HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

PHẠM CHÍ HÙNG

NGHIÊN CỨU CÔNG NGHỆ LSTM VÀ GIẢI PHÁP CHO BÀI TOÁN DỰ ĐOÁN LƯỢNG HÀNH KHÁCH ĐI MÁY BAY

LUẬN VĂN THẠC SĨ KỸ THUẬT

(Theo định hướng ứng dụng)

HÀ NỘI - 2019

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

PHẠM CHÍ HÙNG

NGHIÊN CỨU CÔNG NGHỆ LSTM VÀ GIẢI PHÁP CHO BÀI TOÁN DỰ ĐOÁN LƯỢNG HÀNH KHÁCH ĐI MÁY BAY

Chuyên ngành: Hệ thống Thông tin

Mã số: 8.48.01.04

LUẬN VĂN THẠC SĨ KỸ THUẬT

(Theo định hướng ứng dụng)

NGƯỜI HƯỚNG DẪN KHOA HỌC: TS. NGUYỄN VĂN THỦY

HÀ NỘI - 2019

LỜI CAM ĐOAN

Tôi là Phạm Chí Hùng, học viên khóa M17CQIS01B, khoa Đào tạo Sau đại học chuyên ngành Hệ thống Thông tin. Tôi xin cam đoan luận văn "Nghiên cứu công nghệ LSTM và giải pháp cho bài toán dự đoán lượng hành khách đi máy bay" là do tôi nghiên cứu, tìm hiểu và phát triển dưới sự hướng dẫn của TS. Nguyễn Văn Thủy. Luận văn không phải sự sao chép từ các tài liệu, công trình nghiên cứu của người khác mà không ghi rõ trong tài liệu tham khảo. Tôi xin chịu trách nhiệm về lời cam đoan này.

Tác giả luận văn

Phạm Chí Hùng

LỜI CẢM ƠN

Em xin chân thành gửi lời cảm ơn sâu sắc đến thầy giáo, TS. Nguyễn Văn Thủy - Giảng viên khoa Công nghệ Thông tin 1 - Học viện Công nghệ Bưu chính Viễn thông. Thầy đã định hướng nghiên cứu, chỉ bảo tận tình, đôn đốc đầy trách nhiệm, cho em các ý kiến đóng góp rất giá trị trong suốt quá trình làm nghiên cứu khoa học, làm luận văn, đồng thời tạo điều kiện thuận lợi để em hoàn thành luận văn này.

Em xin chân thành cảm ơn toàn thể các thầy cô Khoa Đào tạo Sau Đại học; Khoa Công nghệ Thông tin 1 - Học viện Công nghệ Bưu chính Viễn thông đã truyền đạt những kiến thức bổ ích và lý thú, giúp ích cho em trên con đường học tập và nghiên cứu của mình.

Tôi cũng xin được cảm ơn tới gia đình, những người thân, các đồng nghiệp và bạn bè đã thường xuyên quan tâm, động viên, chia sẻ kinh nghiệm, cung cấp các tài liệu hữu ích trong thời gian học tập, nghiên cứu cũng như trong suốt quá trình thực hiện luận văn tốt nghiệp.

Cuối cùng, tôi cũng xin cảm ơn tất cả những người bạn đã đóng góp ý kiến, động viên, giúp đỡ tôi hoàn thành luận văn này.

Hà Nội, ngày tháng năm 2019

Tác giả luận văn

Phạm Chí Hùng

MỤC LỤC

DANH MỤC HÌNH VẼ

DANH MỤC BẢNG BIỂU

DANH MỤC TỪ VIẾT TẮT

Từ viết tắt	Tiếng Anh	Tiếng Việt
ANN	Artificial Neural Network	Mạng nơ-ron nhân tạo
CPU	Central Processing Unit	Bộ xử lý trung tâm
IATA	International Air Transport Association	Hiệp hội Vận tải Hàng không Quốc tế
LSTM	Long Short-Term Memory	Mạng bộ nhớ dài ngắn
OS	Operating System	Hệ điều hành
RAM	Random Access Memory	Bộ nhớ truy nhập ngẫu nhiên
RNN	Recurrent Neural Network	Mạng nơ-ron hồi quy

MỞ ĐẦU

1. Tính cấp thiết của đề tài

Ngày nay, ngành hàng không là một ngành công nghiệp vô cùng phát triển, nó phục vụ nhu cầu đi lại, giao thương giữa các quốc gia, cùng với đó là sự phát triển kinh tế của các nước có đường bay quốc tế. Theo dự đoán, vào năm 2035, lượng hành khách sử dụng dịch vụ hàng không sẽ đạt 7,2 tỷ lượt, tăng gần gấp đôi con số 3,8 tỷ trong năm nay (số liệu do Hiệp hội Vận tải Hàng không Quốc tế (IATA) cung cấp). Ông Alexandre De Juniac, Tổng Giám đốc điều hành của IATA nhận xét nhu cầu đi lại bằng đường hàng không trong hai thập kỷ tới sẽ gấp đôi. Cũng theo dự báo trên, IATA cho rằng khu vực châu Á - Thái Bình Dương sẽ là nơi có nhu cầu di chuyển bằng đường không cao nhất thế giới. Để đáp ứng nhu cầu phục vụ hành khách một cách tốt nhất, việc phải có một hệ thống dự đoán lượng hành khách đi máy bay là rất cần thiết.

2. Tổng quan vấn đề nghiên cứu

Theo dự báo của IATA, ông Alexandre De Juniac đã đưa ra ba kịch bản dự báo về lĩnh vực hàng không trong giai đoạn 20 năm tới. Kịch bản thứ nhất dự báo tăng gấp đôi lượng hành khách. Kịch bản thứ hai đưa ra nhịp độ tăng trưởng hành khách hàng không gần gấp ba lần so với năm 2016. Kịch bản cuối cùng ước tính 7,2 tỷ lượt khách sử dụng dịch vụ hàng không vào năm 2035.

Cùng với đó, thị trường hàng không Trung Quốc sẽ thay thế Mỹ trở thành thị trường hàng không lớn nhất thế giới (tính cả đường bay nội địa và quốc tế) vào năm 2029. Ấn Độ cũng sẽ thay nước Anh chiếm vị trí thứ ba vào năm 2026, trong khi Indonesia sẽ lọt vào top 10 trong danh sách của IATA.

Nhận thấy nhu cầu quan trọng của việc dự đoán lượng hành khách có nhu cầu đi lại bằng đường hàng không, tôi đề xuất một phương pháp sử dụng công nghệ LSTM để dự đoán lượng hành khách đi máy bay quốc tế.

3. Mục đích nghiên cứu

- Nghiên cứu về vấn đề dự báo chuỗi thời gian, áp dụng dự đoán lượng hành khách đi máy bay quốc tế.
- Nghiên cứu và ứng dụng công nghệ LSTM.

4. Đối tượng và phạm vi nghiên cứu

4.1. Đối tượng nghiên cứu

- Công nghệ LSTM (Long Short-Term Memory).
- Vấn đề dự báo lượng hành khác đường bay quốc tế.

4.2. Phạm vi nghiên cứu

- Giới hạn nghiên cứu về công nghệ LSTM (Long Short-Term Memory).
- Nghiên cứu bài toán dự đoán chuỗi theo thời gian.

5. Phương pháp nghiên cứu

5.1. Phương pháp nghiên cứu lý thuyết

- Đọc và phân tích tài liệu về công nghệ LSTM, nghiên cứu vấn đề dự đoán chuỗi thời gian thực.

5.2. Phương pháp thực nghiệm

- Xây dựng ứng dụng xem xét vấn đề dự đoán lượng hành khách quốc tế.
- Thử nghiệm, đánh giá kết quả.

CHƯƠNG 1
TỔNG QUAN MẠNG NƠ-RON HỒI QUY

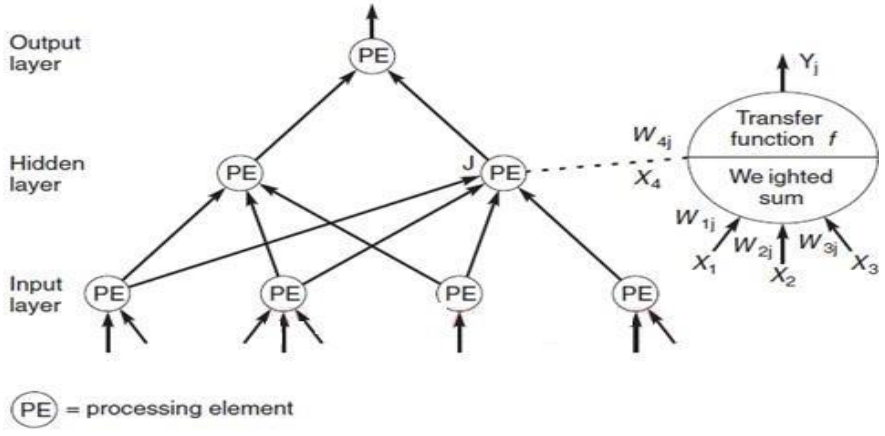
Trong chương này tôi sẽ giới thiệu về cơ sở lý thuyết về mạng nơ-ron nhân tạo, cách thức hoạt động của mạng nơ-ron, các phiên bản mở rộng của mạng nơ-ron nhân tạo.

1.1 Mạng nơ-ron nhân tạo

1.1.1 Kiến trúc mạng nơ-ron nhân tạo

Mạng nơ-ron nhân tạo (Artificial Neural Network – ANN) là một mô hình xử lý thông tin được mô phỏng dựa trên hoạt động của hệ thống thần kinh của sinh vật, bao gồm số lượng lớn các Nơ-ron được gắn kết để xử lý thông tin. ANN được giới thiệu năm 1943 bởi nhà thần kinh học Warren McCulloch và nhà logic học Walter Pits, ANN hoạt động giống như bộ não của con người, được học bởi kinh nghiệm (thông qua việc huấn luyện), có khả năng lưu giữ các tri thức và sử dụng các tri thức đó trong việc dự đoán các dữ liệu chưa biết (unseen data) [1].

Một mạng nơ-ron là một nhóm các nút nối với nhau, mô phỏng mạng nơ- ron thần kinh của não người. Mạng nơ-ron nhân tạo được thể hiện thông qua ba thành phần cơ bản: mô hình của nơ-ron, cấu trúc và sự liên kết giữa các nơ-ron. Trong nhiều trường hợp, mạng nơ-ron nhân tạo là một hệ thống thích ứng, tự thay đổi cấu trúc của mình dựa trên các thông tin bên ngoài hay bên trong chạy qua mạng trong quá trình học.



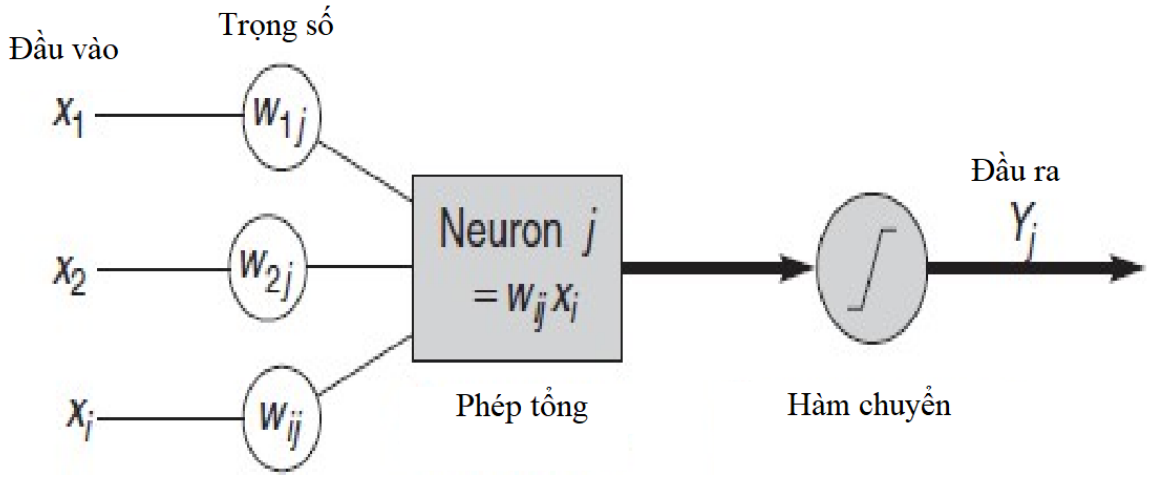
Hình 1.1 Kiến trúc mạng nơ-ron nhân tạo

Kiến trúc chung của một ANN được thể hiện ở hình 1.1, gồm 3 thành phần đó là lớp vào, lớp ẩn và lớp ra [2]

Trong đó, lớp vào thể hiện cho các đầu vào của mạng; lớp ẩn gồm các nơ-ron, nhận dữ liệu đầu vào từ các nơ-ron ở lớp trước đó và chuyển đổi các đầu vào này cho các lớp xử lý tiếp theo; lớp ra thể hiện cho các đầu ra của mạng. Trong một mạng ANN có thể có nhiều lớp ẩn.

Mạng nơron nhân tạo ngày nay gần giống với bộ não con người như: Có khả năng học, tức là sử dụng những kinh nghiệm để cải thiện hiệu suất. Khi thu thập được đủ một lượng mẫu thì ANN có thể khái quát hóa rất cao. Một ANN có thể nhận ra được một ký tự viết tay, có thể phát hiện bom ở sân bay…

1.1.2 Hoạt động của mạng nơ-ron nhân tạo



Hình 1.2 Quá trình xử lý thông tin của một mạng nơ-ron nhân tạo.

Hoạt động của mạng nơ-ron nhân tạo được thể hình ở hình 1.2 với 3 chu trình:

Đầu vào: Mỗi đầu vào tương ứng với 1 đặc trưng của dữ liệu. Ví dụ như trong ứng dụng của ngân hàng xem xét có chấp nhận cho khách hàng vay tiền hay không thì mỗi input là một thuộc tính của khách hàng như thu nhập, nghề nghiệp, tuổi, số con,…

Đầu ra: Kết quả của một ANN là một giải pháp cho một vấn đề, ví dụ như với bài toán xem xét chấp nhận cho khách hàng vay tiền hay không thì đầu ra là “có” hoặc “không”.

Trọng số liên kết: Đây là thành phần rất quan trọng của một ANN, nó thể hiện mức độ quan trọng, độ mạnh của dữ liệu đầu vào đối với quá trình xử lý thông tin chuyển đổi dữ liệu từ lớp này sang lớp khác. Quá trình học của ANN thực ra là quá trình điều chỉnh các trọng số của các dữ liệu đầu vào để có được kết quả mong muốn.

Hàm tổng: Tính tổng trọng số của tất cả các đầu vào được đưa vào mỗi nơ-ron. Hàm tổng của một nơ-ron đối với n đầu vào được tính theo công thức sau:

$$Y = \sum_{i=1}^{n} X_i W_i \tag{1}$$

Hàm chuyển đổi: Hàm tổng của một nơ-ron cho biết khả năng kích hoạt của nơ-ron đó còn gọi là kích hoạt bên trong. Các nơ-ron này có thể sinh ra một đầu ra hoặc không trong mạng ANN, nói cách khác rằng có thể đầu ra của một nơ-ron có thể được chuyển đến lớp tiếp trong mạng nơ-ron theo hoặc không. Mối quan hệ giữa hàm tổng và kết quả đầu ra được thể hiện bằng hàm chuyển đổi.

Việc lựa chọn hàm chuyển đổi có tác động lớn đến kết quả đầu ra của mạng ANN. Hàm chuyển đổi phi tuyến được sử dụng phổ biến trong mạng ANN là *sigmoid* hoặc *tanh*.

Hàm Sigmoid: $$f(z) = \frac{1}{1 + \exp(-z)}$$

Hàm Tanh: $$f(z) = \tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{2}$$

Trong đó, hàm *tanh* là phiên bản thay đổi tỉ lệ của *sigmoid* , tức là khoảng giá trị đầu ra của hàm chuyển đổi thuộc khoảng [-1, 1] thay vì [0,1] nên chúng còn gọi là hàm chuẩn hóa.

Kết quả xử lý tại các đầu ra của nơ-ron đôi khi rất lớn, vì vậy hàm chuyển đổi được sử dụng để xử lý đầu ra này trước khi chuyển đến lớp tiếp theo. Đôi khi thay vì sử dụng hàm chuyển đổi, người ta sử dụng giá trị ngưỡng để kiểm soát các đầu ra của các nơ-ron tại một lớp nào đó trước khi chuyển các đầu ra này đến các lớp tiếp theo.

Nếu đầu ra của một nơ-ron nào đó nhỏ hơn giá trị ngưỡng thì nó sẽ không được chuyển đến lớp tiếp theo.

Mạng nơ-ron của chúng ta dự đoán dựa trên lan truyền thẳng là các phép nhân ma trận cùng với hàm kích hoạt để thu được kết quả đầu ra. Nếu đầu vào x là vector 2 chiều thì ta có thể tính kết quả dự đoán bằng công thức sau:

$$\begin{aligned} y_1 &= xW_1 + b_1 \\ y_T &= \tanh(y_1) \\ y_2 &= a_1 W_2 + b_2 \\ y_T &= \hat{y} = softmax(y_2) \end{aligned} \tag{3}$$

Trong đó, y_i là đầu vào của tầng thứ i, Y_T là đầu ra của tầng thứ i sau khi áp dụng hàm kích hoạt. W_1, b_1, W_2, b_2 là các thông số cần tìm của mô hình mạng nơ-ron.

Huấn luyện để tìm các thông số cho mô hình tương đương với việc tìm các thông số W_1, b_1, W_2, b_2, sao cho độ lỗi của mô hình đạt được là thấp nhất. Ta gọi hàm độ lỗi của mô hình là hàm suy hao.

Nếu ta có N dòng dữ liệu huấn luyện và C nhóm phân lớp, khi đó hàm suy hao giữa giá trị dự đoán $\hat{y}$ và y tính như sau:

$$L(y, \hat{y}) = -\frac{1}{N} \sum_{n \in N} \sum_{i \in C} y_{n,i} log \hat{y}_{n,i} \tag{4}$$

Ý nghĩa công thức trên nghĩa là: lấy tổng trên toàn bộ tập huấn luyện và cộng dồn vào hàm loss nếu kết quả phân lớp sai. Độ dị biệt giữa hai giá trị $\hat{y}$ và y càng lớn thì độ lỗi càng cao. Mục tiêu của chúng ta là tối thiểu hóa hàm lỗi này. Ta có thể sử dụng phương pháp giảm độ dốc để tối tiểu hóa hàm lỗi. Có hai loại giảm độ dốc, một loại với tốc độ học cố định được gọi là giảm độ dốc hàng loạt, loại còn lại có tốc độ học thay đổi theo quá trình huấn luyện được gọi là giảm độ dốc ngẫu nhiên.

Phương pháp giảm độ dốc cần các đường dốc là các vector có được bằng cách lấy đạo hàm của hàm suy hao theo từng thông số $\frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial b_1}, \frac{\partial L}{\partial W_2}, \frac{\partial L}{\partial b_2}$. Để tính các đường

dốc này, ta sử dụng thuật toán lan truyền ngược. Đây là cách hiệu quả để tính đường dốc khởi điểm từ lớp đầu ra.

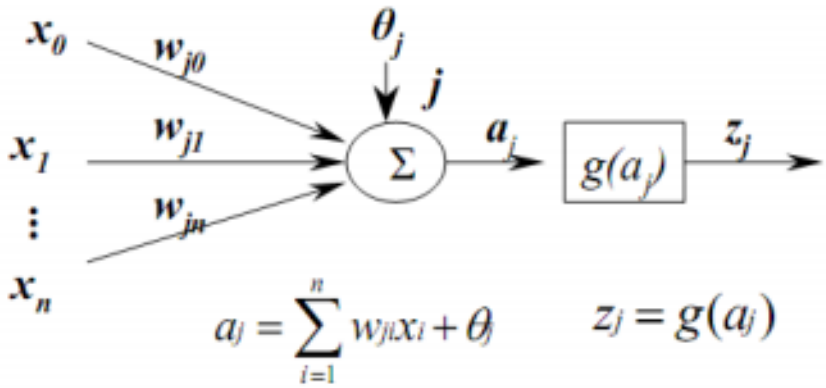
Áp dụng lan truyền ngược ta có các đại lượng:

$$\begin{aligned}
&\delta_3 = y - \hat{y} \\
&\delta_2 = (1 - tanh^2 z_1) * \delta_3 W_2^T \\
&\frac{\partial L}{\partial W_2} = a_1^T \delta_3 \\
&\frac{\partial L}{\partial b_2} = \delta_3 \\
&\frac{\partial L}{\partial W_1} = x^T \delta_2 \\
&\frac{\partial L}{\partial b_1} = \delta_2
\end{aligned} \tag{5}$$

1.2 Các thành phần cơ bản của mạng nơ-ron nhân tạo

1.2.1 Đơn vị xử lý

Còn được gọi là một nơ-ron hay một nút, thực hiện một công việc rất đơn giản: nó nhận tín hiệu vào từ các đơn vị phía trước hay một nguồn bên ngoài và sử dụng chúng để tính tín hiệu ra sẽ được lan truyền sang các đơn vị khác.



Hình 1.3 Đơn vị xử lý

Trong đó:

x_i : các đầu vào

w_{ji} : các trọng số tương ứng với các đầu vào

θ_j : độ lệch

a_j : đầu vào mạng

z_j : đầu ra của nơron

g(x): hàm chuyển (hàm kích hoạt).

Trong một mạng nơron có ba kiểu đơn vị:

- Các đơn vị đầu vào nhận tín hiệu từ bên ngoài.
- Các đơn vị đầu ra gửi dữ liệu ra bên ngoài.
- Các đơn vị ẩn, tín hiệu vào và tín hiệu ra của nó nằm trong mạng.

Mỗi đơn vị j có thể có một hoặc nhiều đầu vào: x0, x1, x2, xn, nhưng chỉ có một đầu ra z_j. Một đầu vào tới một đơn vị có thể là dữ liệu từ bên ngoài mạng, hoặc đầu ra của một đơn vị khác, hoặc là đầu ra của chính nó.

1.2.2 Hàm kết hợp

Mỗi một đơn vị trong một mạng kết hợp các giá trị đưa vào nó thông qua các liên kết với các đơn vị khác, sinh ra một giá trị gọi là đầu vào mạng. Hàm thực hiện nhiệm vụ này gọi là hàm kết hợp, được định nghĩa bởi một luật lan truyền cụ thể. Trong phần lớn các mạng nơron, chúng ta giả sử rằng mỗi một đơn vị cung cấp một bộ cộng như là đầu vào cho đơn vị mà nó có liên kết. Tổng đầu vào đơn vị j đơn giản chỉ là tổng trọng số của các đầu ra riêng lẻ từ các đơn vị kết nối cộng thêm ngưỡng hay độ lệch θ_j :

$$a_j = \sum_{i=1}^{n} w_{ji} x_i + \theta j \qquad (6)$$

Trường hợp $w_{ji} > 0$, nơ-ron được coi là đang ở trong trạng thái kích thích. Tương tự, nếu như $w_{ji} < 0$, nơ-ron ở trạng thái kiềm chế. Chúng ta gọi các đơn vị với luật lan truyền như trên là các đơn vị sigma. Trong một vài trường hợp người ta cũng có thể sử dụng các luật lan truyền phức tạp hơn. Một trong số đó là luật sigma-pi, có dạng như sau:

$$a_j = \sum_{i=1}^{n} w_{ji} \prod_{k=1}^{m} x_{ik} + \theta j \quad (7)$$

Rất nhiều hàm kết hợp sử dụng một "độ lệch" hay "ngưỡng" để tính đầu vào mạng tới đơn vị. Đối với một đơn vị đầu ra tuyến tính, thông thường, θ_j được chọn là hằng số và trong bài toán xấp xỉ đa thức $\theta_j = 1$.

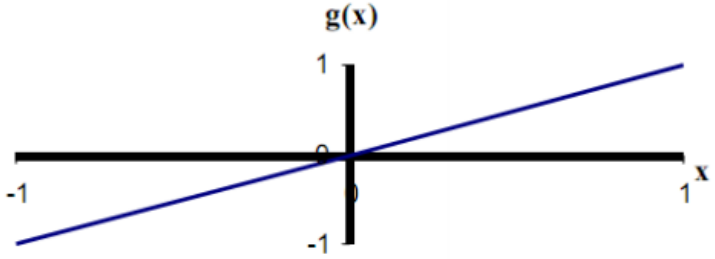
1.2.3 Hàm kích hoạt

Phần lớn các đơn vị trong mạng nơ-ron chuyển đầu vào mạng bằng cách sử dụng một hàm vô hướng gọi là hàm kích hoạt, kết quả của hàm này là một giá trị gọi là mức độ kích hoạt của đơn vị. Loại trừ khả năng đơn vị đó thuộc lớp ra, giá trị kích hoạt được đưa vào một hay nhiều đơn vị khác. Các hàm kích hoạt thường bị ép vào một khoảng giá trị xác định, do đó thường được gọi là các hàm bẹp . Các hàm kích hoạt hay được sử dụng là:

** Hàm đồng nhất*

$$g(x) = x \quad (8)$$

Nếu coi các đầu vào là một đơn vị thì chúng sẽ sử dụng hàm này. Đôi khi một hằng số được nhân với đầu vào mạng để tạo ra một hàm đồng nhất.



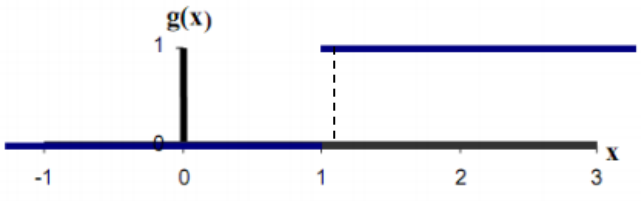
Hình 1.4 Hàm đồng nhất

** Hàm bước nhị phân*

Hàm này cũng được biết đến với tên "Hàm ngưỡng". Đầu ra của hàm này được giới hạn vào một trong hai giá trị:

$$g(x) = \begin{cases} 1, nếu\ (x \geq \theta) \\ 0, nếu\ (x < \theta) \end{cases}$$

Dạng hàm này được sử dụng trong các mạng chỉ có một lớp. (9)
Trong hình vẽ sau, θ được chọn bằng 1.

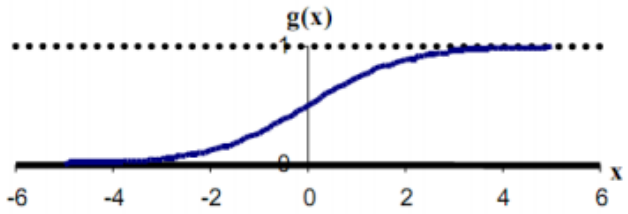


Hình 1.5 Hàm bước nhị phân

** Hàm sigmoid*

$$g(x) = \frac{1}{1 + e^{-x}} \tag{10}$$

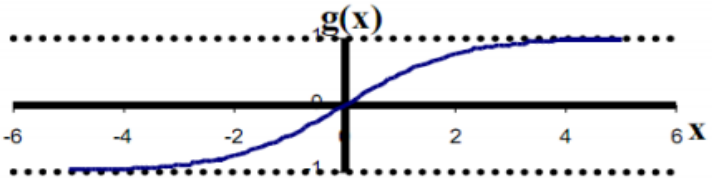
Hàm này đặc biệt thuận lợi khi sử dụng cho các mạng được huấn luyện (bởi thuật toán lan truyền ngược, bởi vì nó dễ lấy đạo hàm, do đó có thể giảm đáng kể tính toán trong quá trình huấn luyện. Hàm này được ứng dụng cho các chương trình ứng dụng mà các đầu ra mong muốn rơi vào khoảng [0,1].



Hình 1.6 Hàm Sigmoid

** Hàm sigmoid lưỡng cực*

$$g(x) = \frac{1-e^{-x}}{1+e^{-x}} \tag{11}$$



Hình 1.7 Hàm Sigmoid lưỡng cực

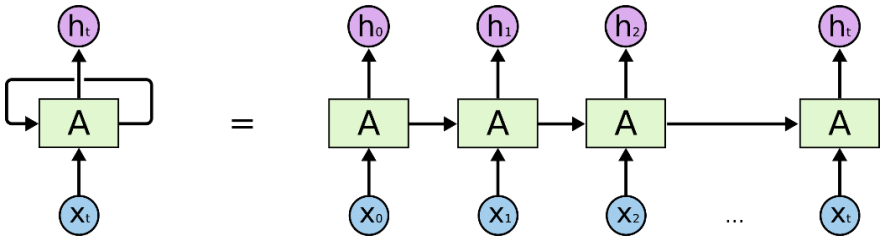
Hàm này có các thuộc tính tương tự hàm sigmoid. Nó làm việc tốt đối với các ứng dụng có đầu ra yêu cầu trong khoảng [-1,1].

1.3 Mạng nơ-ron hồi quy

Mạng nơ-ron hồi quy (Recurrent Neural Network - RNN) là một trong những mô hình học sâu được đánh giá có nhiều ưu điểm trong các tác vụ xử lý ngôn ngữ tự nhiên

1.3.1 Khái niệm RNN

Ý tưởng của RNN đó là thiết kế một mạng nơ-ron sao cho có khả năng xử lý được thông tin dạng chuỗi, ví dụ một câu là một chuỗi gồm nhiều từ.



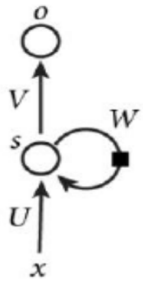
Hình 1.8 Mô hình mạng nơ-ron hồi quy

Mô hình của RNN được thể hiện ở hình 1.8, có nghĩa là thực hiện lặp lại cùng một tác vụ cho mỗi thành phần trong chuỗi. Trong đó, kết quả đầu ra tại thời điểm hiện tại phụ thuộc vào kết quả tính toán của các thành phần ở những thời điểm trước đó.

Nói cách khác, RNN là một mô hình có trí nhớ, có khả năng nhớ được thông tin đã tính toán trước đó. Không như các mô hình mạng nơ-ron truyền thống, đó là thông tin đầu vào hoàn toàn độc lập với thông tin đầu ra. Về lý thuyết, RNN có thể nhớ được thông tin của chuỗi có chiều dài bất kì, nhưng trong thực tế mô hình này chỉ nhớ được thông tin ở vài bước trước đó.

1.3.2 Quá trình xử lý thông tin của RNN

Quá trình xử lý thông tin trong mạng hồi qui được mô tả trong hình 1.9. Trong đó, x_t là đầu vào tại thời điểm thứ t, o_t là đầu ra tại thời điểm thứ t, S_t là trạng thái ẩn tại thời điểm thứ t, chính là "bộ nhớ" của mạng. S_t được tính dựa trên các trạng thái ẩn trước kết hợp với đầu vào tại thời điểm thứ t.



Hình 1.9 Quá trình xử lý thông tin trong RNNs

Quá trình này có thể được biểu diễn bằng mô hình toán sau (Mikolov *et al.*, 2014):

$$S_t = f(Ux_t + Ws_{t-1}) \quad (12)$$

với (*U, V, W*) là ba tham số của mạng.

Hàm *f* thường được sử dụng nhất là hàm tanh hoặc hàm RELU (Rojas Raúl, 2013)

Với khả năng "nhớ" được, đặc điểm chung của mạng nơ-ron hồi quy là có khả năng xử lý thông tin dạng chuỗi và các dữ liệu thời gian.

Về lý thuyết, RNNs có thể nhớ được thông tin của chuỗi có chiều dài bất kỳ nhưng trong thực tế thì mô hình này chỉ có khả năng nhớ được thông tin ở một vài bước trước đó (Schmidhuber and Hochreiter, 1997)

1.3.3 Các ứng dụng của RNN

- **Mô hình ngôn ngữ và phát sinh văn bản**

Mô hình ngôn ngữ cho ta biết xác suất của một câu trong một ngôn ngữ là bao nhiêu (ví dụ xác suất ***p("hôm qua là thứ năm") = 0.001; p("năm thứ hôm là qua") = 0***). Đây cũng là bài toán dự đoán xác suất từ tiếp theo của một câu cho trước là bao nhiêu.

Từ bài toán này, chúng ta có thể mở rộng thành bài toán phát sinh văn bản. Mô hình này cho phép ta phát sinh ra văn bản mới dựa vào tập dữ liệu huấn luyện. Ví dụ, khi huấn luyện mô hình này bằng các văn bản truyện Kiều, ta có thể phát sinh được các đoạn văn tựa truyện Kiều. Tuỳ theo loại dữ liệu huấn luyện, ta sẽ có nhiều loại ứng dụng khác nhau.

- **Dịch máy**

Bài toán dịch máy (Machine Translation) [11, 12] tương tự như mô hình ngôn ngữ. Trong đó, đầu vào là chuỗi các từ của ngôn ngữ nguồn (ví dụ tiếng Đức), đầu ra là chuỗi các từ của ngôn ngữ đích (ví dụ tiếng Anh). Điểm khác biệt ở đây đó là đầu ra chỉ có thể dự đoán được khi đầu vào đã hoàn toàn được phân tích. Điều này là do từ được dịch ra phải có đầy đủ thông tin của các từ trước đó.

- **Phát sinh mô tả cho ảnh (Generating Image Descriptions)**

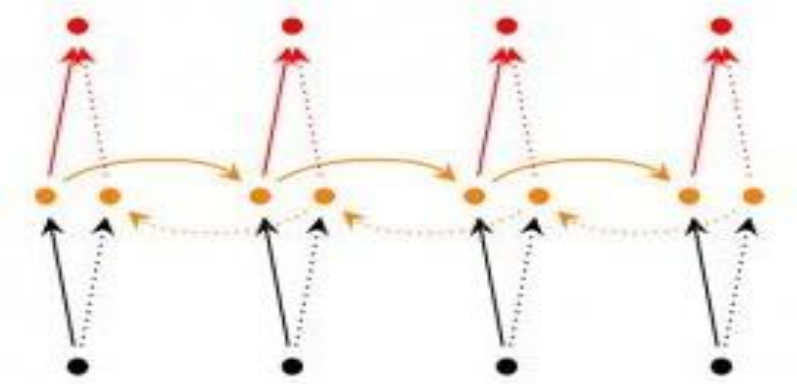
RNN kết hợp với Convolution Neural Networks [13] có thể phát sinh ra được các đoạn mô tả cho ảnh. Mô hình này hoạt động bằng cách tạo ra những câu mô tả từ các đặc trưng rút trích được trong bức ảnh.

1.3.4 Các phiên bản mở rộng của RNN

Trong vài năm qua, các nhà nghiên cứu đã phát triển nhiều loại mạng RNNs ngày càng tinh vi để giải quyết các mặt hạn chế của RNN. Dưới đây, là một số phiên bản mở rộng của RNN.

- **RNN hai chiều:**

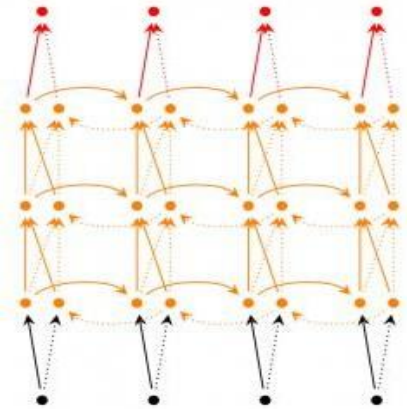
Dựa trên ý tưởng đầu ra tại thời điểm t không chỉ phụ thuộc vào các thành phần trước đó mà còn phụ thuộc vào các thành phần trong tương lai. Ví dụ, để dự đoán một từ bị thiếu trong chuỗi, ta cần quan sát các từ bên trái và bên phải xung quanh từ đó. Mô hình này chỉ gồm hai RNNs nạp chồng lên nhau. Trong đó, các trạng thái ẩn được tính toán dựa trên cả hai thành phần bên trái và bên phải của mạng.



Hình 1.10 Mạng RNN hai chiều.

- **RNN hai chiều sâu**

Tương tự như RNN hai chiều, điểm khác biệt đó là mô hình này gồm nhiều tầng RNN hai chiều tại mỗi thời điểm. Mô hình này sẽ cho ta khả năng thực hiện các tính toán nâng cao nhưng đòi hỏi tập huấn luyện của chúng ta phải đủ lớn.



Hình 1.11 Mạng RNN nhiều tầng.

- **Mạng bộ nhớ ngắn hạn (LSTM)**

Mô hình này có cấu trúc tương tự như RNNs nhưng có cách tính toán khác đối với các trạng thái ẩn. Bộ nhớ trong LSTMs được gọi là hạt nhân. Ta có thể xem đây là một hộp đen nhận thông tin đầu vào gồm trạng thái ẩn và giá trị. Bên trong các hạt nhân này, chúng sẽ quyết định thông tin nào cần lưu lại và thông tin nào cần xóa đi, nhờ vậy mà mô hình này có thể lưu trữ được thông tin xa.

1.4 Kết luận chương 1

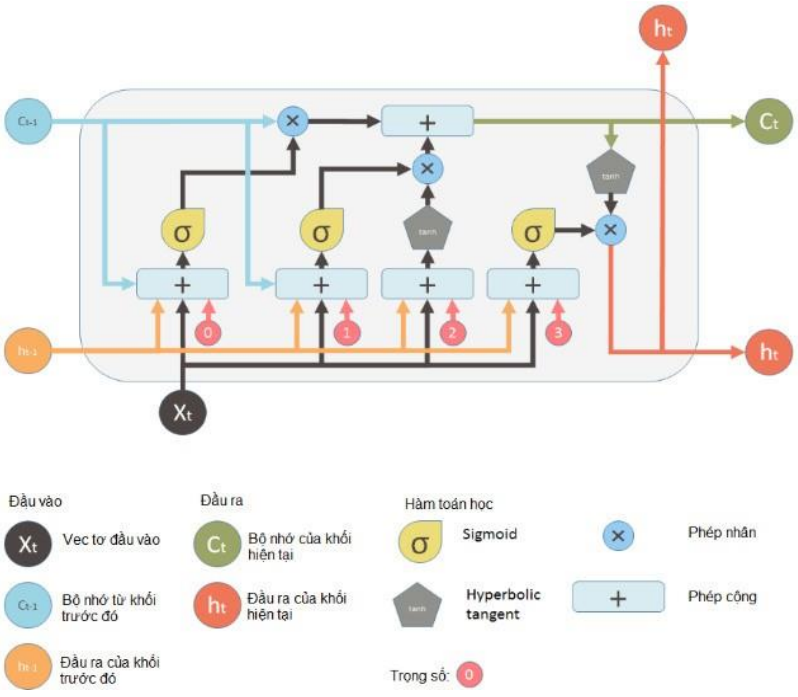
LSTM là một bước tiến lớn trong việc sử dụng RNN. Ý tưởng của nó giúp cho tất cả các bước của RNN có thể truy vấn được thông tin từ một tập thông tin lớn hơn, nó giúp giải quyết vấn đề dự đoán chuỗi thời gian. Cho nên trong đồ án này chúng tôi tập trung nghiên cứu cho bài toán dự đoán hành khách lượng hành khách đi máy bay quốc tế. Chi tiết mô hình mạng này được giới thiệu trong Chương 2.

CHƯƠNG 2
ỨNG DỤNG CÔNG NGHỆ LSTM CHO VIỆC DỰ ĐOÁN LƯỢNG HÀNH KHÁCH ĐI MÁY BAY QUỐC TẾ

Chương này sẽ giới thiệu về bài toán ước lượng hành khách đi máy bay quốc tế và về mạng cải tiến LSTM: Kiến trúc, mô hình, quy trình hoạt động. Đây cũng là cơ sở để xây dựng thực nghiệm trong Chương 3.

2.1 Kiến trúc mạng LSTM

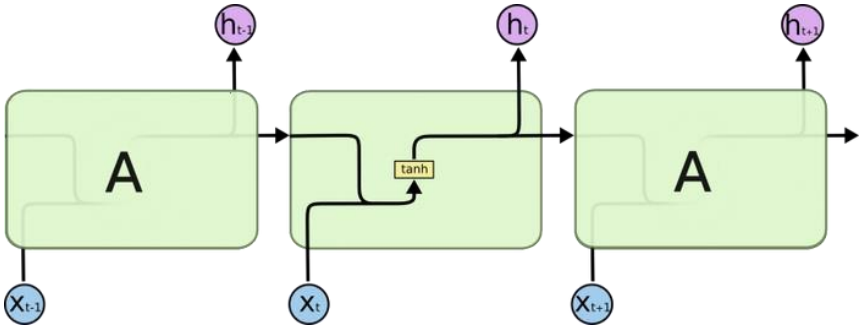
Mạng bộ nhớ ngắn hạn (LSTM) là một kiến trúc học sâu được coi như là một sự mở rộng của mạng nơ-ron hồi qui (RNN) với khả năng học các phụ thuộc dài hạn (Schmidhuber and Hochreiter, 1997; Mikolov *et al.*, 2014; Rojas, 2013). Đây là một mô hình mạng có trí nhớ, có khả năng "nhớ" được các thông tin đã tính toán trước đó. Kết quả tại thời điểm hiện tại không những phụ thuộc vào đầu vào tại thời điểm hiện tại mà còn phụ thuộc vào kết quả tính toán của các thành phần ở những thời điểm trước.



Hình 2.1 Cấu trúc của mô hình LSTM

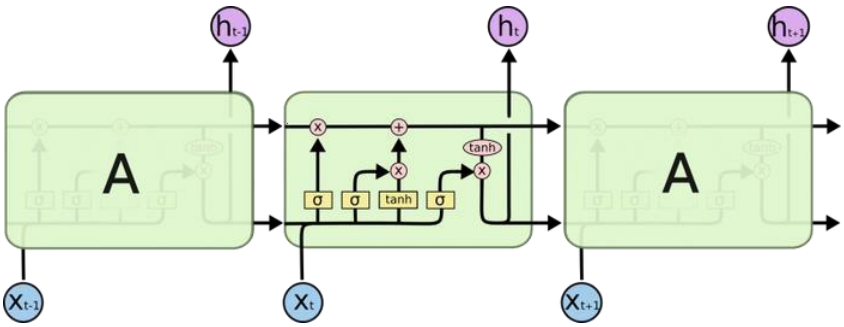
Trong hình 2.2, cấu trúc mạng LSTM gồm có 4 tầng và tương tác với nhau một cách đặc biệt. Cốt lõi của mạng LSTM bao gồm trạng thái nhớ và cổng. Trạng thái tế bào giống như băng chuyền, chạy xuyên suốt qua tất cả các nút mạng giúp thông tin được truyền đạt dễ dàng, còn cổng là nơi sàng lọc thông tin đi qua nó, chúng được kết hợp bởi một tầng mạng sigmoid.

Mạng LSTM được thiết kế nhằm loại bỏ vấn đề phụ thuộc dài hạn [10]. Ví dụ mô hình RNN ở hình 2.2, các lớp đều mắc nối với nhau thành các mô-đun mạng nơ-ron. Trong RNN chuẩn, mô-đun lặp lại này có cấu trúc rất đơn giản chỉ gồm một lớp đơn giản là lớp tanh.



Hình 2.2 Các mô-đun lặp của mạng RNN chứa một lớp

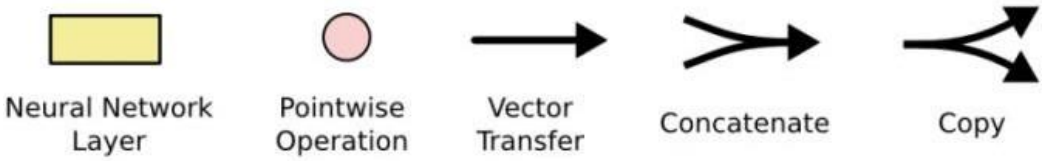
Mạng LSTM có cấu trúc mắt xích tương tự, nhưng các mô-đun lặp có cấu trúc khác hẳn. Để tăng khả năng nhớ thì bước xử lý lặp của LSTM sử dụng 4 lớp thay vì 1 lớp như RNN. Hình 2. mô tả sự khác nhau giữa cấu trúc 1 hạt nhân (mô-đun lặp) trong RNN và LSTM. Mấu chốt của khả năng "nhớ lâu" của LSTM là cấu trúc "trạng thái nhớ", là đường kẻ ngang phía trên trong mô-đun lặp. Các thông tin có thể được thêm hoặc bớt vào trạng thái hạt nhân, dựa trên qui định của các cổng, là các phép toán được đặt trong vòng tròn bên trong trạng thái hạt nhân.



Hình 2.3 Các mô-đun lặp của mạng LSTM chứa bốn lớp

Trong đó, các ký hiệu sử dụng trong mạng LSTM được giải nghĩa sau đây:

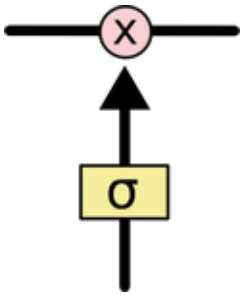
- Hình chữ nhật nền vàng là các lớp ẩn của mạng nơ-ron
- Hình tròn nền hồng biểu diễn toán tử theo từng điểm
- Đường kẻ gộp lại với nhau biểu thị phép nối các toán hạng
- Và đường rẽ nhánh biểu thị cho sự sao chép từ vị trí này sang vị trí khác



Hình 2.4 Các kí hiệu sử dụng trong mạng LSTM

2.2 Quá trình xử lý thông tin của LSTM

Mạng LSTM có khả năng thêm hoặc bớt thông tin vào trạng thái hạt nhân, được quy định một cách cẩn thận bởi các cấu trúc gọi là cổng. Các cổng này là một cách (tuỳ chọn) để định nghĩa thông tin băng qua. Chúng được tạo bởi hàm sigmoid và một toán tử nhân theo từng điểm.



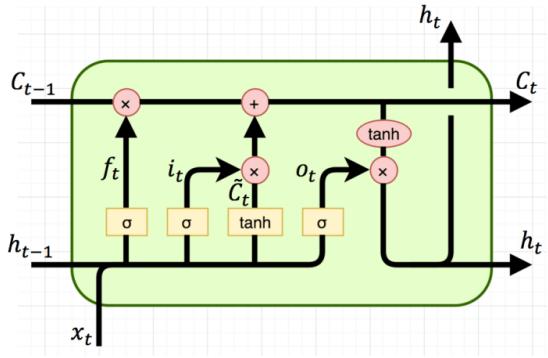
Hình 2.5 Cổng trạng thái LSTM.

Hàm kích hoạt Sigmoid có giá trị từ 0 – 1, mô tả độ lớn thông tin được phép truyền qua tại mỗi lớp mạng. Nếu ta thu được 0, điều này có nghĩa là "không cho bất kỳ cái gì đi qua", ngược lại nếu thu được giá trị là 1, thì có nghĩa là "cho phép mọi thứ đi qua".

Một mạng LSTM gồm có 3 cổng để duy trì hoạt động trạng thái của hạt nhân.

Bước đầu tiên của mô hình LSTM là quyết định xem thông tin nào chúng tôi cần loại bỏ khỏi trạng thái hạt nhân. Tiến trình này được thực hiện thông qua một lớp sigmoid gọi là "cổng chặn". Đầu vào là h_{t-1} và x_t, đầu ra là một giá trị nằm trong

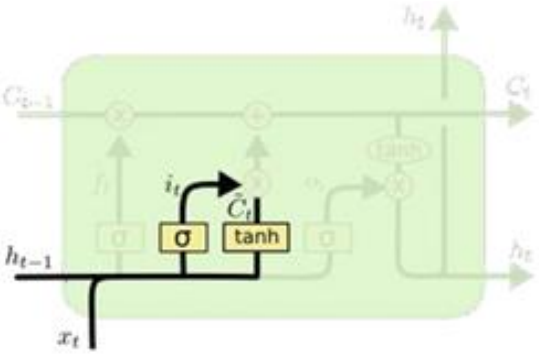
khoảng [0, 1] cho trạng thái hạt nhân C_{t-1}. 1 tương đương với "giữ lại thông tin", 0 tương đương với "loại bỏ thông tin".



$$f_t = \sigma(W_f.[h_{t-1}, x_t] + b_f) \quad (12)$$

Hình 2.6 Bước thứ 1 quy trình xử lý của LSTM

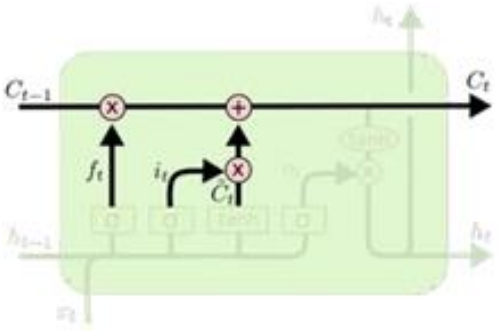
Bước tiếp theo, chúng tôi cần quyết định thông tin nào cần được lưu lại tại trạng thái hạt nhân. Chúng tôi có hai phần. Một, lớp sigmoid đơn được gọi là "cổng chặn" quyết định các giá trị chúng tôi sẽ cập nhật. Tiếp theo, một lớp tanh tạo ra một vector ứng viên mới $\widetilde{C}_t$ được thêm vào trong ô trạng thái.



$$i_t = \sigma(W_i.[h_{t-1}, x_t] + b_i)$$
$$\widetilde{C}_t = \tanh(W_C.[h_{t-1}, x_t] + b_c) \quad (13)$$

Hình 2.7 Bước thứ 2 quy trình xử lý của LSTM

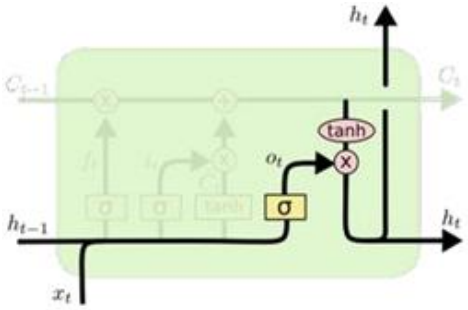
Tiếp theo, chúng tôi sẽ kết hợp hai thành phần này lại để cập nhật vào trạng thái hạt nhân. Đây là giá trị ứng viên mới, tỉ lệ số lượng giá trị mà chúng tôi muốn cập nhật f_t cho mỗi trạng thái.



$$C_t = f_t * C_{t-1} + i_t * \widetilde{C}_t \quad (14)$$

Hình 2.8 Bước thứ 3 quy trình xử lý của LSTM

Cuối cùng, chúng tôi cần quyết định xem thông tin đầu ra là gì. Đầu ra này cần dựa trên trạng thái hạt nhân, nhưng sẽ được lọc bớt thông tin. Đầu tiên, áp dụng lớp sigmoid đơn để quyết định xem phần nào của trạng thái hạt nhân sẽ ra đầu ra. Sau đó, ta sẽ đẩy trạng thái hạt nhân qua (đẩy giá trị vào khoảng -1 và 1) và nhân với một cổng sigmoid đầu ra, để giữ lại những phần ta muốn ra ngoài.



$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$
$$h_t = o_t * \tanh(C_t) \quad (15)$$

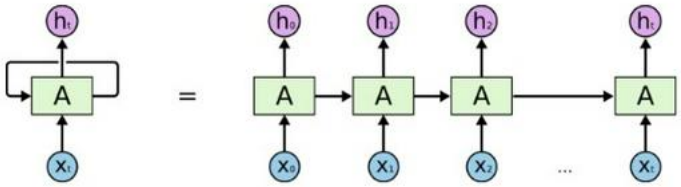
Hình 2.9 Bước cuối cùng quy trình xử lý của LSTM

2.3 Các kỹ thuật LSTM sử dụng trong thử nghiệm

2.3.1 LSTM hồi quy

Bài toán ước lượng hành khách có thể coi như một bài toán hồi quy. Hay nói cách khác, với số lượng hành khách (tính theo đơn vị hàng ngàn) trong tháng này, số lượng hành khách trong tháng tới là bao nhiêu?

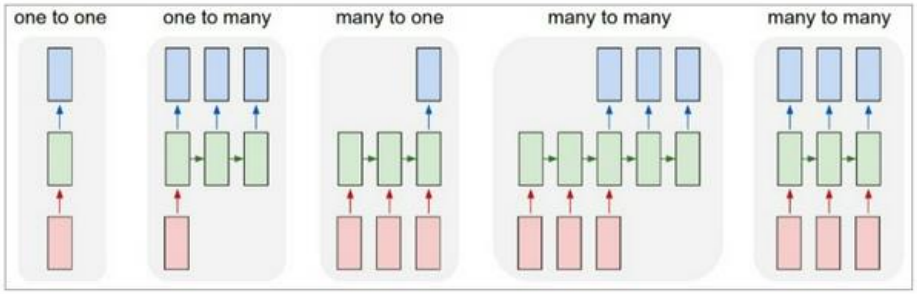
Thông thường dữ liệu của bài toán là tập dữ liệu hai cột: cột đầu tiên chứa số lượng hành khách (t) của tháng này và cột thứ hai chứa số lượng hành khách của tháng tới (t + 1), được dự đoán.



Hình 2.10 Mạng nơ-ron hồi quy

Trong hình 2.10, A là mạng nơ-ron hồi quy. Nó nhận một đầu vào x_t , tiến hành xử lý và đưa ra đầu ra h_t . Điểm đặc biệt của A là nó sẽ lưu lại giá trị của h_t để sử dụng cho đầu vào tiếp theo. Có thể coi một mạng neural hồi quy là một chuỗi những mạng con giống hệt nhau, mỗi mạng sẽ truyền thông tin nó vừa xử lý cho mạng phía sau nó.

Chuỗi các đầu vào x_0**,** x_1**, ... ,** x_t là những sự kiện xảy ra theo thứ tự thời gian. Những sự kiện này đều có mối liên hệ về thông tin với nhau và thông tin của chúng sẽ được giữ lại để xử lý sự kiện tiếp theo trong mạng neural hồi quy. Vì tính chất này, mạng nơ-ron hồi quy phù hợp cho những bài toán với dữ liệu đầu vào dưới dạng chuỗi với các sự kiện trong chuỗi có mối liên hệ với nhau. Vì vậy, mạng nơ-ron hồi quy có ứng dụng quan trọng trong các bài toán xử lý ngôn ngữ tự nhiên như: Dịch máy, Phân loại ngữ nghĩa, Nhận dạng giọng nói... Một trong những điểm mạnh của mạng nơ-ron hồi quy là cho phép tính toán trên một chuỗi các vector.



Hình 2.11 Hoạt động của mạng nơ-ron hồi quy

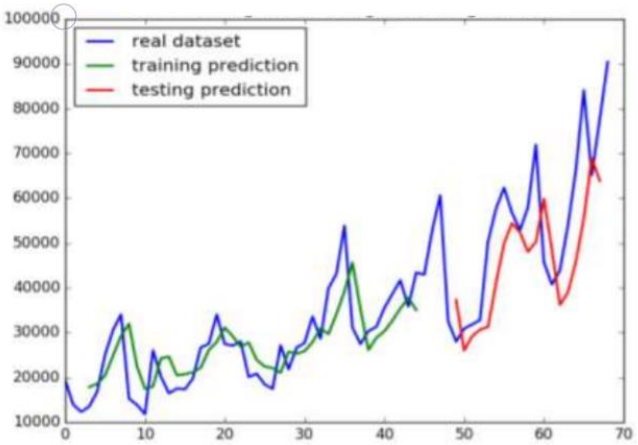
Kiểu hoạt động của mạng nơ-ron hồi quy được thể hiện ở hình 2.11. Mỗi hình chữ nhật là 1 vector và các mũi tên thể hiện các hàm biến đổi. Vector đầu vào có màu đỏ, vector đầu ra có màu xanh biển và vector trạng thái thông tin trao đổi giữa các mạng con có màu xanh lá.

Mạng nơ-ron hồi quy nhận một vector đầu vào x và đưa ra vector đầu ra y. Để có thể lưu trữ được thông của các sự kiện trong quá khứ, mạng neural hồi quy lưu trữ trong chính nó một vector trạng thái ẩn h. Vector trạng thái này sẽ lưu giữ những thông tin của những sự kiện đã được xử lý bằng cách cập nhật lại giá trị mỗi khi một sự kiện mới được xử lý.

Kiến trúc này có một nhược điểm: Nếu kích thước chuỗi vector đầu vào x là rất lớn thì việc tính toán véc-tơ trạng thái ẩn h sẽ phải đi qua nhiều lớp tính toán. Trong quá trình, để cập nhật các trọng số, vì việc đạo hàm phải đi qua nhiều lớp tính toán của vector h nên các giá trị cập nhật cũng sẽ lớn dần lên, việc cập nhật các trọng số không theo ý muốn và khiến mạng không ổn định. Vì vậy, các biến thể nâng cấp của mạng nơ-ron hồi quy đã ra đời để khắc phục vấn đề này. LSTM đã bổ sung thêm cơ chế loại bỏ những thông tin không cần thiết ra khỏi vector trạng thái ẩn h, từ đó đã khắc phục được một phần nhược điểm trên.

2.3.2 LSTM hồi quy sử dụng phương thức cửa sổ

Một mô hình dự đoán có nhiều biến số một lần để dự đoán bước tiếp theo là phương thức cửa sổ. Ví dụ: giá trị tại t và giá trị tại t + 1 được sử dụng để dự đoán giá trị tại thời điểm t + 2, có thể được phát triển bằng cách sử dụng thời gian hiện tại t và các lần trước t-1 làm biến đầu vào để dự đoán t + 1. Khi được tạo thành mô hình hồi quy, biến đầu vào là t-1 và t và biến đầu ra là t + 1.



Hình 2.12 LSTM sử dụng phương thức cửa sổ

Các kết quả dự đoán được đưa ra được hiển thị trong hình 2.12. Đường màu xanh trong hình 2.12, cho thấy một biểu đồ của thực tế dữ liệu. Đường màu xanh lá cây hiển thị biểu đồ dự đoán trong quá trình đào tạo. Và đường màu đỏ hiển thị kết quả dự đoán trong quá trình thử nghiệm.

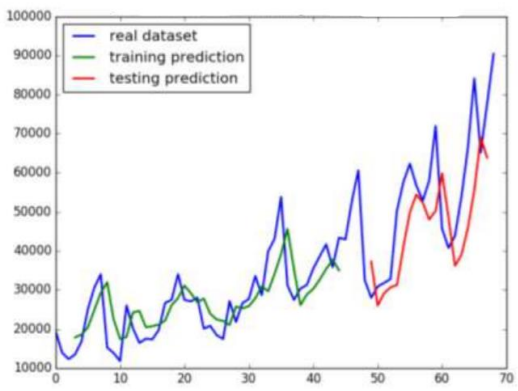
2.3.3 LSTM hồi quy sử dụng bước thời gian

Phương pháp này áp dụng phép chiếu lăn hoặc nó được gọi là mô hình xác nhận đi bộ. Mỗi bước thời gian của bộ dữ liệu thử nghiệm sẽ chạy cùng một lúc. Mô hình được sử dụng để ước tính một lần trước. Trên thực tế, nó giống như dự đoán thông thường. Quan sát các chuyến thăm của khách du lịch sẽ có sẵn mỗi tháng và sẽ được sử dụng cho các bước tính của tháng tiếp theo.

Các quan sát được thực hiện ở ba bước trước (t-1, t-2, t-3) và được sử dụng làm đầu vào để dự đoán quan sát tại thời điểm hiện tại (t). Dự đoán sử dụng bước đa thời gian theo khái niệm trình tự để trình tự với một chuỗi dài hơn. Trình tự được sử dụng trong nghiên cứu này được thực hiện như sau:

January	February	March	→	April
February	March	April	→	May
March	April	May	→	June
April	May	June	→	July
...	...	...	...	...
Month t-2	Month t-1	Month t	→	Month t+1

Hình 2.13 Trình tự bước thời gian



Hình 2.14 LSTM sử dụng bộ nhớ giữa các bước

Các kết quả dự đoán được đưa ra được hiển thị trong hình 2.14. Đường màu xanh trong hình 2.14 cho thấy một biểu đồ của dữ liệu thực tế. Đường màu xanh lá cây hiển thị biểu đồ dự đoán trong quá trình đào tạo. Và đường màu đỏ hiển thị kết quả dự đoán trong quá trình thử nghiệm. Trước quá trình đào tạo, một chuyển đổi dữ liệu được thực hiện ra để có được quy mô theo yêu cầu kích hoạt. Trong nghiên cứu này, dữ liệu bán lại đã được thực hiện tại một giá trị từ -1 đến 1 để hoàn thành chức năng kích hoạt của tiếp tuyến hyperbol tiếp tuyến của mô hình LSTM. Việc chuyển đổi này sẽ được đưa ra một quy trình đảo ngược sau quá trình dự đoán thành trả lại giá trị cho quy mô ban đầu.

2.3.4 LSTM sử dụng bộ nhớ giữa các bước

Việc chuẩn bị dữ liệu cho mạng LSTM bao gồm các bước thời gian. Một số vấn đề trình tự có thể có số bước thời gian khác nhau trên mỗi mẫu. Ví dụ: có thể có các phép đo của một máy vật lý dẫn đến một điểm thất bại hoặc một điểm đột biến. Mỗi sự cố sẽ là một mẫu, các quan sát dẫn đến sự kiện sẽ là các bước thời gian và các biến được quan sát sẽ là các tính năng. Các bước thời gian cung cấp một cách khác để diễn đạt vấn đề chuỗi thời gian của chúng tôi. Giống như ở bên trong ví dụ của phương thức LSTM cửa sổ, chúng tôi có thể thực hiện các bước thời gian trước trong chuỗi thời gian của mình làm đầu vào để dự đoán đầu ra ở bước thời gian tiếp theo.

Thay vì đưa ra các quan sát trong quá khứ như các tính năng đầu vào riêng biệt, chúng ta có thể sử dụng chúng như các bước thời gian của một tính năng đầu vào, đây thực sự là một khung chính xác hơn của vấn đề.

Mạng LSTM có bộ nhớ có khả năng ghi nhớ trong các chuỗi dài. Thông thường, trạng thái trong mạng được đặt lại sau mỗi đợt đào tạo khi khớp mô hình, cũng như mỗi lệnh gọi dự đoán hoặc đánh giá. Chúng ta có thể giành quyền kiểm soát tốt hơn khi trạng thái bên trong của mạng LSTM bị xóa trong Keras bằng cách làm cho lớp LSTM có trạng thái. Điều này có nghĩa là nó có thể xây dựng trạng thái trên toàn bộ chuỗi đào tạo và thậm chí duy trì trạng thái đó nếu cần để đưa ra dự đoán. Nó đòi hỏi dữ liệu đào tạo không được xáo trộn khi lắp mạng. Nó cũng yêu cầu thiết lập

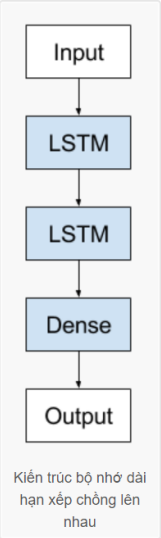
lại rõ ràng trạng thái mạng sau mỗi lần tiếp xúc với dữ liệu huấn luyện bằng cách gọi đến trạng thái đặt lại.

2.3.5 LSTM xếp chồng sử dụng bộ nhớ giữa các bước

Kiến trúc LSTM xếp chồng là những lợi ích tương tự có thể được khai thác với LSTM. Các mạng LSTM hoạt động trên dữ liệu chuỗi, điều đó có nghĩa là việc thêm các lớp sẽ thêm các mức độ trừu tượng của các quan sát đầu vào theo thời gian. Trong thực tế, quan sát khúc dữ liệu theo thời gian hoặc đại diện cho vấn đề ở quy mô thời gian khác nhau. Sự kiện xây dựng một RNN sâu bằng cách xếp chồng nhiều trạng thái ẩn lặp lại lên nhau. Cách tiếp cận này có khả năng cho phép trạng thái ẩn ở mỗi cấp hoạt động ở các khoảng thời gian khác nhau.

Các LSTM xếp chồng hoặc LSTM sâu được giới thiệu bởi Graves, trong việc áp dụng các LSTM của họ để nhận dạng giọng nói, đánh bại điểm chuẩn về một vấn đề tiêu chuẩn đầy thách thức. RNN vốn đã có thời gian sâu, vì trạng thái ẩn của chúng là một chức năng của tất cả các trạng thái ẩn trước đó. Câu hỏi truyền cảm hứng này là liệu RNN cũng có thể hưởng lợi từ độ sâu trong không gian; đó là từ việc xếp chồng nhiều lớp ẩn lặp lại lên nhau, giống như các lớp cấp trước được xếp chồng lên nhau trong các mạng sâu thông thường.

Trong cùng một công việc, họ thấy rằng độ sâu của mạng quan trọng hơn số lượng ô nhớ trong một lớp nhất định để mô hình hóa kỹ năng. Các LSTM xếp chồng hiện là một kỹ thuật ổn định cho các vấn đề dự đoán trình tự đầy thách thức. Một kiến trúc LSTM xếp chồng có thể được định nghĩa là một mô hình LSTM bao gồm nhiều lớp LSTM. Một lớp LSTM ở trên cung cấp một đầu ra chuỗi thay vì một đầu ra giá trị duy nhất cho lớp LSTM bên dưới. Cụ thể, một bước đầu ra cho mỗi bước thời gian đầu vào, thay vì một bước thời gian đầu ra cho tất cả các bước thời gian đầu vào.



Hình 2.15 LSTM xếp chồng sử dụng bộ nhớ giữa các bước

2.4 Nghiên cứu vấn đề dự báo chuỗi thời gian, lượng hành khách đi máy bay quốc tế

2.4.1 Phân tích yêu cầu

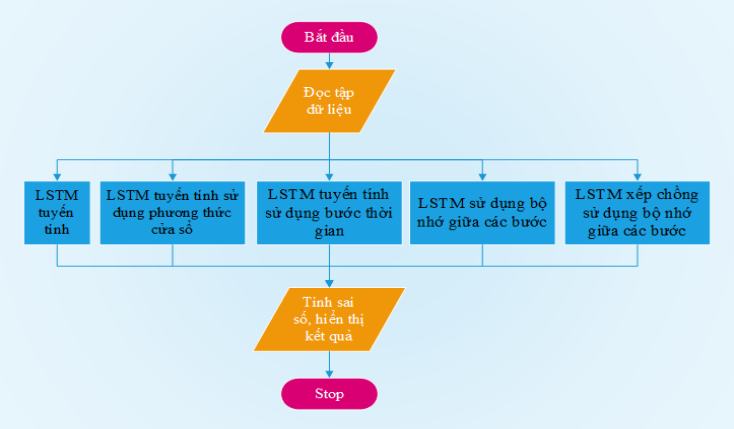
Lượng hành khách đi máy bay quốc tế không chỉ phụ thuộc vào nhu cầu đi lại thực tế của hành khách mà còn chịu ảnh hưởng bởi nhiều yếu tố khác như tình hình kinh tế, thời tiết, các dịp nghỉ lễ, các sự kiện diễn ra trong năm... Hệ thống dự báo lượng hành khách đi máy bay quốc tế được xây dựng nhằm giúp hãng hàng không chuẩn bị đáp ứng với lưu lượng hành khách trong tương lai

Có nhiều phương pháp có thể áp dụng để dự đoán lượng hành khách đi máy bay quốc tế tuy nhiên các yếu tố chi phối như tình hình kinh tế, thời tiết…đều là những yếu tố biến động lớn, khó định lượng do đó sai số dự đoán thường khá lớn.

Do đó luận văn áp dụng mô hình mạng LSTM được sử dụng để dự đoán lượng hành khách đi máy bay quốc tế dựa trên dữ liệu của tập thông tin lượng hành khách của hãng hàng không ***American Airlines*** từ 07/2005 đến 03/2016 mà không cần hoặc có thêm các thông tin như tình hình kinh tế, thời tiết, các dịp nghỉ lễ, các sự kiện diễn ra trong năm.

Năm kịch bản sử dụng mô hình mạng LSTM sẽ được thực nghiệm để tìm ra phương pháp cho sai số tối ưu.

2.4.2 Mô hình thử nghiệm



Hình 2.16 Mô hình thực nghiệm hệ thống dự đoán lượng hành khách đi máy bay

Mô hình thực nghiệm hệ thống dự đoán lượng hành khách đi máy bay được thể hiện trong hình 2.16. Hệ thống chia làm 5 mô-đun với các kỹ thuật LSTM:

- LSTM hồi quy

  Với số lượng hành khách (tính theo đơn vị hàng ngàn) trong tháng này, số lượng hành khách trong tháng tới là bao nhiêu?

- LSTM hồi quy sử dụng phương thức cửa sổ

  Kỹ thuật này đó là sử dụng nhiều bước thời gian gần đây để đưa ra dự đoán cho bước tiếp theo. Đây được gọi là cửa sổ và kích thước của cửa sổ là một tham số có thể được điều chỉnh cho từng vấn đề.

  Ví dụ, với thời gian hiện tại (t), chúng tôi muốn dự đoán giá trị ở lần tiếp theo trong chuỗi (t+1), chúng tôi có thể sử dụng thời gian hiện tại (t), cũng như hai lần trước (t-1 và t-2) là các biến đầu vào.

- LSTM hồi quy sử dụng bước thời gian

  Các bước thời gian cung cấp một cách khác để diễn đạt vấn đề chuỗi thời gian. Giống như phương thức cửa sổ, chúng tôi có thể thực hiện các bước thời gian trước trong chuỗi thời gian của mình làm đầu vào để dự đoán đầu ra ở các bước tiếp theo. Thay vì diễn đạt các quan sát trong quá khứ như các tính năng đầu vào riêng biệt, chúng tôi có thể sử dụng chúng như các bước thời gian của một tính năng đầu vào.

- LSTM sử dụng bộ nhớ giữa các bước

  Mạng LSTM có bộ nhớ, có khả năng ghi nhớ trong các chuỗi dài. Thông thường, trạng thái trong mạng được đặt lại sau mỗi đợt đào tạo khi khớp mô hình. Chúng tôi có thể xây dựng trạng thái trên toàn bộ chuỗi đào tạo và duy trì trạng thái đó nếu cần để đưa ra dự đoán.

- LSTM xếp chồng sử dụng bộ nhớ giữa các bước

  Một trong những lợi ích to lớn của LSTM là có thể được được đào tạo thành công khi xếp chồng vào các kiến trúc mạng sâu. Các mạng LSTM có thể được xếp chồng lên nhau. Một bổ sung cho cấu hình được yêu cầu là một lớp LSTM trước mỗi lớp LSTM tiếp theo phải trả về chuỗi.

Dữ liệu CSV đầu vào được đọc và chuyển thành tập dữ liệu cho các mô hình LSTM. Sau khi thực hiện dự đoán, hệ thống sẽ tính toán sai số và hiển thị đồ thị kết quả để so sánh độ lệch sai số.

2.3.3 Các bước xử lý

Hình 2.17 Các bước xử lý của mô-đun dự đoán lượng hành khách đi máy bay

Theo hình 2.17, ở bước đầu tiên ta sẽ tiến hành nhập thư viện cho module dự đoán. Tiếp theo, ta đọc tập dữ liệu trong thư viện vừa nhập có định dạng ".csv". Dữ liệu sau đó sẽ được chuyển đổi sang tập dữ liệu dạng ma trận. Ma trận dữ liệu sau khi chuẩn hóa được chuyển đổi thành tập huấn luyện và tập kiểm thử. Tiếp theo đó, ta tiến hành tạo mô hình mạng LSTM phục vụ công viện dự đoán. Bước cuối cùng sẽ là tính toán sai số và hiển thị kết quả.

2.5 Cài đặt ứng dụng

Thực nghiệm được tiến hành trên 1 máy tính với cấu hình như sau:

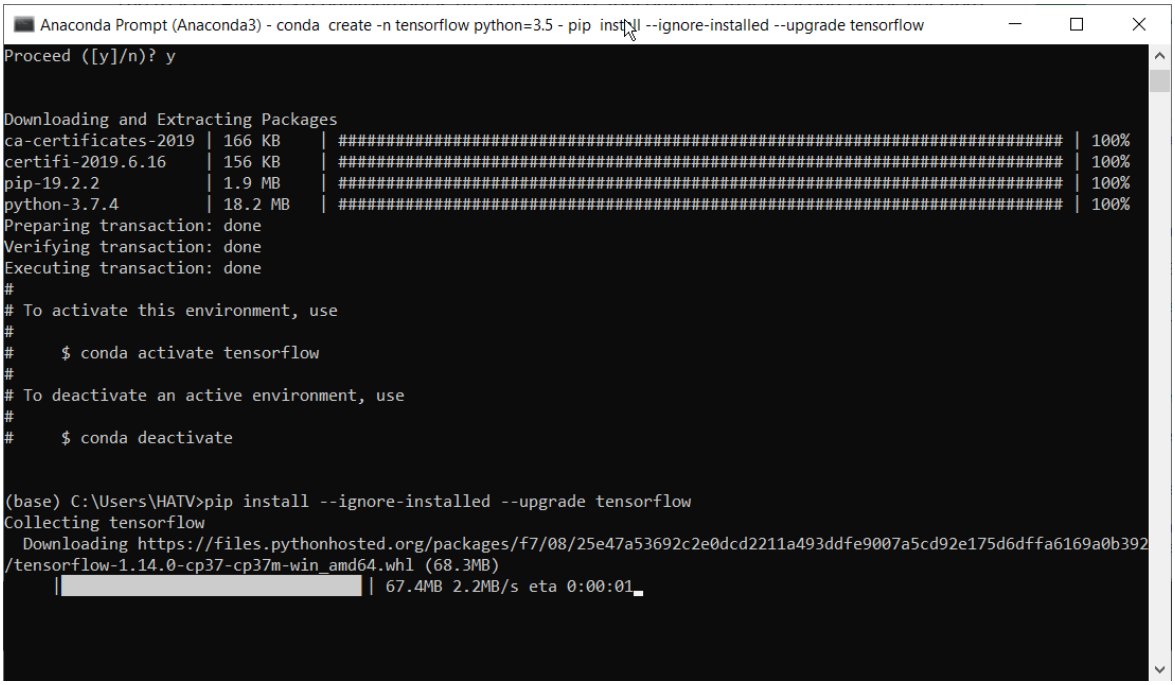
- CPU Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz

- RAM 2 GB

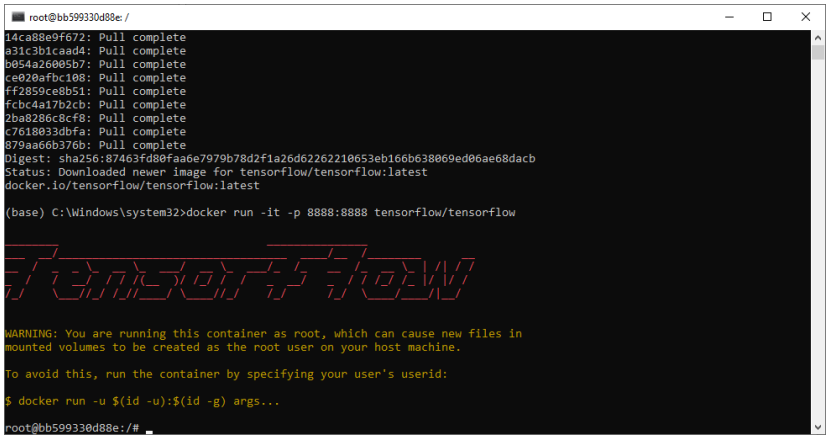
- OS Windows 10

- Ngôn ngữ lập trình Python

Các thư viện và phần mềm hỗ trợ học sâu được sử dụng trong thực nghiệm là Anaconda, Keras, Tensoflow và PyCharm.



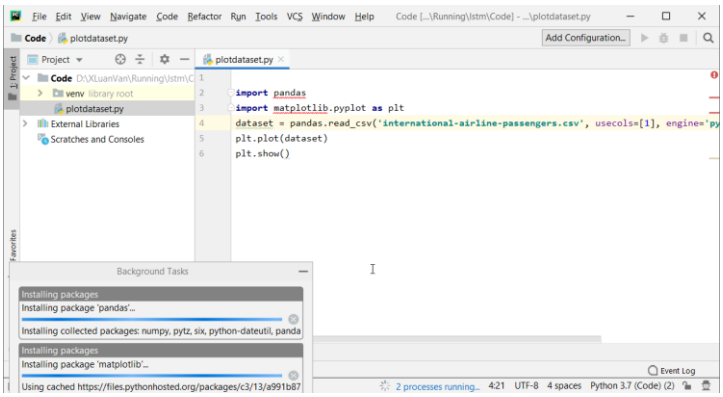
Hình 2.18 Cài đặt Tensorflow



Hình 2.19 Môi trường phát triển Tensorflow

TensorFlow ™ là một hệ thống học máy hoạt động ở quy mô lớn và trong môi trường phức tạp. TensorFlow [14, 15] sử dụng đồ thị luồng dữ liệu để đại diện cho sự tính toán, chia sẻ trạng thái, và các hoạt động biến đổi trạng thái đó. Nó ánh xạ các nút của một đồ thị luồng dữ liệu trên nhiều máy trong một cụm, và bên trong một máy trên nhiều thiết bị tính toán, bao gồm CPU, GPU đa lõi, các chíp ASIC tùy biến được gọi là đơn vị xử lý ten-xơ (TPUs). Kiến trúc này rất linh hoạt cho phép cho các nhà phát triển ứng dụng: trong khi trước đây "tham số máy chủ" thiết kế quản lý trạng thái chia sẻ được xây dựng sẵn trên hệ thống, TensorFlow cho phép các nhà phát triển để thử nghiệm các tối ưu hoá mới và các thuật toán huấn luyện.

TensorFlow hỗ trợ một loạt các ứng dụng, với sự hỗ trợ đặc biệt mạnh mẽ cho việc huấn luyện và suy luận trên các mạng học sâu [3]. Google đã phát hành TensorFlow như là một dự án mã nguồn mở, và nó đã trở thành sử dụng rộng rãi cho các nghiên cứu học máy. Công cụ sử dụng để viết code là phần mềm IDE Pycharm.



Hình 2.20 Phần mềm IDE Pycharm

2.6 Kết luận chương

Trong chương 2, chúng ta đã tìm hiểu được kiến trúc và quy trình xử lý thông tin của mạng LSTM. Vấn đề được đặt ra để giải quyết bài toán dự báo chuỗi thời gian, lượng hành khách đi máy bay quốc tế. Chúng tôi đã nêu ra được mô hình thử nghiệm và các bước xử lý của mô-đun dự đoán. Thử nghiệm và đánh giá kết quả sẽ được chúng tôi nêu ra trong Chương 3.

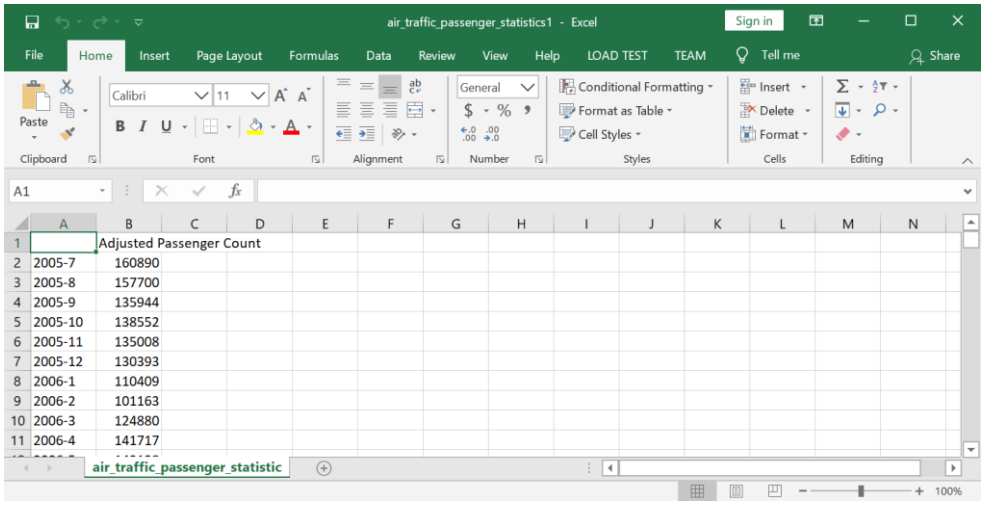
CHƯƠNG 3
THỬ NGHIỆM VÀ ĐÁNH GIÁ KẾT QUẢ

3.1 Giới thiệu bài toán ước lượng hành khách đi máy bay quốc tế

Ngày nay, ngành hàng không là một ngành công nghiệp vô cùng phát triển, nó phục vụ nhu cầu đi lại, giao thương giữa các quốc gia, cùng với đó là sự phát triển kinh tế của các nước có đường bay quốc tế. Để đáp ứng nhu cầu phục vụ hành khách một cách tốt nhất, việc phải có một hệ thống dự đoán lượng hành khách đi máy bay là rất cần thiết.

Trong luận văn này, tôi sử dụng tập dữ liệu của hãng hàng không quốc tế American Airlines.

Với dữ liệu đầu vào một tập số lượng hành khách đi máy bay quốc tế theo tháng và năm, từ 07/2005 đến 03/2016, với 129 lượt quan sát. Bộ dữ liệu có sẵn miễn phí từ trang web https://data.world/data-society/air-traffic-passenger-data dưới dạng tệp tin CSV.



Hình 3.1 Dữ liệu đầu vào số hành khách đi máy bay quốc tế

Cột đầu tiên là số liệu thời gian theo đơn vị tháng (yyyy-mm)

Cột thứ hai là số lượng hành khách đi máy bay trong thời gian tháng đó

3.2 Các kịch bản thử nghiệm

Luận văn thực hiện đánh giá độ chính xác của bài toán dự đoán lượng hành khách đi máy bay từ tập dữ liệu sô lượng hành khách theo tháng theo phương pháp LSTM với 5 trường hợp:

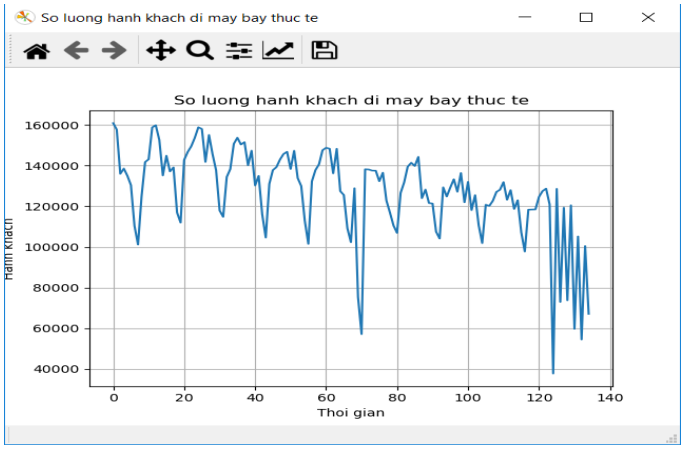
- LSTM hồi quy
- LSTM hồi quy sử dụng phương thức cửa sổ
- LSTM hồi quy sử dụng bước thời gian
- LSTM sử dụng bộ nhớ giữa các bước
- LSTM xếp chồng sử dụng bộ nhớ giữa các bước

5 kịch bản trên đều sử dụng chung một tập dữ liệu, sau đó chuyển đổi thành tập huấn luyện và tập thử nghiệm, đưa vào mạng LSTM tương ứng để đánh giá sai số trung bình với tập huấn luyện và tập thử nghiệm.

3.3 Một số kết quả thực nghiệm

3.3.1 Kết quả thử nghiệm độ chính xác dự đoán lượng hành khách đi máy bay

Ta có thể tải tập dữ liệu này một cách dễ dàng bằng thư viện Pandas. Ta không quan tâm đến ngày, cho rằng mỗi quan sát được phân tách bằng cùng một khoảng thời gian một tháng. Do đó, khi ta tải tập dữ liệu, ta có thể loại trừ cột đầu tiên. Sau khi tải, chúng ta có thể dễ dàng vẽ toàn bộ dữ liệu.



Hình 3.2 Đồ thị đầu vào số hành khách đi máy bay quốc tế

3.3.2 Phương pháp LSTM hồi quy

Theo phương pháp này đầu ra của thử nghiệm:

Epoch 98/100

- 0s - loss: 0.0196

Epoch 99/100

- 0s - loss: 0.0197

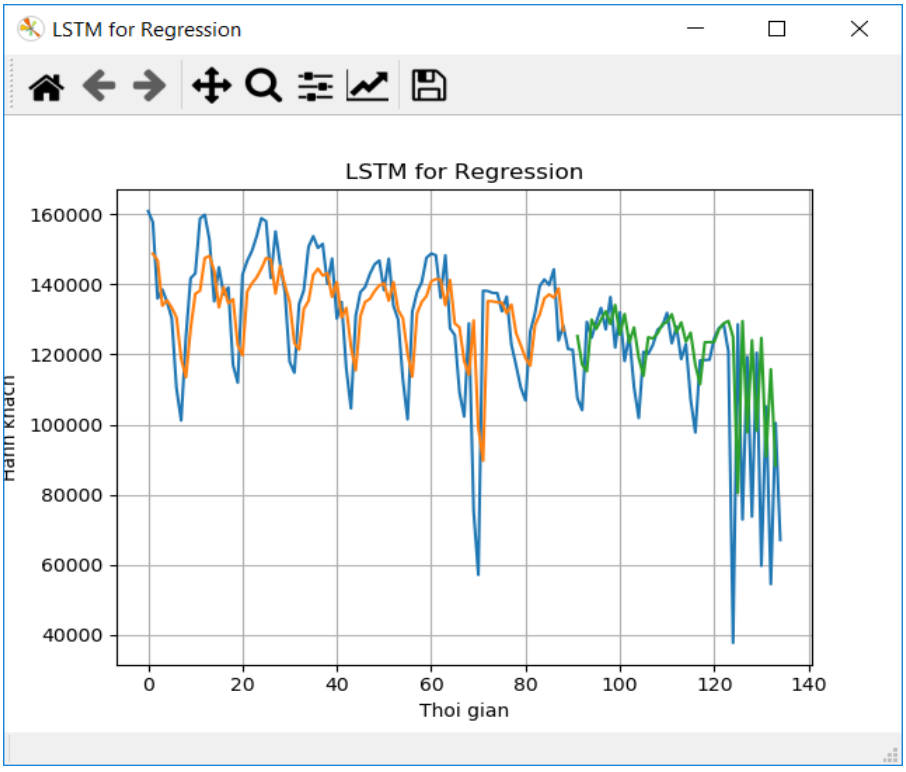
Epoch 100/100

- 0s - loss: 0.02

Train Score: 14528.83 RMSE

Test Score: 11019.19 RMSE

Chúng ta có thể thấy rằng mô hình có lỗi trung bình khoảng 14528.83 hành khách trên tập dữ liệu huấn luyện và khoảng 11019.19 hành khách trên tập dữ liệu thử nghiệm.



Hình 3.3 LSTM hồi quy

3.3.3 Phương pháp LSTM hồi quy sử dụng phương thức cửa sổ

Theo phương pháp này đầu ra của thử nghiệm:

Epoch 98/100

- 0s - loss: 0.0198

Epoch 99/100

- 0s - loss: 0.0195

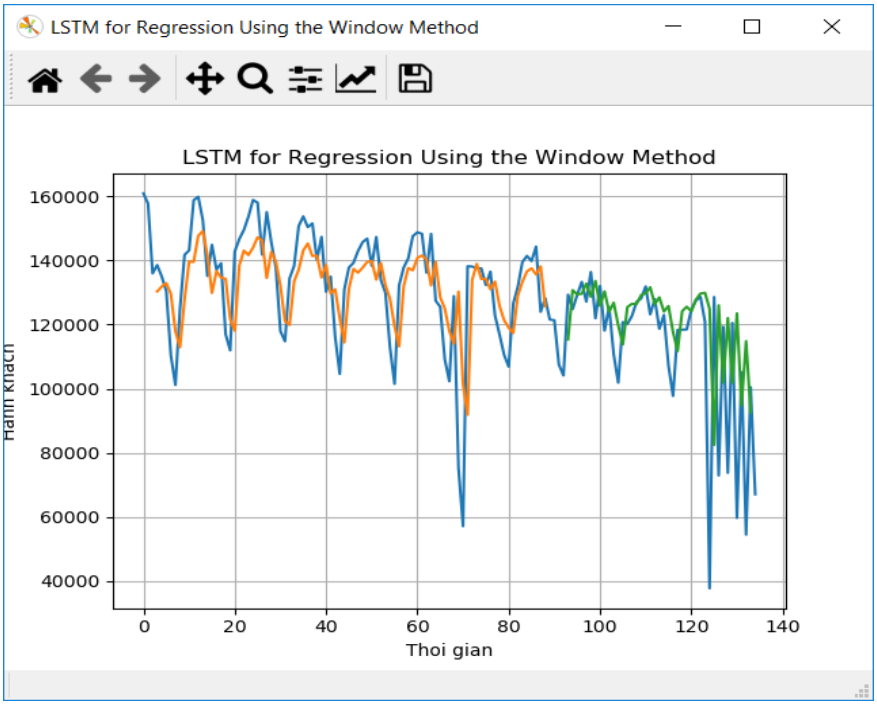
Epoch 100/100

- 0s - loss: 0.019

Train Score: 14517.25 RMSE

Test Score: 10185.20 RMSE

Chúng ta có thể thấy rằng mô hình có lỗi trung bình khoảng 14517.25 hành khách trên tập dữ liệu huấn luyện và khoảng 10185.20 hành khách trên tập dữ liệu thử nghiệm. Ta thấy rằng lỗi đã tăng nhẹ so với LSTM tuyến tính. Kích thước cửa sổ và kiến trúc mạng không được điều chỉnh: đây chỉ là một minh họa về cách đóng khung một vấn đề dự đoán



Hình 3.4 LSTM hồi quy sử dụng phương thức cửa sổ

3.3.4 Phương pháp LSTM hồi quy sử dụng bước thời gian

Theo phương pháp này đầu ra của thử nghiệm:

Epoch 98/100

- 0s - loss: 0.022

Epoch 99/100

- 0s - loss: 0.0216

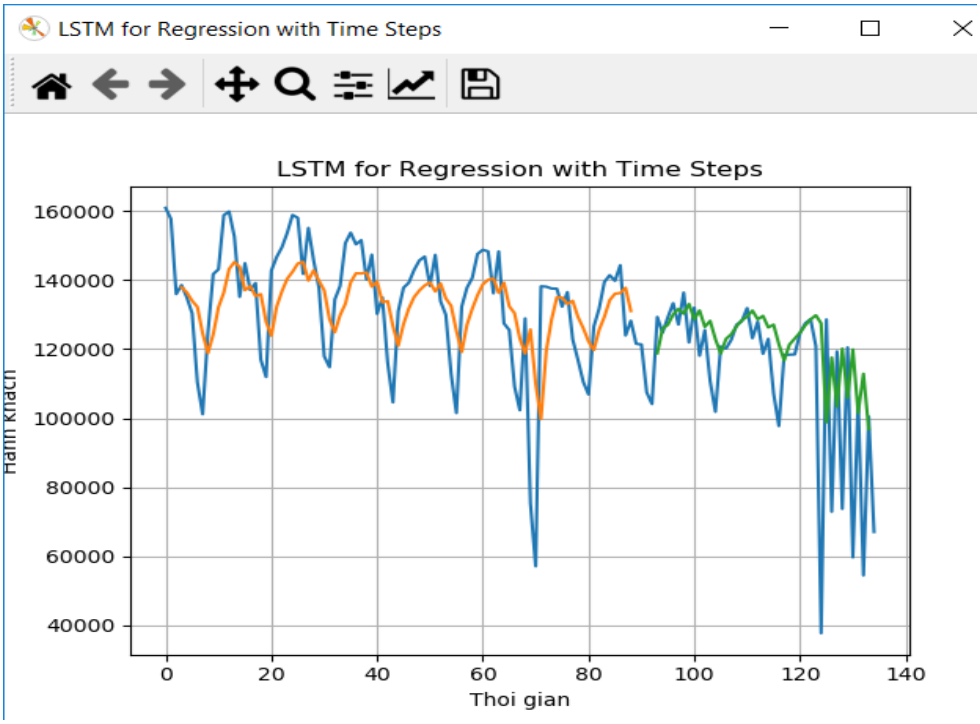
Epoch 100/100

- 0s - loss: 0.0212

Train Score: 15010.01 RMSE

Test Score: 9636.49 RMSE

Chúng ta có thể thấy rằng mô hình có lỗi trung bình khoảng 15010.01 hành khách trên tập dữ liệu huấn luyện và khoảng 9636.49 hành khách trên tập dữ liệu thử nghiệm.



Hình 3.5 LSTM hồi quy sử dụng bước thời gian

3.3.5 Phương pháp LSTM sử dụng bộ nhớ giữa các bước

Theo phương pháp này đầu ra của thử nghiệm:

Epoch 1/1

- 0s - loss: 0.022

Epoch 1/1

- 0s - loss: 0.022

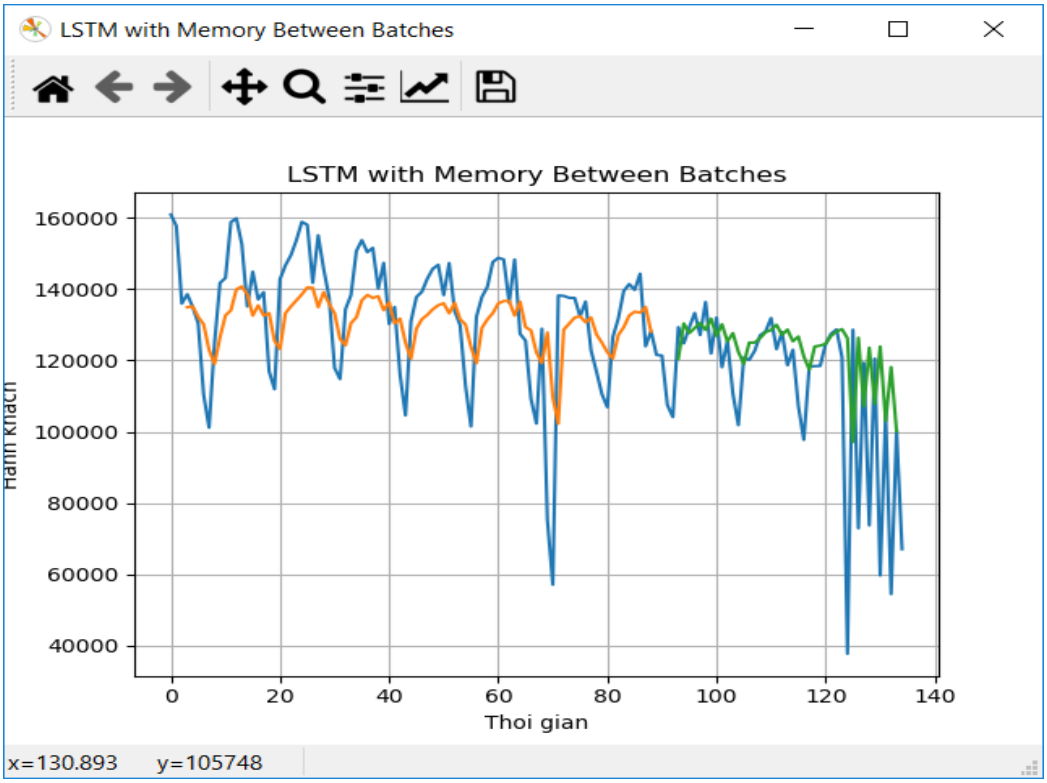
Epoch 1/1

- 0s - loss: 0.022

Train Score: 15152.41 RMSE

Test Score: 10315.36 RMSE

Chúng ta có thể thấy rằng mô hình có lỗi trung bình khoảng 15152.41 hành khách trên tập dữ liệu huấn luyện và khoảng 10315.36 hành khách trên tập dữ liệu thử nghiệm



Hình 3.6 LSTM sử dụng bộ nhớ giữa các bước

3.3.6 LSTM xếp chồng sử dụng bộ nhớ giữa các bước

Theo phương pháp này đầu ra của thử nghiệm:

Epoch 1/1

- 0s - loss: 0.0228

Epoch 1/1

- 0s - loss: 0.0227

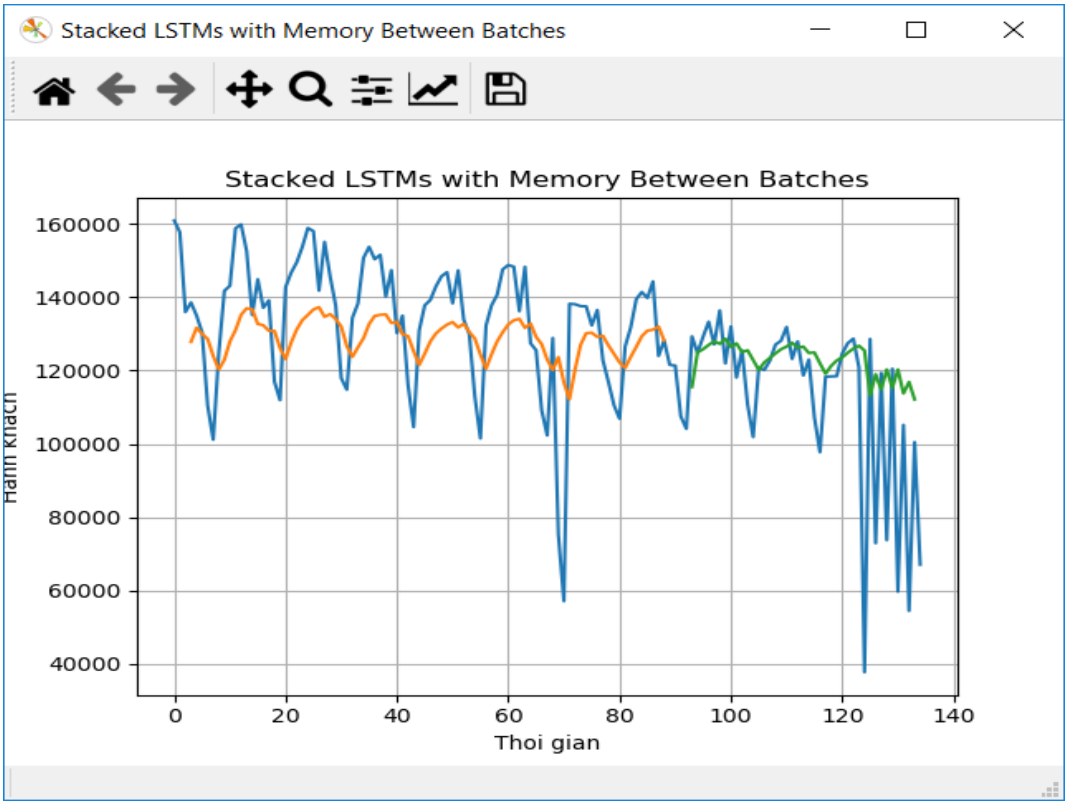
Epoch 1/1

- 0s - loss: 0.0226

Train Score: 15520.61 RMSE

Test Score: 9260.33 RMSE

Chúng ta có thể thấy rằng mô hình có lỗi trung bình khoảng 15520.61 hành khách trên tập dữ liệu huấn luyện và khoảng 9260.33 hành khách trên tập dữ liệu thử nghiệm



Hình 3.7 LSTM xếp chồng sử dụng bộ nhớ giữa các bước

3.4 Kết luận chương

Tổng kết dữ liệu lỗi trung bình của 5 kịch bản cho ta Bảng 3.1

Bảng 3.1 Đánh giá kết quả dự đoán hành khách đi máy bay quốc tế

	Lỗi trung bình (Tập huấn luyện)	**Lỗi trung bình (Tập kiểm nghiệm)**
LSTM hồi quy	14528,72 (0,087%)	11018,91 (0,066%)
LSTM hồi quy sử dụng phương thức cửa sổ	14157,42 (0,085%)	10185,59 (0,061%)
LSTM hồi quy sử dụng bước thời gian	15010,17 (0,09%)	9636,19 (0,058%)
LSTM sử dụng bộ nhớ giữa các bước	15151,38 (0,09%)	10310,33 (0,062%)
LSTM xếp chồng sử dụng bộ nhớ giữa các bước	15520,49 (0,093%)	9260,16 (0,055%)

Có thể thấy để dự đoán số lượng hành khách đi máy bay quốc tế thì phương pháp LSTM hồi quy sử dụng phương thức cửa sổ cho mức độ lỗi trung bình thấp nhất (0,085%). Khi áp dụng các kỹ thuật khác tỉ lệ lỗi trung bình tăng lên, đặc biệt phương pháp LSTM xếp chồng sử dụng bộ nhớ giữa các bước cho sai số khá cao (0,093%).

KẾT LUẬN

Luận văn đã trình bày được khái niệm, kiến thức cơ bản về công nghệ LSTM, áp dụng công nghệ mạng nơ-ron nhân tạo trong việc dự đoán lượng hành khách đi máy bay quốc tế. Các kết quả của Luận văn bao gồm:

Đã đề xuất và thử nghiệm phương pháp dự đoán lượng hành khách đi máy bay áp dụng công nghệ LSTM. Làm thế nào để phát triển các mạng LSTM cho hồi quy, khung cửa sổ và thời gian dựa trên các vấn đề dự đoán chuỗi thời gian. Làm thế nào để phát triển và đưa ra dự đoán bằng cách sử dụng các mạng LSTM duy trì trạng thái qua các quá trình. Phát triển ứng dụng ứng dụng trên hệ điều hành windows.

Về hạn chế của luận văn, do thời gian và nguồn lực và kiến thức của học viên còn có hạn trong việc nghiên cứu nên các kết quả của luân văn mới thực hiện được việc đự đoán hành khách đi máy bay của hãng. Chưa thực nghiệm hệ thống với các thuật toán khác để có cơ sở so sánh, đánh giá hiệu quả

Trong thời gian tới học viên mong muốn nghiên cứu sâu hơn để cải thiện hiệu suất, tăng tốc độ xử lý dữ liệu với dữ liệu lớn. Nghiên cứu các phương pháp nâng cao độ chính xác dự đoán lượng hành khách đi máy bay. Xây dựng hệ thống hoàn chỉnh với tập dữ liệu lớn và triển khai thử nghiệm ứng dụng trên các nền tảng khác.

TÀI LIỆU THAM KHẢO

Tiếng Việt

[1] Nguyễn Đình Thúc và Hoàng Đức Hải, *"Trí tuệ nhân tạo – Mạng nơ ron, phương pháp và ứng dụng*", Nhà xuất bản Giáo dục, Hà nội.

[2] Bùi Công Cường, Nguyễn Doãn Phước, "*Hệ mờ, mạng nơron và ứng dụng"*. Nhà xuất bản Khoa học và kỹ thuật. Hà Nội 2001.

[3] Nguyễn Mạnh Tùng, *"Nghiên cứu ứng dụng mạng nơron nhân tạo cho các bài toán đo lường"*. Luận án tiến sĩ kỹ thuật, trọờng đại học Bách Khoa Hà Nội, 2003.

Tiếng Anh

[4] Sepp Hochreiter; Jürgen Schmidhuber (1997). *"Long short-term memory"*. *Neural Computation*. **9** (8): 1735–1780. doi:10.1162/neco.1997.9.8.1735. PMID 9377276.

[5] Felix A. Gers; Jürgen Schmidhuber; Fred Cummins (2000). *"Learning to Forget: Continual Prediction with LSTM"*. *Neural Computation*. **12** (10): 2451–2471.

[6] *"The Large Text Compression Benchmark"*. Retrieved 2017-01-13.

[7] Graves, A.; Liwicki, M.; Fernández, S.; Bertolami, R.; Bunke, H.; Schmidhuber, J. (May 2009). "*A Novel Connectionist System for Unconstrained Handwriting Recognition*". *IEEE Transactions on Pattern Analysis and Machine Intelligence*. **31**(5): 855–868.

[8] Graves, Alex; Mohamed, Abdel-rahman; Hinton, Geoffrey (2013-03-22). "*Speech Recognition with Deep Recurrent Neural Networks*". arXiv:1303.5778

[9] Beaufays, Françoise (August 11, 2015). *"The neural networks behind Google Voice transcription"*. *Research Blog*. Retrieved 2017-06-27.

[10] Sak, Haşim; Senior, Andrew; Rao, Kanishka; Beaufays, Françoise; Schalkwyk, Johan (September 24, 2015). *"Google voice search: faster and more accurate"*. *Research Blog*. Retrieved 2017-06-27.

[11] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, Yoshua Bengio, Sep 2014. *"Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation".*

[12] Ilya Sutskever, Oriol Vinyals, Quoc V. Le, 14 Dec 2014. *"Sequence to Sequence Learning with Neural Networks*" pp. 1–9.

[13] Andrej Karpathy, Li Fei-Fei, 2015. "*Deep Visual-Semantic Alignments for Generating Image Descriptions"*.

[14] Christopher Olah, 27 Aug 2015. *"Understanding LSTM Networks".*

[15] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, …, 31 May 2016. "*TensorFlow: A system for large-scale machine learning*". In Arxiv preprint arXiv:1605.08695.

[16] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, …, 16 Mar 2016. "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems". In Arxiv preprint arXiv:1603.04467

Online

[17] http://bnews.vn/iata-du-doan-luong-hanh-khach-di-may-bay-dat-3-8-ty-luot-nam-nay/26702.html

PHỤ LỤC

```
# Đồ thị đầu vào
import pandas
import matplotlib.pyplot as plt
dataset = pandas.read_csv('international-airline-passengers.csv', usecols=[1],
engine='python', skipfooter=3)
fig = plt.figure()
fig.canvas.set_window_title('So luong hanh khach di may bay thuc te')
plt.plot(dataset)
plt.xlabel('Thoi gian')
plt.ylabel('Hanh khach')
plt.title('So luong hanh khach di may bay thuc te')
plt.grid(True)
plt.show()
# LSTM hồi quy
import numpy
import matplotlib.pyplot as plt
from pandas import read_csv
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
# convert an array of values into a dataset matrix
def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset) - look_back - 1):
        a = dataset[i:(i + look_back), 0]
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    return numpy.array(dataX), numpy.array(dataY)
# fix random seed for reproducibility
numpy.random.seed(7)
# load the dataset
dataframe = read_csv('international-airline-passengers.csv', usecols=[1],
```

```
engine='python', skipfooter=3)
dataset = dataframe.values
dataset = dataset.astype('float32')
# normalize the dataset
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)
# split into train and test sets
train_size = int(len(dataset) * 0.67)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size, :], dataset[train_size:len(dataset), :]
# reshape into X=t and Y=t+1
look_back = 1
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)
# reshape input to be [samples, time steps, features]
trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
# create and fit the LSTM network
model = Sequential()
model.add(LSTM(4, input_shape=(1, look_back)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, epochs=100, batch_size=1, verbose=2)
# make predictions
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)
# invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])
# calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:, 0]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:, 0]))
print('Test Score: %.2f RMSE' % (testScore))
# shift train predictions for plotting
```

```
trainPredictPlot = numpy.empty_like(dataset)
trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[look_back:len(trainPredict) + look_back, :] = trainPredict
# shift test predictions for plotting
testPredictPlot = numpy.empty_like(dataset)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict) + (look_back * 2) + 1:len(dataset) - 1, :] =
testPredict
# plot baseline and predictions
fig = plt.figure()
fig.canvas.set_window_title('LSTM for Regression')
plt.plot(scaler.inverse_transform(dataset))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.xlabel('Thoi gian')
plt.ylabel('Hanh khach')
plt.title('LSTM for Regression')
plt.grid(True)
plt.show()
# LSTM hồi quy sử dụng phương thức cửa sổ
import numpy
import matplotlib.pyplot as plt
from pandas import read_csv
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error


# convert an array of values into a dataset matrix
def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset) - look_back - 1):
        a = dataset[i:(i + look_back), 0]
        dataX.append(a)
```

```
        dataY.append(dataset[i + look_back, 0])
    return numpy.array(dataX), numpy.array(dataY)
# fix random seed for reproducibility
numpy.random.seed(7)
# load the dataset
dataframe = read_csv('international-airline-passengers.csv', usecols=[1],
engine='python', skipfooter=3)
dataset = dataframe.values
dataset = dataset.astype('float32')
# normalize the dataset
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)
# split into train and test sets
train_size = int(len(dataset) * 0.67)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size, :], dataset[train_size:len(dataset), :]
# reshape into X=t and Y=t+1
look_back = 3
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)
# reshape input to be [samples, time steps, features]
trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
# create and fit the LSTM network
model = Sequential()
model.add(LSTM(4, input_shape=(1, look_back)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, epochs=100, batch_size=1, verbose=2)
# make predictions
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)
# invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])
```

```
# calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:, 0]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:, 0]))
print('Test Score: %.2f RMSE' % (testScore))
# shift train predictions for plotting
trainPredictPlot = numpy.empty_like(dataset)
trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[look_back:len(trainPredict) + look_back, :] = trainPredict
# shift test predictions for plotting
testPredictPlot = numpy.empty_like(dataset)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict) + (look_back * 2) + 1:len(dataset) - 1, :] =
testPredict
# plot baseline and predictions
fig = plt.figure()
fig.canvas.set_window_title('LSTM for Regression Using the Window Method')
plt.plot(scaler.inverse_transform(dataset))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.xlabel('Thoi gian')
plt.ylabel('Hanh khach (x1000)')
plt.title('LSTM for Regression Using the Window Method')
plt.grid(True)
plt.show()
# LSTM hồi quy sử dụng bước thời gian
import numpy
import matplotlib.pyplot as plt
from pandas import read_csv
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
# convert an array of values into a dataset matrix
def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
```

```
    for i in range(len(dataset) - look_back - 1):
        a = dataset[i:(i + look_back), 0]
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    return numpy.array(dataX), numpy.array(dataY)
# fix random seed for reproducibility
numpy.random.seed(7)
# load the dataset
dataframe = read_csv('international-airline-passengers.csv', usecols=[1],
engine='python', skipfooter=3)
dataset = dataframe.values
dataset = dataset.astype('float32')
# normalize the dataset
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)
# split into train and test sets
train_size = int(len(dataset) * 0.67)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size, :], dataset[train_size:len(dataset), :]
# reshape into X=t and Y=t+1
look_back = 3
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)
# reshape input to be [samples, time steps, features]
trainX = numpy.reshape(trainX, (trainX.shape[0], trainX.shape[1], 1))
testX = numpy.reshape(testX, (testX.shape[0], testX.shape[1], 1))
# create and fit the LSTM network
model = Sequential()
model.add(LSTM(4, input_shape=(look_back, 1)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, epochs=100, batch_size=1, verbose=2)
# make predictions
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)
# invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
```

```
testY = scaler.inverse_transform([testY])
# calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:, 0]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:, 0]))
print('Test Score: %.2f RMSE' % (testScore))
# shift train predictions for plotting
trainPredictPlot = numpy.empty_like(dataset)
trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[look_back:len(trainPredict) + look_back, :] = trainPredict
# shift test predictions for plotting
testPredictPlot = numpy.empty_like(dataset)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict) + (look_back * 2) + 1:len(dataset) - 1, :] =
testPredict
# plot baseline and predictions
fig = plt.figure()
fig.canvas.set_window_title('LSTM for Regression with Time Steps')
plt.plot(scaler.inverse_transform(dataset))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.xlabel('Thoi gian')
plt.ylabel('Hanh khach (x1000)')
plt.title('LSTM for Regression with Time Steps')
plt.grid(True)
plt.show()
# LSTM sử dụng bộ nhớ giữa các bước
import numpy
import matplotlib.pyplot as plt
from pandas import read_csv
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
# convert an array of values into a dataset matrix
def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
```

```
    for i in range(len(dataset) - look_back - 1):
        a = dataset[i:(i + look_back), 0]
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    return numpy.array(dataX), numpy.array(dataY)
# fix random seed for reproducibility
numpy.random.seed(7)
# load the dataset
dataframe = read_csv('international-airline-passengers.csv', usecols=[1],
engine='python', skipfooter=3)
dataset = dataframe.values
dataset = dataset.astype('float32')
# normalize the dataset
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)
# split into train and test sets
train_size = int(len(dataset) * 0.67)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size, :], dataset[train_size:len(dataset), :]
# reshape into X=t and Y=t+1
look_back = 3
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)
# reshape input to be [samples, time steps, features]
trainX = numpy.reshape(trainX, (trainX.shape[0], trainX.shape[1], 1))
testX = numpy.reshape(testX, (testX.shape[0], testX.shape[1], 1))
# create and fit the LSTM network
model = Sequential()
model.add(LSTM(4, input_shape=(look_back, 1)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, epochs=100, batch_size=1, verbose=2)
# make predictions
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)
# invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
```

```
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])
# calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:, 0]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:, 0]))
print('Test Score: %.2f RMSE' % (testScore))
# shift train predictions for plotting
trainPredictPlot = numpy.empty_like(dataset)
trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[look_back:len(trainPredict) + look_back, :] = trainPredict
# shift test predictions for plotting
testPredictPlot = numpy.empty_like(dataset)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict) + (look_back * 2) + 1:len(dataset) - 1, :] =
testPredict
# plot baseline and predictions
fig = plt.figure()
fig.canvas.set_window_title('LSTM for Regression with Time Steps')
plt.plot(scaler.inverse_transform(dataset))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.xlabel('Thoi gian')
plt.ylabel('Hanh khach (x1000)')
plt.title('LSTM for Regression with Time Steps')
plt.grid(True)
plt.show()
# LSTM xếp chồng sử dụng bộ nhớ giữa các bước
import numpy
import matplotlib.pyplot as plt
from pandas import read_csv
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
```

```
from sklearn.metrics import mean_squared_error
# convert an array of values into a dataset matrix
def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset) - look_back - 1):
        a = dataset[i:(i + look_back), 0]
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    return numpy.array(dataX), numpy.array(dataY)
# fix random seed for reproducibility
numpy.random.seed(7)
# load the dataset
dataframe = read_csv('international-airline-passengers.csv', usecols=[1],
engine='python', skipfooter=3)
dataset = dataframe.values
dataset = dataset.astype('float32')
# normalize the dataset
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)
# split into train and test sets
train_size = int(len(dataset) * 0.67)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size, :], dataset[train_size:len(dataset), :]
# reshape into X=t and Y=t+1
look_back = 3
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)
# reshape input to be [samples, time steps, features]
trainX = numpy.reshape(trainX, (trainX.shape[0], trainX.shape[1], 1))
testX = numpy.reshape(testX, (testX.shape[0], testX.shape[1], 1))
# create and fit the LSTM network
batch_size = 1
model = Sequential()
model.add(LSTM(4, batch_input_shape=(batch_size, look_back, 1), stateful=True,
return_sequences=True))
model.add(LSTM(4, batch_input_shape=(batch_size, look_back, 1), stateful=True))
model.add(Dense(1))
```

```
model.compile(loss='mean_squared_error', optimizer='adam')
for i in range(100):
    model.fit(trainX, trainY, epochs=1, batch_size=batch_size, verbose=2,
shuffle=False)
    model.reset_states()
# make predictions
trainPredict = model.predict(trainX, batch_size=batch_size)
model.reset_states()
testPredict = model.predict(testX, batch_size=batch_size)
# invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])
# calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:, 0]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:, 0]))
print('Test Score: %.2f RMSE' % (testScore))
# shift train predictions for plotting
trainPredictPlot = numpy.empty_like(dataset)
trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[look_back:len(trainPredict) + look_back, :] = trainPredict
# shift test predictions for plotting
testPredictPlot = numpy.empty_like(dataset)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict) + (look_back * 2) + 1:len(dataset) - 1, :] =
testPredict
# plot baseline and predictions
fig = plt.figure()
fig.canvas.set_window_title('Stacked LSTMs with Memory Between Batches')

plt.plot(scaler.inverse_transform(dataset))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)

plt.xlabel('Thoi gian')
```

```
plt.ylabel('Hanh khach (x1000)')
plt.title('Stacked LSTMs with Memory Between Batches')
plt.grid(True)
plt.show()
```