**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC NGOẠI NGỮ - TIN HỌC TP.HCM**

**KHOA CÔNG NGHỆ THÔNG TIN**

**KHÓA LUẬN TỐT NGHIỆP**

**TÌM HIỂU CƠ CHẾ ATTENTION**

**VÀ ÁP DỤNG CHO BÀI TOÁN**

**PHÂN TÍCH CẢM XÚC**

**GIẢNG VIÊN HƯỚNG DẪN: ThS. Trần Khải Thiện**

**SINH VIÊN THỰC HIỆN:**

**Nhan Kim Thành – 15DH110281**

**Trần Đức Khang – 15DH110240**

**TP.HỒ CHÍ MINH - 2019**

LỜI CẢM ƠN

🙞🕮🙜

Để hoàn thành khóa luận đề tài nghiên cứu khoa học này, bên cạnh sự nỗ lực của bản thân đã vận dụng và tìm tòi học hỏi, chúng em đã nhận được sự giúp đỡ, hỗ trợ từ nhiều cơ quan, tổ chức, và cá nhân. Nghiên cứu khoa học được hoàn thành dựa trên nhiều tham khảo, kinh nghiệm từ các kết quả nghiên cứu liên quan, các bài báo chuyên ngành của nhiều tác giả ở các tổ chức nghiên cứu khác.

Với tình cảm chân thành, chúng em xin dành lời cảm ơn chân thành đối với các giáo viên trong trường Đại học Ngoại Ngữ Tin Học Thành phố Hồ Chí Minh, đặc biệt là giáo viên hướng dẫn trực tiếp của nhóm nghiên cứu – Thạc Sĩ Trần Khải Thiện. Cảm ơn thầy đã dành nhiều thời gian cũng như công sức và truyền đạt nhiều kiến thức trong việc theo dõi và hướng dẫn nhóm hoàn thành đề tài nghiên cứu.

Do vẫn còn giới hạn về kiến thức, thời gian và khả năng luận lý của nhóm nên không thể tránh khỏi nhiều thiếu sót, kính mong sự chỉ dẫn của các giáo viên cũng như đóng góp ý kiến từ những người quan tâm đề tài của nhóm, để có thể tiếp thu và cải thiện hơn.

Xin chân thành cảm ơn!

MỤC LỤC

[Chương 1. GIỚI THIỆU CHUNG 1](#_Toc11027223)

[Chương 2. BÀI TOÁN SENTIMENT ANALYSIS 3](#_Toc11027224)

[2.1 Giới thiệu về text classification 3](#_Toc11027225)

[2.2 Bài toán phân tích cảm xúc 3](#_Toc11027226)

[2.3 Những thách thức đối với bài toán phân tích cảm xúc: 4](#_Toc11027227)

[Chương 3. CÁC HƯỚNG TIẾP CẬN PHÂN TÍCH CẢM XÚC 7](#_Toc11027228)

[Chương 4. CÁC MÔ HÌNH MẠNG NEURAL 10](#_Toc11027229)

[4.1 Mạng nơ-rôn (neural network): 10](#_Toc11027230)

[4.2 Mạng neural tích chập (CNN): 13](#_Toc11027231)

[4.3 Mạng neural quy hồi (RNN): 17](#_Toc11027232)

[4.3.1 Lan truyền ngược liên hồi (BPTT – backpropagation through time): 19](#_Toc11027233)

[4.3.2 Vấn đề mất mát đạo hàm (vanishing gradient problem): 21](#_Toc11027234)

[4.4 Mạng neural phụ thuộc gần xa (LSTM : Long-Short Term Memory): 21](#_Toc11027235)

[4.5 Mô hình đề cử trong bài toán phân tích cảm xúc: 27](#_Toc11027236)

[4.5.1 Mô hình seq2seq (sequence to sequence): 27](#_Toc11027237)

[4.5.2 Cơ chế Attention trong mô hình seq2seq: 28](#_Toc11027238)

[Chương 5. THỰC NGHIỆM VÀ ĐÁNH GIÁ 31](#_Toc11027239)

[5.1. Dataset 31](#_Toc11027240)

[5.2. Cấu hình máy và thư viện cài đặt 31](#_Toc11027241)

[5.3. Thực nghiệm mô hình RNN-Attention-Network: 32](#_Toc11027242)

[**5.3.1. Xử lý dữ liệu:** 32](#_Toc11027243)

[**5.3.2 Xây dựng mô hình dựa trên RNN-Attention-network** 34](#_Toc11027244)

[**5.3.3 Thực nghiệm RNN-Attention-network với dữ liệu Book** 37](#_Toc11027245)

[**5.3.4 Thực nghiệm RNN-Attention-network với dữ liệu DVD** 40](#_Toc11027246)

[**5.3.5 Thực nghiệm RNN-Attention-network với dữ liệu Electronics** 42](#_Toc11027247)

[**5.3.6 Thực nghiệm RNN-Attention-network với dữ liệu Kitchen** 45](#_Toc11027248)

[**5.3.7 Thực nghiệm RNN-Attention-network với dữ liệu IMDB Keras** 47](#_Toc11027249)

[**5.3.8 So sánh các mô hình** 50](#_Toc11027250)

[KẾT LUẬN 51](#_Toc11027251)

[TÀI LIỆU THAM KHẢO 52](#_Toc11027252)

**DANH SÁCH BẢNG**

(Nếu có)

**DANH SÁCH HÌNH**

[Hình 3‑1 7](file:///C:\Users\HT\Downloads\Mau-Bao-cao-KLTN%20(3).docx#_Toc11012557)

[Hình 3‑2 8](file:///C:\Users\HT\Downloads\Mau-Bao-cao-KLTN%20(3).docx#_Toc11012558)

[Hình 4‑1 10](file:///C:\Users\HT\Downloads\Mau-Bao-cao-KLTN%20(3).docx#_Toc11012559)

[Hình 4‑2 12](#_Toc11012560)

[Hình 4‑3 12](#_Toc11012561)

[Hình 4‑4 13](#_Toc11012562)

[Hình 4‑5 14](file:///C:\Users\HT\Downloads\Mau-Bao-cao-KLTN%20(3).docx#_Toc11012563)

[Hình 4‑6 15](file:///C:\Users\HT\Downloads\Mau-Bao-cao-KLTN%20(3).docx#_Toc11012564)

[Hình 4‑7 16](#_Toc11012565)

[Hình 4‑8 18](#_Toc11012566)

[Hình 4‑9 19](#_Toc11012567)

[Hình 4‑10 21](#_Toc11012568)

[Hình 5‑1. Hierarchical Attention Network 27](#_Toc11012569)

[Hình 5‑2 Training and validation loss (Book) 31](#_Toc11012570)

[Hình 5‑3 Training and validation accuracy (Book) 31](#_Toc11012571)

[Hình 5‑4 Biểu đồ kết quả các model (Book) 32](#_Toc11012572)

[Hình 5‑5 Training and validation loss (DVD) 33](#_Toc11012573)

[Hình 5‑6 Training and validation accuracy (DVD) 34](#_Toc11012574)

[Hình 5‑7 Biểu đồ kết quả các model (DVD) 34](#_Toc11012575)

[Hình 5‑8 Training and validation loss (Electronics) 36](#_Toc11012576)

[Hình 5‑9 Training and validation accuracy (Electronics) 36](#_Toc11012577)

[Hình 5‑10 Biểu đồ kết quả các model (Electronics) 37](#_Toc11012578)

[Hình 5‑11 Training and validation loss (Kitchen) 38](#_Toc11012579)

[Hình 5‑12 Training and validation accuracy (Kitchen) 39](#_Toc11012580)

[Hình 5‑13 Biểu đồ kết quả các model (Kitchen) 39](#_Toc11012581)

[Hình 5‑14 Training and validation loss (IMDB) 41](#_Toc11012582)

[Hình 5‑15 Training and validation accuracy (IMDB) 41](#_Toc11012583)

[Hình 5‑16 Biểu đồ kết quả các model (IMDB) 42](#_Toc11012584)

# GIỚI THIỆU CHUNG

* **Giới thiệu tổng quan về trí tuệ nhân tạo**

Hiện nay, mọi thứ đang trên đà hiện đại hóa, mọi vấn đề đều được xử lý bởi công nghệ nói chung, trong đó nổi trội nhất là mảng trí tuệ nhân tạo nói riêng. Trí tuệ nhân tạo, gọi là AI (Artificial Intelligence) ra đời nhằm thay thế con người trong việc xử lý các vấn đề một cách tự động, hiệu quả cao và chính xác hơn.

Thuật ngữ trí tuệ nhân tạo được sử dụng bởi một nhóm các nhà khóa học máy tính tại hội nghị Dartmouth vào năm 1956, những thập kỷ sau đó, lĩnh vực này đã có những bước phát triển mạnh mẽ vượt bật, và được dự đoán sẽ là chìa khóa mở ra một tương lai mới đối với nền văn minh nhân loại.

Trí tuệ nhân tạo hiện đang được áp dụng trong các mọi lĩnh vực như sinh học, khoa học, giảng dạy, an ninh,… Những ông lớn trong ngành công nghệ cũng áp dụng chúng vào ứng dụng để cải thiện và quản lý. Ví dụ như khả năng nhận dạng khuôn mặt của Facebook hay trình phân loại hình ảnh của Pinterest,…

Vậy trí tuệ nhân tạo bắt đầu từ đâu? Trí tuệ nhân tạo được đào tạo từ một lượng lớn dữ liệu và thực hiện việc học hỏi, đưa ra quyết định hoặc dự đoán về vấn đề liên quan bằng các thuật toán. Việc đào tạo này gọi là học máy (Machine Learning). Học máy cung cấp những kỹ thuật, phương pháp để áp dụng vào thực hiện các khả năng của máy móc, một trong các lĩnh vực ứng dụng tốt nhất cho máy học trong nhiều năm qua là *thị giác máy tính* (computer vision), *khả năng phát hiện hình ảnh* (image detection) hay *xử lý ngôn ngữ tự nhiên* (natural language processing).

Phương pháp tiếp cận tiếp theo là khả năng khiến AI tự đào tạo bản thân, phương pháp này được gọi là Học sâu (Deep Learning). Deep learning tập trung xử lý các vấn đề liên quan đến mạng thần kinh, được dựa trên mạng thần kinh sinh học của con người, nhằm nâng cao trí tuệ nhân tạo trong các bài toán về nhận diện giọng nói, khuôn mặt hay xử lý ngôn ngữ tự nhiên. Trong vài năm qua, học sâu đã trở thành lĩnh vực nổi nhất trong khoa học máy tính, như khả năng dịch tự dộng, nhận thức sự vật, nhận diện giọng nói, phân tích cảm xúc...

# BÀI TOÁN SENTIMENT ANALYSIS

## 2.1 Giới thiệu về text classification

Bài toán về xử lý văn bản (text classification) xuất hiện khá nhiều trong các ứng dụng về xử lý ngôn ngữ tự nhiên (NLP – Natural Language Processing) được sử dụng nhiều trong kinh doanh và được thực hiện bằng các kỹ thuật học máy, học sâu. Mục đích chính của việc phân loại văn bản được chia thành nhiều mục như:

* Tìm hiểu về cảm xúc của người dùng thông qua các trang mạng xã hội.
* Xác định và phân loại các loại thư rác (spam email).
* Phân loại các bài báo thành các chủ đề xác định.
* Ứng dụng vào tạo chatbot.
* Tóm tắt văn bản.
* Dịch đoạn văn.

## 2.2 Bài toán phân tích cảm xúc

Về đề tài này, chúng ta sẽ tìm hiểu sâu bài toán xử lý văn bản trong vấn đề phân loại cảm xúc người dùng (sentiment analysis). Phân tích cảm xúc (sentiment analysis) còn được gọi là khai thác ý kiến, đây là một trong các lĩnh vực trong xử lý ngôn ngữ tự nhiên, được xây dựng trên các hệ thống nhằm xác định và trích xuất ý kiến người dùng dưới dạng văn bản.

Hiện nay, bài toán phân tích cảm xúc là một chủ đề rất được quan tâm và phát triển vì nó được ứng dụng nhiều trong thực tế, đặc biệt trong lĩnh vực kinh doanh. Một số lượng lớn các phản hồi, ý kiến của người dùng từ các trang web, các diễn đàn, blog và mạng xã hội, với sự giúp đỡ của hệ thống phân tích cảm xúc người dùng, những thông tin phi cấu trúc trên có thể được tự động chuyển thành dữ liệu có cấu trúc về ý kiến, phản hồi của công chúng về sản phẩm, dịch vụ, thương hiệu hoặc bất kỳ chủ đề nào.

Có rất nhiều loại trong phân tích cảm xúc, các công cụ phân tích thường tâp trung vào các loại chính, gồm (tích cực, tiêu cực, trung tính) để hệ thống có thể xác định biểu cảm và cảm xúc như (giận dữ, vui vẻ, buồn bã, thất vọng, …) hoặc xác định ý định như (hứng thú, không hứng thú). Trong đề tài này, chúng ta chỉ tập trung vào cảm xúc tích cự và tiêu cực.

Đây là một bình luận về đề tài sách theo hướng tích cực.

*“I read this book many , many years ago on a very long flight. I could n't put it down.”*

Còn đoạn bình luận này thì mang ý chê bai về đề tài phim.

*“I would n't recommend this movie. Unless you have nothing else to do for the night and want to watch it alone.”*

## 2.3 Những thách thức đối với bài toán phân tích cảm xúc:

Trong những năm gần đây, hầu hết việc phân tích cảm xúc đều tập trung phát triển các loại cảm xúc chính xác hơn bằng cách xử lý một số thách thức và hạn chế trong lĩnh vực này.

* Tính chủ quan và giọng điệu (subjectivity and tone):

Việc xác định tính chủ quan và tính khách quan trong đoạn văn rất quan trọng cũng như việc phân tích giọng điệu của nó. Thực tế, văn bản được gọi là khách quan thường không chứa cảm xúc rõ ràng. Như ví dụ sau:

*“the package is nice.”*

*“the package is red.”*

Hầu hết mọi người sẽ nhận định đoạn đầu tiên là tích cực và đoạn thứ hai là trung tính. Tất cả các vị ngữ (tính từ, động từ, danh từ) không nên được xem như nhau trong việc chúng tạo ra cảm xúc. Với ví dụ trên, từ *‘nice’* mang tính chủ quan hơn là từ *‘red’*.

* Bối cảnh và phân cực (context and polarity):

Tất cả các ý kiến được nêu ra tại một số thời điểm, một số nơi, và bởi một số người, thì chúng ta sẽ nắm bắt được nội dung câu chuyện đó. Mọi câu nói đều đi chung với bối cảnh, nên việc phân tích cảm xúc mà không dựa vào bối cảnh sẽ gây khá nhiều khó khăn. Tuy nhiên, máy móc không thể hiểu được bối cảnh nếu nó không được đề cập một cách rõ ràng. Một trong những vấn đề nảy sinh từ bối cảnh sẽ dẫn đến thay đổi trong tính phân cực (loại cảm xúc của câu văn).

*“Everything of it.”*

*“Absolutely nothing!”*

Với ví dụ trên đóng vai trò là câu trả lời, giả sử câu hỏi là *“What did you like about the event?”*, đoạn đầu tiên sẽ trở nên tích cực và đoạn sau tiêu cực. Còn nếu ta thay đổi câu hỏi thành *“What did you DISLIKE about the event?”,* tính phân cực của hai đoạn trên sẽ đảo ngược lại.

* Mỉa mai và châm biếm (irony and sarcasm):

Sự khác biệt giữa nghĩa đen và nghĩa bóng, và một định nghĩa khác của sự xúc phạm hoặc kiểu chế giễu hơn của sự mỉa mai thường thay đổi cảm xúc từ tích cực sang tiêu cực, trong khi cảm xúc tiêu cực hoặc trung tính có thể chuyển sang hướng tích cực. Tuy nhiên, việc xác định sự mỉa mai hoặc hàm ý trong đoạn văn cần có nhiều phân tích về bối cảnh, do vậy, việc xác định một cách tự động sẽ rất khó khăn.

Ví dụ như câu hỏi *“Have you had a nice customer experience with us?”*

*“Yeah. Sure”*

*“Not one, but many!”*

Giả sử chúng ta nghe câu phản hồi đầu tiên nhiều lần liên tục, ta sẽ nghĩ rằng câu trả lời đó là tiêu cực. Vấn đề ở đây là không có văn bản gợi ý để máy có thể biết rằng đó là cảm xúc tiêu cực khi lặp lại nhiều lần, từ *‘yeah’* và *‘sure’* là những từ tích cực hoặc trung tính.

Với câu trả lời thứ hai, theo ngữ cảnh thì đây là cảm xúc tích cực. Nhưng với những ngữ cảnh khác thì câu trả lời trên thể hiện sự tiêu cực.

* So sánh (comparations):

Việc xử lý sự so sánh trong văn bản cũng là một thử thách đối với bài toán phân tích cảm xúc.

*“This product is second to none.”*

*“This is better than old tools.”*

*“This is better than nothing.”*

Trong ví dụ đầu tiên, dịch ra có nghĩa là ‘không sản phẩm nào khác có thể sánh được’, đây là một dạng câu so sánh mà ta không cần ngữ cảnh vẫn có thể phân loại một cách chính xác.

Với ví dụ thứ hai và ba thì hơi khó để phân loại. Chúng ta thường sẽ phân loại ví dụ hai vào tính tích cực và cái cuối vào trung tính. Nhưng ngữ cảnh sẽ quyết định việc phân loại.

* Biểu tượng cảm xúc (emojis):

Biểu tượng cảm xúc đúng với cái tên gọi của nó, mục đích được dùng để biểu diễn cảm xúc của người dùng được đính kèm trong đoạn văn, nó đơn giản là sự kết hợp của các ký tự unicode để tạo ra những hình ảnh mang biểu cảm. Theo Guibon, biểu tượng cảm xúc được chia ra hai loại chính, đó là *Western Emojis* (ví dụ như :D :v ) và *Eastern Emojis* (ví dụ như ( ¯ \ \_ (ツ) \_ / ¯) ).

Để phân tích cảm xúc của văn bản có đính kèm những biểu tượng này, chúng ta cần chú ý đến cấp độ ký tự (character level) cũng như cấp độ của cụm từ (word-level)

# CÁC HƯỚNG TIẾP CẬN PHÂN TÍCH CẢM XÚC

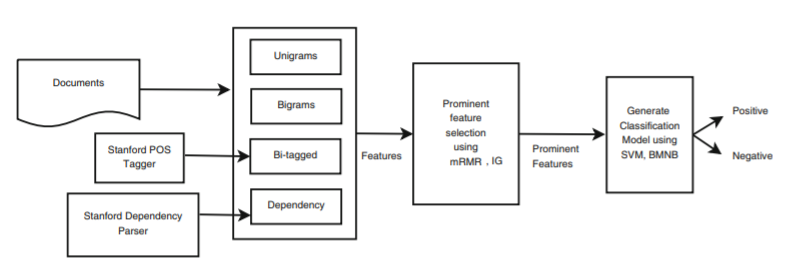
Có ba hướng tiếp cận phân tích cảm xúc trong văn bản: theo hướng học máy (machine learning approach), theo định hướng ngữ nghĩa (semantic orientation approach) và theo học sâu (deep learning approach).

* **Tiếp cận theo hướng học máy**

Các giải thuật học máy đã và đang được sử dụng rộng rãi cho bài toán phân tích cảm xúc, Bag-of-Word (BoW) là thuật toán điển hình được sử dụng phổ biến đối với bài toán này.

Phương pháp BoW chú trọng phần từ và bỏ qua tầm quan trọng của ngữ nghĩa và tính chủ quan của đoạn văn. Tất cả các từ trong đoạn văn đều được xem là quan trọng như nhau.

Phương pháp này trả ra độ cao của chiều trong các không gian tính năng (high dimensionality of the feature space), thuật toán của học máy sẽ giảm thiểu độ cao này với sự giúp đỡ của kỹ thuật lựa chọn tính năng, chỉ chọn những tính năng quan trọng bằng cách loại bỏ những tính năng dư thừa không liên quan. Gần đây, các mô hình hoạt động trên nền học máy đạt được nhiều thành công trong lĩnh vực phân tích cảm xúc.

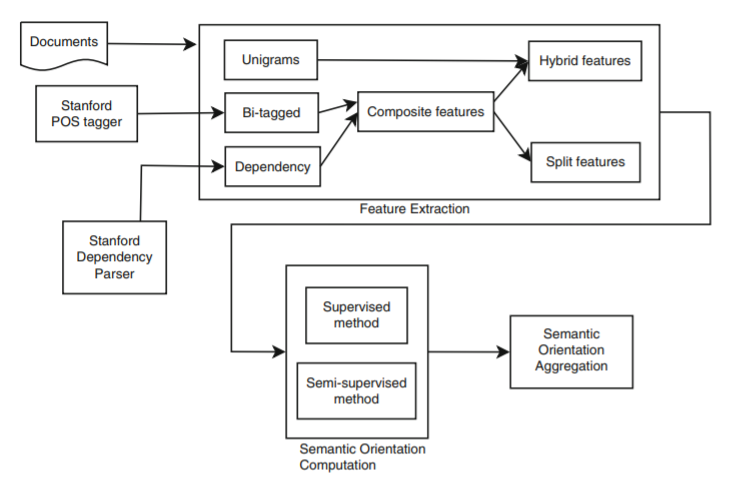


Hình 3‑1

* **Tiếp cận theo hướng ngữ nghĩa**

Tiếp cận theo hướng ngữ nghĩa được phân loại thành hướng tiếp cận dựa trên ngữ liệu (Corpus-based approach) và hướng tiếp cận dựa trên từ vựng (Lexicon-based approach).

Hướng tiếp cận dựa trên ngữ liệu chủ yếu phụ thuộc vào phương pháp xác định tính phân cực của cụm từ, cách tiếp cận này không hoạt động tốt do tính phân cực của từ phải dựa trên bối cảnh và không có kho ngữ liệu nào có thể cung cấp tính phân cực của từ dựa vào bối cảnh.

Đối với hướng tiếp cận về từ vựng, còn được gọi là hướng tiếp cận dựa trên kiến thức (knowledge-based approach), hướng này phụ thuộc trên những cơ sở kiến thức đã được phát triển như SentiWordNet, WordNet, … Hướng tiếp cận này là sự bao quát về các cơ sở kiến thức vì hầu hết những kiến thức có sẵn đều chứa kiến thức tổng hợp, điều này không đủ để có thể xác định tính phân cực của văn bản.

Hình 3‑2

Hướng tiếp cận theo hướng ngữ nghĩa chủ yếu dựa vào các từ mang cảm xúc, sau khi xác định bằng nhiều phương thức khác nhau, cuối cùng tính phân cực của đoạn văn sẽ được xác định bởi một tập hợp các hướng ngữ nghĩa của tất cả chức năng.

* **Tiếp cận theo deep learning:**

Với sự phát triển nhanh chóng tốc độ xử lý của GPU, CPU. Điều này làm tiền đề cho phương pháp học sâu bằng mạng thần kinh phát triển mạnh mẽ.

Đối với bài toán phân tích cảm xúc, phương pháp học sâu sẽ sử dụng các mô hình mạng phổ biến để giải quyết như mạng CNN (mạng nơ-rôn tích chập), RNN (mạng nơ-rôn quy hồi), cùng biến thể là LSTM (mạng nơ-rôn phụ thuộc gần xa), vượt trội hơn nữa là sử dụng cơ chế Attention trong mô hình seq2seq.

Chúng ta sẽ đi sâu vào từng mô hình ở chương tiếp theo.

# CÁC MÔ HÌNH MẠNG NEURAL

## 4.1 Mạng nơ-rôn (neural network):

Mạng neural là một mô hình mạng toán học được lập trình mô phỏng theo mạng thần kinh của con người.

Hình 4‑1

Kết cấu của một mạng neural là các neural riêng lẻ, được gọi là các *preceptron*. Một *preceptron* nhận vào một hoặc nhiều thông tin đầu vào nhưng trả ra một kết quả duy nhất. Khi neural nhận tín hiệu đầu vào từ các *dendrite*, khi tín hiệu vượt qua một ngưỡng (threshold) thì tín hiệu sẽ được truyền đi sang một neural khách theo sợi *axon*. Công thức tính output *y* như sau:

*y = a ()*

Giải thích đơn giản thì output sẽ trả ra một kết quả dựa trên tổng của các input và trọng số của chúng. Đầu vào là các *x* cùng trọng số *w* tương ứng.

*0 if ≤ threshold*

*output =*

*1 if > threshold*

Ví dụ như vấn đề đặt ra là “đi chơi”, cùng các trường hợp sau:

*“Thời tiết có nắng không?”*

*“Chuẩn bị mọi thứ cho chuyến đi chưa?”*

*“Có bạn bè đi cùng không?”*

*“Đã giải quyết xong công việc chưa?”*

Output sẽ trả ra kết quả có hoặc không, cùng với nhiều trường hợp quyết định việc đó. Những trường hợp đó sẽ là các và trọng số ở đây mang nghĩa là độ ưu tiên,ví dụ như = 0.3, = 0.45, = 0.08, = 0.9.

Với đầu ra và đầu vào là dạng nhị phân, chúng ta rất khó để điều chỉnh một lượng nhỏ đầu vào để đầu ra thay đổi chút ít, nên chúng ta sẽ mở rộng chúng ra trong khoảng [0, 1], được tính bởi công thức sigmoid  *f* :

*f =* ( với *z* là )

Hàm *sigmoid* là một hàm kích hoạt phi tuyến tính, giống như các hàm *tanh, ReLU, …* Hàm kích hoạt phải là một hàm phi tuyến tính, vì nếu là tuyến tính thì khi kết hợp với các phép toán tuyến tính thì đầu ra cũng là một thao tác tuyến tính, lúc này phép tính sẽ trở nên vô nghĩa.

Đây là một số hàm kích hoạt phi tuyến tính như đã nêu trên:



Hình 4‑2

Kiến trúc của một mạng neural, còn được gọi là các preceptron đa tầng, gồm có ba kiểu tầng: tầng dữ liệu đầu vào (input layer), tâng dữ liệu đầu ra (output layer) và tầng thể hiện tính logic của mạng (hidden layer). Một mạng neural chỉ có duy nhất một tầng đầu vào và một tầng đầu ra, nhưng lại có một hoặc nhiều các tầng ẩn.

Đầu ra của mạng neural trả ra bao nhiêu node, phụ thuộc vào bài toán đang xử lý. Đối với bài toán phân tích cảm xúc, kết quả quan tâm là tính phân cực của đoạn văn gồm tích cực (positive) và tiêu cực (negative), nên lúc này, tầng output sẽ trả ra hai node.



Hình 4‑3

## 4.2 Mạng neural tích chập (CNN):

Các hệ thống xử lý ảnh như các trang mạng xã hội Facebook, hay trang tìm kiếm hàng đầu Google đã đưa và ứng dụng của họ những chức năng thông minh như nhận diện khuôn mặt, phát hiện thông tin người dùng nhờ hình ảnh, … những thành tựu trên được gọi là Thị giác máy tính (Computer Vision). Mạng CNN là một trong những mô hình học sâu được xây dựng để xử lý hình ảnh, bên cạnh đó, CNN còn được dùng để xử lý giọng nói, văn bản, âm thanh, …

Mạng CNN (Convolution Neural Network) được gọi là mạng tích chập do nó sử dụng một biểu thức toán tích chập, CNN là mạng neural xử lý dữ liệu theo dạng lưới, trượt trên các ma trận.



Hình 4‑4

Mạng CNN là tập hợp của nhiều lớp chồng lên nhau, gồm có các lớp Convolution, lớp Pooling, lớp Fully-Connected, … sau đó kết hợp với các hàm kích hoạt phi tuyến tính (activate function) để tạo ra các thông tin trừu tượng hơn cho tầng tiếp theo. Các tầng liên kết với nhau thông qua cơ chế tích chập, như tên gọi của mạng này, tầng tiếp theo sẽ là kết quả tích chập từ tầng trước đó, mỗi tầng được áp đặt các bộ lọc (filters) khác nhau.



Hình 4‑5

Như đã nêu trên, mạng CNN không chỉ có thể xử lý trong lĩnh vực hình ảnh, chúng ta cũng có thể áp dụng vào các bài toán về phân loại văn bản (text classification).

Đối với xử lý hình ảnh, đầu vào của mạng là các điểm ảnh (pixels), các bộ lọc sẽ trượt qua các điểm ảnh này. Còn trong xử lý văn bản, đầu vào sẽ là các mệnh đề, các câu văn được biểu diễn dưới dạng một ma trận, mỗi dòng của ma trận sẽ ứng với một từ, cũng có thể là một ký tự, các bộ lọc sẽ trượt qua tất cả các dòng của ma trận. Mỗi dòng của ma trận chính là một vector đại diện cho một từ, thông thường các vector này được biểu diễn ở mức thấp (low-dimensional) như word2vec hay glove.

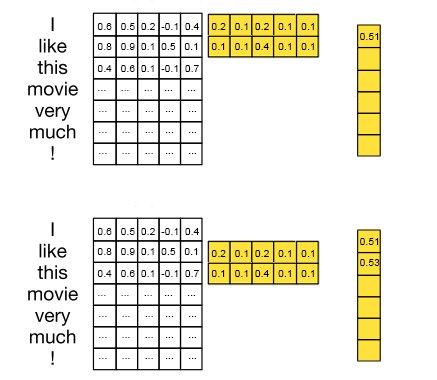
Dưới đây là sơ đồ hoạt động của mạng CNN áp dụng vào bài toán xử lý ngôn ngữ tự nhiên:



Hình 4‑6

Giả sử đầu vào của sơ đồ trên là một câu *“I like this movie very much !”*, gồm 7 từ nên ma trận có 7 dòng bao gồm cả dấu câu, ở đây tác giả chọn chiều vector là 5, nên ta có được ma trận 7x5.

Mô hình chia ra ba vùng với kích thước từ là 2, 3 và 4, mỗi vùng áp dụng 2 bộ lọc (filters).



Hình 4‑7

Các bộ lọc sẽ trượt qua ma trận theo dòng. Giá trị đầu tiên, bộ lọc (kích thước 2 từ) được thể hiện bằng ma trận 2x5 màu vàng bên trên, trượt qua 2 vector từ đầu tiên là *“I”* và *“like”* . Kết quả của phần tử đầu tiên trong output màu vàng bên phải là tổng của các phép nhân phần tử của 2 hàng đầu tiên với 2 hàng của ma trận 2x5. Nó thực hiện như sau:

0.51 = 0.6 x 0.2 + 0.5 x 0.1 + 0.2 x 0.2 + ... + 0.5 x 0.1 + 0.1 x 0.1

Và tiếp theo bộ lọc sẽ trượt qua hai từ *“like”* và *“this”*, và kết quả phần tử thứ hai sẽ là:

0.53 = 0.8 x 0.2 + 0.9 x 0.1 + 0.1 x 0.2 + … + -0.1 x 0.1 + 0.7 x 0.1

Cứ như vậy, bộ lọc màu vàng sẽ lùi xuống một dòng sau khi tích chập hai từ trước đó, vậy nên ma trận bên phải có kích thước là 6 x 1. Để bảo đảm giá trị của map đặc trưng, chúng ta cần sử dụng một activation function như ReLU. Áp dụng ReLU vẫn cho chúng ta ma trận có kích thước là 6 x 1.

Kết quả là ta có 6 output là 2 map đặc trưng của mỗi vùng. Lưu ý rằng kích thước của map đặc trưng phụ thuộc vào ma trận văn bản và ma trận của bộ lọc. Nói cách khác, kích thước của map đặc trưng không cố định, để đưa map đặc trưng này về kích thước như nhau, hoặc trong nhiều trường hợp người ta chỉ muốn giữ lại các đặc trưng **tiêu biểu.**Chúng ta có thể sử dụng lớp max-pooling để lấy ra các giá trị lớn nhất trong map đặc trưng. Điều này giúp giảm chiều dữ liệu, tăng tốc độ tính toán.

Trong ví dụ trên, tác giả sử dụng *1-max-pooling* để lấy giá trị lớn nhất trong từng map đặc trưng, việc này giúp cho output có cùng kích cỡ. Sau khi áp dụng 1-max pooling, chúng ta đã có những vector có kích thước cố định là 1×1 của 6 thành phần(bằng số bộ lọc). Sau đó thực hiện việc kết hợp tất cả các đặc trưng ở bước trước lại. Đưa ra một vector đặc trưng cuối cùng để đưa vào fully-connected để giải quyết việc phân loại, thường chúng ta sẽ qua một lớp softmax để đưa output về xác suất các nhãn có tổng bằng 1.

## 4.3 Mạng neural quy hồi (RNN):

Mạng RNN (Recurrent neural network) hay còn gọi là mạng neural quy hồi, RNN đã được áp dụng trong nhiều bài toán như phân tích giọng nói, mô tả hình ảnh, đặc biệt là trong lĩnh vực NLP (xử lý ngôn ngữ tự nhiên).

RNN lấy ý tưởng từ việc sử dụng các chuỗi thông tin. Đối với các mạng neural thường thì các thông tin đầu vào và đầu ra độc lập, không liên kết với nhau, những mô hình này không thích hợp với các bài toán dự đoán từ tiếp theo. Mạng RNN thì khác, chúng có khả năng để nhớ các thông tin trước đó, vì các tác vụ của đầu vào là kết quả đầu ra của tác vụ trước đó. Trên lý thuyết, mô hình RNN có thể sử dụng được thông tin của một văn bản dài, nhưng trên thực tế, mô hình truyền thống RNN chỉ có thể nhớ được một vài bước trước đó.



Hình 4‑8

Hình trên miêu tả sự triển khai của mô hình RNN, đây là một chuỗi mạng neural tuần tự. Với là tham số đầu vào tại thời điểm *t* , Với đầu vào là một câu như *“Hello everyone”* thì là một vector tương ứng với từ thứ hai là *“everyone”*. là trạng thái ẩn tại thời điểm *t*, nó cũng chính là **bộ nhớ** của mạng. Theo hình mô tả, được tính dựa trên các trạng thái ẩn trước nó và đầu vào tại bước đó, công thức như sau:

=  *f* ( *Uxt + Wst – 1*)

Hàm *f* là một hàm kích hoạt phi tuyến tích như *tanh* hay *ReLU.* Đối với phần tử đầu tiên trong chuỗi thì trạng thái ẩn được tính dựa trên trạng thái ẩn trước đó , phần tử này từ được khởi tạo với giá trị bằng 0. *U, V* và *W* là các ma trận tham số. Đầu ra tại thời điểm *t* là , được thể hiện bằng .

Như đã đề cập trước đó, mạng RNN truyền thống có khả năng ghi nhớ thông tin và áp dụng việc dự đoán trong các loại văn bản ngắn, nhưng với những văn bản dài, điều này không thể vì sự phụ thuộc gần xa (long-term dependencies), điều này xảy ra do tính mất mát đạo hàm khi khoảng cách càng lớn.

### 4.3.1 Lan truyền ngược liên hồi (BPTT – backpropagation through time):

Công thức cơ bản của RNN có dạng như sau:

=  *f* ( *Uxt + Wst – 1*)

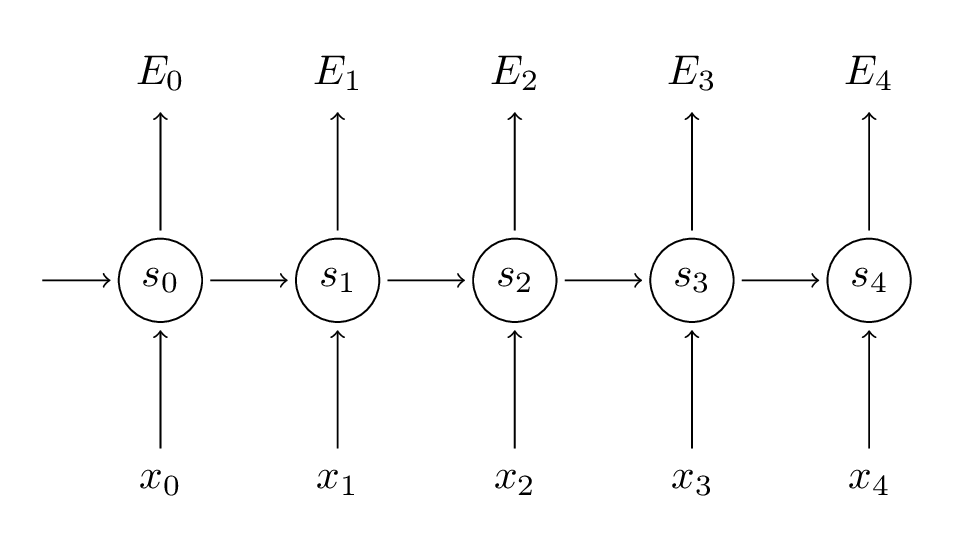
.

Ở đây, thay đổi ký hiệu từ thành để thống nhất với tài liệu sẽ tham chiếu tới. Ta cũng sẽ định nghĩa hàm mất mát (hàm lỗi) dạng *cross entropy* như sau:

= *-* log( )

*E* (*y,* ) = = -

là từ chính xác tại thời điểm *t*, còn chính là từ mà ta sẽ dự đoán. Vì mỗi chuỗi đầy đủ (một câu) là một mẫu huấn luyện, nên tổng số lỗi chính là tổng của tất cả các lỗi xuất hiện ở mỗi bước (mỗi từ).



Hình 4‑9

Mục đích của ta là tính đạo hàm lỗi với tham số U, V, W tương ứng và sau đó học các tham số này bằng cách sử dụng thuật toán SGD (Stochastic Gradient Descent), đây là thuật toán giúp ta tính đạo hàm của hàm mất mát dựa trên chỉ một điểm dữ liệu rồi cập nhật lại tại một thời điểm. Tương tự việc cộng tổng các lỗi, ta cũng sẽ cộng tổng các đạo hàm tại mỗi bước cho mỗi mẫu huấn luyện:

Để tính đạo hàm, ta dùng quy tắc chuỗi vi phân, quy tắc được dùng cho việc truyền ngược lỗi của giải thuật lan truyền ngược. Ta sử dụng làm ví dụ:

= = () \otimes 

Trong đó, = và \otimes  là biểu thức nhân ngoài của 2 vector. Qua phép triển khai này, ta thấy rằng chỉ phụ thuộc vào các giá trị ở bước hiện tại: , ,. Ta cũng thấy rằng tính đạo hàm cho *V* chỉ đơn giản là phép nhân ma trận.

Nhưng với *W* và *U* thì phép tính lại khá phức tạp hơn.

=

Với = *tanh* ( *Ux3 + Ws2* ) phụ thuộc vào *s2* mà *s2*  lại phụ thuộc và *W* và *s1*,…Vì vậy với *W*, ta không thể xem *s2* là hằng số để tính toán như với *V.* Ta phải áp dụng quy tắc chuỗi:

=

Như vậy, đối với tham số W, ta phải cộng tất cả các đầu ra ở các bước trước để tính được đạo hàm. Nói cách khác, ta phải truyền ngược đạo hàm từ *t = 3* về tới *t = 0.*



Hình 4‑10

### 4.3.2 Vấn đề mất mát đạo hàm (vanishing gradient problem):

Đối với câu văn dài, RNN không thể liên kết được các từ cách xa nhau nên việc học các câu dài sẽ dễ bị thất bị. Ta cùng xem công thức đạo hàm trên:

=

Ở đây, cũng tuân theo quy tắc chuỗi đạo hàm. Ví dụ : . Các thành phần trong công thức đều là các vector vì đạo hàm cho vector cũng là vector, nên kết quả sẽ là một ma trận, trong đấy các phần tử tương ứng sẽ được tính theo phép toán *pointwise* với đạo hàm tương ứng.

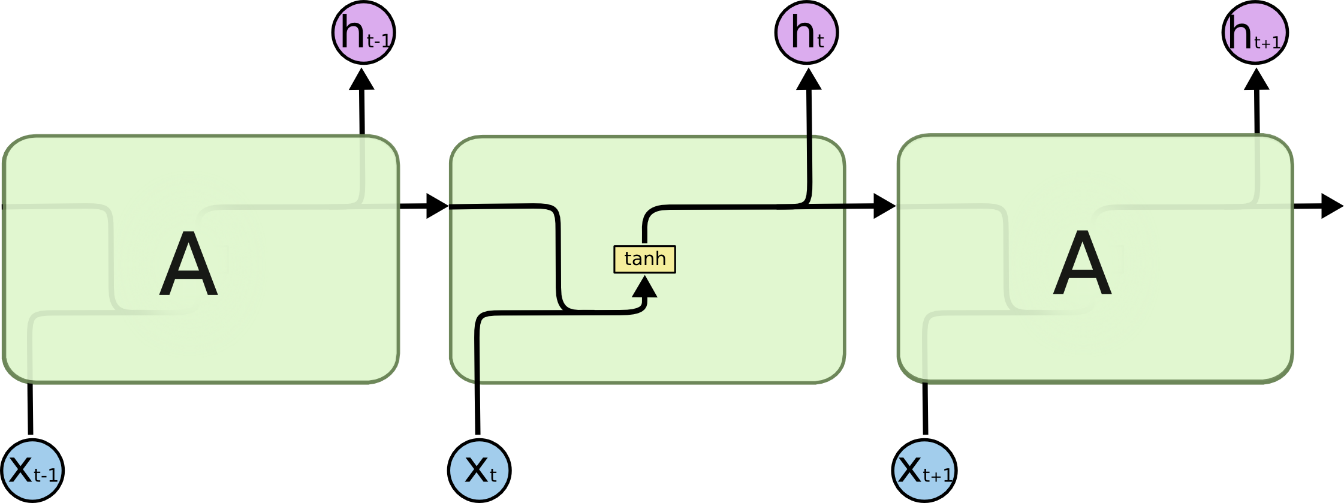
< *WIP … >*

## 4.4 Mạng neural phụ thuộc gần xa (LSTM : Long-Short Term Memory):

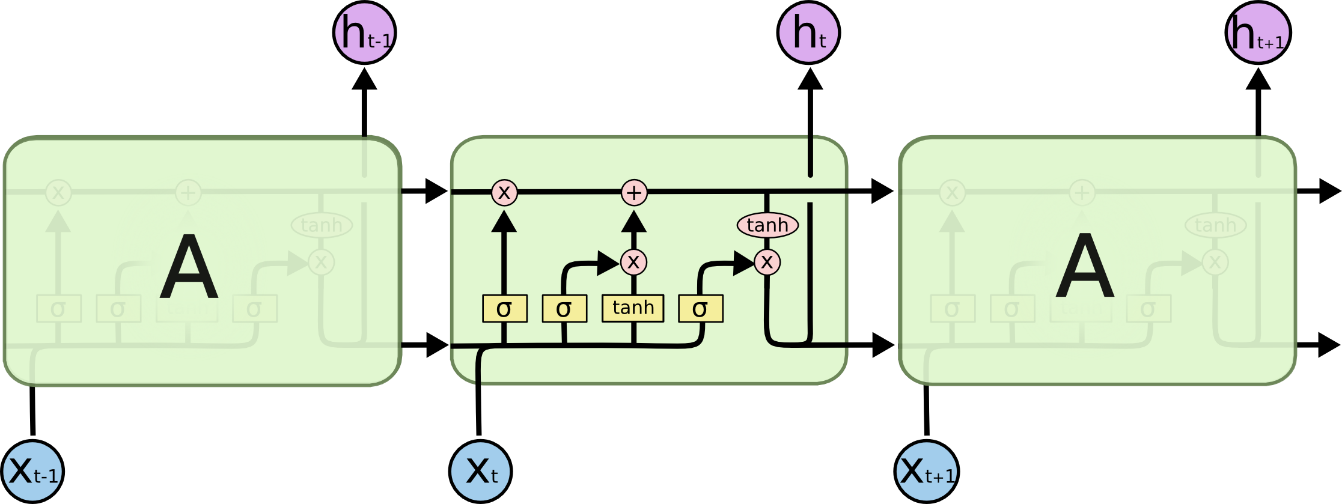
Đối với mạng RNN, ta không thể học được khi câu văn quá dài do tính mất mát đạo hàm đã nói trên. Một phương pháp phổ biến được sử dụng để khắc phục điểm này, đó là mạng LSTM (Long-Short Term Memory), đây là kiến trúc mạng nhớ ngắn-dài cùng cơ chế cổng GRU(Gated Recurrent Unit). Vào năm 1997, LSTM lần đầu được đề xuất bởi Hochreiter và Schmidhuber và giờ nó là một trong những mô hình mạng phổ biến nhất trong lĩnh vực xử lý ngôn ngữ tự nhiên. Đối với GRU, mới được giới thiệu 2014, nó là phiên bản đơn giản hóa của LSTM. Cả hai kiến trúc đó đều là biến thể của RNN, được thiết kế nhằm tránh vấn đề mất mát đạo hàm và hiệu quả cao trong việc học các phụ thuộc xa.

Như đã đề cập, mạng LSTM có kiến trúc tương tự như RNN, chỉ khác nhau ở cách tính toán tại trạng thái ẩn. Ý tưởng để giải quyết vấn đề đạo hàm bị triệt tiêu dần là chia vector trạng thái ẩn thành hai nửa: nửa đầu là các ô nhớ (cell-memory) để lưu giữ trí nhớ và nửa kia là bộ nhớ hoạt động (working memory).

Đây là mô hình của mạng RNN truyền thống, với một hàm phi tuyến tính trong trạng thái ẩn:



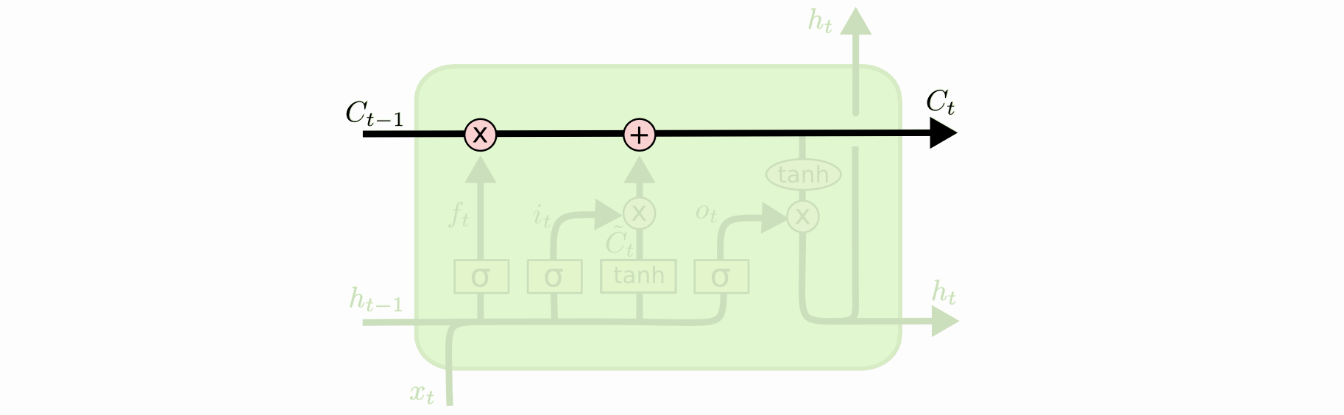
LSTM cũng tương tự, nhưng trong trạng thái ẩn lại có 4 tầng tương tác với nhau:



Đây là một vài ký hiệu được sử dụng trong sơ đồ trên :



* Ô màu vàng là các hàm phi tuyến tính, với sơ đồ trên thì là hàm *tanh* và *sigmoid*.
* Ô màu hồng là các phép toán như cộng vector, nhân vector.
* Mũi tên là các vector.
* Mũi tên gọp lại biểu diễn việc kết hợp.
* Mũi tên rẽ nhánh chỉ nội dung được sao chép và rẽ sang nhánh khác.
* **Cách thức hoạt động của LSTM**

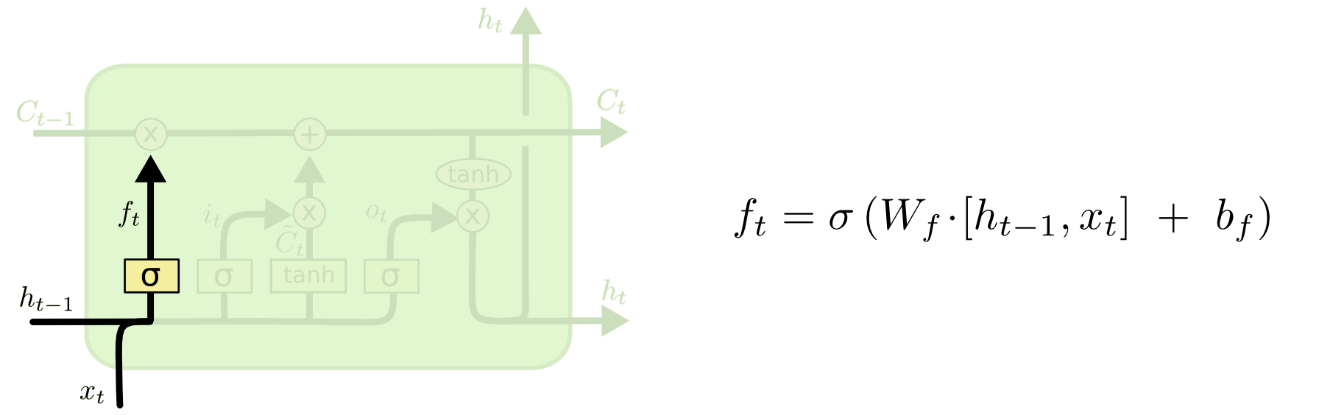


Đường trên cùng trong sơ đồ chỉ trạng thái tế bào (cell state) của LSTM. Đường này kết nối các nút mạng với nhau. Đầu vào của node này là kết quả đầu ra của node trước đó. Biểu thức “x” ở đây là phép nhân của vector đầu vào với “cổng quên” (forget gate).



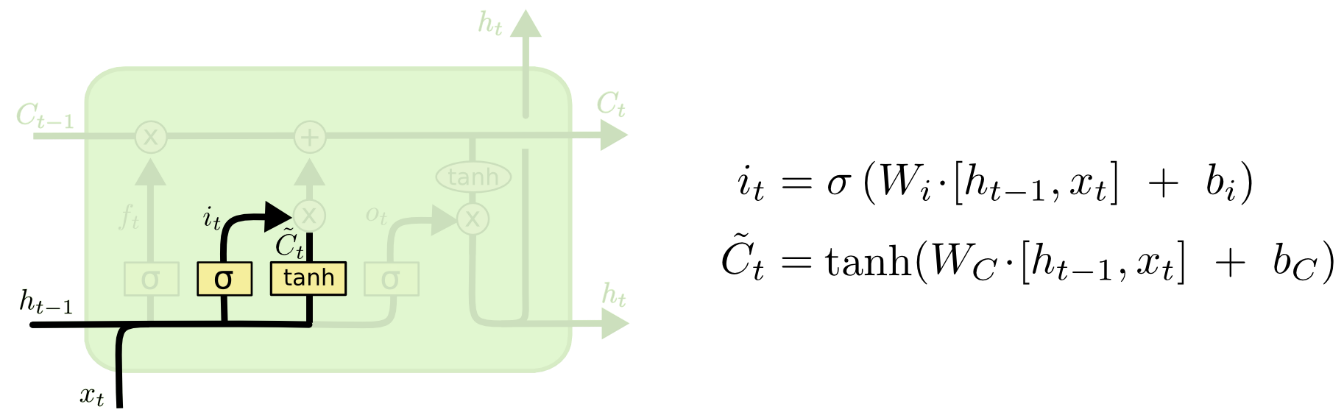
Cổng quên giúp ta lược bỏ những thông tin không cần thiết ở trạng thái trước đó. Hàm *sigmoid* có khoảng giá trị từ [0, 1]. Khi đem nhân hàm *sigmoid* với vector đầu vào, nếu giá trị của hàm *sigmoid* là 0 thì tích sẽ là 0, lúc này có nghĩa là không có thông tin nào qua cả, còn nếu giá trị là 1 thì tất cả thông tin đi qua nó.

LSTM hoạt động gồm 3 cổng như vậy, đó là cổng vào (input gate), cổng ra (output gate) và cổng quên (forget gate).



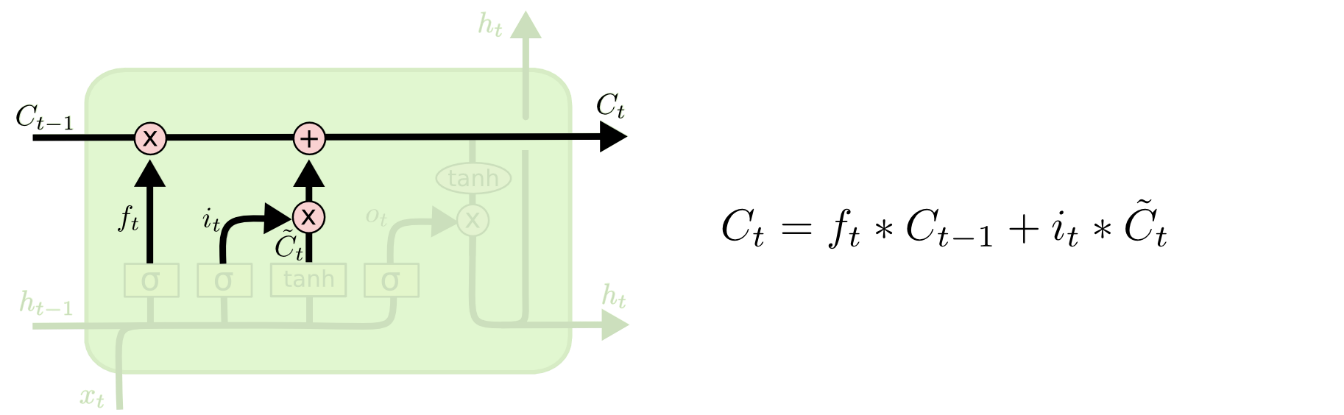
Đầu tiên, mô hình LSTM sẽ quyết định việc thông tin nào cần qua như đã nói trên. Nó sẽ lấy đầu vào là trạng thái ẩn trước đó và input tại thại điểm *t* hiện tại là . Lúc này đưa vào hàm *sigmoid* và trả ra kết quả trong khoảng [0, 1] cho mỗi trạng thái tế bào trước đó , ở đây là bias (độ lệch).

Trường hợp cổng quên được sử dụng như với giới tính của nhân vật, khi đề cập tới một người khác thì ta sẽ “quên” giới tính của nhân vật trước đó đã lưu.

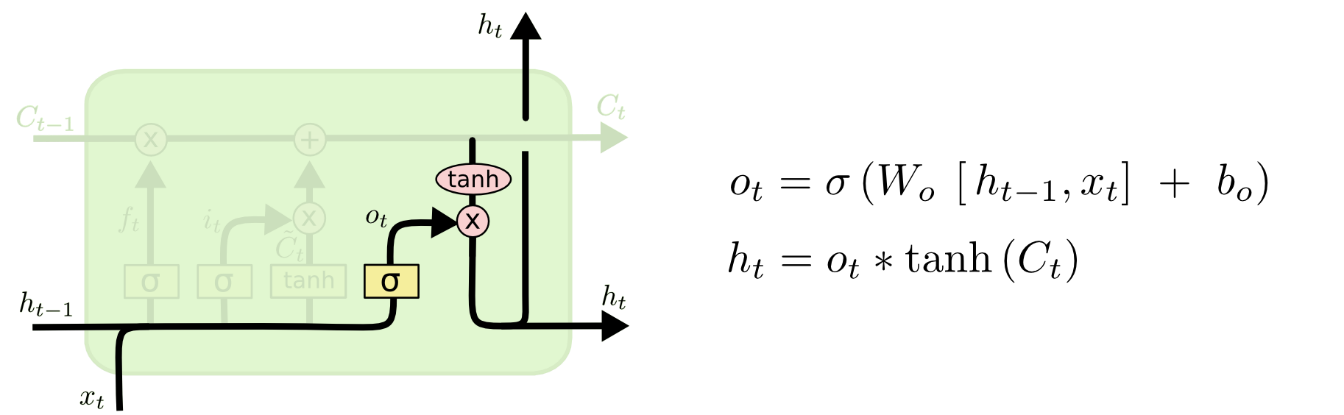


Bước tiếp theo là quyết định những thông tin gì sẽ được lưu vào trạng thái tế bào. Quá trình này chia ra 2 phần, đầu tiên là sử dụng hàm *sigmoid* để quyết định thông tin sẽ được vào ở “cổng vào”, cổng vào cũng giống như cổng quên, tham số trong hàm *sigmoid* là trạng thái ẩn trước đó và input tại thời điểm *t*. Phần thứ hai là tầng *tanh* để tạo ra giá trị mới để thêm vào trạng thái. Sau đó chúng ta kết hợp 2 giá trị trên để tạo ra một cập nhật cho trạng thái.

Ví dụ như ta muốn thêm một giới tính của nhân vật mới này vào trạng thái tế bào hiện tại và thay thế giới tính cũ của nhân vật trước đó.



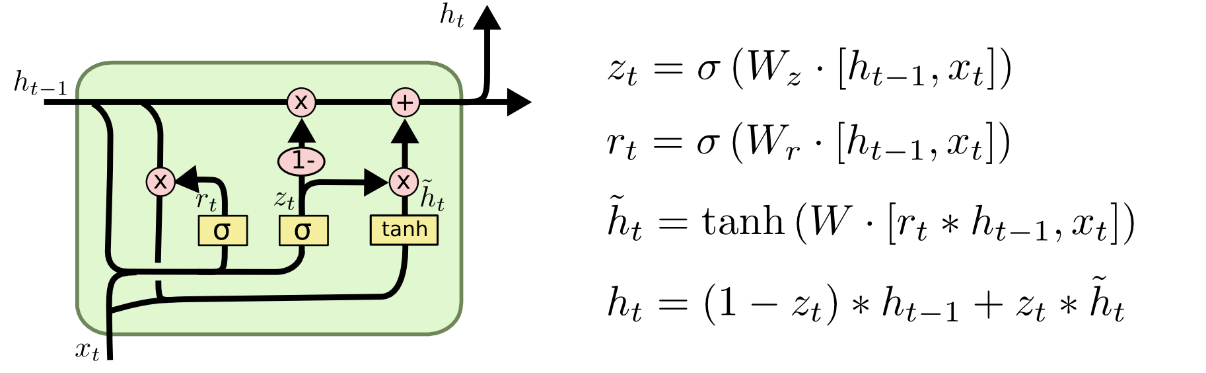
Tiếp theo, ta cập nhật trạng thái tế bào cũ thành trạng thái mới là . LSTM sẽ nhân lại kết quả của cổng vào và sau đó cộng vào tích của và .



Cuối cùng, mô hình sẽ quyết định đầu ra là những thông tin gì bằng cổng ra. Cũng như cổng vào và cổng quên, hàm *sigmoid* sẽ quyết định giá trị trong khoảng [0, 1], tiếp theo nó sẽ đưa trạng thái tế bào qua một hàm *tanh* để có giá trị trả về là [-1, 1], đây là giải pháp cho vấn đề mất mát đạo hàm vì giá trị 0 ở giữa -1 và 1 giúp dữ liệu chạy dài thêm. Sau cùng là thực hiện phép nhân với cổng ra để cho ra giá trị đầu ra mong muốn.

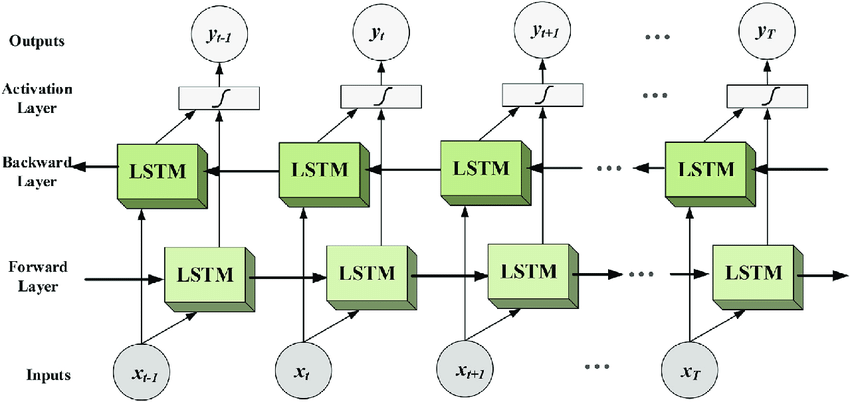
* **Cơ chế cổng của LSTM (GRU – Gated Recurrent Unit):**

GRU (Gated Recurrent Unit) được giới thiệu bởi Cho. Et al. vào 2014. Nó là sự kết hợp các cổng quên và cổng vào thành cổng “cập nhật” (update gate), nó kết hợp trạng thái tế bào và trạng thái ẩn với nhau tạo ra một thay đổi khác. Kết quả là mô hình của ta sẽ đơn giản hơn mô hình LSTM chuẩn và ngày càng trở nên phổ biến. Dưới đây là sơ đồ của GRU:



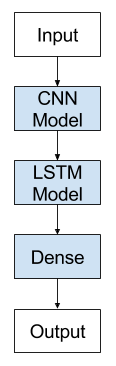
* **BiLSTM (Bidirectional Long-Short Term Memory):**

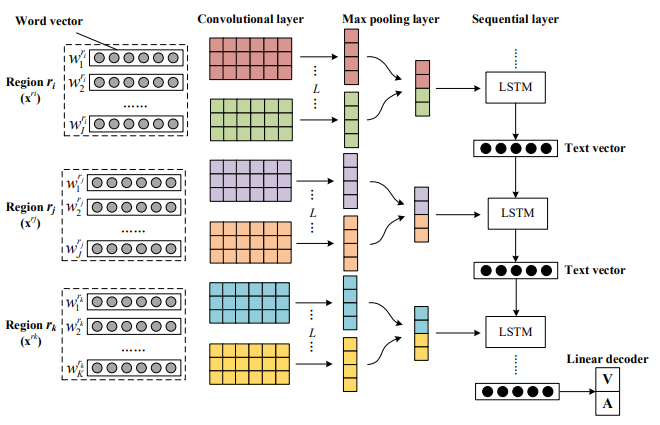
Mô hình LSTM 2 chiều, đây là một biến thể của mô hình LSTM, được giới thiệu bởi Graves và Schmidhuber vào năm 2005 (đoạn cite: Alex Graves and Jurgen Schmidhuber. 2005. Frame- ¨ wise phoneme classification with bidirectional lstm and other neural network architectures. Neural Networks, 18(5-6):602–610.). Mô hình này huấn luyện 2 mô hình LSTM, LSTM thứ hai tương tự cái đầu tiên nhưng sẽ có chiều ngược lại, để chúng ta khai thác được những tính năng nổi trội của câu văn đầu vào cả quá khứ và tương lai, tại một thời điểm xác định.



* **Mô hình kết hợp CNN-LSTM:**

Đây là mô hình kết hợp giữa hai mô hình CNN và LSTM để hỗ trợ cho việc dự đoán chuỗi trong thời gian trực quang và ứng dụng trong mô tả văn bản từ các chuỗi hình ảnh. CNN có khả năng trích xuất thông tin nổi trội nhưng lại phụ thuộc vào vấn đề gần xa, LSTM sẽ khắc phục điểm đó. Cấu trúc của mô hình này như sau:





## Mô hình đề cử trong bài toán phân tích cảm xúc:

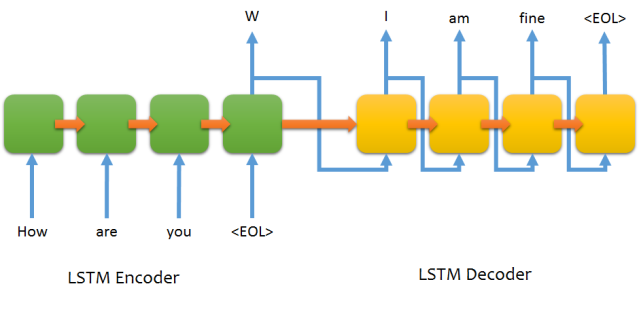
Đối với các bài toán về phân loại văn bản (text classification) như dịch máy, chatbot, phân tích cảm xúc, tóm tắt văn bản … có những kiến trúc được áp dụng như vec2seq, seq2vec và seq2seq.

Vec2seq (vector to sequence), đây là mô hình đưa vào những vector, đầu ra trả về một đoạn văn bản, thường áp dụng cho việc mô tả hình ảnh đầu vào. Seq2vec (sequence to vector), đầu vào là các đoạn văn, đầu ra là các vector. Seq2seq (sequence to sequence) thường được áp dụng nhiều trong dịch máy hay chatbot, đưa vào những đoạn văn bản và kết quả đưa ra là một đoạn văn bản khác. Trong đề tài này, chúng ta sẽ tìm hiểu về kiến trúc seq2seq.

### Mô hình seq2seq (sequence to sequence):

Đây là mô hình gồm hai mạng neural thành phần được gọi là bộ mã hóa (encoder) và bộ giải mã (decoder), đầu vào và đầu ra không nhất thiết phải chung số lượng giống nhau. Encoder sẽ mã hóa đoạn văn sang vector có độ dài cố định, decoder sẽ giải mã vector sang văn bản bằng việc sinh ra từng từ trong chuỗi đầu ra cho đến khi gặp từ khóa báo kết thúc câu như EOL (end of line).

Trong mô hình seq2seq, ta có thể sử dụng các kiểu kiến trúc mạng phổ biến như CNN hoặc RNN.

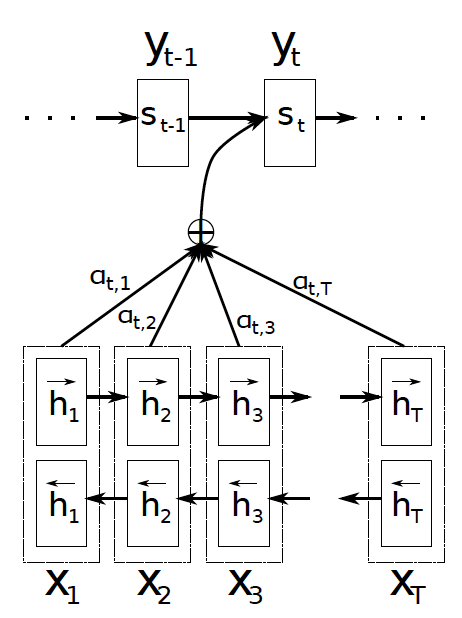


Mô hình seq2seq cơ bản có nhược điểm là cần bộ RNN decoder sử dụng toàn bộ thông tin mã hóa từ chuỗi đầu vào cho dù chuỗi đó ngắn hay dài, thêm vào đó RNN encoder cần phải mã hóa chuỗi đầu vào thành một vector duy nhất và có độ dài cố định.

Ràng buộc này khá hạn chế trong thực tế, việc sinh từ tại một thời điểm trong chuỗi đầu ra nhiều khi phải phụ thuộc nhiều hơn vào một số những thành phần nhất định trong chuỗi đầu vào. Đối với dịch máy, khi dịch chuỗi đầu vào từ ngôn ngữ này sang ngôn ngữ khác, chúng ta thường phải quan tâm nhiều đến ngữ cảnh của từ hiện tại so với các từ khác trong câu.

### 4.5.2 Cơ chế Attention trong mô hình seq2seq:

Một kỹ thuật được giới thiệu bởi Bahdanau vào năm 2014, cơ chế Attention, sẽ giải quyết vấn đề trên. Thay vì chuỗi đầu vào được mã hóa thành một vector duy nhất, RNN encoder mã hóa thành một dãy các vector, sau đó RNN decoder áp dụng cơ chế Attention mềm (soft Attention) để lấy tổng có trọng số của dãy các vector mã hóa. Các trọng số này được tính bằng một mạng neural truyền thẳng.



Nghĩa của từ Attention là tập trung, trong deep learning, cơ chế Attention này sẽ tập trung vào những phần nhất định của dữ liệu đầu vào tại một thời điểm. Context vector là vector được tạo ra từ trọng số của các encoder output tại thời điểm thứ *j*, với là trọng số biểu thị mức độ cần được ‘attention’ của từng trạng thái ẩn của encoder:

Với :

Tham số *T* là tổng các timesteps của encoder, context vector sẽ tạo ra một sequence *s* mới, trong đó phụ thuộc vào và ở điểm output thứ *t – 1.*

được gọi là *aligment score*, khi đưa trạng thái ẩn của của decoder và trạng thái ẩn của encoder và một hàm , hàm này được gọi là *aligment model.* Với mục đích đánh giá từ tại vị trí *j* (encoder) và từ output tại vị trí *i* (decoder) bằng việc gán một trọng số .

Aligment model có dạng :

Đơn giản hóa bằng hàm *softmax* để tổng attention score = 1, ta thu được .

Ngoài aligment model trên, ta cũng có thể sử dụng các model khác sau:

* + Content-based Attention:
  + General Attention:
  + Dot Product:

Tóm lại, cơ chế Attention giúp mô hình có thể tập trung vào các phần dữ liệu quan trọng, bằng việc tạo ra một aligment model *a* giúp tính các aligment score để tái thiết lập trọng số “reweight” các hidden state của encoder.

# THỰC NGHIỆM VÀ ĐÁNH GIÁ

## 5.1. Dataset

|  |  |
| --- | --- |
| Dataset | Source |
| Book | GVHD |
| DVD | GVHD |
| Electronics | GVHD |
| Kitchen | GVHD |
| IMDB movie reviews | Dataset Keras |

**Tập dữ liệu cho 4 lĩnh vực khác nhau**

Chúng ta có 4 tập dữ liệu cho 4 lĩnh vực khác nhau lần lượt là: Book, DVD, Electronics, Kitchen. Với mỗi tập dữ liệu trên được chia làm 2 file positive (tích cực) và negative (tiêu cực). Mỗi file bao gồm 1000 file text được đánh số thứ tự từ 000 đến 999 có nội dung là các bình luận tích cực hoặc tiêu cực dựa trên thư mục đã phân loại. Sau đó chia dữ liệu thành 2 phần là Train gồm 1800 mẫu và Test gồm 200 mẫu để tiến hành huấn luyện và chạy thực nghiệm.

**Tập dữ liệu IMDB Movie Keras**

Bộ dữ liệu gồm 25.000 đánh giá phim từ IMDB, được gắn nhãn bởi 2 loại cảm xúc (tích cực / tiêu cực). Các đánh giá đã được xử lý trước và mỗi đánh giá được mã hóa dưới dạng một chuỗi các chỉ mục từ (số nguyên). Để thuận tiện, các từ được lập chỉ mục theo tần số chung trong tập dữ liệu, do đó, ví dụ số nguyên "3" mã hóa từ xuất hiện thường xuyên thứ 3 trong tập dữ liệu.

## 5.2. Cấu hình máy và thư viện cài đặt

- Operating System: Windows 10 Pro 64-bit

- Processor: Intel® Core™ i7-5700HQ CPU @ 2.70GHz (8 CPUs), ~ 2.7GHz

- RAM: 16GB

- GPU: NVIDIA GeForce GTX 950M

- PyCharm 2018.2.4 (Community Edition)

- Python == 3.6

- Keras == 2.2.4

- Pandas == 0.23.4

- BeautifulSoup == 4.6.3

- Numpy == 1.15.4

- Matplotlib == 3.0.2

## 5.3. Thực nghiệm mô hình RNN-Attention-Network:

### **5.3.1. Xử lý dữ liệu:**

* Với tập dữ liệu GVHD cung cấp:

**- Bước 1: Chia dữ liệu thành tập Train và tập Test**

Chúng ta sẽ sử dụng 100 đánh giá tích cực và 100 đánh giá tiêu cực cuối cùng để làm bộ kiểm tra (gồm 200 đánh giá) và 1800 đánh giá còn lại làm tập dữ liệu để huấn luyện. Ở đây chúng ta chia 90% dữ liệu để huấn luyện và 10% để kiểm tra. Việc phân chia có thế được thực hiện dễ dàng bằng cách sử dụng tên tệp của các đánh giá. Trong đó, các đánh giá có tên 000 đến 899 dành cho dữ liệu huấn luyện và các đánh giá có tên từ 900 trở đi là để kiểm tra.

**- Bước 2: Loading and Cleaning Reviews**

Tách các token trên khoảng trắng.

Xóa tất cả các dấu câu khỏi từ.

Xóa tất cả các ký tự không thuộc bảng chữ cái.

Loại bỏ tất cả các từ được biết đến là stopword (am, is, are, he, she, it,...).

Loại bỏ tất cả các từ có độ dài nhỏ hơn hoặc bằng 1 ký tự.

**- Bước 3: Xác định từ vựng**

Việc xác định từ vựng của các từ đã biết khi sử dụng mô hình Bag of Words hoặc mô hình Embedding khá là quan trọng. Càng nhiều từ càng thể hiện được độ lớn của các tài liệu. Do đó, điều quan trọng là hạn chế các từ được tin tưởng có thể dùng để dự đoán được. Điều này rất khó để biết trước và thường rất quan trọng để kiểm tra các giả thuyết khác nhau về cách xây dựng một bộ từ vựng hữu ích.

Ở Bước 2 chúng ta đã biết cách loại bỏ các dấu câu và chữ số khỏi tập từ vựng. Chúng ta có thể lặp lại điều này cho tất cả các dữ liệu còn lại và xây dựng một bộ từ vựng gồm các từ đã biết.

Dựa vào đó chúng ta có thể phát triển từ vựng dưới dạng bộ đếm (Counter). Đó là một bộ từ điển ánh xạ các từ và số lượng của chúng cho phép chúng ta dễ dàng cập nhật và truy vấn. Sau đó chúng ta có thể lướt qua bộ từ vựng và loại bỏ tất cả các từ có tỷ lệ xuất hiện thấp, chẳng hạn như các từ chỉ được sử dụng một hoặc hai lần trong tất cả các đánh giá. Cuối cùng, bộ từ vựng có thể được lưu vào một tệp mới gọi là vocab.txt mà sau này chúng ta có thể tải và sử dụng để lọc các đánh giá trước khi mã hóa chúng cho việc khởi tạo mô hình.

* Với tập dữ liệu IMDB Keras:

**- Bước 1: Import dữ liệu từ thư viện Keras**

Để sử dụng được dữ liệu IMDB Movíe reviews sentiment classification của Keras trước hết phải import dữ liệu từ thư viện Keras qua lệnh:

**from** keras.datasets **import** imdb

**- Bước 2: Load và xử lý dữ liệu**

Để load chia dữ liệu thành các tập huấn luyện (train) và kiểm tra (test) ta dùng lệnh:

(x\_train,y\_train),(x\_test,y\_test)=imdb.load\_data(**num\_words**=max\_features)

Hàm load\_data được Keras cung cấp để load và chia dữ liệu với tập huấn luyện và kiểm tra mỗi tập gồm 25000 mẫu. Với tham số **num\_words** là một số nguyên hoặc không có (none) cho biết những từ thường xuyên xuất hiện nhất để xem xét. Trong đó, **x\_train, x\_test** là một danh sách các chuỗi, là danh sách các chỉ mục (số nguyên). Nếu tham số **num\_words** được xác định, giá trị tối đa của chỉ mục là **num\_words-1**; **y\_train,y\_test** là danh sách các nhãn bằng số nguyên (1 hoặc 0).

Để đảm bảo rằng tất cả các chuỗi trong danh sách x\_train,x\_test có đều có cùng độ dài chúng ta dùng hàm **pad\_sequences:**

x\_train = **sequence.pad\_sequences**(x\_train, **maxlen**=maxlen)

x\_test = **sequence.pad\_sequences**(x\_test, **maxlen**=maxlen)

Theo mặc định, điều này được thực hiện bằng cách đệm 0 ở đầu mỗi chuỗi cho đến khi mỗi chuỗi có cùng độ dài với chuỗi dài nhất. Ví dụ:

>>>pad\_sequences([[1, 2 , 3], [3, 4, 5, 6], [7, 8]])

array([[0, 1, 2, 3],

[3, 4, 5, 6],

[0, 0, 7, 8]], dtype=int32)

[3, 4, 5, 6] là chuỗi dài nhất, vì vậy 0 sẽ được đệm vào các chuỗi khác để độ dài của chúng phù hợp với [3, 4, 5, 6]. Nếu muốn pad đến cuối chuỗi có thể đặt thêm tham số **padding=’post’**. Nếu muốn chỉ định độ dài tối đa của mỗi chuỗi, chúng ta có thể sử dụng **maxlen** để xác định. Điều này sẽ cắt ngắn tất cả các chuỗi dài hơn **maxlen.**

>>>pad\_sequences([[1, 2 , 3], [3, 4, 5, 6], [7, 8]], **maxlen**=3)

array([[1, 2, 3],

[4, 5, 6],

[0, 7, 8]], dtype=int32)

### **5.3.2 Xây dựng mô hình dựa trên RNN-Attention-network**

Nhóm xây dựng mô hình RNN-Attention-network xử lý bài toán phân tích cảm xúc dựa theo mô hình của tác giả Yang năm 2016. Dưới đây là mô hình mà nhóm đã áp dụng trong phần thực nghiệm:

[w(1), w(2), … , w(t)

W(t) = t’th word in a given review

X = [x(1),x(2),...,x(n)], x(t) = We(w(t))

Word Embedding Layer

h(t) = RNN(x(t), h(t-1))

RNN Layer

u(t) = tanh(Ww.h(t) + bw)

α(t) = softmax(u(t).uw)

s = sum(α(t) \* h(t))

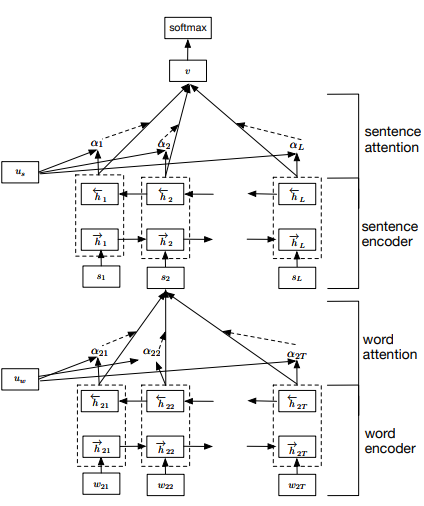
Attention Layer

output = softmax(wo.s)

Softmax Layer

[0,1]

Theo tác giả Yang cấu trúc mô hình Attention ông nghiên cứu là Mạng Attention phân cấp (Hierarchical Attention Networks). Tổng thể của kiến trúc Hierarchical Attention Networks (HAN) được thể hiện trong hình:



Hình 5‑1. Hierarchical Attention Network

Kiến trúc trên bao gồm nhiều phần khác nhau: một bộ mã hóa chuỗi từ, một lớp word-level attention, một bộ mã hóa câu và lớp sentence-level attention.

**GRU-based mã hóa chuỗi:**

GRU sử dụng cơ chế cổng để theo dõi trạng thái của chuỗi mà không cần sử dụng cái cell nhớ riêng biệt. Có hai loại cổng: cổng cài đặt lại (reset gate) và cổng cập nhật (update gate). Chúng cùng nhau kiểm soát cách thông tin được cập nhật cho trạng thái. Tại thời điểm t, GRU tính toán trạng thái mới theo công thức:



Đây là phép nội suy tuyến tính giữa trạng thái trước đó và trạng thái mới hiện tại  đã được tính toán với trình tự thông tin mới. Cổng quyết định bao nhiêu thông tin trong quá khứ được lưu giữ và bao nhiêu thông tin mới được thêm vào. được cập nhật dưới dạng:



Với là vector chuỗi ở thời điểm t. Trạng thái ứng cử  được tính theo cách tương tự như mạng nơ-ron hồi quy (Recurrent Neural Network - RNN):



Ở đây, là cổng cài đặt lại kiểm soát bao nhiêu trạng thái trước đó đóng góp cho trạng thái ứng cử. Nếu bằng 0, thì nó sẽ quên trạng thái trước đó. Cổng cài đặt lại được cập nhật như sau:



**Word Encoder**

Ở đây, nhóm sử dụng trực tiếp nhúng từ (word embeddings). Để mô hình hoàn chỉnh hơn chúng ta có thể sử dung GRU để có được vectơ từ trực tiếp từ các ký tự. Nhóm dùng một lớp Embedding để đơn giản hóa mô hình hơn.

**Word Attention**

Không phải tất cả các từ đều góp phần như nhau để thể hiện ý nghĩa của câu. Do đó, cơ chế attetion được dùng để trích xuất những từ quan trọng đối với ý nghĩa của câu văn và tổng hợp các đặc trưng-hình thái đó tạo thành một vectơ câu. Đặc biệt,







Đó là, trước tiên chúng ta đưa ghi chú từ vào one-layer MLP để có , biểu diễn ngầm định của , sau đó chúng ta đo lường tầm quan trọng của từ cũng như độ tương đồng của bằng một vector ngữ cảnh theo bậc của từ , và lấy được kết quả là một trọng số có độ quan trọng đã được chuẩn hóa bằng hàm softmax. Sau đó, chúng ta tính tổng các trọng số của các ghi chú từ, kết quả được ghi nhận là một vector câu vector ngữ cảnh có thể được xem như là một đại diện bậc cao của truy vấn "từ chỉ thông tin là gì" qua các từ, được dùng trong trong các mạng bộ nhớ. Vector ngữ cảnh từ (từ vựng) được khởi tạo ngẫu nhiên trong suốt quá trình huấn luyện.

### 5.3.3 Thực nghiệm RNN-Attention-network với dữ liệu Book

**Cấu trúc mô hình:**

Layer (type) Output Shape Param # Connected to

===========================================================

input\_1 (InputLayer) (None, 200) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

embedding\_1 (Embedding) (None, 200, 100) 1080400 input\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

gru\_1 (GRU) (None, 200, 100) 60300 embedding\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_1 (Dense) (None, 200, 1) 101 gru\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

flatten\_1 (Flatten) (None, 200) 0 dense\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

activation\_1 (Activation) (None, 200) 0 flatten\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

repeat\_vector\_1 (RepeatVector) (None, 100, 200) 0 activation\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

permute\_1 (Permute) (None, 200, 100) 0 repeat\_vector\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

multiply\_1 (Multiply) (None, 200, 100) 0 gru\_1[0][0]

permute\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

lambda\_1 (Lambda) (None, 100) 0 multiply\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_2 (Dense) (None, 1) 101 lambda\_1[0][0]

**Số lượng tham số:**

Total params: 1,140,902

Trainable params: 60,502

Non-trainable params: 1,080,400

**Score:**

|  |  |
| --- | --- |
| Time | 95 seconds |
| Error | 18.5% |
| Loss | 46.18% |
| Accuracy | 81.50% |

**Plot Result Model**



Hình 5‑2 Training and validation loss (Book)



Hình 5‑3 Training and validation accuracy (Book)

**Kết quả thực nghiệm với một số mô hình khác:**

Hình 5‑4 Biểu đồ kết quả các model (Book)

### **5.3.4 Thực nghiệm RNN-Attention-network với dữ liệu DVD**

**Cấu trúc mô hình:**

Layer (type) Output Shape Param # Connected to

===========================================================

input\_1 (InputLayer) (None, 200) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

embedding\_1 (Embedding) (None, 200, 100) 1109900 input\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

gru\_1 (GRU) (None, 200, 100) 60300 embedding\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_1 (Dense) (None, 200, 1) 101 gru\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

flatten\_1 (Flatten) (None, 200) 0 dense\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

activation\_1 (Activation) (None, 200) 0 flatten\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

repeat\_vector\_1 (RepeatVector) (None, 100, 200) 0 activation\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

permute\_1 (Permute) (None, 200, 100) 0 repeat\_vector\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

multiply\_1 (Multiply) (None, 200, 100) 0 gru\_1[0][0]

permute\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

lambda\_1 (Lambda) (None, 100) 0 multiply\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_2 (Dense) (None, 1) 101 lambda\_1[0][0]

**Số lượng tham số:**

Total params: 1,170,402

Trainable params: 60,502

Non-trainable params: 1,109,900

**Score**

|  |  |
| --- | --- |
| Time | 138 seconds |
| Error | 18% |
| Loss | 45.64% |
| Accuracy | 82% |

**Plot Result Model**



Hình 5‑5 Training and validation loss (DVD)



Hình 5‑6 Training and validation accuracy (DVD)

**Kết quả thực nghiệm với mô hình khác:**

Hình 5‑7 Biểu đồ kết quả các model (DVD)

### **5.3.5 Thực nghiệm RNN-Attention-network với dữ liệu Electronics**

**Cấu trúc mô hình:**

Layer (type) Output Shape Param # Connected to

===========================================================

input\_1 (InputLayer) (None, 200) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

embedding\_1 (Embedding) (None, 200, 100) 538900 input\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

gru\_1 (GRU) (None, 200, 100) 60300 embedding\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_1 (Dense) (None, 200, 1) 101 gru\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

flatten\_1 (Flatten) (None, 200) 0 dense\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

activation\_1 (Activation) (None, 200) 0 flatten\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

repeat\_vector\_1 (RepeatVector) (None, 100, 200) 0 activation\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

permute\_1 (Permute) (None, 200, 100) 0 repeat\_vector\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

multiply\_1 (Multiply) (None, 200, 100) 0 gru\_1[0][0]

permute\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

lambda\_1 (Lambda) (None, 100) 0 multiply\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_2 (Dense) (None, 1) 101 lambda\_1[0][0]

**Số lượng tham số:**

Total params: 599,402

Trainable params: 60,502

Non-trainable params: 538,900

**Score**

|  |  |
| --- | --- |
| Time | 99 seconds |
| Error | 18.5% |
| Loss | 41.67% |
| Accuracy | 81.50% |

**Plot Result Model**



Hình 5‑8 Training and validation loss (Electronics)



Hình 5‑9 Training and validation accuracy (Electronics)

**Kết quả thực nghiệm với mô hình khác:**

Hình 5‑10 Biểu đồ kết quả các model (Electronics)

### **5.3.6 Thực nghiệm RNN-Attention-network với dữ liệu Kitchen**

**Cấu trúc mô hình:**

Layer (type) Output Shape Param # Connected to

===========================================================

input\_1 (InputLayer) (None, 200) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

embedding\_1 (Embedding) (None, 200, 100) 488200 input\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

gru\_1 (GRU) (None, 200, 100) 60300 embedding\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_1 (Dense) (None, 200, 1) 101 gru\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

flatten\_1 (Flatten) (None, 200) 0 dense\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

activation\_1 (Activation) (None, 200) 0 flatten\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

repeat\_vector\_1 (RepeatVector) (None, 100, 200) 0 activation\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

permute\_1 (Permute) (None, 200, 100) 0 repeat\_vector\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

multiply\_1 (Multiply) (None, 200, 100) 0 gru\_1[0][0]

permute\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

lambda\_1 (Lambda) (None, 100) 0 multiply\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_2 (Dense) (None, 1) 101 lambda\_1[0][0]

===========================================================**Số lượng tham số:**

Total params: 548,702

Trainable params: 60,502

Non-trainable params: 488,200

**Score**

|  |  |
| --- | --- |
| Time | 145 seconds |
| Error | 19% |
| Loss | 53.46% |
| Accuracy | 81% |

**Plot Result Model**



Hình 5‑11 Training and validation loss (Kitchen)



Hình 5‑12 Training and validation accuracy (Kitchen)

**Kết quả thực nghiệm với mô hình khác:**

Hình 5‑13 Biểu đồ kết quả các model (Kitchen)

### **5.3.7 Thực nghiệm RNN-Attention-network với dữ liệu IMDB Keras**

**Cấu trúc mô hình:**

Layer (type) Output Shape Param # Connected to

===========================================================

input\_1 (InputLayer) (None, 200) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

embedding\_1 (Embedding) (None, 200, 128) 640000 input\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

gru\_1 (GRU) (None, 200, 100) 68700 embedding\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_1 (Dense) (None, 200, 1) 101 gru\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

flatten\_1 (Flatten) (None, 200) 0 dense\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

activation\_1 (Activation) (None, 200) 0 flatten\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

repeat\_vector\_1 (RepeatVector) (None, 100, 200) 0 activation\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

permute\_1 (Permute) (None, 200, 100) 0 repeat\_vector\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

multiply\_1 (Multiply) (None, 200, 100) 0 gru\_1[0][0]

permute\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

lambda\_1 (Lambda) (None, 100) 0 multiply\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_2 (Dense) (None, 1) 101 lambda\_1[0][0]

===========================================================

**Số lượng tham số:**

Total params: 708,902

Trainable params: 708,902

Non-trainable params: 0

**Score:**

|  |  |
| --- | --- |
| Time | 31.9 minutes |
| Error | 13% |
| Loss | 38.03% |
| Accuracy | 87% |

**Plot Result Model**

Hình 5‑14 Training and validation loss (IMDB)

****

Hình 5‑15 Training and validation accuracy (IMDB)

**Kết quả thực nghiệm với mô hình khác:**

Hình 5‑16 Biểu đồ kết quả các model (IMDB)

### **5.3.8 So sánh các mô hình**

Sau khi chạy thực nghiệm với các tập dữ liệu trên, ta nhận thấy rằng tỉ lệ val\_acc khi chạy thực nghiệm bằng mô hình RNN-Attention-network có tỉ lệ đúng cao nhất so với các mô hình khác (CNN, CNN-LSTM, Bi-LSTM). Tuy nhiên, thời gian chạy của mô hình RNN-Attention-network so với các mô hình khác chiếm thời gian nhiều hơn.

Trong quá trình chạy thực nghiệm, nhóm đã thay đổi các tham số như batch\_size, số lượng epochs, maxlen (độ dài tối đa của mỗi chuỗi) khi dùng hàm pad\_sequence để thu được kết quả tốt nhất. Tuy nhiên, tùy vào tập dữ liệu mà nhóm dùng cũng ảnh hưởng đến kết quả khi chạy mô hình do số lượng đánh giá của mỗi tập dữ liệu là không giống nhau.

Nhìn chung khi chạy thực nghiệm với số lượng epoch giống nhau trên các mô hình khác nhau thì mô hình RNN-Attention-network có độ ổn định cao hơn các mô hình khác không bị overfit. Khi tiến hành chạy dự đoán một đoạn văn để phân tích xem đoạn văn đó là tiêu cực hay tích cực thì mô hình RNN-Attention-network mang lại tính chính xác cao hơn.

# KẾT LUẬN

1. Những vấn đề đã làm được

2. Những vấn đề chưa làm được

3. Hướng phát triển

# TÀI LIỆU THAM KHẢO