Lecture 4: Mô Hình Chuỗi

Giới thiệu

Chào các bạn, hôm nay chúng ta sẽ tìm hiểu về mô hình chuỗi trong Xử lý Ngôn ngữ Tự nhiên. Trước tiên, tôi sẽ trình bày lý do tại sao chúng ta cần mô hình chuỗi, các loại mô hình chuỗi khác nhau, bao gồm mạng nơ-ron hồi tiếp (recurrent neural networks), mạng tích chập (convolutional networks) và cơ chế chú ý (attention).

NLP and Sequential Data

Mô hình chuỗi trong NLP rất quan trọng vì ngôn ngữ chứa đầy dữ liệu tuần tự. Dữ liệu này có thể là các từ trong câu, các ký tự trong từ, cho đến các câu trong một đoạn văn hay tài liệu. Nó cũng có thể bao gồm nhiều tài liệu theo thời gian, nhiều bài đăng trên mạng xã hội, và nhiều dạng khác. Nói chung, có rất nhiều chuỗi trong NLP.

Long-distance Dependencies in Language

Một điểm quan trọng mà tôi đã đề cập trước đây là sự phụ thuộc xa trong ngôn ngữ. Ví dụ, có sự đồng nhất về số và giới tính. Để tạo ra một mô hình ngôn ngữ mạch lạc, bạn cần xử lý sự đồng nhất này. Chẳng hạn, nếu bạn nói "he does not have very much confidence in", thì từ tiếp theo phải là "himself". Ngược lại, nếu bạn nói "she does not have very much confidence in", thì từ tiếp theo phải là "herself". Sự đồng nhất về giới tính này không phổ biến trong tiếng Anh, nhưng lại rất thường gặp trong các ngôn ngữ khác như tiếng Pháp và nhiều ngôn ngữ khác trên thế giới.

Ngoài ra, còn có những yếu tố như sở thích lựa chọn (selectional preferences). Ví dụ, trong câu "The rain has lasted as long as the life of the queen" và "The rain has lasted as long as the life of the clouds", trong tiếng Anh Mỹ, cách duy nhất để biết từ nào xuất hiện trước là bạn cần có ý tưởng ngữ nghĩa rằng chúng phải đồng nhất với nhau theo một cách nào đó.

Can be Complicated!

Còn có nhiều loại kiến thức thực tế khác mà bạn cần phải giữ trong các ngữ cảnh dài, và điều này có thể rất phức tạp. Một ví dụ thú vị là câu: "the trophy would not fit in the brown suitcase because it was too big". Ở đây, "it" ám chỉ cái gì? Đáp án là "the trophy". Nhưng nếu câu là "the

trophy would not fit in the brown suitcase because it was too small", thì "it" lại ám chỉ "the suitcase".

Đây được gọi là thách thức Winograd Schema. Winograd schema là một loại thách thức ngôn ngữ, trong đó bạn tạo ra hai ví dụ có sự khác biệt tối thiểu nhưng câu trả lời lại khác nhau. Những ví dụ này rất hữu ích để kiểm tra khả năng của các mô hình ngôn ngữ.

Types of Sequential Prediction Problems

Types of Prediction: Binary, Multi-class, Structured

Trong phần này, chúng ta sẽ khám phá các loại vấn đề dự đoán tuần tự. Đầu tiên, chúng ta đã đề cập đến phân loại nhị phân và đa lớp, đó là khi chúng ta phân loại giữa hai lớp hoặc nhiều lớp khác nhau. Tuy nhiên, còn một loại dự đoán khác gọi là dự đoán cấu trúc (structured prediction). Dự đoán cấu trúc xảy ra khi bạn có một số lượng nhãn rất lớn, không phải là một số hữu hạn.

Ví dụ, nếu bạn nhận vào một đầu vào và muốn dự đoán từ loại của tất cả các từ trong đầu vào đó, và giả sử có khoảng 50 loại từ loại khác nhau, số lượng nhãn cho mỗi câu sẽ là bao nhiêu? Đó sẽ là mọi tổ hợp của các từ loại cho từng từ. Cụ thể, nếu có 50 lựa chọn cho mỗi từ, và giả sử có 60 từ, số lượng tổ hợp sẽ là 50 mũ 60. Điều này trở nên không khả thi.

Ý tưởng cơ bản của dự đoán cấu trúc là không giống như mô hình ngôn ngữ, nơi bạn dự đoán toàn bộ chuỗi cùng một lúc. Thay vào đó, bạn thường dự đoán từng phần tử một và sau đó tính toán xác suất có điều kiện của phần tử tiếp theo dựa trên phần tử hiện tại hoặc các yếu tố khác. Đây là cách mà chúng ta giải quyết các vấn đề dự đoán cấu trúc.

Types of Prediction: Unconditioned vs. Conditioned

Trong lĩnh vực dự đoán, chúng ta thường phân biệt giữa hai loại dự đoán: dự đoán không có điều kiện (unconditioned prediction) và dự đoán có điều kiện (conditioned prediction). Dự đoán không có điều kiện không phải là phương pháp phổ biến, nhưng nó liên quan đến việc dự đoán xác suất của một biến đơn lẻ hoặc tạo ra một biến đơn lẻ $P(X)$. Ngược lại, dự đoán có điều kiện là quá trình dự đoán xác suất của một biến đầu ra dựa trên một biến đầu vào nhất định $P(Y|X)$.

Cụ thể, trong dự đoán không có điều kiện, chúng ta sẽ tập trung vào việc xác định xác suất của một biến mà không cần xem xét các yếu tố khác. Trong khi đó, với dự đoán có điều kiện, chúng ta sẽ xem xét mối quan hệ giữa biến đầu vào và biến đầu ra.

Types of Unconditioned Prediction

Trong NLP, một trong những phương pháp quan trọng để thực hiện dự đoán từ là sử dụng các mô hình tự hồi tiếp từ trái sang phải. Những mô hình này cho phép chúng ta xây dựng các mô hình ngôn ngữ mà không có giới hạn về ngữ cảnh, tức là chúng có thể xem xét toàn bộ chuỗi

từ trước đó. Tuy nhiên, trong thực tế, việc sử dụng các mô hình này gặp khó khăn do số lượng tham số quá lớn và tính phân tán cao, khiến cho việc ước lượng tham số trở nên khó khăn.

Left-to-right Autoregressive Prediction: $P(X) = \prod_{i=1}^{|X|} P\left(x_i | x_1, \ldots, x_{i-1}\right)$

Thay vào đó, chúng ta thường sử dụng các mô hình n-gram, chẳng hạn như mô hình trigram, trong đó chúng ta giới hạn độ dài ngữ cảnh. Cụ thể, khi dự đoán từ, mô hình sẽ không tham chiếu đến tất cả các đầu ra trước đó. Ví dụ, khi dự đoán từ thứ nhất, chúng ta không điều kiện hóa trên bất kỳ từ nào sau nó. Khi dự đoán từ thứ hai, chúng ta điều kiện hóa trên từ thứ nhất, và khi dự đoán từ thứ ba, chúng ta điều kiện hóa trên từ thứ hai. Tương tự, khi dự đoán từ thứ tư, chúng ta sẽ điều kiện hóa trên từ thứ ba và thứ hai, nhưng không phải từ thứ nhất. Đây chính là cách hoạt động của một mô hình trigram.

Left-to-right Markov Chain (order n-1): $P(X) = \prod_{i=1}^{|X|} P\left(x_i | x_{i-n+1}, \ldots, x_{i-1}\right)$

Ngoài ra, xem xét các mô hình dự đoán độc lập, như mô hình unigram, trong đó không điều kiện hóa trên bất kỳ ngữ cảnh trước đó nào. Một mô hình khác là dự đoán hai chiều, trong đó khi dự đoán mỗi phần tử, chúng ta dựa vào tất cả các phần tử khác, không phải phần tử hiện tại. Điều này có thể được thực hiện thông qua các mô hình ngôn ngữ được che giấu (masked language models).

Independent Prediction: $P(X) = \prod_{i=1}^{|X|} P\left(x_i\right)$

Bidirectional Prediction: $P(X) = \prod_{i=1}^{|X|} P\left(x_i | x_{\neq i}\right)$

Tuy nhiên, cần lưu ý rằng để có một xác suất hợp lệ, chúng ta cần dự đoán các phần tử dựa trên tất cả các phần tử đã được dự đoán trước đó, và không thể dự đoán dựa trên các phần tử tương lai. Do đó, mô hình này không thực sự là một mô hình xác suất, nhưng đôi khi nó được sử dụng để học các biểu diễn có thể được áp dụng cho các tác vụ khác sau này.

Types of Conditioned Prediction

Khi nói về xác suất có điều kiện, chúng ta thường đề cập đến các mô hình dự đoán có điều kiện, nơi mà một nguồn dữ liệu đầu vào X được sử dụng để tính toán xác suất của một đầu ra khác.

Ví dụ: Auto-regressive conditioned prediction: $P(Y|X) = \prod_{i=1}^{|X|} P\left(y_i | X, y_1, \ldots, y_{i-1}\right)$

Điều này có thể được áp dụng trong các sequence-to-sequence models hoặc trong các mô hình ngôn ngữ, nơi mà người dùng cung cấp một prompt và mô hình sẽ dự đoán đầu ra tiếp theo, như cách mà chúng ta thường làm với Chat GPT.

Ngoài ra, còn có một khái niệm khác là "non-auto-regressive conditioned prediction", thường được áp dụng trong các tác vụ như gán nhãn chuỗi (sequence labeling) hoặc dịch máy không tự hồi tiếp (non-auto-regressive machine translation).

Non-autoregressive Conditioned Prediction: $P(Y|X) = \prod_{i=1}^{|X|} P\left(y_i|X\right)$

Basic Modeling Paradigm: Extract Features → Predict

Một trong những mô hình cơ bản mà chúng ta thường sử dụng là phương pháp trích xuất đặc trưng và dự đoán. Mô hình này tương tự như mô hình "bag of words". Cụ thể, trong mô hình "bag of words", chúng ta trích xuất các đặc trưng từ văn bản đầu vào và dựa vào những đặc trưng này để đưa ra dự đoán.

Khi thực hiện mô hình chuỗi, phương pháp trích xuất đặc trưng sẽ có sự khác biệt. Đầu vào là văn bản X, từ đó chúng ta trích xuất các đặc trưng H và dự đoán nhãn Y. Đối với bài toán phân loại văn bản (text classification), quy trình thường diễn ra như sau: chúng ta sử dụng một bộ trích xuất đặc trưng để chuyển đổi chuỗi văn bản thành một vector duy nhất. Dựa vào vector này, chúng ta sẽ thực hiện dự đoán.

Ngược lại, trong bài toán gán nhãn chuỗi, chúng ta sẽ trích xuất một vector cho mỗi đối tượng mà chúng ta muốn dự đoán. Ví dụ, trong bài toán gán nhãn từ loại (part of speech tagging), mỗi từ trong văn bản sẽ tương ứng với một vector riêng, và từ đó, chúng ta sẽ dự đoán thông tin cho từng từ.

An Aside: More on Sequence Labeling

Sequence Labeling

Gán nhãn chuỗi (sequence labeling) cho phép chúng ta biểu diễn nhiều vấn đề khác nhau bằng cách dự đoán một chuỗi nhãn đầu ra Y có độ dài bằng với chuỗi văn bản đầu vào X.

Một trong những ứng dụng phổ biến của gán nhãn chuỗi là part of speech tagging: chúng ta xác định từ loại cho từng từ trong câu. Ngoài ra, gán nhãn chuỗi còn được sử dụng trong lemmatization, tức là dự đoán dạng gốc của mỗi từ. Điều này rất hữu ích cho việc chuẩn hóa, chẳng hạn như khi bạn muốn tìm tất cả các trường hợp của một động từ cụ thể hoặc một danh từ cụ thể.

Cần lưu ý rằng lemmatization khác với stemming. Trong khi stemming chỉ đơn giản là cắt bỏ các hậu tố (ví dụ: cắt bỏ "S" trong danh từ số nhiều), nó không thể thực hiện các phép chuẩn hóa phức tạp như chuyển đổi "saw" thành "see".

Một khái niệm khác liên quan là morphological tagging (gán nhãn hình thái học). Đây là một phiên bản nâng cao của gán nhãn từ loại, cho phép dự đoán các thông tin chi tiết hơn như động từ quá khứ, danh từ số nhiều, hoặc dạng cụ thể của động từ. Mặc dù gán nhãn hình thái

học có thể không quá thú vị trong tiếng Anh do ngôn ngữ này có cấu trúc hình thái đơn giản, nhưng nó trở nên hấp dẫn hơn trong các ngôn ngữ phức tạp như tiếng Nhật hay tiếng Hindi.

Span Labeling

Một trong những nhiệm vụ phổ biến trong gán nhãn chuỗi là gán nhãn khoảng (span labeling). Nhiệm vụ này nhằm mục đích dự đoán các khoảng và nhãn, thường được áp dụng trong nhận diện thực thể tên (named entity recognition). Ví dụ, khi nói "Graham Nub is teaching at Carnegie Mellon University", chúng ta cần xác định các thực thể như người, tổ chức, địa điểm, hay cơ quan chính phủ, …

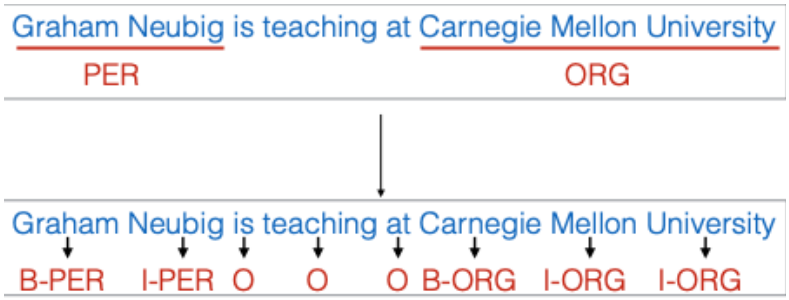
Ngoài ra, còn có các nhiệm vụ như phân đoạn cú pháp (syntactic chunking), đó là tìm kiếm tất cả các cụm danh từ và cụm động từ. Một nhiệm vụ khác là gán nhãn vai trò ngữ nghĩa (semantic role labeling), giúp xác định ai là người thực hiện hành động, hành động đó là gì, và nơi diễn ra hành động. Điều này rất hữu ích cho việc phân tích các mối quan hệ "ai làm gì với ai".

Một nhiệm vụ phức tạp hơn là liên kết thực thể (entity linking), đó là nhận diện thực thể tên và liên kết chúng với các cơ sở dữ liệu như WikiData hoặc Wikipedia. Nhiệm vụ này rất quan trọng trong các ứng dụng như tổng hợp tin tức. Ví dụ, nó giúp tìm tất cả các bài báo về một người nổi tiếng hoặc theo dõi các đề cập đến sản phẩm của công ty trên mạng xã hội.

Span Labeling as Sequence Labeling

Việc gán nhãn cho các thực thể trong văn bản có thể được coi là một dạng gán nhãn chuỗi (sequence labeling). Đó là, trong quá trình gán nhãn, chúng ta sẽ dự đoán các thẻ bắt đầu (B), bên trong (I) và không phải thực thể (O) cho từng từ hoặc khoảng không gian trong văn bản.

Ví dụ, nếu chúng ta có một đoạn văn với các thực thể, chúng ta sẽ chuyển đổi chúng thành các thẻ như sau: "begin person" cho một thực thể người, "in person" cho các từ bên trong thực thể đó, và "o" cho những từ không thuộc thực thể nào. Sau khi gán nhãn, chúng ta có thể chuyển đổi các thẻ này trở lại thành các khoảng không gian thực thể ban đầu, giúp cho việc dự đoán trở nên dễ dàng hơn.

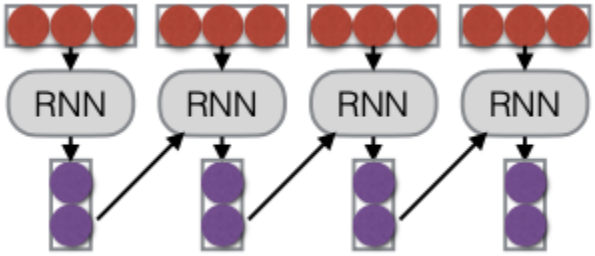


Types of Sequence Models

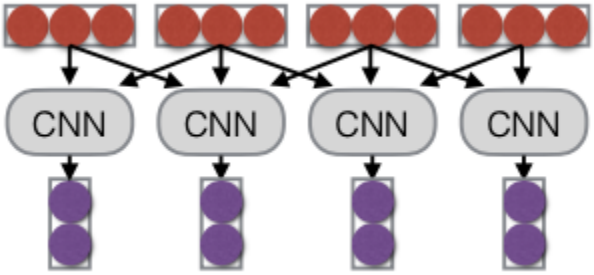
Three Major Types of Sequence Models

Trong phần này, chúng ta sẽ khám phá cách mô hình hóa các chuỗi trong học máy, với ba loại mô hình chuỗi chính.

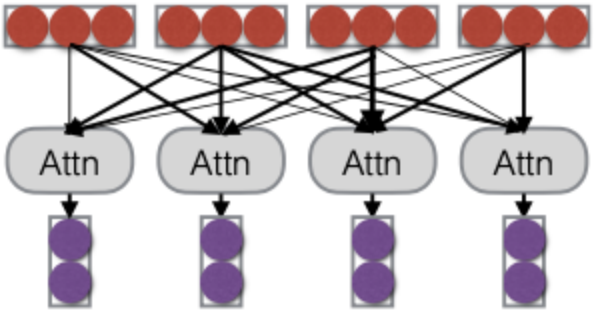
Loại đầu tiên là Mô hình hồi tiếp (Recurrent Model). Mô hình này dựa vào việc điều kiện hóa các biểu diễn dựa trên một mã hóa của lịch sử. Cách hoạt động của nó là sử dụng các vector đầu vào, thường là các word embeddings hoặc các embeddings từ lớp trước của mô hình. Một mạng nơ-ron hồi tiếp (Recurrent Neural Network - RNN) sẽ bắt đầu bằng cách tiếp nhận vector đầu vào đầu tiên, và ở mỗi bước tiếp theo, nó sẽ nhận thêm vector đầu vào và vector ẩn từ bước trước đó. Quá trình này tiếp tục cho đến khi hoàn thành chuỗi.



Loại thứ hai là Mô hình tích chập (Convolutional Model). Mô hình này điều kiện hóa các biểu diễn dựa trên ngữ cảnh cục bộ. Trong mô hình này, các đầu vào sẽ được điều kiện hóa dựa trên từ hiện tại và các từ xung quanh bên trái hoặc bên phải. Ví dụ, một phép tích chập kích thước ba sẽ xem xét từ hiện tại cùng với hai từ lân cận. Bạn cũng có thể có các phép tích chập với kích thước lớn hơn như năm hoặc bảy.



Cuối cùng, chúng ta có Mô hình chú ý (Attention Model). Mô hình này điều kiện hóa các biểu diễn dựa trên một trung bình có trọng số của tất cả các token trong chuỗi. Điều này có nghĩa là chúng ta sẽ xem xét tất cả các token trong chuỗi, nhưng mức độ điều kiện hóa cho mỗi token sẽ khác nhau. Chúng ta có thể chú ý nhiều hơn đến một token nhất định và ít hơn đến các token khác.



Một điểm quan trọng cần lưu ý là độ phức tạp tính toán của từng loại mô hình. Độ phức tạp này có thể được biểu diễn dựa trên độ dài chuỗi n, và kích thước cửa sổ tích chập W. Đối với mạng nơ-ron hồi tiếp, độ phức tạp tính toán là O(n), tức là tuyến tính. Đối với mô hình tích chập, độ phức tạp là O(nW), và đối với mô hình chú ý, độ phức tạp là O(n^2).

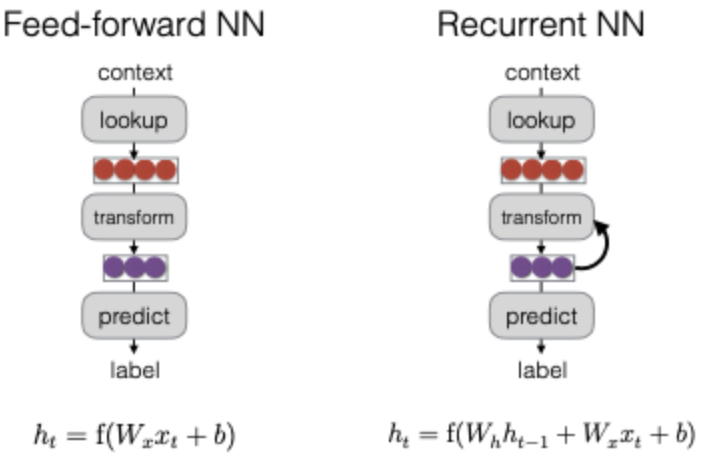
Điều này có nghĩa là đối với các chuỗi dài, mạng nơ-ron hồi tiếp có thể xử lý hiệu quả hơn, vì nó có thể chạy trên chuỗi dài lên đến 20 triệu token trong thời gian tuyến tính, miễn là có đủ bộ nhớ. Ngược lại, mô hình chú ý sẽ gặp khó khăn hơn với chuỗi dài. Tuy nhiên, mô hình chú ý và tích chập có thể được song song hóa dễ dàng, trong khi mô hình hồi tiếp thì không.

A Sequence Model: Recurrent Neural Networks

Recurrent Neural Networks (Elman 1990)

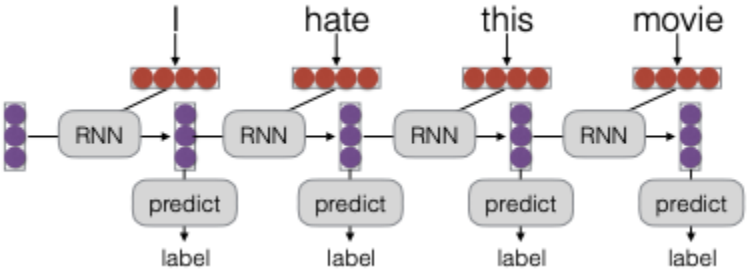
Recurrent Neural Networks (RNN) được phát minh vào khoảng năm 1990 và được thiết kế để ghi nhớ thông tin. Cách hoạt động của RNN khác với mạng nơ-ron truyền thẳng (feedforward neural network). Trong một mạng nơ-ron truyền thẳng, chúng ta thực hiện một phép biến đổi tuyến tính trên đầu vào, sau đó áp dụng một hàm phi tuyến như tanh hoặc ReLU để tạo ra một trạng thái ẩn (hidden state) và đưa ra dự đoán.

Ngược lại, trong RNN, chúng ta không chỉ sử dụng đầu vào mà còn kết hợp với trạng thái ẩn trước đó. Cụ thể, một mạng nơ-ron Elman, một dạng của RNN, sẽ nhận đầu vào cùng với trạng thái ẩn trước đó, từ đó thực hiện phép nhân với trạng thái ẩn này. Điều này cho phép RNN duy trì thông tin qua các bước thời gian, giúp nó trở thành công cụ mạnh mẽ trong việc xử lý chuỗi dữ liệu như văn bản hay âm thanh.



Unrolling in Time

Trong phần này, chúng ta sẽ khám phá cách xử lý một chuỗi dữ liệu trong mạng nơ-ron hồi tiếp. Quá trình bắt đầu với một trạng thái ban đầu, có thể là một vector toàn số không, được khởi tạo ngẫu nhiên hoặc được học từ dữ liệu. Dựa trên trạng thái này, chúng ta sẽ chạy nó qua một hàm RNN để tính toán trạng thái ẩn và từ đó đưa ra dự đoán.

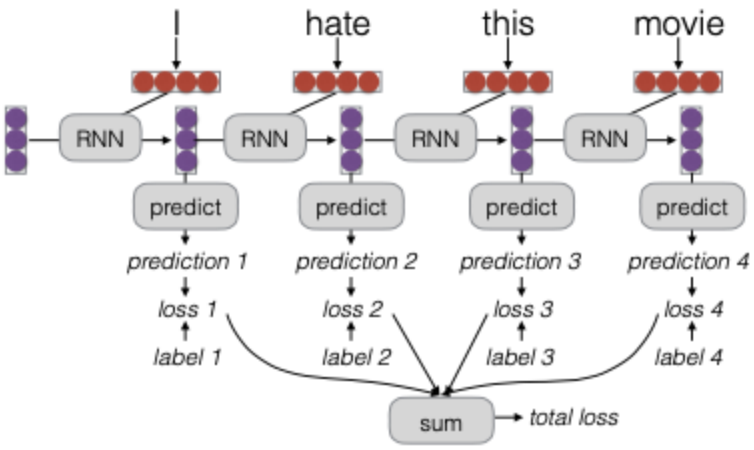


Điều quan trọng cần lưu ý là hàm RNN này là giống hệt nhau, bất kể nó xuất hiện ở vị trí nào trong chuỗi. Nhờ vào đặc điểm này, số lượng tham số của mô hình luôn không thay đổi, bất kể độ dài của chuỗi đầu vào. Đây là một yếu tố quan trọng trong việc xây dựng các mô hình chuỗi, giúp đảm bảo tính nhất quán và hiệu quả trong quá trình học.

Training RNNs

Khi huấn luyện các mạng nơ-ron hồi tiếp, chúng ta cần nhớ rằng có thể thực hiện quá trình này miễn là chúng ta có một đồ thị có hướng không chu trình (DAG) để tính toán hàm mất mát. Quá trình này bao gồm hai bước chính: lan truyền thuận (forward propagation) và lan truyền ngược (back propagation), từ đó chúng ta có thể tính toán và cập nhật các tham số của mô hình.

Giả sử chúng ta đang thực hiện nhiệm vụ gán nhãn chuỗi, trong đó mỗi dự đoán của mô hình tương ứng với một từ loại. Mỗi nhãn dự đoán là xác suất cho các từ loại trong chuỗi đó, trong khi mỗi nhãn thực tế là một nhãn từ loại đúng. Từ những dự đoán này, chúng ta sẽ tính toán negative log likelihood của nhãn từ loại, từ đó thu được một giá trị mất mát.



Nếu có nhiều dự đoán, chúng ta sẽ có nhiều giá trị loss khác nhau. Để giải quyết vấn đề này, chúng ta sẽ cộng tất cả các giá trị loss lại với nhau, tạo thành một total loss. Bây giờ, chúng ta đã có một đồ thị có hướng không chu trình với một nút cuối (terminal node), cho phép chúng ta thực hiện quá trình lan truyền ngược.

RNN Training

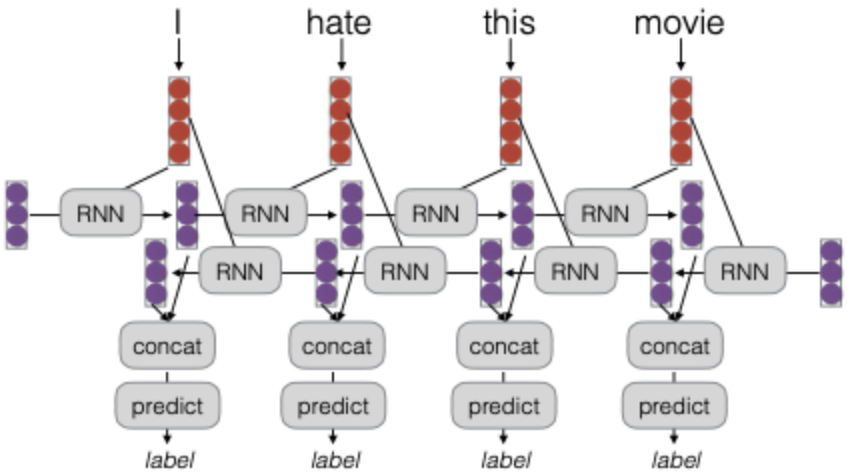
Khi đã có được giá trị loss, chúng ta có thể thực hiện quá trình lan truyền ngược. Cụ thể, chúng ta sẽ chạy lan truyền ngược và giá trị loss sẽ được truyền ngược vào tất cả các tham số trong mạng.

Một điểm quan trọng cần lưu ý là các tham số trong RNN được chia sẻ qua các bước thời gian khác nhau. Điều này có nghĩa là các đạo hàm liên quan đến các tham số sẽ được tổng hợp qua tất cả các bước thời gian. Quá trình này được gọi là "backpropagation through time" (BPTT).

Cách thức hoạt động của BPTT là các tham số của hàm RNN sẽ chỉ được cập nhật một lần, nhưng chúng sẽ nhận được thông tin từ nhiều vị trí khác nhau trong mạng. Cụ thể, các tham số này có thể được cập nhật từ bốn vị trí khác nhau trong mạng, điều này giúp tối ưu hóa quá trình học. Quá trình này không chỉ áp dụng cho RNN mà còn cho tất cả các mô hình chuỗi mà chúng ta sẽ thảo luận trong bài viết hôm nay.

Bi-RNNs

Một trong những mô hình phổ biến được sử dụng là bidirectional RNNs. Mô hình này rất hữu ích cho các tác vụ như gán nhãn chuỗi. Cách hoạt động của nó là chạy hai RNN: một từ đầu vào và một từ cuối vào, sau đó kết hợp kết quả của chúng để đưa ra dự đoán.



Một câu hỏi thường gặp là liệu việc sử dụng hai RNN có làm thay đổi độ phức tạp tính toán hay không. Thực tế, điều này không làm thay đổi độ phức tạp tiệm cận, vì việc nhân với hai không ảnh hưởng đến ký hiệu Big O. Tuy nhiên, việc này sẽ làm tăng gấp đôi thời gian thực thi.

Vanishing Gradients

Vanishing Gradients

Trong RNN, một vấn đề nổi bật mà chúng ta cần chú ý là hiện tượng "vanishing gradients" (triệt tiêu gradient). Hiện tượng này xảy ra khi chúng ta thực hiện quá trình lan truyền ngược trong RNN. Cụ thể, khi tính toán một nhiệm vụ dự đoán, chẳng hạn như hồi quy, chúng ta sẽ nhập vào một loạt các token và tính toán hồi quy ở cuối cùng bằng hàm squared loss (bình phương mất mát).

Khi sử dụng RNN tiêu chuẩn, gradient có thể lớn ở các đơn vị RNN đầu tiên, nhưng sau đó sẽ giảm dần do tác động của các hàm phi tuyến. Ví dụ, nếu chúng ta sử dụng hàm tanh, gradient của hàm này sẽ đạt cực đại ở 1 và gần như bằng 0 ở các giá trị khác. Điều này có nghĩa là nếu đầu vào nằm ở một giá trị xa, như -3, gradient sẽ gần như biến mất, gây khó khăn cho quá trình học.

Một số người có thể nghĩ rằng để khắc phục vấn đề này, chúng ta có thể nhân hàm tanh với 100 để tăng cường gradient. Tuy nhiên, điều này lại dẫn đến vấn đề ngược lại, gọi là exploding gradients (bùng nổ gradient), khi gradient tăng lên không kiểm soát và làm hỏng quá trình tối ưu hóa.

Vấn đề này không chỉ xảy ra với các hàm phi tuyến mà còn liên quan đến việc nhân ma trận trọng số và các phép biến đổi khác. Bất kỳ khi nào chúng ta thay đổi đầu vào thành đầu ra, gradient sẽ có thể lớn hơn 1 hoặc nhỏ hơn 1, dẫn đến các vấn đề về độ dốc biến mất hoặc bùng nổ.

Điều quan trọng là nếu có thông tin quan trọng trong mô hình của bạn, việc tìm cách tạo ra một con đường trực tiếp từ thông tin đó đến nơi bạn thực hiện dự đoán có thể cải thiện hiệu suất của mô hình. Ngược lại, nếu có thông tin không quan trọng, việc đặt nó xa hơn hoặc tạo ra một con đường gián tiếp sẽ giúp mô hình không bị phân tâm bởi những thông tin không cần thiết.

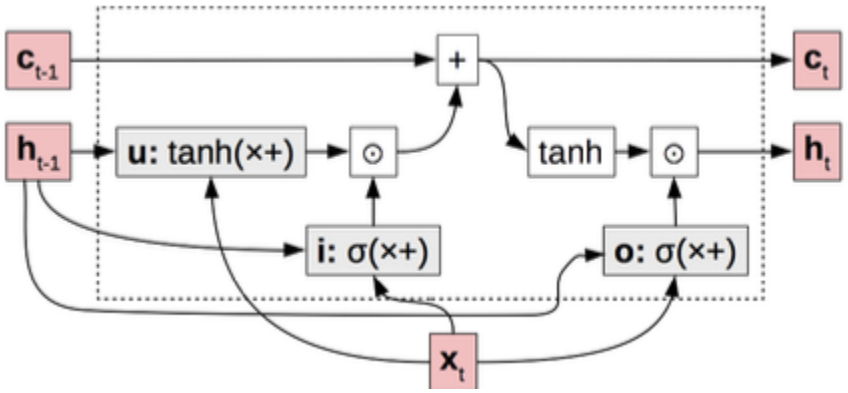
A Solution: Long Short-Term Memory

Để giải quyết triệt tiêu gradient, một trong những phương pháp hiệu quả là sử dụng mô hình Long Short-Term Memory (LSTM). Ý tưởng cơ bản của LSTM là tạo ra các kết nối cộng dồn giữa các bước thời gian, giúp duy trì thông tin qua các bước thời gian mà không bị mất đi.

Tại sao sử dụng các kết nối cộng dồn? Vì phép cộng là phép toán duy nhất không làm thay đổi gradient. Cụ thể, hàm đồng nhất được định nghĩa là $f(x) = x$, và khi lấy đạo hàm, ta luôn nhận được giá trị bằng 1. Điều này đảm bảo rằng gradient luôn được duy trì.

Trong cấu trúc của LSTM, có một thành phần gọi là memory cell (ô nhớ), được truyền đi một cách tuyến tính. Bên cạnh đó, LSTM còn có ba cổng chính:

1. Cổng cập nhật u (Update Gate): Quyết định xem có nên cập nhật trạng thái ẩn hay không.
2. Cổng đầu vào i (Input Gate): Xác định mức độ thông tin đầu vào nào sẽ được tiếp nhận.
3. Cổng đầu ra o (Output Gate): Quyết định mức độ thông tin nào từ ô nhớ sẽ được xuất ra.



Các cổng này giúp kiểm soát luồng thông tin, cho phép mô hình học cách bật hoặc tắt chúng theo nhu cầu.

Ngoài LSTM, còn có các biến thể khác như Gated Recurrent Units (GRU), đơn giản hơn nhưng vẫn dựa trên nguyên tắc kết nối cộng dồn và kiểm soát luồng thông tin. Nguyên lý này không chỉ xuất hiện trong LSTM mà còn trong nhiều kiến trúc khác nhau.

Một khái niệm quan trọng khác là "residual connections" (kết nối dư). Kết nối này cho phép thông tin được truyền từ đầu đến cuối của mạng qua nhiều lớp, giúp ngăn ngừa vấn đề gradient biến mất khi đi qua nhiều lớp. Điều này rất quan trọng trong các mô hình hiện đại như Transformer, Llama, GPT, và nhiều mô hình khác.

Cuối cùng, RNN và các mô hình tương tự được sử dụng rộng rãi trong việc mô hình hóa chuỗi dài, thường kết hợp với các mô hình dựa trên attention.

Convolution

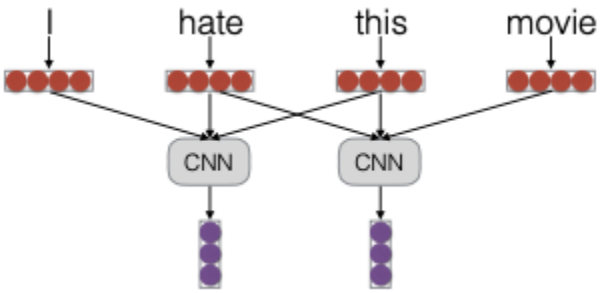
Convolution

Convolution là một kỹ thuật quan trọng trong lĩnh vực xử lý ngôn ngữ tự nhiên và xử lý hình ảnh. Lý do chính khiến convolution được áp dụng rộng rãi trong hai lĩnh vực này là do cách mà chúng xử lý thông tin.

Khi làm việc với ngôn ngữ, một câu như "this is wonderful" chỉ được chia thành ba token. Tuy nhiên, khi xử lý âm thanh, câu này sẽ được phân tích thành nhiều khung hình khác nhau. Điều này cho thấy rằng, trong ngôn ngữ, mỗi token đã mang lại một ý nghĩa ngữ nghĩa nhất định. Ngược lại, khi nhìn vào âm thanh hoặc pixel trong hình ảnh, chúng ta không thể dễ dàng nhận ra ý nghĩa ngữ nghĩa từ những đơn vị nhỏ này.

Vì vậy, convolution trở thành một công cụ hữu ích trong việc xử lý âm thanh và hình ảnh. Nó cho phép chúng ta xây dựng các mô hình convolutional không chỉ trên từ mà còn trên cả ký tự.

Vậy convolution thực sự là gì? Về cơ bản, nó liên quan đến việc lấy một cửa sổ cục bộ xung quanh một đầu vào và chạy nó qua một mô hình. Một cách đơn giản để hình dung là coi nó như một mạng nơ-ron feed forward, nơi chúng ta có thể kết hợp tất cả các vector xung quanh và chạy chúng qua một phép biến đổi tuyến tính. Ví dụ, chúng ta có thể kết hợp các vector như x_{t-1}, x_t, x_{t+1} để tạo ra đầu ra mới.

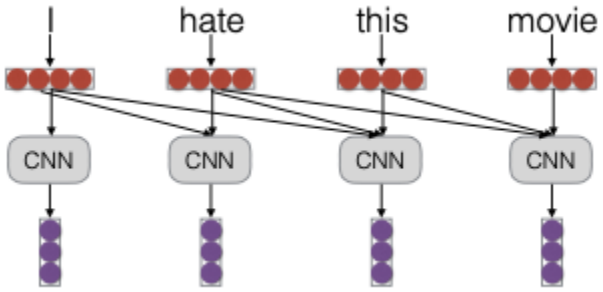


$$h_t = f\left(W[x_{t-1};\ x_t;\ x_{t+1}]\right)$$

Convolution for Auto-regressive Models

Trong NLP, convolution không chỉ được áp dụng trong các mô hình hồi quy tự động mà còn có thể được sử dụng để cải thiện khả năng dự đoán. Thông thường, chúng ta hình dung rằng việc dự đoán dựa trên ba từ: từ trước đó, từ hiện tại và từ tiếp theo. Cách tiếp cận này có thể hiệu quả cho các tác vụ như sequence labeling, nhưng lại không phù hợp cho mô hình ngôn ngữ, vì trong ngữ cảnh này, chúng ta không thể nhìn vào tương lai.

Tuy nhiên, có một giải pháp đơn giản: sử dụng một convolution chỉ nhìn vào quá khứ để dự đoán từ tiếp theo dựa trên từ hiện tại và các từ trước đó. Thực tế, phương pháp này tương đương với mô hình ngôn ngữ feed forward. Bạn cũng có thể coi đây là một mô hình ngôn ngữ convolutional.



Convolution là một trong ba mô hình hóa chuỗi trong NLP, nhưng hiện nay nó ít được sử dụng nhất.

Attention

Basic Idea (Bahdanau et al. 2015)

Về cơ bản, attention cho phép chúng ta mã hóa từng token trong một chuỗi thành một vector. Khi chúng ta có một chuỗi đầu vào mà chúng ta muốn mã hóa, quá trình attention sẽ thực hiện một phép kết hợp tuyến tính của các vector, với các trọng số được xác định bởi attention weight. Điều này giúp mô hình tập trung vào những phần quan trọng của chuỗi đầu vào.

Cross Attention (Bahdanau et al. 2015)

Có hai loại cơ chế chú ý (attention) quan trọng mà chúng ta cần nắm rõ. Loại đầu tiên là cross attention, trong đó mỗi phần tử trong một chuỗi sẽ chú ý đến các phần tử của một chuỗi khác. Cơ chế này thường được sử dụng trong các mô hình encoder-decoder. VD: T5 và BERT.

Giả sử chúng ta có một câu tiếng Nhật và muốn dịch nó sang tiếng Anh. Khi chúng ta tạo ra từ đầu tiên trong tiếng Nhật, từ này có thể là "これ" (Cái này). Để tạo ra từ đầu tiên, chúng ta sẽ thực hiện một phép tính tổng có trọng số của tất cả các embedding của câu tiếng Nhật, và sẽ chú ý nhiều nhất đến từ "これ" vì nó tương ứng với từ "this" trong bước tiếp theo.

Trong quá trình tạo ra các từ tiếp theo, chúng ta sẽ chú ý đến các từ khác nhau vì mỗi từ sẽ tương ứng với các từ khác nhau trong tiếng Anh. Ví dụ, khi chúng ta tạo ra từ "is", chúng ta sẽ chú ý đến từ tương ứng trong câu tiếng Nhật. Tuy nhiên, khi tạo ra từ "and", có thể không có từ nào trong câu tiếng Nhật tương ứng với từ này, dẫn đến việc trọng số chú ý sẽ phân bố đều, không có đỉnh rõ ràng.

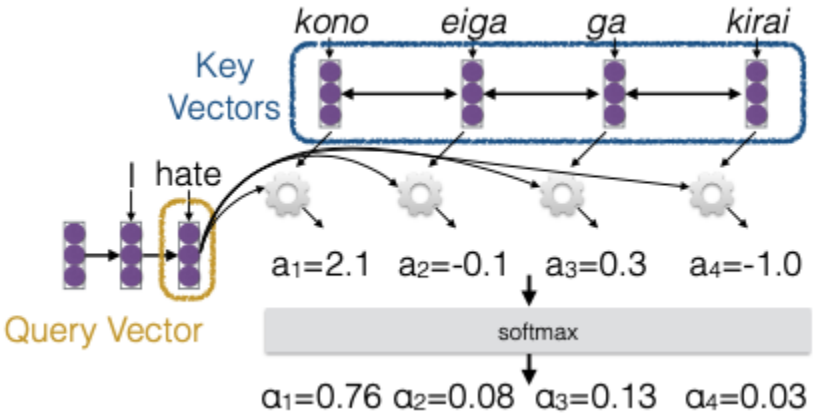
Self Attention (Cheng et al. 2016, Vaswani et al. 2017)

Self-attention cho phép mỗi phần tử trong một chuỗi chú ý đến các phần tử khác trong cùng một chuỗi. Điều này tương tự như cách mà chúng ta đã sử dụng mạng nơ-ron hồi tiếp và mạng nơ-ron tích chập để xử lý dữ liệu chuỗi.

Một ví dụ điển hình để minh họa cho sự cần thiết của self-attention là khi chúng ta muốn mã hóa một câu tiếng Anh trước khi dịch sang tiếng Nhật. Trong trường hợp này, mặc dù ý nghĩa của câu có thể khá rõ ràng, nhưng cách dịch sẽ phụ thuộc rất nhiều vào các từ khác trong câu. Ví dụ, nếu câu có chứa từ "is", chúng ta cần chú ý đến các từ khác để xác định cách dịch chính xác, vì "is" có thể có nhiều cách hiểu khác nhau tùy thuộc vào ngữ cảnh.

Calculating Attention (1)

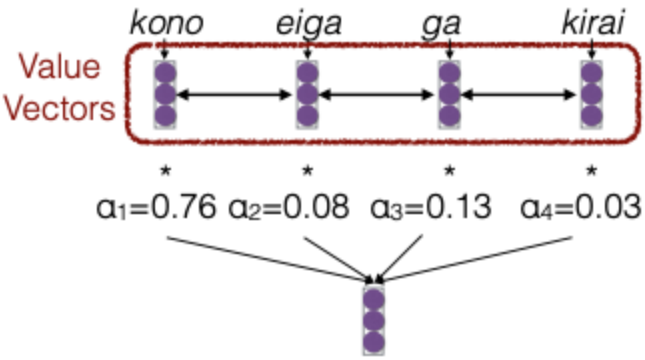
Trong quá trình dịch thuật từ tiếng Nhật sang tiếng Anh, chúng ta thường sử dụng mô hình encoder-decoder, trong đó đầu vào đã được mã hóa và chúng ta sẽ tạo ra đầu ra bằng một mạng nơ-ron hồi tiếp (RNN). Khi dự đoán từ tiếp theo, chúng ta sẽ sử dụng một vector truy vấn (query vector) để xác định những thông tin nào cần chú ý. Bên cạnh đó, chúng ta cũng có các vector khóa (key vectors) để quyết định những thông tin nào sẽ được chú ý.



Đối với mỗi cặp query-key, chúng ta tính toán một trọng số (weight) bằng cách sử dụng một hàm nhất định. Việc sử dụng cùng một hàm cho tất cả các cặp là rất quan trọng, vì điều này cho phép chúng ta mở rộng cho các chuỗi có độ dài không giới hạn. Sau khi tính toán các trọng số, chúng ta sẽ chuẩn hóa chúng để tổng cộng bằng 1 thông qua hàm softmax. Ví dụ, trong một trường hợp cụ thể, trọng số có thể là 0.76.

Calculating Attention (2)

Trong bước thứ hai, sau khi chúng ta có được các giá trị attention, cần lưu ý rằng những giá trị này không thực sự là xác suất, mặc dù chúng nằm trong khoảng từ 0 đến 1 và tổng lại bằng 1. Điều này là do chúng ta chỉ sử dụng chúng để kết hợp nhiều vector lại với nhau. Vì vậy, tôi thường không gọi chúng là "attention probabilities" mà chỉ gọi là "attention values" hoặc "normalized attention values".



Khi đã có các trọng số attention, chúng ta sẽ có các vector giá trị (value vectors) mà chúng ta muốn kết hợp để tạo ra encoding. Chúng ta sẽ thực hiện một phép cộng có trọng số của các vector này để thu được một tổng cuối cùng. Tổng này có thể được sử dụng ở bất kỳ phần nào trong mô hình mà chúng ta muốn. Cách sử dụng phổ biến nhất là có nhiều lớp self-attention, như trong một mô hình Transformer. Tuy nhiên, nó cũng có thể được áp dụng trong các bộ giải mã (decoder) hoặc các cấu trúc khác.

A Graphical Example

Trong phần này, tôi muốn chia sẻ một ví dụ đồ họa từ bài báo gốc về cơ chế attention. Nhìn chung, bạn có thể thấy rằng các trọng số attention trong nhiệm vụ dịch tiếng Anh sang tiếng Pháp thực sự chồng chéo với những gì mà chúng ta mong đợi.

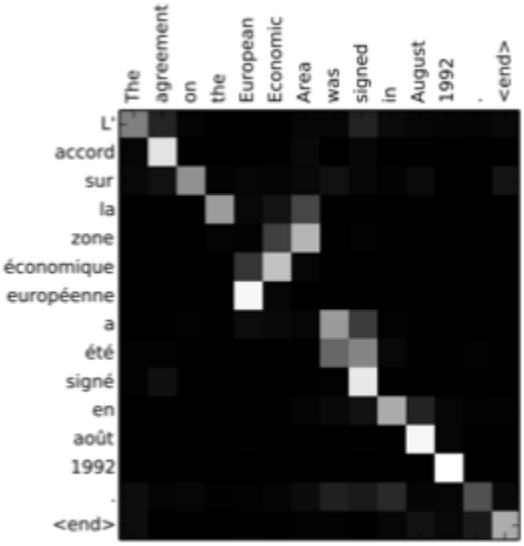


Image from Bahdanau et al. (2015)

Nếu bạn có thể đọc cả tiếng Anh và tiếng Pháp, bạn sẽ nhận ra rằng đó là những từ có nghĩa tương đồng với nhau. Mô hình thậm chí còn học cách sắp xếp lại các từ một cách hợp lý. Tất cả những điều này đều được thực hiện hoàn toàn không có giám sát, nghĩa là bạn không bao giờ cung cấp cho mô hình thông tin về những gì nó nên chú ý. Mọi thứ đều được học thông qua quá trình gradient descent, và mô hình học cách làm điều này bằng cách làm cho các embedding của các vector key và query gần nhau hơn.

Attention Score Functions

Trong phần này, chúng ta sẽ khám phá cách tính toán hàm điểm chú ý (attention score function) trong các mô hình học sâu, đặc biệt là trong kiến trúc Transformer. Đầu tiên, chúng ta cần hiểu rằng trong ngữ cảnh này, Q đại diện cho truy vấn (query) và K đại diện cho khóa (key).

Trong bài báo gốc về Attention (Bahdanau et al. 2015), một mạng nơ-ron đa lớp đã được sử dụng để tính toán hàm điểm chú ý. Cụ thể, nó kết hợp vector truy vấn và vector khóa, nhân với một ma trận trọng số, tính toán hàm tanh và sau đó đưa qua một vector trọng số khác.

$$a(q,\ k) = w_2^T tanh\Big(W_1[q;\ k]\Big)$$

Phương pháp này rất linh hoạt và có khả năng biểu diễn cao, nhưng nó cũng yêu cầu nhiều tham số và thời gian tính toán, do đó không còn được sử dụng rộng rãi nữa.

Một phương pháp được đề xuất bởi Luong et al. 2015 là hàm bilinear. Hàm này sử dụng vector khóa, vector truy vấn và một ma trận ở giữa để tính toán.

$$a(q,\ k) = q^T W k$$

Phương pháp này cho phép kết hợp các vector khóa và truy vấn một cách hiệu quả. Ngoài ra, một số nghiên cứu cũng đã thử nghiệm với dot product, trong đó tính toán bằng cách nhân vector truy vấn với vector khóa.

$$a(q,\ k) = q^T k$$

Tuy nhiên, nhược điểm của phương pháp này là yêu cầu vector truy vấn và vector khóa phải nằm trong cùng một không gian, điều này có thể gây khó khăn khi làm việc với dữ liệu lớn.

Để khắc phục vấn đề này, phương pháp scaled dot product đã được phát triển.

$$a(q,\ k) = q^T k/\sqrt{|k|}$$

Một vấn đề với dot product là độ lớn của nó tăng lên khi kích thước vector lớn hơn. Để giải quyết điều này, chúng ta chia cho căn bậc hai của độ dài của một trong các vector. Việc này giúp chuẩn hóa kết quả, ngăn không cho giá trị trở nên quá lớn. Câu hỏi đặt ra là tại sao lại sử dụng căn bậc hai thay vì chỉ đơn giản là chia cho độ dài. Lý do là khi cộng nhiều biến ngẫu nhiên từ cùng một phân phối, phương sai của tổng sẽ tăng lên theo căn bậc hai, do đó việc chia cho căn bậc hai giúp chuẩn hóa tốt hơn.

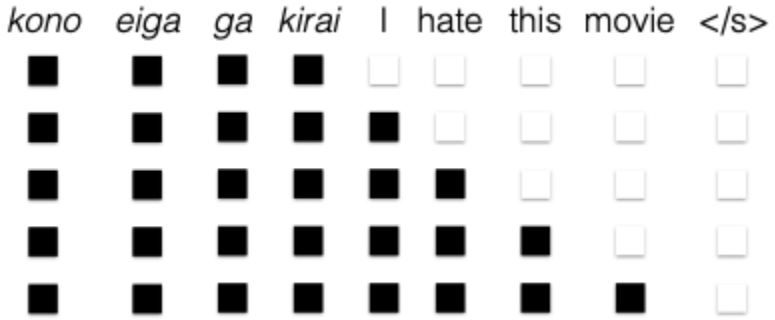
Hiện nay, trong các mô hình Transformer, người ta thường sử dụng trạng thái ẩn từ các khóa và truy vấn, nhân với các ma trận trọng số và sau đó chuẩn hóa bằng căn bậc hai. Điều này thực chất là một mô hình bilinear đã được chuẩn hóa, thường được gọi là "scaled dot product attention". Đây là phương pháp phổ biến nhất hiện nay trong NLP.

Masking for Training

Khi huấn luyện một mô hình tự hồi quy (autoregressive model), điều quan trọng là chúng ta không nên tham chiếu đến thông tin trong tương lai. Nếu làm như vậy, chúng ta sẽ vi phạm nguyên tắc và tạo ra một mô hình không xác suất, điều này sẽ không hiệu quả khi chúng ta cần

sinh ra dữ liệu từ trái sang phải. Do đó, chúng ta cần ngăn chặn việc sử dụng thông tin từ tương lai.

Trong một mô hình không có điều kiện, chúng ta muốn hoàn toàn tránh xa thông tin trong tương lai. Ngược lại, trong một mô hình có điều kiện, chúng ta có thể sử dụng thông tin hai chiều để tính toán các đại diện, nhưng không được phép làm điều đó ở phía mục tiêu.



Để ngăn chặn việc chú ý đến thông tin tương lai, chúng ta sử dụng một mặt nạ. Cụ thể, chúng ta sẽ thêm một số âm vô cùng vào các giá trị chú ý (attention values) mà không muốn sử dụng. Khi áp dụng hàm softmax, những giá trị này sẽ trở thành 0, giúp chúng ta không chú ý đến chúng. Quá trình này được gọi là attention mask và rất quan trọng trong các mô hình học sâu.

Applications of Sequence Models

Encoding Sequences

Trong phần này, chúng ta sẽ khám phá các ứng dụng của mô hình chuỗi. Một trong những ứng dụng chính là mã hóa chuỗi. Khi sử dụng RNN, bạn có thể mã hóa một chuỗi bằng cách lấy giá trị cuối cùng và sử dụng nó để tạo ra đầu ra. Phương pháp này rất hữu ích cho các bài toán phân loại nhị phân hoặc đa lớp. Hiện nay, nó cũng được áp dụng rộng rãi trong việc biểu diễn câu cho tìm kiếm. Ví dụ, bạn có thể xây dựng một chỉ mục tìm kiếm lớn với các vector và thực hiện tìm kiếm gần nhất theo vector để tìm câu tương tự nhất.

Trên thực tế, thay vì chỉ sử dụng vector cuối cùng, việc lấy trung bình hoặc giá trị lớn nhất của tất cả các vector thường mang lại kết quả tốt hơn. Điều này đặc biệt đúng nếu mô hình của bạn chưa được huấn luyện để tạo ra các vector đầu ra chất lượng từ vector cuối cùng. Do đó, việc sử dụng trung bình của tất cả các vector có thể là một lựa chọn hợp lý hơn trong nhiều trường hợp.

Encoding Tokens

Trong NLP, việc mã hóa các token cho nhiệm vụ gán nhãn chuỗi và mô hình ngôn ngữ là rất quan trọng. Một trong những điểm nổi bật của mô hình RNN là tính chất tuần tự của nó, nghĩa là để tính toán một RNN, bạn phải chờ cho nó hoàn thành trước khi tiếp tục với bước tiếp theo. Điều này tạo ra một nút thắt cổ chai lớn, đặc biệt khi sử dụng phần cứng hiện đại như GPU hay TPU, vốn rất hiệu quả trong việc xử lý song song.

Ngược lại, các mô hình attention và Transformers, mặc dù có độ phức tạp tính toán cao hơn ($O(n^2)$), lại có thể thực hiện các phép toán song song mà không phải chờ đợi, giúp tăng tốc độ tính toán đáng kể. Đây là lý do chính khiến các mô hình attention trở nên phổ biến trong thời gian gần đây.

Một lý do khác khiến các mô hình attention dễ học hơn là do chúng giảm thiểu vấn đề triệt tiêu gradient. Trong một RNN, để dự đoán một token ở vị trí xa, thông tin phải đi qua nhiều lớp phi tuyến, dẫn đến việc mất mát thông tin. Trong khi đó, với attention, thông tin có thể được truyền trực tiếp mà không cần phải đi qua nhiều lớp, miễn là trọng số attention được tối ưu hóa tốt.

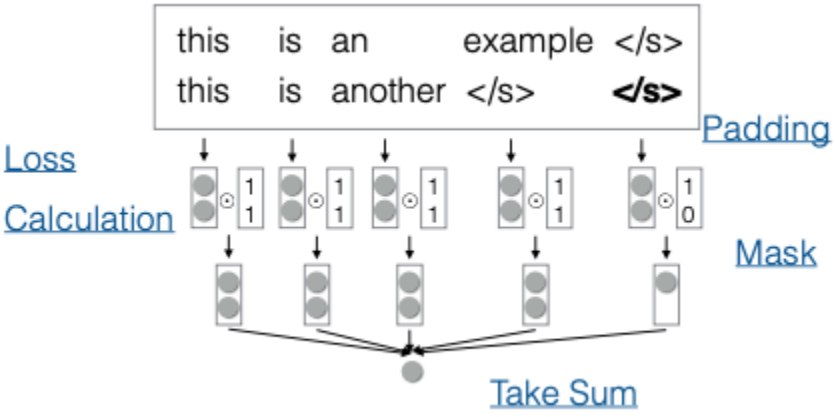
Tuy nhiên, khi nói đến việc sinh ngôn ngữ, các mô hình Transformers lại không hiệu quả bằng RNN. Điều này là do trong quá trình sinh, bạn cần phải chờ đợi để tạo ra token tiếp theo trước khi có thể mã hóa nó, dẫn đến việc không thể thực hiện song song. Do đó, một số hệ thống dịch máy đã kết hợp giữa một encoder mạnh mẽ dựa trên Transformer và một decoder RNN nhỏ gọn để tối ưu hóa hiệu suất.

Efficient Tricks for Sequence Modeling

Handling Mini-batching

Khi xử lý mini batching trong RNN, chúng ta gặp một số thách thức hơn so với các mạng nơ-ron truyền thẳng. Một trong những lý do là trong RNN, mỗi từ phụ thuộc vào từ trước đó, và các chuỗi có độ dài khác nhau.

Để giải quyết vấn đề này, chúng ta sử dụng kỹ thuật padding và masking. Cụ thể, chúng ta có thể thêm một số token bổ sung vào cuối các chuỗi để đảm bảo tất cả chúng có cùng độ dài. Trong các mô hình kiểu encoder-decoder, chúng ta cũng có thể thêm padding vào đầu chuỗi để tất cả bắt đầu từ cùng một vị trí, điều này đặc biệt hữu ích khi sử dụng các mô hình RNN.



Khi tính toán tổn thất (loss), chúng ta nhân tổn thất với một mask để loại bỏ các token mà chúng ta không quan tâm, sau đó tính tổng các giá trị này. May mắn thay, hầu hết các thao tác này đã được triển khai sẵn trong các thư viện như PyTorch hay Hugging Face Transformers, vì vậy bạn không cần phải lo lắng quá nhiều về chi tiết.

Bucketing/ Sorting

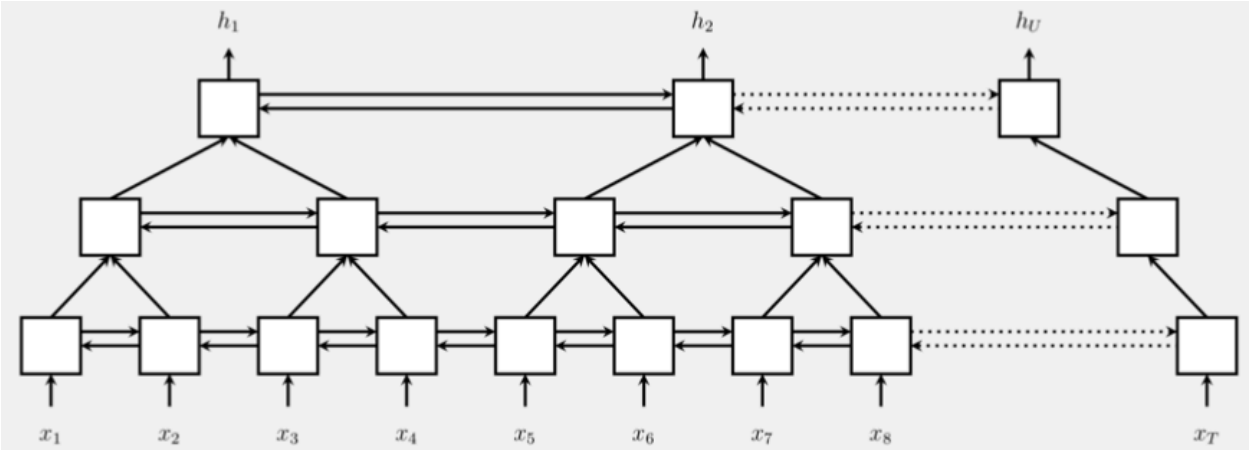
Khi ta sử dụng các câu có độ dài rất khác nhau trong cùng một mini batch, điều này có thể dẫn đến lãng phí tài nguyên tính toán. Ví dụ, nếu hầu hết các movie reviews chỉ dài khoảng 10 từ, nhưng trong mini batch lại có một review dài đến một nghìn từ, thì chúng ta sẽ phải padding hầu hết các câu ngắn đến 990 từ, gây lãng phí lớn khi xử lý.

Một giải pháp cho vấn đề này là sắp xếp các câu theo độ dài trước khi tạo mini batch. Bằng cách này, các câu có độ dài tương tự sẽ được nhóm lại trong cùng một batch, giúp giảm thiểu lãng phí tài nguyên. Tuy nhiên, cần lưu ý rằng nếu bạn xác định số lượng câu trong mini batch, chẳng hạn như 64 sequences, mà một trong số đó lại có độ dài lên đến một nghìn tokens, bạn có thể gặp phải vấn đề thiếu bộ nhớ GPU, dẫn đến việc quá trình huấn luyện bị dừng lại.

Để giải quyết vấn đề này, thư viện Hugging Face Transformers đã cung cấp các tùy chọn phù hợp. Tuy nhiên, việc sắp xếp lại dữ liệu cũng có thể làm giảm tính ngẫu nhiên trong phân phối dữ liệu, điều này có thể ảnh hưởng đến hiệu quả của phương pháp tối ưu hóa như stochastic gradient descent, vốn rất phụ thuộc vào việc dữ liệu được sắp xếp ngẫu nhiên.

Strided Architectures (e.g. Chan et al. 2015)

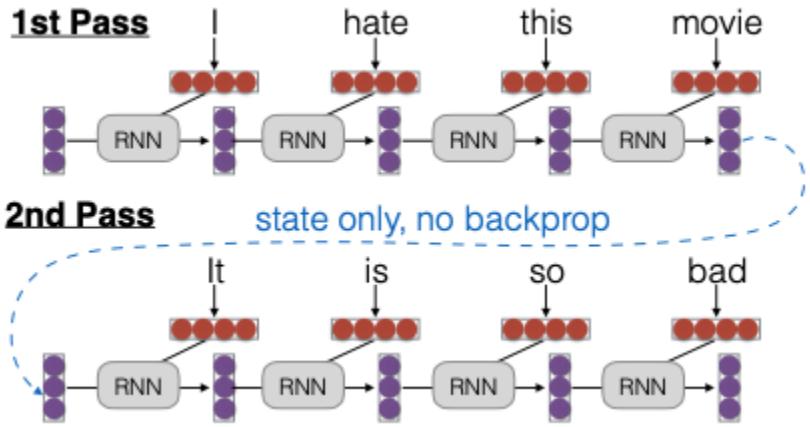
Một kiến trúc mà chúng ta cần lưu ý là kiến trúc có bước nhảy (strided architectures). Kiến trúc này xuất hiện và có tên gọi khác nhau trong nhiều mô hình. Cụ thể, trong RNN, chúng được gọi là "pyramidal RNNs", trong mạng tích chập là "strided architectures", và trong các mô hình attention là "sparse attention". Ý tưởng chính của kiến trúc này là trong một mô hình đa lớp, không phải tất cả các đầu vào từ lớp trước đều được xử lý.



Ví dụ, trong một RNN, bạn có thể xử lý hai đầu vào cho lớp thứ hai, nhưng lại bỏ qua một số đầu vào. Điều này có nghĩa là bạn sẽ có một trạng thái tương ứng với cặp đầu vào đầu tiên, một trạng thái khác cho cặp thứ hai, và tiếp tục như vậy. Nhờ đó, bạn có thể giảm dần độ dài của chuỗi đầu vào mỗi khi xử lý, điều này rất hữu ích khi làm việc với các chuỗi dài.

Truncated BPTT

Trong phần cuối, chúng ta sẽ thảo luận về khái niệm "truncated backpropagation through time" (BPTT) trong mạng nơ-ron hồi tiếp (RNN). Phương pháp này cho phép chúng ta thực hiện quá trình lan truyền ngược qua các đoạn ngắn hơn, nhưng vẫn khởi tạo với trạng thái từ đoạn trước đó. Cụ thể, khi chạy một RNN, chúng ta có thể xử lý một đoạn dữ liệu có độ dài nhất định, chẳng hạn như 4 hoặc 400 bước thời gian. Khi chuyển sang đoạn tiếp theo, chúng ta chỉ cần truyền trạng thái ẩn mà không cần giữ lại toàn bộ đồ thị tính toán trước đó. Điều này giúp chúng ta duy trì thông tin từ trạng thái trước mà không cần cập nhật các tham số dựa trên kết quả của đoạn trước.



Phương pháp này không chỉ được áp dụng rộng rãi trong RNN mà còn trong nhiều kiến trúc Transformer hiện đại, như Transformer-XL, được phát triển tại CMU. Các mô hình mới như Mistral cũng sử dụng các kỹ thuật tương tự, cho thấy rằng phương pháp này vẫn rất phổ biến và hiệu quả trong nghiên cứu hiện nay.

Resources:

https://phontron.com/class/anlp2024/lectures/#sequence-modeling-jan-25