

TRƯỜNG ĐẠI HỌC THỦY LỢI
KHOA CÔNG NGHỆ THÔNG TIN



BÀI TẬP LỚN
HỌC PHẦN: PHÂN TÍCH CHUỖI THỜI GIAN

ĐỀ TÀI: Xây dựng mô hình dự báo giá cổ phiếu Vietcombank

Giáo viên hướng dẫn: Trần Anh Đạt

Nhóm 01 - 63TTNT:

Nguyễn Trung Tuyên	2151260823
Nguyễn Đức Tuấn	2151264692
Nguyễn Anh Tuấn	2151264691
Bùi Trung Quốc	2151260827

Hà Nội, tháng 06 năm 2024

MỤC LỤC

Mở đầu	1
CHƯƠNG 1. LÝ THUYẾT	2
1.1. Mô hình Transformer kết hợp với Kalman Filter	2
1.1.1. Mô hình Transformer	2
1.1.2. Kalman Filter	3
1.2. Mô hình BiLSTM kết hợp với Attention	4
1.2.1. Mạng Long Short Term Memory (LSTM)	4
1.2.2. Mạng BiLSTM	6
1.3. Mô hình Temporal Convolutional Network	7
1.4. Mô hình Garch	9
CHƯƠNG 2. XÂY DỰNG MÔ HÌNH	12
2.1. Tập dữ liệu	12
2.2. Mô hình Transformer kết hợp với Kalman	12
2.3. Mô hình BiLSTM kết hợp với Attention	16
2.4. Mô hình Temporal Convolutional Networks	19
2.5. Mô hình Garch	22
KẾT LUẬN	33
TÀI LIỆU THAM KHẢO	35

Mở đầu

Trong bối cảnh tài chính cạnh tranh khốc liệt hiện nay, việc dự đoán giá cổ phiếu của các ngân hàng lớn như Vietcombank không chỉ là một nhu cầu thiết yếu mà còn là chìa khóa quan trọng trong việc quản lý rủi ro và xây dựng chiến lược đầu tư cho nhà đầu tư và tổ chức tài chính. Đối mặt với sự biến động liên tục của thị trường và yêu cầu ngày càng cao từ phía khách hàng, khả năng dự báo chính xác giá cổ phiếu trở nên cực kỳ quan trọng, đảm bảo thành công và ổn định cho các hoạt động giao dịch và đầu tư.

Các nhà nghiên cứu và chuyên gia phân tích thị trường đã không ngừng áp dụng các mô hình dự báo tiên tiến để cải thiện độ chính xác của dự báo. Trong đề tài này, nhóm sẽ sử dụng bốn mô hình chính: Transformer, BiLSTM, Temporal Convolutional Network (TCN), và GARCH. Transformer là một mô hình mạng nơ-ron tiên tiến, nổi bật với khả năng xử lý dữ liệu chuỗi thời gian dài một cách hiệu quả. BiLSTM, viết tắt của Bidirectional Long Short-Term Memory, là một biến thể của LSTM cho phép mô hình xử lý dữ liệu theo cả hai chiều thời gian. Temporal Convolutional Network (TCN) là một loại mạng nơ-ron tích chập, thiết kế đặc biệt để xử lý dữ liệu chuỗi thời gian với độ trễ thấp. Cuối cùng, GARCH, viết tắt của Generalized Autoregressive Conditional Heteroskedasticity, là một mô hình đặc biệt hữu ích trong việc dự báo biến động và xu hướng của dữ liệu tài chính với sự biến thiên không đều.

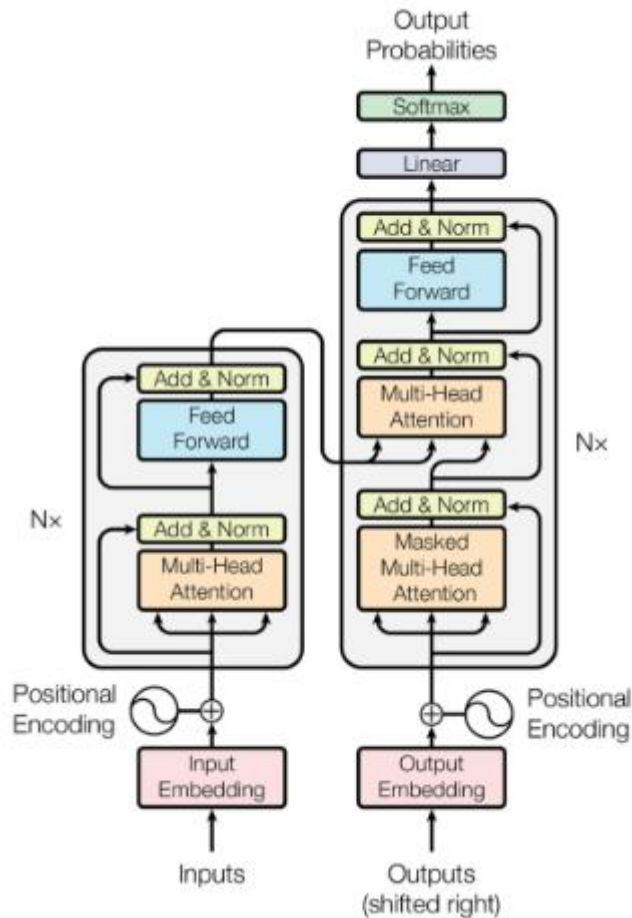
Nhóm sẽ so sánh độ chính xác giữa giá trị dự báo của các mô hình này và từ đó đưa ra kết luận về mô hình nào phù hợp nhất với bài toán dự báo giá cổ phiếu Vietcombank.

CHƯƠNG 1. LÝ THUYẾT

1.1. Mô hình Transformer kết hợp với Kalman Filter

1.1.1. Mô hình Transformer

Mô hình Transformer là mô hình học sâu được ra mắt vào năm 2017, được sử dụng chủ yếu trong lĩnh vực xử lý ngôn ngữ tự nhiên nhưng gần đây mô hình này cũng đã được áp dụng vào các bài toán liên quan đến chuỗi thời gian như là dự báo giá cổ phiếu, dự báo thời tiết,...



Hình 1.1. Cấu trúc mô hình Transformer

Transformer được cấu tạo bởi hai thành phần chính: Encoder và Decoder. Với các bài toán như NLP, CV thì mô hình này sẽ được sử dụng tất cả các thành phần mà nó có. Nhưng với bài toán dự báo giá cổ phiếu thì mô hình chỉ được sử dụng thành phần Encoder để chuyển đổi chuỗi dữ liệu thời gian đầu vào thành dạng thích hợp để dự đoán.

Encoder là cấu trúc xếp chồng lên nhau của 6 layers xác định¹. Mỗi layer bao gồm 2 layer con (sub-layer) ở trong nó. Sub-layer đầu tiên chính là MultiHead Self-Attention, layer thứ hai chính là các fullconnected feed-forward layer. Với mỗi sub-layer sẽ được sử dụng kết nối residual ngay sau layer normalization.

Trong phần Embedding và Positional Encoding, các giá trị đầu vào đã được chuyển đổi thành các vector có chiều cao hơn thông qua embedding. Điều này giúp cho dữ liệu chuỗi thời gian được chuyển đổi thành dạng có thể xử lý được. Do mô hình transformer không có cơ chế tuần tự tự nhiên như mô hình RNN, nên cần có cách biểu thị thông tin vị trí của các phần tử trong chuỗi, chính vì vậy Positional Encoding được thêm vào embeddings để cung cấp thông tin về thứ tự. Công thức được tính như sau:

$$p_t^i = f(t)^i = \begin{cases} \sin(w_k * t) & \text{if } i = 2k \\ \cos(w_k * t) & \text{if } i = 2k + 1 \end{cases}$$

Trong đó

$$w_k = \frac{1}{10000^{2k/d}}$$

Một thành phần khác cũng quan trọng trong Transformer đó là Self-attention, giúp cho mô hình có thể học được mối quan hệ giữa các phần tử khác nhau trong chuỗi dữ liệu. Được tính theo công thức sau:

$$Attention(Q, K, V) = softmax(\frac{QK^t}{\sqrt{dk}})V$$

Trong đó Q là Query, K là Key, V là Value là các ma trận được tạo từ vector đầu vào thông qua các trọng số học được và dk là chiều của vector key. Ở trong encoder không chỉ có một lớp Self-Attention mà nó sử dụng nhiều lớp Self-Attention song song hay còn gọi là MultiHead Attention nhằm giúp cho mô hình có thể học được từ nhiều góc độ khác nhau. Và được tính theo công thức sau:

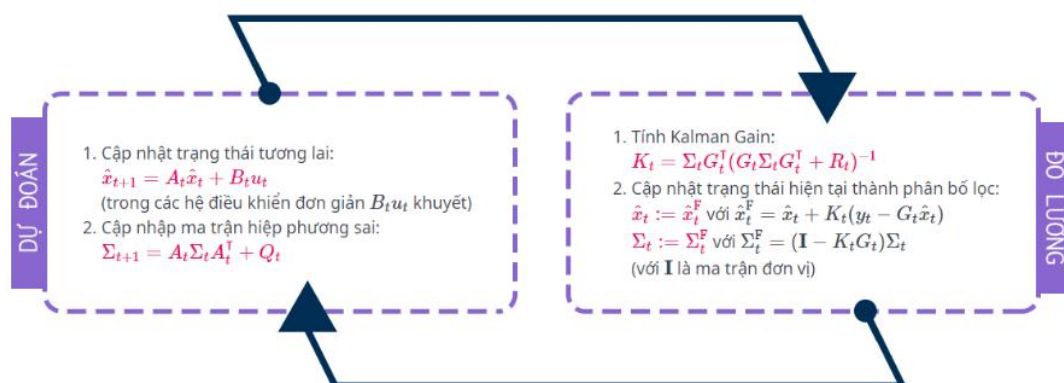
$$Multihead(Q, K, V) = concat(head1, head2, \dots, headh)W_o$$

Với head là giá trị được tính tại tầng Self-Attention và W_o là trọng số học được cho việc kết hợp các đầu chú ý. Sau mỗi lớp attention, dữ liệu sẽ được qua một mạng nơ ron truyền thẳng gồm hai lớp Dense với hàm kích hoạt Relu ở giữa để tăng khả năng biểu diễn của mô hình bằng cách áp dụng các biến đổi phi tuyến tính. Để đảm bảo tính ổn định và hiệu quả trong quá trình huấn luyện, Layer Normalization được áp dụng sau mỗi lớp sub-layer và các kết nối dư để giữ lại thông tin từ các lớp trước đó. Với cấu trúc này, mô hình có khả năng học và biểu diễn các mối quan hệ phức tạp trong dữ liệu chuỗi thời gian, giúp dự đoán chính xác và hiệu quả.

1.1.2. Kalman Filter

Kalman Filterⁱⁱ là một thuật toán tối ưu hóa và lọc tuyến tính được sử dụng rộng rãi để dự đoán và cập nhật trạng thái của một hệ thống động qua thời gian,

ngay cả khi trạng thái đó bị nhiễu hoặc không quan sát được. Trong bài toán dự báo giá cổ phiếu này, Kalman được sử dụng để làm mịn dữ liệu đầu vào và giảm nhiễu từ đó cải thiện độ chính xác của các dự đoán. Thuật toán này hoạt động dựa trên hai giai đoạn chính: Dự đoán và đo lường được biểu diễn như hình dưới đây:



Hình 1.2. Cấu trúc hoạt động thuật toán Kalman

Các bước áp dụng Kalman Filter trong mô hình dự báo giá cổ phiếu gồm:

Bước 1: Khởi tạo Kalman Filter

Bước 2: Áp dụng giai đoạn dự đoán

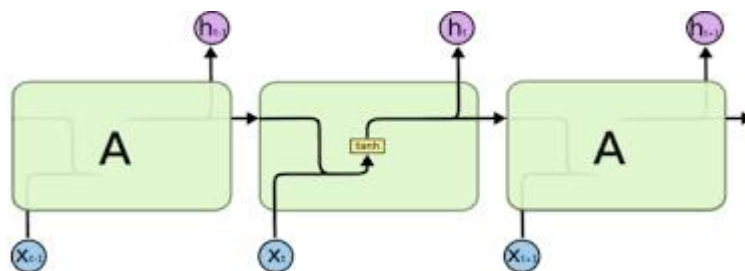
Bước 3: Áp dụng giai đoạn đo lường

Bước 4: Lặp lại cho tất cả các điểm dữ liệu

1.2. Mô hình BiLSTM kết hợp với Attention

1.2.1. Mạng Long Short Term Memory (LSTM)

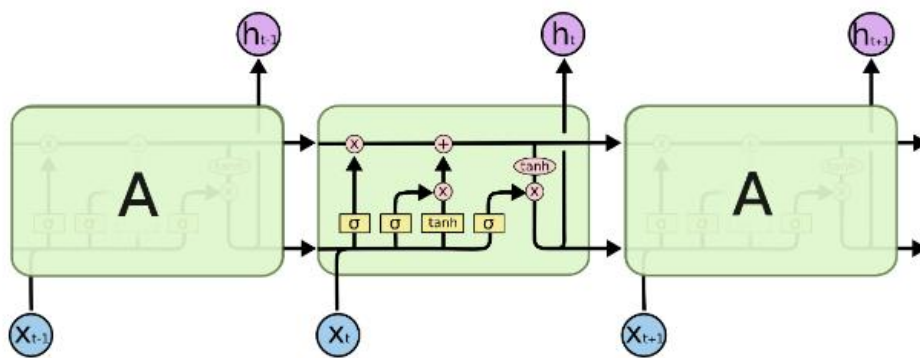
LSTM (Long Short-Term Memory) là một loại mạng nơ-ron hồi quy (Recurrent Neural Network - RNN) được thiết kế để xử lý và dự đoán các dữ liệu theo chuỗi (sequence data). LSTM giải quyết vấn đề gradient biến mất (vanishing gradient problem) thường gặp ở các mạng RNN thông thường, giúp nó có khả năng ghi nhớ thông tin trong khoảng thời gian dài.



Hình 1.3. Cấu trúc mạng RNN

LSTM bao gồm các thành phần chính sau:

- **Cell State (Trạng thái tế bào):** Lưu trữ thông tin của mạng theo thời gian.
- **Forget Gate (Cổng quên):** Quyết định thông tin nào nên được giữ lại và thông tin nào nên được loại bỏ khỏi cell state.
- **Input Gate (Cổng vào):** Quyết định thông tin nào từ input hiện tại sẽ được thêm vào cell state.
- **Output Gate (Cổng ra):** Quyết định phần nào của cell state sẽ được sử dụng để tạo ra output của LSTM.



Hình 1.4. Cấu trúc mạng LSTM

Nhiệm vụ của các thành phần trong mạng LSTM:

Forget Gate: Quyết định những thông tin nào từ trạng thái tế bào trước đó sẽ bị lãng quên hay là không.

Input Gate: Xác định thông tin mới nào sẽ được lưu trữ trong trạng thái tế bào.

Hidden State: Cập nhật trạng thái tế bào dựa trên output của cổng forget và input gate.

Output Gate: Quyết định thông tin nào từ trạng thái tế bào sẽ được output ra ngoài.

Ô nhớ (Cell State): trong LSTM lưu trữ thông tin qua nhiều bước thời gian, giúp duy trì và quản lý thông tin dài hạn

Công thức tính toán các thành phần trong mạng LSTM:

$$I_t = \sigma(X_t W_{xi} + H_{t-1} W_{hi} + b_i)$$

$$F_t = \sigma(X_t W_{xf} + H_{t-1} W_{hf} + b_f)$$

$$O_t = \sigma(X_t W_{xo} + H_{t-1} W_{ho} + b_o)$$

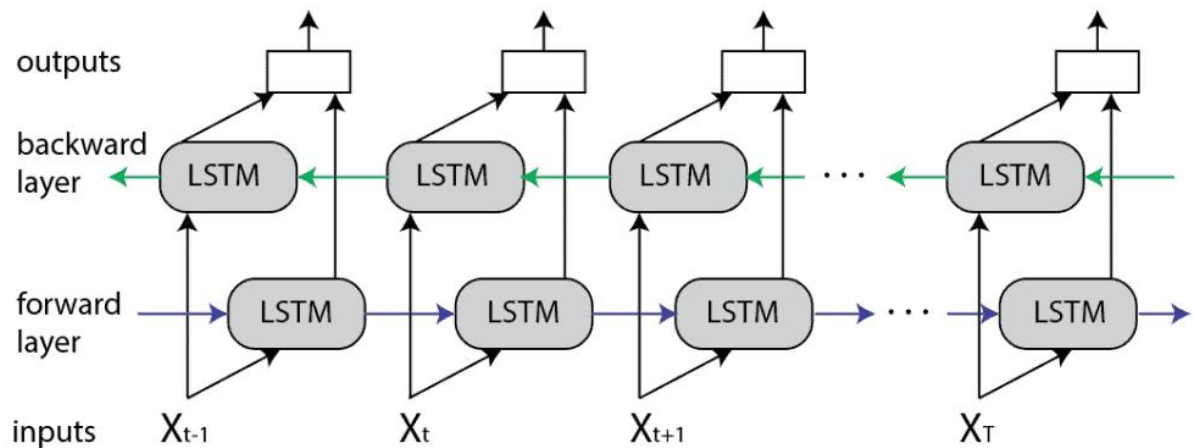
$$\tilde{\mathbf{C}}_t = \tanh(\mathbf{X}_t \mathbf{W}_{xc} + \mathbf{H}_{t-1} \mathbf{W}_{hc} + \mathbf{b}_o)$$

$$\mathbf{C}_t = \mathbf{F}_t * \mathbf{C}_{t-1} + \mathbf{I}_t * \tilde{\mathbf{C}}_t$$

$$\mathbf{H}_t = \mathbf{O}_t * \tanh(\mathbf{C}_t)$$

1.2.2. Mạng BiLSTM

BiLSTM (Bidirectional Long Short-Term Memory) là một mở rộng của mạng LSTM truyền thống, cho phép mạng học từ cả hai hướng của chuỗi dữ liệu. Trong khi LSTM tiêu chuẩn chỉ xử lý dữ liệu theo một chiều (từ quá khứ đến hiện tại), BiLSTM xử lý dữ liệu theo cả hai chiều (từ quá khứ đến hiện tại và từ hiện tại đến quá khứ), giúp cải thiện hiệu suất của mô hình trong nhiều ứng dụng, đặc biệt là trong xử lý ngôn ngữ tự nhiên (NLP).



Hình 1.5. Cách hoạt động của mạng BiLSTM

Cấu trúc của BiLSTM

BiLSTM bao gồm hai LSTM riêng biệt:

- **LSTM hướng tiến (Forward LSTM):** Xử lý chuỗi dữ liệu theo chiều từ trái sang phải.
- **LSTM hướng lùi (Backward LSTM):** Xử lý chuỗi dữ liệu theo chiều từ phải sang trái.

Đầu ra của hai LSTM này được kết hợp lại để tạo ra đầu ra cuối cùng của BiLSTM.

Hoạt động của BiLSTM

Hoạt động của BiLSTM được thực hiện qua các bước sau:

- **Xử lý hướng tiến:** Chuỗi dữ liệu được đưa vào LSTM hướng tiến và tạo ra một chuỗi các trạng thái ẩn (hidden states).
- **Xử lý hướng lùi:** Chuỗi dữ liệu được đưa vào LSTM hướng lùi và tạo ra một chuỗi các trạng thái ẩn (hidden states).
- **Kết hợp đầu ra:** Đầu ra của hai LSTM được kết hợp lại để tạo ra đầu ra cuối cùng của BiLSTM.

Ưu điểm của BiLSTM so với LSTM trong bài toán chuỗi thời gian

1. **Khai thác ngữ cảnh từ cả hai phía:**
 - **LSTM:** Chỉ xử lý dữ liệu theo một hướng (từ quá khứ đến hiện tại), bỏ qua thông tin từ tương lai.
 - **BiLSTM:** Xử lý dữ liệu theo cả hai hướng (quá khứ đến hiện tại và hiện tại đến quá khứ), giúp tận dụng toàn bộ ngữ cảnh của chuỗi thời gian, làm tăng độ chính xác trong dự báo.
2. **Nhận diện các mẫu phức tạp hơn:**
 - **LSTM:** Chỉ dựa vào thông tin từ quá khứ, có thể bỏ sót các mối quan hệ quan trọng xuất hiện sau đó.
 - **BiLSTM:** Nhận diện và học các mẫu phức tạp và các mối quan hệ dài hạn trong chuỗi thời gian một cách hiệu quả hơn nhờ xem xét thông tin từ cả hai hướng.
3. **Giảm thiểu thông tin bị bỏ sót:**
 - **LSTM:** Có nguy cơ bỏ sót thông tin quan trọng từ tương lai gần vì chỉ nhìn vào quá khứ.
 - **BiLSTM:** Đảm bảo rằng các thông tin quan trọng ở cả hai đầu của chuỗi được xem xét, giảm thiểu khả năng bỏ sót thông tin quan trọng.
4. **Cải thiện dự báo trong khoảng thời gian dài:**
 - **LSTM:** Có thể gặp khó khăn khi dự báo chuỗi thời gian dài hạn do chỉ học từ dữ liệu quá khứ.
 - **BiLSTM:** Học từ cả quá khứ và tương lai, thường dự báo chính xác hơn trong các bài toán chuỗi thời gian dài hạn.

Nhờ các ưu điểm này, BiLSTM thường vượt trội hơn LSTM trong các ứng dụng đòi hỏi phân tích và dự báo chuỗi thời gian chính xác, như dự báo tài chính, phân tích xu hướng thị trường, và dự báo thời tiết.

1.3. Mô hình Temporal Convolutional Network

Input: là một chuỗi thời gian có dạng X

Output: dự đoán ra Y với kích thước tương tự X , nhưng với các giá trị dự đoán cho chuỗi thời gian.

Vì vậy một mạng mô hình hóa chuỗi TCN là một hàm ánh xạ một vector có T phần tử sang một vector khác có T' phần tử

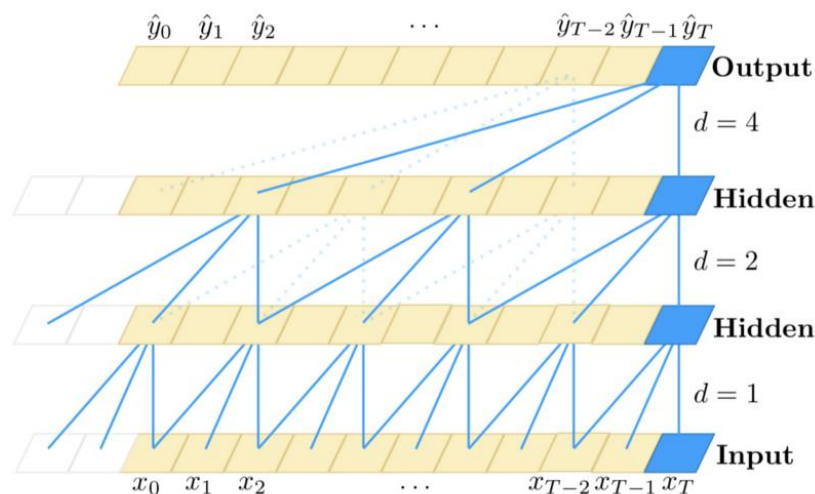
Causal Convolutions: là một phương pháp sử dụng trong mạng TCN được thiết kế để đảm bảo rằng mô hình chỉ có thể "nhìn thấy" thông tin từ quá khứ hoặc hiện tại mà không thể "nhìn thấy" các thông tin từ tương lai. Các giá trị đầu ra của mỗi thời điểm chỉ có thể phụ thuộc vào các giá trị đầu vào từ các thời điểm trước đó hoặc cùng thời điểm. Điều này đảm bảo rằng khi mô hình được sử dụng để dự đoán hoặc sinh ra dữ liệu mới, không có thông tin từ tương lai được sử dụng.

Công thức của causal convolutions được biểu diễn như sau:

$$y(t) = \sum_{k=0}^{K-1} w(k).x(t-k)$$

Trong đó: K là kích thước của cửa sổ convolution, các giá trị x_t với $k=0,1, \dots, K-1$ (tức là các giá trị từ thời điểm $t-K+1$ đến t) được sử dụng trong phép tính này.

Dilated Convolutions: là một phương pháp đặc biệt trong mạng nơ-ron sử dụng để mở rộng lĩnh vực nhìn của các đơn vị tích chập mà không làm tăng đáng kể số lượng tham số so với tích chập thông thường.

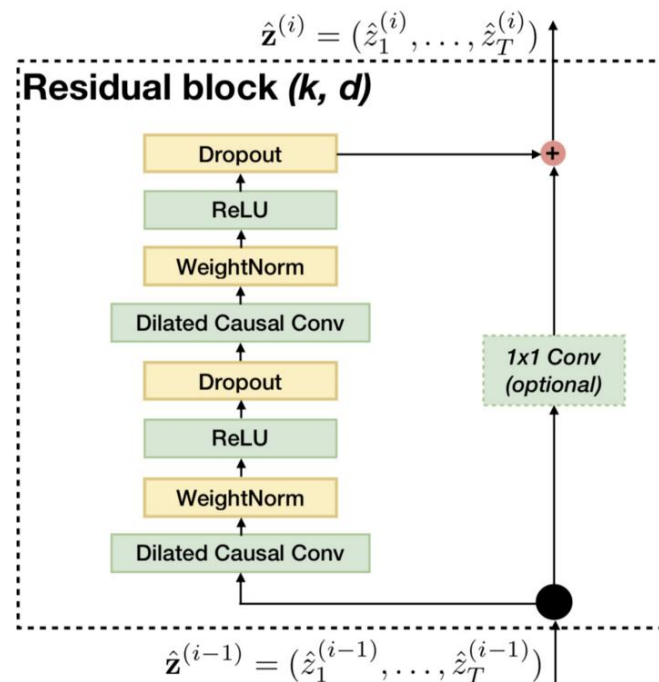


Có thể theo dõi các đường kết nối màu xanh từ các lớp trên xuống các lớp dưới để xem liệu chúng có đạt tới tất cả các đầu vào ở dưới cùng, có nghĩa là dự đoán tại thời điểm sử dụng tất cả các đầu vào trong lịch sử được bao phủ bởi các convolution mở rộng.

Residual connections: là một cơ chế nhằm giải quyết vấn đề của việc huấn luyện các mô hình sâu khi xảy ra hiện tượng biến mất gradient và khó khăn trong việc học các đặc trưng phức tạp

Residual block cho phép mỗi lớp học các sửa đổi cho phép nhận dạng và hoạt động tốt với các mạng rất sâu

Dưới đây là Residual block của TCN cơ bản:



Residual block bao gồm hai lớp convolution gây ra mở rộng, chuẩn hóa trọng số, kích hoạt ReLU và dropout. Có thêm một convolution 1×1 tùy chọn nếu số kênh đầu vào khác với số kênh đầu ra từ convolution gây ra mở rộng (số bộ lọc của convolution gây ra mở rộng thứ hai).

Như vậy: mô hình TCN=1D FCN + Dilated Causal Convolutions

1.4. Mô hình Garch

Mô hình GARCH (Generalized Autoregressive Conditional Heteroskedasticity) được phát triển bởi Tim Bollerslev vào năm 1986, là một mở rộng của mô hình ARCH (Autoregressive Conditional Heteroskedasticity) của Robert Engle được giới thiệu vào năm 1982. Mô hình GARCH được sử dụng rộng rãi trong lĩnh vực tài chính để dự đoán biến động của các chuỗi thời gian, đặc biệt là chuỗi giá tài sản tài chính.

Khái Niệm Cơ Bản của Mô Hình GARCH

- **Biến Động Điều Kiện (Conditional Volatility):** Biến động điều kiện là biến động tại thời điểm hiện tại được xác định dựa trên thông tin quá khứ. Đặc trưng của các chuỗi thời gian tài chính là có những khoảng thời gian biến động cao xen kẽ với những khoảng thời gian biến động thấp, được gọi là tính tự tương quan bậc cao của biến động. Mô hình GARCH bắt đầu từ giả thiết rằng biến động có thể thay đổi theo thời gian và phụ thuộc vào giá trị quá khứ của chính nó và của các phần dư.

- Mô Hình ARCH: Mô hình ARCH được sử dụng để mô tả sự phụ thuộc của phương sai điều kiện vào các giá trị quá khứ của các phần dư. Công thức cơ bản của mô hình ARCH(q) là:

$$\sigma_t^2 = \alpha_0 + \alpha_1 \epsilon_{t-1}^2 + \alpha_2 \epsilon_{t-2}^2 + \dots + \alpha_q \epsilon_{t-q}^2$$

Trong đó σ_t^2 là phương sai điều kiện tại thời điểm t, ϵ_t là phần dư tại thời điểm t, $\alpha_0, \alpha_1, \dots, \alpha_q$ là các tham số của mô hình

- Mô Hình GARCH: Mô hình GARCH là mở rộng của mô hình ARCH bằng cách thêm vào phương sai điều kiện các giá trị quá khứ của chính nó. Công thức của mô hình GARCH(p, q) là:

$$\sigma_t^2 = \alpha_0 + \sum_{i=1}^q \alpha_i \epsilon_{t-i}^2 + \sum_{j=1}^p \beta_j \sigma_{t-j}^2$$

Trong đó β_j là các tham số mới được thêm vào mô hình. Mô hình GARCH(1,1) là dạng đơn giản nhất và phổ biến nhất của mô hình GARCH, với công thức:

$$\sigma_t^2 = \alpha_0 + \alpha_1 \epsilon_{t-1}^2 + \beta_1 \sigma_{t-1}^2$$

Ước Lượng và Kiểm Định Mô Hình GARCH

- Ước Lượng Mô Hình: Các tham số của mô hình GARCH thường được ước lượng bằng phương pháp Maximum Likelihood Estimation (MLE). Quy trình MLE tìm các giá trị của các tham số mà làm tối đa hóa hàm hợp lý của các quan sát.

- Kiểm Định Mô Hình: Sau khi ước lượng các tham số, cần phải kiểm định mô hình để đảm bảo tính hợp lý và hiệu lực của nó. Các kiểm định thường dùng bao gồm kiểm định LM (Lagrange Multiplier) để kiểm tra sự hiện diện của ARCH effect và kiểm định Ljung-Box để kiểm tra tính tự tương quan của phần dư.

Ứng Dụng của Mô Hình GARCH

- Dự Đoán Biến Động: Một trong những ứng dụng chính của mô hình GARCH là dự đoán biến động tương lai của các chuỗi thời gian tài chính. Ví dụ, biến động dự đoán có thể được sử dụng để xác định mức độ rủi ro của các tài sản tài chính, giúp nhà đầu tư ra quyết định hợp lý.

- Định Giá Tùy Chọn: Mô hình GARCH cũng được sử dụng trong định giá tùy chọn, đặc biệt là trong việc ước lượng phương sai tương lai của giá tài sản cơ sở, một yếu tố quan trọng trong việc định giá tùy chọn.

- Quản Lý Rủi Ro: Các tổ chức tài chính sử dụng mô hình GARCH để quản lý rủi ro, xác định mức độ rủi ro của danh mục đầu tư và thiết lập các biện pháp phòng ngừa thích hợp.

Các Ưu Điểm và Hạn Chế của Mô Hình GARCH

- Ưu Điểm:

- Mô hình GARCH linh hoạt và có khả năng mô tả tốt các đặc tính của biến động trong các chuỗi thời gian tài chính.

- Mô hình dễ dàng được mở rộng và tùy chỉnh để phù hợp với các đặc điểm cụ thể của dữ liệu.

- GARCH có thể được sử dụng trong nhiều ứng dụng khác nhau, từ dự đoán biến động đến định giá tùy chọn và quản lý rủi ro.

- Hạn Chế:

- Việc ước lượng các tham số của mô hình có thể phức tạp và yêu cầu nhiều dữ liệu.

- Mô hình GARCH có thể không phù hợp với các chuỗi thời gian có tính bất ổn định hoặc có cấu trúc phức tạp.

- Mô hình GARCH giả định rằng các phần dư là bình thường, điều này có thể không đúng trong nhiều trường hợp thực tế.

CHƯƠNG 2. XÂY DỰNG MÔ HÌNH

2.1. Tập dữ liệu

Dữ liệu được nhóm sử dụng trong đề tài này nói về dữ liệu giá cổ phiếu Vietcombank trong khoảng thời gian 01/2010 - 05/2024 có 3592 mẫu dữ liệu.

Dữ liệu được tải về tại trang web: Market Watch.

Trong mỗi mẫu dữ liệu chứa các thông tin về:

- DATE: Ngày giao dịch
- OPEN: Giá mở cửa
- HIGH: Giá cao nhất
- LOW: Giá thấp nhất
- CLOSE: Giá đóng cửa
- VOLUME: Tổng khối lượng giao dịch ngày

	Open	High	Low	Close	Volume
Date					
2010-01-04	13842	14519	13842	14519	2032674
2010-01-05	15020	15167	14519	14667	3175256
2010-01-06	14166	14667	14136	14136	1629290
2010-01-07	14136	14166	13842	13842	704530
2010-01-08	14019	14078	13812	13842	1193887
...
2024-05-27	90400	90600	89700	90200	1258620
2024-05-28	89800	90700	89800	90700	1814349
2024-05-29	90800	91000	89800	89800	1186438
2024-05-30	89500	89700	88500	88600	2400732
2024-05-31	88700	89000	87200	87200	3921773

3592 rows × 5 columns

Hình 2.1. Dữ liệu về giá cổ phiếu Vietcombank

2.2. Mô hình Transformer kết hợp với Kalman

Như đã trình bày ở phần trên, trong mô hình Transformer này nhóm đã sử dụng kết hợp với Kalman Filter nhằm làm mịn dữ liệu, giảm nhiễu do những lần tăng giảm bất thường của giá cổ phiếu.



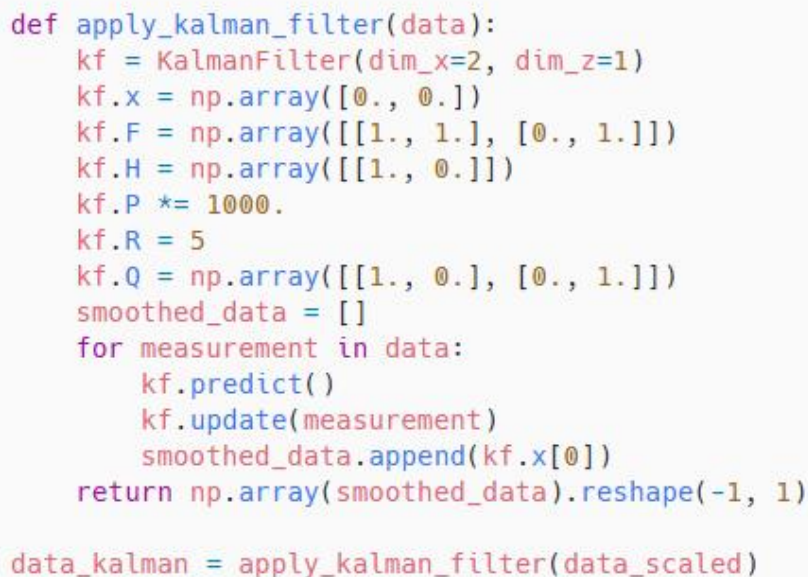
```

# Load and prepare the dataset
file_path =
'/mnt/d/code/TimeSeries/Nhom1_Cuoiki_TimeSeries/Data/vcb_2010-
2024.csv'
df = pd.read_csv(file_path)
df['Date'] = pd.to_datetime(df['Date'], dayfirst=True)
df = df[::-1].reset_index(drop=True)
df.set_index('Date', inplace=True)
data = df[['Close']].values
scaler = MinMaxScaler(feature_range=(0, 1))
data_scaled = scaler.fit_transform(data)

```

Hình 2.2. Đọc dữ liệu và thực hiện chuẩn hóa dữ liệu

Trên hình 2.2 là quá trình đọc dữ liệu và thực hiện chuyển dữ liệu về miền 0,1 giúp chuẩn hóa dữ liệu, loại bỏ các giá trị ngoại biên, nhằm giúp mô hình có thể tập trung vào các xu hướng chung.



```

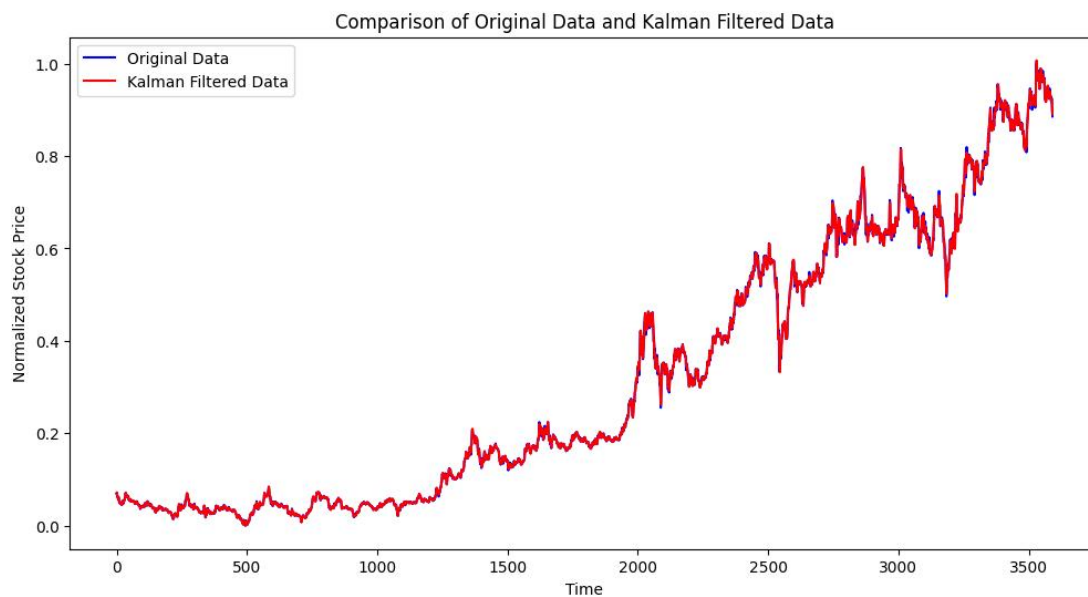
def apply_kalman_filter(data):
    kf = KalmanFilter(dim_x=2, dim_z=1)
    kf.x = np.array([0., 0.])
    kf.F = np.array([[1., 1.], [0., 1.]])
    kf.H = np.array([[1., 0.]])
    kf.P *= 1000.
    kf.R = 5
    kf.Q = np.array([[1., 0.], [0., 1.]])
    smoothed_data = []
    for measurement in data:
        kf.predict()
        kf.update(measurement)
        smoothed_data.append(kf.x[0])
    return np.array(smoothed_data).reshape(-1, 1)

data_kalman = apply_kalman_filter(data_scaled)

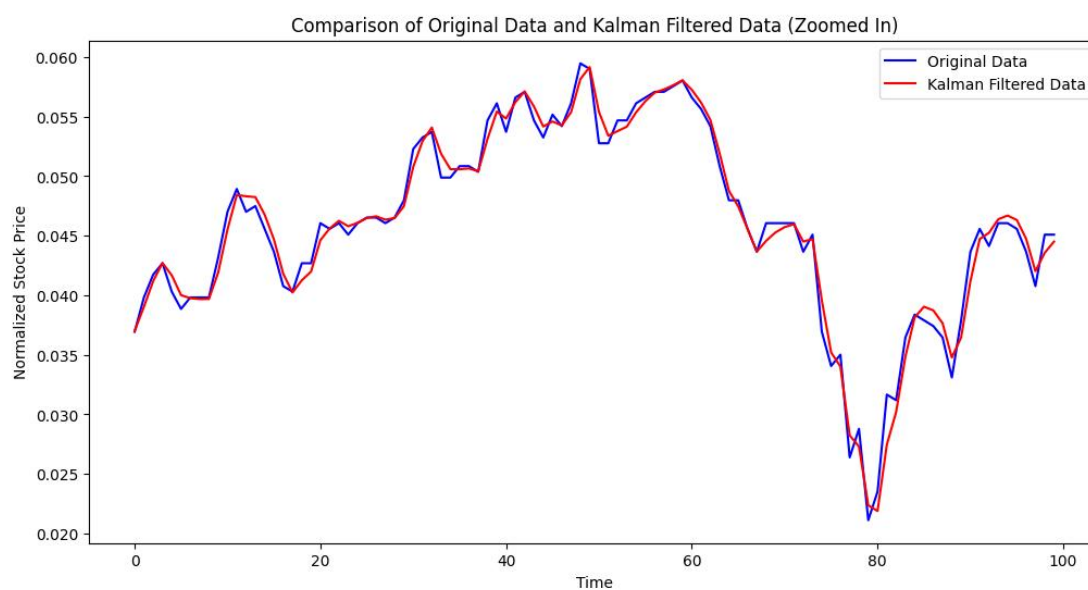
```

Hình 2.3. Đưa dữ liệu qua Kalman Filter

Sau khi đã chuẩn hóa dữ liệu về miền 0,1 tiếp tục đưa dữ liệu qua Kalman Filter để làm mịn và giảm nhiễu. Quan sát hình 2.4 và 2.5 dưới đây là dữ liệu trước và sau khi đưa qua Kalman Filter:



Hình 2.4. Dữ liệu trước và sau khi qua Kalman Filter



Hình 2.5. Dữ liệu trước và sau khi qua Kalman Filter (Zoom in)

Có thể thấy rằng sau khi đưa dữ liệu qua Kalman Filter thì dữ liệu đã mịn hơn, những điểm dữ liệu tăng giảm bất thường gây nhiễu cũng đã được làm mịn. Từ đó giúp bảo tồn các xu hướng chính của dữ liệu.

Với mô hình này, dữ liệu được chia thành 2 tập nhỏ hơn: train chiếm 80%, test chiếm 20% và nhóm đã chia thành 2 kịch bản nhỏ hơn để thực hiện so sánh với nhau trước khi so sánh với các mô hình khác:

Kịch bản 1: Sử dụng dữ liệu của 30 ngày trước để dự đoán ngày tiếp theo.

Kịch bản 2: Sử dụng dữ liệu của 60 ngày trước để dự đoán ngày tiếp theo.



```
inputs = Input(shape=(X_train.shape[1], X_train.shape[2]))
x = transformer_encoder(inputs, head_size=256, num_heads=4,
ff_dim=4, dropout=0.1)
x = GlobalAveragePooling1D(data_format='channels_first')(x)
x = Dropout(0.1)(x)
x = Dense(20, activation="relu")(x)
outputs = Dense(1, activation="linear")(x)

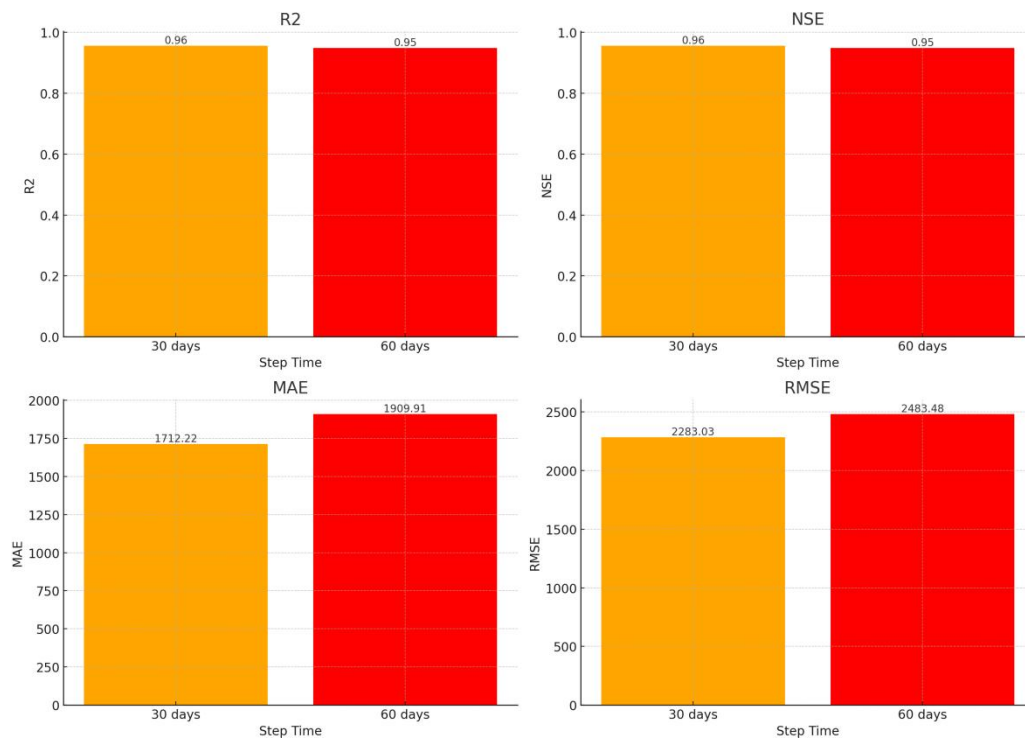
model = Model(inputs=inputs, outputs=outputs)
model.compile(optimizer="adam", loss="mean_squared_error")

# Model Summary
model.summary()

# Train the model
model.fit(X_train, y_train, validation_data=(X_test, y_test),
epochs=100, batch_size=64, verbose=1)
```

Hình 2.6. Code xây khai báo và huấn luyện mô hình

Ở cả 2 kịch bản đều được khai báo và huấn luyện mô hình với đoạn code ở hình 2.6, chỉ khác nhau là dữ liệu truyền vào là khác nhau như đã chia kịch bản ở trên. Sau nhiều lần huấn luyện các mô hình, dưới đây là so sánh kết quả giữa 2 kịch bản:



Hình 2.7. So sánh kết quả của 2 kịch bản

Quan sát hình 2.7, với màu vàng là các độ đo của kịch bản 1, màu đỏ là của kịch bản 2. Có thể thấy rằng ở cả 2 độ đo về độ chính xác và 2 độ đo về độ lỗi thì kịch bản 1 đều cho ra kết quả tốt hơn. Vì vậy, bằng việc chia làm 2 kịch bản là sử dụng 30, 60 ngày trước để dự báo ngày tiếp theo đã giúp nhóm tìm được khoảng thời gian phù hợp với bộ dữ liệu nhất và cũng đưa ra kết quả tốt nhất.

Kết luận cho mô hình Transformer + Kalman: Sử dụng 30 ngày trước để dự báo ngày tiếp theo cho thấy hiệu suất của mô hình + Kalman tốt hơn so với kịch bản sử dụng 60 ngày.

2.3. Mô hình BiLSTM kết hợp với Attention

Biến đổi và chia dữ liệu:

```
vcb.shape
✓ 0.0s
(3592, 5)

data = vcb.filter(['CLOSE'])
dataset = data.values
training_data_len = int(np.ceil( len(dataset) * 0.8 ))
training_data_len
✓ 0.0s
2874
```

Sử dụng hàm filter trong pandas để lấy dữ liệu riêng cột CLOSE, sau đó tách 80% dữ liệu để làm tập huấn luyện với hàm ceil trong numpy để làm tròn số dữ liệu lấy để huấn luyện.

Chuẩn hóa và xây dựng dữ liệu huấn luyện:

Chuẩn hóa dữ liệu

```
scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(dataset)
```

✓ 0.0s

Xây dựng và huấn luyện mô hình LSTM

```
train_data = scaled_data[0:int(training_data_len), :]
# Split the data
x_train = []
y_train = []
# chia 60 điểm dữ liệu vào x_train
# điểm dữ liệu thứ 61 vào y_train
# học từ 60 điểm dữ liệu trước
for i in range(70, len(train_data)):
    x_train.append(train_data[i-60:i, 0])
    y_train.append(train_data[i, 0])
    if i <= 61:
        print(x_train)
        print(y_train)
        print()

x_train, y_train = np.array(x_train), np.array(y_train)
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
```

✓ 0.0s

Sử dụng MinMaxScaler trong sklearn để chuẩn hóa dữ liệu về khoảng [0,1] điều này sẽ giúp tối ưu việc huấn luyện mô hình cho sau này.

Việc tạo dữ liệu huấn luyện sẽ được chia như sau:

Lấy 60 điểm dữ liệu trước đó đặt làm X_train

Lấy dữ liệu thứ 61 đặt làm y_train

Từ đó ta thu được bộ dữ liệu huấn luyện gồm X_train và y_train.

Huấn luyện mô hình:

```
from tensorflow.keras.layers import LSTM, Bidirectional, Dense, RepeatVector
from tensorflow.keras.layers import Attention
from tensorflow.keras.models import Sequential

model = Sequential()
model.add(Bidirectional(LSTM(128, return_sequences=True, input_shape=(x_train.shape[1], 1))))
model.add(Attention())
model.add(RepeatVector(x_train.shape[1]))
model.add(Bidirectional(LSTM(64, return_sequences=False)))
model.add(Dense(25))
model.add(Dense(1))

model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(x_train, y_train, batch_size=16, epochs=10)
```

✓ 59.3s

Sau khi thu được bộ dữ liệu huấn luyện sẽ đưa qua mô hình BiLSTM kết hợp với attention để thực hiện huấn luyện mô hình.

Tạo mô hình Sequential sau đó add các layer vào mô hình.

Layer attention được thêm vào thực hiện attention mechanism để tăng cường khả năng học tập của mô hình.

RepeatVector: Lặp lại đầu ra của LSTM trước đó để chuẩn bị cho việc đưa vào một LSTM hai chiều khác.

“return_sequences=True”: để trả về chuỗi đầy đủ thay vì chỉ output của lần cuối cùng.

“return_sequences=False”: chỉ trả về output của lần cuối cùng (không còn là chuỗi).

Kết quả tính toán độ chính xác của mô hình với các hàm tính toán độ chính xác

```
predictions = model.predict(x_test)
predictions = scaler.inverse_transform(predictions)

r2 = R2(y_test, predictions)
nse = NSE(y_test, predictions)
mae = MAE(y_test, predictions)
rmse = np.sqrt(np.mean((predictions - y_test) ** 2))
print("R2: ", r2)
print("NSE: ", nse)
print("MAE: ", mae)
print("RMSE: ", rmse)
```

23/23 ————— 2s 68ms/step
R2: 0.9499569269498501
NSE: 0.9486156284542654
MAE: 1842.8497018628134
RMSE: 2479.716040129308

Dựa vào thông số trên ta nhận thấy mô hình dự báo chính xác đến 0.95 với độ đo R2 và 0.94 với độ đo NSE, đối với độ đo sai số trung bình tuyệt đối MAE thì kết quả sai chỉ dừng lại ở khoảng 1842 và với RMSE sai số bình phương trung bình thì kết quả ở mức 2479.



Nhận xét: Kết quả dự báo có thể dự báo được đúng các xu hướng tăng giảm hay sideway như kết quả thực tế từ đó cho thấy mô hình có khả năng đưa vào thực tế để dự đoán giá cổ phiếu VCB.

2.4. Mô hình Temporal Convolutional Networks

Đọc dữ liệu đầu vào: lấy cột 'DATE' làm index và sắp xếp lại theo thời gian tăng dần loại bỏ các cột không cần thiết chỉ sử dụng cột 'CLOSE'.

```
df=pd.read_csv('vcb_24.csv',index_col='DATE')
df.index = pd.to_datetime(df.index, dayfirst=True)
df = df.sort_index()
df=df[['CLOSE']]
df
✓ 0.0s
```

Hình 2.4.1. Đọc dữ liệu

Tạo dữ liệu train và test: dữ liệu được chia làm hai phần train và test theo tỉ lệ 8:2 sau đó sẽ lấy liên tiếp các giá trị trong i ngày gán làm X và giá trị ngày i+1 làm nhãn Y.

```
# Chia dữ liệu thành train và test theo tỉ lệ 8:2
train_size = int(len(df) * 0.8)
train_df = df[:train_size]
test_df = df[train_size:]

# Tạo dữ liệu theo ngày dự đoán
def create_graph_data(df, window_size):
    x = []
    y = []
    for i in range(len(df) - window_size-1):
        x.append(df.iloc[i:i+window_size, 0])
        y.append(df.iloc[i+window_size, 0])
    return torch.tensor(x, dtype=torch.float), torch.tensor(y, dtype=torch.float)

window_size = 60
x_train, y_train = create_graph_data(train_df, window_size)
x_test, y_test=create_graph_data(test_df, window_size)
```

Hình 2.4.2. Tạo dữ liệu train và test

Mô hình được chạy với hai kịch bản là dùng 30 ngày và 60 ngày để dự đoán cho ngày tiếp theo.

Định nghĩa mô hình: mô hình được xây dựng với 4 tầng tích chập thời gian và hàm kích hoạt ReLU.


```
# Định nghĩa mô hình TCN
class TCN(nn.Module):
    def __init__(self, input_size, output_size, num_channels, kernel_size, dropout):
        super(TCN, self).__init__()
        self.tcn = nn.Sequential(
            nn.Conv1d(input_size, num_channels, kernel_size=kernel_size, padding=(kernel_size - 1) // 2),
            nn.ReLU(),
            nn.Dropout(dropout),
            nn.Conv1d(num_channels, num_channels, kernel_size=kernel_size, padding=(kernel_size - 1) // 2),
            nn.ReLU(),
            nn.Dropout(dropout),
            nn.Conv1d(num_channels, num_channels, kernel_size=kernel_size, padding=(kernel_size - 1) // 2),
            nn.ReLU(),
            nn.Dropout(dropout),
            nn.Conv1d(num_channels, output_size, kernel_size=1)
        )

    def forward(self, x):
        return self.tcn(x)
```

Hình 2.4.3. Định nghĩa mô hình

Khởi tạo và train mô hình:

Khởi tạo: mô hình được khởi tạo với các tham số như hình 2.4.4 với hàm mất mát được sử dụng là hàm MSE

```
# Khởi tạo mô hình
input_size = X_train.shape[1]
output_size = 1 # Dự đoán một giá trị (CLOSE)
num_channels = 128
kernel_size = 3
dropout = 0.01
learning_rate = 0.001
epochs = 200

model = TCN(input_size, output_size, num_channels, kernel_size, dropout)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
```

Hình 2.4.4. Khởi tạo mô hình

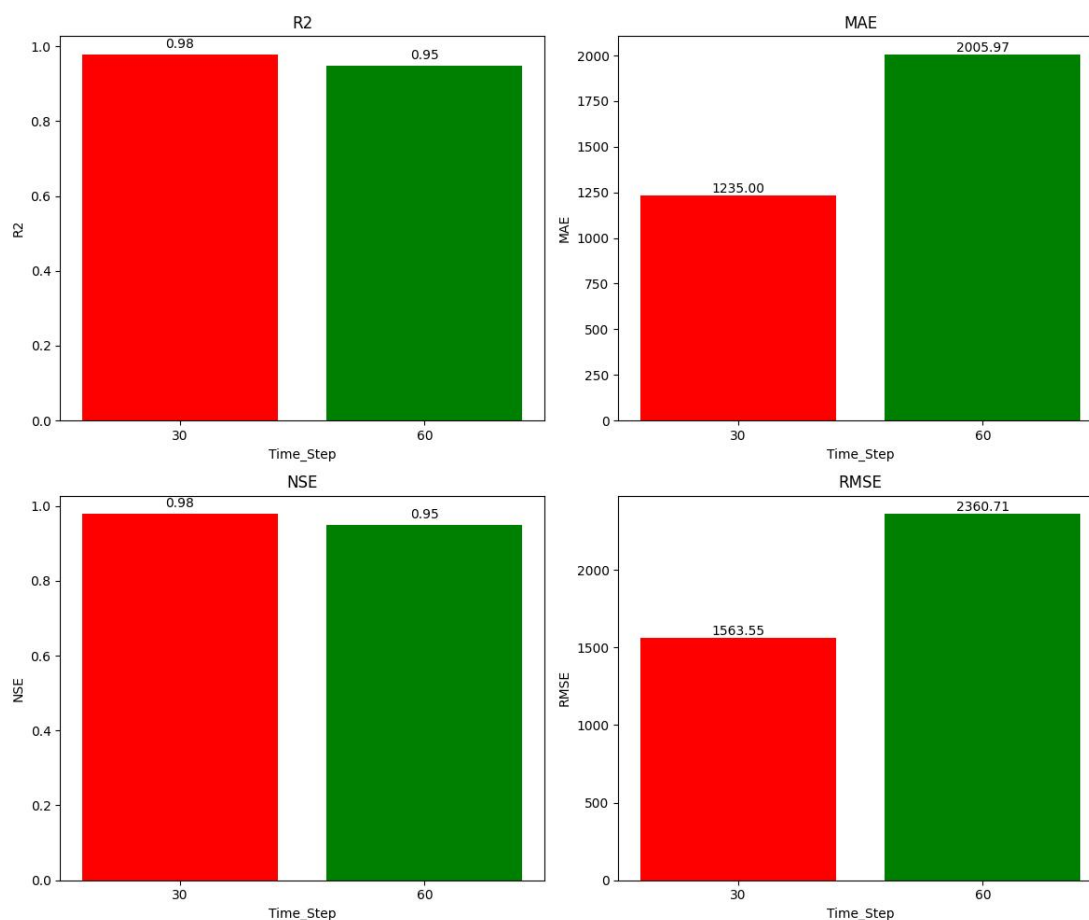
Train mô hình:

```
# Huấn luyện mô hình TCN
for epoch in range(epochs):
    model.train()
    for batch_X, batch_y in train_loader:
        batch_X, batch_y = batch_X.to(device), batch_y.to(device)
        optimizer.zero_grad()
        outputs = model(batch_X.permute(1,0)) # TCN yêu cầu (batch)
        loss = criterion(outputs.squeeze(), batch_y)
        loss.backward()
        optimizer.step()

    print(f"Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}")
```

Hình 2.4.5. Train mô hình

So sánh kết quả chạy:



Hình 2.4.6. So sánh kết quả

Hình 2.4.6 là kết quả chạy hai kịch bản nói trên, cột màu đỏ của kịch bản 30 ngày cột màu xanh của kịch bản 60 ngày, có thể thấy được kết quả của các độ đo khá tốt đặc biệt kịch bản 30 ngày cho kết quả tốt hơn so với 60 ngày

Kết luận mô hình Temporal Convolutional Networks: Mô hình hiệu quả trong việc dự báo chuỗi thời gian ngắn hạn và dài hạn. Cho kết quả tốt hơn với những khoảng thời gian ngắn như 30 ngày so với 60 ngày.

2.5. Mô hình Garch

Biến động (volatility)ⁱⁱⁱ đo lường mức độ dao động của giá của một công cụ tài chính trong một khoảng thời gian nhất định. Biến động không đo lường hướng của sự thay đổi giá, mà chỉ đo lường mức độ phân tán của chúng. Giá dao động càng mạnh thì mức độ biến động càng cao.

Biến Động Lịch Sử (Historical Volatility - HV) hoặc Biến Động Đã Thực Hiện: Đây là biến động thực tế được quan sát trong một khoảng thời gian trước đây, như một tháng hoặc một năm.

Thay đổi phần trăm hàng ngày của giá có thể tính bằng phương pháp `pct_change()` trong Python.

```
vcb['returns'] = 100 * vcb.Close.pct_change().dropna()

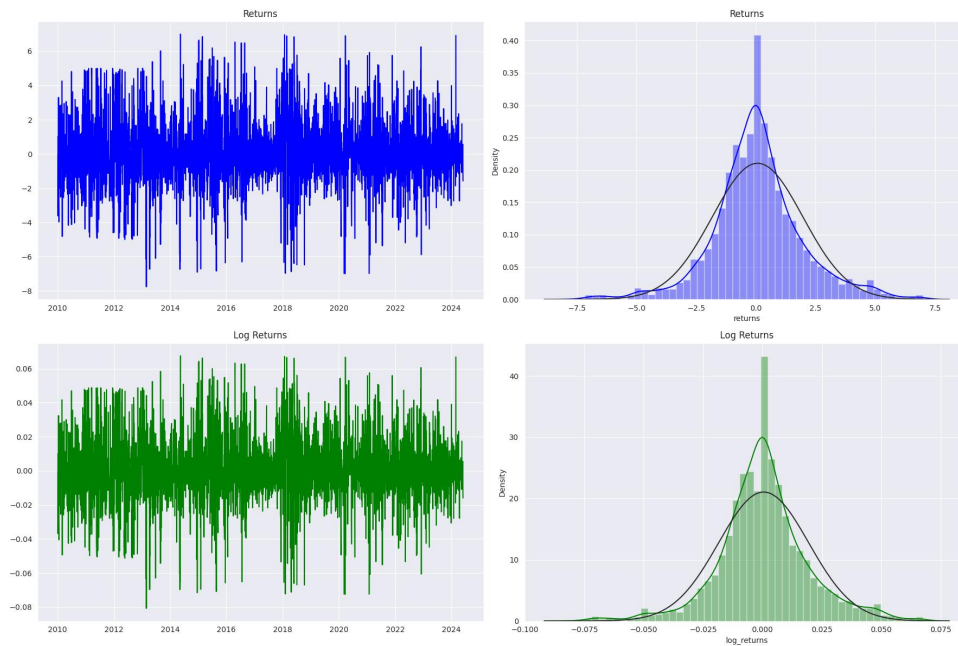
vcb['log_returns'] = np.log(vcb.Close/vcb.Close.shift(1))

vcb.head()
```

	Open	High	Low	Close	Volume	returns	log_returns
Date							
2010-01-04	13842	14519	13842	14519	2032674	NaN	NaN
2010-01-05	15020	15167	14519	14667	3175256	1.019354	0.010142
2010-01-06	14166	14667	14136	14136	1629290	-3.620372	-0.036875
2010-01-07	14136	14166	13842	13842	704530	-2.079796	-0.021017
2010-01-08	14019	14078	13812	13842	1193887	0.000000	0.000000

Lợi Ích của Việc Sử Dụng Lợi Nhuận Log:

- **Ổn Định:** Lợi nhuận log giúp loại bỏ các đặc tính không ổn định của dữ liệu chuỗi thời gian.
- **Tính Cộng Dồn:** Lợi nhuận log có thể được cộng dồn qua các khoảng thời gian để cho ra lợi nhuận tích lũy.
- **Chuẩn Hóa:** Lợi nhuận log chuẩn hóa quy mô của các thay đổi giá, làm cho chúng phù hợp hơn cho một số loại mô hình toán học.



Returns và Log Returns của VCB

```
vcb.returns.describe()
✓ 0.0s

count    3591.000000
mean      0.067912
std       1.897321
min       -7.758922
25%      -0.931974
50%       0.000000
75%       0.967136
max        6.989506
Name: returns, dtype: float64

vcb.log_returns.describe()
✓ 0.0s

count    3591.000000
mean      0.000499
std       0.018959
min       -0.080765
25%      -0.009363
50%       0.000000
75%       0.009625
max        0.067561
Name: log_returns, dtype: float64
```

Phân Tích Biểu Đồ và Mô Tả Thống Kê

1. Return

Biểu đồ Return cho thấy sự biến động mạnh mẽ với nhiều giá trị đỉnh và đáy. Điều này phản ánh sự dao động giá của cổ phiếu Vietcombank qua thời gian.

- Giá trị trung bình của Return là 0.067912, cho thấy xu hướng tăng nhẹ.
- Độ lệch chuẩn cao (1.897321) cho thấy mức độ biến động lớn.
- Giá trị nhỏ nhất (-7.758922) và giá trị lớn nhất (6.989506) cho thấy có những giai đoạn biến động rất mạnh.

2. Log Return

Biểu đồ Log Return có vẻ ít biến động hơn so với Return và giá trị nằm gần trung tâm hơn.

- **Giá trị trung bình của Log Return là 0.000499, gần như bằng 0, cho thấy không có xu hướng tăng hoặc giảm rõ ràng.**
- **Độ lệch chuẩn thấp (0.018959) cho thấy mức độ biến động ít hơn so với Return.**
- **Giá trị nhỏ nhất (-0.080765) và giá trị lớn nhất (0.067561) đều nằm trong phạm vi hẹp hơn so với Return, cho thấy biến động giá ít mạnh hơn.**

Hàm Tính Độ Biến Động Thực Tế Hàng Ngày

Tính toán độ biến động thực tế hàng ngày bằng cách lấy căn bậc hai của tổng bình phương của các lợi nhuận logarit trong một khoảng thời gian cụ thể.

```
def realized_volatility_daily(series_log_return):  
    """  
    Get the daily realized volatility which is calculated as the square root  
    of sum of squares of log returns within a specific window interval  
    """  
    n = len(series_log_return)  
    return np.sqrt(np.sum(series_log_return**2)/(n - 1))  
✓ 0.0s
```

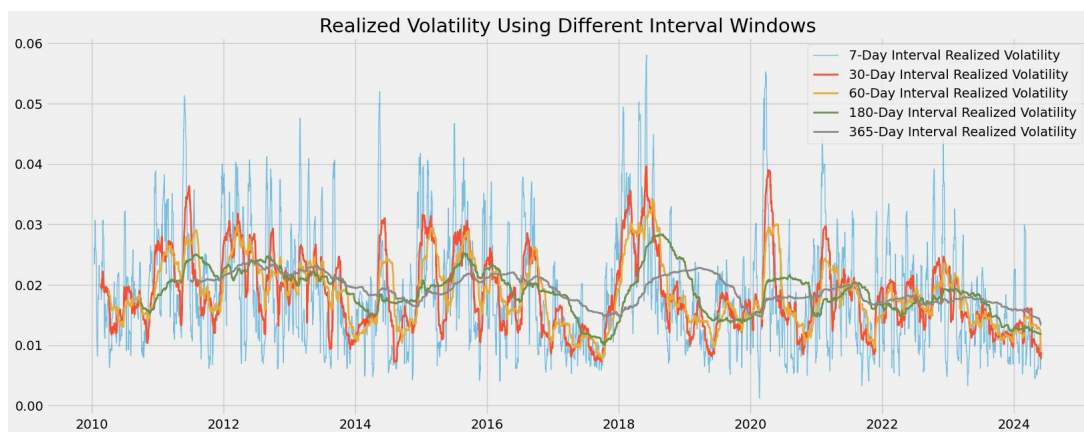
Tính toán độ biến động thực tế cho các khoảng thời gian để chọn kịch bản

```
intervals = [7, 30, 60, 180, 365]
vols_vcb = {}

# ITERATE OVER intervals LIST
for i in intervals:
    # GET DAILY LOG RETURNS USING THAT INTERVAL
    vols = vcb.log_returns.rolling(window=i)\
        .apply(realized_volatility_daily).values

    vols_vcb[i] = vols

# CONVERT vols_vcb FROM DICTIONARY TO PANDAS DATAFRAME
vols_vcb = pd.DataFrame(vols_vcb, columns=intervals, index=vcb.index)
```



Sau khi tính toán độ biến động thực tế cho các khoảng thời gian, nhóm quyết định chọn khoảng thời gian 30 ngày

Lý Do Chọn Khoảng Thời Gian 30 Ngày

1. Giảm Nhiễu Loạn Dữ Liệu:
 - Khoảng thời gian 7 ngày: Thường quá ngắn và có thể gây ra quá nhiều nhiễu loạn, khiến việc quan sát các mẫu có ý nghĩa trở nên khó khăn.
 - Khoảng thời gian 30 ngày: Giảm nhiễu loạn và cung cấp một bức tranh rõ ràng hơn về xu hướng và mẫu trong dữ liệu.
2. Tránh Làm Mượt Quá Mức:
 - Khoảng thời gian dài hơn (như 60, 180 ngày): Có thể làm mượt dữ liệu quá mức, dẫn đến việc mất đi các dao động quan trọng và độ biến động có thể trở lại giá trị trung bình nhanh hơn.
 - Khoảng thời gian 30 ngày: Đủ dài để nắm bắt các xu hướng nhưng không quá dài để làm mượt các biến động quan trọng.
3. Tránh Lãng Phí Dữ Liệu:

- Khoảng thời gian quá dài: Sử dụng khoảng thời gian quá dài có thể dẫn đến việc bỏ qua nhiều dữ liệu ở phần đầu của tập dữ liệu, làm giảm số lượng điểm dữ liệu có sẵn cho phân tích.
- Khoảng thời gian 30 ngày: Giúp tối ưu hóa việc sử dụng dữ liệu có sẵn mà không lãng phí quá nhiều điểm dữ liệu ban đầu.

Tính toán độ biến động thực tế hiện tại và tương lai của lợi nhuận logarit cho cổ phiếu Vietcombank (VCB) bằng cách sử dụng khoảng thời gian 30 ngày (INTERVAL_WINDOW) và dịch chuyển dữ liệu 7 ngày (n_future).

```
INTERVAL_WINDOW = 30
n_future = 7

# GET BACKWARD LOOKING REALIZED VOLATILITY
vcb['vol_current'] = vcb.log_returns.rolling(window=INTERVAL_WINDOW)\
    .apply(realized_volatility_daily)

# GET FORWARD LOOKING REALIZED VOLATILITY
vcb['vol_future'] = vcb.log_returns.shift(-n_future)\
    .rolling(window=INTERVAL_WINDOW)\
    .apply(realized_volatility_daily)

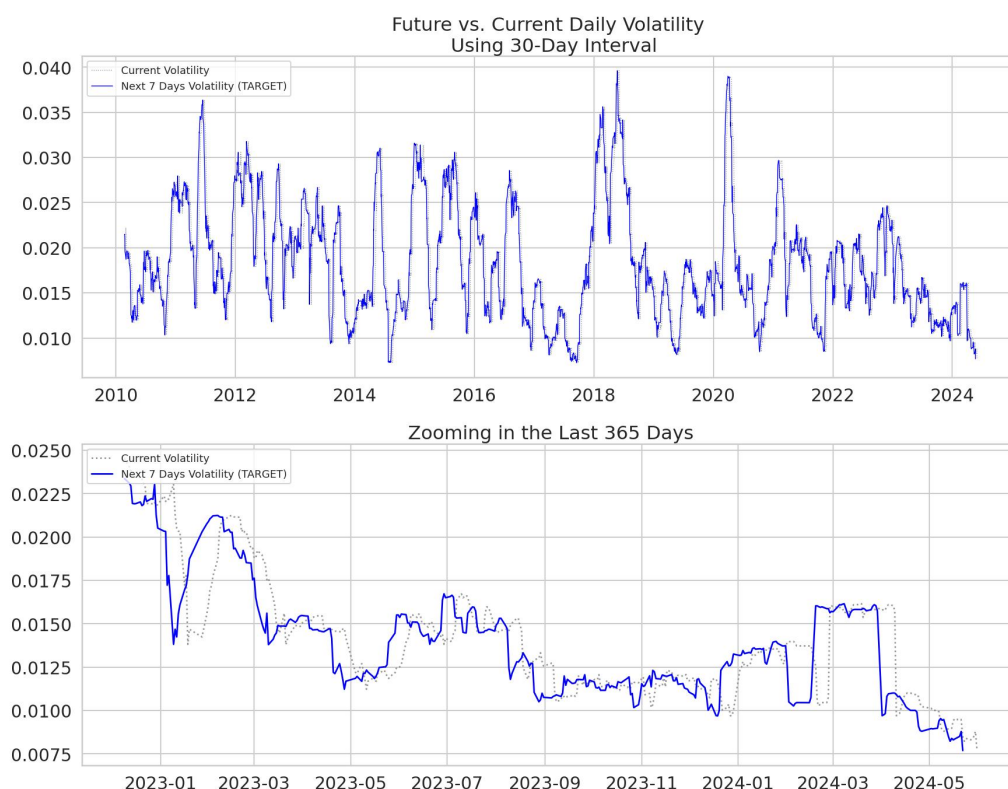
vcb.describe()
```

vol_current:

- Đây là độ biến động thực tế hiện tại.
- Nó được tính toán bằng cách sử dụng lợi nhuận logarit của cổ phiếu trong khoảng thời gian 30 ngày trước đó.
- **Mục đích:** Đo lường mức độ dao động của giá cổ phiếu trong 30 ngày qua.
- **Ví dụ:** Nếu ngày hôm nay là ngày 31, thì vol_current tại ngày 31 sẽ tính toán độ biến động từ ngày 1 đến ngày 30.

vol_future:

- Đây là độ biến động thực tế tương lai.
- Nó được tính toán bằng cách dịch chuyển lợi nhuận logarit 7 ngày về phía trước, sau đó sử dụng lợi nhuận logarit trong khoảng thời gian 30 ngày để tính toán độ biến động.
- **Mục đích:** Đo lường mức độ dao động của giá cổ phiếu trong 30 ngày tới, so với thời điểm hiện tại
- **Ví dụ:** Nếu ngày hôm nay là ngày 31, thì vol_future tại ngày 31 sẽ tính toán độ biến động từ ngày 8 đến ngày 37.



Xét 365 ngày gần nhất:

- Trong khoảng thời gian 365 ngày gần nhất, độ biến động thực tế hiện tại và tương lai vẫn duy trì sự dao động tương tự như xu hướng chung.
- Có thể thấy rõ các thay đổi nhỏ và xu hướng ngắn hạn của thị trường.

Chi Tiết Các Giai Đoạn:

- Đầu năm 2023, có một giai đoạn biến động mạnh, sau đó giảm dần đến giữa năm 2023.
- Từ giữa năm 2023 đến cuối năm 2023, độ biến động giữ mức thấp hơn nhưng vẫn có những dao động nhỏ.
- Đầu năm 2024, có một vài đợt tăng đột biến ngắn hạn trong độ biến động.

Chia Tập Dữ Liệu Thành Các Phần Train-Validation-Test

Thông Tin về Tập Dữ Liệu

- Tổng cộng hơn 3400 điểm dữ liệu sử dụng được, bao gồm một khoảng thời gian hơn 14 năm từ tháng 1 năm 2010 đến tháng 6 năm 2024. Chia Tập Dữ Liệu

Kích Thước Các Phần

1. **Test Set:** 30 điểm dữ liệu gần nhất

2. **Validation Set:** 1 năm (365 ngày)
3. **Training Set:** Phần còn lại

```
# PRE-DETERMINE DESIRED TEST & VALIDATION SIZES
test_size = 30
val_size = 365

# CONVERT TO INDICES
split_time_1 = len(vcb) - (val_size + test_size)
split_time_2 = len(vcb) - test_size

# GET CORRESPONDING DATETIME INDICES FOR EACH SET
train_idx = vcb.index[:split_time_1]
val_idx = vcb.index[split_time_1:split_time_2]
test_idx = vcb.index[split_time_2:]
✓ 0.0s
```

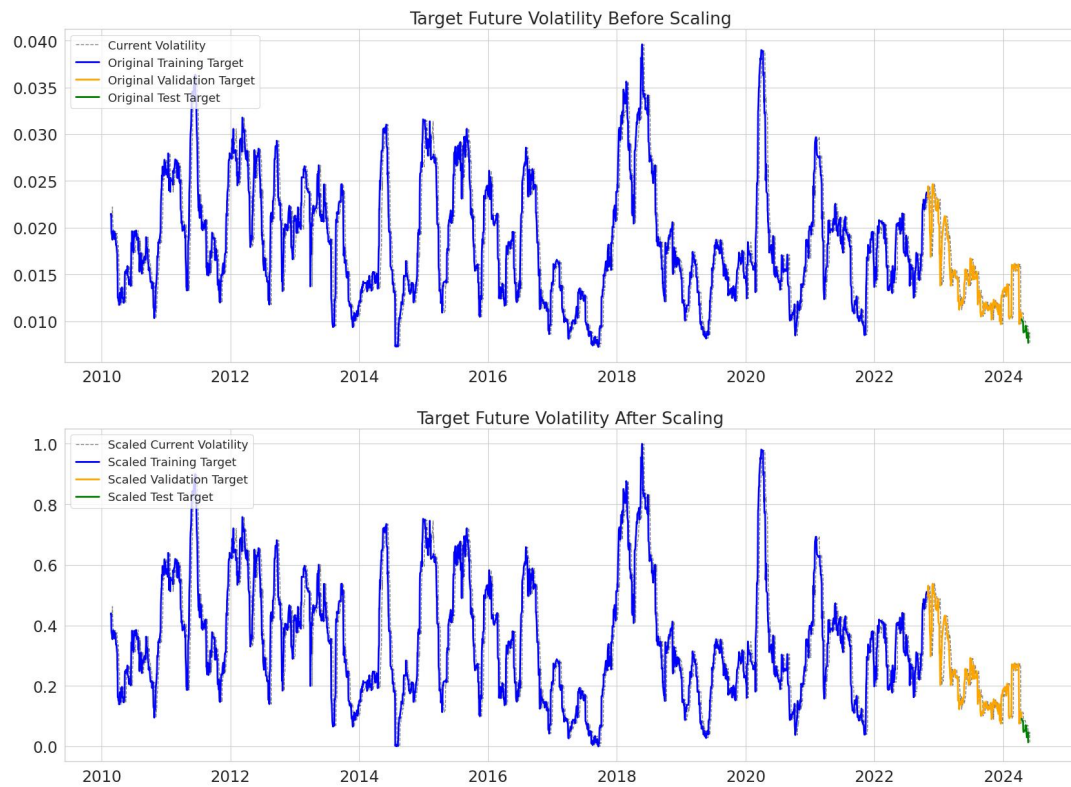
```
print(f'TRAINING \tFrom: {train_idx[0]} \tto: {train_idx[-1]} \t{len(train_idx)} days')
print(f'VALIDATION \tFrom: {val_idx[0]} \tto: {val_idx[-1]} \t{len(val_idx)} days')
print(f'TEST \t\tFrom: {test_idx[0]} \tto: {test_idx[-1]} \t{len(test_idx)} days')
✓ 0.0s
```

TRAINING	From: 2010-01-05 00:00:00	to: 2022-10-27 00:00:00	3196 days
VALIDATION	From: 2022-10-28 00:00:00	to: 2024-04-15 00:00:00	365 days
TEST	From: 2024-04-16 00:00:00	to: 2024-05-31 00:00:00	30 days

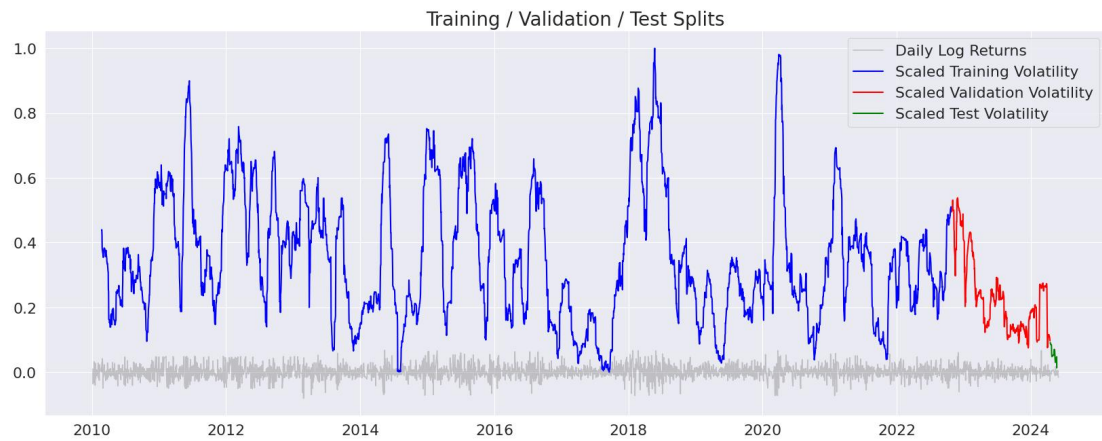
Mục tiêu dự đoán (target): vol_future - đại diện cho độ biến động thực tế hàng ngày của 7 ngày tương lai (trung bình độ biến động hàng ngày từ $t + n_future - \text{INTERVAL_WINDOW}$ đến $t + n_future$).

Ví dụ: Với n_future là 7 và INTERVAL_WINDOW là 30, giá trị muốn dự đoán tại bước thời gian t sẽ là trung bình độ biến động hàng ngày từ bước thời gian $t-22$ đến bước thời gian $t+7$.

Tiền xử lý dữ liệu bằng MinMaxScaler



Train-Validation-Test Visualization



Chạy mô hình GARCH(1, 1)

Khai báo mô hình

```
gm_1 = arch_model(r_train, vol='GARCH', p=1, q=1)
result_1 = gm_1.fit(disps='off')
print()
print(result_1.summary())
```

✓ 0.0s

Constant Mean - GARCH Model Results

```
=====
Dep. Variable:          returns    R-squared:                0.000
Mean Model:             Constant Mean  Adj. R-squared:           0.000
Vol Model:              GARCH         Log-Likelihood:         -6466.75
Distribution:           Normal        AIC:                   12941.5
Method:                Maximum Likelihood  BIC:                   12965.8
Date:                  Sun, Jun 16 2024  No. Observations:      3196
Time:                  15:51:27         Df Residuals:           3195
                                Df Model:              1
                                Mean Model
=====
```

	coef	std err	t	P> t	95.0% Conf. Int.
mu	0.0670	2.913e-02	2.300	2.144e-02	[9.908e-03, 0.124]

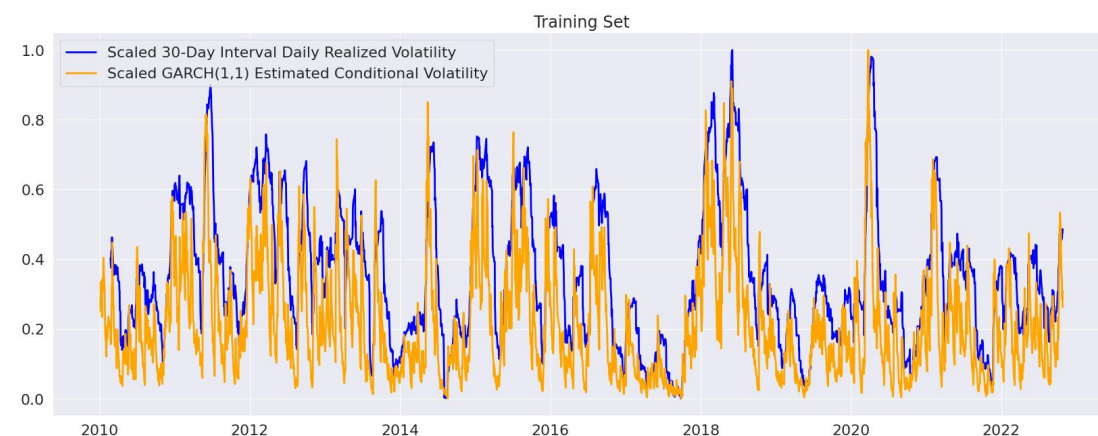
Volatility Model

```
=====
```

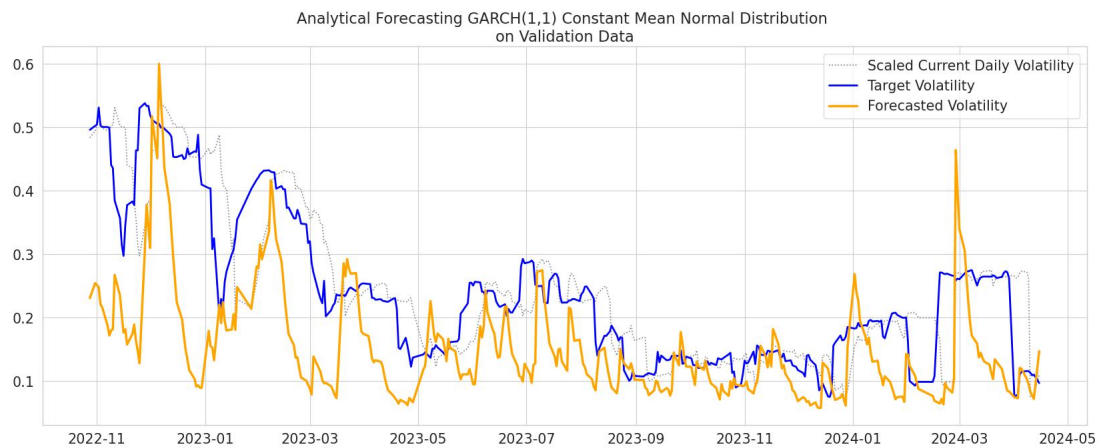
	coef	std err	t	P> t	95.0% Conf. Int.
omega	0.2384	9.954e-02	2.395	1.661e-02	[4.333e-02, 0.434]
alpha[1]	0.1110	2.432e-02	4.563	5.032e-06	[6.331e-02, 0.159]
beta[1]	0.8259	4.820e-02	17.133	8.482e-66	[0.731, 0.920]

```
=====
```

Kết quả trên tập Train



Kết quả trên tập Validation



Kết quả đánh giá mô hình

```
# APPEND METRICS OUTPUTS TO perf_df DATAFRAME
log_perf(y_val_scaled, gm_1_preds_scaled,
         'GARCH(1,1), Constant Mean, Normal Dist')
```

✓ 0.0s

	Model	Validation RMSPE	Validation RMSE
0	GARCH(1,1), Constant Mean, Normal Dist	0.419829	0.126422

Độ Chính Xác Của Mô Hình:

- RMSPE và RMSE đều cho thấy mô hình GARCH(1,1) có độ chính xác khá cao khi dự đoán độ biến động của cổ phiếu Vietcombank.
- RMSPE ở mức 41.98% có thể được coi là khá tốt trong bối cảnh tài chính, nơi mà sự biến động và các yếu tố ngẫu nhiên thường rất cao.

Ý Nghĩa Thực Tiễn:

- RMSE cho thấy rằng độ lệch trung bình giữa giá trị thực tế và giá trị dự đoán là khoảng 12.64%. Điều này có nghĩa là mô hình có thể dự đoán khá chính xác độ biến động hàng ngày của cổ phiếu Vietcombank.
- Một RMSE thấp cho thấy mô hình có khả năng nắm bắt được các xu hướng và mẫu trong dữ liệu một cách hiệu quả.

KẾT LUẬN

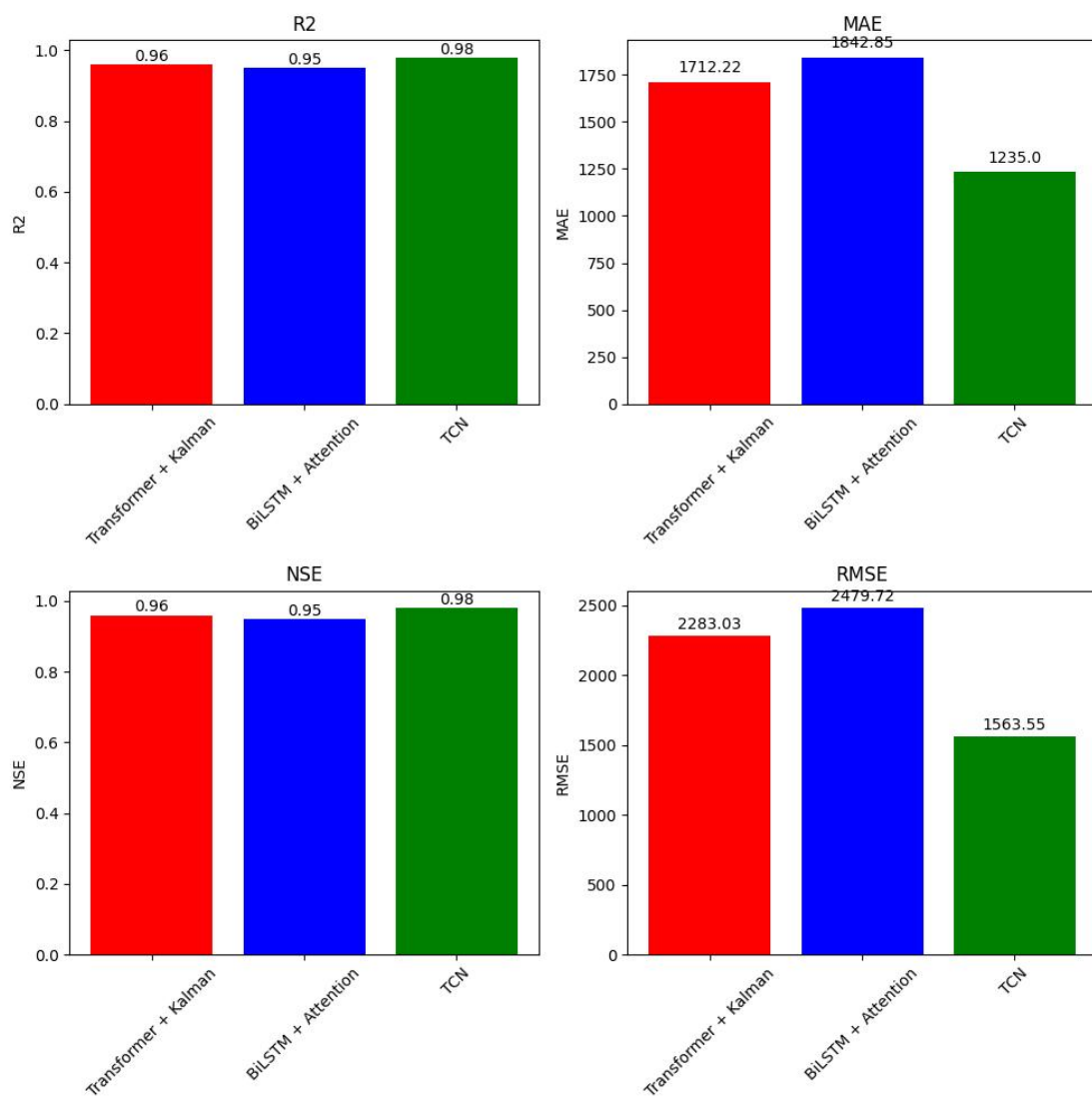
Sau khi đã xây dựng các mô hình Transformer + Kalman, BiLSTM + Attention, TCN và GARCH để thực hiện dự báo giá cổ phiếu Vietcombank, nhóm đã thu được một số kết quả như sau:

Dựa vào biểu đồ, mô hình có độ đo R^2 cao nhất là TCN (0.98), tiếp đến là Transformer + Kalman (0.96) và thấp nhất là BiLSTM + Attention (0.95). Điều này cho thấy TCN là mô hình có khả năng giải thích biến động trong dữ liệu tốt nhất. Về MAE, mô hình TCN đạt giá trị thấp nhất (1235.00), biểu thị rằng TCN dự báo với sai số tuyệt đối trung bình nhỏ nhất, cho thấy độ chính xác cao hơn so với hai mô hình còn lại. Tương tự, TCN cũng đạt giá trị NSE cao nhất (0.98), thể hiện khả năng dự báo tốt nhất so với các mô hình khác.

Mặc dù Transformer + Kalman và BiLSTM + Attention có hiệu suất kém hơn, nhưng BiLSTM + Attention lại có độ chính xác thấp nhất trong các mô hình được so sánh. Với giá trị RMSE thấp nhất (1563.55), TCN dự báo với độ lệch chuẩn trung bình của lỗi nhỏ nhất, phản ánh rằng dự báo của nó ổn định hơn và ít biến động hơn.

Mô hình GARCH, mặc dù là một công cụ mạnh mẽ để dự báo sự biến động của dữ liệu tài chính, lại gặp khó khăn trong việc tính toán các chỉ số độ đo tương tự như các mô hình khác trong bối cảnh này. GARCH chuyên dùng để dự báo biến động (volatility) thay vì giá trị cụ thể, do đó không thể so sánh trực tiếp về độ đo R^2 , MAE, NSE, và RMSE. Bản chất của GARCH là tập trung vào việc nắm bắt sự không đều đặn trong biến động, điều này rất hữu ích trong việc quản lý rủi ro hơn là dự báo chính xác giá cổ phiếu.

Kết luận, TCN là mô hình tốt nhất trong cả bốn chỉ số (R^2 , MAE, NSE, RMSE), chứng tỏ rằng nó có khả năng dự báo chính xác và ổn định nhất cho giá cổ phiếu Vietcombank. Với những kết quả này, có thể kết luận rằng mô hình TCN là lựa chọn phù hợp nhất cho bài toán dự báo giá cổ phiếu Vietcombank. Trong khi đó, GARCH vẫn giữ vai trò quan trọng trong việc dự báo biến động và quản lý rủi ro.



Biểu đồ cột so sánh độ đo của các mô hình

TÀI LIỆU THAM KHẢO

- ⁱ Pham Nam, “Giải mã kiến trúc Transformer trong paper *Attention is all you need*”, <https://viblo.asia/p/giai-ma-kien-truc-transformer-trong-paper-attention-is-all-you-need-RnB5pJeGZPG>, truy cập ngày 10/06/2024.
- ⁱⁱ Lê Quang Tiến, “Kalman Filter và bài toán chuỗi thời gian”, <https://thetalog.com/machine-learning/kalman-filter/>, truy cập ngày 11/06/2024.
- ⁱⁱⁱ Bollerslev, T. (1986). Generalized Autoregressive Conditional Heteroskedasticity. *Journal of Econometrics*, 31(3), 307-327.