

**TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM  
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG  
KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO CUỐI KỲ  
MÔN NHẬP MÔN HỌC MÁY**

# **DỰ ĐOÁN GIÁ CỔ PHIẾU BẰNG CÁC MÔ HÌNH HỌC MÁY**

*Người hướng dẫn:* **THẦY LÊ ANH CƯỜNG**

*Người thực hiện:* **CAO NGUYỄN KỲ DUYÊN – 51900491**

**HÀ THỊ THANH THẢO – 51900558**

**LÊ KIỀU ANH – 51800660**

**Lớp : 19050402, 18050402**

**Khoá : 22 & 23**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2022**

**TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM  
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG  
KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO CUỐI KỲ  
MÔN NHẬP MÔN HỌC MÁY**

# **DỰ ĐOÁN GIÁ CỔ PHIẾU BẰNG CÁC MÔ HÌNH HỌC MÁY**

*Người hướng dẫn:* **THẦY LÊ ANH CƯỜNG**

*Người thực hiện:* **CAO NGUYỄN KỲ DUYÊN – 51900491**

**HÀ THỊ THANH THẢO – 51900558**

**LÊ KIỀU ANH – 51800660**

**Lớp : 19050402, 18050402**

**Khoá : 22 & 23**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2022**

## **LỜI CẢM ƠN**

Nhóm 7 xin chân thành gửi lời cảm ơn đến thầy Lê Anh Cường đã giúp đỡ, hướng dẫn và dìu dắt chúng em trong quá trình học tập và tìm hiểu về bộ môn “Nhập môn học máy”. Nhờ như vậy, chúng em có thể thực hiện bài báo cáo này một cách tốt nhất và có thể đạt được một kết quả tốt nhất. Một lần nữa, nhóm 7 xin bày tỏ lòng biết ơn của mình đến với thầy Lê Anh Cường và chúc thầy sẽ luôn thành công trên con đường dạy học của mình.

## **ĐỒ ÁN ĐƯỢC HOÀN THÀNH TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

Tôi xin cam đoan đây là sản phẩm đồ án của riêng tôi và được sự hướng dẫn của Thầy Lê Anh Cường. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong đồ án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

**Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung đồ án của mình.** Trường đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

*TP. Hồ Chí Minh, ngày 23 tháng 11 năm 2022*

*Tác giả*

*(ký tên và ghi rõ họ tên)*

*Cao Nguyễn Kỳ Duyên*

*Hà Thị Thanh Thảo*

*Lê Kiều Anh*

## **PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN**

### **Phần xác nhận của GV hướng dẫn**

---

---

---

---

---

---

---

---

Tp. Hồ Chí Minh, ngày      tháng      năm  
(kí và ghi họ tên)

### **Phần đánh giá của GV chấm bài**

---

---

---

---

---

---

---

---

Tp. Hồ Chí Minh, ngày      tháng      năm  
(kí và ghi họ tên)

## TÓM TẮT

Trong bài báo cáo cuối kì này chúng em thực hiện phân tích bài toán và đưa ra một giải pháp dựa vào học máy để dự đoán giá cổ phiếu SHB trong tương lai với các mốc là 14 ngày và 7 tuần. Cụ thể chúng em thực hiện thu thập dữ liệu cổ phiếu SHB, phân tích bài toán xác định thông tin nào được sử dụng làm đầu vào để dự đoán giá tương lai và thực hiện các mô hình học máy RNN và MLP cùng với áp dụng các phương pháp xử lý overfitting rồi so sánh với mô hình không áp dụng overfitting, bên cạnh đó chúng em cũng sử dụng các tập đặc trưng khác nhau để tìm ra tập đặc trưng hiệu quả nhất ảnh hưởng đến kết quả dự đoán. Cuối cùng chúng em nghiên cứu lý thuyết và áp dụng mô hình deep learning Long short term memory (LSTM) vào bài toán của chúng em.

## MỤC LỤC

LỜI CẢM ƠN .....	i
PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN .....	iii
TÓM TẮT .....	iv
MỤC LỤC .....	1
DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ .....	4
CHƯƠNG 1 – GIỚI THIỆU .....	6
1.1 Giới thiệu về đề tài .....	6
1.2 Khái niệm về cổ phiếu .....	6
CHƯƠNG 2 – THU THẬP VÀ CHỌN LỌC DỮ LIỆU .....	9
2.1 Thu thập dữ liệu .....	9
2.1.1 Trang web sử dụng để thu thập .....	9
2.1.2 API sử dụng .....	9
2.2 Chọn lọc dữ liệu .....	12
CHƯƠNG 3 – XÂY DỰNG MÔ HÌNH HỌC MÁY .....	16
3.1 Mô hình học máy .....	16
3.1.1 Reccurent Neural Network .....	16
3.1.1.1 Khái niệm về RNN .....	16
3.1.1.2 Ứng dụng .....	17
3.1.1.3 Xây dựng và thực hiện dự đoán .....	18
3.1.1.4 Kết quả thu được và đánh giá .....	24
3.1.2 Multi-Layer Perceptron .....	27
3.1.2.1 Khái niệm về MLP .....	27
3.1.2.2 Ứng dụng .....	28
3.1.2.3 Xây dựng và thực hiện dự đoán .....	29
3.1.2.4 Kết quả thu được và đánh giá .....	35
3.1.3 So sánh các mô hình .....	39

3.1.3.1 Mô hình RNN .....	40
3.1.3.2 Mô hình MLP .....	41
3.1.3.3 Kết luận .....	43
3.2 Overfitting .....	44
3.2.1 Khái niệm về overfitting .....	44
3.2.2 Phương pháp tránh overfitting .....	47
3.2.2.1 Validation .....	48
3.2.2.2 Cross-validation .....	49
3.2.2.3 Early stopping .....	50
3.2.2.4 Dropout .....	51
3.2.2.5 Regularization .....	51
3.2.3 Áp dụng chống overfitting vào các mô hình .....	53
3.2.3.1 RNN .....	53
3.2.3.2 MLP .....	62
3.3 Tìm ra tập đặc trưng quan trọng .....	71
CHƯƠNG 4 – HỌC SÂU .....	76
4.1 Khái niệm về học sâu .....	76
4.2 So sánh học sâu với học máy .....	76
4.3 Mô hình LSTM .....	76
4.3.1 Khái niệm về LSTM .....	77
4.3.2 Cấu trúc của LSTM .....	77
4.3.3 Ứng dụng .....	80
4.3.4 Xây dựng mô hình .....	80
4.3.5 Kết quả thu được và đánh giá .....	88
TÀI LIỆU THAM KHẢO .....	92



## **DANH MỤC CHỮ VIẾT TẮT**

### **CÁC CHỮ VIẾT TẮT**

MLP	Multi-layer Perceptron
RNN	Recurrent Neural Network
LTSM	Long Short-term memory
SGD	Stochastic Gradient Descent
GD	Gradient Descent
kNN	K-Nearest Neighbors

## DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ

### DANH MỤC HÌNH VÀ BẢNG

<i>Hình 2.1.2a: API sử dụng .....</i>	10
<i>Hình 2.1.2b: Lấy API từ filter stock price.....</i>	10
<i>Hình 2.1.2c: Dữ liệu sau khi được crawl về .....</i>	12
<i>Hình 2.2: Đồ thị dữ liệu giao dịch của SHB theo ngày và tuần .....</i>	14
<i>Hình 3.1.1.1: Kiến trúc RNN.....</i>	16
<i>Hình 3.1.1.4a: Kết quả dự đoán giá cổ phiếu theo tập test khi dùng mô hình RNN .....</i>	25
<i>Hình 3.1.1.4b: Kết quả dự đoán giá cổ phiếu 14 ngày tiếp theo khi dùng mô hình RNN .....</i>	25
<i>Hình 3.1.1.4c: Kết quả dự đoán giá cổ phiếu theo tập test khi dùng mô hình RNN.....</i>	26
<i>Hình 3.1.1.4d: Kết quả dự đoán giá cổ phiếu 7 tuần tiếp theo khi dùng mô hình RNN .....</i>	26
<i>Hình 3.1.2.1 Multi-Layer Perceptron .....</i>	27
<i>Hình 3.1.2.4a: In dự đoán theo tập test của mô hình MLP .....</i>	36
<i>Hình 3.1.2.4b: Kết quả dự đoán theo tập test của mô hình MLP .....</i>	36
<i>Hình 3.1.2.4c: In dự đoán 14 ngày tiếp theo của mô hình MLP.....</i>	37
<i>Hình 3.1.2.4d: Kết quả dự đoán 14 ngày tiếp theo của mô hình MLP .....</i>	37
<i>Hình 3.1.2.4e: In dự đoán tập test của mô hình MLP.....</i>	38
<i>Hình 3.1.2.4f: Kết quả dự đoán theo tập test của mô hình MLP .....</i>	38
<i>Hình 3.1.2.4g: Kết quả dự đoán 7 tuần tiếp theo của mô hình MLP .....</i>	39
<i>Hình 3.1.2.4h: Kết quả dự đoán 7 tuần tiếp theo của mô hình MLP.....</i>	39
<i>Hình 3.1.3.1a: Huấn luyện mô hình RNN theo ngày .....</i>	40
<i>Hình 3.1.3.1b: Huấn luyện mô hình RNN theo tuần.....</i>	41
<i>Hình 3.1.3.2a: Huấn luyện mô hình MLP theo ngày .....</i>	42
<i>Hình 3.1.3.2b: Huấn luyện mô hình MLP theo tuần.....</i>	43
<i>Bảng 3.1.3.3: So sánh mô hình RNN và MLP .....</i>	44

<i>Hình 3.2.1a: Dấu hiệu của overfitting</i> .....	45
<i>Hình 3.2.1b: Ví dụ về underfitting, good fit</i> .....	46
<i>Hình 3.2.1c: Ví dụ về overfitting</i> .....	46
<i>Hình 3.2.2.1: Ảnh hưởng của độ phức tạp lên các loại error</i> .....	48
<i>Hình 3.2.2.2: Cross-validation</i> .....	50
<i>Hình 3.2.2.3: Early stopping</i> .....	51
<i>Hình 3.2.2.4: Dropout</i> .....	51
<i>Hình 3.2.3.1a: Tập test</i> .....	57
<i>Hình 3.2.3.1b: Dự đoán 14 ngày tiếp theo sử dụng chống overfitting cho mô hình RNN</i> .....	57
<i>Hình 3.2.3.1c: Tập test</i> .....	61
<i>Hình 3.2.3.1d: Dự đoán 7 tuần tiếp theo sử dụng chống overfitting cho mô hình RNN</i>	62
<i>Hình 3.2.3.2a: Tập test</i> .....	66
<i>Hình 3.2.3.2b: Dự đoán 14 ngày tiếp theo sử dụng chống overfitting cho mô hình MLP</i> .....	66
<i>Hình 3.2.3.2c: Tập test</i> .....	71
<i>Hình 3.2.3.2d: Dự đoán 7 tuần tiếp theo sử dụng chống overfitting cho mô hình MLP</i> .....	71
<i>Hình 4.3.5a: Dự đoán tập test</i> .....	88
<i>Hình 4.3.5c: Kết quả dự đoán 14 ngày tiếp theo</i> .....	89
<i>Hình 4.3.5d: Kết quả dự đoán 14 ngày tiếp theo</i> .....	89
<i>Hình 4.3.5e: In dự đoán theo tập test</i> .....	90
<i>Hình 4.3.5f: Kết quả dự đoán theo tập test</i> .....	90
<i>Hình 4.3.5g: In dự đoán 7 tuần tiếp theo</i> .....	90
<i>Hình 4.3.5h: Kết quả dự đoán 7 tuần tiếp theo</i> .....	91

## CHƯƠNG 1 – GIỚI THIỆU

### 1.1 Giới thiệu về đề tài

Thị trường chứng khoán là nơi các công ty phát hành cổ phiếu của họ để mở rộng hoạt động kinh doanh và các nhà đầu tư có thể mua/bán cổ phiếu cho nhau ở mức giá cụ thể. Các nhà đầu tư trên khắp thế giới có thể mua cổ phiếu của một công ty và được hưởng cổ tức hàng năm cho cổ phiếu của họ. Họ cũng có thể bán cổ phiếu của mình bất cứ lúc nào và có thể kiếm lợi nhuận bằng cách bán với giá cao hơn giá mua. Thị trường chứng khoán đang trở thành một địa điểm đầu tư quan trọng và quy mô của thị trường đang tăng lên mỗi ngày. Mặc dù đầu tư vào thị trường chứng khoán có vẻ sinh lợi, nhưng việc dự đoán biến động cổ phiếu trên thị trường tài chính cạnh tranh là một thách thức, ngay cả đối với các nhà giao dịch và chuyên gia chứng khoán giàu kinh nghiệm. Tuy nhiên, cổ phiếu Lý thuyết dự báo giá thường gây tranh cãi: trong vòng một phần của giây, giá của một cổ phiếu có thể dao động mạnh như vậy phương pháp khác nhau trong vài thập kỷ qua. Tuy nhiên, giờ đây, với sự ra đời của trí tuệ nhân tạo, việc tự động dự đoán thị trường chứng khoán thông qua dữ liệu lớn và khả năng tính toán nâng cao đã trở nên khả thi. Trong bài báo cáo này, nhóm thực hiện đề tài dự đoán giá cổ phiếu SHB bằng các mô hình học máy theo các mốc thời gian 14 ngày đến 7 tuần.

### 1.2 Khái niệm về cổ phiếu

Cổ phiếu là chứng chỉ do công ty cổ phần phát hành, bút toán ghi sổ hoặc dữ liệu điện tử xác nhận quyền sở hữu một hoặc một số cổ phần của công ty đó (theo quy định tại khoản 1, điều 121, luật doanh nghiệp năm 2020 hiện đang áp dụng tại Việt Nam). Đồng thời, cổ phiếu là một loại hàng hoá được lưu thông trên thị trường, nói cách khác cổ phiếu được chuyển nhượng từ người này sang người khác, cổ phiếu có thể được dùng làm tài sản thừa kế, tài sản thế chấp, cầm cố trong các quan hệ tín dụng.

Thông thường hiện nay các công ty cổ phần thường phát hành hai dạng cổ phiếu: Cổ phiếu thường (common stock) và cổ phiếu ưu đãi (preferred stock). Cổ phiếu có các nội dung sau: tên, trụ sở công ty, số và ngày cấp giấy chứng nhận đăng ký kinh

doanh; số lượng cổ phần và loại cổ phần được biểu hiện bằng cổ phiếu, mệnh giá mỗi cổ phần và tổng mệnh giá số cổ phần ghi trên cổ phiếu; tên cổ đông (đối với cổ phiếu có ghi tên); tóm tắt về thủ tục chuyển nhượng cổ phần; chữ ký mẫu của người đại diện theo pháp luật và dấu của công ty; số đăng ký tại sở đăng ký cổ đông của công ty và ngày phát hành cổ phiếu; các nội dung khác liên quan tới quyền và nghĩa vụ của cổ đông (đối với cổ phiếu của cổ phần ưu đãi).

Cổ phiếu được đánh giá qua các chỉ số sau:

- *Chỉ số P/E - Price to Earning*: Là chỉ số đánh giá mối quan hệ giữa giá thị trường của cổ phiếu với thu nhập trên một cổ phiếu. Nếu chỉ số P/E thấp chứng tỏ cổ phiếu đang bị định giá thấp, có lợi cho nhà đầu tư.
- *Chỉ số EPS - Earning per Share*: Thu nhập trên mỗi cổ phần hay lợi nhuận sau thuế trên mỗi cổ phần sẽ nhận được sau khi công ty đã chia hết tiền lãi cho cổ đông. EPS cao chứng tỏ công ty đang có tốc độ phát triển tốt, ổn định.
- *Chỉ số P/B - Price to Book Value Ratio*: Tỷ lệ giá trên sổ sách. Chỉ số này dùng để so sánh giá trị thực tế của một cổ phiếu so với giá trị ghi sổ của cổ phiếu đó. Chỉ số P/B càng thấp chứng tỏ nhà đầu tư đang trả ít hơn so với giá trị sổ sách ghi nhận.
- *Chỉ số DPR - Tỷ lệ chi trả cổ tức*: Thể hiện mức chi trả cổ tức cho các cổ đông so với mức mà công ty kiếm được. Số tiền không được trả cho các cổ đông được công ty giữ lại để tái đầu tư tăng trưởng, trả nợ hoặc thêm vào lợi nhuận giữ lại. Chỉ số DPR cao cho thấy công ty đang phát triển và tạo ra lợi nhuận tốt.

Giá cổ phiếu là mức giá của cổ phiếu tại một thời điểm nhất định, có nghĩa là số tiền mà nhà đầu tư cần bỏ ra để sở hữu cổ phiếu đang giao dịch trên thị trường. Giá cổ phiếu là yếu tố quan trọng mà nhà đầu tư quan tâm khi quyết định mua, bán cổ phiếu.

Giá tham chiếu là mức giá cơ sở để tính giới hạn giá cổ phiếu giao động trong phiên giao dịch, dựa vào đây để xác định giá sàn và giá dịch của ngày giao dịch hiện tại. Trong đó, giá tham chiếu ở các sàn giao dịch sẽ khác nhau.

Giá sàn là mức giá thấp nhất hay mức giá kịch sàn mà nhà đầu tư có thể đặt lệnh mua/bán trong ngày giao dịch. Cách tính giá sàn như sau:  $\text{Giá sàn} = \text{Giá tham chiếu} \times (100\% + \text{Biên độ dao động})$

Giá trần là mức giá cao nhất hay mức giá kịch trần mà nhà đầu tư có thể đặt lệnh mua/bán trong ngày giao dịch. Cách tính giá trần như sau:  $\text{Giá trần} = \text{Giá tham chiếu} \times (100\% - \text{Biên độ dao động})$ .

Phát hành cổ phiếu được xem là một hoạt động kêu gọi vốn đối với những công ty cổ phần. Vì vậy, cổ phiếu chính là chứng chỉ được công ty phát hành, bút toán ghi sổ hay là các dữ liệu điện tử nhằm xác nhận quyền sở hữu số cổ phần của công ty đó. Việc phát hành cổ phiếu giúp cho công ty có thể kêu gọi được nguồn vốn từ những nhà đầu tư bên ngoài.


## CHƯƠNG 2 – THU THẬP VÀ CHỌN LỌC DỮ LIỆU

### 2.1 Thu thập dữ liệu

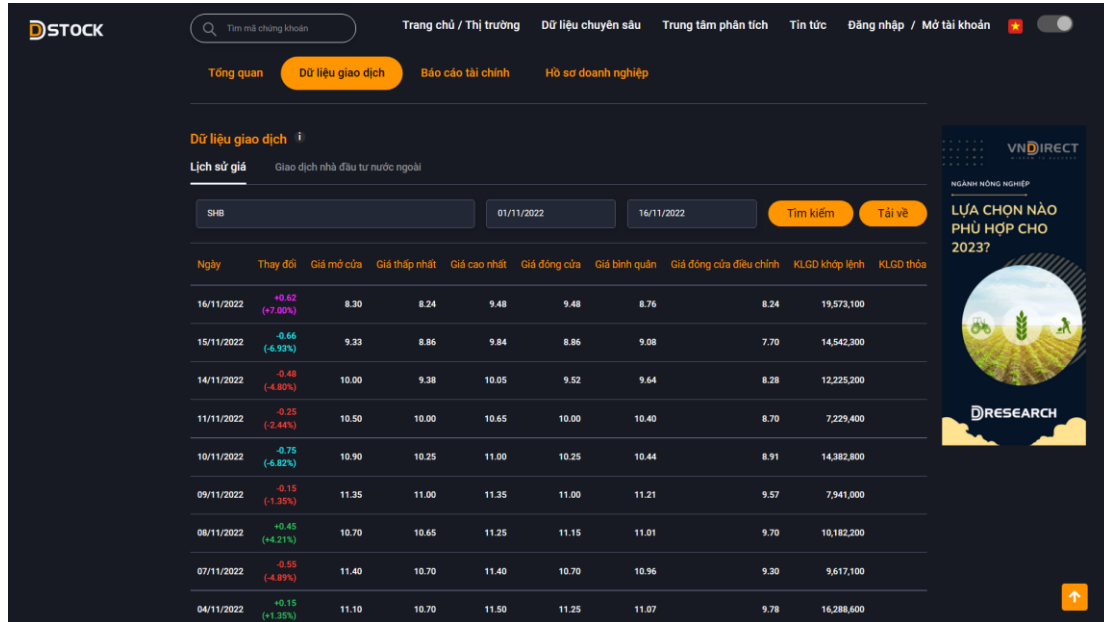
#### 2.1.1 Trang web sử dụng để thu thập

 *Import các thư viện sử dụng trong bài:*

```
import pandas as pd
import requests
from datetime import datetime, timedelta
import json
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
from sklearn.model_selection import train_test_split
from numpy import hstack
from keras.models import Sequential
from keras.layers import Input
from keras.layers import Dense
from keras.layers import SimpleRNN, Dropout, LSTM
from keras.callbacks import EarlyStopping
from keras.regularizers import l2
import numpy as np
```

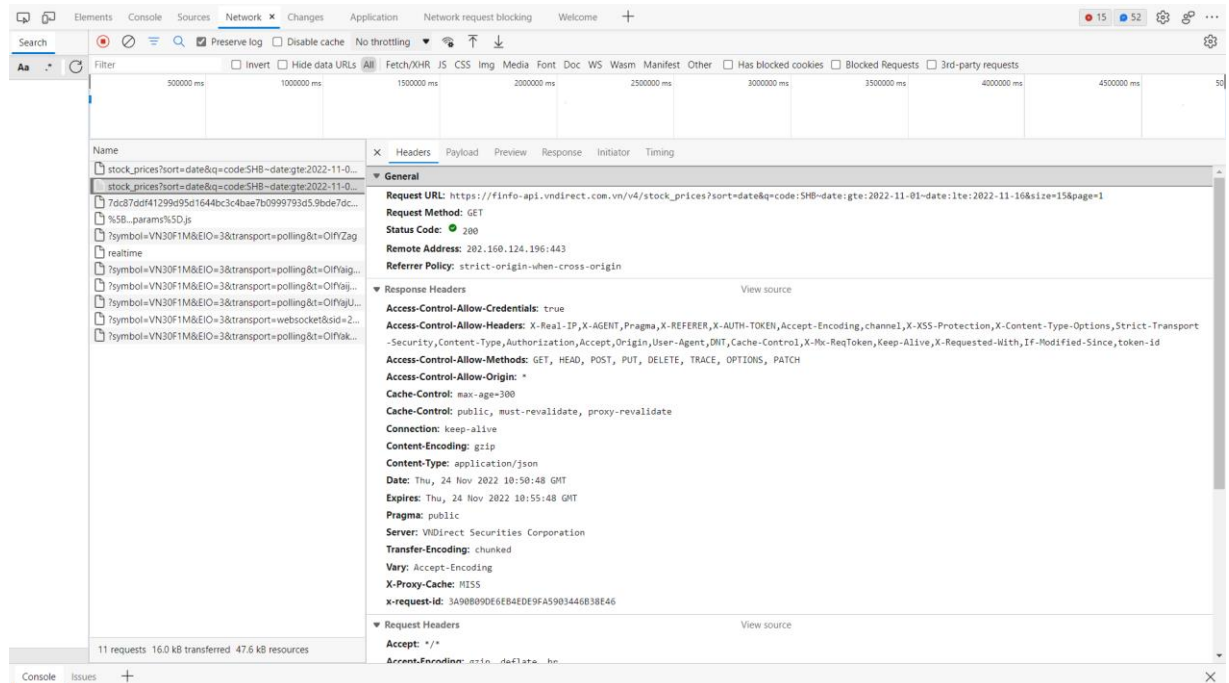
 *Trang web sử dụng để lấy dữ liệu: <https://dstock.vndirect.com.vn/>*

#### 2.1.2 API sử dụng



Hình 2.1.2a: API sử dụng

Ở mục dữ liệu giao dịch của một cổ phiếu, chọn cổ phiếu SHB. Chọn ngày tháng năm để làm mốc lấy thông tin giao dịch và bấm tìm kiếm sẽ cho ra API của trang.



Hình 2.1.2b: Lấy API từ filter stock price



✚ **API lấy từ trang web với các thông tin như sau:**

*[https://finfoapi.vndirect.com.vn/v4/stock\\_prices?sort=date:asc&q=code:SHB](https://finfoapi.vndirect.com.vn/v4/stock_prices?sort=date:asc&q=code:SHB~date:gte:2022-11-01~date:lte:2022-11-16&size=15&page=1)*

*[B~date:gte:2022-11-01~date:lte:2022-11-16&size=15&page=1](https://finfoapi.vndirect.com.vn/v4/stock_prices?sort=date:asc&q=code:SHB~date:gte:2022-11-01~date:lte:2022-11-16&size=15&page=1)*

**sort = date:asc** là sắp xếp dữ liệu theo date tăng dần

**q=code** : SHB là lấy dữ liệu giao dịch theo mã cổ phiếu SHB

**date** : gồm 2 element

*gte:2022-11-01* là lớn hơn hoặc bằng ngày 2022-11-01

*lte:2022-11-16* là nhỏ hơn hoặc bằng ngày 2022-11-16

size=15 nhận về 15 item trên 1 trang

page=1 trang 1

```
stockSymbol = "SHB" #Mã ngân hàng muốn lấy thông tin

stockApiUrl = f"https://finfo-
api.vndirect.com.vn/v4/stock_prices?sort=date:asc&q=code:{stockSymbol}&s
ize=99999" #Đường dẫn này sẽ lấy dữ liệu sắp xếp theo thứ tự ngày tăng d
ần
res = requests.get(stockApiUrl, headers={'User-
agent': 'Mozilla/5.0'}) #Gọi API lấy dữ liệu về
stockData = json.loads(res.text) #Chuyển sang json và gán vào biến stoc
kData

print(json.dumps(stockData, indent=4)) #In ra dữ liệu json thu về
```

Sử dụng thư viện *request* để gọi đến api của vndirect.

Dữ liệu trả về có dạng json chuỗi, tiếp dụng dùng thư viện *json* để biến nó thành dictionary để dễ dàng truy xuất dữ liệu hơn.

✚ **Ta sẽ tiến hành chỉnh sửa lại API để crawl dữ liệu :**

*[https://finfoapi.vndirect.com.vn/v4/stock\\_prices?sort=date:asc&q=code:SHB](https://finfoapi.vndirect.com.vn/v4/stock_prices?sort=date:asc&q=code:SHB)*

*B size=99999*

✚ **Ta sẽ lấy tất cả dữ liệu của SHB theo thứ tự ngày tăng dần:**

```

1 // 20221124203645
2 // https://finfo-api.vndirect.com.vn/v4/stock_prices?sort=date:asc&q=code:SHB~date:gte:2022-11-01~date:lte:2022-11-16&size=15&page=1
3
4 {
5   "data": [
6     {
7       "code": "SHB",
8       "date": "2022-11-01",
9       "time": "15:11:01",
10      "floor": "HOSE",
11      "type": "STOCK",
12      "basicPrice": 11.45,
13      "ceilingPrice": 12.25,
14      "floorPrice": 10.65,
15      "open": 11.6,
16      "high": 11.95,
17      "low": 11.55,
18      "close": 11.6,
19      "average": 11.73,
20      "adOpen": 10.086,
21      "adHigh": 10.391,
22      "adLow": 10.043,
23      "adClose": 10.086,
24      "adAverage": 10.199,
25      "nmVolume": 12823500,
26      "nmValue": 150434885000,
27      "ptVolume": 0.0,
28      "ptValue": 0.0,
29      "change": 0.15,
30      "adChange": 0.1304,
31      "pctChange": 1.31
32    },
33    {
34      "code": "SHB",
35      "date": "2022-11-02",
36      "time": "15:11:01",
37      "floor": "HOSE",
38      "type": "STOCK",
39      "basicPrice": 11.45,
40      "ceilingPrice": 12.25,
41      "floorPrice": 10.65,
42      "open": 11.6,
43      "high": 11.95,
44      "low": 11.55,
45      "close": 11.6,
46      "average": 11.73,
47      "adOpen": 10.086,
48      "adHigh": 10.391,
49      "adLow": 10.043,
50      "adClose": 10.086,
51      "adAverage": 10.199,
52      "nmVolume": 12823500,
53      "nmValue": 150434885000,
54      "ptVolume": 0.0,
55      "ptValue": 0.0,
56      "change": 0.15,
57      "adChange": 0.1304,
58      "pctChange": 1.31
59    }
60  ]
61 }

```

Hình 2.1.2c: Dữ liệu sau khi được crawl về

## 2.2 Chọn lọc dữ liệu

### ✚ Xác định các feature muốn dùng:

Chúng ta có thể thấy, sau khi crawl dữ liệu về thì có rất nhiều element không cần thiết. Vì vậy, chúng ta cần chọn lọc các *feature* cần thiết và đưa nó vào *dataframe*.

```

features = ['date', 'open', 'ceilingPrice', 'floorPrice', 'close'] #Mảng chứa các features cần lấy
dataframe = pd.DataFrame(columns=features) #Tạo dataframe
dataframe

```

Các *feature* sử dụng cho mô hình bao gồm ngày tháng năm (*date*), giá mở cửa (*open*), giá trần (*ceilingPrice*), giá sàn (*floorPrice*), giá đóng cửa (*close*). Sau đó, tạo 1 *dataframe* với các cột là tên những *feature* muốn sử dụng cho mô hình.

```

for data in stockData['data']: #Chạy vòng lặp dữ liệu giao dịch
    dataframe.loc[len(dataframe.index)] = [data[value] for value in features] #Tạo mảng với các dữ liệu cần lấy theo mảng features và thêm vào dataframe

```

```
dataframe['date'] = pd.to_datetime(dataframe['date']) #Biến cột date từ dạng string thành dạng datetime
dataframe.to_csv(f"{stockSymbol}_dataset_date.csv", index=False) #Lưu dataframe thành file csv
dataframe #In dataframe
```

Chạy vòng lặp trong *stockData['data']*, lấy ra các dữ liệu, trong đó có những giá trị như *date*, *close*, *open*,... ta sẽ lấy ra theo mảng *features* để khai báo rồi thêm mảng đó vào trong *dataframe*.

Sau khi kết thúc vòng lặp, chuyển cột *date* từ dạng *string* sang dạng *datetime* để có thể sử dụng để vẽ đồ thị. Ngoài ra, lưu *dataframe* đó thành 1 file csv với tên *SHB\_dataset\_date.csv*.

```
dataframe_w = dataframe.groupby([pd.Grouper(key='date', freq='W-MON')], as_index=True).mean().reset_index()
dataframe_w = dataframe_w.fillna(method='ffill')
dataframe_w.to_csv(f"{stockSymbol}_dataset_week.csv", index=False) #Lưu dataframe_w thành file csv
dataframe_w #In dataframe_w
```

Theo yêu cầu bài là dự đoán ngày và tuần, ta sẽ dựa trên *dataframe* trên, để thực hiện *group* lại theo cột ngày, lấy thứ 2 làm mốc sau đó tính trung bình những dữ liệu đó.

Sao khi *group* lại sẽ có những tuần có giá trị NaN. Giá trị NaN có thể được hiểu như giá trị *missing* trong bộ dữ liệu. Cho nên để khắc phục ta thực hiện *fillna* theo phương pháp *forward fill*, nghĩa là nếu như cell bị NaN, nó sẽ lấy giá trị cell trước nó điền vào. Cuối cùng lưu lại thành 1 file csv với tên *SHB\_dataset\_week.csv*.

```
fig, axs = plt.subplots(1,2, figsize = (20,10)) #Tạo plot
axs[0].set_title(f"Dữ liệu giao dịch của {stockSymbol} theo ngày") #Tiêu đề cho plot 1
axs[1].set_title(f"Dữ liệu giao dịch của {stockSymbol} theo tuần") #Tiêu đề cho plot 2

dates = dataframe['date'] #Lấy cột date từ dataframe theo ngày
weeks = dataframe_w['date'] #Lấy cột date từ dataframe theo tuần
for feature in features[1:]:
```

```

    axs[0].plot(dates, dataframe[feature]) #Vẽ đường tương ứng với các f
eature được chọn trừ date theo ngày
    axs[1].plot(weeks, dataframe_w[feature]) #Vẽ đường tương ứng với các
feature được chọn trừ date theo tuần

plt.legend(features[1:]) #Tạo chú thích
plt.tight_layout() #Giảm lề
plt.show() #In hình

```

Chúng ta sẽ tiến hành trực quan hóa 2 dữ liệu, tạo 1 *figure* với 2 *plot* để vẽ đồ thị. Đặt tên cho 2 *plot* đó bằng *setTitle*. Lấy cột *date* từ *dataframe* ngày và *dataframe* tuần. Tiếp theo, cho chạy vòng lặp cho mảng *features[1:]*, lấy hết các *features* trừ *date*. Thực hiện vẽ những *feature* đó và tạo chú thích cho sơ đồ.



Hình 2.2: Đồ thị dữ liệu giao dịch của SHB theo ngày và tuần

### Các hàm hỗ trợ:

```

# Hàm này sẽ tách dataframe thành X, y
# Trong đó
# mỗi phần tử X là chứa {n_steps_in} ngày trước
# mỗi phần tử X là chứa ngày càng dự đoán
def split_sequences(sequences, n_steps_in):
    X, y = list(), list()
    for i in range(len(sequences)):
        end_ix = i + n_steps_in

```

```

    out_end_ix = end_ix + 1
    if out_end_ix > len(sequences):
        break

    seq_x, seq_y = sequences[i:end_ix, :], sequences[end_ix:out_end_ix, :]
    X.append(seq_x)
    y.append(seq_y)
    return np.array(X), np.array(y)

# Hàm này sẽ thêm vào mảng dates {n} phần tử, {distance} mỗi phần tử thêm vào cách bao nhiêu ngày so với phần tử trước đó
# Ngoài ra nếu ngày đó là thứ 7 hay chủ nhật thì không được thêm vào dates
def dates_add(dates, n, distance=1):
    target = len(dates) + n
    tmp_date = dates.iloc[-1]
    while( len(dates) < target):
        tmp_date = tmp_date + pd.DateOffset(days=distance)

        if (tmp_date.weekday() < 5):
            dates = dates.append(pd.Series([tmp_date]))
    return dates

```

Hàm *split\_sequences()* sẽ nhận *sequences* và *n\_steps\_in* để tạo tập dữ liệu, từ đó dùng để *train* mô hình là *X* và *y*. Nó sẽ lặp theo *i* là từ 0 đến chiều dài của *sequences* (số row của dataset). Tiếp đó sẽ lấy từ *i* đến *i + n\_steps\_in* phần tử làm *X* và lấy *i + n\_steps\_in + 1* làm *y* và cứ thế lặp cho đến hết *sequences* sẽ trả về tập *X* và *y*.

Hàm *dates\_add()* sẽ nhận vào mảng *dates*, *n* là số ngày muốn thêm, *distance* là khoảng cách ngày thêm. Ví dụ như: 1 ngày 2 ngày. Sử dụng vòng lặp *while*, nếu chiều dài của mảng *dates* nhỏ hơn *target* (là chiều dài mảng *dates* ban đầu + *n* ngày thêm). Sau đó, sẽ lấy phần tử cuối cùng trong mảng thêm *distance* ngày vào và kiểm tra nếu như là thứ 7, chủ nhật thì sẽ không thêm vào, ngược lại thì được thêm. Bởi vì cỗ phiếu sẽ không giao dịch vào những ngày cuối tuần.

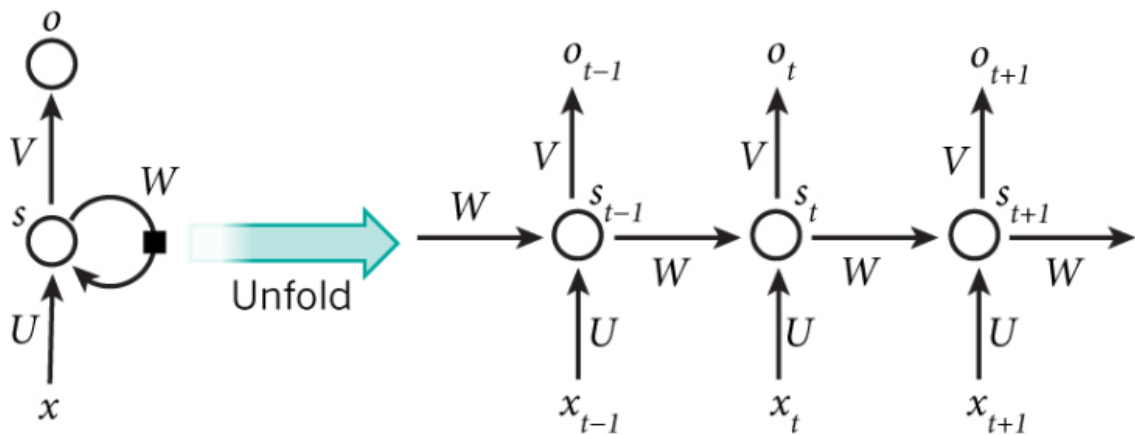
## CHƯƠNG 3 – XÂY DỰNG MÔ HÌNH HỌC MÁY

### 3.1 Mô hình học máy

#### 3.1.1 Reccurent Neural Network

##### 3.1.1.1 Khái niệm về RNN

Reccurent Neural Network (RNN) là một mô hình rất quan trọng trong lĩnh vực xử lý ngôn ngữ tự nhiên. Mô hình RNN ra đời với ý tưởng chính của là sử dụng và xử lý các chuỗi các thông tin. Trong các mạng neural truyền thống thì tất cả các đầu vào và đầu ra độc lập với nhau. Nghĩa là chúng không có liên kết thành chuỗi với nhau. Thế nhưng các mô hình này không phù hợp trong khá nhiều bài toán. Mô hình RNN được gọi là hồi quy bởi vì chúng thực hiện cùng một tác vụ cho mọi phần tử của một chuỗi với đầu ra phụ thuộc vào các phép tính trước đó. Chúng ta có thể hiểu là RNN có khả năng ghi nhớ các thông tin được tính toán trước đó. Theo lý thuyết, RNN có thể dùng được thông tin của một văn bản rất dài, tuy nhiên trên thực tế thì nó chỉ có thể nhớ được một vài bước trước đó mà thôi. Một mạng RNN có dạng cơ bản như sau:



Hình 3.1.1.1: Kiến trúc RNN

Hình trên mô tả sự triển khai nội dung của một mô hình RNN. Ở đây có thể hiểu đơn giản là ta vẽ ra một mạng neural chuỗi tuần tự. Khi đó sự tính toán ở bên trong RNN được thực hiện như sau:

- $X_t$  là đầu vào tại bước  $t$ . Cho  $x_t$  là một vector onehot tương ứng.
- $S_t$  chính là trạng thái ẩn tại bước  $t$ . Nó chính là bộ nhớ của mạng.  $s_t$  được tính toán dựa theo tất cả các trạng thái ẩn phía trước và trạng thái đầu vào tại bước đó:  $s_t = f(Ux_t + Ws_{t-1})$ . Hàm  $f$  thông thường là một hàm phi tuyến tính như tang hyperbolic (tanh) hoặc ReLu. Để tính toán cho phần tử ẩn đầu tiên ta cần khởi tạo thêm  $s_{-1}$ , thường thì giá trị khởi tạo sẽ bằng 0.
- $O_t$  chính là đầu ra tại bước  $t$ . Nếu muốn dự đoán từ kế tiếp có thể xuất hiện trong câu thì  $o_t$  chính là một vector xác suất các từ trong danh sách từ vựng:  

$$o_t = \text{softmax}(Vs_t).$$

***Ưu điểm của mô hình RNN:***

- Có khả năng xử lý đầu vào với độ dài nào lớn
- Kích thước mô hình không tăng theo kích thước dữ liệu đầu vào
- Có thể sử dụng các thông tin cũ trong quá trình tính toán
- Trọng số được chia sẻ trong suốt thời gian thực hiện mô hình

***Nhược điểm của mô hình RNN:***

- Thời gian tính toán chậm
- Khó để truy xuất thông tin từ khoảng thời gian dài trước đây
- Không thể xem xét bất kì trạng thái đầu vào sau này cho trạng thái hiện tại

### 3.1.1.2 Ứng dụng

Mô hình RNN có thể áp dụng vào nhiều bài toán như: dịch máy, mô hình hóa ngôn ngữ và tự sinh từ, chuyển giọng nói thành văn bản, mô tả hình ảnh,...

▪ *Dịch máy*

Dịch máy nghĩa là ta cần phải dịch một chuỗi ngôn ngữ nguồn sang một ngôn ngữ đích (ví dụ như dịch tiếng Anh sang tiếng Việt Nam). Nếu nghĩ theo cách đơn giản thì có lẽ bạn sẽ nghĩ nó thật dễ dàng đúng không, việc ta cần làm là chỉ cần ánh xạ từ đó đến nghĩa của nó trong database rồi ghép chúng lại với nhau. Nhưng thực tế mọi thứ không đơn giản như vậy bởi vì mỗi từ khi đi cùng một từ khác thì nghĩa của nó sẽ thay đổi, và

một từ có thể có rất nhiều nghĩa trong từng hoàn cảnh khác nhau, vậy nên đó là lý do ta cần dùng đến RNN để tạo ra một câu dịch sát cả về nghĩa và mượt trong văn phong. Để làm được điều đó thì ta cần phải xem xét và xử lý qua tất cả các chuỗi đầu vào.

- *Mô hình hóa ngôn ngữ và sinh văn bản.*

Mô hình hóa ngôn ngữ tự nhiên cho phép ta dự đoán được xác suất của một từ sẽ xuất hiện sau một chuỗi các từ ngữ trước nó. Một điều thú vị của việc có thể dự đoán được từ kế tiếp chính là ta có thể xây dựng được một mô hình tự sinh từ cho phép máy tính có thể tự tạo ra các văn bản mới từ tập hợp mẫu và xác suất đầu ra của mỗi từ. Do đó, tùy thuộc vào mô hình ngôn ngữ mà ta có thể tạo ra được nhiều văn bản khác nhau. Trong mô hình ngôn ngữ, đầu vào thông thường là một chuỗi các từ và đầu ra là một chuỗi các từ dự đoán. Khi huấn luyện mạng neural, ta sẽ gán  $o_t = x_{t+1}$  bởi vì ta muốn đầu ra tại bước  $t$  chính là từ tiếp theo của câu.

- *Nhận dạng giọng nói*

Nhận dạng giọng nói chính là đưa vào chuỗi tín hiệu âm thanh sau đó ta có thể dự đoán được chuỗi các đoạn ngữ âm đi kèm với xác suất của chúng.

- *Mô tả hình ảnh*

Trong ứng dụng này mạng convolution neural network thường được sử dụng để phát hiện ra các đối tượng có trong ảnh sau đó RNN sẽ sinh ra các câu có nghĩa mô tả bức ảnh đó.

### 3.1.1.3 Xây dựng và thực hiện dự đoán

#### *Dữ liệu ngày :*

```
n_steps_in = 14 #Lấy {n_steps_in} ngày trước để dự đoán ngày hôm nay

dataset = dataframe[features[1:]].to_numpy() #Chuyển dataframe thành mảng
X, y = split_sequences(dataset, n_steps_in) #Chia dataset thành X và y theo {n_steps_in}

n_input = X.shape[1] * X.shape[2] #Tính chiều dài input
n_output = y.shape[1] * y.shape[2] #Tính chiều dài output
```



```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=len(X)-n_steps_in, shuffle = False)
#Chia dữ liệu tập train và test trong đó test chứa {n_steps_in} để thực hiện dự đoán
```

Ta sẽ thực hiện tiền xử lý dữ liệu như tạo  $X$  và  $y$  bằng hàm *split\_sequences()*. Tiếp đó tính chiều dài của input và output lần lượt gán vào  $n\_input$ ,  $n\_ouput$  để sử dụng cho quá trình xây dựng mô hình. Cuối cùng sử dụng *train\_test\_split* với tham số *shuffle* là *false* để thực hiện chia dữ liệu thành tập test và tập train.

```
model = Sequential() #Khởi tạo model
model.add(SimpleRNN(20)) #Thêm lớp SimpleRNN
model.add(Dense(n_output)) #Thêm lớp Dense với n_output là số lượng tham số đầu ra
model.compile(optimizer='adam', loss='mae')
```

Khi đã xong quá trình tiền xử lý dữ liệu, ta sẽ tiếp tục tiến hành xây dựng mô hình sử dụng *Sequential*, với lớp đầu tiên là *SimpleRNN* với số *unit* xử lý của lớp đó là 20. Lớp *Dense* sẽ được dùng để trả về kết quả, nó sẽ nhận tham số  $n\_ouput$  và trả về số lượng output tương ứng. Sau đó, ta tiến hành *compile* mô hình với *optimizer* (thuật toán tối ưu) là 'adam' và hàm *loss* (hàm mất mát) là 'mae' (mean absolute error).

```
start_time = datetime.now()
model.fit(X_train, y_train, epochs=1000, shuffle=False, verbose=2)
print(f"Thời gian train: {(datetime.now()-start_time).total_seconds()} giây")
```

Thực hiện *train* mô hình, với *epochs* là 1000, *shuffle* là false. Sau khi train xong sẽ in ra thời gian train mô hình.

```
test_predict = model.predict(X_test, verbose=0)
test_predict
```

Thực hiện dự đoán mô hình thông qua tập test.

```
fig, axs = plt.subplots(4,figsize = (20,10)) #Tạo plot
```

```

dates = dataframe['date'][-(n_steps_in):] #Lấy cột date
titles = ['Giá mở cửa', 'Giá trần', 'Giá sàn', 'Giá đóng cửa'] #Khai báo
mảng tiêu đề

for idx, feature in enumerate(features[1:]): #Lặp vẽ 4 tương ứng với các
feature được chọn trừ date
    axs[idx].plot(dates, dataframe[feature] [-
(n_steps_in):]) #Vẽ đường thực tế
    axs[idx].plot(dates, test_predict[:,idx]) #Vẽ đường dự đoán
    axs[idx].legend([f'true {feature}', f'predict {feature}']) #Tạo chú th
ích
    axs[idx].set_title(titles[idx]) #Tạo tiêu đề

plt.tight_layout() #Giảm lề
plt.show() #In hình

```

Sau khi dự đoán tập test, ta thực hiện trực quan hóa chúng để quan sát dự đoán của mô hình có chính xác không. Tạo *figure* với 4 *plot*, sau đó lấy cột *date* ứng với chiều dài của tập *test*. Tạo mảng *titles* chứa tiêu đề các *plot*.

Tiếp theo, ta thực hiện lặp *enumerate* cho *features* ngoại trừ *date* ra. Vẽ đường thực tế và đường dự đoán. Mỗi vòng lặp sẽ vẽ *plot* cho 1 *features* và thứ tự *plot* ứng với *idx* sau đó set tiêu đề cho *plot*.

```

predicts = list(dataframe[features[1:]][-14:].to_numpy())
#Lấy dữ liệu 14 ngày trước

while(len(predicts) < 14+14): #Thực hiện vào lặp dự đoán 14 ngày tiếp th
eo
    X_predicts = np.array([predicts[-
14:]] #Lấy 14 ngày trước từ mảng predicts
    predict = model.predict(X_predicts, verbose=0) #Thực hiện dự đoán
    predicts.append(predict[0]) #Thêm dự đoán vào mảng predicts để tiếp tậ
c dự đoán

predicts = np.array(predicts) #Chuyển predicts lại thành np array
predicts

```

Thực hiện dự đoán 14 ngày trong tương lai bằng cách sử dụng 14 ngày dữ liệu có sẵn (14 ngày này ứng với *n\_steps\_in*).

Ta sẽ tạo mảng *X\_predicts* chứa dữ liệu 14 ngày trước đó và thực hiện dự đoán sau đó rồi thêm lại vào mảng *predict* và lặp lại như thế 14 lần.

```
fig, axs = plt.subplots(4, figsize = (20,10)) #Tạo plot

dates = dataframe['date'][-(n_steps_in):] #Lấy cột date
dates = dates_add(dates, 14) #Thêm vào mảng date 14 ngày
titles = ['Giá mở cửa', 'Giá trần', 'Giá sàn', 'Giá đóng cửa'] #Khai báo
mảng tiêu đề

for idx, feature in enumerate(features[1:]): #Lặp vẽ 4 tương ứng với các
feature được chọn trừ date
    axs[idx].plot(dates[:14], predicts[:14,idx]) #Vẽ đường thực tế
    axs[idx].plot(dates, np.concatenate((test_predict[:,idx], predicts[14:
,idx]))) #Vẽ đường dự đoán
    axs[idx].legend([f'true {feature}', f'predict {feature}']) #Tạo chú th
ích
    axs[idx].set_title(titles[idx]) #Tạo tiêu đề

plt.tight_layout() #Giảm lề
plt.show() #In hình
```

Sau khi dự đoán tập 14 ngày tiếp theo, ta thực hiện trực quan hóa chúng để quan sát dự đoán của mô hình. Tạo *figure* với 4 *plot*, sau đó lấy cột *date* ứng với chiều dài của tập *test*. Sau đó thêm vào mảng *dates* 14 ngày tiếp theo. Tạo mảng *titles* chứa tiêu đề các *plot*.

Thực hiện lặp *enumerate* cho *features* ngoại trừ *date* ra. Vẽ đường thực tế và đường dự đoán. Mỗi vòng lặp sẽ vẽ *plot* cho 1 *features* và thứ tự *plot* ứng với *idx* sau đó set tiêu đề cho *plot*.

### Dữ liệu tuần:

```
n_steps_in = 7 #Lấy {n_steps_in} tuần trước để dự đoán tuần này

dataset = dataframe_w[features[1:]].to_numpy() #Chuyển dataframe tuần th
ành mảng
```

```
X, y = split_sequences(dataset, n_steps_in) #Chia dataset thành X và y theo {n_steps_in}

n_input = X.shape[1] * X.shape[2] #Tính chiều dài input
n_output = y.shape[1] * y.shape[2] #Tính chiều dài output

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=len(X) -
(n_steps_in, shuffle = False) #Chia dữ liệu tập train và test trong đó test chứa {n_steps_in} để thực hiện dự đoán
```

Thực hiện tiền xử lý dữ liệu như tạo  $X$  và  $y$  bằng hàm `split_sequences()`. Tính chiều dài của input và output lần lượt gán vào  $n\_input$ ,  $n\_output$  để sử dụng cho quá trình xây dựng mô hình. Cuối cùng sử dụng `train_test_split` với tham số `shuffle` là `false` thực hiện chia dữ liệu thành tập test và tập train.

```
model = Sequential() #Khởi tạo model
model.add(SimpleRNN(20)) #Thêm lớp SimpleRNN
model.add(Dense(n_output)) #Thêm lớp Dense với n_output là số lượng tham số đầu ra
model.compile(optimizer='adam', loss='mae')
```

Tiến hành xây dựng mô hình sử dụng `Sequential()`, với lớp đầu tiên là `SimpleRNN` với 20 là số unit xử lý của lớp đó, tiếp đó lớp `Dense` sẽ dùng để trả về kết quả, nhận tham số  $n\_output$  nó sẽ trả về số lượng output tương ứng. Sau đó ta `compile` mô hình với `optimizer adam` và hàm `loss` là `mae` (mean absolute error).

```
start_time = datetime.now()
model.fit(X_train, y_train, epochs=1000, shuffle=False, verbose=2)
print(f"Thời gian train: {(datetime.now() - start_time).total_seconds()} giây")
```

Thực hiện train mô hình, với `epochs` là 1000, `shuffle` là `false`. Sau khi train xong sẽ in ra thời gian train mô hình.

```
test_predict = model.predict(X_test, verbose=0)
test_predict
```

Thực hiện dự đoán mô hình thông qua tập test.

```
fig, axs = plt.subplots(4,figsize = (20,10)) #Tạo plot

dates = dataframe_w['date'][-(n_steps_in):] #Lấy cột date
titles = ['Giá mở cửa', 'Giá trần', 'Giá sàn', 'Giá đóng cửa'] #Khai báo
mảng tiêu đề

for idx, feature in enumerate(features[1:]): #Lặp vẽ 4 tương ứng với các
feature được chọn trừ date
    axs[idx].plot(dates, dataframe_w[feature] [-
(n_steps_in):]) #Vẽ đường thực tế
    axs[idx].plot(dates, test_predict[:,idx]) #Vẽ đường dự đoán
    axs[idx].legend([f'true {feature}', f'predict {feature}']) #Tạo chú th
ích
    axs[idx].set_title(titles[idx]) #Tạo tiêu đề

plt.tight_layout() #Giảm lề
plt.show() #In hình
```

Sau khi dự đoán tập test, ta thực hiện trực quan hóa chúng để quan sát dự đoán của mô hình có chính xác không. Tạo figure với 4 plot, sau đó lấy cột date ứng với chiều dài của tập test. Tạo mảng titles chứa tiêu đề các plot.

Thực hiện lặp *enumerate* cho *features* ngoại trừ *date* ra. Vẽ đường thực tế và đường dự đoán. Mỗi vòng lặp sẽ vẽ plot cho 1 features và thứ tự plot ứng với *idx* sau đó set tiêu đề cho plot.

```
predicts = list(dataframe[features[1:]][-7:].to_numpy())
#Lấy dữ liệu 7 tuần trước

while(len(predicts) < 7+7): #Thực hiện vào lặp dự đoán 7 tuần tiếp theo
    X_predicts = np.array([predicts[-
7:]]) #Lấy 7 tuần trước từ mảng predicts
    predict = model.predict(X_predicts, verbose=0) #Thực hiện dự đoán
    predicts.append(predict[0]) #Thêm dự đoán vào mảng predicts để tiếp t
c dự đoán

predicts = np.array(predicts) #Chuyển predicts lại thành np array
predicts[7:] #In dự đoán
```

Thực hiện dự đoán 7 tuần trong tương lai bằng cách sử dụng 7 tuần dữ liệu có sẵn (7 tuần này ứng với  $n\_steps\_in$ ).

Ta sẽ tạo mảng  $X\_predicts$  chứa dữ liệu 7 tuần trước đó và thực hiện dự đoán sau đó thêm lại vào mảng  $predict$  và lặp lại như thế 7 lần.

```
fig, axs = plt.subplots(4, figsize = (20,10)) #Tạo plot

dates = dataframe_w['date'][-(n_steps_in):] #Lấy cột date
dates = dates_add(dates, 7, distance=7) #Thêm vào mảng date 7 tuần
titles = ['Giá mở cửa', 'Giá trần', 'Giá sàn', 'Giá đóng cửa'] #Khai báo
mảng tiêu đề

for idx, feature in enumerate(features[1:]): #Lặp vẽ 4 tương ứng với các
feature được chọn trừ date
    axs[idx].plot(dates[:7], predicts[:7,idx]) #Vẽ đường thực tế
    axs[idx].plot(dates, np.concatenate((test_predict[:,idx], predicts[7:,
idx]))) #Vẽ đường dự đoán
    axs[idx].legend([f'true {feature}', f'predict {feature}']) #Tạo chú th
ích
    axs[idx].set_title(titles[idx]) #Tạo tiêu đề

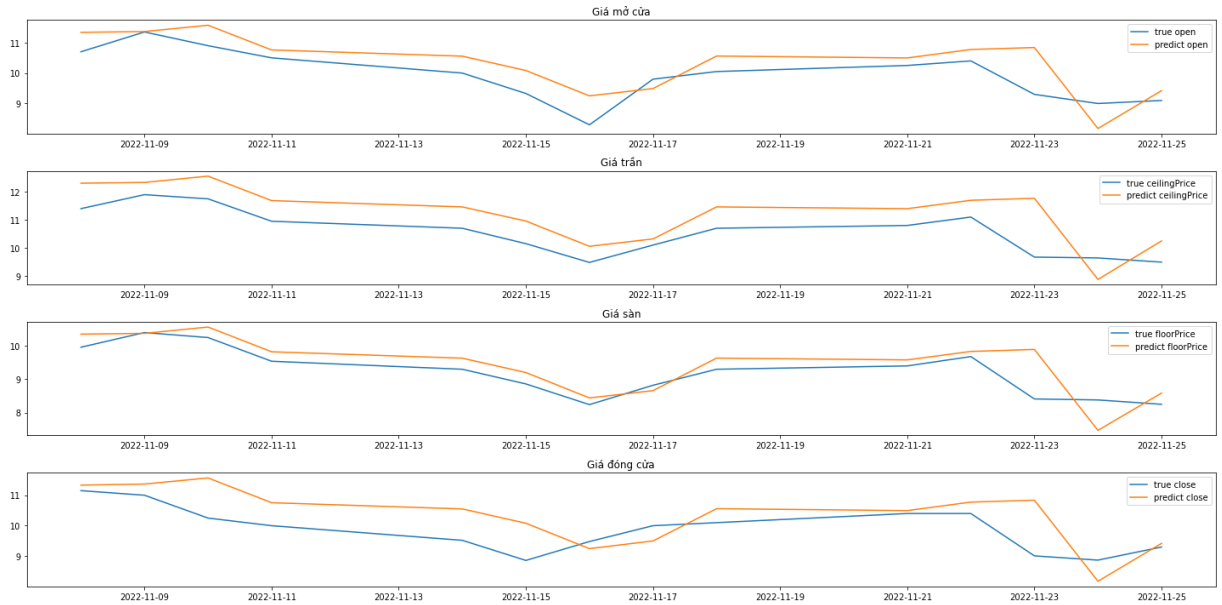
plt.tight_layout() #Giảm lề
plt.show() #In hình
```

Sau khi dự đoán tập 7 tuần tiếp theo, ta thực hiện trực quan hóa chúng để quan sát dự đoán của mô hình. Tạo figure với 4 plot, sau đó lấy cột date ứng với chiều dài của tập test. Sau đó thêm vào mảng day 7 tuần tiếp theo. Tạo mảng titles chứa tiêu đề các plot.

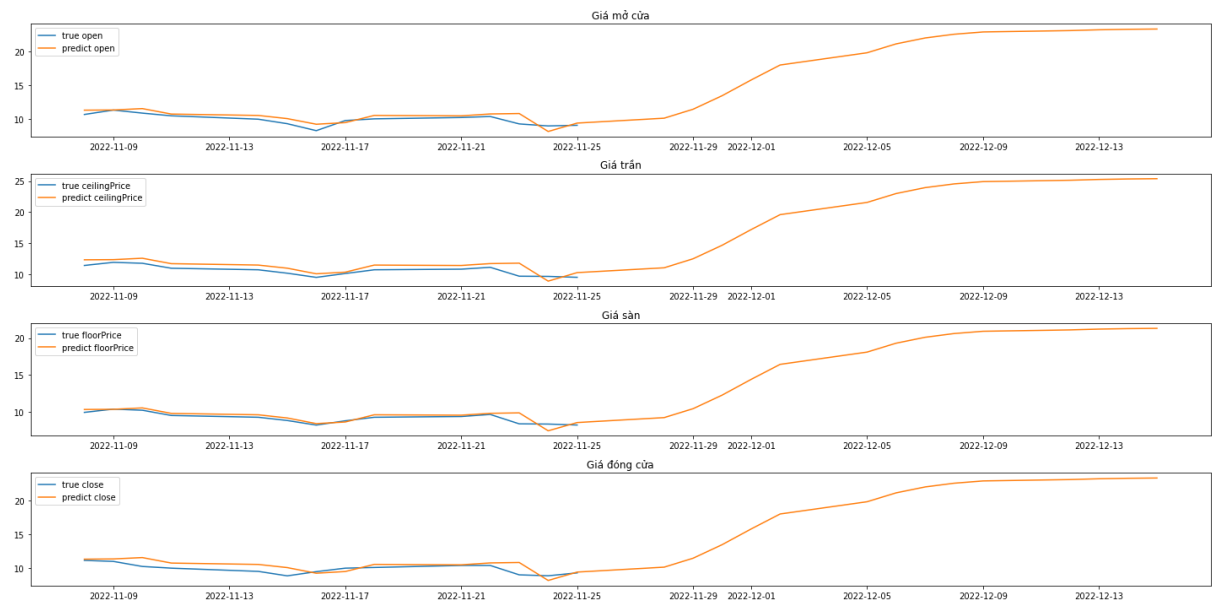
Thực hiện lặp *enumerate* cho *features* ngoại trừ *date* ra. Vẽ đường thực tế và đường dự đoán. Mỗi vòng lặp sẽ vẽ plot cho 1 features và thứ tự plot ứng với *idx* sau đó set tiêu đề cho plot.

#### 3.1.1.4 Kết quả thu được và đánh giá

 **Dữ liệu ngày :**

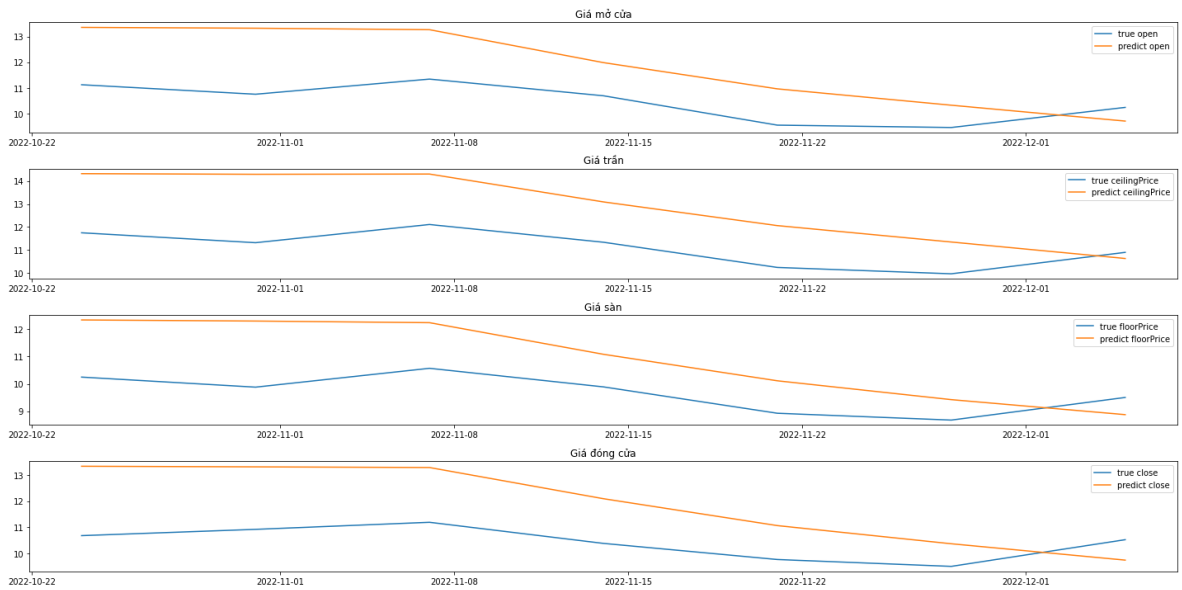


Hình 3.1.1.4a: Kết quả dự đoán giá cổ phiếu theo tập test khi dùng mô hình RNN

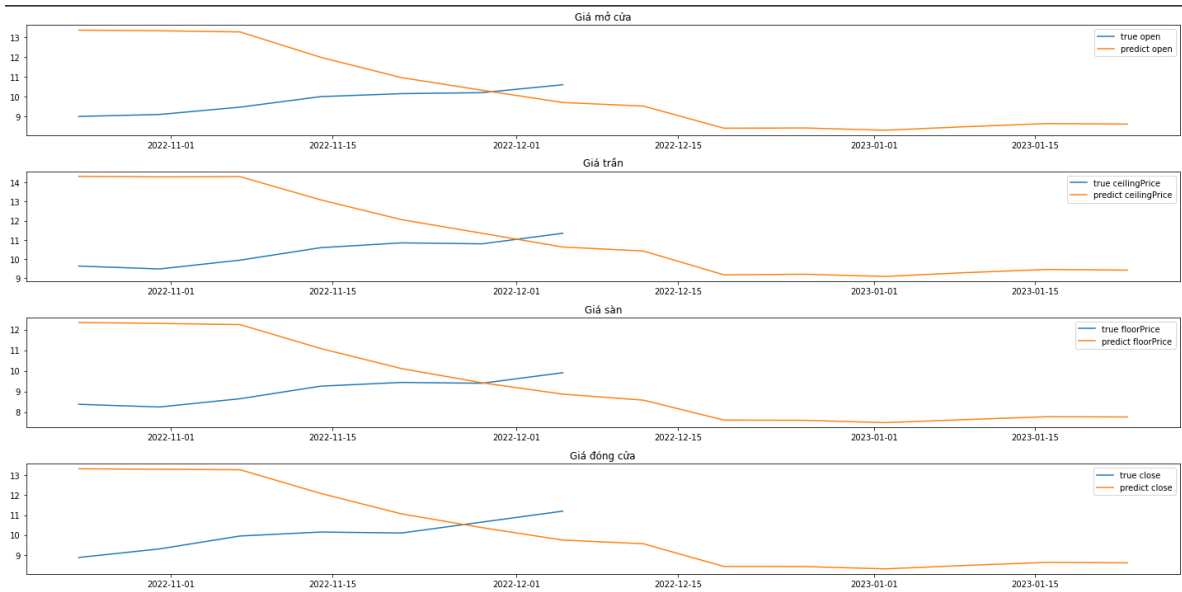


Hình 3.1.1.4b: Kết quả dự đoán giá cổ phiếu 14 ngày tiếp theo khi dùng mô hình RNN

### Dữ liệu tuần:



Hình 3.1.1.4c: Kết quả dự đoán giá cổ phiếu theo tập test khi dùng mô hình RNN



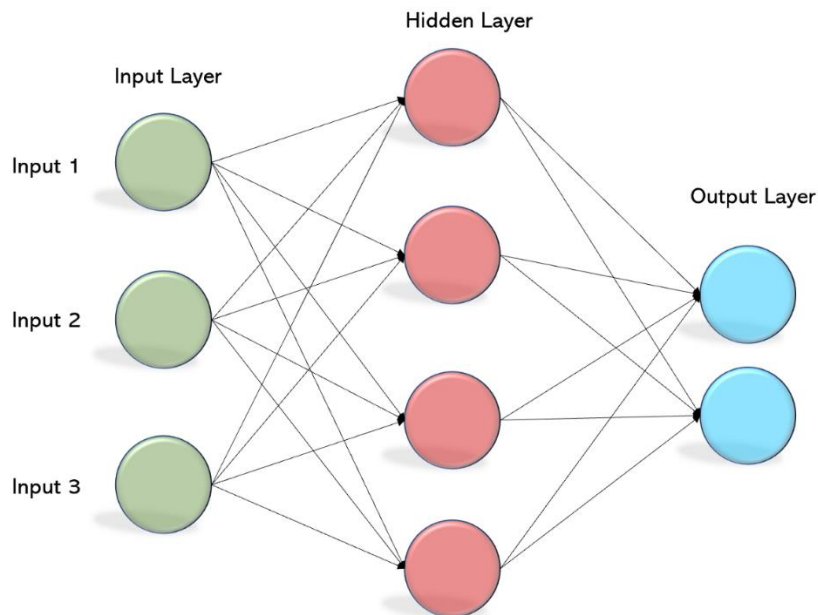
Hình 3.1.1.4d: Kết quả dự đoán giá cổ phiếu 7 tuần tiếp theo khi dùng mô hình RNN



### 3.1.2 Multi-Layer Perceptron

#### 3.1.2.1 Khái niệm về MLP

Multi-layer Perceptron (Perceptron nhiều lớp) bởi vì nó là tập hợp của các perceptron chia làm nhiều nhóm, mỗi nhóm tương ứng với một layer. Trong hình trên ta có một ANN với 3 lớp: input layer (lớp đầu vào), output layer (lớp đầu ra) và hidden layer (lớp ẩn). Số lượng layer trong một MLP được tính bằng số hidden layers cộng với 1. Tức là khi đếm số layers của một MLP, ta không tính input layers. Số lượng layer trong một MLP thường được ký hiệu là  $L$ . Thông thường khi giải quyết một bài toán ta chỉ quan tâm đến input và output của một model, do vậy trong MLP nói riêng và ANN nói chung ngoài lớp input và output thì các lớp neuron ở giữa được gọi chung là hidden (ẩn không phải là không nhìn thấy mà đơn giản là không quan tâm đến).



Hình 3.1.2.1 Multi-Layer Perceptron

MLP có thể được áp dụng cho dự báo chuỗi thời gian. Một thách thức với việc sử dụng MLP để dự báo chuỗi thời gian là chuẩn bị dữ liệu. Cụ thể, các quan sát độ trễ phải được làm phẳng thành các vector đặc trưng.

Kiến trúc của một mạng MLP tổng quát:

- Đầu vào là các vector  $(x_1, x_2, \dots, x_p)$  trong không gian  $p$  chiều, đầu ra là các vector  $(y_1, y_2, \dots, y_q)$  trong không gian  $q$  chiều. Đối với các bài toán phân loại,  $p$  chính là kích thước của mẫu đầu vào,  $q$  chính là số lớp cần phân loại.
- Mỗi neural thuộc tầng sau liên kết với tất cả các neuron thuộc tầng liền trước nó.
- Đầu ra của neural tầng trước là đầu vào của neuron thuộc tầng liền sau nó.

Hoạt động của mạng MLP như sau: tại tầng đầu vào các neural nhận tín hiệu vào xử lý (tính tổng trọng số, gửi tới hàm truyền) rồi cho ra kết quả (là kết quả của hàm truyền); kết quả này sẽ được truyền tới các neural thuộc tầng ẩn thứ nhất; các neuron tại đây tiếp nhận như là tín hiệu đầu vào, xử lý và gửi kết quả đến tầng ẩn thứ 2. Quá trình tiếp tục cho đến khi các neural thuộc tầng ra cho kết quả.

***Ưu điểm của mô hình MLP:***

- Có thể áp dụng cho các bài toán phi tuyến tính phức tạp.
- Hoạt động tốt với dữ liệu đầu vào lớn.
- Tỷ lệ chính xác tương tự có thể đạt được ngay với các dữ liệu nhỏ.
- Cung cấp nhanh dự đoán sau quá trình đào tạo.

***Nhược điểm của mô hình MLP:***

- Khó biết mỗi biến độc lập là biến phụ thuộc ở mức độ nào. Tính toán khó khăn và tốn thời gian
- Mô hình hoạt động đúng phụ thuộc vào chất lượng đào tạo.

### 3.1.2.2 Ứng dụng

MLP phù hợp với các vấn đề dự đoán phân loại trong đó đầu vào được gán một lớp hoặc nhãn. Chúng cũng phù hợp với các bài toán dự đoán hồi quy trong đó một đại lượng có giá trị thực được dự đoán với một tập hợp các đầu vào. Dữ liệu thường được cung cấp ở định dạng bảng, chẳng hạn như trong tệp CSV hoặc bảng tính.

Chúng rất linh hoạt và có thể được sử dụng nói chung để tìm hiểu ánh xạ từ đầu vào đến đầu ra. Tính linh hoạt này cho phép chúng được áp dụng cho các loại dữ liệu khác. Ví dụ: các pixel của hình ảnh có thể được giảm xuống thành một hàng dữ liệu dài và được đưa vào MLP. Các từ của tài liệu cũng có thể được rút gọn thành một hàng dữ liệu dài và được đưa vào MLP. Ngay cả các quan sát độ trễ cho một vấn đề dự đoán chuỗi thời gian cũng có thể được giảm xuống thành một hàng dữ liệu dài và được cung cấp cho MLP.

MLP là một giải pháp học máy phổ biến trong những năm 1980, tìm thấy các ứng dụng trong nhiều lĩnh vực khác nhau như nhận dạng giọng nói, nhận dạng hình ảnh và phần mềm dịch máy nhưng sau đó phải đối mặt với sự cạnh tranh mạnh mẽ từ các máy vector hỗ trợ đơn giản hơn.

### 3.1.2.3 Xây dựng và thực hiện dự đoán

#### **Dữ liệu ngày:**

```
n_steps_in = 14 #Lấy {n_steps_in} ngày trước để dự đoán ngày hôm nay

dataset = dataframe[features[1:]].to_numpy() #Chuyển dataframe thành mảng
X, y = split_sequences(dataset, n_steps_in) #Chia dataset thành X và y theo {n_steps_in}

n_input = X.shape[1] * X.shape[2] #Tính chiều dài input
X = X.reshape((X.shape[0], n_input)) #Chuyển X shape từ (?, {n_steps_in}, 4) thành (?, {n_steps_in}*4)
n_output = y.shape[1] * y.shape[2] #Tính chiều dài output
y = y.reshape((y.shape[0], n_output)) #Chuyển y shape từ (?, {n_steps_in}, 4) thành (?, {n_steps_in}*4)
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=len(X) -
n_steps_in, shuffle = False) #Chia dữ liệu tập train và test trong đó test chứa {n_steps_in} để thực hiện dự đoán
```

Thực hiện tiền xử lý dữ liệu như tạo X và y bằng hàm *split\_sequences()*.

Tiếp đó tính chiều dài của input và output lần lượt gán vào  $n\_input$ ,  $n\_output$  để sử dụng cho quá trình xây dựng mô hình. Cuối cùng sử dụng *train\_test\_split* với tham số *shuffle* là *false* thực hiện chia dữ liệu thành tập *test* và tập *train*.

```
model = Sequential() #Khởi tạo model
model.add(Dense(100, activation='relu', input_dim=n_input)) #Thêm lớp Dense với n_input là số lượng tham số đầu vào
model.add(Dense(n_output)) #Thêm lớp Dense với n_output là số lượng tham số đầu ra
model.compile(optimizer='adam', loss='mse')
```

Tiến hành xây dựng mô hình, khởi tạo sử dụng *Sequential()*, với lớp đầu tiên là *Dense* với 100 là số *unit* xử lý của lớp đó cùng với hàm *activation relu*. Tiếp theo, lớp *Dense* sẽ dùng để trả về kết quả, nhận tham số  $n\_output$  nó sẽ trả về số lượng *output* tương ứng.

Sau đó ta *compile* mô hình với *optimizer adam* và hàm *loss* là *mae* (mean squared error).

```
start_time = datetime.now()
model.fit(X_train, y_train, epochs=1000, shuffle=False, verbose=2)
print(f"Thời gian train: {(datetime.now() - start_time).total_seconds()} giây")
```

Thực hiện train mô hình, với *epochs* là 1000, *shuffle* là *false*. Sau khi train xong sẽ in ra thời gian train mô hình.

```
test_predict = model.predict(X_test, verbose=0)
test_predict
```

Thực hiện dự đoán mô hình thông qua tập *test*.

```
fig, axs = plt.subplots(4, figsize = (20,10)) #Tạo plot
dates = dataframe['date'][-(n_steps_in):] #Lấy cột date
titles = ['Giá mở cửa', 'Giá trần', 'Giá sàn', 'Giá đóng cửa'] #Khai báo mảng tiêu đề
```

```

for idx, feature in enumerate(features[1:]): #Lặp vẽ 4 tương ứng với các
    feature được chọn trừ date
    axs[idx].plot(dates, dataframe[feature] [-
(n_steps_in):]) #Vẽ đường thực tế
    axs[idx].plot(dates, test_predict[:,idx]) #Vẽ đường dự đoán
    axs[idx].legend([f'true {feature}', f'predict {feature}']) #Tạo chú th
ích
    axs[idx].set_title(titles[idx]) #Tạo tiêu đề

plt.tight_layout() #Giảm lề
plt.show() #In hình

```

Sau khi dự đoán tập *test*, ta thực hiện trực quan hóa chúng để quan sát dự đoán của mô hình có chính xác không. Tạo *figure* với 4 *plot*, sau đó lấy cột *date* ứng với chiều dài của tập *test*. Tạo mảng *titles* chứa tiêu đề các *plot*.

Thực hiện lặp *enumerate* cho *features* ngoại trừ *date* ra. Vẽ đường thực tế và đường dự đoán. Mỗi vòng lặp sẽ vẽ *plot* cho 1 *features* và thứ tự *plot* ứng với *idx* sau đó set tiêu đề cho *plot*.

```

predicts = list(dataframe[features[1:]] [-14:].to_numpy())
#Lấy dữ liệu 14 ngày trước

while(len(predicts) < 14+14): #Thực hiện vào lặp dự đoán 14 ngày tiếp th
eo
    X_predicts = np.array([predicts[-
14:]] ) #Lấy 14 ngày trước từ mảng predicts
    predict = model.predict(X_predicts, verbose=0) #Thực hiện dự đoán
    predicts.append(predict[0]) #Thêm dự đoán vào mảng predicts để tiếp tụ
c dự đoán

predicts = np.array(predicts) #Chuyển predicts lại thành np array
predicts

```

Thực hiện dự đoán 14 ngày trong tương lai bằng cách sử dụng 14 ngày dữ liệu có sẵn (14 ngày này ứng với *n\_steps\_in*).

Ta sẽ tạo mảng *X\_predicts* chứa dữ liệu 14 ngày trước đó và thực hiện dự đoán sau đó thêm lại vào mảng *predict* và lặp lại như thế 14 lần.

```

fig, axs = plt.subplots(4, figsize = (20,10)) #Tạo plot
dates = dataframe['date'][-(n_steps_in):] #Lấy cột date
dates = dates_add(dates, 14) #Thêm vào mảng date 14 ngày
titles = ['Giá mở cửa', 'Giá trần', 'Giá sàn', 'Giá đóng cửa'] #Khai báo
mảng tiêu đề

for idx, feature in enumerate(features[1:]): #Lặp vẽ 4 tương ứng với các
feature được chọn trừ date
    axs[idx].plot(dates[:14], predicts[:14,idx]) #Vẽ đường thực tế
    axs[idx].plot(dates, np.concatenate((test_predict[:,idx], predicts[14:
,idx]))) #Vẽ đường dự đoán
    axs[idx].legend([f'true {feature}', f'predict {feature}']) #Tạo chú th
ích
    axs[idx].set_title(titles[idx]) #Tạo tiêu đề

plt.tight_layout() #Giảm lề
plt.show() #In hình

```

Sau khi dự đoán tập 14 ngày tiếp theo, ta thực hiện trực quang hóa chúng để quang sát dự đoán của mô hình. Tạo figure với 4 plot, sau đó lấy cột date ứng với chiều dài của tập test. Sau đó thêm vào mảng day 14 ngày tiếp theo. Tạo mảng titles chứa tiêu đề các plot.

Thực hiện lặp *enumerate* cho *features* ngoại trừ *date* ra. Vẽ đường thực tế và đường dự đoán. Mỗi vòng lặp sẽ vẽ plot cho 1 features và thứ tự plot ứng với *idx* sau đó set tiêu đề cho plot.

### **Dữ liệu tuần:**

```

n_steps_in = 7 #Lấy {n_steps_in} ngày trước để dự đoán ngày hôm nay

dataset = dataframe_w[features[1:]].to_numpy() #Chuyển dataframe tuần th
ành mảng
X, y = split_sequences(dataset, n_steps_in) #Chia dataset thành X và y t
heo {n_steps_in}

n_input = X.shape[1] * X.shape[2] #Tính chiều dài input
X = X.reshape((X.shape[0], n_input)) #Chuyển X shape từ (?, {n_steps_in}
, 4) thành (?, {n_steps_in}*4)
n_output = y.shape[1] * y.shape[2] #Tính chiều dài output
y = y.reshape((y.shape[0], n_output)) #Chuyển y shape từ (?, {n_steps_in}
, 4) thành (?, {n_steps_in}*4)

```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=len(X) -
n_steps_in, shuffle = False) #Chia dữ liệu tập train và test trong đó tes
t chứa {n_steps_in} để thực hiện dự đoán
```

Thực hiện tiền xử lý dữ liệu như tạo  $X$  và  $y$  bằng hàm *split\_sequences()*. Tính chiều dài của input và output lần lượt gán vào  $n\_input$ ,  $n\_ouput$  để sử dụng cho quá trình xây dựng mô hình. Cuối cùng sử dụng *train\_test\_split* với tham số *shuffle* là *false* thực hiện chia dữ liệu thành tập test và tập train.

```
model = Sequential() #Khởi tạo model
model.add(Dense(100, activation='relu', input_dim=n_input)) #Thêm lớp De
nse với n_input là số lượng tham số đầu vào
model.add(Dense(n_output)) #Thêm lớp Dense với n_output là số lượng tham
số đầu ra
model.compile(optimizer='adam', loss='mse')
```

Tiến hành xây dựng mô hình, sử dụng *Sequential()*. Lớp đầu tiên là *Dense* với 100 là số *unit* xử lý của lớp đó cùng với hàm *activation relu*, tiếp đó lớp *Dense* sẽ dùng để trả về kết quả, nhận tham số  $n\_ouput$  nó sẽ trả về số lượng output tương ứng.

Sau đó, *compile* mô hình với *optimizer adam* và hàm *loss* là *mae* (mean squared error).

```
start_time = datetime.now()
model.fit(X_train, y_train, epochs=1000, shuffle=False, verbose=2)
print(f"Thời gian train: {(datetime.now() -
start_time).total_seconds()} giây")
```

Thực hiện train mô hình, với *epochs* là 1000, *shuffle* là *false*. Sau khi train xong sẽ in ra thời gian train mô hình.

```
test_predict = model.predict(X_test, verbose=0)
test_predict
```

Thực hiện dự đoán mô hình thông qua tập test.

```

fig, axs = plt.subplots(4, figsize = (20,10)) #Tạo plot

dates = dataframe_w['date'][-(n_steps_in):] #Lấy cột date
titles = ['Giá mở cửa', 'Giá trần', 'Giá sàn', 'Giá đóng cửa'] #Khai báo
mảng tiêu đề

for idx, feature in enumerate(features[1:]): #Lặp vẽ 4 tương ứng với các
feature được chọn trừ date
    axs[idx].plot(dates, dataframe_w[feature] [-
(n_steps_in):]) #Vẽ đường thực tế
    axs[idx].plot(dates, test_predict[:,idx]) #Vẽ đường dự đoán
    axs[idx].legend([f'true {feature}', f'predict {feature}']) #Tạo chú th
ích
    axs[idx].set_title(titles[idx]) #Tạo tiêu đề

plt.tight_layout() #Giảm lề
plt.show() #In hình

```

Sau khi dự đoán tập test, ta thực hiện trực quan hóa chúng để quan sát dự đoán của mô hình có chính xác không. Tạo figure với 4 plot, sau đó lấy cột date ứng với chiều dài của tập test. Tạo mảng titles chứa tiêu đề các plot.

Thực hiện lặp *enumerate* cho *features* ngoại trừ *date* ra. Vẽ đường thực tế và đường dự đoán. Mỗi vòng lặp sẽ vẽ plot cho 1 features và thứ tự plot ứng với *idx* sau đó set tiêu đề cho plot.

```

predicts = list(dataframe[features[1:]][-7:]).to_numpy()
#Lấy dữ liệu 7 tuần trước

while(len(predicts) < 7+7): #Thực hiện vào lặp dự đoán 7 tuần tiếp theo
    X_predicts = np.array(predicts[-
7:]) #Lấy 7 ngày trước từ mảng predicts
    X_predicts = X_predicts.reshape((1, n_input)) #Chuyển từ dạng (?, {n_s
teps_in}, 4) sang (?, {n_steps_in}*4)

    predict = model.predict(X_predicts, verbose=0) #Thực hiện dự đoán
    predicts.append(predict[0]) #Thêm dự đoán vào mảng predicts để tiếp tậ
c dự đoán

predicts = np.array(predicts) #Chuyển predicts lại thành np array

```



`predicts`

Thực hiện dự đoán 7 tuần trong tương lai bằng cách sử dụng 7 tuần dữ liệu có sẵn (7 tuần này ứng với *n\_steps\_in*).

Ta sẽ tạo mảng *X\_predicts* chứa dữ liệu 7 tuần trước đó và thực hiện dự đoán sau đó thêm lại vào mảng *predict* và lặp lại như thế 7 lần.

```
fig, axs = plt.subplots(4, figsize = (20,10)) #Tạo plot

dates = dataframe_w['date'][-(n_steps_in):] #Lấy cột date
dates = dates_add(dates, 7, distance=7) #Thêm vào mảng date 7 tuần
titles = ['Giá mở cửa', 'Giá trần', 'Giá sàn', 'Giá đóng cửa'] #Khai báo
mảng tiêu đề

for idx, feature in enumerate(features[1:]): #Lặp vẽ 4 tương ứng với các
feature được chọn trừ date
    axs[idx].plot(dates[:7], predicts[:7,idx]) #Vẽ đường thực tế
    axs[idx].plot(dates, np.concatenate((test_predict[:,idx], predicts[7:,
idx]))) #Vẽ đường dự đoán
    axs[idx].legend([f'true {feature}', f'predict {feature}']) #Tạo chú th
ích
    axs[idx].set_title(titles[idx]) #Tạo tiêu đề

plt.tight_layout() #Giảm lề
plt.show() #In hình
```

Sau khi dự đoán tập 7 tuần tiếp theo, ta thực hiện trực quan hóa chúng để quan sát dự đoán của mô hình. Tạo figure với 4 plot, sau đó lấy cột date ứng với chiều dài của tập test. Sau đó thêm vào mảng day 7 tuần tiếp theo. Tạo mảng titles chứa tiêu đề các plot.

Thực hiện lặp *enumerate* cho *features* ngoại trừ *date* ra. Vẽ đường thực tế và đường dự đoán. Mỗi vòng lặp sẽ vẽ plot cho 1 features và thứ tự plot ứng với *idx* sau đó set tiêu đề cho plot.

### 3.1.2.4 Kết quả thu được và đánh giá

 **Dữ liệu ngày:**

```
array([[ 9.60756 , 10.47454 ,  8.768526 ,  9.663989 ],
       [ 9.004007 ,  9.8176565,  8.239626 ,  9.0548315],
       [ 9.326159 , 10.167444 ,  8.516064 ,  9.38647  ],
       [ 9.992181 , 10.899145 ,  9.07981  , 10.07011  ],
       [10.001663 , 10.92297  ,  9.088495 , 10.064494 ],
       [10.343517 , 11.2880745,  9.384139 , 10.420899 ],
       [10.315494 , 11.307508 ,  9.380229 , 10.418673 ],
       [ 8.98792  ,  9.809316 ,  8.199908 ,  9.052899 ],
       [ 8.782697 ,  9.59462  ,  8.000716 ,  8.854669 ],
       [ 9.236932 , 10.108172 ,  8.389183 ,  9.318417 ],
       [ 9.829619 , 10.821043 ,  8.89674  ,  9.925589 ],
       [10.126193 , 11.126701 ,  9.143099 , 10.219041 ],
       [10.096383 , 11.089139 ,  9.114321 , 10.173598 ],
       [10.433965 , 11.477216 ,  9.392269 , 10.513042 ]], dtype=float32)
```

Hình 3.1.2.4a: In dự đoán theo tập test của mô hình MLP



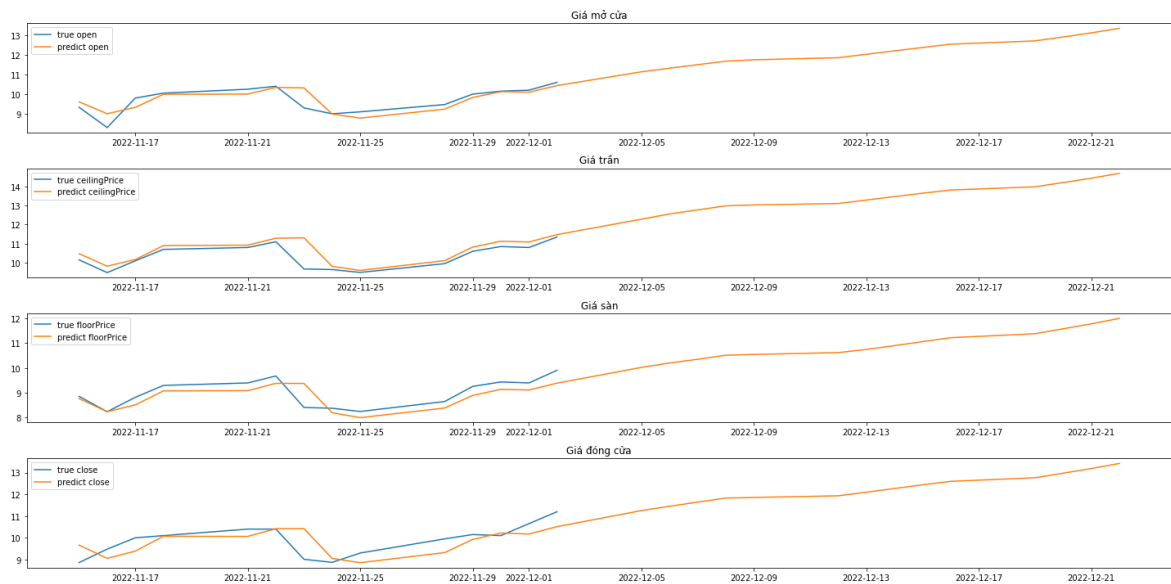
Hình 3.1.2.4b: Kết quả dự đoán theo tập test của mô hình MLP

```

array([[11.13405609, 12.28501987, 10.02651978, 11.25259781],
       [11.31743526, 12.56233501, 10.20376015, 11.44999599],
       [11.4965229 , 12.77411461, 10.35295868, 11.64226246],
       [11.67262363, 12.98899651, 10.51601601, 11.82733822],
       [11.74544334, 13.0330019 , 10.54860878, 11.85518837],
       [11.84993935, 13.11049271, 10.62277699, 11.93187714],
       [12.02462196, 13.28998566, 10.74939728, 12.0974369 ],
       [12.19848347, 13.47503948, 10.90386391, 12.26794434],
       [12.36881733, 13.64980888, 11.06389904, 12.43549156],
       [12.53719234, 13.81807327, 11.22344112, 12.59660244],
       [12.70501137, 13.98695374, 11.38367462, 12.76212597],
       [12.90505123, 14.21139622, 11.5831337 , 12.9726944 ],
       [13.11212063, 14.43843651, 11.7816391 , 13.18556976],
       [13.34528065, 14.69524288, 12.00226212, 13.42171764]])

```

Hình 3.1.2.4c: In dự đoán 14 ngày tiếp theo của mô hình MLP

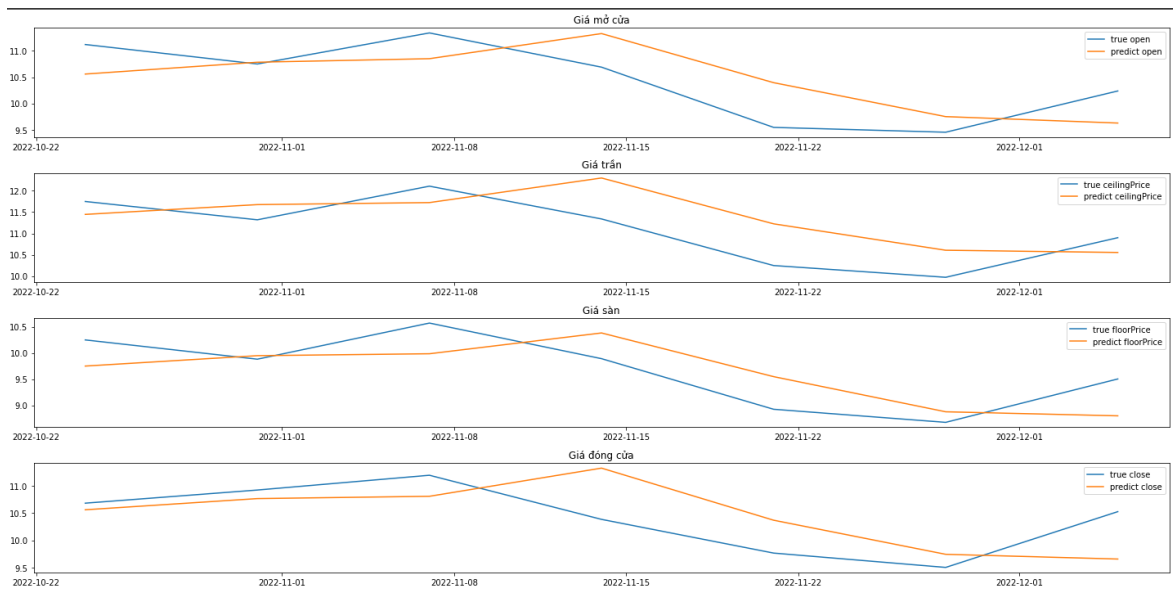


Hình 3.1.2.4d: Kết quả dự đoán 14 ngày tiếp theo của mô hình MLP

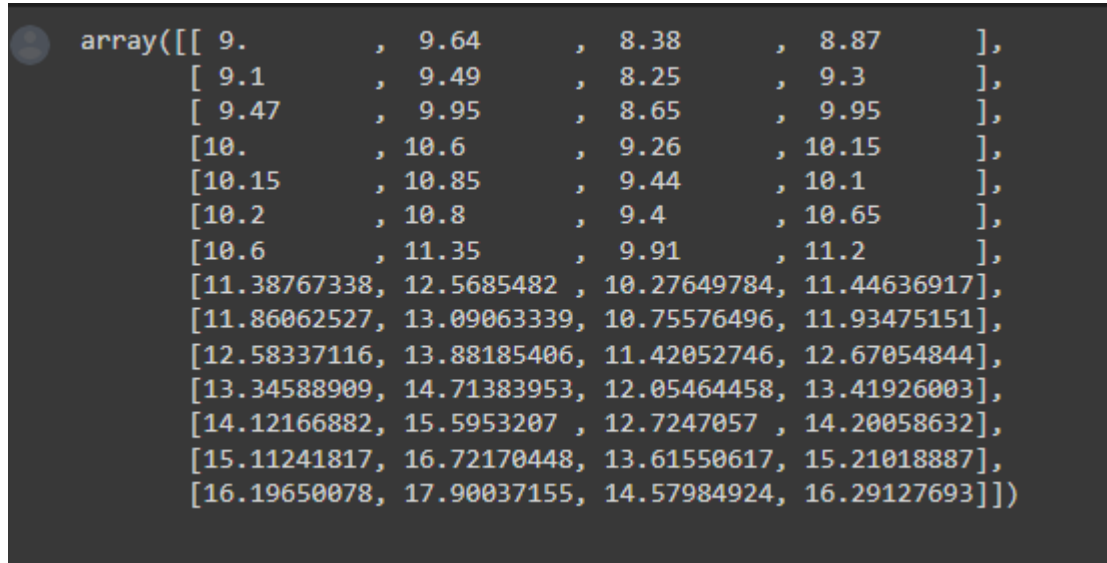
🚦 **Dữ liệu tuần:**

```
array([[10.558976 , 11.446758 , 9.748853 , 10.559181 ],
       [10.783684 , 11.676642 , 9.947391 , 10.763691 ],
       [10.850843 , 11.7242155, 9.983881 , 10.805662 ],
       [11.327844 , 12.301719 , 10.380887 , 11.320387 ],
       [10.395355 , 11.225461 , 9.546922 , 10.368546 ],
       [ 9.750425 , 10.60513 , 8.8758545, 9.74556 ],
       [ 9.6283 , 10.552873 , 8.798722 , 9.65918 ]], dtype=float32)
```

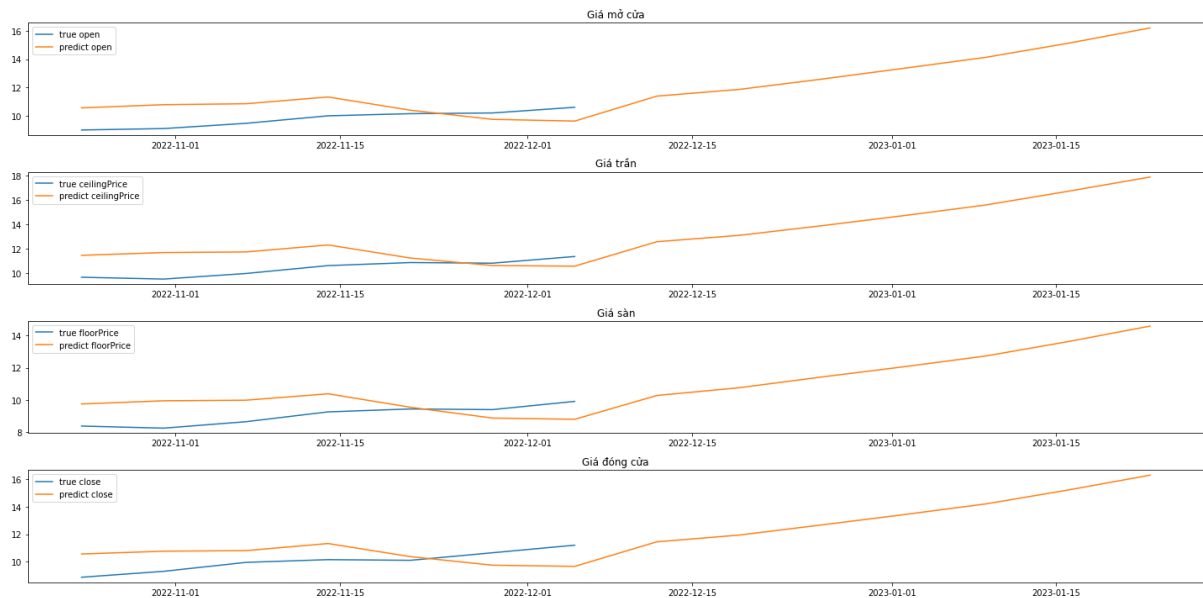
Hình 3.1.2.4e: In dự đoán tập test của mô hình MLP



Hình 3.1.2.4f: Kết quả dự đoán theo tập test của mô hình MLP



Hình 3.1.2.4g: Kết quả dự đoán 7 tuần tiếp theo của mô hình MLP



Hình 3.1.2.4h: Kết quả dự đoán 7 tuần tiếp theo của mô hình MLP

### 3.1.3 So sánh các mô hình

Sau khi thực hiện huấn luyện cả 2 mô hình, chúng ta có thể nhận thấy một số khác biệt ở cả hai.

### 3.1.3.1 Mô hình RNN

#### 📊 Dữ liệu ngày

```
Epoch 990/1000
77/77 - 1s - loss: 0.6663 - 687ms/epoch - 9ms/step
Epoch 991/1000
77/77 - 1s - loss: 0.6528 - 683ms/epoch - 9ms/step
Epoch 992/1000
77/77 - 1s - loss: 0.6504 - 673ms/epoch - 9ms/step
Epoch 993/1000
77/77 - 1s - loss: 0.6491 - 678ms/epoch - 9ms/step
Epoch 994/1000
77/77 - 1s - loss: 0.6463 - 679ms/epoch - 9ms/step
Epoch 995/1000
77/77 - 1s - loss: 0.6437 - 670ms/epoch - 9ms/step
Epoch 996/1000
77/77 - 1s - loss: 0.6414 - 704ms/epoch - 9ms/step
Epoch 997/1000
77/77 - 1s - loss: 0.6469 - 688ms/epoch - 9ms/step
Epoch 998/1000
77/77 - 1s - loss: 0.6492 - 654ms/epoch - 8ms/step
Epoch 999/1000
77/77 - 1s - loss: 0.6508 - 673ms/epoch - 9ms/step
Epoch 1000/1000
77/77 - 1s - loss: 0.6512 - 696ms/epoch - 9ms/step
Thời gian train: 690.330523 giây
```

Hình 3.1.3.1a: Huấn luyện mô hình RNN theo ngày

Ta có thể thấy sau lần train 1000: độ mất mát loss chỉ còn 0.6508 so với lần đầu train là 9.0091. Thời gian train của mô hình là: 690.330523s

#### 📊 Dữ liệu tuần

```

Epoch 990/1000
16/16 - 0s - loss: 1.4866 - 88ms/epoch - 6ms/step
Epoch 991/1000
16/16 - 0s - loss: 1.4919 - 89ms/epoch - 6ms/step
Epoch 992/1000
16/16 - 0s - loss: 1.4866 - 90ms/epoch - 6ms/step
Epoch 993/1000
16/16 - 0s - loss: 1.4917 - 91ms/epoch - 6ms/step
Epoch 994/1000
16/16 - 0s - loss: 1.4865 - 92ms/epoch - 6ms/step
Epoch 995/1000
16/16 - 0s - loss: 1.4918 - 99ms/epoch - 6ms/step
Epoch 996/1000
16/16 - 0s - loss: 1.4864 - 94ms/epoch - 6ms/step
Epoch 997/1000
16/16 - 0s - loss: 1.4916 - 99ms/epoch - 6ms/step
Epoch 998/1000
16/16 - 0s - loss: 1.4863 - 103ms/epoch - 6ms/step
Epoch 999/1000
16/16 - 0s - loss: 1.4915 - 97ms/epoch - 6ms/step
Epoch 1000/1000
16/16 - 0s - loss: 1.4862 - 95ms/epoch - 6ms/step
Thời gian train: 100.550878 giây

```

Hình 3.1.3.1b: Huấn luyện mô hình RNN theo tuần

Ta có thể thấy sau lần train 1000: độ mất mát loss chỉ còn 1.4862 so với lần đầu train là 11.1367. Thời gian train của mô hình là: 100.550878

### 3.1.3.2 Mô hình MLP

🚦 Dữ liệu ngày

```

Epoch 990/1000
77/77 - 0s - loss: 0.1717 - 159ms/epoch - 2ms/step
Epoch 991/1000
77/77 - 0s - loss: 0.1741 - 170ms/epoch - 2ms/step
Epoch 992/1000
77/77 - 0s - loss: 0.1748 - 156ms/epoch - 2ms/step
Epoch 993/1000
77/77 - 0s - loss: 0.1756 - 150ms/epoch - 2ms/step
Epoch 994/1000
77/77 - 0s - loss: 0.1760 - 154ms/epoch - 2ms/step
Epoch 995/1000
77/77 - 0s - loss: 0.1752 - 145ms/epoch - 2ms/step
Epoch 996/1000
77/77 - 0s - loss: 0.1816 - 143ms/epoch - 2ms/step
Epoch 997/1000
77/77 - 0s - loss: 0.1773 - 153ms/epoch - 2ms/step
Epoch 998/1000
77/77 - 0s - loss: 0.1727 - 156ms/epoch - 2ms/step
Epoch 999/1000
77/77 - 0s - loss: 0.1736 - 152ms/epoch - 2ms/step
Epoch 1000/1000
77/77 - 0s - loss: 0.1751 - 148ms/epoch - 2ms/step
Thời gian train: 202.241989 giây

```

Hình 3.1.3.2a: Huấn luyện mô hình MLP theo ngày

Ta có thể thấy sau lần train 1000: độ mất mát loss chỉ còn 0.6508 so với lần đầu train là 3.2867. Thời gian train của mô hình là: 100.550878

📊 Dữ liệu tuần



```

Epoch 990/1000
16/16 - 0s - loss: 1.0568 - 35ms/epoch - 2ms/step
Epoch 991/1000
16/16 - 0s - loss: 1.4457 - 33ms/epoch - 2ms/step
Epoch 992/1000
16/16 - 0s - loss: 2.1354 - 32ms/epoch - 2ms/step
Epoch 993/1000
16/16 - 0s - loss: 3.3734 - 35ms/epoch - 2ms/step
Epoch 994/1000
16/16 - 0s - loss: 2.8274 - 30ms/epoch - 2ms/step
Epoch 995/1000
16/16 - 0s - loss: 0.7907 - 38ms/epoch - 2ms/step
Epoch 996/1000
16/16 - 0s - loss: 0.6031 - 30ms/epoch - 2ms/step
Epoch 997/1000
16/16 - 0s - loss: 0.7587 - 32ms/epoch - 2ms/step
Epoch 998/1000
16/16 - 0s - loss: 0.6899 - 39ms/epoch - 2ms/step
Epoch 999/1000
16/16 - 0s - loss: 0.6780 - 32ms/epoch - 2ms/step
Epoch 1000/1000
16/16 - 0s - loss: 0.6708 - 33ms/epoch - 2ms/step
Thời gian train: 37.857601 giây

```

Hình 3.1.3.2b: Huấn luyện mô hình MLP theo tuần

Ta có thể thấy sau lần train 1000: độ mất mát loss chỉ còn 0.6708 so với lần đầu train là 15.2782. Thời gian train của mô hình là: 37.857601s

### 3.1.3.3 Kết luận

Từ các kết quả trên ta có thể thấy được ở mô hình dữ liệu ngày và tuần, chúng ta có thời gian train và độ mất mát của hai mô hình:

	RNN	MLP
Dữ liệu ngày	Thời gian train chậm: 690.330523s Độ mất mát nhiều: 0.6508	Thời gian train nhanh hơn: 202.241989 Độ mất mát tương đương RNN: 0.6508

Dữ liệu tuần	Thời gian train chậm:	Thời gian train nhanh hơn:
	100.550878s Độ mất mát nhiều: 1.4862	37.857601s Độ mất mát ít hơn: 0.6708

Bảng 3.1.3.3: So sánh mô hình RNN và MLP

Từ đó, chúng ta có thể kết luận mô hình MLP hiệu quả hơn RNN .

## 3.2 Overfitting

Khi huấn luyện một mô hình học máy, chúng ta sẽ gặp các trường hợp là underfitting, overfitting và good fit. Trong đó, good fit là kết quả tốt nhất, còn underfitting và overfitting đều khiến mô hình giảm đi độ chính xác. Hiện tượng mô hình quá khớp (overfitting) là một trong những nguyên nhân quan trọng làm giảm hiệu quả của mô hình học sâu.

### 3.2.1 Khái niệm về overfitting

Overfitting là hiện tượng khi mô hình xây dựng thể hiện quá chi tiết bộ dữ liệu đào tạo, quá tốt, quá khớp so với dữ liệu huấn luyện. Điều này có nghĩa là cả dữ liệu nhiều, hoặc dữ liệu bất thường trong tập đào tạo đều được chọn và học để đưa ra quy luật mô hình. Những quy luật này sẽ không có ý nghĩa nhiều khi áp dụng với bộ dữ liệu mới có thể có dạng dữ liệu nhiều khác. Theo một cách dễ hiểu có nghĩa rằng chúng đạt kết quả tốt trong tập dữ liệu đào tạo nhưng lại kém trong tập dữ liệu kiểm tra (thực tế). Khi đó, nó ảnh hưởng tiêu cực tới độ chính xác của mô hình nói chung.

Hiện tượng overfitting xảy ra khi:

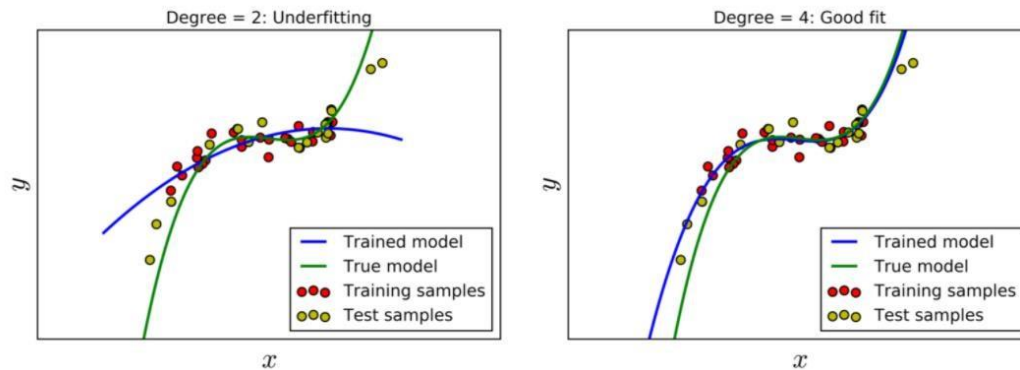
- Trong các mô hình phi tham số hoặc phi tuyến do phương sai quá cao và khi mà phương sai quá thấp mà bias quá cao.
- Thời gian training của mô hình hoặc độ phức tạp về kiến trúc của nó có thể làm cho mô hình bị quá tải.
- Nếu mô hình huấn luyện quá lâu trên dữ liệu huấn luyện hoặc quá phức tạp, nó sẽ bị nhiễu và thông tin sẽ không liên quan đến tập dữ liệu.

- Dữ liệu được sử dụng để đào tạo không được làm sạch và chứa các giá trị rác. Mô hình nắm bắt được nhiễu trong dữ liệu đào tạo và không tổng quát hóa được việc học của mô hình.
- Kích thước dữ liệu đào tạo không đủ và mô hình đào tạo trên dữ liệu đào tạo hạn chế.
- Kiến trúc của mô hình có một số lớp thần kinh được xếp chồng lên nhau. Mạng nơ-ron sâu rất phức tạp và đòi hỏi một lượng thời gian đáng kể để đào tạo và thường dẫn đến việc trang bị quá nhiều cho bộ đào tạo.



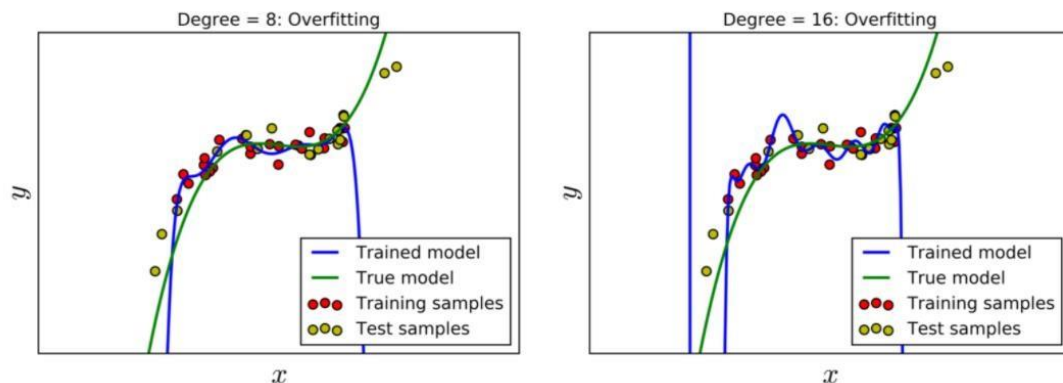
*Hình 3.2.1a: Dấu hiệu của overfitting*

Để hiểu rõ hơn về overfitting, ta có các hình ảnh ví dụ như sau:



Hình 3.2.1b: Ví dụ về underfitting, good fit

Trong hình 11, ta thấy trong trường hợp underfitting, mô hình huấn luyện (train model) quá đơn giản, không khớp so với dữ liệu huấn luyện. Còn bên good fit, mô hình có độ phức tạp vừa đủ, mang tính tổng quát, nó khớp với dữ liệu huấn luyện, dữ liệu test và gần giống với mô hình thực tế (true model).



Hình 3.2.1c: Ví dụ về overfitting

Trong hình 12, ta thấy mô hình huấn luyện (train model) khá phức tạp, nó quá khớp với dữ liệu huấn luyện, nhưng lại không mang tính tổng quát và khác xa so với mô hình thực tế (true model). Điều này dẫn đến việc mô hình sẽ dự đoán sai những dữ liệu mới không có trong dữ liệu huấn luyện. Ví dụ như những điểm test samples nằm trên cùng trong hình 12, mô hình huấn luyện hoàn toàn không thể chạm tới nó được.

**Các định nghĩa chính về Overfitting:**

- *Bias*: Bias đo lường sự khác biệt giữa dự đoán của mô hình và giá trị mục tiêu. Nếu mô hình được đơn giản hóa quá mức, thì giá trị dự đoán sẽ khác xa với dữ liệu thật dẫn đến sai lệch nhiều hơn.
- *Variance*: Phương sai là thước đo sự không nhất quán của các dự đoán khác nhau trên các tập dữ liệu khác nhau. Nếu hiệu suất của mô hình được kiểm tra trên các tập dữ liệu khác nhau thì dự đoán càng gần, phương sai càng ít. Phương sai cao là một dấu hiệu của việc overfitting, trong đó mô hình mất khả năng tổng quát hóa.
- *Sự cân bằng phương sai sai lệch*: Một mô hình tuyến tính đơn giản được kỳ vọng sẽ có độ chệch cao và phương sai thấp do mô hình ít phức tạp hơn và ít tham số có thể huấn luyện hơn. Mặt khác, các mô hình phi tuyến tính phức tạp có xu hướng quan sát một hành vi ngược lại. Trong một kịch bản lý tưởng, mô hình sẽ có sự cân bằng tối ưu giữa độ chệch và phương sai.
- *Model generalization*: Tổng quát hóa mô hình có nghĩa là mô hình được đào tạo tốt như thế nào để trích xuất các mẫu dữ liệu hữu ích và phân loại các mẫu dữ liệu không nhìn thấy.
- *Feature selection*: Liên quan đến việc chọn một tập hợp con các tính năng từ tất cả các tính năng được trích xuất đóng góp nhiều nhất vào hiệu suất của mô hình. Bao gồm tất cả các tính năng không cần thiết làm tăng độ phức tạp của mô hình và các tính năng dư thừa có thể làm tăng đáng kể thời gian training.

### 3.2.2 Phương pháp tránh overfitting

Để tránh overfitting, ta có thể sử dụng một số phương pháp như sau:

- Validation
- Cross-validation
- Regularization
- Early stopping

- Dropout

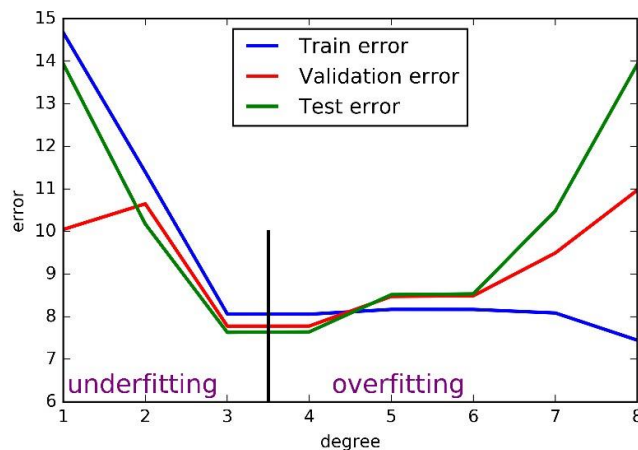
### 3.2.2.1 Validation

Để biết được chất lượng của mô hình với dữ liệu chưa nhìn thấy, phương pháp đơn giản nhất là trích từ tập training data ra một tập con nhỏ (tập con này có vai trò như test data) và thực hiện việc đánh giá mô hình trên tập con nhỏ này. Tập con nhỏ được trích ra từ training set này được gọi là validation set. Lúc này, training set là phần còn lại của training set ban đầu. Train error được tính trên training set mới này và có một khái niệm nữa được định nghĩa tương tự như trên validation error, tức là error được tính trên tập validation.

Với khái niệm này, ta sẽ tìm mô hình sao cho cả train error và validation error đều nhỏ, qua đó có thể dự đoán được rằng test error cũng nhỏ. Phương pháp thường được sử dụng là sử dụng nhiều mô hình khác nhau. Mô hình nào cho validation error nhỏ nhất sẽ là mô hình tốt.

Thông thường, ta bắt đầu từ mô hình đơn giản, sau đó tăng dần độ phức tạp của mô hình. Tới khi nào validation error có chiều hướng tăng lên thì chọn mô hình ngay trước đó. Chú ý rằng mô hình càng phức tạp, train error có xu hướng càng nhỏ đi.

Sau đây là ví dụ cụ thể với độ phức tạp là bậc của một đa thức.



Hình 3.2.2.1: Ảnh hưởng của độ phức tạp lên các loại error

Từ hình trên, ta có thể nhận xét rằng:

- Khi bậc của đa thức quá nhỏ (mô hình quá đơn giản), sẽ xảy ra hiện tượng underfitting. Khi bậc của đa thức càng cao (mô hình quá phức tạp), sẽ xảy ra hiện tượng overfitting.
- Nếu bậc của đa thức nằm trong khoảng từ 3 đến 4, đó là kết quả good fit mà chúng ta mong muốn. Để tối ưu mô hình, tránh overfitting, ta phải giữ cho độ phức tạp nằm trong khoảng này.
- Đầu tiên, chúng ta bắt đầu từ bậc 1, sau đó nâng dần bậc lên, cho đến khi validation error có dấu hiệu tăng (tại bậc 4), thì ta dừng lại, và chọn mô hình với bậc là 3.

### 3.2.2.2 Cross-validation

Cross-validation (hay còn gọi là k-fold cross validation) là một phiên bản cải tiến của validation. Phương pháp này có thể được sử dụng trong trường hợp số lượng dữ liệu bị hạn chế, chúng ta không thể lấy quá nhiều dữ liệu cho validation set vì training set có thể sẽ không đủ dữ liệu để huấn luyện, và chúng ta cũng không thể lấy quá ít dữ liệu cho validation set vì sẽ xảy ra overfitting ở training set.

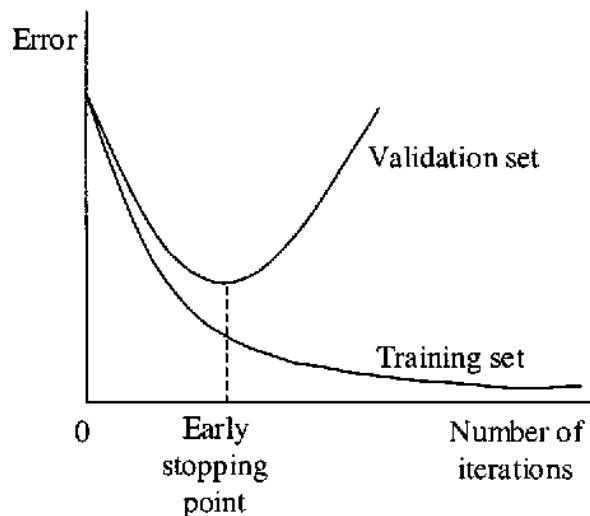
Phương pháp này giúp đánh giá mô hình dựa trên nhiều tập validation khác nhau. Có nghĩa là, thay vì chỉ chọn ra 1 tập validation set và 1 tập training set, ta sẽ chọn ra k tập validation set không có phần tử chung và k tập training set tương ứng. Kích thước của các tập này gần bằng nhau. Mô hình sẽ được đánh giá dựa trên trung bình của các train error và validation error.



Hình 3.2.2.2: Cross-validation

### 3.2.2.3 Early stopping

Phương pháp early stopping sẽ dừng việc huấn luyện khi đến một thời điểm nào đó, cụ thể là lúc mô hình có dấu hiệu của hiện tượng overfitting. Thông thường, trong quá trình chúng ta huấn luyện mô hình, loss của tập train và tập test sẽ cùng nhau giảm qua mỗi epoch. Tuy nhiên, đến một lúc nào đó thì loss của tập train giảm nhưng loss của tập test lại tăng, đó là dấu hiệu của hiện tượng overfitting. Nếu áp dụng early stopping thì việc huấn luyện sẽ dừng ngay lúc này.

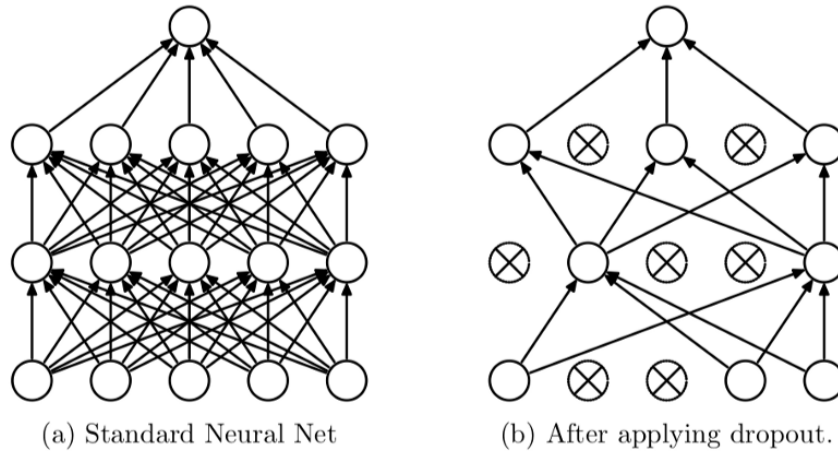




Hình 3.2.2.3: *Early stopping*

## 3.2.2.4 Dropout

Phương pháp dropout giúp ta tránh hiện tượng overfitting khi huấn luyện mô hình mạng nơ-ron nhiều lớp (deep neural network). Phương pháp này làm giảm độ phức tạp của mô hình, tránh overfitting bằng cách tắt đi ngẫu nhiên một số node trong mạng nơ-ron, có nghĩa là gán cho các node này giá trị bằng 0. Sau đó, chúng ta vẫn tiến hành train mô hình theo các bước tính toán feedforward và backpropagation như thường. Phương pháp không chỉ giúp làm giảm độ phức tạp của mô hình, mà còn giúp làm giảm đi lượng tính toán, khiến mô hình chạy nhanh hơn.

Hình 3.2.2.4: *Dropout*

## 3.2.2.5 Regularization

Regularization là một kỹ thuật phạt hệ số, thêm vào hàm (loss) một số hạng. Trong một mô hình overfit, các hệ số thường bị thổi phồng. Số hạng này thường dùng để đánh giá độ phức tạp của mô hình. Khi số hạng này càng lớn thì độ phức tạp của mô hình càng cao:

$$J_{reg}(\theta) = J(\theta) + \lambda R(\theta)$$

Trong đó:

$J_{reg}(\theta)$ : Hàm mất mát khi áp dụng phương pháp Regularization

$J(\theta)$ : Hàm mất mát

$\lambda$ : Giá trị này nếu quá lớn sẽ làm giảm hàm mất mát, khiến mô hình huấn luyện bị sai, dẫn đến underfitting

$R(\theta)$ : Một hàm regularization được thêm vào

Có hai loại kỹ thuật hồi quy hóa chính: Regularization and Lasso Regularization.

- Hồi quy Lasso (còn được gọi là hồi quy L1) là một kỹ thuật chính quy được sử dụng để giảm độ phức tạp của mô hình, nó sửa đổi các mô hình được trang bị quá mức hoặc không phù hợp bằng cách thêm penalty tương đương với tổng các giá trị tuyệt đối của các hệ số. Hồi quy Lasso cũng thực hiện tối thiểu hóa hệ số, nhưng thay vì bình phương độ lớn của các hệ số, nó lấy giá trị thực của các hệ số. Điều này có nghĩa là tổng hệ số cũng có thể bằng 0 do có các hệ số âm.

$$\sum_{i=1}^M (y_i - \hat{y}_i)^2 = \sum_{i=1}^M (y_i - \sum_{j=0}^P w_j * x_{ij})^2 + \lambda \sum_{j=0}^P |w_j|$$

*Cost function for Lasso regression*

- Hồi quy Ridge (còn được gọi là hồi quy L2) thực hiện hồi quy hóa bằng cách thu nhỏ các hệ số hiện có, nó sửa đổi các mô hình được trang bị quá mức hoặc không phù hợp bằng cách thêm penalty tương đương với tổng bình phương độ lớn của các hệ số. Điều này có nghĩa là hàm toán học đại diện cho mô hình học máy được giảm thiểu và các hệ số được tính toán. Độ lớn của các hệ số được bình phương và thêm vào.

$$\sum_{i=1}^M (y_i - \hat{y}_i)^2 = \sum_{i=1}^M (y_i - \sum_{j=0}^P w_j * x_{ij})^2 + \lambda \sum_{j=0}^P w_j^2$$

*Cost function for ridge regression*

*Sự khác biệt giữa Ridge Regression với Lasso Regression:*

- Hồi quy Ridge chỉ giúp giảm bớt sự overfitting trong mô hình trong khi vẫn giữ tất cả các tính năng có trong mô hình. Nó làm giảm độ phức tạp của mô hình bằng cách thu nhỏ các hệ số trong khi hồi quy Lasso giúp giảm vấn đề trang bị quá mức trong mô hình cũng như lựa chọn tính năng tự động.
- Hồi quy Lasso có xu hướng làm cho các hệ số về 0 tuyệt đối trong khi hồi quy Ridge không bao giờ đặt giá trị của hệ số về 0 tuyệt đối.

### 3.2.3 Áp dụng chống overfitting vào các mô hình

#### 3.2.3.1 RNN

 **Dữ liệu ngày:**

```
n_steps_in = 14 #Lấy {n_steps_in} ngày trước để dự đoán ngày hôm nay

dataset = dataframe[features[1:]].to_numpy() #Chuyển dataframe thành mảng
X, y = split_sequences(dataset, n_steps_in) #Chia dataset thành X và y theo {n_steps_in}

n_input = X.shape[1] * X.shape[2] #Tính chiều dài input
n_output = y.shape[1] * y.shape[2] #Tính chiều dài output

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=len(X)-n_steps_in, shuffle = False)
#Chia dữ liệu tập train và test trong đó test chứa {n_steps_in} để thực hiện dự đoán
```

Thực hiện tiền xử lý dữ liệu như tạo  $X$  và  $y$  bằng hàm `split_sequences()`. Tính chiều dài của input và output lần lượt gán vào `n_input`, `n_output` để sử dụng cho quá trình xây dựng mô hình. Cuối cùng sử dụng `train_test_split` với tham số `shuffle` là `false` thực hiện chia dữ liệu thành tập test và tập train.

```
model = Sequential() #Khởi tạo model
```

```
model.add(SimpleRNN(20, kernel_regularizer=l2(0.01), recurrent_regularizer=l2(0.01), bias_regularizer=l2(0.01))) #Thêm lớp SimpleRNN với regularizer
model.add(Dropout(0.1)) #Thêm lớp Drop Out
model.add(Dense(n_output)) #Thêm lớp Dense với n_output là số lượng tham số đầu ra
model.compile(optimizer='adam', loss='mae')
```

Tiến hành xây dựng mô hình, sử dụng *Sequential()*. Lớp đầu tiên là *SimpleRNN* với 20 là số *unit* xử lý của lớp đó, cùng với *kernel\_regularizer*, *recurrent\_regularizer*, *bias\_regularizer* có chung tham số *l2(0.01)* giúp cho mô hình trở nên đơn giản hơn. Tiếp đến là lớp *Dropout*, với tham số 0.1 là rate loại bỏ trọng số giúp cho mô hình đơn giản hơn. Lớp *Dense* sẽ dùng để trả về kết quả, nhận tham số *n\_output* nó sẽ trả về số lượng output tương ứng.

Sau đó *compile* mô hình với *optimizer adam* và hàm *loss* là *mae* (mean absolute error)

```
callback = EarlyStopping(monitor='loss', patience=50, restore_best_weights=True) #EarlyStop nếu 50 epoch không thay đổi loss trả về mô hình với loss thấp nhất

start_time = datetime.now() #Lưu thời gian bắt đầu train
model.fit(X_train, y_train, epochs=1000, shuffle=False, verbose=2, callbacks=[callback]) #Train mô hình
print(f"Thời gian train: {(datetime.now()-start_time).total_seconds()} giây") #In thời gian train
```

Khai báo *callback* là *EarlyStopping* có nhiệm vụ giám sát giá trị *loss*, *patience 50* là nếu giá trị *loss* không thay đổi qua 50 epoch thì kết thúc train, *restore\_best\_weight = True* sẽ trả về mô hình có *loss* thấp nhất.

Thực hiện train mô hình, với *epochs* là 1000, *shuffle* false cùng với *callback*. Sau khi train xong sẽ in ra thời gian train mô hình.

```
test_predict = model.predict(X_test, verbose=0)
test_predict
```

Thực hiện dự đoán mô hình thông qua tập test.

```
fig, axs = plt.subplots(4, figsize = (20,10)) #Tạo plot

dates = dataframe['date'][-(n_steps_in):] #Lấy cột date
titles = ['Giá mở cửa', 'Giá trần', 'Giá sàn', 'Giá đóng cửa'] #Khai báo
mảng tiêu đề

for idx, feature in enumerate(features[1:]): #Lặp vẽ 4 tương ứng với các
feature được chọn trừ date
    axs[idx].plot(dates, dataframe[feature] [-
(n_steps_in):]) #Vẽ đường thực tế
    axs[idx].plot(dates, test_predict[:,idx]) #Vẽ đường dự đoán
    axs[idx].legend([f'true {feature}', f'predict {feature}']) #Tạo chú th
ích
    axs[idx].set_title(titles[idx]) #Tạo tiêu đề

plt.tight_layout() #Giảm lề
plt.show() #In hình
```

Sau khi dự đoán tập test, ta thực hiện trực quang hóa chúng để quang sát dự đoán của mô hình có chính xác không. Tạo figure với 4 plot, sau đó lấy cột date ứng với chiều dài của tập test. Tạo mảng titles chứa tiêu đề các plot.

Thực hiện lặp *enumerate* cho *features* ngoại trừ *date* ra. Vẽ đường thực tế và đường dự đoán. Mỗi vòng lặp sẽ vẽ plot cho 1 features và thứ tự plot ứng với *idx* sau đó set tiêu đề cho plot.

```
predicts = list(dataframe[features[1:]][-14:].to_numpy())
#Lấy dữ liệu 14 ngày trước

while(len(predicts) < 14+14): #Thực hiện vào lặp dự đoán 14 ngày tiếp th
eo
    X_predicts = np.array([predicts[-
14:]] #Lấy 14 ngày trước từ mảng predicts
    predict = model.predict(X_predicts, verbose=0) #Thực hiện dự đoán
    predicts.append(predict[0]) #Thêm dự đoán vào mảng predicts để tiếp tậ
c dự đoán
```

```
predicts = np.array(predicts) #Chuyển predicts lại thành np array
predicts
```

Tiếp tục, ta sẽ thực hiện dự đoán 14 ngày trong tương lai bằng cách sử dụng 14 ngày dữ liệu có sẵn (14 ngày này ứng với  $n\_steps\_in$ ).

Tạo mảng  $X\_predicts$  chứa dữ liệu 14 ngày trước đó và thực hiện dự đoán sau đó thêm lại vào mảng  $predict$  và lặp lại như thế 14 lần.

```
fig, axs = plt.subplots(4,figsize = (20,10)) #Tạo plot

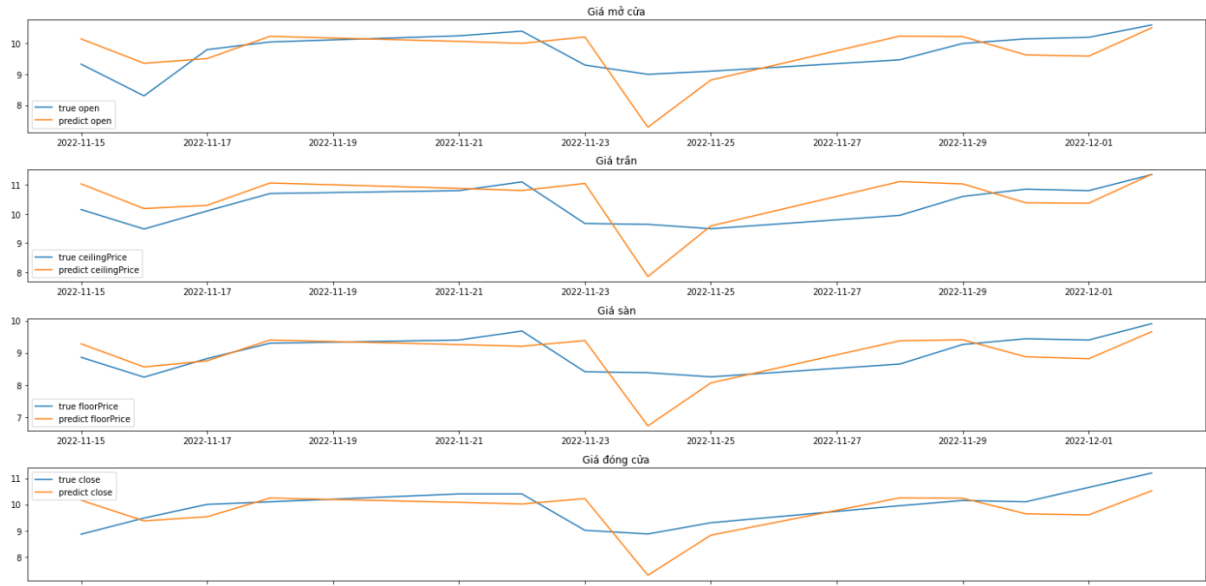
dates = dataframe['date'][-(n_steps_in):] #Lấy cột date
dates = dates_add(dates, 14) #Thêm vào mảng date 14 ngày
titles = ['Giá mở cửa', 'Giá trần', 'Giá sàn', 'Giá đóng cửa'] #Khai báo
mảng tiêu đề

for idx, feature in enumerate(features[1:]): #Lặp vẽ 4 tương ứng với các
feature được chọn trừ date
    axs[idx].plot(dates[:14], predicts[:14,idx]) #Vẽ đường thực tế
    axs[idx].plot(dates, np.concatenate((test_predict[:,idx], predicts[14:
,idx]))) #Vẽ đường dự đoán
    axs[idx].legend([f'true {feature}', f'predict {feature}']) #Tạo chú th
ích
    axs[idx].set_title(titles[idx]) #Tạo tiêu đề

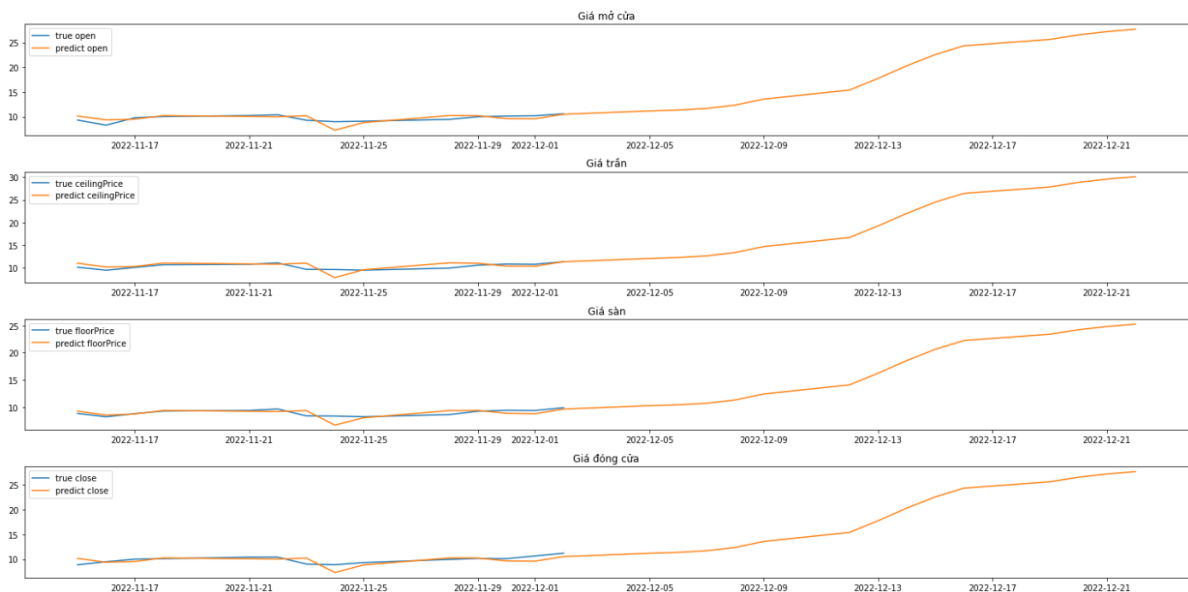
plt.tight_layout() #Giảm lề
plt.show() #In hình
```

Sau khi dự đoán tập 14 ngày tiếp theo, ta thực hiện trực quan hóa chúng để quan sát dự đoán của mô hình. Tạo figure với 4 plot, sau đó lấy cột date ứng với chiều dài của tập test. Sau đó thêm vào mảng day 14 ngày tiếp theo. Tạo mảng titles chứa tiêu đề các plot.

Thực hiện lặp *enumerate* cho *features* ngoại trừ *date* ra. Vẽ đường thực tế và đường dự đoán. Mỗi vòng lặp sẽ vẽ plot cho 1 features và thứ tự plot ứng với *idx* sau đó set tiêu đề cho plot.



Hình 3.2.3.1a: Tập test



Hình 3.2.3.1b: Dự đoán 14 ngày tiếp theo sử dụng chống overfitting cho mô hình RNN

 **Dữ liệu tuần:**

```
n_steps_in = 7 #Lấy {n_steps_in} tuần trước để dự đoán tuần này
```

```
dataset = dataframe_w[features[1:]].to_numpy() #Chuyển dataframe tuần th
ành mảng
X, y = split_sequences(dataset, n_steps_in) #Chia dataset thành X và y t
heo {n_steps_in}

n_input = X.shape[1] * X.shape[2] #Tính chiều dài input
n_output = y.shape[1] * y.shape[2] #Tính chiều dài output

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=len
(X) -
n_steps_in, shuffle = False) #Chia dữ liệu tập train và test trong đó tes
t chứa {n_steps_in} để thực hiện dự đoán
```

Thực hiện tiền xử lý dữ liệu như tạo X và y bằng hàm *split\_sequences()*. Tính chiều dài của input và output lần lượt gán vào *n\_input*, *n\_output* để sử dụng cho quá trình xây dựng mô hình.

Cuối cùng sử dụng *train\_test\_split* với tham số *shuffle* là false thực hiện chia dữ liệu thành tập test và tập train.

```
model = Sequential() #Khởi tạo model
model.add(SimpleRNN(20, kernel_regularizer=l2(0.01), recurrent_regularize
r=l2(0.01), bias_regularizer=l2(0.01))) #Thêm lớp SimpleRNN với regulari
zer
model.add(Dropout(0.1)) #Thêm lớp Drop Out
model.add(Dense(n_output)) #Thêm lớp Dense với n_output là số lượng tham
số đầu ra
model.compile(optimizer='adam', loss='mae')
```

Ta tiến hành xây dựng mô hình, sử dụng Sequential, với lớp đầu tiên là SimpleRNN với số 20 là số unit xử lý của lớp đó, cùng với kernel\_regularizer, recurrent\_regularizer, bias\_regularizer có chung tham số l2(0.01) giúp cho mô hình trở nên đơn giản hơn. Sau đó là lớp Dropout, với tham số 0.1 là rate loại bỏ trọng số giúp cho mô hình đơn giản hơn. Tiếp đó lớp Dense sẽ dùng để trả về kết quả, nhận tham số n\_output nó sẽ trả về số lượng output tương ứng.

Sau đó ta compile mô hình với optimizer adam và hàm loss là mae (mean absolute error)



```
callback = EarlyStopping(monitor='loss', patience=50, restore_best_weights=True) #EarlyStop nếu 50 epoch không thay đổi loss trả về mô hình với loss thấp nhất

start_time = datetime.now() #Lưu thời gian bắt đầu train
model.fit(X_train, y_train, epochs=1000, shuffle=False, verbose=2, callbacks=[callback]) #Train mô hình
print(f"Thời gian train: {(datetime.now() - start_time).total_seconds()} giây") #In thời gian train
```

Khai báo callback là EarlyStopping có nhiệm vụ giám sát giá trị loss, patience 50 là nếu giá trị loss không thay đổi qua 50 epoch thì kết thúc train, restore\_best\_weight True sẽ trả về mô hình có loss thấp nhất

Thực hiện train mô hình, với epochs là 1000 không shuffle cùng với callback. Sau khi train xong sẽ in ra thời gian train mô hình.

```
test_predict = model.predict(X_test, verbose=0)
test_predict
```

Thực hiện dự đoán mô hình thông qua tập test.

```
fig, axs = plt.subplots(4, figsize = (20,10)) #Tạo plot

dates = dataframe_w['date'][-(n_steps_in):] #Lấy cột date
titles = ['Giá mở cửa', 'Giá trần', 'Giá sàn', 'Giá đóng cửa'] #Khai báo mảng tiêu đề

for idx, feature in enumerate(features[1:]): #Lặp vẽ 4 tương ứng với các feature được chọn trừ date
    axs[idx].plot(dates, dataframe_w[feature][-(n_steps_in):]) #Vẽ đường thực tế
    axs[idx].plot(dates, test_predict[:,idx]) #Vẽ đường dự đoán
    axs[idx].legend([f'true {feature}', f'predict {feature}']) #Tạo chú thích
    axs[idx].set_title(titles[idx]) #Tạo tiêu đề

plt.tight_layout() #Giảm lề
plt.show() #In hình
```

Sau khi dự đoán tập test, ta thực hiện trực quan hóa chúng để quan sát dự đoán của mô hình có chính xác không. Tạo figure với 4 plot, sau đó lấy cột date ứng với chiều dài của tập test. Tạo mảng titles chứa tiêu đề các plot.

Tiếp theo, ta thực hiện lặp enumerate features ngoại trừ date ra và vẽ đường thực tế, đường dự đoán. Mỗi vòng lặp sẽ vẽ plot cho 1 features và thứ tự plot ứng với idx sau đó set tiêu đề cho plot.

```
predicts = list(dataframe[features[1:]][-7:].to_numpy())
#Lấy dữ liệu 7 tuần trước

while(len(predicts) < 7+7): #Thực hiện vào lặp dự đoán 7 tuần tiếp theo
    X_predicts = np.array([predicts[-
7:]] #Lấy 7 tuần trước từ mảng predicts
    predict = model.predict(X_predicts, verbose=0) #Thực hiện dự đoán
    predicts.append(predict[0]) #Thêm dự đoán vào mảng predicts để tiếp tục dự đoán

predicts = np.array(predicts) #Chuyển predicts lại thành np array
predicts[7:] #In dự đoán
```

Tiếp tục, ta sẽ thực hiện dự đoán 7 tuần trong tương lai bằng cách sử dụng 7 tuần dữ liệu có sẵn (7 tuần này ứng với n\_steps\_in).

Ta sẽ tạo mảng X\_predicts chứa dữ liệu 7 tuần trước đó và thực hiện dự đoán sau đó thêm lại vào mảng predict và lặp lại như thế 7 lần.

```
fig, axs = plt.subplots(4,figsize = (20,10)) #Tạo plot

dates = dataframe_w['date'][-(n_steps_in):] #Lấy cột date
dates = dates_add(dates, 7, distance=7) #Thêm vào mảng date 7 tuần
titles = ['Giá mở cửa', 'Giá trần', 'Giá sàn', 'Giá đóng cửa'] #Khai báo mảng tiêu đề

for idx, feature in enumerate(features[1:]): #Lặp vẽ 4 tương ứng với các feature được chọn trừ date
    axs[idx].plot(dates[:7], predicts[:7,idx]) #Vẽ đường thực tế
    axs[idx].plot(dates, np.concatenate((test_predict[:,idx], predicts[7:,idx]))) #Vẽ đường dự đoán
```

```

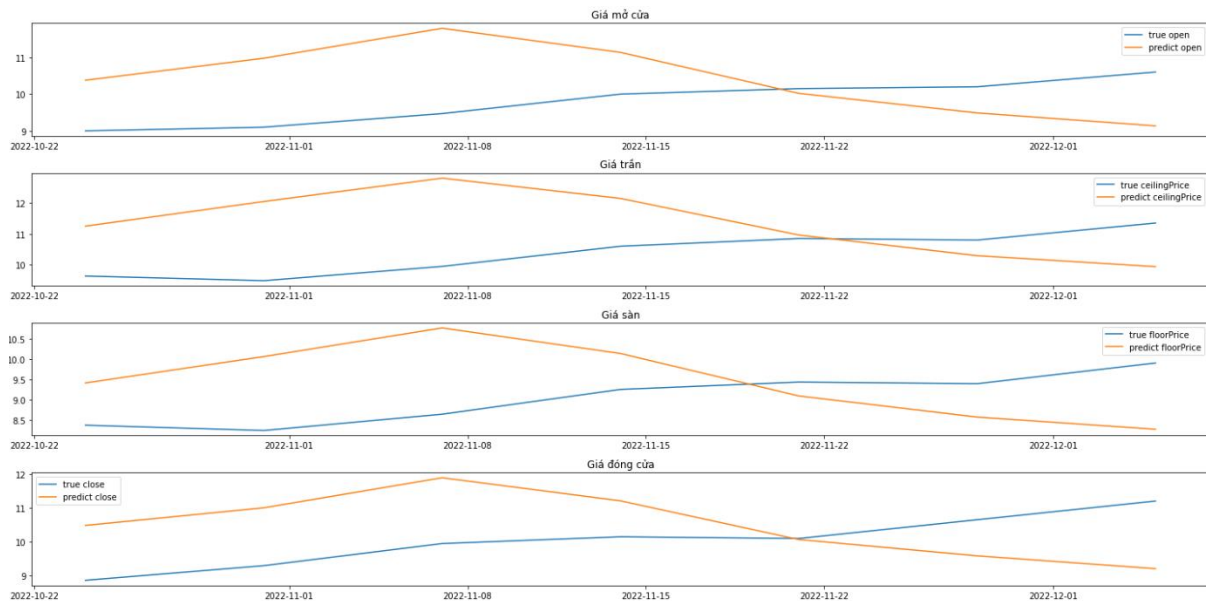
    axs[idx].legend([f'true {feature}', f'predict {feature}']) #Tạo chú th
ích
    axs[idx].set_title(titles[idx]) #Tạo tiêu đề

plt.tight_layout() #Giảm lề
plt.show() #In hình

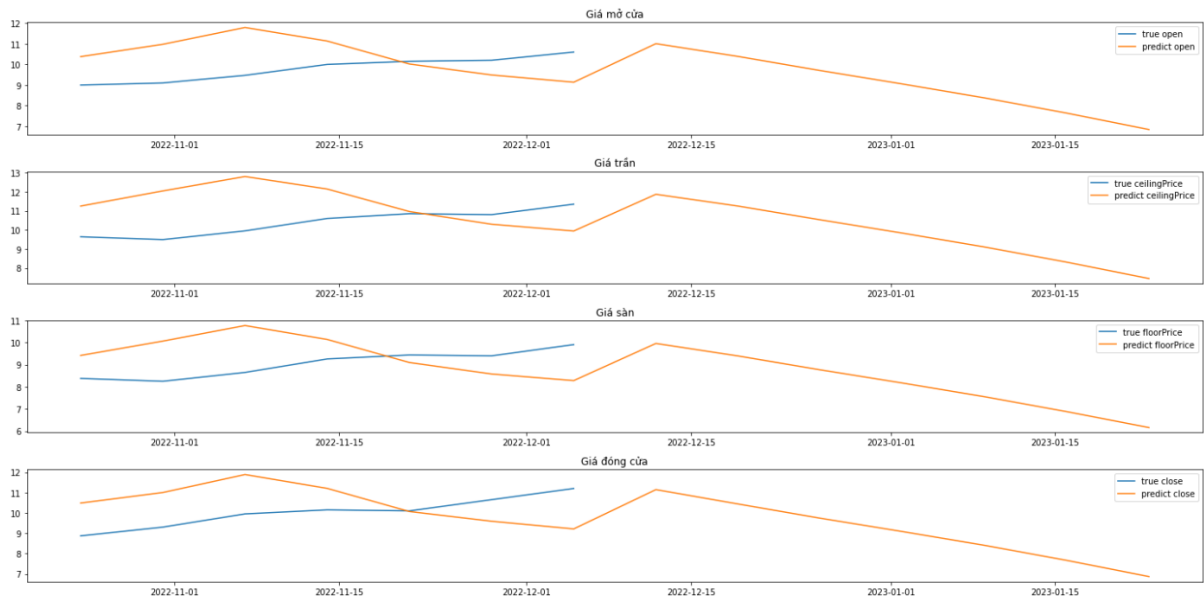
```

Sau khi dự đoán tập 7 tuần tiếp theo, ta thực hiện trực quan hóa chúng để quan sát dự đoán của mô hình. Tạo figure với 4 plot, sau đó lấy cột date ứng với chiều dài của tập test. Sau đó thêm vào mảng day 7 tuần tiếp theo. Tạo mảng titles chứa tiêu đề các plot.

Tiếp theo, ta thực hiện lặp enumerate features ngoại trừ date ra và vẽ đường thực tế, đường dự đoán. Mỗi vòng lặp sẽ vẽ plot cho 1 features và thứ tự plot ứng với idx sau đó set tiêu đề cho plot.



*Hình 3.2.3.1c: Tập test*



Hình 3.2.3.1d: Dự đoán 7 tuần tiếp theo sử dụng chống overfitting cho mô hình RNN

### 3.2.3.2 MLP

 **Dữ liệu ngày:**

```
n_steps_in = 14 #Lấy {n_steps_in} ngày trước để dự đoán ngày hôm nay

dataset = dataframe[features[1:]].to_numpy() #Chuyển dataframe thành mảng
X, y = split_sequences(dataset, n_steps_in) #Chia dataset thành X và y theo {n_steps_in}

n_input = X.shape[1] * X.shape[2] #Tính chiều dài input
X = X.reshape((X.shape[0], n_input)) #Chuyển X shape từ (?, 7, 4) thành (?, 28)

n_output = y.shape[1] * y.shape[2] #Tính chiều dài output
y = y.reshape((y.shape[0], n_output)) #Chuyển y shape từ (?, 7, 4) thành (?, 28)

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=len(X) -
n_steps_in, shuffle = False) #Chia dữ liệu tập train và test trong đó test chứa {n_steps_in} để thực hiện dự đoán
```

Thực hiện tiền xử lý dữ liệu như tạo X và y bằng hàm `split_sequences`. Tiếp đó tính chiều dài của input và output lần lượt gán vào `n_input`, `n_output` để sử dụng cho quá trình xây dựng mô hình.

Cuối cùng sử dụng `train_test_split` với tham số `shuffle` là `false` thực hiện chia dữ liệu thành tập test và tập train.

```
model = Sequential() #Khởi tạo model
model.add(Dense(100, activation='relu', input_dim=n_input, kernel_regularizer=l2(0.01), bias_regularizer=l2(0.01))) #Thêm lớp Dense với n_input là số lượng tham số đầu vào và regularizer
model.add(Dropout(0.01)) #Thêm lớp Drop Out
model.add(Dense(n_output)) #Thêm lớp Dense với n_output là số lượng tham số đầu ra
model.compile(optimizer='adam', loss='mse')
```

Ta tiến hành xây dựng mô hình sử dụng `Sequential`, với lớp đầu tiên là `Dense` với số 100 là số unit xử lý của lớp đó cùng với hàm activation `relu`, `kernel_regularizer` và `bias_regularizer` có cùng tham số `l2(0.01)` sẽ giúp cho mô hình đơn giản hơn. Sau đó là lớp `Dropout` với rate là 0.01 để bỏ bớt trọng số giúp mô hình đơn giản hơn. Tiếp đó lớp `Dense` sẽ dùng để trả về kết quả, nhận tham số `n_output` nó sẽ trả về số lượng output tương ứng.

Sau đó ta compile mô hình với optimizer `adam` và hàm loss là `mae` (mean squared error)

```
callback = EarlyStopping(monitor='loss', patience=50, restore_best_weights=True) #EarlyStop nếu 50 epoch không thay đổi loss trả về mô hình với loss thấp nhất

start_time = datetime.now() #Lưu thời gian bắt đầu train
model.fit(X_train, y_train, epochs=1000, shuffle=False, verbose=2, callbacks=[callback]) #Train mô hình
print(f"Thời gian train: {(datetime.now() - start_time).total_seconds()} giây") #In ra thời gian train
```

Khai báo callback là `EarlyStopping` có nhiệm vụ giám sát giá trị loss, `patience` 50 là nếu giá trị loss không thay đổi qua 50 epoch thì kết thúc train, `restore_best_weight` `True` sẽ trả về mô hình có loss thấp nhất

Thực hiện train mô hình, với epochs là 1000 không shuffle cùng với callback. Sau khi train xong sẽ in ra thời gian train mô hình.

```
test_predict = model.predict(X_test, verbose=0)
test_predict
```

Thực hiện dự đoán mô hình thông qua tập test.

```
fig, axs = plt.subplots(4, figsize = (20,10)) #Tạo plot

dates = dataframe['date'][-(n_steps_in):] #Lấy cột date
titles = ['Giá mở cửa', 'Giá trần', 'Giá sàn', 'Giá đóng cửa'] #Khai báo
mảng tiêu đề

for idx, feature in enumerate(features[1:]): #Lặp vẽ 4 tương ứng với các
feature được chọn trừ date
    axs[idx].plot(dates, dataframe[feature][-
(n_steps_in):]) #Vẽ đường thực tế
    axs[idx].plot(dates, test_predict[:,idx]) #Vẽ đường dự đoán
    axs[idx].legend([f'true {feature}', f'predict {feature}']) #Tạo chú th
ích
    axs[idx].set_title(titles[idx]) #Tạo tiêu đề

plt.tight_layout() #Giảm lề
plt.show() #In hình
```

Sau khi dự đoán tập test, ta thực hiện trực quan hóa chúng để quan sát dự đoán của mô hình có chính xác không. Tạo figure với 4 plot, sau đó lấy cột date ứng với chiều dài của tập test. Tạo mảng titles chứa tiêu đề các plot.

Tiếp theo, ta thực hiện lặp enumerate features ngoại trừ date ra và vẽ đường thực tế, đường dự đoán. Mỗi vòng lặp sẽ vẽ plot cho 1 features và thứ tự plot ứng với idx sau đó set tiêu đề cho plot.

```
predicts = list(dataframe[features[1:]][-14:].to_numpy())
#Lấy dữ liệu 14 ngày trước
```

```

while(len(predicts) < 14+14): #Thực hiện vào lặp dự đoán 14 ngày tiếp theo
    X_predicts = np.array([predicts[-14:]] #Lấy 14 ngày trước từ mảng predicts
    predict = model.predict(X_predicts, verbose=0) #Thực hiện dự đoán
    predicts.append(predict[0]) #Thêm dự đoán vào mảng predicts để tiếp tục dự đoán

predicts = np.array(predicts) #Chuyển predicts lại thành np array
predicts

```

Tiếp tục, ta sẽ thực hiện dự đoán 14 ngày trong tương lai bằng cách sử dụng 14 ngày dữ liệu có sẵn (14 ngày này ứng với `n_steps_in`).

Ta sẽ tạo mảng `X_predicts` chứa dữ liệu 14 ngày trước đó và thực hiện dự đoán sau đó thêm lại vào mảng `predict` và lặp lại như thế 14 lần.

```

fig, axs = plt.subplots(4,figsize = (20,10)) #Tạo plot

dates = dataframe['date'][-(n_steps_in):] #Lấy cột date
dates = dates_add(dates, 14) #Thêm vào mảng date 14 ngày
titles = ['Giá mở cửa', 'Giá trần', 'Giá sàn', 'Giá đóng cửa'] #Khai báo mảng tiêu đề

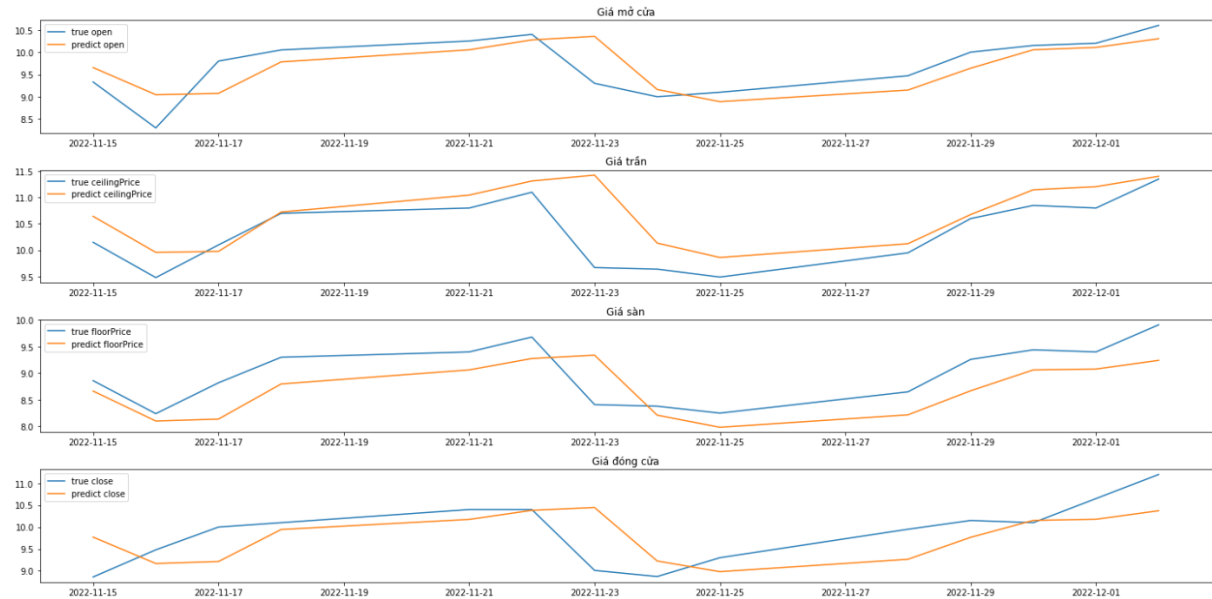
for idx, feature in enumerate(features[1:]): #Lặp vẽ 4 tương ứng với các feature được chọn trừ date
    axs[idx].plot(dates[:14], predicts[:14,idx]) #Vẽ đường thực tế
    axs[idx].plot(dates, np.concatenate((test_predict[:,idx], predicts[14:,idx]))) #Vẽ đường dự đoán
    axs[idx].legend([f'true {feature}', f'predict {feature}']) #Tạo chú thích
    axs[idx].set_title(titles[idx]) #Tạo tiêu đề

plt.tight_layout() #Giảm lề
plt.show() #In hình

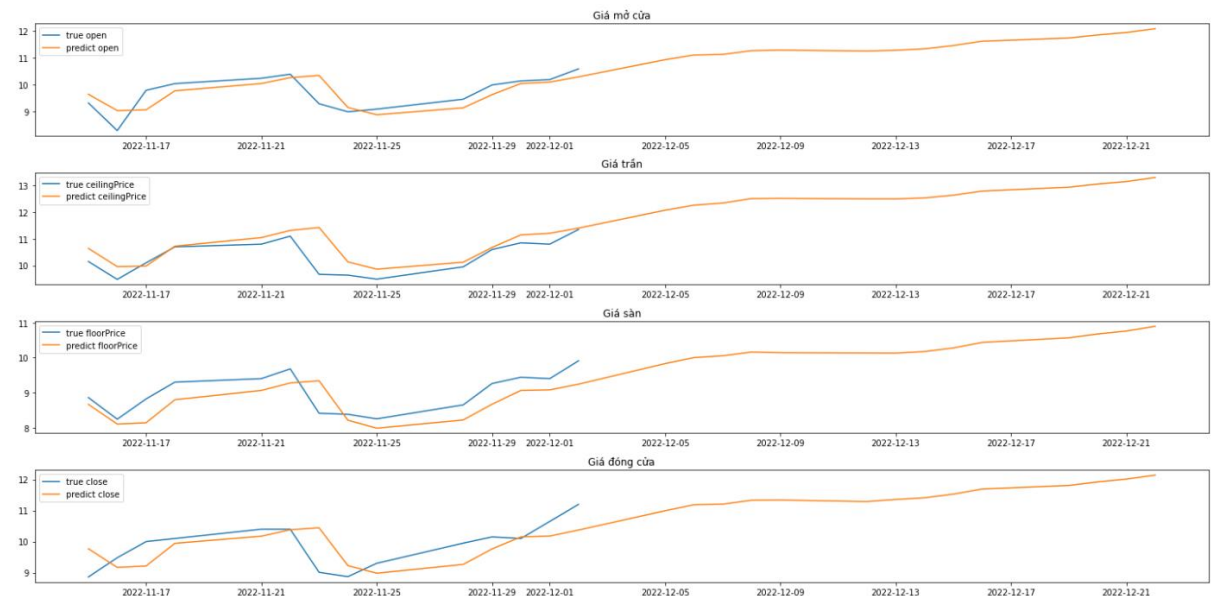
```

Sau khi dự đoán tập 14 ngày tiếp theo, ta thực hiện trực quan hóa chúng để quan sát dự đoán của mô hình. Tạo figure với 4 plot, sau đó lấy cột date ứng với chiều dài của tập test. Sau đó thêm vào mảng day 14 ngày tiếp theo. Tạo mảng titles chứa tiêu đề các plot.

Tiếp theo, ta thực hiện lặp enumerate features ngoại trừ date ra và vẽ đường thực tế, đường dự đoán. Mỗi vòng lặp sẽ vẽ plot cho 1 features và thứ tự plot ứng với idx sau đó set tiêu đề cho plot.



Hình 3.2.3.2a: Tập test



Hình 3.2.3.2b: Dự đoán 14 ngày tiếp theo sử dụng chống overfitting cho mô hình MLP



### **Dữ liệu tuần:**

```
n_steps_in = 7 #Lấy {n_steps_in} ngày trước để dự đoán ngày hôm nay

dataset = dataframe[features[1:]].to_numpy() #Chuyển dataframe thành mảng
X, y = split_sequences(dataset, n_steps_in) #Chia dataset thành X và y theo {n_steps_in}

n_input = X.shape[1] * X.shape[2] #Tính chiều dài input
X = X.reshape((X.shape[0], n_input)) #Chuyển X shape từ (?, 7, 4) thành (?, 28)
n_output = y.shape[1] * y.shape[2] #Tính chiều dài output
y = y.reshape((y.shape[0], n_output)) #Chuyển y shape từ (?, 7, 4) thành (?, 28)

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=len(X) -
n_steps_in, shuffle = False) #Chia dữ liệu tập train và test trong đó test chứa {n_steps_in} để thực hiện dự đoán
```

Phần này sẽ thực hiện tiền xử lý dữ liệu như tạo X và y bằng hàm `split_sequences`. Tiếp đó tính chiều dài của input và output lần lượt gán vào `n_input`, `n_output` để sử dụng cho quá trình xây dựng mô hình.

Cuối cùng sử dụng `train_test_split` với tham số `shuffle` là `false` thực hiện chia dữ liệu thành tập test và tập train.

```
model = Sequential() #Khởi tạo model
model.add(Dense(100, activation='relu', input_dim=n_input, kernel_regularizer=l2(0.01), bias_regularizer=l2(0.01))) #Thêm lớp Dense với n_input là số lượng tham số đầu vào và regularizer
model.add(Dropout(0.01)) #Thêm lớp Drop Out
model.add(Dense(n_output)) #Thêm lớp Dense với n_output là số lượng tham số đầu ra
model.compile(optimizer='adam', loss='mse')
```

Ta tiến hành xây dựng mô hình sử dụng `Sequential`, với lớp đầu tiên là `Dense` với số 100 là số unit xử lý của lớp đó cùng với hàm `activation relu`, `kernel_regularizer` và `bias_regularizer` có cùng tham số `l2(0.01)` sẽ giúp cho mô hình đơn giản hơn. Sau đó là

lớp Dropout với rate là 0.01 để bỏ bớt trọng số giúp mô hình đơn giản hơn. Tiếp đó lớp Dense sẽ dùng để trả về kết quả, nhận tham số `n_output` nó sẽ trả về số lượng output tương ứng.

Sau đó ta compile mô hình với optimizer adam và hàm loss là mae (mean squared error)

```
callback = EarlyStopping(monitor='loss', patience=50, restore_best_weights=True) #EarlyStop nếu 50 epoch không thay đổi loss trả về mô hình với loss thấp nhất

start_time = datetime.now() #Lưu thời gian bắt đầu train
model.fit(X_train, y_train, epochs=1000, shuffle=False, verbose=2, callbacks=[callback]) #Train mô hình
print(f"Thời gian train: {(datetime.now() - start_time).total_seconds()} giây") #In ra thời gian train
```

Khai báo callback là EarlyStopping có nhiệm vụ giám sát giá trị loss, patience 50 là nếu giá trị loss không thay đổi qua 50 epoch thì kết thúc train, `restore_best_weight True` sẽ trả về mô hình có loss thấp nhất

Thực hiện train mô hình, với epochs là 1000 không shuffle cùng với callback. Sau khi train xong sẽ in ra thời gian train mô hình.

```
test_predict = model.predict(X_test, verbose=0)
test_predict
```

Thực hiện dự đoán mô hình thông quá tập test.

```
fig, axs = plt.subplots(4, figsize = (20,10)) #Tạo plot

dates = dataframe_w['date'][-(n_steps_in):] #Lấy cột date
titles = ['Giá mở cửa', 'Giá trần', 'Giá sàn', 'Giá đóng cửa'] #Khai báo mảng tiêu đề

for idx, feature in enumerate(features[1:]): #Lập vẽ 4 tương ứng với các feature được chọn trừ date
```

```

    axs[idx].plot(dates, dataframe_w[feature] [-
(n_steps_in):]) #Vẽ đường thực tế
    axs[idx].plot(dates, test_predict[:,idx]) #Vẽ đường dự đoán
    axs[idx].legend([f'true {feature}', f'predict {feature}']) #Tạo chú th
ích
    axs[idx].set_title(titles[idx]) #Tạo tiêu đề

plt.tight_layout() #Giảm lề
plt.show() #In hình

```

Sau khi dự đoán tập test, ta thực hiện trực quan hóa chúng để quan sát dự đoán của mô hình có chính xác không. Tạo figure với 4 plot, sau đó lấy cột date ứng với chiều dài của tập test. Tạo mảng titles chứa tiêu đề các plot.

Tiếp theo, ta thực hiện lặp enumerate features ngoại trừ date ra và vẽ đường thực tế, đường dự đoán. Mỗi vòng lặp sẽ vẽ plot cho 1 features và thứ tự plot ứng với idx sau đó set tiêu đề cho plot.

```

predicts = list(dataframe[features[1:]] [-7:].to_numpy())
#Lấy dữ liệu 7 tuần trước

while(len(predicts) < 7+7): #Thực hiện vào lặp dự đoán 7 tuần tiếp theo
    X_predicts = np.array(predicts [-
7:]) #Lấy 7 ngày trước từ mảng predicts
    X_predicts = X_predicts.reshape((1, n_input)) #Chuyển từ dạng (?, {n_s
teps_in}, 4) sang (?, {n_steps_in}*4)

    predict = model.predict(X_predicts, verbose=0) #Thực hiện dự đoán
    predicts.append(predict[0]) #Thêm dự đoán vào mảng predicts để tiếp tậ
c dự đoán

predicts = np.array(predicts) #Chuyển predicts lại thành np array
predicts

```

Tiếp tục, ta sẽ thực hiện dự đoán 7 tuần trong tương lai bằng cách sử dụng 7 tuần dữ liệu có sẵn (7 tuần này ứng với n\_steps\_in).

Ta sẽ tạo mảng X\_predicts chứa dữ liệu 7 tuần trước đó và thực hiện dự đoán sau đó thêm lại vào mảng predict và lặp lại như thế 7 lần.

```

fig, axs = plt.subplots(4,figsize = (20,10)) #Tạo plot

dates = dataframe_w['date'][-(n_steps_in):] #Lấy cột date
dates = dates_add(dates, 7, distance=7) #Thêm vào mảng date 7 tuần
titles = ['Giá mở cửa', 'Giá trần', 'Giá sàn', 'Giá đóng cửa'] #Khai báo
mảng tiêu đề

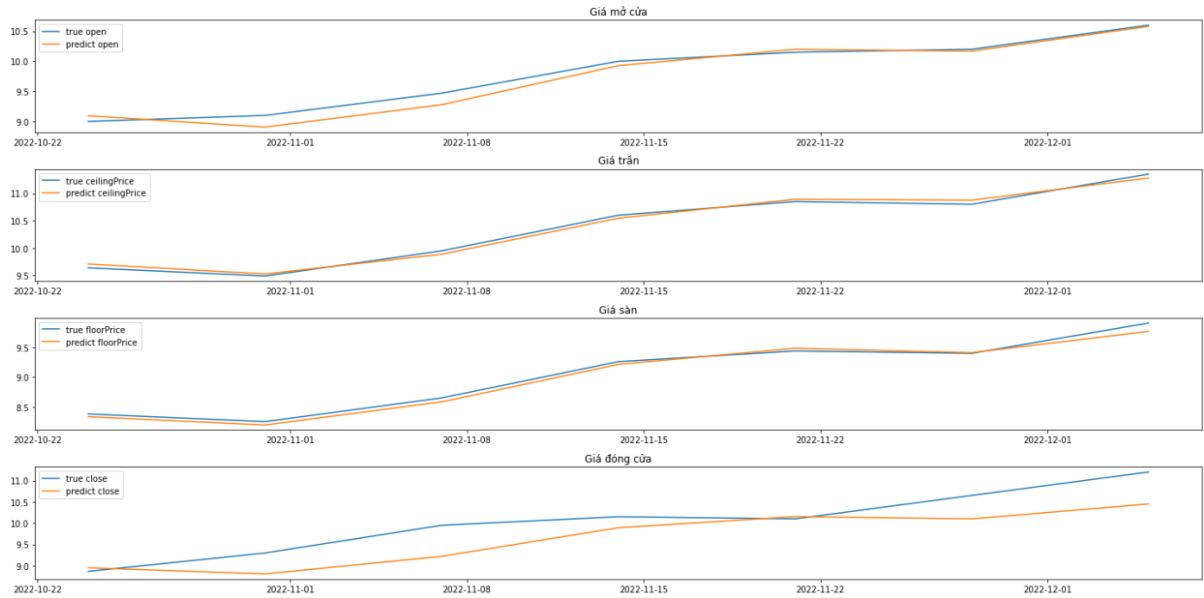
for idx, feature in enumerate(features[1:]): #Lặp vẽ 4 tương ứng với các
feature được chọn trừ date
    axs[idx].plot(dates[:7], predicts[:7,idx]) #Vẽ đường thực tế
    axs[idx].plot(dates, np.concatenate((test_predict[:,idx], predicts[7:,
idx]))) #Vẽ đường dự đoán
    axs[idx].legend([f'true {feature}', f'predict {feature}']) #Tạo chú th
ích
    axs[idx].set_title(titles[idx]) #Tạo tiêu đề

plt.tight_layout() #Giảm lề
plt.show() #In hình

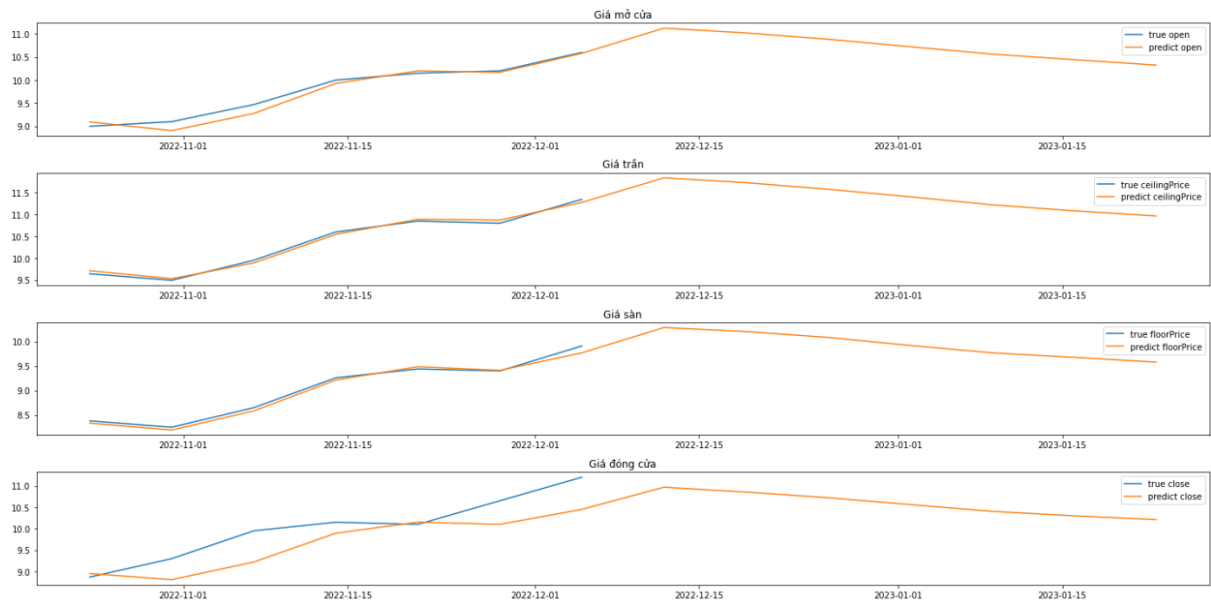
```

Sau khi dự đoán tập 7 tuần tiếp theo, ta thực hiện trực quan hóa chúng để quan sát dự đoán của mô hình. Tạo figure với 4 plot, sau đó lấy cột date ứng với chiều dài của tập test. Sau đó thêm vào mảng day 7 tuần tiếp theo. Tạo mảng titles chứa tiêu đề các plot.

Tiếp theo, ta thực hiện lặp enumerate features ngoại trừ date ra và vẽ đường thực tế, đường dự đoán. Mỗi vòng lặp sẽ vẽ plot cho 1 features và thứ tự plot ứng với idx sau đó set tiêu đề cho plot.



Hình 3.2.3.2c: Tập test



Hình 3.2.3.2d: Dự đoán 7 tuần tiếp theo sử dụng chống overfitting cho mô hình MLP

### 3.3 Tìm ra tập đặc trưng quan trọng

 **Hàm sử dụng:**

```
def MLP(chose features, n steps in):
```

```

dataset = dataframe[chose_features].to_numpy() #Chuyển dataframe thành
mảng
X, y = split_sequences(dataset, n_steps_in) #Chia dataset thành X và y
theo {n_steps_in}

n_input = X.shape[1] * X.shape[2] #Tính chiều dài input
X = X.reshape((X.shape[0], n_input)) #Chuyển X shape từ (?, {n_steps_in}, {len(chose_features)}) thành (?, {n_steps_in}*{len(chose_features)})
n_output = y.shape[1] * y.shape[2] #Tính chiều dài output
y = y.reshape((y.shape[0], n_output)) #Chuyển y shape từ (?, {n_steps_in}, {len(chose_features)}) thành (?, {n_steps_in}*{len(chose_features)})

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=len(X)-
n_steps_in, shuffle = False) #Chia dữ liệu tập train và test trong đó test chứa {n_steps_in} để thực hiện dự đoán

model = Sequential() #Khởi tạo model
model.add(Dense(100, activation='relu', input_dim=n_input, kernel_regularizer=l2(0.01), bias_regularizer=l2(0.01))) #Thêm lớp Dense với n_input là số lượng tham số đầu vào và regularizer
model.add(Dropout(0.01)) #Thêm lớp Drop Out
model.add(Dense(n_output)) #Thêm lớp Dense với n_output là số lượng tham số đầu ra
model.compile(optimizer='adam', loss='mse')

callback = EarlyStopping(monitor='loss', patience=50, restore_best_weights=True) #EarlyStop nếu 50 epoch không thay đổi loss trả về mô hình với loss thấp nhất

start_time = datetime.now() #Lưu thời gian bắt đầu train
model.fit(X_train, y_train, epochs=1000, shuffle=False, verbose=2, callbacks=[callback]) #Train mô hình
print(f"Thời gian train: {(datetime.now()-start_time).total_seconds()} giây") #In thời gian train
return model, X_test

def PredictNDrawGraph(model, chose_features, n_steps_in, X_test):
    test_predict = model.predict(X_test, verbose=0) #Dự đoán tập test
    predicts = list(dataframe[chose_features][:-n_steps_in:].to_numpy()) #Lấy dữ liệu {n_steps_in} ngày trước

```

```

    while(len(predicts) < n_steps_in*2): #Thực hiện vào lặp dự đoán {n_steps_in} ngày tiếp theo
        X_predicts = np.array(predicts[-n_steps_in:]) #Lấy {n_steps_in} ngày trước từ mảng predicts
        X_predicts = X_predicts.reshape((1, len(chose_features)*n_steps_in))
        #Chuyển từ dạng (?, {n_steps_in}, {len(chose_features)}) sang (?, {n_steps_in}*{len(chose_features)})

        predict = model.predict(X_predicts, verbose=0) #Thực hiện dự đoán
        predicts.append(predict[0]) #Thêm dự đoán vào mảng predicts để tiếp tục dự đoán

    predicts = np.array(predicts) #Chuyển predicts lại thành np array

    fig, axs = plt.subplots(len(chose_features), 1, squeeze=False, figsize=(20,10)) #Tạo plot
    axs = axs.reshape((len(chose_features),))

    dates = dataframe['date'][-(n_steps_in):] #Lấy cột date
    dates = dates_add(dates, n_steps_in) #Thêm vào mảng date
    titles = ['Giá mở cửa', 'Giá trần', 'Giá sàn', 'Giá đóng cửa'] #Khai báo mảng tiêu đề
    chose_titles = [titles[features[1:].index(chose_feature)] for chose_feature in chose_features]

    for idx, feature in enumerate(chose_features): #Lặp vẽ 4 tương ứng với các feature được chọn trừ date
        axs[idx].plot(dates[:n_steps_in], predicts[:n_steps_in,idx]) #Vẽ đường thực tế
        axs[idx].plot(dates, np.concatenate((test_predict[:,idx], predicts[n_steps_in:,idx]))) #Vẽ đường dự đoán
        axs[idx].legend([f'true {feature}', f'predict {feature}']) #Tạo chú thích
        axs[idx].set_title(chose_titles[idx]) #Tạo tiêu đề

    plt.tight_layout() #Giảm lề
    plt.show() #In hình

```

Hàm *MLP()* sẽ lại tổng hợp quá trình tiền xử lý và train mô hình đã có chống overfitting. Sử dụng 2 tham số đầu vào *chosen\_features* là mảng chứa những *feature*

chọn ra để huấn luyện và  $n\_steps\_in$  là input vào để dự đoán output. Hàm  $MLP()$  sẽ trả về mô hình và tập test.

Hàm  $PredictNDrawGraph()$  tổng hợp quá trình dự đoán 14 ngày tiếp theo và vẽ hình minh họa. Với 4 tham số là  $model$  là mô hình đã train,  $chosen\_features$  là mảng chứa feature được chọn ra huấn luyện,  $n\_steps\_in$  số ngày sẽ dự đoán và  $X\_test$  nhận vào tập test.

#### **Tập close:**

```
chose_features = ['close'] #Chọn 2 feature clsóe
n_steps_in = 14 #Số ngày trước để dự đoán hôm sau

model, X_test = MLP(chose_features, n_steps_in) #Train mô hình và trả về
tập test
```

Chọn 1 *feature* để huấn luyện và số ngày để input vào là 14. Sau đó thực hiện train

```
PredictNDrawGraph(model, chose_features, n_steps_in, X_test) #Vẽ hình dự
đoán 14 ngày tiếp theo
```

Sau khi train xong, gọi hàm này để thực hiện vẽ hình

#### **Tập open, close:**

```
chose_features = ['open', 'close'] #Chọn 2 feature là open và close
n_steps_in = 14 #Số ngày trước để dự đoán hôm sau

model, X_test = MLP(chose_features, n_steps_in) #Train mô hình và trả về
tập test
```

Chọn 2 *feature* để huấn luyện và số ngày để input vào là 14. Sau đó thực hiện train

```
PredictNDrawGraph(model, chose_features, n_steps_in, X_test) #Vẽ hình dự
đoán 14 ngày tiếp theo
```

Sau khi train xong, gọi hàm này để thực hiện vẽ hình

#### **Tập ceilingPrice, floorPrice, close:**



```

chose_features = ['ceilingPrice', 'floorPrice', 'close'] #Chọn 2 feature
là ceilingPrice, floorPrice và close
n_steps_in = 14 #Số ngày trước để dự đoán hôm sau

model, X_test = MLP(chose_features, n_steps_in) #Train mô hình và trả về
tập test

```

Chọn 3 *feature* để huấn luyện và số ngày để input vào là 14. Sau đó thực hiện train

```

PredictNDrawGraph(model, chose_features, n_steps_in, X_test) #Vẽ hình dự
đoán 14 ngày tiếp theo

```

Sau khi train xong, gọi hàm này để thực hiện vẽ hình

## CHƯƠNG 4 – HỌC SÂU

### 4.1 Khái niệm về học sâu

Học sâu là một chức năng của trí tuệ nhân tạo (AI), bắt chước hoạt động của bộ não con người trong việc xử lý dữ liệu và tạo ra các mẫu để sử dụng cho việc ra quyết định. Học sâu đã phát triển cùng với thời đại kỹ thuật số, dẫn đến sự bùng nổ dữ liệu ở nhiều dạng khác nhau từ mọi nơi trên thế giới. Những dữ liệu này, được gọi đơn giản là dữ liệu lớn, được lấy từ các nguồn như mạng xã hội, công cụ tìm kiếm trên internet, nền tảng thương mại điện tử hoặc rạp chiếu phim trực tuyến.

### 4.2 So sánh học sâu với học máy

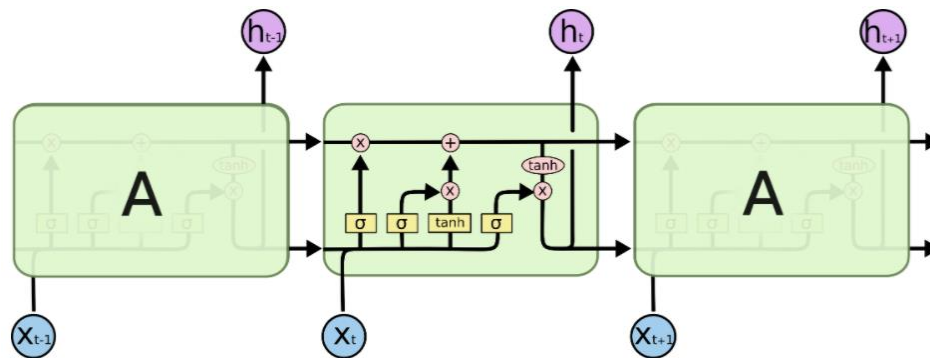
<i>Học sâu</i>	<i>Học máy</i>
<ul style="list-style-type: none"> <li>- Là một tập hợp con của học máy</li> <li>- Học sâu tự động hóa phần lớn phần trích xuất tính năng của quy trình, loại bỏ một số sự can thiệp thủ công của con người</li> <li>- Nó cũng cho phép sử dụng các tập dữ liệu lớn. Khả năng này sẽ đặc biệt thú vị khi chúng ta bắt đầu khám phá việc sử dụng dữ liệu phi cấu trúc nhiều hơn, đặc biệt vì 80-90% dữ liệu của một tổ chức được ước tính là không có cấu trúc.</li> </ul>	<ul style="list-style-type: none"> <li>- Học máy cổ điển, hoặc “không sâu”, phụ thuộc nhiều hơn vào sự can thiệp của con người để học</li> <li>- Các chuyên gia con người xác định thứ bậc của các tính năng để hiểu sự khác biệt giữa các đầu vào dữ liệu, thường yêu cầu nhiều dữ liệu có cấu trúc hơn để tìm hiểu.</li> <li>- Các chuyên gia con người xác định thứ bậc của các tính năng để hiểu sự khác biệt giữa các đầu vào dữ liệu, thường yêu cầu nhiều dữ liệu có cấu trúc hơn để tìm hiểu và nó có thể tự động xác định tập hợp các tính năng giúp phân biệt thứ này với thứ kia.</li> </ul>

### 4.3 Mô hình LSTM

### 4.3.1 Khái niệm về LSTM

Mạng Long Short Term Memory (LSTM) là dạng cải tiến của mạng RNN, một mạng tuần tự cho phép thông tin tồn tại. Nó có khả năng xử lý vấn đề gradient biến mất mà RNN phải đối mặt. Trong RNN, đầu ra từ bước cuối cùng được cung cấp làm đầu vào trong bước hiện tại. LSTM đã giải quyết vấn đề về sự phụ thuộc dài hạn của RNN, trong đó RNN không thể dự đoán từ được lưu trữ trong bộ nhớ dài hạn nhưng có thể đưa ra dự đoán chính xác hơn từ thông tin gần đây. Khi độ dài khoảng cách tăng lên, RNN không mang lại hiệu suất hiệu quả. LSTM theo mặc định có thể lưu giữ thông tin trong một khoảng thời gian dài. Nó được sử dụng để xử lý, dự đoán và phân loại trên cơ sở dữ liệu chuỗi thời gian.

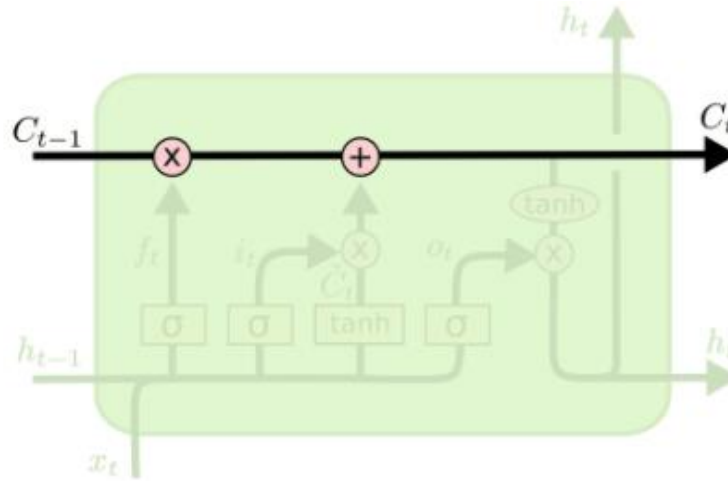
Mọi mạng hồi quy đều có dạng là một chuỗi các mô-đun lặp đi lặp lại của mạng nơ-ron. LSTM cũng có kiến trúc dạng chuỗi như vậy, nhưng các mô-đun trong nó có cấu trúc khác với mạng RNN chuẩn. Thay vì chỉ có một mạng nơ-ron, chúng có tới 4 tầng tương tác với nhau một cách rất đặc biệt.



Hình 4. Mô tả cấu trúc của LSTM

### 4.3.2 Cấu trúc của LSTM

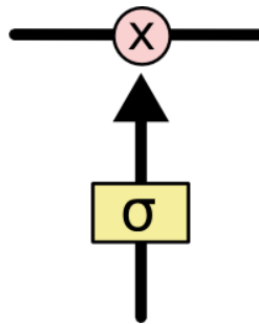
LSTM có cấu trúc chuỗi chứa bốn mạng lưới thần kinh và các khối bộ nhớ khác nhau được gọi là cells. Chìa khóa của LSTM là trạng thái của tế bào (cell-state) - chính đường là đường chạy thông qua của sơ đồ hình vẽ.



Hình Trạng thái tế bào

Trạng thái tế bào là một dạng giống như băng truyền. Nó chạy xuyên suốt tất cả các mắt xích (các nút mạng) và chỉ tương tác tuyến tính đôi chút. Vì vậy mà các thông tin có thể dễ dàng truyền đi thông suốt mà không sợ thay đổi.

Thông tin được giữ lại bởi các tế bào và các thao tác bộ nhớ được thực hiện bởi các cổng (gates). Các cổng là nơi sàng lọc thông tin đi qua nó, chúng được kết hợp bởi một tầng mạng sigmoid và một phép nhân.

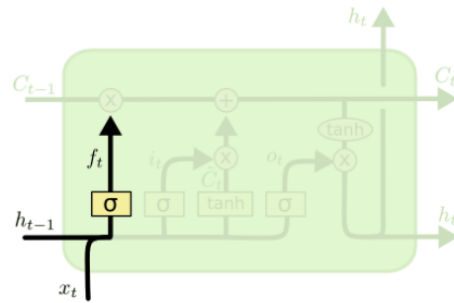


Hình Cổng

Tầng sigmoid sẽ cho đầu ra là một số trong khoảng  $[0,1]$ , mô tả có bao nhiêu thông tin có thể được thông qua. Khi đầu ra là 0 thì có nghĩa không cho thông tin nào qua cả, còn khi 1 thì có nghĩa là cho tất cả thông tin đi qua nó.

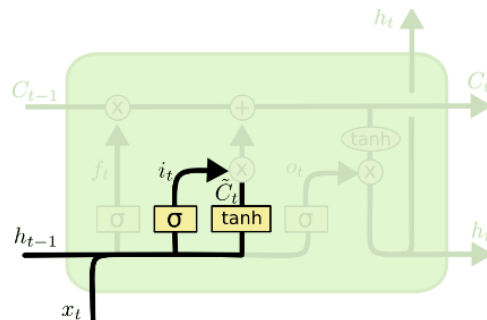
Có 3 cổng:

- Forget gate:** Thông tin không còn hữu ích trong trạng thái ô sẽ bị xóa bằng cổng forget. Hai đầu vào  $x_t$  (đầu vào tại thời điểm cụ thể) và  $h_{t-1}$  (đầu ra của ô trước đó) được đưa đến cổng và nhân với ma trận trọng số, sau đó cộng thêm độ lệch. Kết quả được chuyển qua một chức năng kích hoạt cung cấp đầu ra nhị phân. Nếu đối với một trạng thái ô cụ thể, đầu ra là 0, phần thông tin sẽ bị quên và đối với đầu ra 1, thông tin được giữ lại để sử dụng trong tương lai.



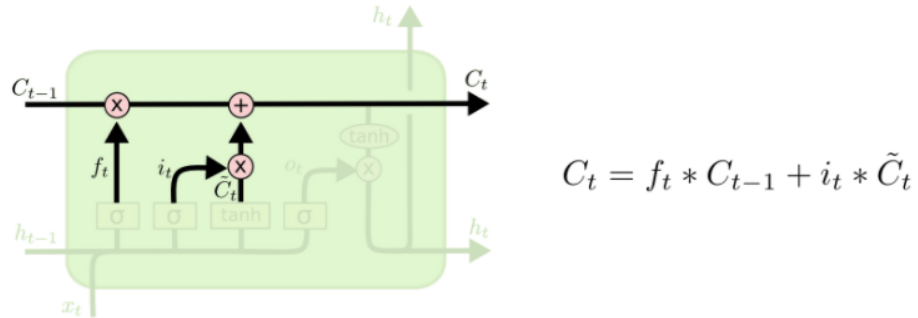
$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

- Input gate:** Việc bổ sung thông tin hữu ích vào trạng thái ô được thực hiện bởi cổng vào. Đầu tiên, thông tin được điều chỉnh bằng hàm sigmoid và lọc các giá trị cần nhớ tương tự như cổng quên sử dụng đầu vào  $h_{t-1}$  và  $x_t$ . Sau đó, một vector được tạo bằng cách sử dụng hàm  $\tanh$  cung cấp đầu ra từ -1 đến +1, chứa tất cả các giá trị có thể có từ  $h_{t-1}$  và  $x_t$ . Cuối cùng, các giá trị của vector và các giá trị quy định được nhân lên để thu được thông tin hữu ích.

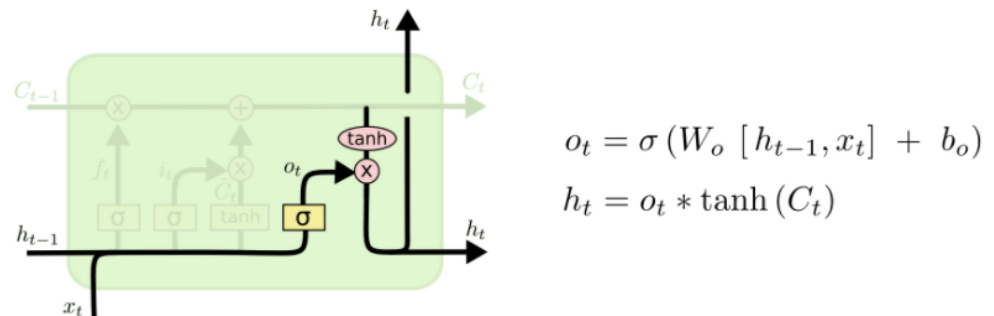


$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$



- Output gate: Nhiệm vụ trích xuất thông tin hữu ích từ trạng thái ô hiện tại để được trình bày dưới dạng đầu ra được thực hiện bởi cổng đầu ra. Đầu tiên, một vector được tạo bằng cách áp dụng hàm tanh trên ô. Sau đó, thông tin được điều chỉnh bằng cách sử dụng hàm sigmoid và lọc theo các giá trị cần nhớ bằng đầu vào  $h_{t-1}$  và  $x_t$ . Cuối cùng, các giá trị của vector và các giá trị quy định được nhân lên để gửi dưới dạng đầu ra và đầu vào cho ô tiếp theo.



### 4.3.3 Ứng dụng

Long short-term memory (LSTM) là một phần mở rộng của kiến trúc artificial recurrent neural network (RNN) được dùng trong lĩnh vực Deep learning. Vì vậy LSTM có các kết nối phản hồi. LSTM có thể xử lý không chỉ các điểm dữ liệu đơn lẻ (ví dụ như hình ảnh) mà còn toàn bộ chuỗi dữ liệu (như speech hay là video). Nó thường được dùng trong các bài toán như là phân loại chủ đề, xử lý ngôn ngữ tự nhiên, dự đoán giá cổ phiếu, dự đoán mực nước, dịch thuật....

### 4.3.4 Xây dựng mô hình

### **Dữ liệu ngày:**

```
n_steps_in = 14 #Lấy {n_steps_in} ngày trước để dự đoán ngày hôm nay

dataset = dataframe[features[1:]].to_numpy() #Chuyển dataframe thành mảng
X, y = split_sequences(dataset, n_steps_in) #Chia dataset thành X và y theo {n_steps_in}

n_input = X.shape[1] * X.shape[2] #Tính chiều dài input
n_output = y.shape[1] * y.shape[2] #Tính chiều dài output

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=len(X)-n_steps_in, shuffle = False)
#Chia dữ liệu tập train và test trong đó test chứa {n_steps_in} để thực hiện dự đoán
```

Phần này sẽ thực hiện tiền xử lý dữ liệu như tạo X và y bằng hàm `split_sequences`.

Tiếp đó tính chiều dài của input và output lần lượt gán vào `n_input`, `n_output` để sử dụng cho quá trình xây dựng mô hình.

Và cuối cùng sử dụng `train_test_split` với tham số `shuffle` là `false` thực hiện chia dữ liệu thành tập test và tập train.

```
model = Sequential() #Khởi tạo model
model.add(LSTM(4, input_shape=(14, 4), kernel_regularizer=l2(0.01), recurrent_regularizer=l2(0.01), bias_regularizer=l2(0.01))) #Thêm lớp Simple RNN với regularizer
model.add(Dropout(0.1)) #Thêm lớp Drop Out
model.add(Dense(n_output)) #Thêm lớp Dense với n_output là số lượng tham số đầu ra
model.compile(optimizer='adam', loss='mae')
```

Ta tiến hành xây dựng mô hình sử dụng `Sequential`, với lớp đầu tiên là `LSTM` với số 4 là số unit xử lý của lớp đó, cùng với `kernel_regularizer`, `recurrent_regularizer`, `bias_regularizer` có chung tham số `l2(0.01)` giúp cho mô hình trở nên đơn giản hơn. Sau đó là lớp `Dropout`, với tham số 0.1 là rate loại bỏ trọng số giúp cho mô hình đơn giản hơn. Tiếp đó lớp `Dense` sẽ dùng để trả về kết quả, nhận tham số `n_output` nó sẽ trả về số lượng output tương ứng.

Sau đó ta compile mô hình với optimizer adam và hàm loss là mae (mean absolute error)

```
callback = EarlyStopping(monitor='loss', patience=50, restore_best_weights=True) #EarlyStop nếu 50 epoch không thay đổi loss trả về mô hình với loss thấp nhất

start_time = datetime.now() #Lưu thời gian bắt đầu train
model.fit(X_train, y_train, epochs=1000, shuffle=False, verbose=2, callbacks=[callback]) #Train mô hình
print(f"Thời gian train: {(datetime.now() - start_time).total_seconds()} giây") #In thời gian train
```

Khai báo callback là EarlyStopping có nhiệm vụ giám sát giá trị loss, patience 50 là nếu giá trị loss không thay đổi qua 50 epoch thì kết thúc train, restore\_best\_weight True sẽ trả về mô hình có loss thấp nhất

Thực hiện train mô hình, với epochs là 1000 không shuffle cùng với callback. Sau khi train xong sẽ in ra thời gian train mô hình.

```
test_predict = model.predict(X_test, verbose=0)
test_predict
```

Thực hiện dự đoán mô hình thông qua tập test.

```
fig, axs = plt.subplots(4, figsize = (20,10)) #Tạo plot

dates = dataframe['date'][-(n_steps_in):] #Lấy cột date
titles = ['Giá mở cửa', 'Giá trần', 'Giá sàn', 'Giá đóng cửa'] #Khai báo mảng tiêu đề

for idx, feature in enumerate(features[1:]): #Lặp vẽ 4 tương ứng với các feature được chọn trừ date
    axs[idx].plot(dates, dataframe[feature][-(n_steps_in):]) #Vẽ đường thực tế
    axs[idx].plot(dates, test_predict[:,idx]) #Vẽ đường dự đoán
    axs[idx].legend([f'true {feature}', f'predict {feature}']) #Tạo chú thích
    axs[idx].set_title(titles[idx]) #Tạo tiêu đề
```



```
plt.tight_layout() #Giảm lề
plt.show() #In hình
```

Sau khi dự đoán tập test, ta thực hiện trực quan hóa chúng để quan sát dự đoán của mô hình có chính xác không. Tạo figure với 4 plot, sau đó lấy cột date ứng với chiều dài của tập test. Tạo mảng titles chứa tiêu đề các plot.

Tiếp theo, ta thực hiện lặp enumerate features ngoại trừ date ra và vẽ đường thực tế, đường dự đoán. Mỗi vòng lặp sẽ vẽ plot cho 1 features và thứ tự plot ứng với idx sau đó set tiêu đề cho plot.

```
predicts = list(dataframe[features[1:]][-14:].to_numpy())
#Lấy dữ liệu 14 ngày trước

while(len(predicts) < 14+14): #Thực hiện vào lặp dự đoán 14 ngày tiếp theo
    X_predicts = np.array([predicts[-14:]] #Lấy 14 ngày trước từ mảng predicts
    predict = model.predict(X_predicts, verbose=0) #Thực hiện dự đoán
    predicts.append(predict[0]) #Thêm dự đoán vào mảng predicts để tiếp tục dự đoán

predicts = np.array(predicts) #Chuyển predicts lại thành np array
predicts
```

Tiếp tục, ta sẽ thực hiện dự đoán 14 ngày trong tương lai bằng cách sử dụng 14 ngày dữ liệu có sẵn (14 ngày này ứng với n\_steps\_in).

Ta sẽ tạo mảng X\_predicts chứa dữ liệu 14 ngày trước đó và thực hiện dự đoán sau đó thêm lại vào mảng predict và lặp lại như thế 14 lần.

```
fig, axs = plt.subplots(4,figsize = (20,10)) #Tạo plot

dates = dataframe['date'][-(n_steps_in):] #Lấy cột date
dates = dates_add(dates, 14) #Thêm vào mảng date 14 ngày
titles = ['Giá mở cửa', 'Giá trần', 'Giá sàn', 'Giá đóng cửa'] #Khai báo mảng tiêu đề
```

```

for idx, feature in enumerate(features[1:]): #Lặp vẽ 4 tương ứng với các
    feature được chọn trừ date
    axs[idx].plot(dates[:14], predicts[:14,idx]) #Vẽ đường thực tế
    axs[idx].plot(dates, np.concatenate((test_predict[:,idx], predicts[14:
,idx]))) #Vẽ đường dự đoán
    axs[idx].legend([f'true {feature}', f'predict {feature}']) #Tạo chú th
ích
    axs[idx].set_title(titles[idx]) #Tạo tiêu đề

plt.tight_layout() #Giảm lề
plt.show() #In hình

```

Sau khi dự đoán tập 14 ngày tiếp theo, ta thực hiện trực quan hóa chúng để quan sát dự đoán của mô hình. Tạo figure với 4 plot, sau đó lấy cột date ứng với chiều dài của tập test. Sau đó thêm vào mảng day 14 ngày tiếp theo. Tạo mảng titles chứa tiêu đề các plot.

Tiếp theo, ta thực hiện lặp enumerate features ngoại trừ date ra và vẽ đường thực tế, đường dự đoán. Mỗi vòng lặp sẽ vẽ plot cho 1 features và thứ tự plot ứng với idx sau đó set tiêu đề cho plot.

### Dữ liệu tuần

```

n_steps_in = 7 #Lấy {n_steps_in} tuần trước để dự đoán tuần này

dataset = dataframe_w[features[1:]].to_numpy() #Chuyển dataframe tuần th
ành mảng
X, y = split_sequences(dataset, n_steps_in) #Chia dataset thành X và y t
heo {n_steps_in}

n_input = X.shape[1] * X.shape[2] #Tính chiều dài input
n_output = y.shape[1] * y.shape[2] #Tính chiều dài output

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=len
(X) -
n_steps_in, shuffle = False) #Chia dữ liệu tập train và test trong đó tes
t chứa {n_steps_in} để thực hiện dự đoán

```

Phần này sẽ thực hiện tiền xử lý dữ liệu như tạo X và y bằng hàm split\_sequences.

Tiếp đó tính chiều dài của input và output lần lượt gán vào `n_input`, `n_output` để sử dụng cho quá trình xây dựng mô hình.

Và cuối cùng sử dụng `train_test_split` với tham số `shuffle` là `false` thực hiện chia dữ liệu thành tập test và tập train.

```
model = Sequential() #Khởi tạo model
model.add(LSTM(20, kernel_regularizer=l2(0.01), recurrent_regularizer=l2(0.01), bias_regularizer=l2(0.01))) #Thêm lớp LSTM với regularizer
model.add(Dropout(0.1)) #Thêm lớp Drop Out
model.add(Dense(n_output)) #Thêm lớp Dense với n_output là số lượng tham số đầu ra
model.compile(optimizer='adam', loss='mae')
```

Ta tiến hành xây dựng mô hình sử dụng `Sequential`, với lớp đầu tiên là `LSTM` với số 20 là số unit xử lý của lớp đó, cùng với `kernel_regularizer`, `recurrent_regularizer`, `bias_regularizer` có chung tham số `l2(0.01)` giúp cho mô hình trở nên đơn giản hơn. Sau đó là lớp `Dropout`, với tham số 0.1 là rate loại bỏ trọng số giúp cho mô hình đơn giản hơn. Tiếp đó lớp `Dense` sẽ dùng để trả về kết quả, nhận tham số `n_output` nó sẽ trả về số lượng output tương ứng.

Sau đó ta compile mô hình với optimizer `adam` và hàm loss là `mae` (mean absolute error).

```
callback = EarlyStopping(monitor='loss', patience=50, restore_best_weights=True) #EarlyStop nếu 50 epoch không thay đổi loss trả về mô hình với loss thấp nhất

start_time = datetime.now() #Lưu thời gian bắt đầu train
model.fit(X_train, y_train, epochs=1000, shuffle=False, verbose=2, callbacks=[callback]) #Train mô hình
print(f"Thời gian train: {(datetime.now() - start_time).total_seconds()} giây") #In thời gian train
```

Khai báo callback là `EarlyStopping` có nhiệm vụ giám sát giá trị `loss`, `patience 50` là nếu giá trị `loss` không thay đổi qua 50 epoch thì kết thúc train, `restore_best_weight True` sẽ trả về mô hình có `loss` thấp nhất

Thực hiện train mô hình, với epochs là 1000 không shuffle cùng với callback. Sau khi train xong sẽ in ra thời gian train mô hình.

```
test_predict = model.predict(X_test, verbose=0)
test_predict
```

Thực hiện dự đoán mô hình thông qua tập test.

```
fig, axs = plt.subplots(4, figsize = (20,10)) #Tạo plot

dates = dataframe_w['date'][-(n_steps_in):] #Lấy cột date
titles = ['Giá mở cửa', 'Giá trần', 'Giá sàn', 'Giá đóng cửa'] #Khai báo
mảng tiêu đề

for idx, feature in enumerate(features[1:]): #Lặp vẽ 4 tương ứng với các
feature được chọn trừ date
    axs[idx].plot(dates, dataframe_w[feature] [-
(n_steps_in):]) #Vẽ đường thực tế
    axs[idx].plot(dates, test_predict[:,idx]) #Vẽ đường dự đoán
    axs[idx].legend([f'true {feature}', f'predict {feature}']) #Tạo chú th
ích
    axs[idx].set_title(titles[idx]) #Tạo tiêu đề

plt.tight_layout() #Giảm lề
plt.show() #In hình
```

Sau khi dự đoán tập test, ta thực hiện trực quan hóa chúng để quan sát dự đoán của mô hình có chính xác không. Tạo figure với 4 plot, sau đó lấy cột date ứng với chiều dài của tập test. Tạo mảng titles chứa tiêu đề các plot.

Tiếp theo, ta thực hiện lặp enumerate features ngoại trừ date ra và vẽ đường thực tế, đường dự đoán. Mỗi vòng lặp sẽ vẽ plot cho 1 features và thứ tự plot ứng với idx sau đó set tiêu đề cho plot.

```
predicts = list(dataframe[features[1:]] [-7:].to_numpy())
#Lấy dữ liệu 7 tuần trước

while(len(predicts) < 7+7): #Thực hiện vào lặp dự đoán 7 tuần tiếp theo
```

```

X_predicts = np.array([predicts[-
7:]] #Lấy 7 tuần trước từ mảng predicts
predict = model.predict(X_predicts, verbose=0) #Thực hiện dự đoán
predicts.append(predict[0]) #Thêm dự đoán vào mảng predicts để tiếp tục dự đoán

predicts = np.array(predicts) #Chuyển predicts lại thành np array
predicts[7:] #In dự đoán

```

Tiếp tục, ta sẽ thực hiện dự đoán 7 tuần trong tương lai bằng cách sử dụng 7 tuần dữ liệu có sẵn (7 tuần này ứng với `n_steps_in`).

Ta sẽ tạo mảng `X_predicts` chứa dữ liệu 7 tuần trước đó và thực hiện dự đoán sau đó thêm lại vào mảng `predict` và lặp lại như thế 7 lần.

```

fig, axs = plt.subplots(4,figsize = (20,10)) #Tạo plot

dates = dataframe_w['date'][-(n_steps_in):] #Lấy cột date
dates = dates_add(dates, 7, distance=7) #Thêm vào mảng date 7 tuần
titles = ['Giá mở cửa', 'Giá trần', 'Giá sàn', 'Giá đóng cửa'] #Khai báo mảng tiêu đề

for idx, feature in enumerate(features[1:]): #Lặp vẽ 4 tương ứng với các feature được chọn trừ date
    axs[idx].plot(dates[:7], predicts[:7,idx]) #Vẽ đường thực tế
    axs[idx].plot(dates, np.concatenate((test_predict[:,idx], predicts[7:,idx]))) #Vẽ đường dự đoán
    axs[idx].legend([f'true {feature}', f'predict {feature}']) #Tạo chú thích
    axs[idx].set_title(titles[idx]) #Tạo tiêu đề

plt.tight_layout() #Giảm lề
plt.show() #In hình

```

Sau khi dự đoán tập 7 tuần tiếp theo, ta thực hiện trực quan hóa chúng để quan sát dự đoán của mô hình. Tạo figure với 4 plot, sau đó lấy cột date ứng với chiều dài của tập test. Sau đó thêm vào mảng day 7 tuần tiếp theo. Tạo mảng titles chứa tiêu đề các plot.

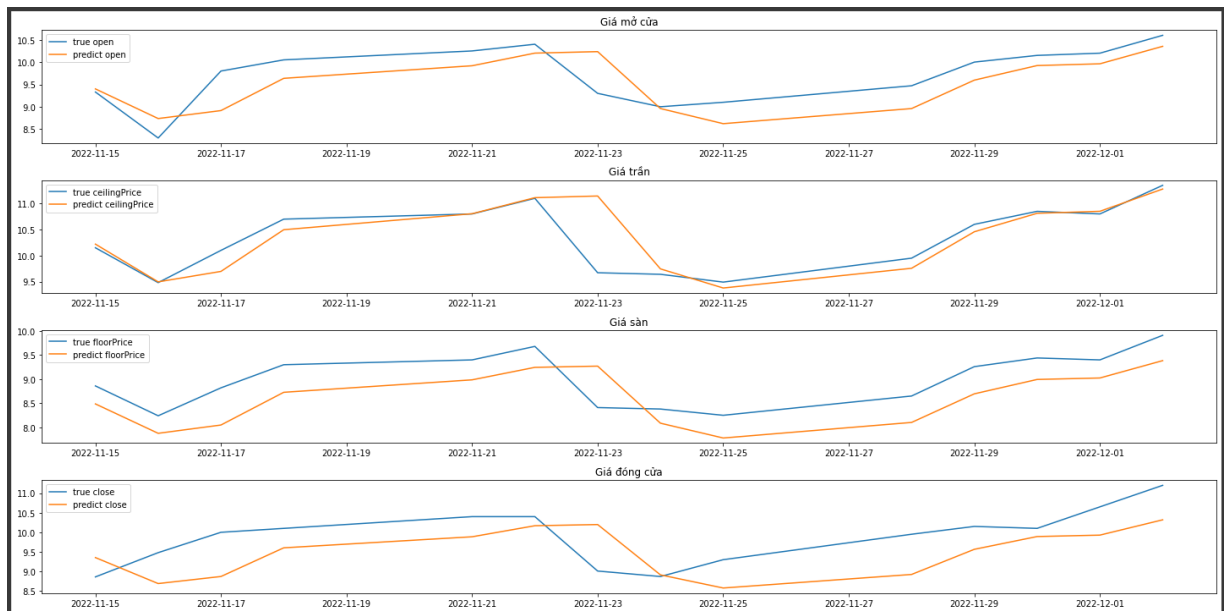
Tiếp theo, ta thực hiện lặp enumerate features ngoại trừ date ra và vẽ đường thực tế, đường dự đoán. Mỗi vòng lặp sẽ vẽ plot cho 1 features và thứ tự plot ứng với idx sau đó set tiêu đề cho plot.

#### 4.3.5 Kết quả thu được và đánh giá

 **Dữ liệu ngày:**

```
array([[ 9.3999405, 10.220026,  8.486164,  9.35412 ],
       [ 8.735869,  9.494812,  7.875838,  8.689816 ],
       [ 8.914093,  9.695768,  8.0481615,  8.870518 ],
       [ 9.636377, 10.496767,  8.727344,  9.603282 ],
       [ 9.918145, 10.803909,  8.985755,  9.885432 ],
       [10.201179, 11.112446,  9.24512,  10.1677475],
       [10.234633, 11.144736,  9.270619, 10.198093 ],
       [ 8.960093,  9.74409,  8.085976,  8.912041 ],
       [ 8.6210985,  9.376788,  7.7784343,  8.576393 ],
       [ 8.959272,  9.755642,  8.101574,  8.922716 ],
       [ 9.596057, 10.457645,  8.695454,  9.565042 ],
       [ 9.922644, 10.812478,  8.9938135,  9.891369 ],
       [ 9.962088, 10.850901,  9.024404,  9.92791 ],
       [10.3533745, 11.278736,  9.38484,  10.318882 ]], dtype=float32)
```

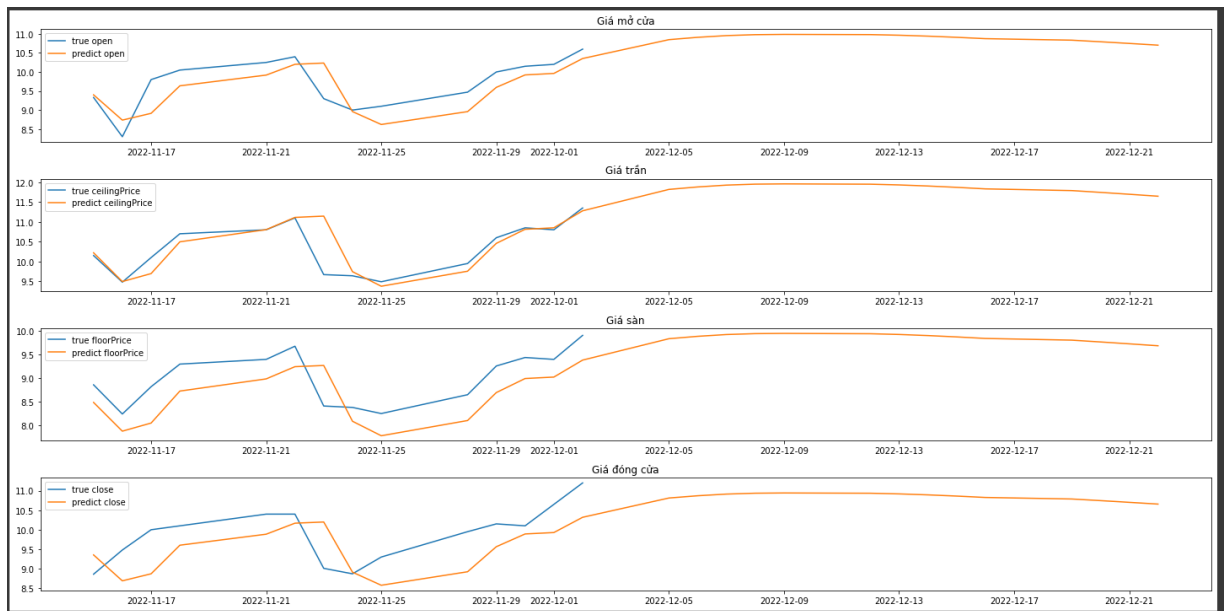
Hình 4.3.5a: Dự đoán tập test



Hình 4.3.5b: Kết quả dự đoán tập test

```
array([[10.84867764, 11.81820488, 9.83906937, 10.81315422],
       [10.90984154, 11.88042641, 9.88983154, 10.87097549],
       [10.95521164, 11.92663097, 9.92745018, 10.91374969],
       [10.97962666, 11.95084476, 9.94691658, 10.93623352],
       [10.98708248, 11.95725441, 9.95165443, 10.94228363],
       [10.98123741, 11.94965458, 9.94478226, 10.93542862],
       [10.96498203, 11.93111038, 9.92883301, 10.91847992],
       [10.94061565, 11.90403938, 9.90581036, 10.89366722],
       [10.9099474 , 11.87034988, 9.87729836, 10.86275101],
       [10.87442589, 11.83153057, 9.84452629, 10.82712364],
       [10.83521557, 11.78881645, 9.80852318, 10.78791904],
       [10.79326725, 11.74324894, 9.77016068, 10.74607658],
       [10.74936485, 11.69565582, 9.73012829, 10.7023592 ],
       [10.70414639, 11.64664841, 9.68892574, 10.65736389]])
```

Hình 4.3.5c: Kết quả dự đoán 14 ngày tiếp theo

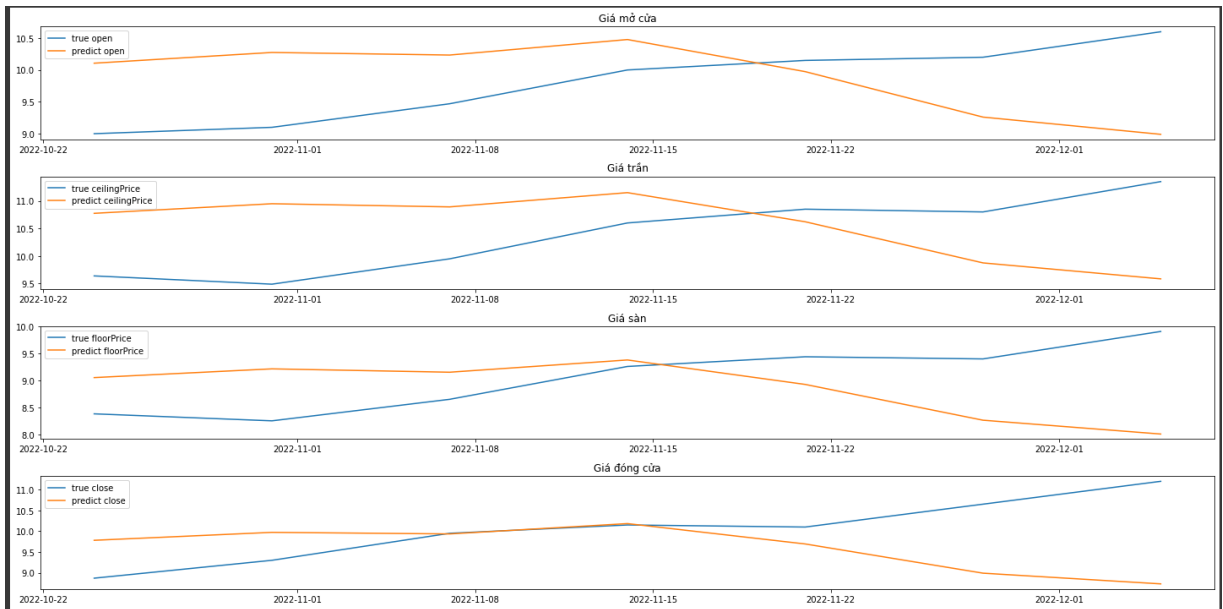


Hình 4.3.5d: Kết quả dự đoán 14 ngày tiếp theo

📊 **Dữ liệu tuần:**

```
array([[10.106488 , 10.774746 , 9.053236 , 9.781181 ],
       [10.274971 , 10.948835 , 9.215374 , 9.972165 ],
       [10.234178 , 10.891438 , 9.153015 , 9.934487 ],
       [10.477985 , 11.149876 , 9.380428 , 10.182775 ],
       [ 9.9739275, 10.6217 , 8.926332 , 9.693042 ],
       [ 9.260516 , 9.876003 , 8.26181 , 8.990412 ],
       [ 8.989966 , 9.586241 , 8.005047 , 8.732797 ]], dtype=float32)
```

Hình 4.3.5e: In dự đoán theo tập test

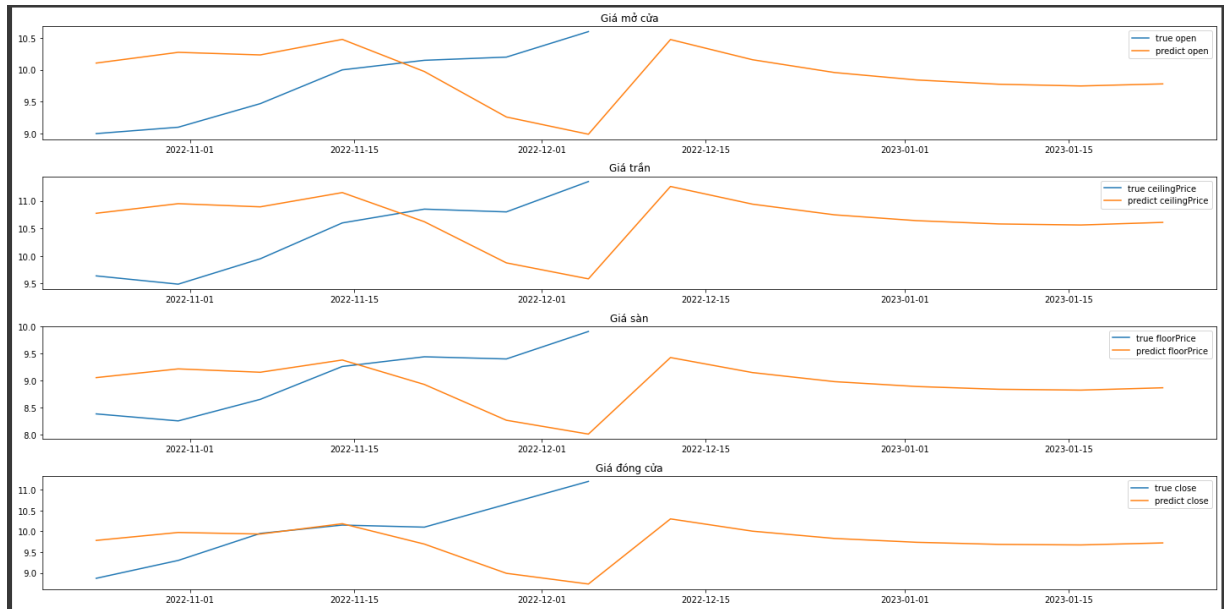


Hình 4.3.5f: Kết quả dự đoán theo tập test

```
array([[10.47564602, 11.26055145, 9.42588997, 10.29841232],
       [10.15883732, 10.94004822, 9.14688683, 10.00341606],
       [ 9.95794487, 10.74662209, 8.97871208, 9.82816505],
       [ 9.84170818, 10.6410141 , 8.88828182, 9.7361517 ],
       [ 9.77433014, 10.58143044, 8.8371563 , 9.68583202],
       [ 9.74728107, 10.56223106, 8.82141304, 9.67227268],
       [ 9.77987766, 10.60995102, 8.86476707, 9.72095966]])
```

Hình 4.3.5g: In dự đoán 7 tuần tiếp theo





*Hình 4.3.5h: Kết quả dự đoán 7 tuần tiếp theo*

## TÀI LIỆU THAM KHẢO

### Tiếng Việt

[1] (2020, April 13). “Học sâu (Deep Learning) là gì? Học sâu và học máy”.

Vietnambiz. <https://vietnambiz.vn/hoc-sau-deep-learning-la-gi-hoc-sau-va-hoc-may-20200410165603552.htm>

[2] Lily, Y. (2021, December 16). “Sự khác biệt giữa AI vs Học máy vs Học sâu vs Mạng nơ-ron nhân tạo - iRenderFarm với AI/Deep Learning”, irender.vn.

<https://irender.vn/su-khac-biet-giua-ai-vs-hoc-may-vs-hoc-sau-vs-mang-no-ron-nhan-tao/>

[3] Hai D. M. (2017, October 20). [RNN] LSTM là gì? Hai’s Blog.

<https://dominhhai.github.io/vi/2017/10/what-is-lstm/>

### Tiếng Anh

[4] Brownlee, J. (2018, November 9). “How to Develop Multilayer Perceptron Models for Time Series Forecasting - MachineLearningMastery.com.”, Machine

Learning Mastery. <https://machinelearningmastery.com/how-to-develop-multilayer-perceptron-models-for-time-series-forecasting/>