TRƯỜNG ĐẠI HỌC THỦY LỢI
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO BÀI TẬP
HỌC PHẦN PHÂN TÍCH CHUỖI THỜI GIAN

DỰ BÁO NHIỆT ĐỘ TRUNG BÌNH Ở DELHI DÙNG MỘT SỐ MÔ HÌNH HỌC MÁY / HỌC SÂU

Nhóm sinh viên thực hiện:
1. Vũ Anh Xuân - 2151264697
2. Đào Thị Trang - 2151264688
3. Khổng Quốc Trung - 2151264690
4. Nguyễn Quang Việt - 2151264694

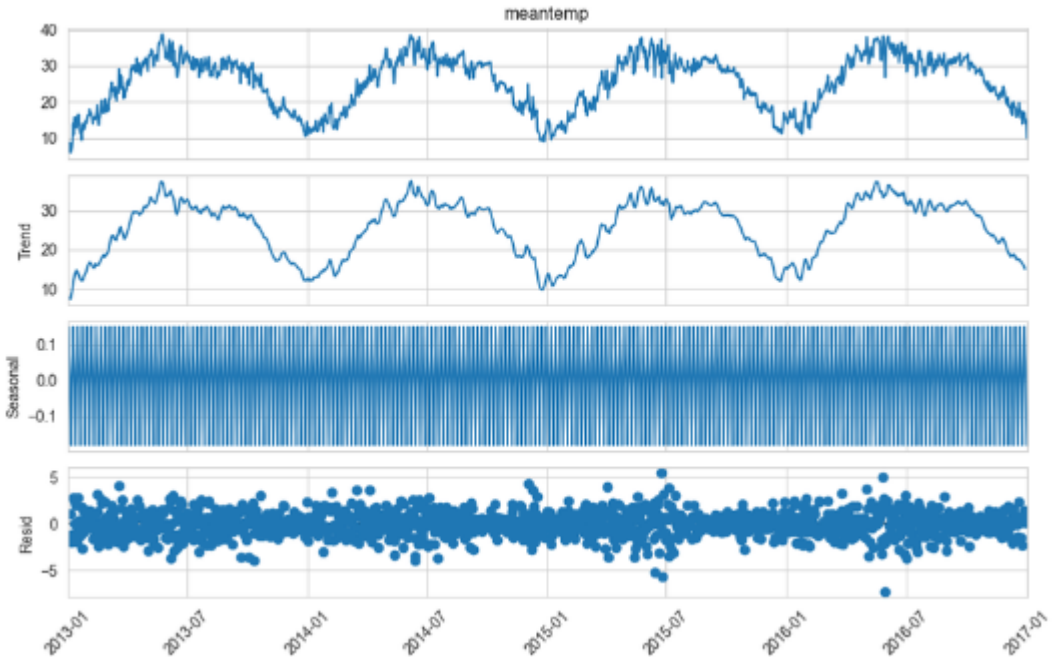
NGƯỜI HƯỚNG DẪN: Th.S Trần Anh Đạt

Hà Nội, tháng 6 năm 2024

Mục lục

1. Lý thuyết

- Chuỗi thời gian (time series) là một chuỗi các điểm dữ liệu xảy ra theo thứ tự liên tiếp trong một khoảng thời gian.
- Một chuỗi thời gian sẽ theo dõi chuyển động của các điểm dữ liệu đã chọn trong một khoảng thời gian xác định.
- Ứng dụng của chuỗi thời gian trải khắp các ngành công nghiệp khác nhau như: quan sát hoạt động sóng điện trong não, đo lượng mưa, dự báo giá cổ phiếu,...
- Các yếu tố cần lưu ý khi phân tích dữ liệu chuỗi thời gian
- Resid: Yếu tố bất thường(Nhiễu trắng).
- Trend: Xu hướng.
- Seasonal: Mùa vụ.



-> Nhìn vào biểu đồ ta thấy biểu đồ thường và biểu đồ xu hướng thấy được xu hướng tổng quan của dữ liệu theo thời gian: Nhiệt độ tăng vào mùa hè từ tháng 4 đến tháng 7, đỉnh điểm vào tháng 6 hàng năm và nhiệt độ giảm sâu vào mùa đông, nhiệt độ thấp nhất vào tháng 1 hàng năm.
Biểu đồ mùa vụ cho thấy nhiệt độ trung bình trong ngày có xu hướng tăng dần theo thời gian, nhưng cũng có biến đổi theo mùa rõ ràng với nhiệt độ cao hơn vào mùa hè và thấp hơn vào mùa đông. Biểu đồ mùa vụ cho thấy biến động tăng hoặc giảm lặp đi lặp lại theo mùa trong một năm.

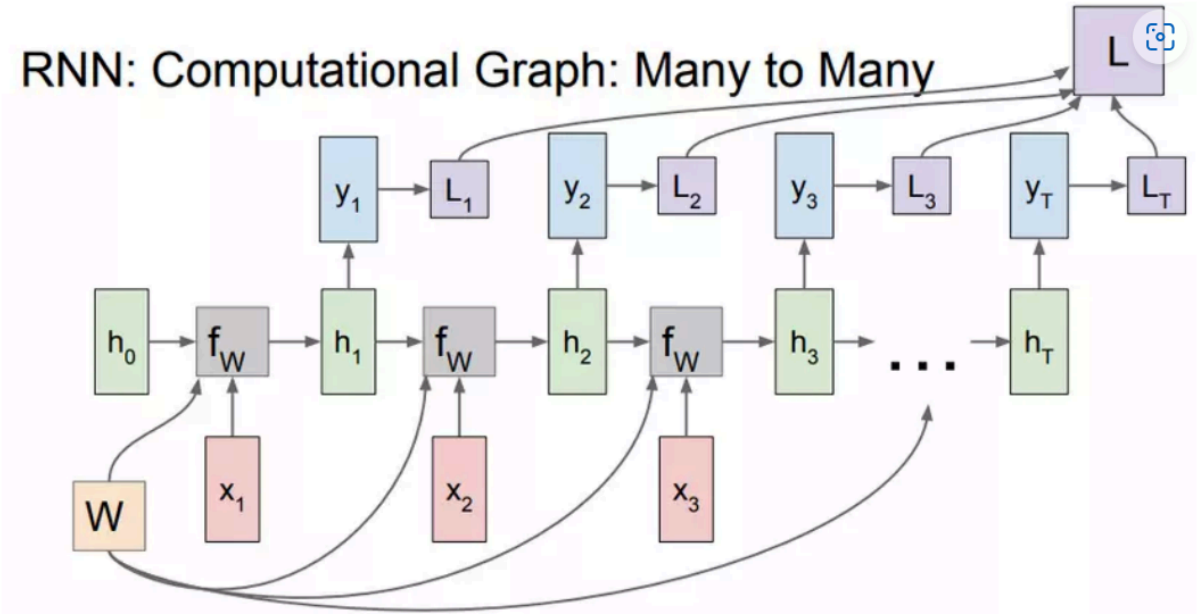
2. Hiểu dữ liệu

Dữ liệu lấy trên Kaggle:
https://www.kaggle.com/datasets/sumanthvrao/daily-climate-time-series-data
gồm 1576 mẫu dữ liệu và 5 cột đặc trưng:

- date: Ngày tháng năm.
- meantemp: Nhiệt độ trung bình trong một ngày.
- humidity: Giá trị độ ẩm trong ngày.
- wind_speed: Tốc độ gió.
- meanpressure: Áp suất.y

Dữ liệu lấy từ 1/2013 đến 4/2017.

3. Mô hình RNN



Giải thích một chút: Nếu như mạng Neural Network chỉ là input layer x đi qua hidden layer h và cho ra output layer y với ***full connected*** giữa các layer thì trong RNN, các input x_t sẽ được kết hợp với hidden layer h_{t-1} bằng hàm f_W để tính toán ra hidden layer h_t hiện tại và output y_t sẽ được tính ra từ h_t, W là tập các trọng số và nó được ở tất cả các cụm, các $L_1, L_2, ..., L_t$ là các hàm mất mát sẽ được giải thích sau. Như vậy kết quả từ các quá trình tính toán trước đã được "nhớ" bằng cách kết hợp thêm h_{t-1} tính ra h_t để tăng độ chính xác cho những dự đoán ở hiện tại. Cụ thể quá trình tính toán được viết dưới dạng toán như sau:

$h_t = f_W(h_{t-1}, x_t)$

Hàm f_W chúng ta sẽ xử dụng hàm **tanh**, công thức trên sẽ trở thành :

$$h_t = tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$
$$y_t = W_{hy}h_t$$

Ưu điểm của RNN

1. Xử lý dữ liệu tuần tự:
    - RNN được thiết kế để xử lý các chuỗi dữ liệu, nơi mà các mẫu dữ liệu có sự liên hệ về thứ tự. Điều này làm cho RNN rất hiệu quả trong các tác vụ như dịch máy, nhận diện giọng nói, và phân tích chuỗi thời gian.
2. Khả năng ghi nhớ ngữ cảnh:
    - RNN có khả năng ghi nhớ thông tin từ các bước trước đó trong chuỗi, giúp nó hiểu và xử lý dữ liệu dựa trên ngữ cảnh trước đó. Điều này đặc biệt quan trọng trong các ứng dụng như xử lý ngôn ngữ tự nhiên (NLP), nơi mà ý nghĩa của một từ có thể phụ thuộc vào các từ trước đó.
3. Mô hình hóa chuỗi dữ liệu phức tạp:
    - RNN có thể mô hình hóa các mối quan hệ phức tạp trong chuỗi dữ liệu, giúp nó có khả năng dự đoán và sinh dữ liệu trong nhiều bài toán khác nhau như tạo văn bản tự động, dự đoán chuỗi thời gian, và nhận dạng mẫu.

Nhược điểm của RNN

1. Vấn đề biến mất và bùng nổ gradient (Vanishing and Exploding Gradient):
    - Khi chuỗi dữ liệu rất dài, gradient trong quá trình huấn luyện có thể trở nên rất nhỏ (vanishing gradient) hoặc rất lớn (exploding gradient). Điều này gây ra khó khăn trong việc cập nhật trọng số của mạng, dẫn đến việc mạng học chậm hoặc không học được.
    - Vanishing Gradient: Khi gradient trở nên quá nhỏ, các trọng số của mạng gần như không thay đổi, làm cho RNN không thể học được các mối quan hệ dài hạn trong dữ liệu.
    - Exploding Gradient: Khi gradient trở nên quá lớn, các trọng số của mạng có thể thay đổi rất nhanh và trở nên không ổn định, gây ra lỗi số học.
2. Khả năng ghi nhớ ngắn hạn:
    - RNN cơ bản có khả năng ghi nhớ thông tin từ các bước trước đó, nhưng gặp khó khăn trong việc ghi nhớ thông tin từ các bước thời gian rất xa trong quá khứ. Điều này hạn chế hiệu quả của RNN trong các tác vụ yêu cầu ghi nhớ dài hạn.
3. Tính toán phức tạp và chi phí cao:
    - Quá trình huấn luyện RNN, đặc biệt là với các chuỗi dữ liệu dài, đòi hỏi nhiều tài nguyên tính toán và thời gian. Điều này có thể làm cho việc huấn luyện RNN trở nên chậm và tốn kém.
4. Khó khăn trong việc song song hóa:
    - Do tính tuần tự của RNN, việc song song hóa trong quá trình huấn luyện và dự đoán trở nên khó khăn. Điều này làm giảm hiệu suất khi xử lý các chuỗi dữ liệu lớn trên các hệ thống đa lõi hoặc GPU

Giải pháp khắc phục nhược điểm của RNN

1. LSTM (Long Short-Term Memory):
    - LSTM giải quyết vấn đề biến mất gradient bằng cách sử dụng các cổng (gates) để kiểm soát luồng thông tin, cho phép nó lưu trữ thông tin trong thời gian dài hơn. Các cổng bao gồm:
        - Cổng quên (Forget Gate): Quyết định thông tin nào nên được quên đi.
        - Cổng đầu vào (Input Gate): Quyết định thông tin nào nên được lưu trữ vào trạng thái ẩn.
        - Cổng đầu ra (Output Gate): Quyết định thông tin nào nên được sử dụng để tạo đầu ra.

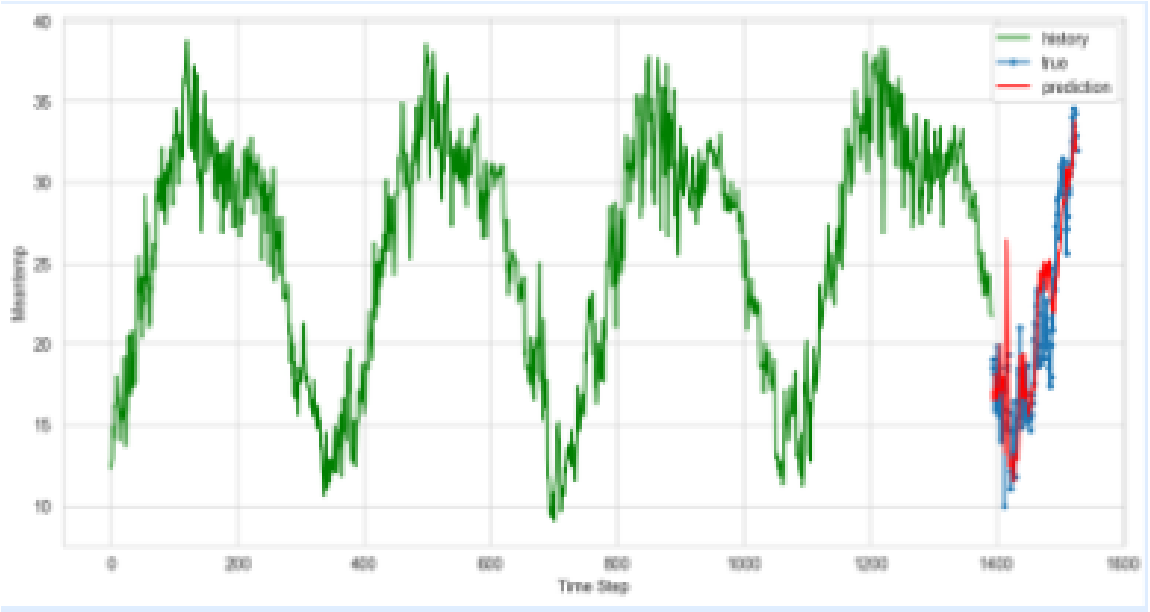
2. GRU (Gated Recurrent Unit):

- GRU đơn giản hóa cấu trúc của LSTM bằng cách kết hợp cổng đầu vào và cổng quên thành một cổng duy nhất gọi là cổng cập nhật (Update Gate), và sử dụng thêm một cổng điều khiển (Reset Gate) để kiểm soát thông tin từ trạng thái ẩn trước đó. GRU vẫn giữ được khả năng ghi nhớ dài hạn trong khi giảm bớt độ phức tạp tính toán.

Code và kết quả thông số đánh giá mô hình:

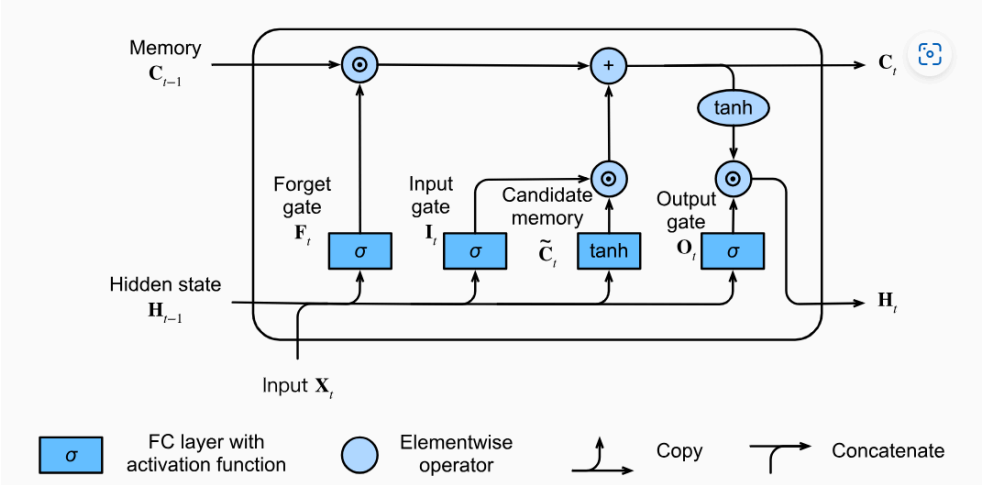
```
model = Sequential()

model.add(SimpleRNN(units=128, activation="tanh", input_shape=(time_steps, n_features), return_sequences=True))
model.add(Dropout(0.2))
model.add(SimpleRNN(units=64, activation="tanh", return_sequences=True))
model.add(SimpleRNN(units=32, activation="tanh"))
model.add(Dense(64, activation="relu"))
model.add(Dense(32, activation="tanh"))
model.add(Dense(16, activation="relu"))
model.add(Dense(1))

model.compile(optimizer="adam", loss="mse")
```

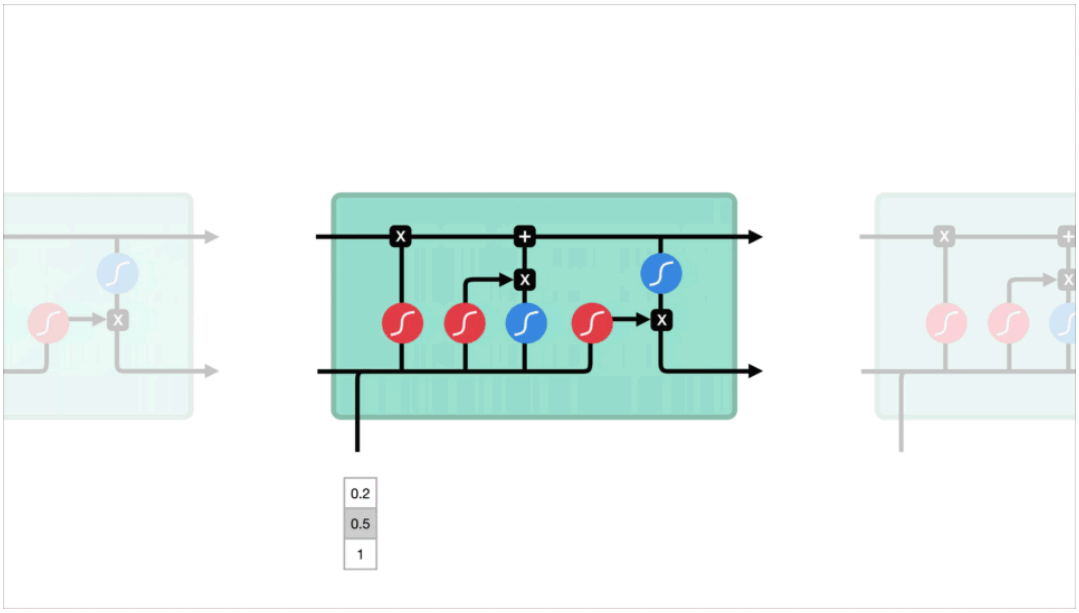
```
early_stop = EarlyStopping(monitor = 'val_loss', patience = 5, restore_best_weights = True)
model.fit(X_train, y_train,
          epochs=30,
          batch_size=64,
          validation_split=0.3,
          callbacks=[early_stop])
```



```
r2_score: 0.7664285654800869
mae: 0.19498599029231536
mse: 0.06023106530595436
rmse: 0.2454201811301474
```

4. Mô hình LSTM





Mạng LSTM có thể bao gồm nhiều tế bào LSTM (LSTM memory cell) liên kết với nhau và kiến trúc cụ thể của mỗi tế bào được biểu diễn như trong Hình 2. Ý tưởng của LSTM là bổ sung thêm trạng thái bên trong tế bào (cell internal state) s_t và ba cổng sàng lọc các thông tin đầu vào và đầu ra cho tế bào bao gồm forget gate f_t, input gate i_t và output gate o_t. Tại mỗi bước thời gian ttt, các cổng đều lần lượt nhận giá trị đầu vào x_t (đại diện cho một phần tử trong chuỗi đầu vào) và giá trị h_{t-1} có được từ đầu ra của memory cell từ bước thời gian trước đó t−1. Các cổng đều đóng vai trò có nhiệm vụ sàng lọc thông tin với mỗi mục đích khác nhau:

- Forget gate: Có nhiệm vụ loại bỏ những thông tin không cần thiết nhận được khỏi cell internal state.
- Input gate: Có nhiệm vụ chọn lọc những thông tin cần thiết nào được thêm vào cell internal state.
- Output gate: Có nhiệm vụ xác định những thông tin nào từ cell internal state được sử dụng như đầu ra.

Trước khi trình bày các phương trình mô tả cơ chế hoạt động bên trong của một tế bào LSTM, chúng ta sẽ thống nhất quy ước một số ký hiệu được sử dụng sau đây:

- x_t là vector đầu vào tại mỗi bước thời gian \(t \).
- $W_{f,x}, W_{f,h}, W_{\tilde{s},x}, W_{\tilde{s},h}, W_{i,x}, W_{i,h}, W_{o,x}, W_{o,h}$ là các ma trận trọng số trong mỗi tế bào LSTM.
- $b_f, b_{\tilde{s}}, b_i, b_o$ là các vector bias.
- f_t, i_t, o_t lần lượt chứa các giá trị kích hoạt lần lượt cho các cổng forget gate, input gate và output gate tương ứng.
- $s_t, \tilde{s}$ lần lượt là các vector đại diện cho cell internal state và candidate value.
- h_t là giá trị đầu ra của tế bào LSTM.

Trong quá trình lan truyền xuôi (forward pass), cell internal state s_t và giá trị đầu ra h_t
được tính như sau:

- Ở bước đầu tiên, tế bào LSTM quyết định những thông tin nào cần được loại bỏ từ cell internal state ở bước thời gian trước đó s_{t-1}. Activation value f_t của forget gate tại thời gian t được tính dựa trên giá trị đầu vào hiện tại x_t và giá trị ẩn h_{t-1} từ tế bào LSTM ở bước trước đó và bias b_f của forget gate. Hàm sigmoid function biến đổi tất cả activation value về miền có giá trị trong khoảng từ 0 (hoàn toàn quên) và 1 (hoàn toàn ghi nhớ):

$$f_t = \sigma\left(W_{f,x}x_t + W_{f,h}h_{t-1} + b_f\right)$$

- Ở bước thứ hai, tế bào LSTM quyết định những thông tin nào cần được thêm vào cell internal state s_t. Bước này bao gồm hai quá trình tính toán đối với $\tilde{s_t}$ và f_t. Candidate value \(\tilde{s}_t\) biểu diễn những thông tin tiềm năng cần được thêm vào cell internal state được tính như sau:

$$\tilde{s_t} = \tanh\left(W_{\tilde{s},x}x_t + W_{\tilde{s},h}h_{t-1} + b_{\tilde{s}}\right)$$

Activation value i_t của input gate theo đó cũng được tính như sau:

$$i_t = \tanh(W_{i,x}x_t + W_{i,h}h_{t-1} + b_i)$$

- Ở bước thứ ba, giá trị mới của cell internal state s_t được tính dựa trên kết quả tính toán thu được từ các bước trước với phép nhân Hadamard theo từng phần tử (Hadamard product) được ký hiệu bằng O:

$$s_t = f_t \circ s_{t-1} + i_t \circ \tilde{s_t}$$

- Ở bước cuối cùng, giá trị đầu ra h_t của tế bào LSTM được tính toán dựa theo hai phương trình sau:

$$o_t = \sigma\left(W_{o,x}x_t + W_{o,h}h_{t-1} + b_o\right)$$

$$h_t = o_t \circ \tanh(s_t)$$

- Cổng vào (input gate): Quyết định thông tin mới nào từ đầu vào hiện tại sẽ được thêm vào ô nhớ.
- Cổng quên (forget gate): Quyết định thông tin nào từ ô nhớ trước đó sẽ bị loại bỏ.
- Cổng ra (output gate): Quyết định phần nào của ô nhớ sẽ được sử dụng để tạo ra đầu ra của LSTM ở bước thời gian hiện tại.

Ưu Điểm của LSTM

1. Khả Năng Ghi Nhớ Dài Hạn

- Trạng Thái Tế Bào (Cell State): LSTM có khả năng lưu trữ thông tin qua các bước thời gian dài nhờ vào trạng thái tế bào, cho phép nó ghi nhớ thông tin dài hạn mà các RNN truyền thống không thể.
- Cổng Quên (Forget Gate): Giúp LSTM chọn lọc thông tin cần giữ lại hoặc loại bỏ, tối ưu hóa việc lưu trữ thông tin theo thời gian.

2. Giải Quyết Vấn Đề Vanishing Gradient

- Cơ Chế Cổng (Gating Mechanism): Các cổng trong LSTM giúp kiểm soát luồng thông tin và gradient, giảm thiểu vấn đề vanishing gradient, cho phép học tốt hơn các phụ thuộc dài hạn.

3. Đa Dạng Ứng Dụng

- Xử Lý Ngôn Ngữ Tự Nhiên (NLP): LSTM rất hiệu quả trong các tác vụ như dịch máy, nhận dạng giọng nói, tạo văn bản tự động.
- Chuỗi Thời Gian: LSTM có thể dự đoán dữ liệu chuỗi thời gian như dự báo giá cổ phiếu, phân tích dữ liệu cảm biến.

4. Khả Năng Tự Điều Chỉnh

- Học Không Giám Sát và Giám Sát: LSTM có thể học từ dữ liệu có nhãn và không có nhãn, cho phép áp dụng trong nhiều lĩnh vực khác nhau.
- Thích Ứng Linh Hoạt: Nhờ vào cơ chế cổng, LSTM có thể tự điều chỉnh để phù hợp với các dạng dữ liệu và bài toán khác nhau.

Nhược Điểm của LSTM

1. Độ Phức Tạp Cao

- Tính Toán Phức Tạp: LSTM có nhiều cổng và trạng thái hơn so với các RNN truyền thống, dẫn đến việc tính toán phức tạp và chậm hơn.
- Số Lượng Tham Số: Do có nhiều cổng và trạng thái, LSTM có nhiều tham số hơn cần phải học, đòi hỏi nhiều tài nguyên tính toán và thời gian huấn luyện hơn.

2. Yêu Cầu Nhiều Tài Nguyên

- Bộ Nhớ và CPU/GPU: LSTM yêu cầu nhiều bộ nhớ và tài nguyên tính toán, đặc biệt khi xử lý các chuỗi dữ liệu dài hoặc huấn luyện trên các tập dữ liệu lớn.

3. Khả Năng Quá Khớp (Overfitting)

- Quá Khớp: Với nhiều tham số, LSTM dễ bị quá khớp với dữ liệu huấn luyện, đặc biệt khi không có đủ dữ liệu huấn luyện hoặc dữ liệu không đa dạng.
- Regularization: Cần áp dụng các kỹ thuật regularization như dropout, L2 regularization để giảm thiểu nguy cơ quá khớp.

4. Khó Khăn Trong Việc Điều Chỉnh Siêu Tham Số (Hyperparameter Tuning)

- Hyperparameter Tuning: LSTM có nhiều siêu tham số cần điều chỉnh như số lượng đơn vị trong từng lớp, tỷ lệ học, kích thước batch, và số lớp LSTM, làm cho quá trình tìm kiếm cấu hình tối ưu trở nên phức tạp và tốn thời gian.

5. Thời Gian Huấn Luyện Lâu

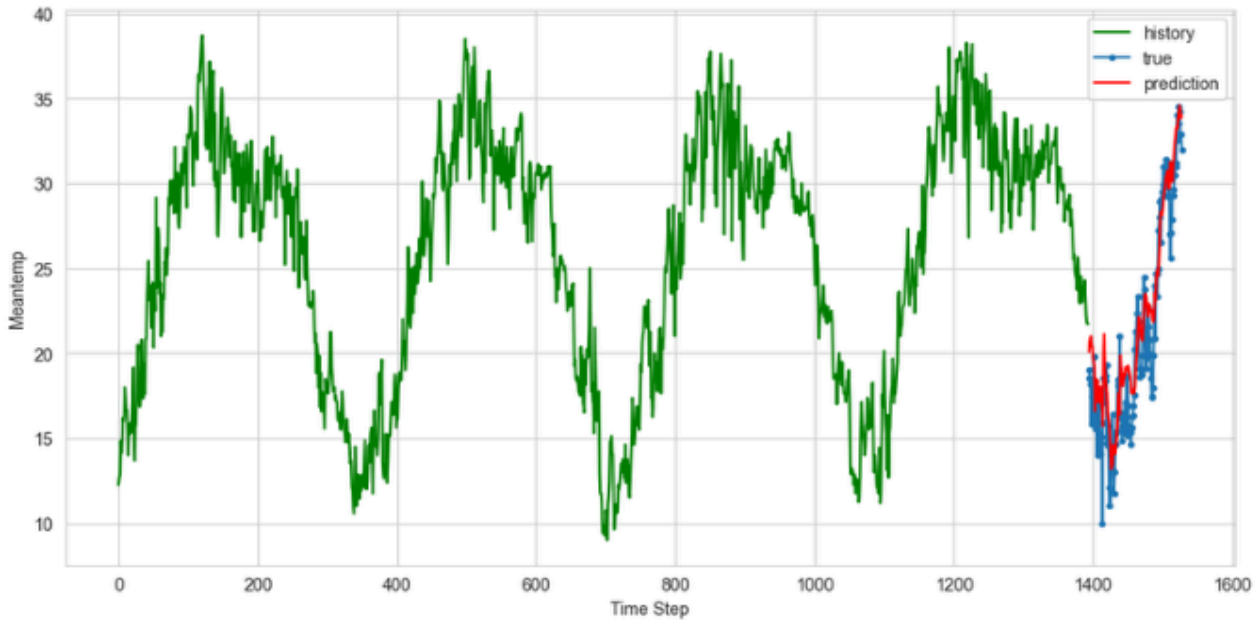
- Thời Gian Huấn Luyện: Do tính toán phức tạp và nhiều tham số, việc huấn luyện LSTM có thể mất rất nhiều thời gian, đặc biệt với các mô hình lớn và dữ liệu phong phú.

Code và kết quả thông số đánh giá mô hình:

```
model = Sequential()
model.add(Bidirectional(LSTM(units=128, input_shape=(time_steps, n_features))))

model.add(Dropout(0.2))

model.add(Dense(units=64, activation='relu'))

model.add(Dense(units=1))

model.compile(optimizer='adam', loss='mse')

model.summary()
```

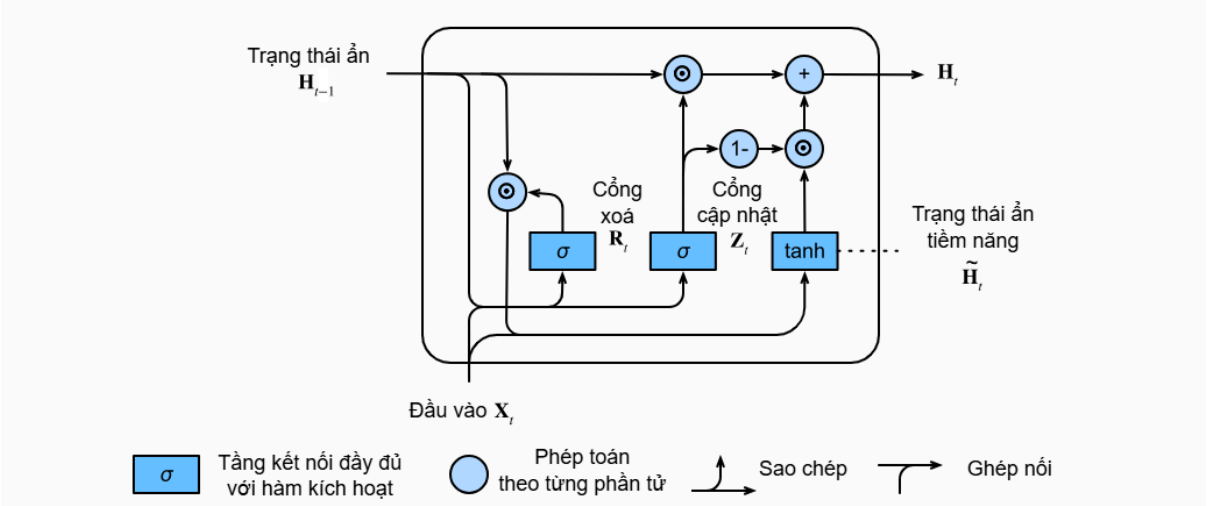
```
early_stop = EarlyStopping(monitor = 'val_loss', patience = 5, restore_best_weights = True)
```

```
model.fit(X_train, y_train,
          epochs=30,
          batch_size=64,
          validation_split=0.3,
          callbacks=[early_stop])
```



```
r2_score: 0.8501874391625632
mae: 0.1601310582659704
mse: 0.0386321647336658
rmse: 0.1965506670903607
```

5. Mô hình GRU



GRU (Gated Recurrent Unit) là một loại mạng nơ-ron hồi quy (Recurrent Neural Network - RNN), được thiết kế để giải quyết một số hạn chế của RNN truyền thống, đặc biệt là vấn đề biến mất hoặc nổ gradient khi huấn luyện mạng nơ-ron dài hạn.

Cấu trúc của GRU
GRU là một biến thể đơn giản hơn của LSTM (Long Short-Term Memory) nhưng vẫn giữ được các ưu điểm chính. Cấu trúc của GRU bao gồm hai loại cổng chính:

- Cổng cập nhật (Update Gate): Kết hợp chức năng của cổng vào và cổng quên trong LSTM, quyết định bao nhiêu phần của trạng thái ẩn trước đó và đầu vào mới sẽ được lưu trữ.
- Cổng xoá (Reset Gate): Quyết định bao nhiêu phần của trạng thái ẩn trước đó sẽ được quên đi.

$$\mathbf{R}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xr} + \mathbf{H}_{t-1} \mathbf{W}_{hr} + \mathbf{b}_r),$$
$$\mathbf{Z}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xz} + \mathbf{H}_{t-1} \mathbf{W}_{hz} + \mathbf{b}_z).$$

$$\tilde{\mathbf{H}}_t = \tanh(\mathbf{X}_t \mathbf{W}_{xh} + (\mathbf{R}_t \odot \mathbf{H}_{t-1}) \mathbf{W}_{hh} + \mathbf{b}_h).$$

$$\mathbf{H}_t = \mathbf{Z}_t \odot \mathbf{H}_{t-1} + (1 - \mathbf{Z}_t) \odot \tilde{\mathbf{H}}_t.$$

Ưu điểm của GRU

- Đơn giản hơn LSTM: GRU có cấu trúc đơn giản hơn so với LSTM (Long Short-Term Memory) vì nó có ít cổng hơn (2 cổng so với 3 cổng của LSTM). Do đó, GRU thường nhanh hơn và yêu cầu ít tài nguyên tính toán hơn, đồng thời có hiệu suất tương đương hoặc tốt hơn trong nhiều trường hợp.
- Hiệu quả trong xử lý tuần tự: GRU thường cho hiệu suất tốt trong các tác vụ liên quan đến chuỗi dữ liệu như xử lý ngôn ngữ tự nhiên và phân tích chuỗi thời gian.
- Giảm hiện tượng mất mát gradient (vanishing gradient):
  GRU sử dụng các cổng (gates) để kiểm soát dòng chảy thông tin, giúp giảm hiện tượng mất mát gradient so với RNN truyền thống. Điều này cho phép GRU học được các phụ thuộc dài hạn trong dữ liệu chuỗi.
- Hiệu suất tốt trong nhiều tác vụ:
  GRU thường hoạt động tốt trong các tác vụ như dịch máy, nhận diện giọng nói, và phân loại văn bản. Hiệu suất của nó thường ngang ngửa hoặc thậm chí vượt trội hơn so với LSTM.
- Tối ưu hóa tốt hơn:
  Vì GRU có ít tham số hơn so với LSTM, việc tối ưu hóa mô hình trở nên dễ dàng hơn, giảm khả năng bị overfitting (quá khớp) trên tập dữ liệu huấn luyện.

Nhược điểm của GRU:

- Không có sự linh hoạt của LSTM:
  Mặc dù GRU thường hoạt động tốt trong nhiều tình huống, nhưng trong một số trường hợp cụ thể, LSTM có thể vượt trội hơn nhờ vào cấu trúc phức tạp hơn với các cổng riêng biệt cho việc ghi nhớ và quên thông tin.
- Thiếu nghiên cứu so với LSTM:
  LSTM được nghiên cứu rộng rãi và có nhiều cải tiến qua thời gian. GRU, mặc dù hiệu quả, nhưng vẫn có ít nghiên cứu hơn và ít hiểu biết sâu rộng về cách tốt nhất để tối ưu hóa và triển khai nó trong một số tác vụ cụ thể.

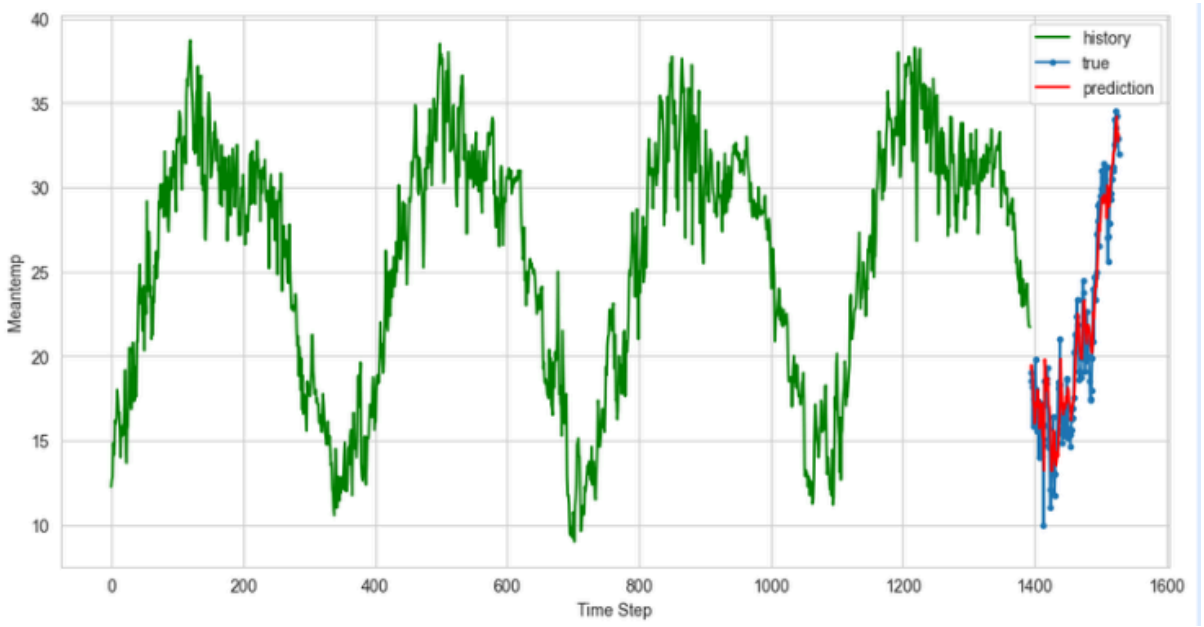
Kết luận

Mô hình GRU là một lựa chọn mạnh mẽ và hiệu quả cho các tác vụ liên quan đến dữ liệu chuỗi thời gian, đặc biệt là khi cần một mô hình với cấu trúc đơn giản và hiệu quả tính toán cao. Tuy nhiên, việc lựa chọn giữa GRU và LSTM (hoặc các mô hình khác) cần dựa vào đặc điểm cụ thể của dữ liệu và yêu cầu của tác vụ mà bạn đang xử lý.

Code và kết quả thông số đánh giá mô hình:

```
model = Sequential()
model.add(Bidirectional(GRU(units=128, return_sequences=True, input_shape=(time_steps, n_features))))
model.add(Dropout(rate=0.2))
model.add(Bidirectional(GRU(units=64, return_sequences=True)))
model.add(Bidirectional(GRU(units=32, return_sequences=False)))

model.add(Dense(units=64, activation='relu'))
model.add(Dropout(rate=0.2))
model.add(Dense(units=32, activation='relu'))
model.add(Dense(units=16, activation='relu'))
model.add(Dense(units=1))

optimizer = Adam(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='mse')
model.summary()
```

```
early_stop = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

model.fit(X_train, y_train,
          epochs=30,
          batch_size=64,
          validation_split=0.3,
          callbacks=[early_stop])
```



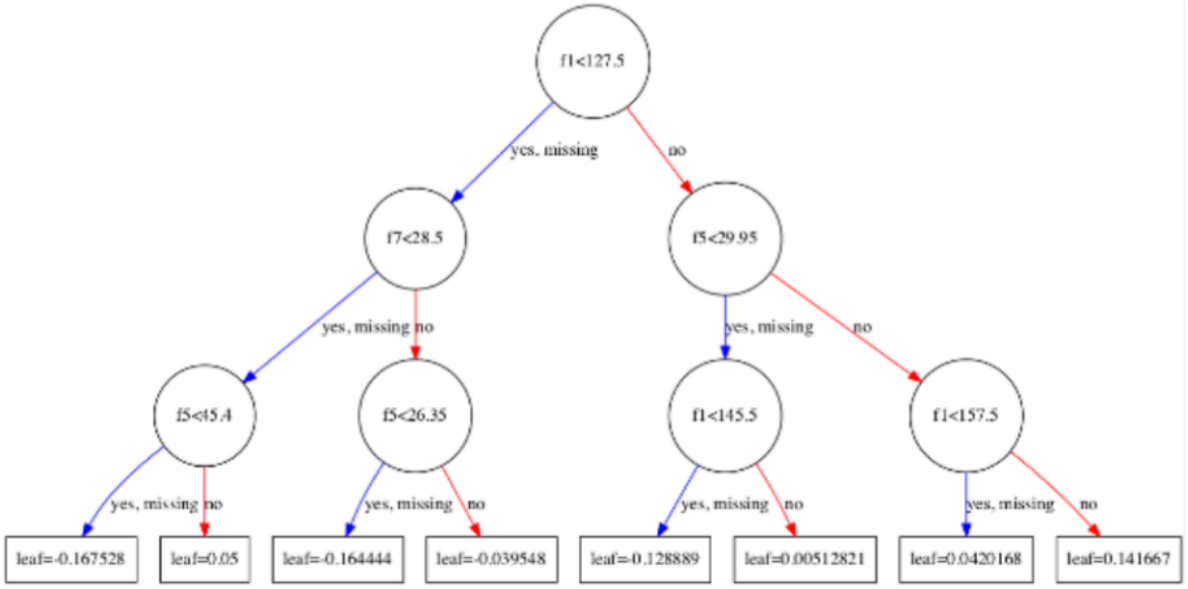

```
r2_score: 0.8958118377135013
mae: 0.13344836002781021
mse: 0.026867001179677752
rmse: 0.16391156511874785
```

6. Mô hình XGboost

XGBoost tối ưu cho các bài toán học máy dựa trên cây quyết định, được thiết kế để có hiệu suất tính toán cao và khả năng mở rộng mạnh mẽ.

Được sử dụng rộng rãi cho các nhiệm vụ học có giám sát, đặc biệt là trong phân loại và hồi quy.

Là một trong những thuật toán phổ biến nhất được sử dụng trong các cuộc thi khoa học dữ liệu và các dự án thực tế vì nó thường cho kết quả rất tốt.



- Cách thức hoạt động:
  1.Khởi tạo mô hình với một cây đơn giản.
  2.Tạo các cây mới:
  - Tại mỗi bước, thêm một cây mới dựa trên các residual errors (phần dư) từ các dự đoán trước đó.
  - Cây mới cố gắng dự đoán các lỗi này, và kết quả dự đoán được tổng hợp lại với các dự đoán trước đó.
  3.Cập nhật mô hình:
  - Kết hợp các dự đoán từ tất cả các cây lại với nhau để đưa ra dự đoán cuối cùng.
  4.Regularization:
  Thực hiện điều chuẩn để tránh overfitting và cải thiện khả năng tổng quát hóa của mô hình.

- Hàm mất mát:
  XGBoost sử dụng một hàm mất mát tổng quát trong quá trình tối ưu hóa. Hàm mất mát này bao gồm cả hàm mất mát chính (objective loss function) và một phần điều chuẩn (regularization term) để ngăn ngừa overfitting

$$\text{Loss} = \sum_{i=1}^{n} L(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

  Trong đó:
  $l(yi,y\hat{}i)$ là hàm mất mát chính (ví dụ như MSE, logistic loss).
  $\Omega(fk)$ là phần điều chuẩn của cây k, thường bao gồm độ phức tạp của cây (số lượng lá, độ sâu của cây).

- Việc tối ưu hàm mất mát giúp mô hình đạt được hiệu suất tốt trên dữ liệu huấn luyện mà vẫn duy trì khả năng tổng quát tốt trên dữ liệu mới (để tránh overfitting).

- **Ưu điểm XGboost:**

  XGBoost hỗ trợ tính toán song song, giúp tăng tốc quá trình huấn luyện mô hình.

  Sử dụng bộ nhớ hiệu quả, cho phép xử lý các tập dữ liệu lớn

  Có thể xử lý các tập dữ liệu lớn và phức tạp trên các hệ thống đơn hoặc hệ thống phân tán.

  Giảm thiểu overfitting thông qua việc điều chuẩn L1 (lasso) và L2 (ridge), giúp mô hình tổng quát hóa tốt hơn.

  Tự động xử lý missing value

  Tự động cắt tỉa cây, tự động bỏ qua những lá, nút không mang giá trị tích cực trong quá trình mở rộng cây

- **Nhược điểm XGboost:**

  Cần Nhiều Tài Nguyên Tính Toán và Bộ Nhớ

  Có nhiều tham số cần được tối ưu hóa để đạt hiệu suất tốt nhất

  XGBoost vẫn có nguy cơ bị overfitting nếu không được điều chỉnh đúng cách

  Thời gian huấn luyện có thể khá dài với các tập dữ liệu rất lớn hoặc khi số lượng cây quyết định (trees) trong mô hình rất lớn

  Phụ thuộc vào chất lượng của dữ liệu đầu vào

- Ứng dụng:

  XGBoost được sử dụng rộng rãi trong nhiều lĩnh vực khác nhau bao gồm:

  Tài chính: Dự đoán giá cổ phiếu, phát hiện gian lận.

  Y tế: Chẩn đoán bệnh, dự đoán kết quả điều trị.

  Tiếp thị: Phân tích hành vi khách hàng, dự đoán hiệu suất chiến dịch.

  Thương mại điện tử: Dự đoán nhu cầu, phân loại sản phẩm.

Code và kết quả thông số đánh giá mô hình:

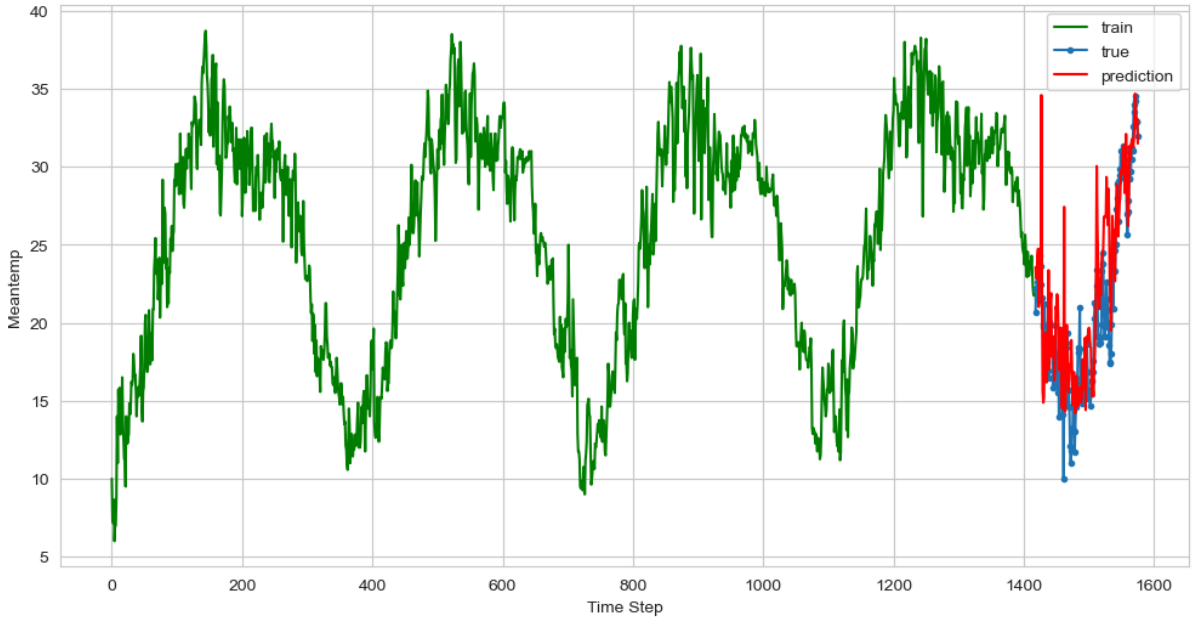
```
reg = xgb.XGBRegressor(n_estimators=1000, early_stopping_rounds=50,
                       learning_rate=0.008)
reg.fit(X_train, y_train,
        eval_set=[(X_train, y_train), (X_test, y_test)],
        verbose=40)
```

```
[0]     validation_0-rmse:7.28986       validation_1-rmse:7.44924
[40]    validation_0-rmse:5.52796       validation_1-rmse:5.64122
[80]    validation_0-rmse:4.28666       validation_1-rmse:4.46818
[120]   validation_0-rmse:3.42162       validation_1-rmse:3.78233
[160]   validation_0-rmse:2.83652       validation_1-rmse:3.42286
[200]   validation_0-rmse:2.44513       validation_1-rmse:3.27548
[240]   validation_0-rmse:2.19109       validation_1-rmse:3.23015
[280]   validation_0-rmse:2.01931       validation_1-rmse:3.24287
[293]   validation_0-rmse:1.97625       validation_1-rmse:3.24786
```

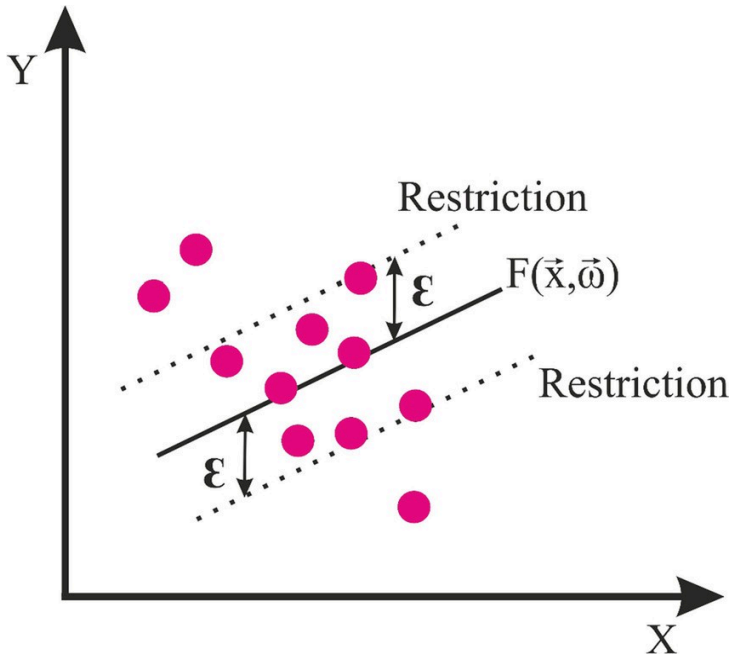
XGBRegressor

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=50,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.008, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=1000, n_jobs=None,
             num_parallel_tree=None, random_state=None, ...)
```

```
r2_score: 0.6845910934898634
mae: 2.355367792491714
mse: 10.429177680383743
rmse: 3.2294237381278634
```


train
true
prediction
Meantemp
Time Step
40
35
30
25
20
15
10
5
0
200
400
600
800
1000
1200
1400
1600

7. Mô hình SVR


Y
X
Restriction
Restriction
F(x⃗,ω⃗)
ε
ε

7.1 Khái niệm cơ bản

- **Hàm hồi quy**: Mục tiêu của SVR là tìm một hàm f(x) sao cho sai số giữa giá trị thực y và giá trị dự đoán f(x) là nhỏ nhất, nhưng với một mức độ chính xác có thể chấp nhận được.
- **Epsilon-insensitive loss**: SVR sử dụng một hàm mất mát đặc biệt gọi là ϵ\epsilonϵ-insensitive loss, nghĩa là nó bỏ qua các lỗi nhỏ hơn ngưỡng ϵ\epsilonϵ. Điều này giúp SVR ít nhạy cảm hơn với nhiễu trong dữ liệu.

7.2 Công thức toán học

Giả sử tập dữ liệu huấn luyện có dạng (xi,yi)(x_i, y_i)(xi,yi) với i=1,2,...,ni=1,2,...,n, trong đó xi là vector đầu vào và yi là giá trị đầu ra tương ứng.

SVR tìm một hàm f(x) dưới dạng:

f(x)=⟨w,x⟩+bf(x)

Trong đó:

- w là vector trọng số
- b là bias

Mục tiêu là tìm các tham số w và b sao cho hàm mất mát ϵ-insensitive loss là nhỏ nhất.

Hàm mất mát:

$$L_\epsilon(y, f(x)) = \begin{cases} 0 & \text{nếu } |y - f(x)| \leq \epsilon \\ |y - f(x)| - \epsilon & \text{nếu } |y - f(x)| > \epsilon \end{cases}$$

Trong đó, y là giá trị thực tế và f(x) là giá trị dự đoán.

Ưu điểm của SVR:

1. **Khả năng tổng quát hóa tốt**:
    - SVR tìm cách tối ưu hóa một biên độ cho phép, không chỉ cố gắng giảm thiểu lỗi huấn luyện mà còn giảm thiểu khả năng overfitting, do đó, nó có khả năng tổng quát hóa tốt trên các dữ liệu chưa nhìn thấy.
2. **Xử lý tốt với không gian đa chiều**:
    - SVR có thể xử lý tốt các bài toán với số chiều lớn (nhiều đặc trưng) nhờ sử dụng các kernel, đặc biệt là RBF kernel.

3. **Hiệu quả với dữ liệu ít**:
    - SVR có thể hoạt động hiệu quả với các bộ dữ liệu nhỏ, vì nó chỉ sử dụng một tập hợp con của các điểm dữ liệu (support vectors) để xác định mô hình hồi quy.
4. **Linh hoạt với các kernel**:
    - Khả năng sử dụng các kernel khác nhau (linear, polynomial, RBF, v.v.) giúp SVR có thể áp dụng cho nhiều loại dữ liệu với các mối quan hệ phức tạp khác nhau.
5. **Robust với nhiễu**:
    - Hàm mất mát ϵ-insensitive loss giúp SVR ít nhạy cảm với các nhiễu nhỏ trong dữ liệu, vì các lỗi nhỏ hơn ngưỡng ϵ\epsilonϵ sẽ bị bỏ qua.

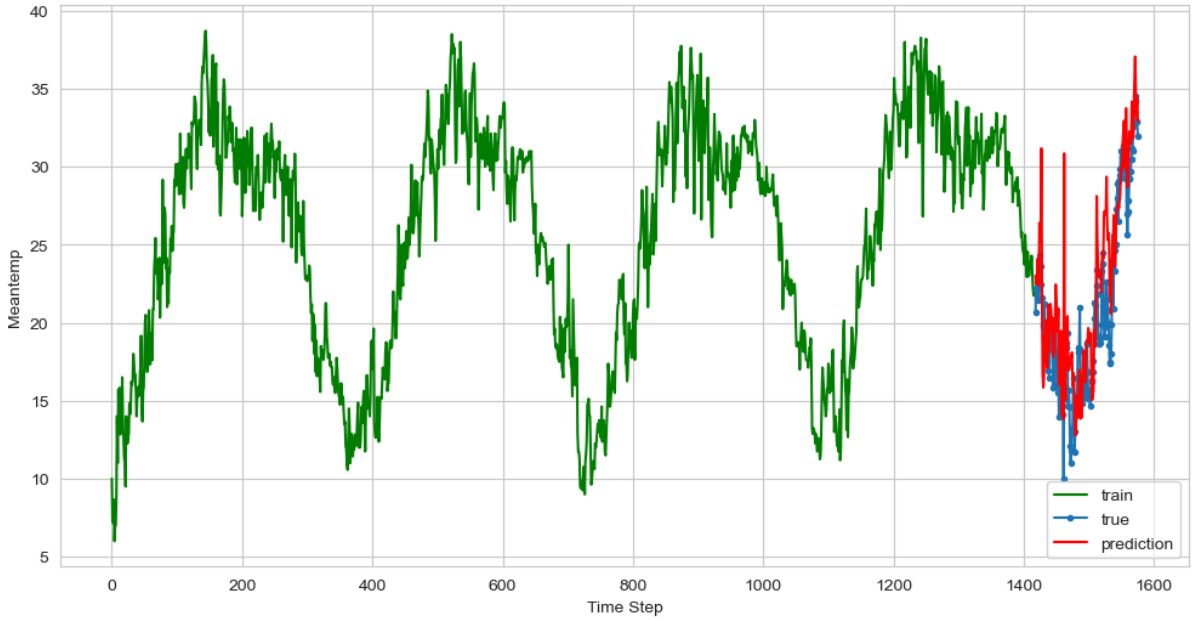
Nhược điểm SVR:

1. **Khó điều chỉnh tham số**:
    - SVR có nhiều tham số cần điều chỉnh, bao gồm CCC, ϵ\epsilonϵ, và các tham số của kernel. Việc tìm ra các giá trị tối ưu cho các tham số này có thể đòi hỏi nhiều thời gian và công sức.
2. **Hiệu suất tính toán**:
    - Đối với các bộ dữ liệu lớn, SVR có thể trở nên chậm và tốn nhiều bộ nhớ vì nó yêu cầu giải quyết một bài toán tối ưu hóa phức tạp. Thời gian huấn luyện tăng nhanh khi kích thước dữ liệu tăng.
3. **Khả năng mở rộng kém**:
    - Khi số lượng dữ liệu lớn, số lượng support vectors tăng, dẫn đến thời gian dự đoán và yêu cầu bộ nhớ tăng đáng kể.
4. **Khó hiểu và giải thích**:
    - Mô hình SVR, đặc biệt khi sử dụng các kernel phức tạp như RBF, khó giải thích hơn so với các mô hình hồi quy tuyến tính thông thường.
5. **Cần chuẩn hóa dữ liệu**:
    - SVR yêu cầu dữ liệu đầu vào phải được chuẩn hóa (scaling) để hoạt động hiệu quả, điều này có thể cần thêm bước xử lý dữ liệu.

Code và kết quả thông số đánh giá mô hình:

```
svr_model = SVR(C=26)
svr_model.fit(X_train, y_train)
y_pred = svr_model.predict(X_test)
eval_metrics(y_test, y_pred)
```

```
r2_score: 0.7056339543160548
mae: 2.3399853033514177
mse: 9.733383332379548
rmse: 3.1198370682424343
```



8. Tổng kết

	RNN	*LSTM*	*GRU*	*XGboost*	*SVR*
R2_Score	0.811641	0.830417	0.891224	0.684591	0.705633
MAE	0.173643	0.171675	0.135503	2.355367	2.339985
MSE	0.048572	0.043730	0.028049	10.429177	9.733383
RMSE	0.220390	0.209117	0.167480	3.229423	3.119837

Trong thời đại công nghệ thông tin phát triển mạnh mẽ, phân tích chuỗi thời gian đóng vai trò quan trọng trong việc dự báo xu hướng, phân tích dữ liệu và hỗ trợ ra quyết định. Ứng dụng phân tích chuỗi thời gian hiện nay được sử dụng rộng rãi trong các lĩnh vực như tài chính, kinh doanh, y tế và khí tượng học. Mục tiêu của đồ án này là xây dựng một mô hình học máy hiệu quả để phân tích và dự báo chuỗi thời gian.

Để thực hiện được điều này, em đã thu thập thành công dữ liệu chuỗi thời gian từ nhiều nguồn khác nhau. Sau đó, dữ liệu được xử lý và làm sạch, áp dụng các kỹ thuật phân tích chuỗi thời gian, và xây dựng các mô hình học máy như RNN (Recurrent Neural Network), LSTM (Long Short-Term Memory), GRU (Gated Recurrent Unit), XGBoost (Extreme Gradient Boosting), và SVR (Support Vector Regression). Kết quả cho thấy cả năm mô hình đều đạt được độ chính xác cao, tuy nhiên, mô hình GRUcho độ chính xác cao nhất với mức 89.12%.

Từ đó, nhóm em đã sử dụng mô hình LSTM để xây dựng một hệ thống dự báo chuỗi thời gian. Trong tương lai, em sẽ cố gắng cải tiến bằng cách tìm thêm các nguồn dữ liệu khác để có được thông tin đầy đủ và chi tiết hơn, áp dụng các mô hình học sâu tiên tiến hơn để nâng cao độ chính xác, và nghiên cứu sâu hơn về cách tối ưu tham số hiệu quả để tìm được mô hình tốt hơn nữa cho bài toán.

Như vậy, đã đạt được các mục tiêu đề ra ban đầu. Tuy nhiên, do thời gian có hạn và kinh nghiệm của bản thân còn hạn chế, nên đồ án không thể tránh khỏi các sai sót trong quá trình thực hiện. Do đó, em rất mong nhận được những ý kiến đóng góp quý báu của quý thầy cô để hoàn thiện hơn trong tương lai.

Một lần nữa chúng em xin chân thành cảm ơn!