TRƯỜNG ĐẠI HỌC MỎ - ĐỊA CHẤT

BÁO CÁO TỔNG KẾT ĐỀ TÀI
NCKH SINH VIÊN

XÂY DỰNG MÔ HÌNH DỰ BÁO CHUỖI THỜI GIAN BẰNG PYTHON

Hà Nội, 2023

TRƯỜNG ĐẠI HỌC MỎ - ĐỊA CHẤT

BÁO CÁO TỔNG KẾT ĐỀ TÀI
NCKH SINH VIÊN

XÂY DỰNG MÔ HÌNH DỰ BÁO CHUỖI THỜI GIAN BẰNG PYTHON

Trưởng nhóm nghiên cứu: Vũ Đình Hoàng - Công nghệ thông tin (CLC) A1 K66

Thành viên tham gia thực hiện:

Nguyễn Tuấn Nghĩa - Công nghệ phần mềm K65C

Nguyễn Ngọc Minh - Công nghệ thông tin (CLC) A1 K66

Chu Tiến Sơn - Công nghệ thông tin (CLC) A1 K66

Người hướng dẫn: Ths. Nguyễn Thu Hằng

Hà Nội, 2023

MỤC LỤC

DANH MỤC CÁC HÌNH VẼ

DANH MỤC CÁC BẢNG BIỂU

MỞ ĐẦU

1. Tổng quan về mô hình dự báo chuỗi thời gian "Time Series Forecasting"

1.1. Dự báo là gì?

Dự báo là sự tiên đoán những vấn đề sẽ xảy ra trong tương lai dựa trên một cơ sở nào đó. Đây là một vấn đề luôn nhận được sự quan tâm của nhiều nhà khoa học, nhà quản lý bởi vì nó có một vai trò rất quan trọng trong thực tế. Tuy nhiên cho đến nay, dự báo vẫn là bài toán chưa có lời giải cuối cùng (Abbasov & Mamedova, 2003; Tai, 2019). Trong thống kê, dựa trên dữ liệu quá khứ, mô hình để dự báo cho tương lai được thiết lập. Đối với dữ liệu dạng chuỗi, hồi quy và chuỗi thời gian là hai mô hình được áp dụng phổ biến ngày nay.

Time Series Forecasting:

Dự báo chuỗi thời gian (Time Series Forecasting) là một kỹ thuật dự đoán các sự kiện thông qua một chuỗi thời gian. Nó dự đoán các sự kiện trong tương lai bằng cách phân tích các xu hướng trong quá khứ, với giả định rằng các xu hướng trong tương lai sẽ tương tự như vậy.

Dự báo chuỗi thời gian cũng là một lĩnh vực quan trọng của học máy và có thể được coi là một vấn đề học tập có giám sát. Các phương pháp học máy như:

Regression,

Neural Networks,

Support Vector Machines,

Random Forests,

XGBoost

có thể được áp dụng trong trường hợp này.

Dự báo chuỗi thời gian thường được sử dụng cùng với việc phân tích. Phân tích chuỗi thời gian liên quan đến việc phát triển các mô hình tìm kiếm tri thức trong dữ liệu. Dự báo chuỗi thời gian thực hiện bước tiếp theo với kiến thức vừa tìm được. Nó

đòi hỏi phải phát triển các mô hình dựa trên dữ liệu trước đó và áp dụng chúng để thực hiện các quan sát và hướng dẫn các quyết định chiến lược trong tương lai.

1.2. Những phương pháp dự báo chuỗi thời gian

Mô hình Machine Learning:

Mô hình ARIMA (AutoRegressive Integrate Moving Average): Dựa trên giả thuyết chuỗi dừng và phương sai sai số không đổi. Mô hình sử dụng đầu vào chính là những tín hiệu quá khứ của chuỗi được dự báo để dự báo nó. Các tín hiệu đó bao gồm: chuỗi tự hồi qui AR (auto regression) và chuỗi trung bình trượt MA (moving average). Hầu hết các chuỗi thời gian sẽ có xu hướng tăng hoặc giảm theo thời gian, do đó yếu tố chuỗi dừng thường không đạt được. Trong trường hợp chuỗi không dừng thì ta sẽ cần biến đổi sang chuỗi dừng bằng sai phân. Khi đó tham số đặc trưng của mô hình sẽ có thêm thành phần bậc của sai phân d và mô hình được đặc tả bởi 3 tham số ARIMA (p, d, q).

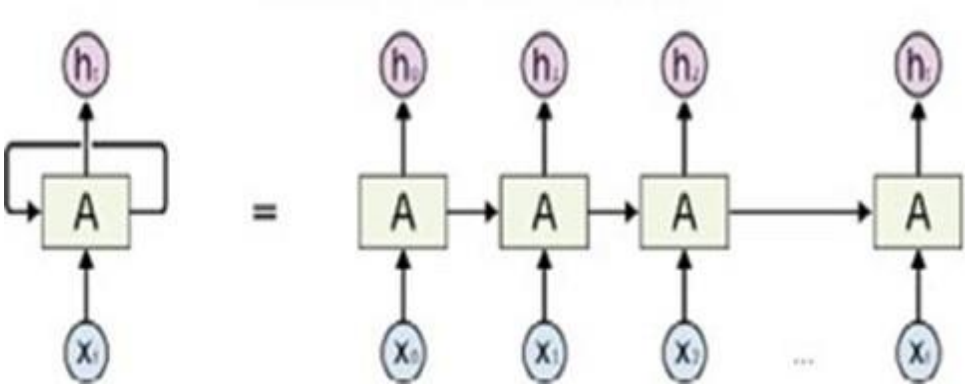
Mô hình SARIMA: Về bản chất đây là mô hình ARIMA nhưng được điều chỉnh đặc biệt để áp dụng cho những chuỗi thời gian có yếu tố mùa vụ. Như chúng ta đã biết về bản chất ARIMA chính là mô hình hồi qui tuyến tính nhưng mối quan hệ tuyến tính thường không giải thích tốt chuỗi trong trường hợp chuỗi xuất hiện yếu tố mùa vụ. Chính vì thế, bằng cách tìm ra chu kì của qui luật mùa vụ và loại bỏ nó khỏi chuỗi ta sẽ dễ dàng hồi qui mô hình theo phương pháp ARIMA.

Mô hình ARIMAX: Là một dạng mở rộng của model ARIMA. Mô hình cũng dựa trên giải định về mối quan hệ tuyến tính giữa giá trị và phương sai trong quá khứ với giá trị hiện tại và sử dụng phương trình hồi qui tuyến tính được suy ra từ mối quan hệ trong quá khứ nhằm dự báo tương lai. Mô hình sẽ có thêm một vài biến độc lập khác và cũng được xem như một mô hình hồi qui động (hoặc một số tài liệu tiếng việt gọi là mô hình hồi qui động thái). Về bản chất ARIMAX tương ứng với một mô hình hồi qui đa biến nhưng chiếm lợi thế trong dự báo nhờ xem xét đến yếu tố tự tương quan được biểu diễn trong phần dư của mô hình. Nhờ đó cải thiện độ chính xác.

Mô hình GARCH: Các giả thuyết về chuỗi dừng và phương sai sai số không đổi đều không dễ đạt được trong thực tế. Trái lại phương sai sai số biến đổi rất dễ xảy ra đối với các chuỗi tài chính, kinh tế bởi thường có những sự kiện không mong đợi và cú sốc kinh tế không lường trước khiến biến động phương sai của chuỗi thay đổi. Trong trường hợp đó nếu áp dụng ARIMA thì thường không mang lại hiệu quả cao cho mô hình. Các nhà kinh tế lượng và thống kê học đã nghĩ đến một lớp mô hình mà có thể dự báo được phương sai để kiểm soát các thay đổi không mong đợi. Dựa trên qui luật của phương sai, kết quả dự báo chuỗi sẽ tốt hơn so với trước đó.

Mô hình Deep learning:

Mô hình RNN (Recurrent Neural Network): Mạng nơ-ron hồi quy (RNN) là loại mạng có cấu trúc được thiết kế để xử lý chuỗi dữ liệu tuần tự. RNN không chỉ sử dụng dữ liệu đầu vào mà còn sử dụng các đầu ra trước đó để dự đoán đầu ra hiện tại. Sự liên kết giữa các nút mạng trong mô hình RNN tạo nên một sơ đồ có hướng và bản thân bộ nhớ trong được sử dụng để xử lý các chuỗi đầu vào linh hoạt. Trạng thái của mỗi nút thay đổi theo thời gian bởi hàm kích hoạt có giá trị thực. Mô hình huấn luyện trong RNN được xác định bằng cách chuyển đổi giữa các trạng thái nên nó luôn có cùng kích thước đầu vào. Mặt khác, cùng một hàm chuyển đổi sẽ có cùng hệ số ở mỗi bước đã được sử dụng trong hệ thống.

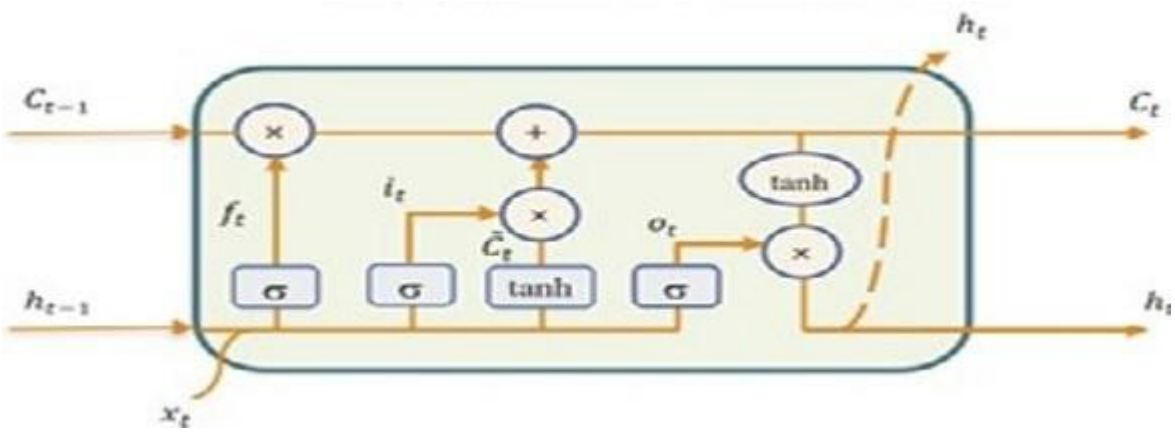


Hình 0-1 Mô hình RNN

Mạng RNN có cấu trúc tuần hoàn, có thể ghi nhớ các đầu vào trước đó, ghi nhớ từng thông tin theo thời gian, do đó rất hữu ích trong dự đoán cho dữ liệu chuỗi thời gian. Thậm chí, mạng RNN còn được sử dụng với các lớp mạng tích chập để mở rộng

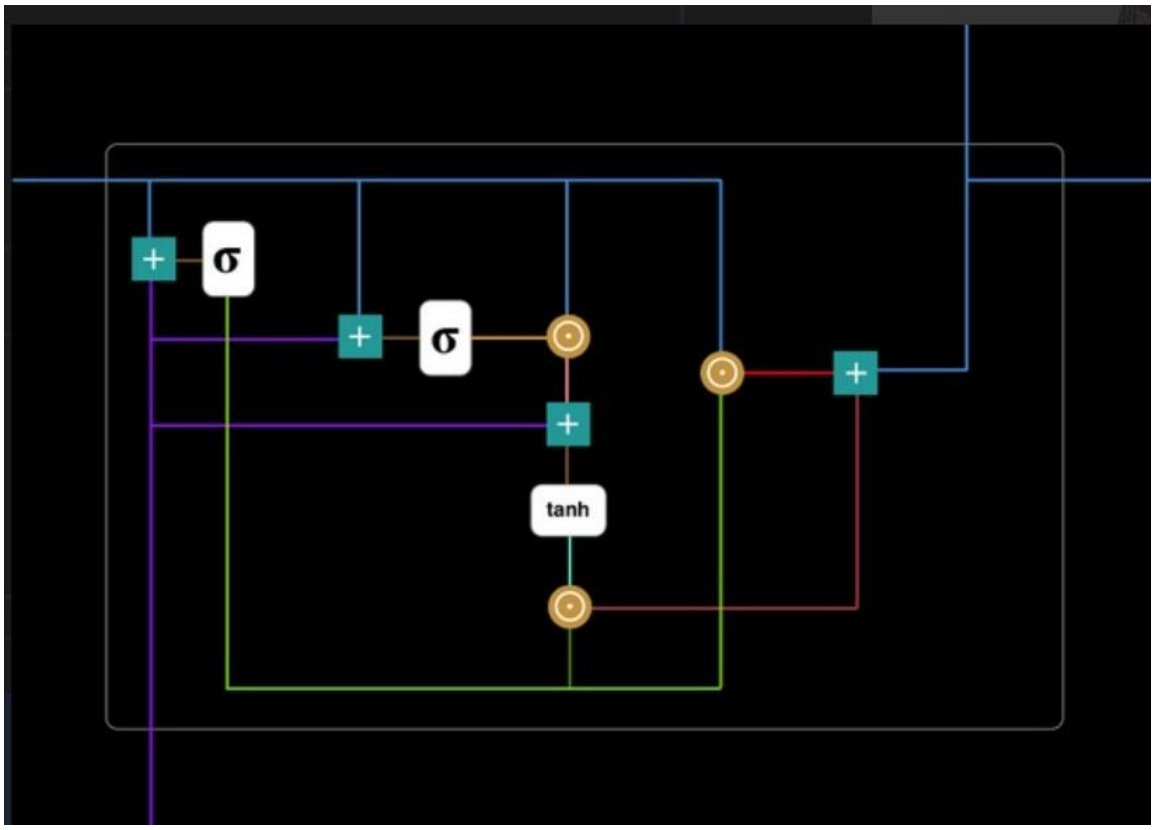
các giá trị trong ma trận pixel một cách hiệu quả. Ngoài ra, mối tương quan giữa các mẫu cũng được giả định thông qua các mô hình hóa chuỗi dữ liệu trong mạng RNN.

Mô Hình LSTM (Long Short - Term Memory): Mạng LSTM là một dạng đặc biệt của RNN, với cấu trúc gồm các cổng có khả năng ghi nhớ và học được các thông tin và ràng buộc trong khoảng thời gian dài mà không cần bất kỳ sự can thiệp nào. Các mạng hồi quy đều có dạng là một chuỗi các mô-đun lặp đi lặp lại của mạng nơ-ron. Đối với mạng RNN chuẩn, các mô-đun có cấu trúc rất đơn giản, thường là một tầng tanh. Mặc dù LSTM cũng có kiến trúc dạng chuỗi như mạng RNN chuẩn, nhưng các mô- đun bên trong sẽ có cấu trúc phức tạp hơn. Thay vì chỉ có một tầng mạng nơ-ron, chúng có tới 4 tầng tương tác với nhau một cách rất đặc biệt.



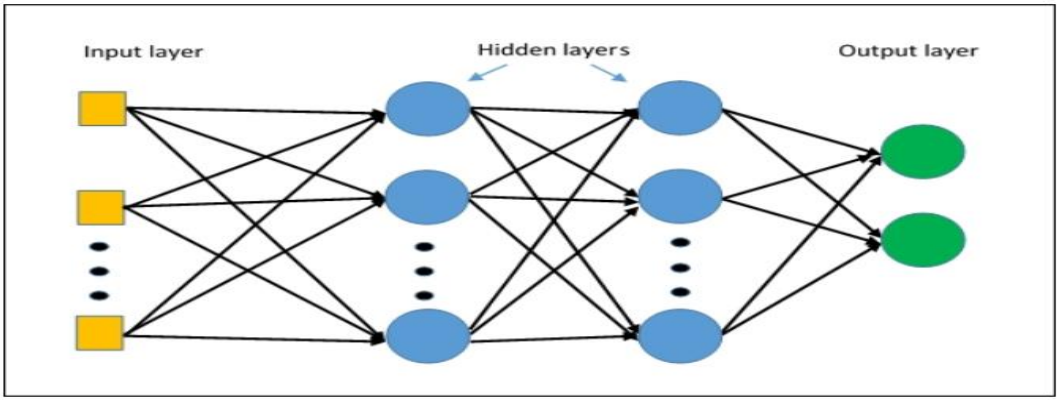
Hình 0-2 Mô hình LSTM

Mô hình GRU (Gated Recurrent Unit): GRU là phiên bản cải thiện của mạng RNN truyền thống. Để giải quyết vấn đề mất mát gradient của mạng RNN truyền thống, GRU đã được sử dụng và gọi là cổng cập nhật và cổng cài đặt lại (update gate và reset gate). Về cơ bản, đó chính là hai vector quyết định thông tin nào sẽ được truyền cho đầu ra. Điều đặc biệt là nó có thể được đào tạo để giữ thông tin từ lâu trước đó, không hề xóa thông tin không liên quan đến dự đoán đầu ra.



Hình 0-3 Mô hình GRU

Mô hình MLP (Multilayer Perceptron): là một mô hình mạng nơ-ron được kết nối đầy đủ tiêu chuẩn. Nó bao gồm các layer của node trong đó mỗi node được kết nối với tất cả các đầu ra từ layer trước và đầu ra của mỗi node được kết nối với tất cả các đầu vào cho các node ở layer tiếp theo.



Hình 0-4 Mô hình MLP

1.3. Ứng dụng mô hình mạng học sâu dự báo sản lượng tiêu thụ năng lượng

Trong cuộc cách mạng công nghệ 4.0, trí tuệ nhân tạo (AI) đang trở thành xu hướng toàn cầu và được ứng dụng phổ biến trong mọi ngành nghề, lĩnh vực, tổ chức, doanh nghiệp. Tại Việt Nam trong những năm gần đây, nhu cầu tiêu thụ năng lượng ngày một tăng, để đảm bảo an ninh năng lượng và phát triển kinh tế bền vững, các giải pháp sử dụng trí tuệ nhân tạo trong phân tích dự báo sản lượng tiêu thụ năng lượng thu hút sự quan tâm của các cấp chính quyền giúp cơ quan chiến lược có thể xác định hương phát triển đúng đắn trong tương lai. Do đó ngày càng có nhiều nghiên cứu về các giải pháp phân tích dự đoán hỗ trợ bởi AI được triển khai và áp dụng trong dự báo sản lượng tiêu thụ năng lượng.

2. Tính cấp thiết của vấn đề dự báo nhu cầu tiêu thụ năng lượng

2.1. An ninh năng lượng là gì?

An ninh năng lượng là đảm bảo cung cấp năng lượng đầy đủ để duy trì sản xuất, phát triển kinh tế quốc gia và đáp ứng nhu cầu tiêu dùng của người dân ở mức ổn định bình thường.

2.2. An ninh năng lượng quốc gia là tiền đề quan trọng thực hiện công nghiệp hóa, hiện đại hóa

Đứng trước yêu cầu của công nghiệp hóa, hiện đại hóa đất nước, nhu cầu sử dụng năng lượng của Việt Nam không ngừng gia tăng, trong khi nguồn cung năng lượng ngày càng cạn kiệt. Do vậy, chúng ta cần có lộ trình cụ thể trong xây dựng mô hình năng lượng sạch trong tương lai.

Ngày 11-2-2020, Tổng Bí thư, Chủ tịch nước Nguyễn Phú Trọng thay mặt Bộ Chính trị ký ban hành Nghị quyết số 55-NQ/TW, *"Về định hướng Chiến lược phát triển năng lượng quốc gia của Việt Nam đến năm 2030, tầm nhìn đến năm 2045"*. Nghị quyết khẳng định, phát triển năng lượng gắn với thực thi chính sách bảo vệ môi trường, nhằm đạt mục tiêu giảm phát thải khí nhà kính, thúc đẩy kinh tế tuần hoàn và phát triển bền vững.

Việt Nam cần phải phát triển ngành năng lượng gắn liền với phát triển kinh tế, để các ngành kinh tế nhận thức được giới hạn của ngành năng lượng, từ đó xây dựng

chiến lược, quy hoạch phát triển phù hợp. Khi toàn bộ nền kinh tế không còn sức ép lên ngành năng lượng, mới có khả năng bảo đảm được an ninh năng lượng quốc gia.

2.3. Nhu cầu tiêu thụ năng lượng toàn cầu

Quốc gia/Châu lục	2020	2021	Tốc độ tăng trưởng		Tỉ trọng năm 2021
			2021	2011-2021	
Bắc Mỹ	*108,79*	*113,70*	*4,8%*	*-0,1%*	*19,1%*
Nam & Trung Mỹ	*26,66*	*28,46*	*7,0%*	*0,3%*	*4,8%*
Châu Âu	*78,93*	*82,38*	*4,7%*	*-0,6%*	*13,8%*
CIS	*37,46*	*40,32*	*7,9%*	*0,9%*	*6,8%*
Các nước Trung Đông	*36,60*	*37,84*	*3,7%*	*2,2%*	*6,4%*
Châu Phi	*18,89*	*19,99*	*6,2%*	*2,1%*	*3,4%*
Châu Á-TBD	*256,69*	*272,45*	*6,4%*	*2,8%*	*45,8%*
Toàn Thế giới	564,01	595,15	5,8%	1,3%	100,0%
- OECD	220,20	229,89	4,7%	-0,2%	38,6%
- Ngoài OECD	343,82	365,26	6,5%	2,4%	61,4%
- EU	57,07	60,11	5,6%	-0,6%	10,1%

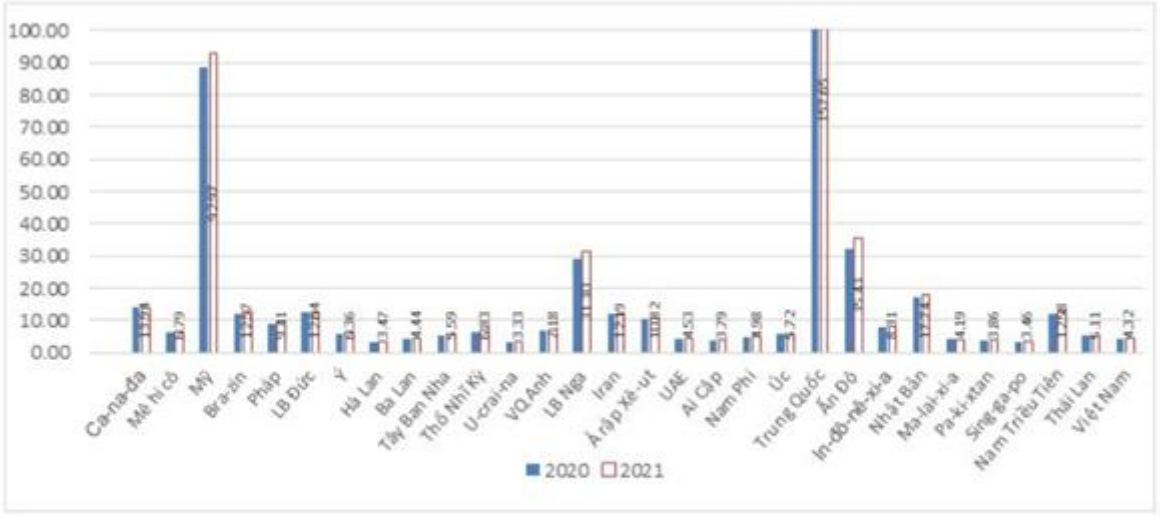
Nguồn: BP Statistical Review of World Energy 2022 | 71st edition.

Bảng 1 Tổng tiêu thụ NLSC của thế giới và các khu vực năm 2020 - 2021. Đơn vị tính: EJ.

Ghi chú: Tốc độ tăng trưởng được điều chỉnh cho các năm nhuận. EJ = Exajoules =1 x 1018 joules.

Trong báo cáo này, năng lượng sơ cấp (bao gồm các loại nhiên liệu được giao dịch thương mại và năng lượng tái tạo hiện đại).

Năng lượng từ tất cả các nguồn sản xuất điện phi hóa thạch được tính trên cơ sở đầu vào tương đương.



Ghi chú: *30 nước đại diện có mức NLSC năm 2021 bằng hoặc lớn hơn của Singapo (3,44 EJ).*

Hình 0-5 Tổng tiêu thụ NLSC (EJ) của các nước đại diện năm 2020-2021

Tổng NLSC tiêu thụ toàn cầu năm 2021 tăng 5,8% so với năm 2020 và trong giai đoạn 2011 - 2021 tăng bình quân 1,3%/năm.

Tuy nhiên, tiêu thụ NLSC của từng châu lục, khu vực có khác biệt với bức tranh chung của thế giới. Cụ thể là:

Bắc Mỹ: Năm 2021 tăng so với năm 2020 là 4,8%, nhưng giảm nhẹ 0,56% so với năm 2011, cả giai đoạn 2011 - 2021 giảm bình quân 0,1%/năm. Năm 2021 chiếm tỷ trọng 19,1% tiêu thụ NLSC toàn cầu, giảm so với năm 2011 (chiếm 21,95%).

Nam và Trung Mỹ: Năm 2021 tăng so với năm 2020 là 7,0% và so với năm 2011 tăng 2,67%. Cả giai đoạn 2011 - 2021 tăng bình quân 0,3%/năm. Năm 2021 chiếm tỷ trọng 4,8% tiêu thụ NLSC toàn cầu, giảm so với năm 2011 (chiếm 5,33%).

Châu Âu: Năm 2021 tăng so với năm 2020 là 4,7% và giảm 5,87% so với năm 2011, cả giai đoạn 2011 - 2021 giảm bình quân 0,6%/năm. Năm 2021 chiếm tỷ trọng 13,8% tiêu thụ NLSC toàn cầu, giảm so với năm 2011 (chiếm 16,8%).

Khối CIS: Năm 2021 tăng so với năm 2020 là 7,9% và so với năm 2011 tăng tương đối cao, tới 9,12%. Cả giai đoạn 2011 - 2021 tăng bình quân 0,9%/năm. Năm

2021 chiếm tỷ trọng 6,8% tiêu thụ NLSC toàn cầu, giảm so với năm 2011 (chiếm 7,1%), nguyên nhân chủ yếu là do tỷ trọng của một số khu vực tăng cao hơn.

Các nước Trung Đông: Năm 2021 tăng so với năm 2020 là 3,7% và so với năm 2011 tăng cao, tới 23,95%. Cả giai đoạn 2011 - 2021 tăng bình quân 2,2%/năm. Năm 2021 chiếm tỷ trọng 6,4% tiêu thụ NLSC toàn cầu, tăng so với năm 2011 (chiếm 5,87%).

Châu Phi: Năm 2021 tăng so với năm 2020 là 6,2% và so với năm 2011 tăng cao, tới 23,32%. Cả giai đoạn 2011 - 2021 tăng bình quân 2,1%/năm. Năm 2021 chiếm tỷ trọng 3,4% tiêu thụ NLSC toàn cầu, tăng so với năm 2011 (chiếm 3,12%).

Châu Á - Thái Bình Dương: Năm 2021 tăng so với năm 2020 là 6,4% và so với năm 2011 tăng cao, tới 31,20%. Cả giai đoạn 2011 - 2021 tăng bình quân 2,8%/năm, là châu lục có mức tăng cao nhất trong năm 2021 và cả giai đoạn 2011 - 2021. Năm 2021 chiếm tỷ trọng 45,8% tiêu thụ NLSC toàn cầu, tăng cao so với năm 2011 (chiếm 39,87%). Trung Quốc có tổng NLSC lớn nhất thế giới với 157,65 EJ.

Khối OECD: Năm 2021 tăng so với năm 2020 là 4,7% và so với năm 2011 giảm 1,74%. Cả giai đoạn 2011 - 2021 giảm nhẹ bình quân 0,2%/năm. Năm 2021 chiếm tỷ trọng 38,6% tiêu thụ NLSC trên toàn thế giới, giảm mạnh so với năm 2011 (chiếm 44,93%).

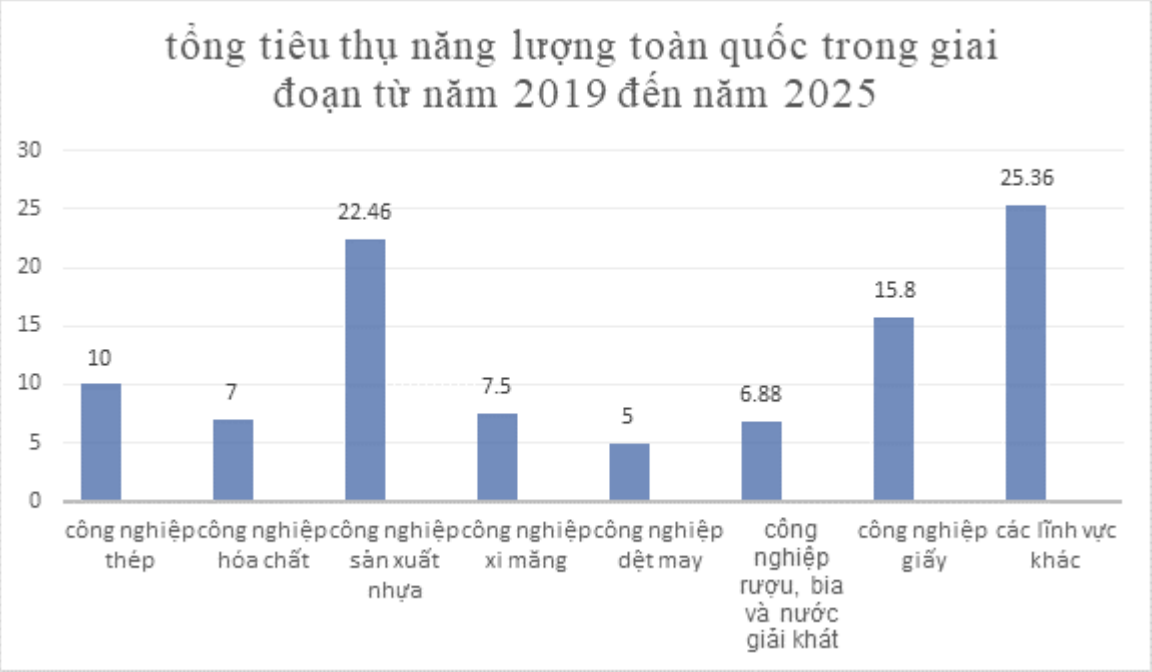
Khối ngoài OECD: Năm 2021 tăng so với năm 2020 là 6,5% và so với năm 2011 tăng cao, tới 27,30%. Cả giai đoạn 2011 - 2021 tăng bình quân 2,4%/năm. Năm 2021 chiếm tỷ trọng 61,4% tiêu thụ NLSC toàn cầu, tăng so với năm 2011 (chiếm 55,07%).

Khối EU: Năm 2021 tăng so với năm 2020 là 5,6% và so với năm 2011 giảm 5,88%. Cả giai đoạn 2011 - 2021 giảm bình quân 0,6%/năm. Năm 2021 chiếm tỷ trọng 10,1% tiêu thụ NLSC toàn cầu, giảm so với năm 2011 (chiếm 12,27%).

2.4. Nhu cầu tiêu thụ năng lượng tại Việt Nam

Năm 2021, Việt Nam tiêu thụ NLSC 4,32 EJ, tăng 2,6% so với năm 2020 và chiếm 0,7% tổng tiêu thụ NLSC toàn cầu. Trong giai đoạn từ 2011 - 2021 có tốc độ tiêu thụ NLSC tăng bình quân 7,2%/năm, vào loại cao trên thế giới.

Trong cơ cấu tiêu thụ NLSC năm 2021 của Việt Nam than chiếm 49,77%; dầu 21,76%; thủy điện 16,44%; NLTT 6,25%; khí đốt 6,02%. Như vậy, than chiếm tỷ trọng cao nhất, còn NLTT nếu cộng cả thủy điện thì có tỷ trọng chiếm 22,69%, vào loại tương đối cao trên thế giới.



Hình 0-6 Tiêu thụ năng lượng toàn quốc năm 2019-2025

Do vậy, vấn đề là đi đôi với tăng cường khai thác các nguồn tài nguyên năng lượng trong nước, nhất là năng lượng tái tạo, cần phải đẩy mạnh nhập khẩu và đầu tư khai thác tài nguyên năng lượng ở nước ngoài, nhất là than, dầu khí. Đồng thời, phải phát huy cao độ việc thực hiện sử dụng năng lượng tiết kiệm và hiệu quả, đặc biệt là sử dụng năng lượng hiệu quả theo định hướng như đã nêu trên nhằm đảm bảo đồng thời đạt 3 mục tiêu: Đảm bảo an ninh năng lượng, tăng trưởng kinh tế bền vững, bảo vệ môi trường xanh - sạch - đẹp.

CHƯƠNG 1 MÔ HÌNH HỌC SÂU TRONG DỰ BÁO PHÂN TÍCH CHUỖI THỜI GIAN

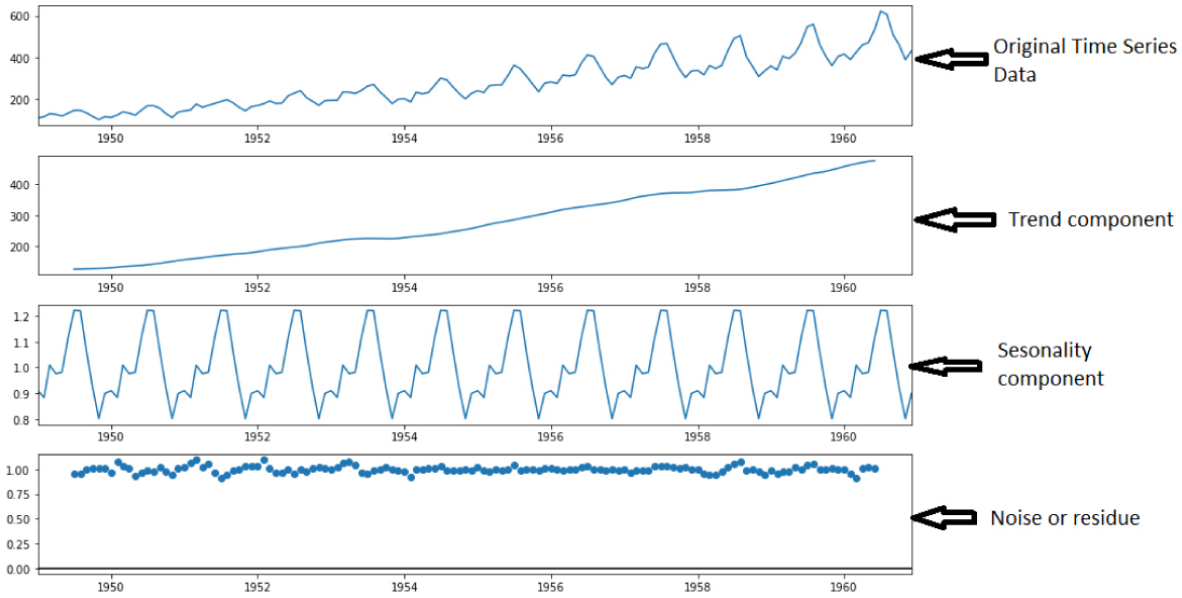
1.1 Tìm hiểu về Time Series

Time series là một loại dữ liệu thống kê thu thập theo thời gian. Nó bao gồm các giá trị được ghi lại trong các khoảng thời gian cố định hoặc không đều. Time series thường được sử dụng để nghiên cứu và phân tích các xu hướng, mô hình hóa các quá trình thời gian, và dự đoán các giá trị trong tương lai. Trong Time-series Data, có hai loại chính:

- Chuỗi thời gian thông thường (regular time series), loại thông thường được gọi là số liệu.
- Chuỗi thời gian bất thường (events) là những sự kiện.

Một dữ liệu chuỗi thời gian thường được phân rã thành 4 thành phần con sau:

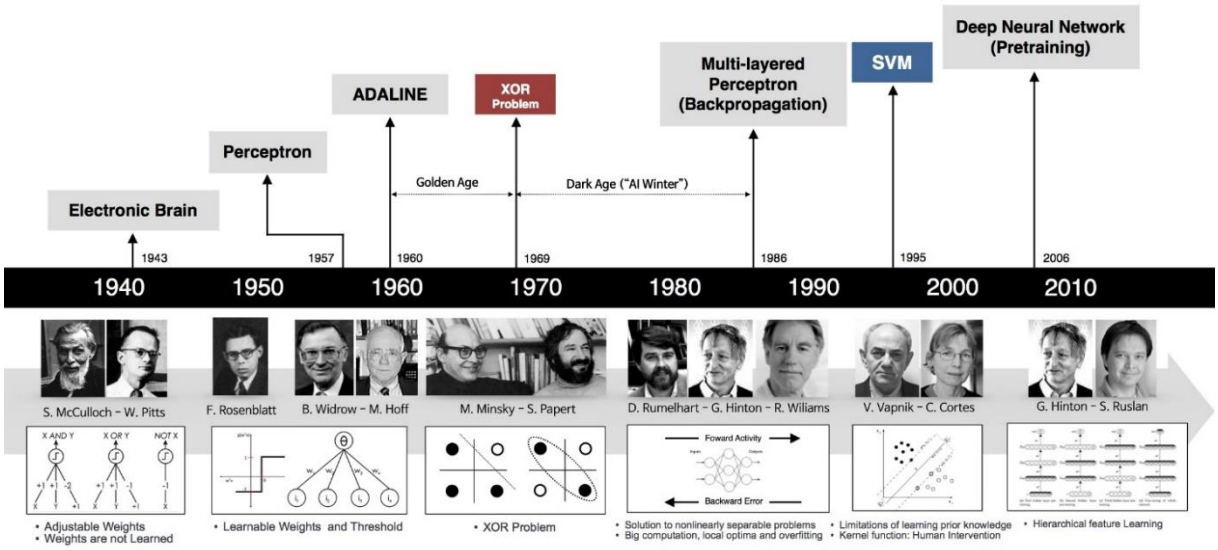
- Xu hướng (Trend): thành phần này chỉ ra xu hướng tổng quan của dữ liệu theo thời gian: lên hoặc xuống, tăng hoặc giảm.
- Mùa vụ (seasonality): thành phần chỉ ra các xu hướng theo mùa, theo tháng, theo quý.
- Chu kì (Cycle): thành phần chu kỳ, nó khác yếu tố mùa vụ ở chỗ thành phần này có sự vận động trong khoảng thời gian dài hơn (nhiều năm).
- Yếu tố bất thường (Irregular remainder): hay còn gọi là nhiễu trắng (white noise) thành phần nhiễu còn lại sau khi trích xuất hết các thành phần ở trên, nó chỉ ra sự bất thường của các điểm dữ liệu.



Hình 1-1 Các thành phần của Time Series

1.2 Tìm hiểu về Deep Learning

Các dấu mốc quan trọng của DeepLearning:



Hình 1-2 Những dấu mốc quan trọng của Deep Learning

- Perceptron (60s)

Một trong những nền móng đầu tiên của neural network và deep learning là perceptron learning algorithm (hoặc gọn là perceptron). Perceptron là một thuật toán supervised learning giúp giải quyết bài toán phân lớp nhị phân, được khởi nguồn bởi

Frank Rosenblatt năm 1957 trong một nghiên cứu được tài trợ bởi Văn phòng nghiên cứu hải quân Hoa Kỳ (U.S Office of Naval Research – từ một cơ quan liên quan đến quân sự). Thuật toán perceptron được chứng minh là hội tụ nếu hai lớp dữ liệu là linearly separable. Với thành công này, năm 1958, trong một hội thảo, Rosenblatt đã có một phát biểu gây tranh cãi. Từ phát biểu này, tờ New York Times đã có một bài báo cho rằng perceptron được Hải quân Hoa Kỳ mong đợi "có thể đi, nói chuyện, nhìn, viết, tự sinh sản, và tự nhận thức được sự tồn tại của mình". (Chúng ta biết rằng cho tới giờ các hệ thống nâng cao hơn perceptron nhiều lần vẫn chưa thể).

Mặc dù thuật toán này mang lại nhiều kỳ vọng, nó nhanh chóng được chứng minh không thể giải quyết những bài toán đơn giản. Năm 1969, Marvin Minsky và Seymour Papert trong cuốn sách nổi tiếng Perceptrons đã chứng minh rằng không thể 'học' được hàm số XOR khi sử dụng perceptron. Phát hiện này làm choáng váng giới khoa học thời gian đó (bây giờ chúng ta thấy việc này khá hiển nhiên). Perceptron được chứng minh rằng chỉ hoạt động nếu dữ liệu là linearly separable.

Phát hiện này khiến cho các nghiên cứu về perceptron bị gián đoạn gần 20 năm. Thời kỳ này còn được gọi là Mùa đông AI thứ nhất (The First AI winter).

- MLP và Backpropagation ra đời (80s)

Geoffrey Hinton tốt nghiệp PhD ngành neural networks năm 1978. Năm 1986, ông cùng với hai tác giả khác xuất bản một bài báo khoa học trên Nature với tựa đề "Learning representations by back-propagating errors". Trong bài báo này, nhóm của ông chứng minh rằng neural nets với nhiều hidden layer (được gọi là multi-layer perceptron hoặc MLP) có thể được huấn luyện một cách hiệu quả dựa trên một quy trình đơn giản được gọi là backpropagation (backpropagation là tên gọi mỹ miều của quy tắc chuỗi – chain rule – trong tính đạo hàm. Việc tính được đạo hàm của hàm số phức tạp mô tả quan hệ giữa đầu vào và đầu ra của một neural net là rất quan trọng vì hầu hết các thuật toán tối ưu đều được thực hiện thông qua việc tính đạo hàm, gradient descent là một ví dụ). Việc này giúp neural nets thoát được những hạn chế của perceptron về việc chỉ biểu diễn được các quan hệ tuyến tính. Để biểu diễn các quan hệ phi tuyến, phía sau mỗi layer là một hàm kích hoạt phi tuyến, ví dụ hàm sigmoid hoặc

tanh. (ReLU ra đời năm 2012). Với hidden layers, neural nets được chứng minh rằng có khả năng xấp xỉ hầu hết bất kỳ hàm số nào qua một định lý được gọi là universal approximation theorem. Neurel nets quay trở lại cuộc chơi.

Thuật toán này mang lại một vài thành công ban đầu, nổi trội là convolutional neural nets (convnets hay CNN) (còn được gọi là LeNet) cho bài toán nhận dạng chữ số viết tay được khởi nguồn bởi Yann LeCun tại AT&T Bell Labs (Yann LeCun là sinh viên sau cao học của Hinton tại đại học Toronto năm 1987-1988). Dưới đây là bản demo được lấy từ trang web của LeNet, network là một CNN với 5 layer, còn được gọi là LeNet-5 (1998).

Mô hình này được sử dụng rộng rãi trong các hệ thống đọc số viết tay trên các check (séc ngân hàng) và mã vùng bưu điện của nước Mỹ.

LeNet là thuật toán tốt nhất thời gian đó cho bài toán nhận dạng ảnh chữ số viết tay. Nó tốt hơn MLP thông thường (với fully connected layer) vì nó có khả năng trích xuất được đặc trưng trong không gian hai chiều của ảnh thông qua các filters (bộ lọc) hai chiều. Hơn nữa, các filter này nhỏ nên việc lưu trữ và tính toán cũng tốt hơn so với MLP thông thường. (Yan LeCun có xuất phát từ Electrical Engineering nên rất quen thuộc với các bộ lọc).

- Mùa đông AI thứ hai (90s - đầu 2000s)

Các mô hình tương tự được kỳ vọng sẽ giải quyết nhiều bài toán image classification khác. Tuy nhiên, không như các chữ số, các loại ảnh khác lại rất hạn chế vì máy ảnh số chưa phổ biến tại thời điểm đó. Ảnh được gán nhãn lại càng hiếm. Trong khi để có thể huấn luyện được mô hình convnets, ta cần rất nhiều dữ liệu huấn luyện. Ngay cả khi dữ liệu có đủ, một vấn đề nan giải khác là khả năng tính toán của các máy tính thời đó còn rất hạn chế.

Một hạn chế khác của các kiến trúc MLP nói chung là hàm mất mát không phải là một hàm lồi. Việc này khiến cho việc tìm nghiệm tối ưu toàn cục cho bài toán tối ưu hàm mất mát trở nên rất khó khăn. Một vấn đề khác liên quan đến giới hạn tính toán của máy tính cũng khiến cho việc huấn luyện MLP không hiệu quả khi số lượng hidden layers lớn lên. Vấn đề này có tên là vanishing gradient.

Nhắc lại rằng hàm kích hoạt được sử dụng thời gian đó là sigmoid hoặc tanh – là các hàm bị chặn trong khoảng (0, 1) hoặc (-1, 1) (Nhắc lại đạo hàm của hàm sigmoid là tích của hai số nhỏ hơn 1). Khi sử dụng backpropagation để tính đạo hàm cho các ma trận hệ số ở các lớp đầu tiên, ta cần phải nhân rất nhiều các giá trị nhỏ hơn 1 với nhau. Việc này khiến cho nhiều đạo hàm thành phần bằng 0 do xấp xỉ tính toán. Khi đạo hàm của một thành phần bằng 0, nó sẽ không được cập nhật thông qua gradient descent!

Những hạn chế này khiến cho neural nets một lần nữa rơi vào thời kỳ băng giá. Vào thời điểm những năm 1990 và đầu những năm 2000, neural nets dần được thay thế bởi support vector machines –SVM. SVMs có ưu điểm là bài toán tối ưu để tìm các tham số của nó là một bài toán lồi – có nhiều các thuật toán tối ưu hiệu quả giúp tìm nghiệm của nó. Các kỹ thuật về kernel cũng phát triển giúp SVMs giải quyết được cả các vấn đề về việc dữ liệu không phân biệt tuyến tính.

Nhiều nhà khoa học làm machine learning chuyển sang nghiên cứu SVM trong thời gian đó, trừ một vài nhà khoa học cứng đầu…

- Cái tên được làm mới – Deep Learning (2006)

Năm 2006, Hinton một lần nữa cho rằng ông biết bộ não hoạt động như thế nào, và giới thiệu ý tưởng của tiền huấn luyện không giám sát (unsupervised pretraining) thông qua deep belief nets (DBN). DBN có thể được xem như sự xếp chồng các unsupervised networks đơn giản như restricted Boltzman machine hay autoencoders.

Lấy ví dụ với autoencoder. Mỗi autoencoder là một neural net với một hidden layer. Số hidden unit ít hơn số input unit, và số output unit bằng với số input unit. Network này đơn giản được huấn luyện để kết quả ở output layer giống với kết quả ở input layer (và vì vậy được gọi là autoencoder). Quá trình dữ liệu đi từ input layer tới hidden layer có thể coi là mã hoá, quá trình dữ liệu đi từ hidden layer ra output layer có thể được coi là giải mã. Khi output giống với input, ta có thể thấy rằng hidden layer với ít unit hơn có để mã hoá input khá thành công, và có thể được coi mang những tính chất của input. Nếu ta bỏ output layer, cố định (freeze) kết nối giữa input và hidden layer, coi đầu ra của hidden layer là một input mới, sau đó huấn luyện một

autoencoder khác, ta được thêm một hidden layer nữa. Quá trình này tiếp tục kéo dài ta sẽ được một network đủ sâu mà output của network lớn này (chính là hidden layer của autoencoder cuối cùng) mang thông tin của input ban đầu. Sau đó ta có thể thêm các layer khác tuỳ thuộc vào bài toán (chẳng hạn thêm softmax layer ở cuối cho bài toán classification). Cả network được huấn luyện thêm một vài epoch nữa. Quá trình này được gọi là tinh chỉnh (fine tuning).

Tại sao quá trình huấn luyện như trên mang lại nhiều lợi ích?

Một trong những hạn chế đã đề cập của MLP là vấn đề vanishing gradient. Những ma trận trọng số ứng với các layer đầu của network rất khó được huấn luyện vì đạo hàm của hàm mất mát theo các ma trận này nhỏ. Với ý tưởng của DBN, các ma trận trọng số ở những hidden layer đầu tiên được tiền huấn luyện (pretrained). Các trọng số được tiền huấn luyện này có thể coi là giá trị khởi tạo tốt cho các hidden layer phía đầu. Việc này giúp phần nào tránh được sự phiền hà của vanishing gradient.

Kể từ đây, neural networks với nhiều hidden layer được đổi tên thành deep learning.

Vấn đề vanishing gradient được giải quyết phần nào (vẫn chưa thực sự triệt để), nhưng vẫn còn những vấn đề khác của deep learning: dữ liệu huấn luyện quá ít, và khả năng tính toán của CPU còn rất hạn chế trong việc huấn luyện các deep networks.

Năm 2010, giáo sư Fei-Fei Li, một giáo sư ngành computer vision đầu ngành tại Stanford, cùng với nhóm của bà tạo ra một cơ sở dữ liệu có tên ImageNet với hàng triệu bức ảnh thuộc 1000 lớp dữ liệu khác nhau đã được gán nhãn. Dự án này được thực hiện nhờ vào sự bùng nổ của internet những năm 2000 và lượng ảnh khổng lồ được upload lên internet thời gian đó. Các bức ảnh này được gán nhãn bởi rất nhiều người (được trả công).

Bộ cơ sở dữ liệu này được cập nhật hàng năm, và kể từ năm 2010, nó được dùng trong một cuộc thi thường niên có tên ImageNet Large Scale Visual Recognition Challenge (ILSVRC). Trong cuộc thi này, dữ liệu huấn luyện được giao cho các đội tham gia. Mỗi đội cần sử dụng dữ liệu này để huấn luyện các mô hình phân lớp, các mô hình này sẽ được áp dụng để dự đoán nhãn của dữ liệu mới (được giữ bởi ban tổ

chức). Trong hai năm 2010 và 2011, có rất nhiều đội tham gia. Các mô hình trong hai năm này chủ yếu là sự kết hợp của SVM với các feature được xây dựng bởi các bộ hand-crafted descriptors (SIFT, HoG, v.v.). Mô hình giành chiến thắng có top-5 error rate là 28% (càng nhỏ càng tốt). Mô hình giành chiến thắng năm 2011 có top-5 error rate là 26%. Cải thiện không nhiều!

Ngoài lề: top-5 error rate được tính như sau. Mỗi mô hình dự đoán 5 nhãn của một bức ảnh. Nếu nhãn thật của bức ảnh nằm trong 5 nhãn đó, ta có một điểm được phân lớp chính xác. Ngoài ra, bức ảnh đó được coi là một error. Top-5 error rate là tỉ lệ số bức ảnh error trong toàn bộ số ảnh kiểm thử với error được tính theo cách này. Top-1 error cộng với classification accuracy (phần trăm) chính bằng 100 phần trăm.

- Đột phá (2012)

Năm 2012, cũng tại ILSVRC, Alex Krizhevsky, Ilya Sutskever, và Geoffrey Hinton (lại là ông) tham gia và đạt kết quả top-5 error rate 16%. Kết quả này làm sững sờ giới nghiên cứu thời gian đó. Mô hình là một Deep Convolutional Neural Network, sau này được gọi là AlexNet.

Trong bài báo này, rất nhiều các kỹ thuật mới được giới thiệu. Trong đó hai đóng góp nổi bật nhất là hàm ReLU và dropout. Hàm ReLU với cách tính và đạo hàm đơn giản (bằng 1 khi đầu vào không âm, bằng 0 khi ngược lại) giúp tốc độ huấn luyện tăng lên đáng kể. Ngoài ra, việc ReLU không bị chặn trên bởi 1 (như softmax hay tanh) khiến cho vấn đề vanishing gradient cũng được giải quyết phần nào. Dropout cũng là một kỹ thuật đơn giản và cực kỳ hiệu quả. Trong quá trình training, nhiều hidden unit bị tắt ngẫu nhiên và mô hình được huấn luyện trên các bộ tham số còn lại. Trong quá trình test, toàn bộ các unit sẽ được sử dụng. Cách làm này khá là có lý khi đối chiếu với con người. Nếu chỉ dùng một phần năng lực đã đem lại hiệu quả thì dùng toàn bộ năng lực sẽ mang lại hiệu quả cao hơn. Việc này cũng giúp cho mô hình tránh được overfitting và cũng được coi giống với kỹ thuật ensemble trong các hệ thống machine learning khác. Với mỗi cách tắt các unit, ta có một mô hình khác nhau. Với nhiều tổ hợp unit bị tắt khác nhau, ta thu được nhiều mô hình. Việc kết hợp ở cuối cùng được coi như sự kết hợp của nhiều mô hình (và vì vậy, nó giống với ensemble learning).

Một trong những yếu tố quan trọng nhất giúp AlexNet thành công là việc sử dụng GPU (card đồ hoạ) để huấn luyện mô hình. GPU được tạo ra cho game thủ, với khả năng chạy song song nhiều lõi, đã trở thành một công cụ cực kỳ phù hợp với các thuật toán deep learning, giúp tăng tốc thuật toán lên nhiều lần so với CPU.

Sau AlexNet, tất cả các mô hình giành giải cao trong các năm tiếp theo đều là các deep networks (ZFNet 2013, GoogLeNet 2014, VGG 2014, ResNet 2015). Tôi sẽ giành một bài của blog để viết về các kiến trúc quan trọng này. Xu thế chung có thể thấy là các mô hình càng ngày càng deep. Xem hình dưới đây.

Những công ty công nghệ lớn cũng để ý tới việc phát triển các phòng nghiên cứu deep learning trong thời gian này. Rất nhiều các ứng dụng công nghệ đột phá đã được áp dụng vào cuộc sống hàng ngày. Cũng kể từ năm 2012, số lượng các bài báo khoa học về deep learning tăng lên theo hàm số mũ. Các blog về deep learning cũng tăng lên từng ngày.

1.2.1 Định nghĩa

Mô hình Deep Learning là một loại mô hình máy học được xây dựng dựa trên kiến trúc mạng nơ-ron sâu (deep neural network) với nhiều tầng ẩn (hidden layers). Deep Learning giúp máy tính học và hiểu dữ liệu phức tạp thông qua việc xử lý và học đại diện (representation learning) từ dữ liệu đầu vào.

Deep Learning (học sâu) có thể được xem là một lĩnh vực con của Machine Learning (học máy) – ở đó các máy tính sẽ học và cải thiện chính nó thông qua các thuật toán. Deep Learning được xây dựng dựa trên các khái niệm phức tạp hơn rất nhiều, chủ yếu hoạt động với các mạng nơ-ron nhân tạo để bắt chước khả năng tư duy và suy nghĩ của bộ não con người.

1.2.2 Các thành phần của Deep Learning

Lớp đầu vào (Input layer): Đây là tầng nhận dữ liệu đầu vào và truyền chúng qua mạng nơ-ron. Số lượng nút (neurons) trong tầng đầu vào phụ thuộc vào kích thước và đặc điểm của dữ liệu đầu vào.

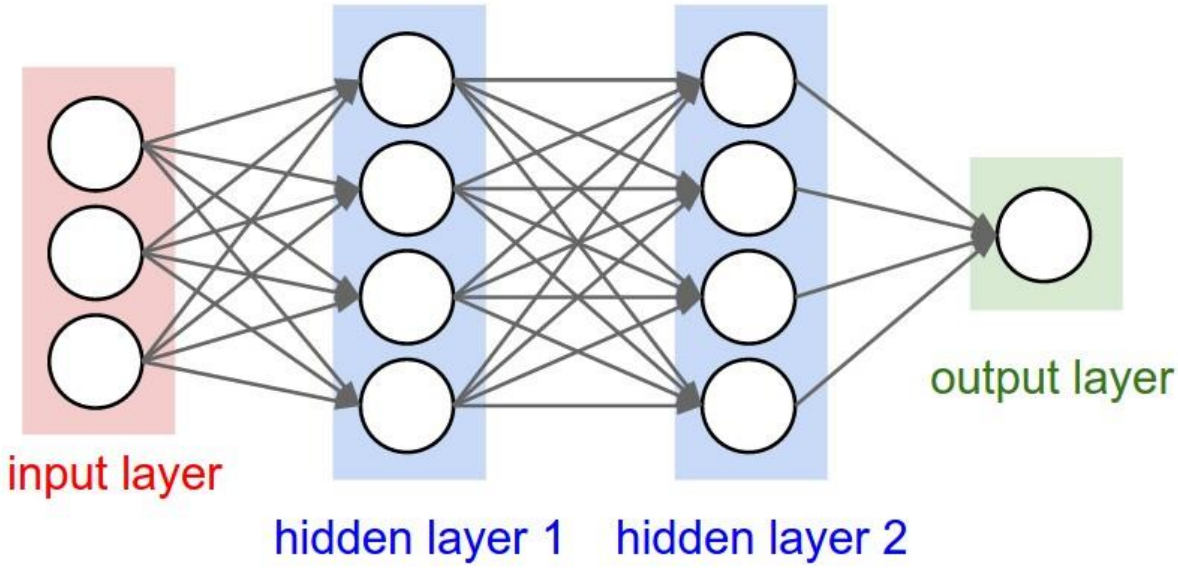
Lớp ẩn (Hidden layers): Đây là những tầng nằm giữa tầng đầu vào và tầng đầu ra. Mỗi tầng ẩn thường chứa nhiều nút nơ-ron và các kết nối trọng số (weight connections) giữa chúng. Mỗi nút trong tầng ẩn tính toán đầu ra dựa trên đầu vào từ các nút ở tầng trước đó và truyền cho tầng kế tiếp.

Hàm kích hoạt (Activation function): Hàm kích hoạt được áp dụng cho đầu ra của mỗi nút trong mạng nơ-ron để giới hạn và hiệu chỉnh giá trị đầu ra. Các hàm kích hoạt thông dụng bao gồm Sigmoid, Tanh, ReLU (Rectified Linear Unit), và Softmax (thường được sử dụng trong các tầng đầu ra của mô hình phân loại).

Lớp đầu ra (Output layer): Đây là tầng cuối cùng trong mạng nơ-ron, đưa ra đầu ra dự đoán cho bài toán cụ thể. Số lượng nút trong tầng đầu ra phụ thuộc vào loại bài toán, ví dụ: một nút cho bài toán dự đoán nhị phân (binary classification), nhiều nút cho bài toán phân loại đa lớp (multiclass classification), hoặc một nút cho bài toán dự đoán giá trị thực (regression).

Hàm mất mát (Loss function): Hàm mất mát đo lường sự sai khác giữa đầu ra dự đoán và giá trị thực tế trong quá trình huấn luyện. Nhiệm vụ của mô hình Deep Learning là tối thiểu hóa giá trị hàm mất mát này để đưa ra dự đoán chính xác.

Thuật toán tối ưu hóa: Deep Learning sử dụng các thuật toán tối ưu hóa như Gradient Descent, Stochastic Gradient Descent (SGD), hoặc các biến thể như Adam, RMSprop để điều chỉnh trọng số và cập nhật mô hình dựa trên độ lỗi (error) tính toán từ hàm mất mát.



Hình 1-3 Cấu trúc của DeepLearning

1.2.3 Cách thức hoạt động của Deep Learning

Deep Learning hoạt động bằng cách xây dựng và huấn luyện mạng nơ-ron sâu để học từ dữ liệu. Quá trình hoạt động của Deep Learning có thể được mô tả như sau:

- Chuẩn bị dữ liệu: Đầu tiên, dữ liệu được chuẩn bị và tiền xử lý để đảm bảo chất lượng dữ liệu đầu vào. Điều này có thể bao gồm các bước như chuẩn hóa, mã hóa, chia thành tập huấn luyện và tập kiểm tra, và xử lý các giá trị thiếu.
- Xây dựng mô hình Deep Learning: Mô hình Deep Learning được xác định bằng cách chọn kiến trúc mạng nơ-ron sâu, bao gồm số lượng tầng ẩn và số lượng nút trong mỗi tầng. Các lựa chọn khác bao gồm hàm kích hoạt, hàm mất mát và thuật toán tối ưu hóa.
- Khởi tạo trọng số: Trọng số của mạng nơ-ron sâu được khởi tạo ngẫu nhiên hoặc sử dụng các phương pháp khởi tạo trọng số thông minh như Xavier hoặc He.
- Lan truyền thuận (Forward propagation): Dữ liệu được đưa vào mô hình và lan truyền qua các tầng nơ-ron từ tầng đầu vào đến tầng đầu ra. Mỗi nút tính toán đầu ra dựa trên đầu vào từ các nút ở tầng trước đó và trọng số kết nối.
- Tính toán đầu ra: Sau khi đi qua các tầng nơ-ron, mô hình tính toán đầu ra dựa trên dữ liệu đầu vào và trọng số.

- Đánh giá đầu ra: Đầu ra của mô hình được so sánh với giá trị thực tế (trong trường hợp huấn luyện) hoặc giá trị mong đợi (trong trường hợp dự đoán) bằng cách sử dụng hàm mất mát. Hàm mất mát đo lường mức độ sai khác giữa đầu ra dự đoán và giá trị thực tế.
- Lan truyền ngược (Backpropagation): Quá trình lan truyền ngược tính toán đạo hàm của hàm mất mát theo trọng số. Đạo hàm này sẽ chỉ ra cách thay đổi trọng số sẽ ảnh hưởng đến giá trị mất mát.
- Cập nhật trọng số: Thuật toán tối ưu hóa được sử dụng để cập nhật trọng số mạng nơ-ron sâu dựa trên đạo hàm tính được trong quá trình lan truyền ngược. Việc cập nhật trọng số nhằm giảm thiểu giá trị mất mát và cải thiện hiệu suất của mô hình.
- Lặp lại quá trình huấn luyện: Quá trình lan truyền thuận, tính toán đầu ra, đánh giá, lan truyền ngược và cập nhật trọng số được lặp lại cho đến khi đạt được điều kiện dừng, chẳng hạn như đạt đủ số lượng epoch (vòng lặp huấn luyện) hoặc khi giá trị mất mát không còn thay đổi đáng kể.
- Đánh giá và dự đoán: Sau khi mô hình được huấn luyện, nó được sử dụng để đánh giá hiệu suất trên tập dữ liệu kiểm tra và dự đoán trên dữ liệu mới.

Quá trình này được thực hiện thông qua việc tự động học các đặc trưng và mối quan hệ phức tạp từ dữ liệu thông qua việc điều chỉnh trọng số mạng nơ-ron sâu.

1.2.4 Một vài mô hình điển hình sử dụng trong Deep Learning

- Multilayer Perceptron (MLP): MLP là một dạng cơ bản của mạng nơ-ron sâu, gồm ít nhất hai tầng ẩn giữa tầng đầu vào và tầng đầu ra. Mỗi nút trong tầng ẩn tính toán đầu ra bằng cách sử dụng hàm kích hoạt phi tuyến, như ReLU (Rectified Linear Unit) hoặc sigmoid. MLP được sử dụng phổ biến trong các bài toán phân loại và dự đoán.
- Convolutional Neural Networks (CNNs): CNNs được phát triển đặc biệt cho xử lý và phân tích dữ liệu không gian như hình ảnh và video. CNNs sử dụng các lớp tích chập (convolutional layers) để trích xuất các đặc trưng từ dữ liệu. Sau đó, các lớp Pooling được sử dụng để giảm kích thước của dữ liệu. Cuối cùng, các lớp fully connected layers được sử dụng để phân loại dữ liệu hoặc thực hiện các tác vụ khác.

- Recurrent Neural Networks (RNNs): RNNs được thiết kế để xử lý dữ liệu theo thời gian và mô hình các phụ thuộc dữ liệu dựa trên thời gian. Mỗi nút trong RNN tính toán dựa trên dữ liệu hiện tại và trạng thái ẩn trước đó. RNNs có khả năng lưu trữ thông tin trong quá khứ và sử dụng nó để ảnh hưởng đến đầu ra hiện tại. Tuy nhiên, RNNs thường gặp vấn đề mất mát thông tin dài hạn. Do đó, các biến thể như LSTM (Long Short-Term Memory) và GRU (Gated Recurrent Unit) được phát triển để khắc phục vấn đề này.
- Generative Adversarial Networks (GANs): GANs là một kiến trúc mạng nơ-ron sâu gồm hai phần: một mạng sinh (generator) và một mạng phân biệt (discriminator). Mạng sinh được huấn luyện để tạo ra các mẫu giống như dữ liệu thực, trong khi mạng phân biệt được huấn luyện để phân biệt giữa dữ liệu thật và dữ liệu tạo ra bởi mạng sinh. Hai mạng này cạnh tranh và cùng cải thiện nhau trong quá trình huấn luyện, dẫn đến việc tạo ra các mẫu chất lượng cao và chân thực.
- Transformer: Transformer là một mô hình Deep Learning được đề xuất cho các tác vụ xử lý ngôn ngữ tự nhiên (NLP). Transformer sử dụng kiến trúc không gian thời gian tự xem (self-attention) để tìm hiểu mối quan hệ giữa các từ trong câu. Mô hình này cho phép việc xử lý đồng thời các từ trong câu và giúp cải thiện hiệu suất trong việc dịch máy, tạo ra các mô hình hợp thành văn bản và nhiều tác vụ NLP khác.

Các mô hình Deep Learning này chỉ là một số ví dụ và không bao gồm tất cả các kiến trúc có sẵn. Mỗi mô hình được thiết kế để giải quyết một tác vụ cụ thể và phù hợp với loại dữ liệu và yêu cầu của bài toán.

1.2.5 Ưu điểm, nhược điểm của Deep Learning

A. Ưu điểm:

- Khả năng học từ dữ liệu phức tạp: Deep Learning có khả năng học các đặc trưng phức tạp và mô hình hóa mối quan hệ phi tuyến giữa dữ liệu. Điều này cho phép nó xử lý các bài toán phức tạp mà các phương pháp truyền thống gặp khó khăn.
- Tự động trích xuất đặc trưng: Deep Learning có khả năng tự động học và trích xuất đặc trưng từ dữ liệu, giảm thiểu sự phụ thuộc vào việc chọn và xử lý đặc trưng

thủ công. Điều này giúp tiết kiệm thời gian và công sức trong quá trình chuẩn bị dữ liệu.

- Hiệu suất cao trên dữ liệu lớn: Deep Learning thường có hiệu suất cao trên dữ liệu lớn. Với sự gia tăng của các nguồn dữ liệu lớn như hình ảnh, video, âm thanh, Deep Learning trở thành một công cụ mạnh mẽ để xử lý và phân tích dữ liệu đó.
- Tính tổng quát hóa tốt: Một mô hình Deep Learning huấn luyện tốt có khả năng tổng quát hóa từ dữ liệu huấn luyện sang dữ liệu mới. Điều này cho phép nó dự đoán và xử lý các mẫu dữ liệu chưa được nhìn thấy trước đó.

B. Nhược điểm:

- Đòi hỏi lượng dữ liệu lớn: Deep Learning yêu cầu lượng dữ liệu lớn để đạt được hiệu suất tốt. Nếu dữ liệu huấn luyện hạn chế, có thể xảy ra hiện tượng overfitting, khi mô hình chỉ học nhớ dữ liệu huấn luyện mà không thể tổng quát hóa cho dữ liệu mới.
- Đòi hỏi khối lượng tính toán lớn: Deep Learning yêu cầu sự tính toán mạnh mẽ và tài nguyên phần cứng để huấn luyện và triển khai mô hình. Các mạng nơ-ron sâu có số lượng tham số lớn, điều này đòi hỏi phần cứng mạnh để đảm bảo hiệu suất và thời gian chạy.
- Khó khăn trong diễn giải: Mô hình Deep Learning thường rất phức tạp và có số lượng tham số lớn, điều này làm cho diễn giải kết quả và hiểu cách mô hình đưa ra dự đoán trở nên khó khăn. Một mô hình Deep Learning thường được coi là một "hộp đen" vì khó xác định cách các quyết định được đưa ra.
- Yêu cầu kỹ thuật cao và kiến thức chuyên sâu: Xây dựng và huấn luyện mô hình Deep Learning đòi hỏi kiến thức chuyên sâu về lĩnh vực này, cũng như kỹ năng lập trình và xử lý dữ liệu. Điều này làm cho việc áp dụng Deep Learning trở nên khó khăn đối với những người không có kiến thức chuyên môn.

Tóm lại, mặc dù Deep Learning có những ưu điểm nổi trội trong việc xử lý dữ liệu phức tạp, nó cũng có nhược điểm và đòi hỏi các yếu tố như lượng dữ liệu lớn, tính toán mạnh mẽ và kiến thức chuyên sâu để áp dụng hiệu quả.

1.2.6 Ứng dụng của Deep Learning

Deep Learning đã có những ứng dụng đa dạng và rộng khắp trong cuộc sống hàng ngày. Dưới đây là một số ví dụ về ứng dụng của Deep Learning:

- Nhận dạng hình ảnh: Deep Learning đã đạt được thành tựu lớn trong việc nhận dạng và phân loại hình ảnh. Nó được sử dụng trong các ứng dụng như nhận dạng khuôn mặt, nhận dạng đối tượng, phân loại ảnh y khoa và tự động gắn nhãn ảnh.
- Xử lý ngôn ngữ tự nhiên (NLP): Deep Learning được sử dụng trong NLP để xử lý và hiểu ngôn ngữ tự nhiên. Các ứng dụng bao gồm dịch máy, phân loại văn bản, phân tích cảm xúc, trả lời câu hỏi và tạo văn bản tự động.
- Tự lái xe: Deep Learning đóng vai trò quan trọng trong công nghệ tự lái xe. Nó được sử dụng để nhận dạng biển báo giao thông, phát hiện và phân loại đối tượng trên đường, dự đoán hành vi của các phương tiện giao thông và quản lý hệ thống lái xe tự động.
- Nhận dạng giọng nói: Deep Learning được sử dụng trong các ứng dụng nhận dạng giọng nói như trợ lý ảo, điều khiển giọng nói và giao tiếp ngôn ngữ tự nhiên.
- Dự đoán chuỗi thời gian: Deep Learning được áp dụng trong việc dự đoán chuỗi thời gian như dự báo thời tiết, dự đoán giá cổ phiếu, dự đoán lưu lượng truy cập trang web và dự đoán nhu cầu hàng hóa.
- Tạo nội dung sáng tạo: Deep Learning có thể được sử dụng để tạo ra nội dung sáng tạo như hình ảnh nghệ thuật, âm nhạc, video và văn bản.
- Y tế và chăm sóc sức khỏe: Deep Learning được sử dụng trong lĩnh vực y tế để hỗ trợ chẩn đoán bệnh, dự đoán kết quả điều trị, phân tích hình ảnh y khoa và tiến hành phân loại bệnh.
- Tự động hóa công việc: Deep Learning có thể được sử dụng để tự động hóa các công việc và quy trình trong các lĩnh vực như sản xuất, logictics và dịch vụ khách hàng.

Những ứng dụng này chỉ là một số ví dụ, và Deep Learning đang được áp dụng rộng rãi trong nhiều lĩnh vực khác nhau, đóng góp vào việc cải thiện cuộc sống hàng ngày và mang lại những tiện ích đáng kể.

1.3 Tìm hiểu về công nghệ sử dụng trong dự án

1.3.1 Ngôn ngữ Python

Python là một ngôn ngữ lập trình thông dịch, được tạo ra vào những năm 1990 bởi Guido van Rossum. Nó là một ngôn ngữ lập trình rất phổ biến và được sử dụng rộng rãi trong nhiều lĩnh vực khác nhau, bao gồm phát triển web, trí tuệ nhân tạo, khoa học dữ liệu, tự động hóa, và nhiều ứng dụng khác.

- Cú pháp đơn giản: Python có cú pháp dễ đọc và dễ hiểu. Các khối mã được định dạng bằng các thụt lề (indentation) thay vì sử dụng dấu ngoặc nhọn như nhiều ngôn ngữ khác.
- Đa mục đích: Python là một ngôn ngữ đa mục đích, có thể được sử dụng cho các mục đích khác nhau như phát triển ứng dụng desktop, web, phân tích dữ liệu, máy học, và nhiều hơn nữa.
- Thư viện và Framework phong phú: Python có một cộng đồng lớn và phong phú, cung cấp nhiều thư viện và framework mạnh mẽ. Ví dụ: NumPy, Pandas, TensorFlow, Django, Flask, và nhiều thư viện khác.
- Dynamic typing: Python là ngôn ngữ kiểu động, điều này có nghĩa là bạn không cần khai báo kiểu dữ liệu trước khi sử dụng biến. Biến có thể được gán bất kỳ giá trị nào và có thể thay đổi kiểu dữ liệu trong quá trình thực thi.
- Hỗ trợ đối tượng: Python hỗ trợ lập trình hướng đối tượng, cho phép bạn tạo ra các lớp, đối tượng và kế thừa để tổ chức và tái sử dụng mã.

Thư viện mà Python thường hỗ trợ trong mô hình dự báo:

- scikit-learn: Scikit-learn là một thư viện học máy phổ biến trong Python. Mặc dù không phải là một thư viện chuyên về dự báo chuỗi thời gian, nhưng scikit-learn cung cấp một số mô hình học máy như hồi quy tuyến tính, hồi quy Ridge và hồi quy Lasso có thể được áp dụng cho dự báo chuỗi thời gian.
- TensorFlow và Keras: TensorFlow là một thư viện học sâu (deep learning) phổ biến và Keras là một giao diện trên nền TensorFlow. Với TensorFlow và Keras, bạn có thể xây dựng và huấn luyện mô hình mạng nơ-ron (neural network) để dự báo chuỗi thời gian.

1.3.2 Jupyter Notebook

Jupyter Notebook là một ứng dụng web mã nguồn mở cho phép bạn tạo và chia sẻ tài liệu tương tác, chứa mã Python và các phần mô tả, hình ảnh, công thức toán học và visualizations. Nó cho phép bạn tạo ra các tệp notebook có chứa mã nguồn Python được thực thi từng phần, giúp bạn khám phá dữ liệu, thực hiện các thí nghiệm, viết và chạy mã nguồn Python một cách tương tác.

Một số tính năng quan trọng của Jupyter Notebook bao gồm:

- Mã tương tác: Jupyter Notebook cho phép bạn viết và chạy mã Python từng ô (cell) một. Bạn có thể thực hiện tính toán, xử lý dữ liệu và hiển thị kết quả ngay tại chỗ.
- Hiển thị kết quả và visualizations: Khi bạn chạy một ô chứa mã Python trong Jupyter Notebook, nó sẽ hiển thị kết quả trực tiếp dưới ô đó. Bạn cũng có thể tạo ra các biểu đồ và trực quan hóa dữ liệu trong cùng tệp notebook.
- Rich text formatting: Bạn có thể sử dụng Markdown để định dạng văn bản, tạo tiêu đề, danh sách, bảng, công thức toán học và nhiều nội dung khác trong các ô tương tác.
- Tích hợp với các công cụ và thư viện: Jupyter Notebook tích hợp với nhiều công cụ và thư viện phổ biến trong cộng đồng Python như NumPy, Pandas, Matplotlib và SciPy, giúp bạn làm việc với dữ liệu và thực hiện các phân tích phức tạp.
- Chia sẻ và xuất bản: Bạn có thể chia sẻ tệp notebook của mình dưới dạng file .ipynb hoặc chuyển đổi chúng thành các định dạng khác như HTML, PDF hoặc slide để chia sẻ với người khác.

Jupyter Notebook rất phổ biến trong cộng đồng khoa học dữ liệu, học máy và nhiều lĩnh vực khác của Python, vì nó cung cấp môi trường tương tác và linh hoạt để thực hiện và chia sẻ các dự án và phân tích.

Jupyter Notebook không phải là một thư viện hay công cụ đặc thù cho dự báo chuỗi thời gian, nhưng nó là một môi trường lập trình linh hoạt và mạnh mẽ, cho phép

bạn sử dụng các thư viện dự báo chuỗi thời gian phổ biến như Statsmodels, Prophet, scikit-learn, TensorFlow, và PyCaret.

Với Jupyter Notebook, bạn có thể thực hiện các bước dự báo chuỗi thời gian, bao gồm:

- Khám phá và xử lý dữ liệu: Sử dụng các thư viện như Pandas và NumPy, bạn có thể đọc và xử lý dữ liệu chuỗi thời gian, tạo các đặc trưng (features), xử lý các giá trị thiếu, và thực hiện các biến đổi dữ liệu khác để chuẩn bị cho quá trình dự báo.
- Xây dựng mô hình dự báo: Sử dụng các thư viện như Statsmodels, Prophet, scikit-learn, TensorFlow, và PyCaret, bạn có thể xây dựng và đào tạo các mô hình dự báo chuỗi thời gian như ARIMA, SARIMA, mạng nơ-ron, hồi quy tuyến tính, hay mô hình tổng hợp khác.
- Đánh giá và tinh chỉnh mô hình: Sử dụng các phương pháp đánh giá và đo lường hiệu suất như độ chính xác, độ lỗi (RMSE, MAE), hệ số xác định (R-squared), hay các phương pháp cross-validation, bạn có thể đánh giá và tinh chỉnh mô hình dự báo của mình để đạt hiệu suất tốt nhất.
- Trực quan hóa và báo cáo: Sử dụng thư viện như Matplotlib, Seaborn và Plotly, bạn có thể trực quan hóa kết quả dự báo, so sánh giữa dự báo và dữ liệu thực tế, và tạo báo cáo có chứa các biểu đồ và visualizations tương tác.

Jupyter Notebook cung cấp một môi trường tương tác và trực quan cho việc thực hiện dự báo chuỗi thời gian, giúp bạn khám phá, phân tích và chia sẻ kết quả dễ dàng.

1.3.3 Mô hình RNN

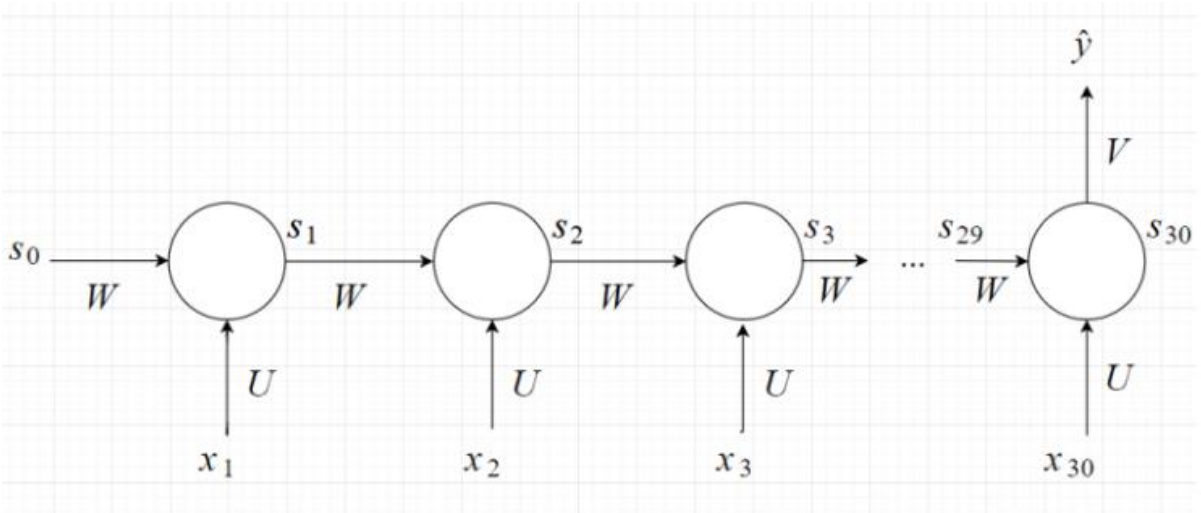
A. Định nghĩa

- Mô hình RNN (Recurrent Neural Network) là một kiểu mô hình mạng nơ-ron nhân tạo được sử dụng rộng rãi trong xử lý dữ liệu chuỗi và dữ liệu có liên quan thời gian. RNN được thiết kế để có khả năng nhớ thông tin từ các bước trước đó và áp dụng nó vào các bước tính toán hiện tại.
- Mô hình RNN có một cấu trúc kết nối đặc biệt giữa các nơ-ron, cho phép truyền ngược thông tin từ các bước trước đó sang các bước sau đó. Các nơ-ron trong

một RNN không chỉ nhận đầu vào từ bước hiện tại mà còn nhận thông tin từ trạng thái ẩn (hidden state) của bước trước. Trạng thái ẩn là một dạng bộ nhớ của mô hình, giúp nắm bắt thông tin liên quan từ quá khứ.

- Các bước tính toán trong mô hình RNN được thực hiện tuần tự qua các thời điểm (time step) của dữ liệu chuỗi. Tại mỗi thời điểm, RNN nhận đầu vào hiện tại và trạng thái ẩn từ bước trước, sau đó tính toán đầu ra và trạng thái ẩn mới. Quá trình này lặp đi lặp lại cho đến khi đạt được đầu ra mong đợi hoặc hết số bước thời gian.
- Mô hình RNN có thể được sử dụng trong nhiều tác vụ khác nhau như dự đoán chuỗi, phân loại chuỗi, dịch máy, tạo văn bản, nhận dạng giọng nói và nhiều ứng dụng khác liên quan đến dữ liệu chuỗi.
- Một trong những vấn đề của mô hình RNN là vấn đề vanishing gradient, khi gradient (đạo hàm) truyền ngược từ lớp cuối cùng đến các lớp đầu tiên bị giảm dần và gây khó khăn trong việc cập nhật trọng số. Để khắc phục vấn đề này, các phiên bản cải tiến của RNN như LSTM (Long Short-Term Memory) và GRU (Gated Recurrent Unit) đã được phát triển.

Tóm lại, mô hình RNN là một kiểu mô hình mạng nơ-ron được sử dụng để xử lý dữ liệu chuỗi và dữ liệu có liên quan thời gian. Nó có khả năng nhớ thông tin từ các bước trước đó và áp dụng nó vào các bước tính toán hiện tại. Mô hình RNN đã được áp dụng thành công trong nhiều lĩnh vực và là công cụ hữu ích trong việc xử lý dữ liệu chuỗi.



Hình 1-4 Cấu trúc mô hình RNN

B. Phân loại bài toán RNN

One to one: mẫu bài toán cho Neural Network (NN) và Convolutional Neural Network (CNN), 1 input và 1 output, ví dụ với CNN input là ảnh và output là ảnh được segment.

One to many: bài toán có 1 input nhưng nhiều output, ví dụ: bài toán caption cho ảnh, input là 1 ảnh nhưng output là nhiều chữ mô tả cho ảnh đấy, dưới dạng một câu.

Many to one: bài toán có nhiều input nhưng chỉ có 1 output, ví dụ bài toán phân loại hành động trong video, input là nhiều ảnh (frame) tách ra từ video, ouptut là hành động trong video

Many to many: bài toán có nhiều input và nhiều output, ví dụ bài toán dịch từ tiếng anh sang tiếng việt, input là 1 câu gồm nhiều chữ: “I love Vietnam” và output cũng là 1 câu gồm nhiều chữ “Tôi yêu Việt Nam”.

1.3.4 Mô hình LSTM

Mô hình LSTM (Long Short-Term Memory) là một dạng đặc biệt của mạng nơ-ron hồi quy (RNN) được thiết kế để xử lý dữ liệu chuỗi và giải quyết vấn đề vanishing gradient trong RNN.

Một LSTM có cấu trúc tương tự như RNN, nhưng nó có thêm các cơ chế đặc biệt như cổng (gates) để điều chỉnh thông tin được lưu trữ và truyền qua lại trong quá trình tính toán.

Các cổng trong LSTM bao gồm:

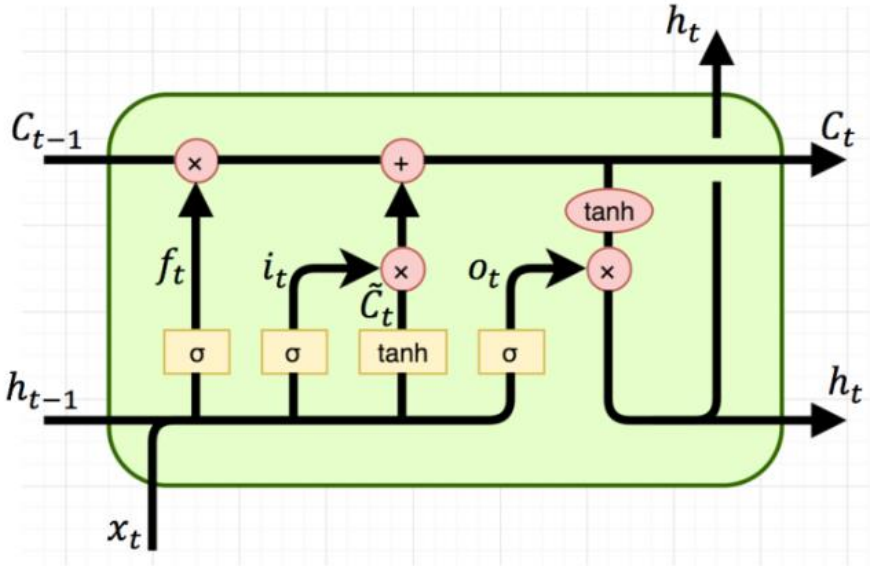
- Cổng quên (Forget gate): Xác định thông tin nào trong trạng thái trước đó cần bị lãng quên. Nó quyết định giữ lại thông tin nào và bỏ qua thông tin nào từ quá khứ.
- Cổng đầu vào (Input gate): Xác định thông tin mới nào sẽ được lưu trữ trong trạng thái tiếp theo. Nó quyết định thông tin mới nào sẽ được cập nhật từ đầu vào hiện tại.
- Cổng đầu ra (Output gate): Xác định phần nào của trạng thái hiện tại sẽ là đầu ra của LSTM. Nó quyết định thông tin nào sẽ được đưa ra từ trạng thái hiện tại.

• Trạng thái ẩn (Cell state): Là một bộ nhớ dài hạn, lưu trữ thông tin từ quá khứ và truyền thông tin tới các bước tính toán tiếp theo trong chuỗi.

• Với các cổng này, LSTM có khả năng lưu trữ thông tin quan trọng từ quá khứ và loại bỏ thông tin không quan trọng. Điều này giúp LSTM giải quyết vấn đề vanishing gradient và có khả năng xử lý được các chuỗi dài và tương quan phức tạp hơn so với RNN thông thường.

• Mô hình LSTM được sử dụng rộng rãi trong các lĩnh vực như xử lý ngôn ngữ tự nhiên, dịch máy, nhận dạng giọng nói, dự báo chuỗi thời gian và nhiều ứng dụng khác liên quan đến dữ liệu chuỗi.

Tóm lại, mô hình LSTM là một dạng đặc biệt của mạng nơ-ron hồi quy, được thiết kế để xử lý dữ liệu chuỗi và giải quyết vấn đề vanishing gradient. Nó sử dụng các cổng để điều chỉnh thông tin và trạng thái trong quá trình tính toán, giúp nó lưu trữ thông tin quan trọng và loại bỏ thông tin không quan trọng. LSTM đã được chứng minh là một công cụ mạnh mẽ trong việc xử lý và dự đoán các dữ liệu chuỗi phức tạp.



Hình 1-5 Cấu trúc mô hình LSTM

Output:ct,ht,ta gọi c là cell state,t là hidden state.

Input: ct-1,ht-1,xt.Trong đó xt là input ở state thứ t của model.

Đọc biểu đồ ở trên ta có thể thấy kí hiệu σ, tanh có nghĩa là ở bước đấy dùng hàm sigma , tanh.

Phép nhân ở đây là phép nhân từng phần tử, phép cộng là cộng ma trận.

F_t,i_t,o_t tương ứng với forget gate,input gate,output gate:

Forget gate: $f_t=\sigma(U_f* x_t+W_f* h_{t-1}+b_f)$

Input gate: $i_t=\sigma(U_i* x_t+W_i* h_{t-1}+bi)$

Output gate: $o_t= \sigma(Uo * xt + Wo * ht - 1 + bo)$

Nhận xét $0<f_t,i_t,o_t<1$;b_f,b_i,b_o là các hệ số bias,hệ số W,U giống như bài RNN

c = tanh $(Uc * xt + Wc * ht - 1 + bc)$

$c_t = ft * ct - 1 + it * ct$, forget gate quyết định xem lấy bao nhiêu từ cell state trước và input gate sẽ quyết định lấy bao nhiêu từ input state và hidden layer của layer trước.

$h_t = ot * \tanh(ct)$,output gate quyết định xem cần lấy bao nhiêu từ cell state để trở thành output của hidden state. Ngoài ra ht cũng đc dùng để tính ra output yt cho state t.

Nhận xét: ht,ct khá giống với RNN,nên model có short tẻm memory.Trong khi đó ct giống như một băng chuyền ở trên mô hình RNN, thông tin nào cần quan trọng và dùng ở sau sẽ được gửi vào dùng khi cần=>có thể mang thông tin từ đi xa.Do đó mô hình LSTM có cả short term memory và long term memory.

CHƯƠNG 2 DỮ LIỆU VÀ MA TRẬN ĐÁNH GIÁ

2.1 Dữ liệu

2.1.1 Giới thiệu

Dữ liệu được lấy từ bộ dữ liệu trên Kaggle có tên "Tetuan City Power Consumption" do người dùng GMKeshav tạo ra(1). Bộ dữ liệu này được thu thập tại thành phố Tetuan, Maroc trong khoảng thời gian năm 2017. Nó bao gồm thông tin về tiêu thụ điện của các hộ gia đình và các toà nhà khác nhau trong thành phố Tetuan. Tác giả đã thu thập dữ liệu từ công ty điện lực của thành phố Tetuan, Maroc và chia sẻ bộ dữ liệu này trên Kaggle để cho các nhà nghiên cứu và dữ liệu học có thể sử dụng và phân tích. Nó được công bố dưới giấy phép CCO, có nghĩa là bộ dữ liệu này là miễn phí và có thể được sử dụng cho mục đích thương mại hoặc phi thương mại mà không cần yêu cầu sự cho phép của tác giả.

Thông tin tập dữ liệu(2):

Bàn thử nghiệm để ước tính sức chứa đã được triển khai trong phòng 6m × 4,6m. Thiết lập bao gồm 7 nút cảm biến và một nút biên trong cấu hình sao với các nút cảm biến truyền dữ liệu đến biên sau mỗi 30 giây bằng cách sử dụng bộ thu phát không dây. Không có hệ thống HVAC nào được sử dụng trong khi bộ dữ liệu đang được thu thập.

Năm loại cảm biến không xâm nhập khác nhau đã được sử dụng trong thí nghiệm này: nhiệt độ, ánh sáng, âm thanh, CO2 và hồng ngoại thụ động kỹ thuật số (PIR). Các cảm biến CO2, âm thanh và PIR cần hiệu chuẩn thủ công. Đối với cảm biến CO2, hiệu chỉnh điểm 0 được thực hiện thủ công trước lần sử dụng đầu tiên bằng cách giữ nó trong môi trường sạch trong hơn 20 phút, sau đó kéo chân hiệu chuẩn (chân HD) xuống mức thấp trong hơn 7 giây. Cảm biến âm thanh về cơ bản là một micrô có bộ khuếch đại tương tự có mức tăng thay đổi được gắn vào nó. Do đó, đầu ra của cảm biến này là tín hiệu tương tự được đọc bởi ADC của vi điều khiển tính bằng vôn. Chiết áp gắn với mức tăng của bộ khuếch đại đã được điều chỉnh để đảm bảo độ nhạy cao

nhất. Cảm biến PIR có hai nút điều chỉnh: một để điều chỉnh độ nhạy và một để điều chỉnh thời gian để đầu ra duy trì ở mức cao sau khi phát hiện chuyển động. Cả hai điều này đã được điều chỉnh đến các giá trị cao nhất. Các nút cảm biến S1-S4 bao gồm cảm biến nhiệt độ, ánh sáng và âm thanh, S5 có cảm biến CO2 và S6 và S7 có một cảm biến PIR, mỗi nút được triển khai trên các gờ trần ở một góc giúp tối đa hóa trường nhìn của cảm biến cho chuyển động phát hiện.

Dữ liệu được thu thập trong khoảng thời gian 4 ngày một cách có kiểm soát với sức chứa trong phòng thay đổi từ 0 đến 3 người. Sự thật cơ bản về số lượng người trong phòng được ghi lại bằng tay.

Thông tin thuộc tính:

Ngày: YYYY/MM/DD
Thời gian: HH:MM:SS
Nhiệt độ: Bằng độ C

Ánh sáng: Bằng Lux
Âm thanh: Bằng Vôn (đầu ra bộ khuếch đại được đọc bởi ADC)
CO2: Bằng PPM
Độ dốc CO2: Độ dốc của các giá trị CO2 được lấy trong một cửa sổ trượt
PIR: Giá trị nhị phân truyền phát hiện chuyển động
Room_Occupancy_Count: Ground Truth

2.1.2 Phân tích

Sử dụng thư viện Pandas để đọc dữ liệu:

```
In [15]: path = "Tetuan City power consumption.csv"
         data = pd.read_csv(path)
         data
```

Out[15]:

	DateTime	Temperature	Humidity	Wind Speed	general diffuse flows	diffuse flows	Zone 1 Power Consumption	Zone 2 Power Consumption	Zone 3 Power Consumption
0	1/1/2017 0:00	6.559	73.8	0.083	0.051	0.119	34055.69620	16128.87538	20240.96386
1	1/1/2017 0:10	6.414	74.5	0.083	0.070	0.085	29814.68354	19375.07599	20131.08434
2	1/1/2017 0:20	6.313	74.5	0.080	0.062	0.100	29128.10127	19006.68693	19668.43373
3	1/1/2017 0:30	6.121	75.0	0.083	0.091	0.096	28228.86076	18361.09422	18899.27711
4	1/1/2017 0:40	5.921	75.7	0.081	0.048	0.085	27335.69620	17872.34043	18442.40964
...	...	...	...	...	...	...	...	...	...
52411	12/30/2017 23:10	7.010	72.4	0.080	0.040	0.096	31160.45627	26857.31820	14780.31212
52412	12/30/2017 23:20	6.947	72.6	0.082	0.051	0.093	30430.41825	26124.57809	14428.81152
52413	12/30/2017 23:30	6.900	72.8	0.086	0.084	0.074	29590.87452	25277.69254	13806.48259
52414	12/30/2017 23:40	6.758	73.0	0.080	0.066	0.089	28958.17490	24692.23688	13512.60504
52415	12/30/2017 23:50	6.580	74.1	0.081	0.062	0.111	28349.80989	24055.23167	13345.49820

52416 rows × 9 columns

Hình 2-1 Thông tin dữ liệu

```
In [14]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 52416 entries, 0 to 52415
Data columns (total 9 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   DateTime                   52416 non-null  object
 1   Temperature                52416 non-null  float64
 2   Humidity                   52416 non-null  float64
 3   Wind Speed                 52416 non-null  float64
 4   general diffuse flows      52416 non-null  float64
 5   diffuse flows              52416 non-null  float64
 6   Zone 1 Power Consumption   52416 non-null  float64
 7   Zone 2  Power Consumption  52416 non-null  float64
 8   Zone 3  Power Consumption  52416 non-null  float64
dtypes: float64(8), object(1)
memory usage: 3.6+ MB
```

Hình 2-2 Các thuộc tính dữ liệu

- Tổng số dòng: 52416 dòng
- Có 9 thuộc tính:
    1. DateTime: ngày giờ
    2. Temperature: nhiệt độ
    3. Humidity: độ ẩm
    4. Wind Speed: tốc độ gió
    5. General diffuse flows: dòng khuếch tán chung
    6. Diffuse flows: dòng khuếch tán
    7. Zone 1 Power Consumption: tiêu thụ điện vùng 1

8. Zone 2 Power Consumption: tiêu thụ điện vùng 2
9. Zone 3 Power Consumption: tiêu thụ điện vùng 3

- Kích thước: 3.6 MB

Mô tả dữ liệu:

```
In [17]: data = pd.read_csv(path, index_col = 'DateTime')
In [18]: data.head()
Out[18]:
```

DateTime	Temperature	Humidity	Wind Speed	general diffuse flows	diffuse flows	Zone 1 Power Consumption	Zone 2 Power Consumption	Zone 3 Power Consumption
1/1/2017 0:00	6.559	73.8	0.083	0.051	0.119	34055.69620	16128.87538	20240.96386
1/1/2017 0:10	6.414	74.5	0.083	0.070	0.085	29814.68354	19375.07599	20131.08434
1/1/2017 0:20	6.313	74.5	0.080	0.062	0.100	29128.10127	19006.68693	19668.43373
1/1/2017 0:30	6.121	75.0	0.083	0.091	0.096	28228.86076	18361.09422	18899.27711
1/1/2017 0:40	5.921	75.7	0.081	0.048	0.085	27335.69620	17872.34043	18442.40964

Hình 2-3 Một số dữ liệu đầu tiên

```
In [19]: data.describe() #mô tả dữ liệu
Out[19]:
```

	Temperature	Humidity	Wind Speed	general diffuse flows	diffuse flows	Zone 1 Power Consumption	Zone 2 Power Consumption	Zone 3 Power Consumption
count	52416.000000	52416.000000	52416.000000	52416.000000	52416.000000	52416.000000	52416.000000	52416.000000
mean	18.810024	68.259518	1.959489	182.696614	75.028022	32344.970564	21042.509082	17835.406218
std	5.815476	15.551177	2.348862	264.400960	124.210949	7130.562564	5201.465892	6622.165099
min	3.247000	11.340000	0.050000	0.004000	0.011000	13895.696200	8560.081466	5935.174070
25%	14.410000	58.310000	0.078000	0.062000	0.122000	26310.668692	16980.766032	13129.326630
50%	18.780000	69.860000	0.086000	5.035500	4.456000	32265.920340	20823.168405	16415.117470
75%	22.890000	81.400000	4.915000	319.600000	101.000000	37309.018185	24713.717520	21624.100420
max	40.010000	94.800000	6.483000	1163.000000	936.000000	52204.395120	37408.860760	47598.326360

Hình 2-4 Dữ liệu được mô tả

Xử lý dữ liệu:

Kiểm tra có giá trị nào bị thiếu không.

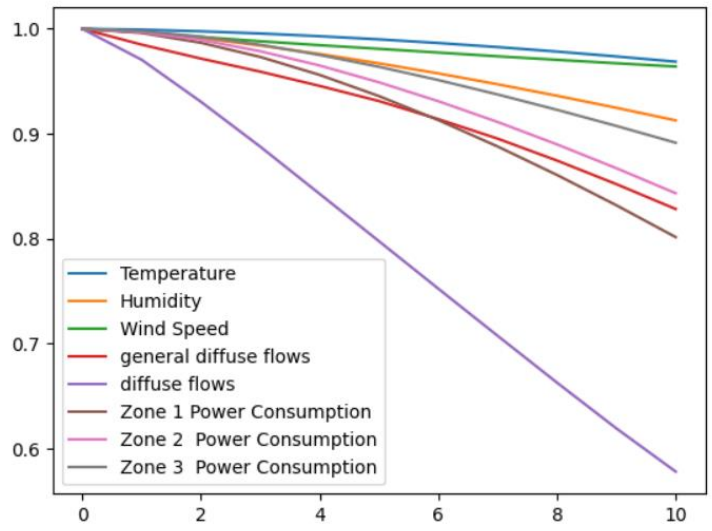
```
In [5]: ## finding all columns that have nan:
        data.columns[data.isnull().any()]
Out[5]: Index([], dtype='object')
```

Hình 2-5 Giá trị thiếu trong dữ liệu

Kiểm tra tính tuyến tính và biến thiên của dữ liệu. Phân tích nó để xem xu hướng và tính thời vụ. Cung cấp quy trình làm việc chung cho dự báo.

Dưới đây, chúng ta có một biểu đồ về cách tự tương quan của các tín hiệu khác nhau thay đổi theo độ trễ để mô hình hóa tốt hơn.

```
In [6]: #To fill in missing values, first check for linearity in each column using autocorelation plots
        def get_acf(data,lags):
            frame = []
            for i in range(lags+1):
                frame.append(data.apply(lambda col: col.autocorr(i), axis=0))
            return pd.DataFrame(frame).plot.line()
        get_acf(data,10)
Out[6]: <Axes: >
```

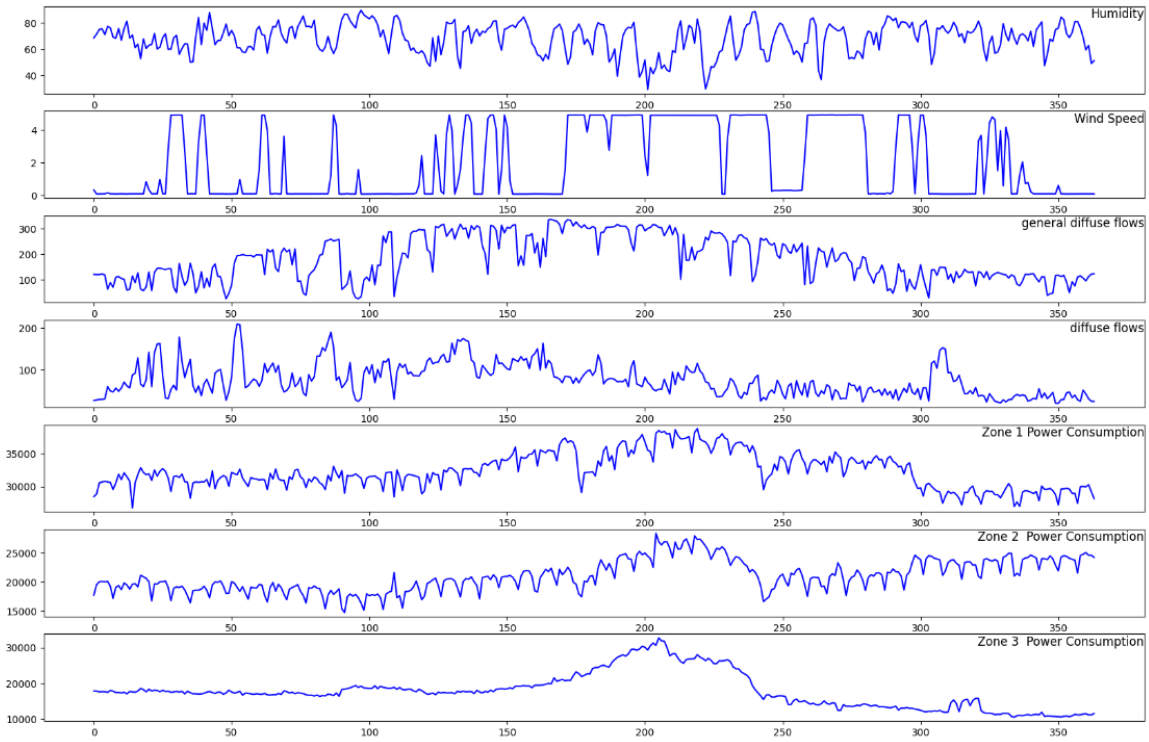


Hình 2-6 Biểu đồ tự tương quan

Chúng ta có thể thực hiện các sơ đồ lõi của các tập hợp khác nhau để xem dữ liệu thay đổi như thế nào. Lập biểu đồ nhanh mỗi ngày.

```
In [8]: #Its imperative from autocorealtion plots that we dont need each minute data, perhaps we can decide roll up the aggregate
        i = 1
        # plot each column
        plt.figure(figsize=(20, 15))
        for counter in range(1,len(clean_data.columns)):
            plt.subplot(len(clean_data.columns), 1, i)
            plt.plot(clean_data.resample('D').mean().values[:, counter], color = 'blue')
            plt.title(clean_data.columns[counter], y=0.8, loc='right')
            i = i+1
        plt.show()
```

Hình 2-7 Lập biểu đồ nhanh mỗi ngày



Hình 2-8 Biểu đồ nhanh mỗi ngày

2.1.3 Chia tệp dữ liệu

Chúng ta có thể chia bộ dữ liệu thành 2 tệp con với tỉ lệ 80:20, tức là 80% dữ liệu được sử dụng để huấn luyện mô hình và 20% dữ liệu được sử dụng để đánh giá hiệu suất của mô hình.

- Tệp huấn luyện (training set) gồm 80% dữ liệu.
- Tệp đánh giá (test set) gồm 20% dữ liệu.

```
# Import the dataset and encode the date
df = pd.read_csv('Tetuan City power consumption.csv')
group = df.groupby('DateTime')
Real_Price = group['Zone 1 Power Consumption'].mean()
print(Real_Price)
```

```
DateTime
1/1/2017 0:00     34055.69620
1/1/2017 0:10     29814.68354
1/1/2017 0:20     29128.10127
1/1/2017 0:30     28228.86076
1/1/2017 0:40     27335.69620
                     ...
9/9/2017 9:10     30628.67257
9/9/2017 9:20     31291.32743
9/9/2017 9:30     31750.08850
9/9/2017 9:40     32419.11504
9/9/2017 9:50     32724.95575
Name: Zone 1 Power Consumption, Length: 52416, dtype: float64
```

*** Split the dataset.**

Hình 2-9 Tập dữ liệu và nhóm ngày

In [3]:
```
# split data
df_train = Real_Price.sample(frac=0.8,random_state=123)
df_test = Real_Price.drop(df_train.index)
print(df_train)
```

```
DateTime
4/19/2017 1:40      24738.85899
8/3/2017 7:20       28058.42397
11/25/2017 20:20    37390.76923
4/8/2017 19:20      42824.88698
12/13/2017 6:10     22904.94297
                       ...
8/1/2017 21:10      48215.22752
7/27/2017 6:10      26305.91362
9/11/2017 0:10      31437.87611
6/17/2017 14:20     35640.79470
5/31/2017 5:40      21516.59016
Name: Zone 1 Power Consumption, Length: 41933, dtype: float64
```

Hình 2-10 Chia tệp dữ liệu

2.2 Ma trận đánh giá

2.2.1 RMSE

Trung bình, sai số bình phương trung bình gốc (RMSE) là một số liệu cho chúng ta biết các giá trị dự đoán của chúng ta khác nhau bao xa so với các giá trị quan sát được trong một mô hình(3). Nó được tính như sau:

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

- $\hat{y}_i$ là giá trị ước lượng
- y_i là biến độc lập
- n = (N – k – 1)
- N: số tổng lượng quan sát
- k: tổng lượng biến

Tính RMSE bằng Python

Giả sử chúng ta có các mảng giá trị dự đoán và thực tế sau:

```
thực tế= [34, 37, 44, 47, 48, 48, 46, 43, 32, 27, 26, 24]
dự đoán = [37, 40, 46, 44, 46, 50, 45, 44, 34, 30, 22, 23]
```

Hình 2-11 Giá trị thực và giá trị dự đoán

Để tính toán RMSE giữa giá trị thực và giá trị dự đoán, chúng ta chỉ cần lấy căn bậc hai của hàm mean_squared_error() từ thư viện sklearn.metrics:

```
#nhập thư viện cần thiết
từ sklearn.metrics nhập mean_squared_error
 từ toán nhập sqrt

#tính toán RMSE
 sqrt(mean_squared_error(thực, trước))

2.4324199198
```

Hình 2-12 Tính RMSE

RMSE là một cách hữu ích để xem mức độ phù hợp của một mô hình với tập dữ liệu. RMSE càng lớn thì chênh lệch giữa giá trị dự đoán và giá trị quan sát càng lớn, nghĩa là mô hình phù hợp với dữ liệu càng tệ. Ngược lại, RMSE càng nhỏ thì mô hình càng phù hợp với dữ liệu.

Có thể đặc biệt hữu ích khi so sánh RMSE của hai mô hình khác nhau với nhau để xem mô hình nào phù hợp với dữ liệu hơn(3).

2.2.2 MAPE

Lỗi tỷ lệ phần trăm tuyệt đối trung bình (MAPE) thường được sử dụng để đo lường độ chính xác dự đoán của các mô hình(4). Nó được tính như sau:

$$MAPE = (1/n) * \sum(\left|\frac{\widehat{y_i} - y_i}{y_i}\right|) * 100$$

- n là số lượng các mẫu trong tập dữ liệu
- $\widehat{y_i}$ là giá trị dự đoán thứ i
- y_i là giá trị thực tế thứ i

MAPE được sử dụng phổ biến vì nó dễ hiểu và dễ giải thích. Ví dụ: giá trị MAPE là 11,5% có nghĩa là chênh lệch trung bình giữa giá trị dự đoán và giá trị thực tế là 11,5%.

Giá trị MAPE càng thấp thì mô hình càng có khả năng dự đoán các giá trị tốt hơn. Ví dụ: một mô hình có MAPE là 5% sẽ chính xác hơn một mô hình có MAPE là 10%.

Cách tính MAPE trong Python

Không có hàm Python tích hợp để tính MAPE, nhưng chúng ta có thể tạo một hàm đơn giản để làm như vậy:

```
nhập numpy dưới dạng np

bản đồ def ( thực tế , dự đoán ):
    thực tế, pred = np.array(actual), np.array(pred)
    trả lại np.mean(np.abs((thực tế - dự đoán) / thực tế)) * 100
```

Hình 2-13 Hàm tính MAPE

Sau đó, chúng ta có thể sử dụng hàm này để tính toán MAPE cho hai mảng: một mảng chứa các giá trị dữ liệu thực tế và một mảng chứa các giá trị dữ liệu dự đoán.

```
thực = [12, 13, 14, 15, 15,22, 27]
trước = [11, 13, 14, 14, 15, 16, 18]

mape (thực tế, dự đoán)

10.8009
```

Hình 2-14 Tính MAPE

Từ kết quả, chúng ta có thể thấy rằng sai số phần trăm tuyệt đối trung bình cho mô hình này là **10,8009%**. Nói cách khác, chênh lệch trung bình giữa giá trị dự đoán và giá trị thực tế là 10,8009%.

Ý nghĩa của MAPE

1. Phụ thuộc vào giá trị thực tế: MAPE chia tỷ lệ phần trăm sai lệch theo giá trị thực tế. Nếu giá trị thực tế hoặc bằng 0, phép chia này có thể dẫn đến sai sót hoặc không xác định. Ví dụ, khi $y_i = 0$, tỷ lệ phần trăm sai lệch sẽ không có ý nghĩa hoặc không thể tính được.

2. Nhạy cảm với giá trị thay đổi gần 0: MAPE không xử lý tốt khi giá trị thực tế gần 0 hoặc có giá trị rất nhỏ. Khi giá trị thực tế xấp xỉ 0, các sai số nhỏ sẽ có ảnh hưởng lớn đến tỷ lệ phần trăm sai lệch, gây ra biểu đồ lệch lớn và không chính xác.

CHƯƠNG 3 ĐỀ XUẤT VÀ ĐÁNH GIÁ KẾT QUẢ MÔ HÌNH

Trong lĩnh vực dự báo chuỗi thời gian, việc nghiên cứu và áp dụng các mô hình học sâu đã đóng vai trò quan trọng trong việc dự báo xu hướng và dự đoán trong tương lai

Trong bối cảnh đó, mô hình LSTM (Long Short-Term Memory) và RNN (Recurrent Neural Network) đã thu hút sự quan tâm và trở thành công cụ quan trọng cho việc dự báo chuỗi thời gian.

Trong bài nghiên cứu này, chúng tôi đề xuất sử dụng mô hình LSTM và RNN để dự báo mức tiêu thụ năng lượng của một thành phố. Mục tiêu chính của chúng tôi là xây dựng một mô hình dự báo chính xác và tin cậy, giúp đưa ra dự đoán về xu hướng tiêu thụ năng lượng trong tương lai.

Mô hình LSTM và RNN là hai mô hình học sâu phổ biến được sử dụng rộng rãi trong việc dự báo chuỗi thời gian. Cả hai mô hình đều có khả năng xử lý thông tin chuỗi và học các mối quan hệ phức tạp trong dữ liệu. Mô hình LSTM được thiết kế đặc biệt để giải quyết vấn đề biến mất đường cong dài và lưu giữ thông tin trong thời gian dài. Trong khi đó, mô hình RNN có khả năng nhớ thông tin từ quá khứ nhưng có hạn chế trong việc xử lý chuỗi dài và quan hệ phụ thuộc xa.

3.1 Đề xuất

Cấu trúc của mô hình LSTM bao gồm các đơn vị LSTM (Long Short-Term Memory) được xếp chồng lên nhau. Mỗi đơn vị LSTM bao gồm các cổng đặc biệt để điều chỉnh thông tin và quá trình truyền thông tin qua thời gian. Các cổng chính bao gồm cổng quên (forget gate) để quyết định xem thông tin nào nên bị loại bỏ, cổng đầu vào (input gate) để quyết định thông tin nào nên được cập nhật và cổng đầu ra (output gate) để quyết định thông tin nào nên được truyền tiếp. Cấu trúc này giúp LSTM lưu trữ và xử lý thông tin trong thời gian dài, giúp nó phù hợp cho dự báo chuỗi thời gian.

```
model = Sequential()
model.add(LSTM(67, input_shape=(train_X.shape[1], train_X.shape[2]), return_sequences=True))
# model.add(Dropout(0.3))
model.add(LSTM(45, return_sequences=True))
model.add(LSTM(30))
model.add(Dropout(0.3))
model.add(Dense(1))
model.compile(loss='mse', optimizer='adam')

# fit network
history = model.fit(train_X, train_y, epochs=100, batch_size=30, validation_split=0.2, verbose=2, shuffle=False)
```

Hình 3-1 Cấu trúc LSTM

Mô hình RNN bao gồm các đơn vị RNN (Recurrent Neural Network) được kết nối với nhau theo thứ tự thời gian. Mỗi đơn vị RNN sẽ nhận đầu vào từ thời điểm trước đó và tạo ra đầu ra tại thời điểm hiện tại. Cấu trúc này cho phép RNN xử lý thông tin chuỗi và truyền thông tin qua thời gian.

```
# Initialising the RNN
regressor = Sequential()

# Adding the input layer and the LSTM layer
regressor.add(LSTM(units = 100, activation = 'sigmoid', input_shape = (None, 1)))

# Adding the output layer
regressor.add(Dense(units = 1))

# Compiling the RNN
regressor.compile(optimizer = 'adam', loss = 'mean_squared_error')

# Fitting the RNN to the Training set
regressor.fit(X_train, y_train, batch_size = 5, epochs =100 )
```

Hình 3-2 Cấu trúc RNN

Các tham số quan trọng trong mô hình LSTM và RNN bao gồm số lượng đơn vị (units) trong mỗi mạng, hàm kích hoạt (activation function) được sử dụng, tốc độ học (learning rate) và số lượng lớp ẩn (hidden layers). Số lượng đơn vị ảnh hưởng đến khả năng mô hình học và xử lý thông tin, trong khi hàm kích hoạt giúp mô hình học các mối quan hệ phi tuyến. Tốc độ học ảnh hưởng đến tốc độ hội tụ của quá trình huấn luyện và số lượng lớp ẩn cho phép mô hình học các mức trừu tượng khác nhau của dữ liệu(5).

3.1.1 RNN

Mạng RNN (Recurrent Neural Network) là một kiến trúc mạng nơ-ron nhân tạo đặc biệt được thiết kế để xử lý dữ liệu tuần tự, có tính chất thời gian. Nó có khả năng

lưu trữ thông tin từ các thời điểm trước đó và sử dụng thông tin đó để dự đoán các thời điểm tiếp theo trong dãy dữ liệu.

Cấu trúc mạng RNN bao gồm các thành phần chính sau:

- Đầu vào (Input): Dữ liệu đầu vào là một chuỗi dữ liệu tuần tự được biểu diễn dưới dạng ma trận có kích thước (sequence_length, input_dimension). Mỗi thời điểm trong chuỗi dữ liệu sẽ được đưa vào một layer RNN.
- Layer RNN (RNN Layer): Layer này thực hiện xử lý dữ liệu tuần tự. Trong một RNN, các đầu vào từ các thời điểm trước đó được kết hợp với đầu vào hiện tại để tính toán đầu ra. Một trong những kiểu RNN phổ biến là LSTM (Long Short-Term Memory), như trong đoạn code bạn đã cung cấp.
- Đầu ra (Output): Đầu ra của mạng RNN có thể là một giá trị dự đoán cho mỗi thời điểm trong chuỗi dữ liệu hoặc chỉ là một giá trị dự đoán ở thời điểm cuối cùng. Đối với bài toán dự đoán dữ liệu dạng số, một layer fully connected (Dense) được sử dụng để tính toán đầu ra.
- Tổng quan, mạng RNN có các layer RNN được xếp chồng lên nhau để xử lý dữ liệu tuần tự theo các thời điểm. Các kết nối giữa các layer cho phép truyền dữ liệu qua thời gian, từ các thời điểm trước đó đến các thời điểm sau đó.

Quá trình huấn luyện (tiếp tục): Quá trình huấn luyện của mạng RNN thường bao gồm các bước sau:

- Chuẩn bị dữ liệu huấn luyện: Dữ liệu huấn luyện được chuẩn bị dưới dạng chuỗi các mẫu dữ liệu tuần tự, với đầu vào và đầu ra tương ứng.
- Xây dựng mô hình: Mô hình RNN được xây dựng bằng cách khởi tạo một mạng RNN và thêm các layer RNN, layer đầu vào và layer đầu ra.
- Biên dịch mô hình: Mô hình được biên dịch bằng cách chỉ định thuật toán tối ưu (optimizer) và hàm mất mát (loss function) để đánh giá sự khác biệt giữa đầu ra dự đoán và giá trị thực tế.
- Huấn luyện mô hình: Mô hình được huấn luyện bằng cách đưa dữ liệu huấn luyện vào mạng và cập nhật các trọng số của các nơ-ron thông qua quá trình lan truyền

ngược. Quá trình này được thực hiện qua nhiều epoch (vòng lặp) để tăng độ chính xác của mô hình.

- Đánh giá mô hình: Sau khi huấn luyện, mô hình được đánh giá trên dữ liệu kiểm tra để đánh giá hiệu suất và độ chính xác của mô hình.
- Dự đoán: Mô hình đã huấn luyện có thể được sử dụng để dự đoán các giá trị đầu ra cho dữ liệu mới.

Mạng RNN có thể được áp dụng cho nhiều loại bài toán dự đoán dữ liệu tuần tự, như dự đoán chuỗi thời gian, dịch máy, nhận dạng giọng nói và nhiều ứng dụng khác. Đặc điểm quan trọng của mạng RNN là khả năng xử lý thông tin tuần tự và khả năng ghi nhớ thông tin từ các thời điểm trước đó, làm cho nó phù hợp với nhiều bài toán có tính chất thời gian.

Tham số của mạng RNN: Mạng RNN có một số tham số quan trọng cần được xác định và điều chỉnh để đạt được hiệu suất tốt. Một số tham số quan trọng bao gồm:

- Số lượng units: Đây là số lượng đơn vị (nơ-ron) trong layer RNN. Số lượng units ảnh hưởng đến khả năng mô hình học và khả năng nắm bắt thông tin từ dữ liệu tuần tự.
- Hàm kích hoạt (activation function): Hàm kích hoạt được áp dụng cho đầu ra của mỗi nơ-ron trong layer RNN. Trong đoạn code, hàm sigmoid được sử dụng làm hàm kích hoạt.
- Kích thước đầu vào (input shape): Đây là kích thước của dữ liệu đầu vào, thường được biểu diễn dưới dạng (None, input_dimension), trong đó input_dimension là số chiều của dữ liệu đầu vào.
- Hàm mất mát (loss function): Đây là hàm được sử dụng để đánh giá sự khác biệt giữa đầu ra dự đoán và giá trị thực tế. Trong đoạn code, hàm mất mát mean squared error (MSE) được sử dụng.
- Thuật toán tối ưu (optimizer): Thuật toán tối ưu được sử dụng để điều chỉnh các trọng số của mạng. Trong đoạn code, thuật toán Adam được sử dụng.

Mạng RNN có khả năng mô hình hóa các quan hệ phức tạp giữa các thời điểm trong dữ liệu tuần tự. Điều này cho phép nó phù hợp cho các bài toán như dự đoán chuỗi thời gian, ngôn ngữ tự nhiên, và xử lý dữ liệu tuần tự khác.

Một ưu điểm khác của mạng RNN là khả năng xử lý dữ liệu đầu vào có độ dài khác nhau. Trong đoạn code, input_shape được đặt là (None, 1), cho phép mạng RNN xử lý dữ liệu đầu vào có số lượng thời điểm không xác định và chiều dữ liệu đầu vào là 1.

Batch size và số epoch: Trong quá trình huấn luyện, batch size (kích thước lô) và số epoch (số lần lặp lại dữ liệu huấn luyện) được chỉ định. Batch size xác định số lượng mẫu dữ liệu được đưa vào mạng cùng một lúc, và số epoch xác định số lần lặp lại việc huấn luyện trên toàn bộ dữ liệu huấn luyện.

Dự đoán: Sau quá trình huấn luyện, mạng RNN đã học được các mẫu và quy luật trong dữ liệu tuần tự. Để dự đoán giá trị cho dữ liệu mới, chúng ta có thể sử dụng mô hình đã được huấn luyện và đưa dữ liệu mới qua mạng để nhận được đầu ra dự đoán.

Mạng RNN là một trong những kiến trúc quan trọng trong lĩnh vực học sâu (deep learning) và xử lý dữ liệu tuần tự. Nó có thể được áp dụng cho nhiều bài toán như dự đoán chuỗi thời gian, xử lý ngôn ngữ tự nhiên, dịch máy, nhận dạng giọng nói và nhiều bài toán khác có tính chất thời gian.

3.1.2 LSTM

A. Mô tả cấu trúc mạng

Trong mạng LSTM, cấu trúc của các đơn vị LSTM cho phép chúng ghi nhớ thông tin từ quá khứ và điều chỉnh thông tin này theo cách phù hợp. Điều này giúp mạng LSTM xử lý được các chuỗi dữ liệu dài và có mối quan hệ phức tạp.

Đầu vào (Input): Đầu vào của mạng LSTM là một chuỗi dữ liệu, thường là dữ liệu chuỗi thời gian. Mỗi phần tử trong chuỗi được gọi là một time step và có thể chứa một hoặc nhiều đặc trưng (features).

Layer LSTM (LSTM Layer): Layer LSTM là thành phần chính của mạng LSTM. Mỗi LSTM layer bao gồm một tập hợp các đơn vị LSTM (LSTM units) hoạt động độc

lập. Mỗi đơn vị LSTM có khả năng lưu trữ và truy cập vào thông tin từ quá khứ và điều chỉnh thông tin này theo cách phù hợp.

Đầu ra (Output): Đầu ra của mạng LSTM thường là một hoặc nhiều đơn vị Dense (fully connected) đưa ra các dự đoán dựa trên thông tin đã học được từ dữ liệu chuỗi đầu vào.

B. Giải thích các hệ số

I. Lớp **Sequential()**

Trong thư viện Keras là một cách tiếp cận đơn giản và phổ biến để xây dựng các mô hình neural network tuần tự. Trong mô hình tuần tự, các lớp được xếp chồng lên nhau theo một thứ tự nhất định, và dữ liệu được truyền qua từng lớp theo chiều tuần tự.

Ưu điểm chính của việc sử dụng lớp **Sequential()** là tính đơn giản và dễ sử dụng. Chúng ta chỉ cần thêm các lớp vào mô hình bằng cách sử dụng phương thức **add()**, mà không cần phải quan tâm đến việc kết nối các lớp hay quản lý đầu vào/đầu ra của từng lớp.

Lớp **Sequential()** được sử dụng để xây dựng một mạng neural network tuần tự với các lớp LSTM xếp chồng lên nhau, lớp Dropout để tránh overfitting, và lớp Dense để tạo ra đầu ra dự đoán.

Tuy nhiên, lớp **Sequential()** có một số hạn chế. Một trong số đó là không thể xây dựng được các mô hình có các nhánh (branches) hoặc các kết nối bỏ qua (skip connections) phức tạp. Điều này giới hạn khả năng mô hình hóa những mô hình có kiến trúc phức tạp hơn, như các mô hình có nhiều đầu vào hoặc nhiều đầu ra, mô hình có mạng nơ-ron tái sử dụng, hoặc mô hình có kết nối bỏ qua để tạo ra các đường dẫn ngắn hơn trong mạng.

II. Phương thức **add()**

Phương thức **add()** được sử dụng để thêm các lớp vào mô hình tuần tự (**Sequential()**). phương thức **add()** được sử dụng để thêm các lớp LSTM, Dropout và

Dense vào mô hình. Các tham số của phương thức **add()** sẽ khác nhau tùy thuộc vào loại lớp mà chúng ta đang thêm vào.

add(LSTM) là một phương thức của lớp **Sequential** trong thư viện Keras để thêm một lớp LSTM (Long Short-Term Memory) vào mô hình. Đối số **units** xác định số lượng đơn vị (nơ-ron) trong lớp LSTM. Khi sử dụng lớp LSTM, chúng ta có thể tuỳ chỉnh các tham số khác như **input_shape** để xác định kích thước đầu vào của mỗi mẫu dữ liệu chuỗi. Cũng có thể sử dụng các tham số khác như **return_sequences** để quyết định liệu lớp LSTM có trả về chuỗi kết quả cho mỗi thời điểm hay chỉ kết quả cuối cùng. Lớp LSTM trong mô hình được thêm vào thông qua phương thức **add()**, và các lớp LSTM có thể được xếp chồng lên nhau hoặc kết hợp với các lớp khác để tạo thành một kiến trúc mạng phức tạp hơn.

Lớp LSTM: Trong lớp LSTM, chúng ta cần chỉ định số lượng đơn vị (units) và các tham số khác như **input_shape**, **return_sequences** (hoặc **return_state**) và **activation**

Units: tham số **units** đề cập đến số lượng đơn vị (nơ-ron) trong một lớp cụ thể. chúng ta cần chỉ định số lượng đơn vị trong lớp đó bằng tham số **units**. Số lượng đơn vị xác định kích thước của đầu ra từ lớp đó và ảnh hưởng đến khả năng học và biểu diễn của mạng neural network.

Trong lớp Dense, ví dụ như **model.add(Dense(units=64))**, **units=64** xác định rằng lớp này sẽ có 64 đơn vị (nơ-ron). Mỗi đơn vị trong lớp Dense sẽ thực hiện một phép tính tuyến tính trên đầu vào của nó, kết hợp với hệ số trọng số tương ứng, và áp dụng một hàm kích hoạt để tạo ra đầu ra.

Số lượng đơn vị (units) trong một lớp ảnh hưởng đến khả năng mô hình học và biểu diễn các mối quan hệ phức tạp giữa dữ liệu đầu vào và đầu ra. Một số lượng đơn vị lớn hơn có thể cho phép mạng học và biểu diễn các mẫu phức tạp hơn, nhưng đồng thời cũng làm tăng độ phức tạp và số lượng tham số của mô hình.

Việc chọn số lượng đơn vị phù hợp là một quá trình thử và sai, có thể dựa trên kiến thức về bài toán cụ thể hoặc thông qua việc thực hiện các phương pháp tối ưu hóa

và đánh giá trên tập huấn luyện. Qua quá trình huấn luyện và đánh giá, chúng ta có thể điều chỉnh số lượng đơn vị để tìm ra giá trị tối ưu cho mô hình của mình.

input_shape: tham số **input_shape** được sử dụng để chỉ định kích thước của đầu vào (input) cho một lớp trong mô hình.

Khi thêm một lớp vào mô hình sử dụng phương thức **add()** trong lớp **Sequential** của thư viện Keras, chúng ta thường cần xác định kích thước của đầu vào cho lớp đó thông qua tham số **input_shape**. Đây là một tuple hoặc list chứa thông tin về kích thước của dữ liệu đầu vào.

Ở đây, **input_shape** được xác định bằng cách sử dụng **train_X.shape[1]** và **train_X.shape[2]**. Đây là thông tin về kích thước của dữ liệu huấn luyện **train_X**. Thông thường, **train_X** là một mảng đa chiều có kích thước (số mẫu, số thời điểm, số đặc trưng). Vì vậy, **train_X.shape[1]** sẽ là số thời điểm trong mỗi mẫu và **train_X.shape[2]** sẽ là số đặc trưng trong mỗi thời điểm.

Thông qua **input_shape**, mô hình sẽ biết kích thước của dữ liệu đầu vào và có thể xác định kích thước các trọng số tương ứng cho các lớp trong mạng neural network. Điều này rất quan trọng để đảm bảo tính nhất quán giữa kích thước của dữ liệu và các phép tính trong mạng.

Lưu ý rằng tham số **input_shape** chỉ cần được xác định cho lớp đầu tiên trong mô hình. Đối với các lớp tiếp theo, Keras sẽ tự động suy ra kích thước đầu vào dựa trên kết quả của lớp trước đó.

Tham số **return_sequences:** được sử dụng trong các lớp LSTM (Long Short-Term Memory) để xác định liệu lớp đó có trả về chuỗi kết quả cho các thời điểm đầu ra hay không.

Một mô hình LSTM có thể được sử dụng để xử lý dữ liệu theo chuỗi, trong đó thông tin từ các thời điểm trước đó được truyền qua và lưu trữ trong các trạng thái ẩn của mạng. Mỗi lớp LSTM có thể có nhiều đơn vị (nơ-ron), và mỗi đơn vị xử lý thông tin từ một thời điểm và truyền thông tin sang các đơn vị tiếp theo.

Trong một mô hình LSTM thông thường, chỉ lớp cuối cùng của mạng sẽ trả về kết quả cho mỗi chuỗi dữ liệu đầu ra, trong khi các lớp LSTM trước đó thường chỉ trả về kết quả cho thời điểm cuối cùng. Điều này được thực hiện bởi tham số **return_sequences**.

Khi **return_sequences=True**, lớp LSTM sẽ trả về chuỗi kết quả cho mỗi thời điểm trong chuỗi đầu vào. Điều này có ý nghĩa là lớp LSTM sẽ có đầu ra có kích thước (số mẫu, số thời điểm, số đơn vị). Thường thì các lớp LSTM trung gian trong mô hình sẽ được đặt **return_sequences=True**, để truyền tiếp thông tin cho các lớp LSTM sau đó.

Khi **return_sequences=False** (giá trị mặc định), chỉ kết quả cho thời điểm cuối cùng trong chuỗi đầu vào sẽ được trả về. Điều này dẫn đến đầu ra có kích thước (số mẫu, số đơn vị). Thường thì lớp LSTM cuối cùng trong mạng sẽ không có tham số **return_sequences**, để thu thập thông tin từ toàn bộ chuỗi đầu vào và đưa ra kết quả tương ứng.

Quyết định sử dụng **return_sequences=True** hay **return_sequences=False** phụ thuộc vào bài toán cụ thể và cách xử lý dữ liệu chuỗi. Trong một số trường hợp, chúng ta có thể muốn lấy thông tin từ mỗi thời điểm trong chuỗi đầu vào, trong khi trong những trường hợp khác, chỉ cần kết quả cuối cùng là đủ để đưa ra dự đoán.

add(Dropout) là một phương thức của lớp **Sequential** trong thư viện Keras để thêm một lớp Dropout vào mô hình.

Lớp Dropout là một phương pháp chính quy hóa (regularization) trong mạng neural network, được sử dụng để ngẫu nhiên tắt (bỏ qua) một số đơn vị (nơ-ron) trong quá trình huấn luyện. Điều này giúp hạn chế hiện tượng quá khớp (overfitting) và cải thiện khả năng tổng quát hóa của mô hình.

Trong Keras, chúng ta có thể sử dụng **add(Dropout(rate))** để thêm một lớp Dropout vào mô hình. Đối số **rate** xác định tỷ lệ (giữa 0 và 1) các đơn vị sẽ bị tắt. Ví

dụ: **add(Dropout(rate=0.3))** sẽ thêm một lớp Dropout với tỷ lệ 0.3, tức là khoảng 30% đơn vị sẽ bị tắt trong quá trình huấn luyện.

Lớp Dropout được thêm vào sau một lớp trước đó trong mô hình, và các đơn vị trong lớp trước đó có khả năng bị tắt trong quá trình huấn luyện. Điều này giúp mạng trở nên chống chịu với nhiễu và không cho phép các đơn vị phụ thuộc quá nhiều vào nhau.

Lớp Dropout thường được sử dụng sau các lớp có số lượng đơn vị lớn, như Dense hoặc LSTM, để giảm hiện tượng quá khớp và cải thiện khả năng tổng quát hóa của mô hình. Ngoài ra, lớp Dropout cũng giúp cân bằng giữa các đơn vị và giảm khả năng một số đơn vị quá phụ thuộc vào những đặc trưng cụ thể trong dữ liệu.

Lớp Dropout trong mạng neural network là một công cụ hữu ích để tăng tính linh hoạt và khả năng chống chịu của mô hình trong quá trình huấn luyện và dự đoán.

add(Dense) là một phương thức của lớp **Sequential** trong thư viện Keras để thêm một lớp Dense vào mô hình.

Lớp Dense là một lớp đầy đủ kết nối (fully connected layer) trong mạng neural network. Nó là một lớp mạng neural cổ điển, trong đó mỗi nơ-ron trong lớp này được kết nối với tất cả các nơ-ron trong lớp trước đó (hoặc đầu vào nếu nó là lớp đầu tiên).

Trong Keras, chúng ta có thể sử dụng **add(Dense(units))** để thêm một lớp Dense vào mô hình. Đối số **units** xác định số lượng đơn vị (nơ-ron) trong lớp Dense. Ví dụ: **add(Dense(units=64))** sẽ thêm một lớp Dense với 64 đơn vị vào mô hình.

Lớp Dense thường được sử dụng làm lớp cuối cùng trong mô hình để dự đoán kết quả cuối cùng. Đầu ra của lớp Dense có thể có kích thước khác nhau tùy thuộc vào bài toán cụ thể, ví dụ như một số đơn vị cho bài toán hồi quy (regression) hoặc số đơn vị tương ứng với số lớp phân loại trong bài toán phân loại (classification).

Lớp Dense trong mạng neural network là một lớp quan trọng để học và biểu diễn các quan hệ phi tuyến tính và phi tuyến tính giữa đầu vào và đầu ra. Các trọng số trong lớp Dense sẽ được cập nhật trong quá trình huấn luyện để tìm ra mối quan hệ tốt nhất giữa các đặc trưng đầu vào và kết quả dự đoán.

III. Phương thức **compile()**

Trong mô hình **Sequential** của thư viện Keras được sử dụng để cấu hình quá trình huấn luyện của mô hình trước khi nó được sử dụng để fit vào dữ liệu huấn luyện.

Khi gọi **compile()**, chúng ta cần chỉ định hai thông số chính:

Hàm mất mát (loss function): Đây là hàm được sử dụng để đo lường mức độ sai khác giữa giá trị dự đoán của mô hình và giá trị thực tế trong quá trình huấn luyện. Trong bài toán hồi quy (regression), **mse** (mean squared error - sai số bình phương trung bình) là một lựa chọn phổ biến để đo lường mất mát. Trong bài toán phân loại (classification), chúng ta có thể sử dụng **categorical_crossentropy** cho phân loại đa lớp hoặc **binary_crossentropy** cho phân loại nhị phân.

Thuật toán tối ưu hóa (optimizer): Đây là thuật toán được sử dụng để tối ưu hóa mô hình dựa trên hàm mất mát và cập nhật các trọng số của mạng. Trong trường hợp này, **adam** (Adaptive Moment Estimation) là một thuật toán tối ưu hóa phổ biến được sử dụng trong mạng neural network. Nó kết hợp giữa hai phương pháp tối ưu hóa khác nhau là Momentum và RMSProp để đạt được tốc độ hội tụ nhanh chóng và khả năng khám phá tốt trong không gian tham số.

IV. Phương thức **fit()**

Trong mô hình **Sequential** của thư viện Keras được sử dụng để huấn luyện mô hình với dữ liệu huấn luyện đã được chuẩn bị trước đó. Nó thực hiện quá trình tối ưu hóa mô hình dựa trên dữ liệu huấn luyện và các tham số được chỉ định.

Dưới đây là giải thích chi tiết về các tham số của phương thức **fit()**:

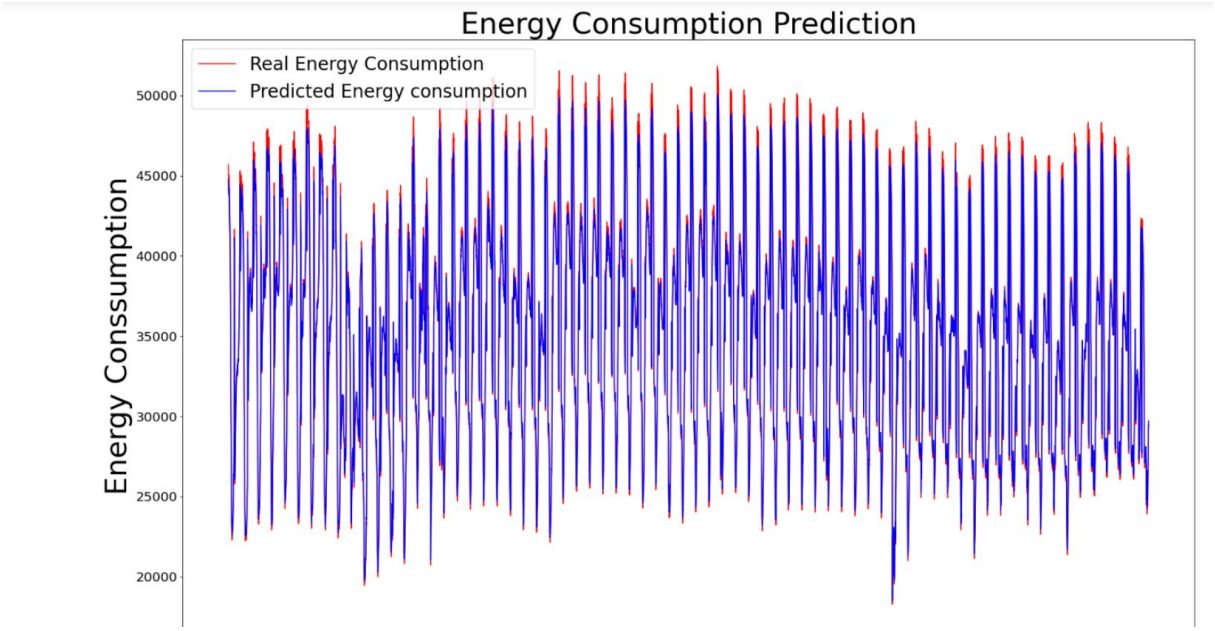
- **x**: Dữ liệu đầu vào (features) để huấn luyện mô hình. Đây có thể là một numpy array, một danh sách các array hoặc một generator.
- **y**: Nhãn tương ứng với dữ liệu huấn luyện. Cũng có thể là một numpy array, một danh sách các array hoặc một generator.
- **epochs**: Số lượng epoch (vòng lặp) mà mô hình sẽ được huấn luyện qua dữ liệu huấn luyện. Mỗi epoch tương ứng với việc sử dụng toàn bộ dữ liệu huấn luyện một lần để cập nhật trọng số của mô hình.

- **batch_size**: Kích thước batch (nhóm) mẫu được sử dụng trong quá trình huấn luyện. Dữ liệu huấn luyện được chia thành các batch nhỏ và trọng số của mô hình được cập nhật sau mỗi batch.
- **validation_data**: Dữ liệu validation (kiểm tra) để đánh giá hiệu suất của mô hình trong quá trình huấn luyện. Có thể là một tuple **(x_val, y_val)** hoặc một generator.
- **verbose**: Xác định mức độ đầu ra thông tin trong quá trình huấn luyện. Có ba giá trị khả dụng: **0** (không hiển thị thông tin), **1** (hiển thị thanh tiến trình) và **2** (hiển thị một dòng thông tin cho mỗi epoch).
- **shuffle**: Cờ chỉ định xem dữ liệu huấn luyện có được xáo trộn (shuffle) trước mỗi epoch hay không.

Phương thức **fit()** sẽ thực hiện quá trình tối ưu hóa mô hình bằng cách lặp lại các epoch và các batch, cập nhật trọng số dựa trên hàm mất mát và thuật toán tối ưu hóa đã được xác định trong **compile()**. Quá trình huấn luyện sẽ tiếp tục cho đến khi hoàn thành số lượng epoch được chỉ định hoặc khi điều kiện dừng được đáp ứng.

3.2 Kết quả

3.2.1 RNN



Hình 3-3 biểu đồ dự đoán mức tiêu thụ năng lượng so với thực tế

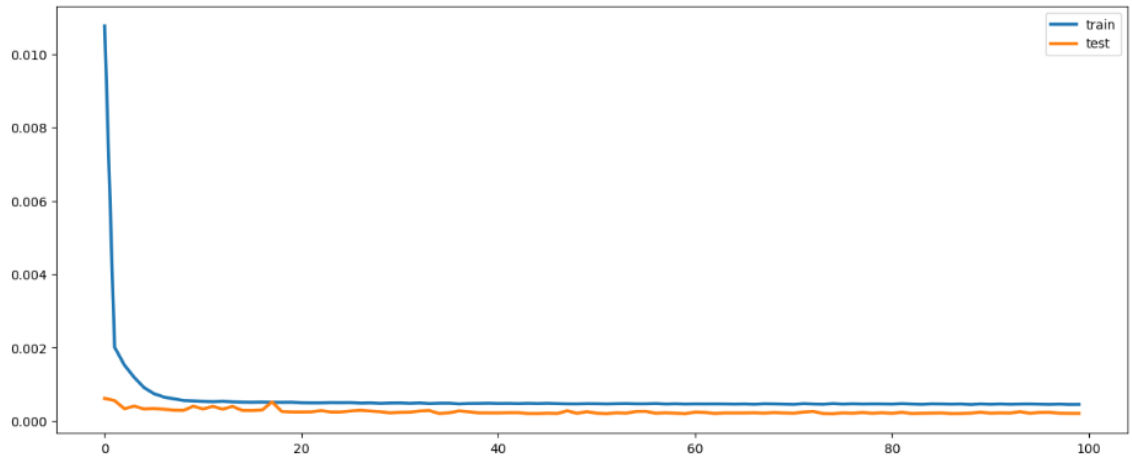
Ma trận đánh giá:

Test (Validation) RMSE = 519.6161644739401

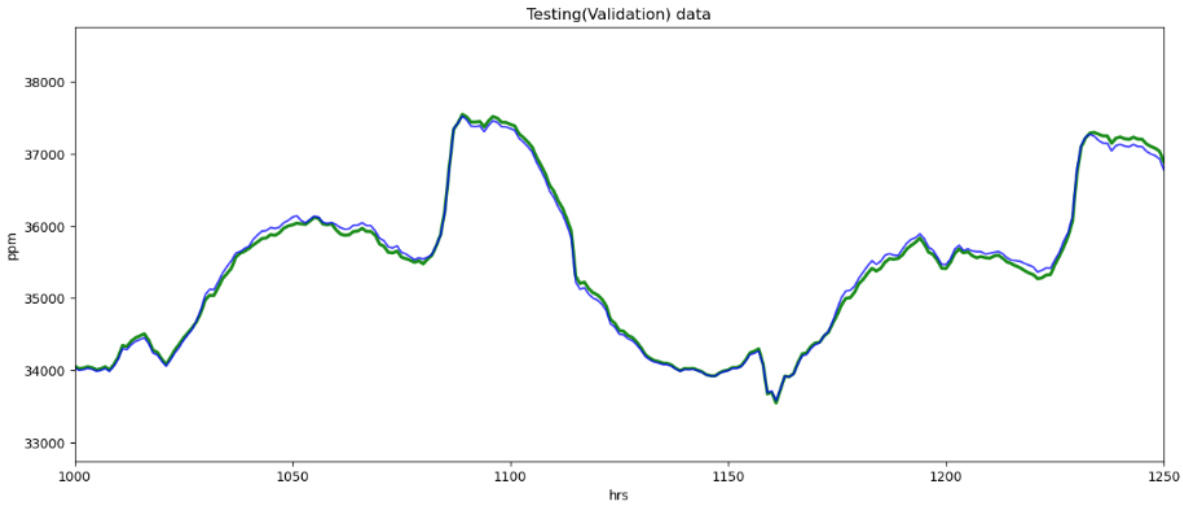
MAPE: 1.3589821083403826 %

3.2.2 LSTM

1 lớp LSTM



Hình 3-4 Biểu đồ quá trình huấn luyện mô hình LSTM để dự đoán mức tiêu thụ điện năng



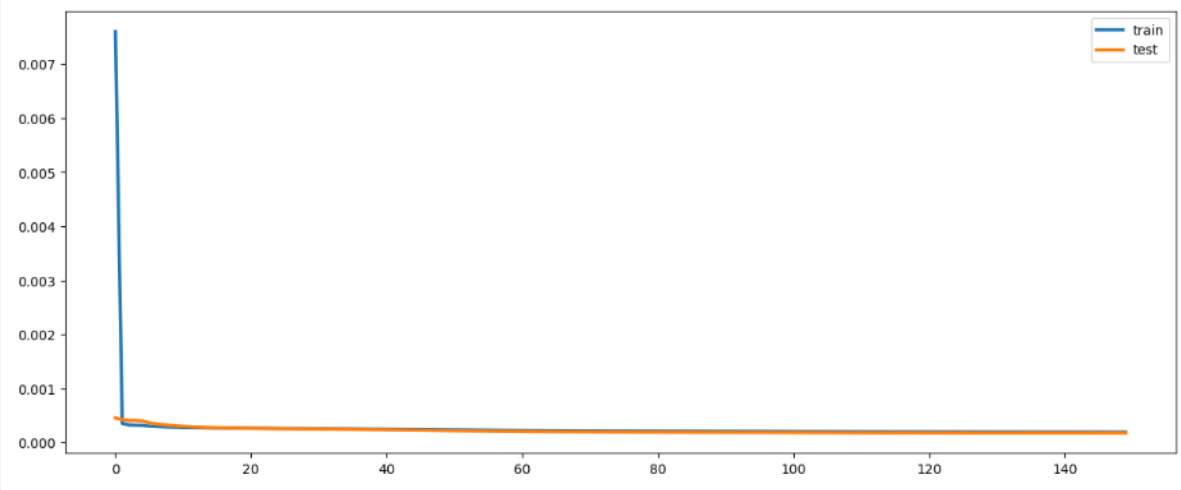
Hình 3-5 Biểu đồ kết quả dự đoán và giá trị thực tế trên tập dữ liệu kiểm tra.

Ma trận đánh giá:

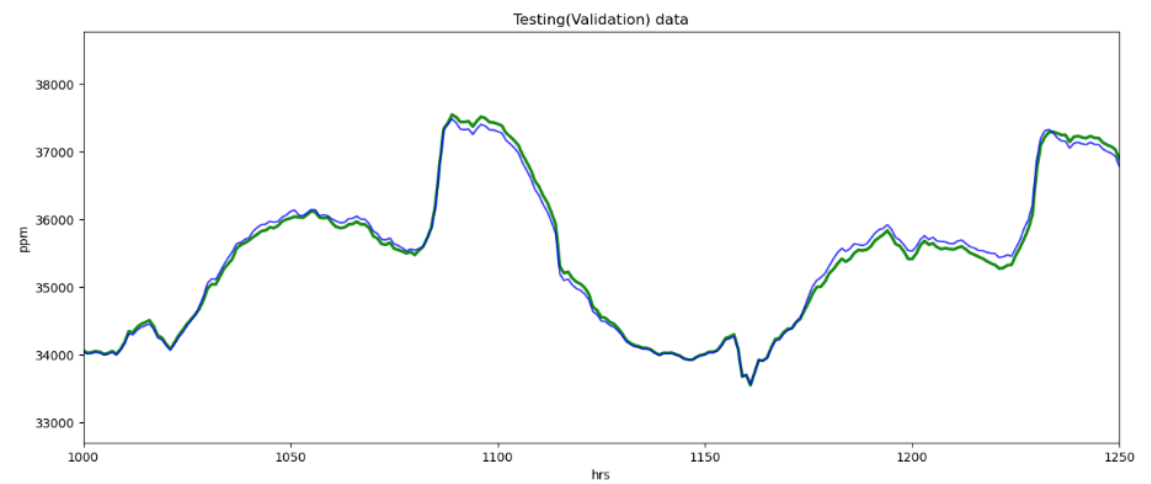
Test (Validation) RMSE = 70.40343

MAPE: 0.16513976734131575 %

2 lớp LSTM



Hình 3-6 Biểu đồ quá trình huấn luyện mô hình LSTM để dự đoán mức tiêu thụ điện năng



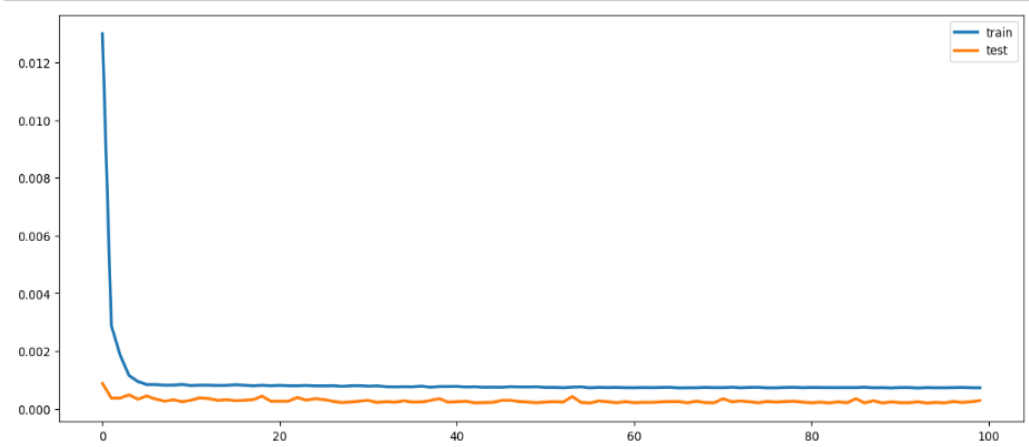
Hình 3-7 Biểu đồ kết quả dự đoán và giá trị thực tế trên tập dữ liệu kiểm tra

Ma trận đánh giá:

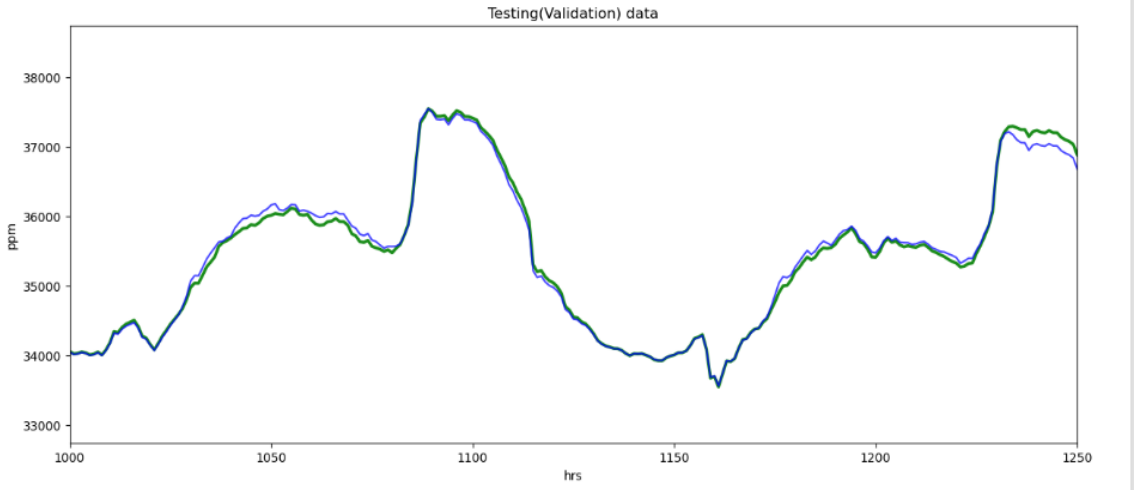
Test (Validation) RMSE = 88.007454

MAPE: 0.2064310247078538 %

3 lớp LSTM



Hình 3-8 Biểu đồ quá trình huấn luyện mô hình LSTM để dự đoán mức tiêu thụ điện năng



Hình 3-9 Biểu đồ kết quả dự đoán và giá trị thực tế trên tập dữ liệu kiểm tra

Ma trận đánh giá:

Test (Validation) RMSE = 96.25525

MAPE: 0.20641274750232697 %

3.3 Giải thích ý nghĩa của biểu đồ

a) Biểu đồ quá trình huấn luyện mô hình LSTM để dự đoán mức tiêu thụ điện năng.

Trên biểu đồ, trục x biểu diễn số lượng epoch (vòng lặp) trong quá trình huấn luyện, trong khi trục y biểu diễn giá trị của hàm loss (mean squared error) trên tập huấn luyện và tập kiểm tra (validation).

Đường màu xanh biểu diễn giá trị hàm loss trên tập huấn luyện (train), trong khi đường màu đỏ biểu diễn giá trị hàm loss trên tập kiểm tra (test). Mục tiêu của quá trình huấn luyện là giảm thiểu hàm loss để đạt được mô hình dự đoán tốt trên cả tập huấn luyện và tập kiểm tra.

Ban đầu, giá trị hàm loss trên cả hai tập dữ liệu là cao. Khi quá trình huấn luyện diễn ra, mô hình được cập nhật dần dần để tối ưu hóa dự đoán. Kết quả là giá trị hàm loss giảm dần theo số lượng epoch.

Tuy nhiên, ta cần quan tâm đến cả giá trị hàm loss trên tập kiểm tra. Nếu giá trị hàm loss trên tập kiểm tra tăng lên trong quá trình huấn luyện sau một số epoch nhất định, có thể cho thấy mô hình đang bị overfitting, tức là quá tập trung vào dữ liệu huấn luyện mà không tổng quát hoá tốt cho dữ liệu mới.

Để đánh giá chất lượng của mô hình, ta có thể so sánh giữa giá trị hàm loss trên tập huấn luyện và tập kiểm tra. Nếu giá trị hàm loss trên tập kiểm tra thấp và tiếp tục giảm theo thời gian, mô hình được cho là có khả năng dự đoán tốt trên dữ liệu mới.

Biểu đồ này cung cấp thông tin quan trọng về hiệu suất của mô hình LSTM trong quá trình huấn luyện và giúp đánh giá xem mô hình có đạt được mục tiêu dự đoán tốt hay không.

b) Biểu đồ kết quả dự đoán và giá trị thực tế trên tập dữ liệu kiểm tra.

Đường màu xanh lá cây trên biểu đồ biểu thị giá trị thực tế của mức tiêu thụ điện năng trên tập kiểm tra. Mỗi điểm trên đường màu xanh lá cây biểu diễn giá trị thực tế tại một thời điểm cụ thể.

Đường màu xanh dương trên biểu đồ biểu thị giá trị dự đoán của mức tiêu thụ điện năng trên tập kiểm tra. Đây là kết quả được mô hình LSTM dự đoán. Đường màu xanh dương cho thấy mức tiêu thụ dự đoán theo thời gian.

Bằng cách so sánh giữa đường màu xanh lá cây và đường màu xanh dương, ta có thể đánh giá hiệu suất của mô hình trong việc dự đoán mức tiêu thụ điện năng. Nếu đường màu xanh dương tiệm cận hoặc trùng với đường màu xanh lá cây, mô hình được cho là có khả năng dự đoán chính xác mức tiêu thụ điện năng trên dữ liệu kiểm tra.

3.4 Đánh giá mô hình

Ma trận / Mô hình	MAPE	RMSE
RNN	1.3589%	519.6161
1 lớp LSTM	0.1651%	70.4034
2 lớp LSTM	0.2064%	88.0074
3 lớp LSTM	0.2064%	96.2552

Bảng 2 So sánh chỉ số đánh giá

Nhìn vào các chỉ số RMSE và MAPE, mô hình LSTM với 1 lớp cho thấy hiệu suất tốt nhất trong số các mô hình đã liệt kê. RMSE của mô hình này là thấp nhất, chỉ là 70.40343, cho thấy mức độ chính xác cao trong việc dự báo. Tương tự, MAPE của mô hình LSTM với 1 lớp cũng là thấp nhất, chỉ là 0.16513976734131575%, đồng nghĩa với việc sai số dự báo thấp.

Trong khi đó, mô hình RNN có kết quả với RMSE là 519.6161644739401 và MAPE là 1.3589821083403826%. Điều này cho thấy với tập dữ liệu dùng để nghiên

cứu này, mô hình RNN dự báo không hiệu quả bằng mô hình LSTM đã được đề xuất ở trên.

Mô hình LSTM với 2 lớp và 3 lớp cũng cho kết quả tốt, nhưng RMSE và MAPE của chúng cao hơn so với LSTM với 1 lớp. Điều này có thể chỉ ra rằng việc thêm các lớp không mang lại cải thiện đáng kể trong hiệu suất dự báo.

Do đó, dựa trên các kết quả cung cấp, mô hình tốt nhất để dự báo trong trường hợp này là LSTM với 1 lớp.

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Kết quả đạt được của đề tài:

Xây dựng mô hình dự báo: Đề tài đã thành công trong việc xây dựng một mô hình dự báo mức tiêu thụ năng lượng cho thành phố Tetouan. Mô hình sử dụng các thông số như nhiệt độ, độ ẩm, tốc độ gió và các dòng chảy phân tán để dự đoán tiêu thụ năng lượng.

Nhóm đã thành công trong việc xây dựng hai mô hình dự báo sử dụng mạng nơ-ron tái phân tầng (RNN) và Long Short-Term Memory (LSTM) để dự báo mức tiêu thụ năng lượng cho thành phố Tetouan. Mô hình của chúng tôi đã được xây dựng bằng cách sử dụng các thông số như nhiệt độ, độ ẩm, tốc độ gió và các dòng chảy phân tán như đầu vào để dự đoán mức tiêu thụ năng lượng.

Phân tích yếu tố ảnh hưởng: Báo cáo đã phân tích sự ảnh hưởng của nhiệt độ, độ ẩm và tốc độ gió đến tiêu thụ năng lượng. Kết quả phân tích này giúp hiểu rõ hơn về tác động của yếu tố thời tiết đến nhu cầu năng lượng của thành phố.

Dự đoán tiêu thụ khu vực: Mô hình đã được áp dụng để dự đoán mức tiêu thụ năng lượng của từng khu vực trong Tetouan. Điều này cung cấp thông tin quan trọng cho việc lập kế hoạch và quản lý năng lượng trong các khu vực cụ thể.

Hạn chế của đề tài:

Dữ liệu đầu vào: Đề tài gặp khó khăn trong việc thu thập và xử lý dữ liệu đầu vào. Cần đảm bảo rằng dữ liệu về tiêu thụ năng lượng của từng khu vực được thu thập đầy đủ và chính xác để tăng độ chính xác của mô hình dự báo.

Yếu tố khác: Báo cáo cần xem xét các yếu tố khác có thể ảnh hưởng đến mức tiêu thụ năng lượng, như tình hình kinh tế, mật độ dân số, công nghiệp và thương mại. Điều này giúp tăng độ chính xác và toàn diện của mô hình dự báo.

Hướng phát triển của đề tài:

Nâng cao độ chính xác: Để cải thiện độ chính xác của dự báo, có thể cần nghiên cứu và sử dụng các phương pháp phân tích dữ liệu tiên tiến và mô hình hóa phức tạp

hơn. Đồng thời, cần tiếp tục cải thiện quá trình thu thập và xử lý dữ liệu đầu vào để đảm bảo tính đầy đủ và chính xác của dữ liệu.

Đưa vào yếu tố bền vững: Đề tài có thể mở rộng để xem xét các yếu tố bền vững khác như sử dụng năng lượng tái tạo, hiệu quả năng lượng và quản lý thải ra môi trường. Điều này giúp tạo ra một mô hình dự báo năng lượng toàn diện hơn, không chỉ tập trung vào tiêu thụ năng lượng mà còn vào việc xem xét cả các khía cạnh bền vững và môi trường.

Phân tích tác động xã hội và kinh tế: Đề tài có thể nghiên cứu tác động của tiêu thụ năng lượng đến các khía cạnh xã hội và kinh tế của thành phố. Việc đánh giá chi phí, tác động môi trường và tác động xã hội của việc tiêu thụ năng lượng sẽ giúp đưa ra các giải pháp và chính sách hiệu quả để quản lý và giảm thiểu tác động tiêu cực.

Áp dụng công nghệ mới: Nghiên cứu có thể tập trung vào việc áp dụng công nghệ mới như trí tuệ nhân tạo, học máy và mạng lưới điện thông minh để cải thiện mô hình dự báo và quản lý năng lượng. Việc sử dụng các công nghệ tiên tiến này có thể giúp tăng cường khả năng dự báo, tối ưu hóa việc sử dụng năng lượng và tạo ra các giải pháp thông minh hơn trong quản lý năng lượng.

Tổng kết:

Tổng kết, đề tài dự báo mức tiêu thụ năng lượng của một thành phố đã đạt được kết quả khả quan trong việc xây dựng mô hình dự báo và phân tích xu hướng tiêu thụ. Tuy nhiên, cần tiếp tục nghiên cứu và phát triển để cải thiện độ chính xác và tính ứng dụng của đề tài, bao gồm nâng cao độ chính xác, tích hợp yếu tố bền vững, phân tích tác động xã hội và kinh tế, và áp dụng công nghệ mới.

Tài liệu tham khảo

1. tetuan-city-power-consumption [Internet]. [cited 16 Tháng Năm 2023]. Available at: https://www.kaggle.com/datasets/gmkeshav/tetuan-city-power-consumption

2. Kidder GW, Montgomery CW. Oxygenation of frog gastric mucosa in vitro. Am J Physiol. Tháng Chạp 1975;229(6):1510–3.

3. rmse-python [Internet]. [cited 19 Tháng Năm 2023]. Available at: https://www.statology.org/rmse-python/

4. mape-python [Internet]. [cited 19 Tháng Năm 2023]. Available at: https://www.statology.org/mape-python/

5. 13853244_Long_Short-term_Memory [Internet]. [cited 24 Tháng Năm 2023]. Available at: https://www.researchgate.net/publication/13853244_Long_Short-term_Memory