]
```

Question 13: Hãy cho biết kết quả MAE sau khi thực hiện huấn luyện mô hình Bi-LSTM

```
a) 2.29
b) 2.56
c) 2.67
d) 2.77
```

8.2 Case study 2 (multivariate to univariate):

```
BiLSTM_multi2uni = BiLSTM(input_size=multi2uni_loader.in_variable,
                          hidden_size=hidden_size,
                          output_size=multi2uni_loader.out_variable,
                          ahead=label_size,
```

```
                            num_layers=num_layers)
BiLSTM_multi2uni_manager = ModelManager(model=BiLSTM_multi2uni,
                                        train_loader=multi2uni_loader.train_loader,
                                        val_loader=multi2uni_loader.val_loader,
                                        lr=learning_rate,
                                        patience=patience)
BiLSTM_multi2uni_manager.train(num_epochs=num_epochs,
                               save_dir=os.path.join(weight_dir, sub_dir))
results.append({
    "Name": BiLSTM_multi2uni_manager.model.__class__.__name__,
    "Type": sub_dir,
    "MAE": BiLSTM_multi2uni_manager.evaluate(loader=multi2uni_loader.test_loader)
})
results[-1]
```

Question 14: Hãy cho biết kết quả MAE sau khi thực hiện huấn luyện mô hình Bi-LSTM cho case study trên

a) 2.47
b) 2.57
c) 2.67
d) 2.74

8.3 Case study 3 (univariate to univariate):

```
BiLSTM_uni2uni = BiLSTM(input_size=uni2uni_loader.in_variable,
                        hidden_size=hidden_size,
                        output_size=uni2uni_loader.out_variable,
                        ahead=label_size,
                        num_layers=num_layers)
BiLSTM_uni2uni_manager = ModelManager(model=BiLSTM_uni2uni,
                                      train_loader=uni2uni_loader.train_loader,
                                      val_loader=uni2uni_loader.val_loader,
                                      lr=learning_rate,
                                      patience=patience)
BiLSTM_uni2uni_manager.train(num_epochs=num_epochs,
                             save_dir=os.path.join(weight_dir, sub_dir))
results.append({
    "Name": BiLSTM_uni2uni_manager.model.__class__.__name__,
    "Type": sub_dir,
    "MAE": BiLSTM_uni2uni_manager.evaluate(loader=uni2uni_loader.test_loader)
})
results[-1]
```

Question 15: Hãy cho biết kết quả MAE sau khi thực hiện huấn luyện mô hình Bi-LSTM cho case study trên

a) 2.47
b) 2.56
c) 2.29
d) 3.08

Thực hành 9: (XGboost Training and Performance Evaluation) Hãy cài đặt lại đoạn chương trình bên dưới huấn luyện và đánh giá performance của giải thuật XGBoost ứng với 3 case study.

9.1 Case study 1 (multivariate to multivariate):

```
XGBoost_multi2multi = XGBRegressor(**xgboost_config)
XGBoost_multi2multi_manager = MachineLearningModelManager(model=XGBoost_multi2multi,
                                                          xtrain=multi2multi_loader.
    X_train,
```

```
                                                    ytrain=multi2multi_loader.
    y_train,
                                                    xval=multi2multi_loader.
    X_val,
                                                    yval=multi2multi_loader.
    y_val)
XGBoost_multi2multi_manager.train(save_dir=os.path.join(weight_dir, sub_dir))
results.append({
    "Name": XGBoost_multi2multi_manager.model.__class__.__name__,
    "Type": sub_dir,
    "MAE": XGBoost_multi2multi_manager.evaluate(x=multi2multi_loader.X_test, y=
    multi2multi_loader.y_test)
})
results[-1]
```

Question 16: Hãy cho biết kết quả MAE sau khi thực hiện huấn luyện mô hình XGBoost

```
a) 2.29
b) 2.56
c) 2.02
d) 2.5
```