ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

TRẦN TRUNG KIÊN

HỌC ĐẶC TRƯNG KHÔNG GIÁM SÁT BẰNG AUTO-ENCODERS

Chuyên ngành: Khoa Học Máy Tính

Mã số chuyên ngành: 60 48 01

LUẬN VĂN THẠC SỸ: KHOA HỌC MÁY TÍNH

NGƯỜI HƯỚNG DẪN KHOA HỌC:

PGS.TS LÊ HOÀI BẮC

Tp. Hồ Chí Minh, Năm 2014

LỜI CẢM ƠN

Trước tiên, em xin gửi lời tri ân sâu sắc đến Thầy Lê Hoài Bắc. Thầy đã rất tận tâm, nhiệt tình hướng dẫn và chỉ bảo em trong suốt quá trình thực hiện luận văn. Không có sự quan tâm, theo dõi chặt chẽ của Thầy chắc chắn em không thể hoàn thành luận văn này.

Em xin chân thành cảm ơn quý Thầy Cô khoa Công Nghệ Thông Tin - trường đại học Khoa Học Tự Nhiên, những người đã ân cần giảng dạy, xây dựng cho em một nền tảng kiến thức vững chắc.

Con xin cảm ơn ba mẹ đã sinh thành, nuôi dưỡng, và dạy dỗ để con có được thành quả như ngày hôm nay. Ba mẹ luôn là nguồn động viên, nguồn sức mạnh hết sức lớn lao mỗi khi con gặp khó khăn trong cuộc sống.

TP. Hồ Chí Minh, 3/2014

Trần Trung Kiên

MỤC LỤC

LỜI CẢM ƠN i

MỤC LỤC ii

DANH MỤC HÌNH ẢNH iii

DANH MỤC BẢNG iv

Chương 1 Giới thiệu 1

Chương 2 Kiến Thức Nền Tảng 11

2.1 Mạng nơ-ron hồi quy (Recurrent neural network) 11

2.1.1 Huấn luyện mạng nơ-ron hồi quy 15

2.1.2 Thách thức trong việc học các phụ thuộc dài hạn 19

2.2 Long short-term memory (LSTM) 22

2.2.1 LSTM xếp chồng (Stacked LSTM) 25

2.2.2 LSTM hai chiều (Bidirectional LSTM) 25

2.3 Mô hình ngôn ngữ 25

2.3.1 Mô hình ngôn ngữ n-gram 25

DANH MỤC HÌNH ẢNH

1.1 Lịch sử tóm tắt của dịch máy 3
1.2 Ba phương pháp dịch máy dựa trên luật 4
1.3 Ví dụ về tập các câu song song trong hai ngôn ngữ 7
1.4 Ví dụ về Kiến trúc *bộ mã hóa - bộ giải mã* trong dịch máy nơ-ron . . . 8
1.5 Kiến trúc bộ mã hóa - bộ giải mã được xây dựng trên mạng nơ-ron hồi quy 8
1.6 Cơ chế Attention trong dịch máy nơ-ron 9

2.1 Mô hình RNN với kết nối vòng 13
2.2 Mô hình RNN dạng dàn trải 14
2.3 Minh họa thuật toán "Back propagation through time" 20
2.4 Một "LSTM cell" 23

DANH MỤC BẢNG

Chương 1

Giới thiệu

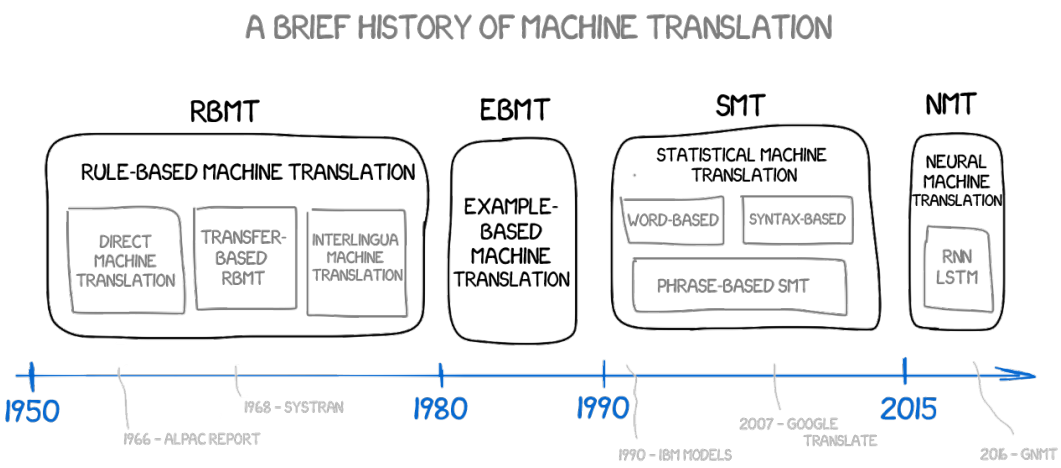
Nhờ vào những cải cách trong giao thông và cơ sở hạ tầng viễn thông mà giờ đây toàn cầu hóa đang trở nên gần với chúng ta hơn bao giờ hết. Trong xu hướng đó nhu cầu giao tiếp và thông hiểu giữa những nền văn hóa là không thể thiếu. Tuy nhiên, những nền văn hóa khác nhau thường kèm theo đó là sự khác biệt về ngôn ngữ, là một trong những trở ngại lớn nhất của sự giao tiếp. Một người phải mất rất nhiều thời gian để thành thạo một ngôn ngữ không phải là tiếng mẹ đẻ, và không thể nào học được nhiều ngôn ngữ cùng lúc. Cho nên, việc phát triển một công cụ để giải quyết vấn đề này là tất yếu. Một trong những công cụ như vậy là *Dịch máy*.

Dịch máy là quá trình chuyển đổi văn bản/tiếng nói từ ngôn ngữ này sang dạng tương ứng của nó trong một ngôn ngữ khác, được thực hiện bởi một chương trình máy tính nhằm mục đích cung cấp bản dịch tốt nhất mà không cần sự trợ giúp của con người. Trong khóa luận này, chúng tôi tập trung nghiên cứu dịch máy trên dữ liệu văn bản.

Dịch máy có một quá trình lịch sử lâu dài. Từ thế kỷ XVII, đã có những ý tưởng về việc cơ giới hóa quá trình dịch thuật. Tuy nhiên, đến thế kỷ XX, những nghiên cứu về dịch máy mới thật sự bắt đầu. Vào những năm 1930, Georges Artsrouni người Pháp và Petr Troyanskii người Nga đã nộp bằng sáng chế cho công trình có tên "máy dịch" của riêng họ. Trong số hai người, công trình của Troyanskii có ý nghĩa hơn. Nó đề xuất không chỉ một phương pháp cho bộ từ điển tự động, mà còn là lược đồ cho việc mã hóa các vai trò ngữ pháp song ngữ và một phác thảo về cách phân tích và tổng hợp có thể hoạt động. Tuy nhiên, những ý tưởng của Troyanskii đã không được biết đến cho đến cuối những năm 1950. Trước đó, máy tính đã được phát minh.

Những nỗ lực xây dựng hệ thống dịch máy bắt đầu ngay sau khi máy tính ra đời. Có thể nói, chiến tranh và sự thù địch giữa các quốc gia là động lực lớn nhất cho dịch máy thời bấy giờ. Trong Thế chiến thứ II, máy tính đã được quân đội Anh sử dụng trong việc giải mã các thông điệp được mã hóa của quân Đức. Việc làm này có thể coi là một dạng ẩn dụ của dịch máy khi người ta cố gắng dịch từ tiếng Đức được mã hóa sang tiếng Anh. Trong thời kỳ chiến tranh lạnh, vào tháng 7/1949, Warren Weaver, người được xem là nhà tiên phong trong lĩnh vực dịch máy, đã viết một bản ghi nhớ đưa ra các đề xuất khác nhau của ông trong lĩnh vực này [**?**]. Những đề xuất đó dựa trên thành công của máy phá mã, sự phát triển của lý thuyết thông tin bởi Claude Shannon và suy đoán về các nguyên tắc phổ quát cơ bản của ngôn ngữ. Trong vòng một năm, một vài nghiên cứu về dịch máy đã bắt đầu tại nhiều trường đại học của Mỹ. Vào ngày 7/1/1954, tại trụ sở chính của IBM ở New York, thử nghiệm Georgetown-IBM được tiến hành. Máy tính IBM 701 đã tự động dịch 49 câu tiếng Nga sang tiếng Anh lần đầu tiên trong lịch sử chỉ sử dụng 250 từ vựng và sáu luật ngữ pháp [**?**]. Thí nghiệm này được xem như là một thành công và mở ra kỉ nguyên cho những nghiên cứu với kinh phí lớn về dịch máy ở Hoa Kỳ. Ở Liên Xô những thí nghiệm tương tự cũng được thực hiện không lâu sau đó.

Trong một thập kỷ tiếp theo, nhiều nhóm nghiên cứu về dịch máy được thành lập. Một số nhóm chấp nhận phương pháp thử và sai, thường dựa trên thống kê với mục tiêu là một hệ thống dịch máy có thể hoạt động ngay lập tức, tiêu biểu như: nhóm nghiên cứu tại đại học Washington (và sau này là IBM) với hệ thống dịch Nga-Anh cho Không quân Hoa Kỳ, những nghiên cứu tại viện Cơ học Chính xác ở Liên Xô và Phòng thí nghiệm Vật lý Quốc gia ở Anh. Trong khi một số khác hướng đến giải pháp lâu dài với hướng tiếp cận lý thuyết bao gồm cả những vấn đề liên quan đến ngôn ngữ cơ bản như nhóm nghiên cứu tại Trung tâm nghiên cứu lý thuyết tại MIT, Đại học Havard và Đơn vị nghiên cứu ngôn ngữ Đại học Cambridge. Những nghiên cứu trong giai đoạn này có tầm quan trọng và ảnh hưởng lâu dài không chỉ cho Dịch máy mà còn cho nhiều ngành khác như Ngôn ngữ học tính toán, Trí tuệ nhân tạo - cụ thể là việc phát triển các từ điển tự động và kỹ thuật phân tích cú pháp. Nhiều nhóm nghiên cứu đã đóng góp đáng kể cho việc phát triển lý thuyết ngôn ngữ. Tuy nhiên, mục tiêu cơ bản của dịch máy là xây dựng hệ thống có khả năng tạo ra bản dịch tốt lại không đạt được dẫn đến một kết quả là vào năm 1966 bản báo cáo từ Ủy ban tư vấn xử lý



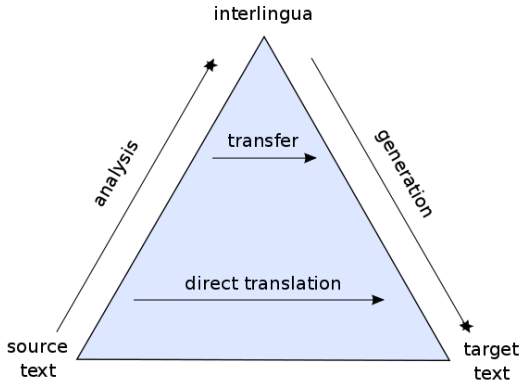
Hình 1.1: Lịch sử tóm tắt của dịch máy, nguồn ảnh: Ilya Pestov trong blog A history of machine translation from the Cold War to deep learning

ngôn ngữ tự động (Automatic Language Processing Advisory) của Hoa Kỳ, tuyên bố rằng dịch máy là đắt tiền, không chính xác và không mang lại kết quả hứa hẹn [**?**]. Thay vào đó, họ đề nghị tập trung vào phát triển các từ điển, điều này đã loại bỏ các nhà nghiên cứu Mỹ ra khỏi cuộc đua trong gần một thập kỷ.

Từ đó đến nay, đã có nhiều hướng tiếp cập đã được sử dụng trong dịch máy với mục tiêu tạo ra bản dịch có độ chính xác cao và giảm thiểu công sức của con người. Trong những năm đầu tiên, để tạo ra bản dịch tốt, các phương pháp thời bấy giờ đều hỏi hỏi những lý thuyết tinh vi về ngôn ngữ học. Hầu hết những hệ thống dịch máy trước những năm 1980 đều là *dịch máy dựa trên luật (Rule-based machine translation - RBMT)*. Những hệ thống này thường bao gồm:

- Một từ điển song ngữ (ví dụ từ điển Anh - Đức)
- Một tập các luật ngữ pháp (ví dụ trong tiếng Đức, từ kết thúc bằng -heit, -keit, -ung là những từ mang giống cái)

Có ba cách tiếp cận khác nhau theo phương pháp dịch máy dựa trên luật. Bao gồm phương pháp dịch máy trực tiếp, dịch máy chuyển giao và dịch máy ngôn ngữ phổ quát. Mặc dù cả ba đều thuộc về RBMT, tuy nhiên chúng khác nhau về độ sâu của đại



Hình 1.2: Kim tự tháp của Bernard Vauquois thể hiện ba phương pháp dịch máy dựa luật theo độ sâu của đại diện trung gian. Bắt đầu từ dịch máy trực tiếp đến dịch máy chuyển dịch và trên cùng là dịch máy ngôn ngữ phổ quát (Nguồn: http://en.wikipedia.org/wiki/Machine_translation)

diện trung gian. Sự khác biệt này được thể hiện qua kim tự tháp Vauquois, minh họa trên hình 1.2

Dịch máy trực tiếp (Direct machine translation - DMT): Đây là phương pháp đơn giản nhất của dịch máy. DMT không dùng bất cứ dạng đại diện nào của ngôn ngữ nguồn, nó chia câu thành các từ, dịch chúng bằng một từ điển song ngữ. Sau đó, dựa trên các luật mà những nhà ngôn ngữ học đã xây dựng, nó chỉnh sửa để bản dịch trở nên đúng cú pháp và ít nhiều đúng về mặt phát âm.

Dịch máy ngôn ngữ phổ quát (Interlingual machine translation - IMT): Trong phương pháp này, câu nguồn được chuyển thành biểu diễn trung gian và biểu diễn này được thống nhất cho tất cả ngôn ngữ trên thế giới (interlingua). Tiếp theo, dạng đại diện này sẽ được chuyển đổi sang bất kỳ ngôn ngữ đích nào. Một trong những ưu điểm chính của hệ thống này là tính mở rộng của nó khi số lượng ngôn ngữ cần dịch tăng lên. Mặc dù trên lý thuyết, phương pháp này trông rất hoàn hảo. Nhưng trong thực tế, thật khó để tạo được một ngôn ngữ phổ quát như vậy.

Dịch máy chuyển giao (Transfer-based machine translation - TMT): dịch máy chuyển giao tương tự như dịch máy ngôn ngữ đại diện ở chỗ, nó cũng tạo ra bản dịch từ biểu diễn trung gian mô phỏng ý nghĩa của câu gốc. Tuy nhiên, không giống như IMT, TMT phụ thuộc một phần vào cặp ngôn ngữ mà nó tham gia vào quá trình dịch. Trên cơ sở sự khác biệt về cấu trúc của ngôn ngữ nguồn và ngôn ngữ đích, một hệ thống TMT có thể được chia thành ba giai đoạn: i) Phân tích, ii) Chuyển giao, iii)

Tạo ra bản dịch. Trong giai đoạn đầu tiên, trình phân tích cú pháp ở ngôn ngữ nguồn được sử dụng để tạo ra biểu diễn cú pháp của câu nguồn. Trong giai đoạn tiếp theo, kết quả của phân tích cú pháp được chuyển đổi thành biểu diễn tương đương trong ngôn ngữ đích. Trong giai đoạn cuối cùng, một bộ phân tích hình thái của ngôn ngữ đích được sử dụng để tạo ra các bản dịch cuối cùng.

Mặc dù đã có một số hệ thống RBMT được đưa vào sử dụng như PROMPT (http://www.promt.com/) và Systrans (http://www.systransoft.com/). Tuy nhiên, bản dịch của hướng tiếp cận này có chất lượng thấp so với nhu cầu của con người và không sử dụng được trừ một số trường hợp đặc biệt. Ngoài ra chúng còn có một số nhược điểm lớn như:

- Các loại từ điển chất lượng tốt có sẵn là không nhiều và việc xây dựng những bộ từ điển mới là rất tốn kém.
- Hầu hết những luật ngôn ngữ được tạo ra bằng tay bởi các nhà ngôn ngữ học. Việc này gây khó khăn và tốn kém khi hệ thống trở nên lớn hơn.
- Các hệ thống RBMT gặp khó khăn trong việc giải quyết những vấn đề như thành ngữ hay sự nhập nhằng về ngữ nghĩa của các từ.

Từ những năm 1980, dịch máy dựa trên *Ngữ liệu* (Corpus-based machine translation) được đề xuất. Điểm khác biệt lớn nhất và cũng là quan trọng nhất của hướng tiếp cận này so với RBMT là thay vì sử dụng các bộ từ điển song ngữ, nó dùng những tập câu tương đương trong hai ngôn ngữ làm nền tảng cho việc dịch thuật. Tập những câu tương đương này được gọi là ngữ liệu. So với từ điển, việc thu thập ngữ liệu đơn giản hơn rất nhiều. Ví dụ như ta có thể tìm thấy nhiều phiên bản trong các ngôn ngữ khác nhau của những văn bản hành chính hay các trang web đa ngôn ngữ. Trước khi dịch máy nơ-ron ra đời, phương pháp dịch máy dựa trên ngữ liệu hiệu quả nhất chính là dịch máy thống kê.

Dịch máy thống kê (Statistical machine translation - SMT): ý tưởng của phương pháp này là thay vì định nghĩa những từ điển và các luật ngữ pháp một cách thủ công, SMT dùng mô hình thống kê để học các từ điển và các luật ngữ pháp này từ ngữ liệu. Những ý tưởng đầu tiên của SMT được giới thiệu đầu tiên bởi Waren Weaver vào năm 1949 bap gồm việc áp dụng lý thuyết thông tin của Claude Shannon vào dịch máy.

SMT được giới thiệu lại vào cuối những năm 1980 và đầu những năm 1990 tại trung tâm nghiên cứu Thomas J. Watson của IBM. SMT là phương pháp được nghiên cứu rộng rãi nhất thời bấy giờ và thậm chí đến hiện tại, nó vẫn là một trong những phương pháp được nghiên cứu nhiều nhất về dịch máy.

Để hiểu rõ hơn về dịch máy thống kê, xét một ví dụ: ta cần dịch một câu f trong tiếng Pháp sang dạng tiếng Anh e của nó. Có nhiều bản dịch có thể có cuả f trong tiếng Anh, việc cần làm là chọn e sao cho nó là bản dịch "tốt nhất" của f. Chúng ta có thể mô hình hóa quá trình này bằng một xác suất có điều kiện $p(e|f)$ với e là những bản dịch có thể có với câu cho trước f. Một cách hợp lý để chọn bản dịch "tốt nhất" là chọn e sao cho nó tối đa xác suất có điều kiện $p(e|f)$. Cách tiếp cận quen thuộc là sử dụng định lý Bayes để viết lại $p(e|f)$:

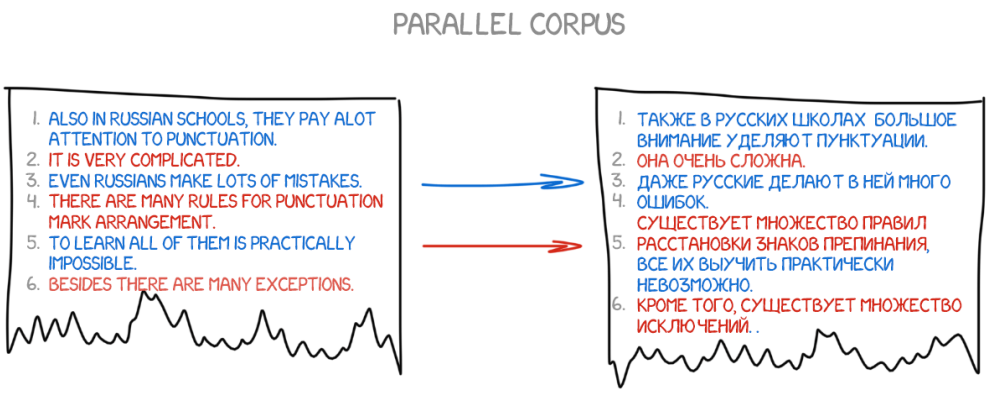
$$p(e|f) = \frac{p(f|e)p(e)}{p(f)} \tag{1.1}$$

Bởi vì f là cố định, tối đa hóa $p(e|f)$ tương đương với tìm e sao cho tối đa hóa $p(f|e)p(e)$. Để làm được điều này, chúng ta dựa vào một tập ngữ liệu là những câu song ngữ Anh - Pháp để suy ra các mô hình $p(f|e)$ và $p(e)$ và sử dụng những mô hình đó để tìm một bản dịch cụ thể $\tilde{e}$ sao cho:

$$\tilde{e} = \arg\max_{e \in e^*} p(e|f) = \arg\max_{e \in e^*} p(f|e)p(e) \tag{1.2}$$

Ở đây, $p(f|e)$ được gọi là *mô hình dịch* (translation model) và $p(e)$ được gọi là *mô hình ngôn ngữ* (language model). Mô hình dịch $p(f|e)$ thể hiện khả năng câu e là một bản dịch của câu f. Những mô hình dịch ban đầu dựa trên từ (word-based) như các mô hình IBM 1-5 (IBM Models 1-5). Những năm 2000, những mô hình dịch dựa trên cụm từ (phrase based) xuất hiện giúp cải thiện khả năng dịch của SMT. Trong khi đó, mô hình ngôn ngữ $p(e)$ thể hiện độ trơn tru của câu e. Ví dụ $p(\text{"tôi đi học"}) > p(\text{"học tôi đi"})$ vì rõ ràng "tôi đi học" là có lý hơn "học tôi đi". Các mô hình ngôn ngữ cho SMT thường được ước lượng bằng các mô hình n-gram được làm mịn, cách làm này cũng là một nhược điểm của SMT. Mô hình ngôn ngữ là một chủ đề quan trọng và sẽ được chúng tôi đề cập cụ thể hơn trong chương 2.

Mặc dù trên thực tế đã có nhiều hệ thống dịch máy được phát triển dựa trên dịch



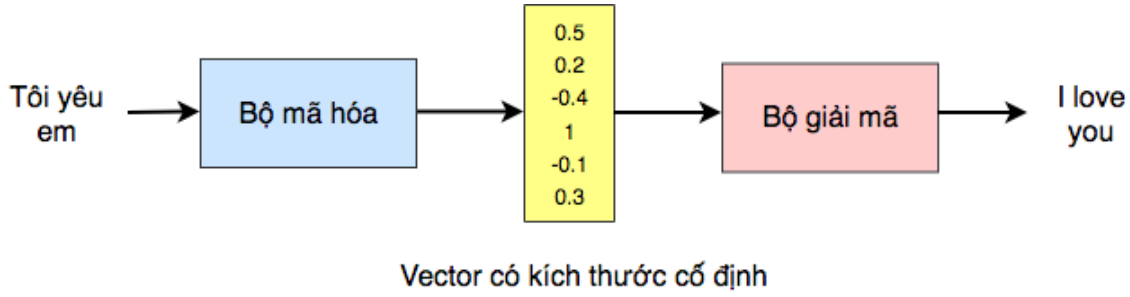
Hình 1.3: Ví dụ về tập các câu song song trong hai ngôn ngữ

máy thống kê thời bấy giờ, tuy nhiên nó không hoạt động thực sự tốt bởi một số nguyên nhân. Một là việc những từ hay đoạn được dịch cục bộ và quan hệ của chúng với những từ cách xa trong câu nguồn thường bị bỏ qua. Hai là mô hình ngôn ngữ n-gram hoạt động không thực sự tốt đối với những bản dịch dài và ta phải tốn nhiều bộ nhớ để lưu trữ chúng. Ngoài ra việc sử dụng nhiều thành phần nhỏ được điều chỉnh riêng biệt như mô hình dịch, mô hình ngôn ngữ,.. cũng gây khó khăn cho việc vận hành và phát triển mô hình này.

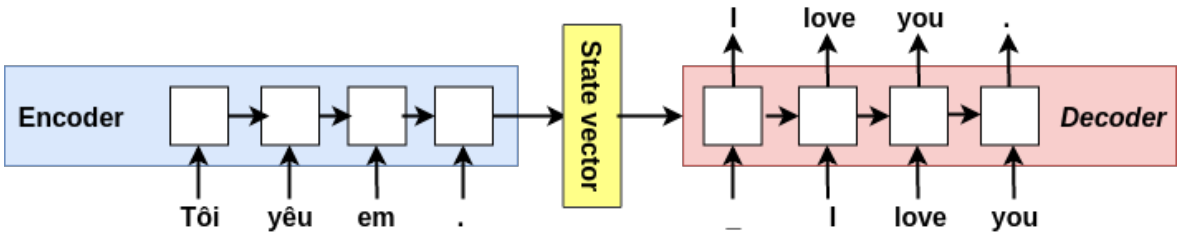
Dịch máy nơ-ron (Neural machine translation) là một hướng tiếp cận mới cho bài toán dịch máy trong những năm gần đây và được đề xuất đầu tiên bởi [**?**], [**?**], [**?**]. Giống như dịch máy thống kê, dịch máy nơ-ron cũng là một phương pháp thuộc hướng tiếp cận dựa trên ngữ liệu, trong khi dịch máy thống kê bao gồm nhiều mô-đun nhỏ được điều chỉnh riêng biệt, dịch máy nơ-ron cố gắng dùng một mạng nơ-ron như là thành phần duy nhất của hệ thống, mọi thiết lập sẽ được thực hiện trên mạng này.

Hầu hết những mô hình dịch máy nơ-ron đều dựa trên kiến trúc *Bộ mã hóa - Bộ giải mã* (Encoder - Decoder) ([**?**], [**?**]). Bộ mã hóa thường là một mạng nơ-ron có tác dụng *"nén"* tất cả thông tin của câu trong ngôn ngữ nguồn vào một vector có kích thước cố định. Bộ giải mã, cũng là một mạng nơ-ron, sẽ tạo bản dịch trong ngôn ngữ đích từ vector có kích thước cố định kia. Toàn bộ hệ thống bao gồm bộ mã hóa và bộ giải mã sẽ được huấn luyện *"end-to-end"* để tạo ra bản dịch, quá trình này được mô tả như hình 1.4.

Trong thực tế cả bộ mã hóa và giải mã thường dựa trên một mô hình mạng nơ-ron tên là *mạng nơ-ron hồi quy* là một thiết kế mạng đặc trưng cho việc xử lý dữ liệu



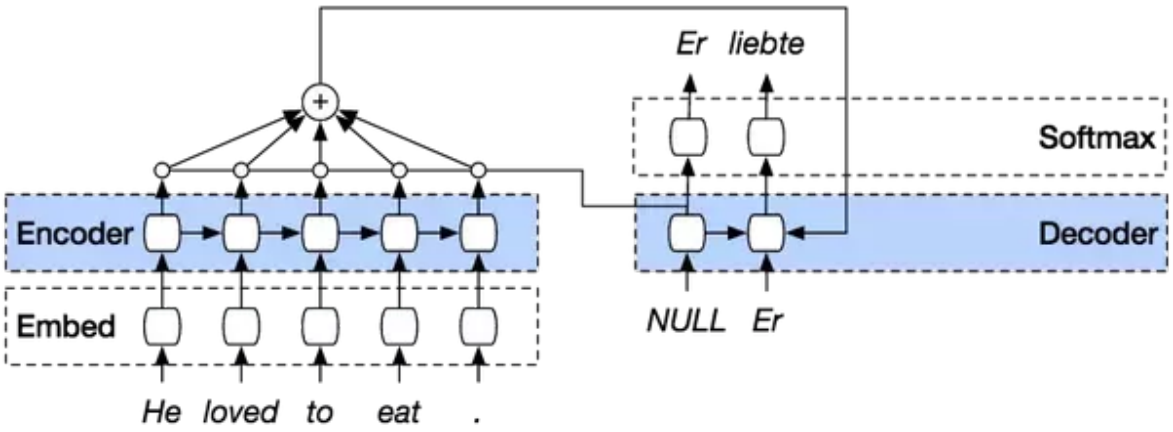
Hình 1.4: Ví dụ về kiến trúc bộ mã hóa - bộ giải mã trong dịch máy nơ-ron



Hình 1.5: Kiến trúc bộ mã hóa - bộ giải mã được xây dựng trên mạng nơ-ron hồi quy

chuỗi. Mạng nơ-ron hồi quy cho phép chúng ta mô hình hóa những dữ liệu có độ dài không xác định, rất thích hợp cho bài toán dịch máy. Hình 1.5 mô tả chi tiết hơn về kiến trúc bộ mã hóa - giải mã sử dụng mạng nơ-ron hồi quy. Đầu tiên bộ mã hóa đọc qua toàn bộ câu nguồn và tạo ra một vector đại diện gọi là *vector trạng thái*. Điều này giúp cho toàn bộ những thông tin cần thiết hay quan hệ giữa các từ đều được tập hợp vào một nơi duy nhất. Bộ giải mã, lúc này đóng vai trò như một mô hình ngôn ngữ để tạo ra từng từ trong ngôn ngữ đích và sẽ dừng lại đến khi một ký tự đặc biệt xuất hiện.

Trong hình 1.5, có thể thấy rằng bộ giải mã tạo ra bản dịch chỉ dựa trên trạng thái ẩn cuối cùng, cũng chính là vector có kích thước cố định được tạo ra ở bộ mã hóa. Vector này phải mã hóa mọi thứ chúng ta cần biết về câu nguồn. Giả sử chúng ta có câu nguồn với độ dài là 50 từ, từ đầu tiên ở câu đích có lẽ sẽ có mối tương quan cao với từ đầu tiên ở câu nguồn. Điều này có nghĩa là bộ giải mã phải xem xét thông tin được mã hóa từ 50 *"time step"* trước đó. Mạng nơ-ron hồi quy được chứng minh là gặp khó khăn trong việc mã hóa những chuỗi dài [**?**]. Để giải quyết vấn đề này, thay vì dùng mạng nơ-ron hồi quy thuần, người ta sử dụng các biến thể của nó như *Long short-term memory (LSTM)*. Trên lý thuyết, LSTM có thể giải quyết vấn đề mất mát thông tin trong chuỗi dài, nhưng trong thực tế vấn đề này vẫn chưa thể được giải quyết hoàn toàn. Một số nhà nghiên cứu đã phát hiện ra rằng đảo ngược chuỗi nguồn trước



Hình 1.6: Cơ chế Attention trong dịch máy nơ-ron

khi đưa vào bộ mã hóa tạo ra kết quả tốt hơn một cách đáng kể [**?**] bởi nó khiến cho những từ đầu tiên được đưa vào bộ mã hóa sau cùng, và được giải mã thành từ tương ứng ngay sau đó. Cách làm này tuy giúp cho bản dịch hoạt động tốt hơn trong thực tế, nhưng nó không phải là một giải pháp về mặt thuật toán. Hầu hết các đánh giá về dịch máy được thực hiện trên các ngôn ngữ như ngôn ngữ có trật tự câu tương đối giống nhau. Ví dụ trật tự dạng "chủ ngữ - động từ - vị ngữ" như tiếng Anh, Đức, Pháp hay Trung Quốc. Đối với dạng ngôn ngữ có một trật tự khác ví dụ "chủ ngữ - vị ngữ - động từ" như tiếng Nhật, đảo ngược câu nguồn sẽ không hiệu quả.

Năm 2015, trong bài báo "Effective Apporoaches to Attention-based Neural Machine Translation" [**?**] được công bố tại hội nghị EMNLP, nhóm tác giả của Đại học Stanford (Minh-Thang Luong, Hieu Pham, Christopher D. Manning) đã đưa ra mô hình dịch máy LSTM-attention, trong đó đưa thêm cơ chế attention vào kiến trúc bộ mã hóa – bộ giải mã (ở đây, bộ mã hóa là một LSTM, bộ giải mã cũng là một LSTM). Attention là cơ chế giải phóng kiến trúc bộ mã hóa - bộ giải mã khỏi nhược điểm chỉ sử dụng một vector có chiều dài cố định làm đại diện cho câu đầu vào. Ý tưởng chính của cơ chế này là ở mỗi thời điểm phát sinh một từ trong bản dịch, bộ giải mã sẽ "chú ý" vào các phần khác nhau của câu nguồn trong quá trình mã hóa. Quan trọng hơn, cơ chế này cho phép mô hình học được cách chọn những phần cần thiết để tập trung vào dựa trên câu nguồn và những gì mà bộ giải mã đã giải mã được.

Cơ chế Attention được miêu tả trong hình 1.6. Ở đây tại mỗi bước tạo ra từ mới, bộ giải mã kết hợp trạng thái ẩn ở bước giải mã trước đó với một vec-tơ gọi là *vec-tơ ngữ cảnh* để sinh ra trạng thái ẩn ở thời điểm hiện tại. Vec-tơ ngữ cảnh này là *tổng có*

trọng số của tất cả các vec-tơ trạng thái ẩn trong lúc mã hóa (đang đề cập đến *global attention*). Với cách tiếp cận này, bộ giải mã không chỉ tạo ra bản dịch dựa trên vec-tơ mã hóa của câu nguồn mà còn tận dụng được cả những trạng thái ẩn ở câu nguồn trong quá trình mã hóa. Điều này giúp cho mô hình dịch máy trở nên hiệu quả hơn. Bên cạnh đó các trọng số tạo nên vec-tơ ngữ cảnh cho biết những từ trong câu đích đã "chú ý" đến những từ nào trong câu nguồn. Việc này giúp chúng ta phần nào giải thích được các kết quả của dịch máy nơ-ron - vốn rất khó giải thích vì tính phức tạp của nó.

Với những ưu điểm đã được trình bày ở trên của phương pháp dịch máy nơ-ron so với các phương pháp trước đó, cũng như là với những lợi ích khi sử dụng cơ chế attention trong dịch máy nơ-ron, trong khóa luận này chúng tôi tìm hiểu về mô hình dịch máy LSTM-attention được đề xuất trong bài báo “Effective Approaches to Attention-based Neural Machine Translation” [**?**] đã nói ở trên. Trong khóa luận, chúng tôi đã thành công xây dựng lại mô hình được đề xuất trong bài báo. Bên cạnh đó chúng tôi cũng thực nhiều những thí nghiệm để kiểm tra độ hiệu quả của mô hình. Những thí nghiệm đó bao gồm: thí nghiệm so sánh giữa mô hình dịch máy nơ-ron sử dụng và không sử dụng attention và các thí nghiệm so sánh giữa các mô hình attention với nhau. Các phần còn lại trong luận văn được trình bày như sau:

- Chương 2 trình bày về những thành nền tảng của kiến trúc bộ mã hóa - giải mã.
- Chương 3 trình bày về cơ chế Attention, đây là phần chính của luận văn. Trong phần này gồm có hai phần nhỏ:
    - *Global attetion*: là cơ chế tập trung vào tất cả các trạng thái ở câu nguồn
    - *Local attetion*: tập trung vào một tập các trạng thái ở câu nguồn tại một thời điểm
- Chương 4 trình bày về các thí nghiệm và các phân tích về kết quả đạt trên hai tập dữ liệu Anh-Đức, Anh-Việt.
- Cuối cùng, kết luận và hướng phát triển được trình bày ở chương 5.

Chương 2

Kiến Thức Nền Tảng

Trong chương này, chúng tôi sẽ trình bày những kiến thức nền tảng trên ba chủ đề bao gồm mạng nơ-ron hồi quy, Long short-term memory và mô hình ngôn ngữ nơ-ron. Mạng nơ-ron hồi quy (RNN) là xương sống của dịch máy nơ-ron. Nó được sử dụng để làm cả bộ mã hóa lẫn bộ giải mã. Ứng với mỗi vai trò, RNN sẽ có một thiết kế riêng. Long short-term memory là phiên bản cải tiến của RNN để giải quyết vấn đề về phụ thuộc dài hạn. Long short-term memory cũng là phiên bản RNN được dùng để xây dựng nên bộ mã hóa - bộ giải trong khóa luận này. Sau đó, dựa trên những kiến thức về mạng nơ-ron hồi quy, chúng tôi nói về khái niệm mô hình ngôn ngữ hồi quy với chức năng tạo ra từ trong bộ giải mã, là bước quan trọng trong dịch máy nơ-ron. Những thành phần này cung cấp kiến thức nền tảng để đi đến mô hình dịch máy nơ-ron theo kiến trúc bộ mã hóa - bộ giải mã mà chúng tôi sẽ trình bày trong chương 3.

Mạng nơ-ron hồi quy (Recurrent neural network)

Trong tự nhiên, dữ liệu không phải lúc nào cũng được sinh ra một cách ngẫu nhiên. Trong một số trường hợp, chúng được sinh ra theo một thứ tự. Xét trong dữ liệu văn bản, ví dụ ta cần điền vào chỗ trống cho câu sau *"Paris là thủ đô của nước __"*. Dễ biết được rằng chỉ có duy nhất một từ phù hợp cho chỗ trống này, đó là *"Pháp"*. Điều này có nghĩa là mỗi từ trong một câu không được tạo ra ngẫu nhiên mà nó được tạo ra dựa trên một liên hệ với những từ đứng trước nó. Các loại dữ liệu khác như những

khung hình trong một bộ phim hoặc các đoạn âm thanh trong một bản nhạc cũng có tính chất tương tự. Những loại dữ liệu mang thứ tự này được gọi chung là dữ liệu chuỗi (sequential data).

Trong quá khứ, một số mô hình xử lý dữ liệu chuỗi bằng cách giả định rằng đầu vào hiện tại có liên hệ với một số lượng xác định đầu vào trước đó, nhiều mô hình tạo ra một cửa sổ trượt để nối mỗi đầu vào hiện tại với một số lượng đầu vào trước đó nhằm tạo ra sự mô phỏng về tính phụ thuộc. Cách tiếp cận này đã được sử dụng cho mô hình *Deep belief network* trong xử lý tiếng nói [**?**]. Nhược điểm của những cách làm này là ta phải xác định trước kích thước của cửa sổ. Một mô hình với kích thước cửa sổ với chiều dài bằng 6 không thể nào quyết định được từ tiếp theo trong câu *"Hổ là loài động vật ăn __"* sẽ là *"thịt"* hay *"cỏ"*. Trong ví dụ này, từ tiếp theo của câu phụ thuộc mật thiết vào từ *"Hổ"* cách nó đúng 6 từ. Trên thực tế, có rất nhiều câu đòi hỏi sự phụ thuộc với nhiều từ xa hơn trước đó. Ta gọi những sự phụ thuộc kiểu như vậy là những *phụ thuộc dài hạn* (long term dependency).

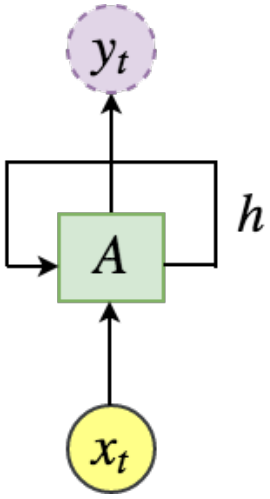
Mạng nơ-ron hồi quy (recurrent neural network) [**?**] gọi tắt là *RNN* là một nhánh của nạng nơ-ron nhân tạo được thiết kế đặc biệt cho việc mô hình hóa dữ liệu chuỗi. Khác với những mô hình đã đề cập giả định sự phụ thuộc chỉ xảy ra trong một vùng có chiều dài cố định. RNN, trên lý thuyết, có khả năng nắm bắt được các phụ thuộc dài hạn với chiều dài bất kỳ. Để làm được điều đó, trong quá trình học, RNN lưu giữ những thông tin cần thiết cho các phụ thuộc dài hạn bằng một vec-tơ được gọi là *trạng thái ẩn*.

Xét một chuỗi đầu vào $x = x_1, x_2, ..., x_n$. Ta gọi h_t là trạng thái ẩn tại *bước thời gian* (timestep) t, là lúc một mẫu dữ liệu x_t được đưa vào RNN để xử lý. Trạng thái ẩn h_t sẽ được tính toán dựa trên mẫu dữ liệu hiện tại x_t và trạng thái ẩn trước đó h_{t-1}. Có thể thể hiện h_t như một hàm hồi quy với tham số là đầu vào hiện tại và chính nó ở thời điểm trước đó:

$$h_t = f(h_{t-1}, x_t) \tag{2.1}$$

trong đó hàm f là một ánh xạ phi tuyến. Có thể hình dung h_t như một đại diện cho những đầu vào mà nó đã xử lý từ thời điểm ban đầu cho đến thời điểm t. Nói một cách khác, RNN sử dụng trạng thái ẩn như một dạng bộ nhớ để lưu giữ thông tin từ một chuỗi. Hình 2.1 thể hiện định nghĩa hồi quy của RNN.

Thông thường, hàm f là một hàm phi tuyến như hàm σ hay hàm tanh. Xét một



Hình 2.1: Mô hình RNN đơn giản với kết nối vòng, h được xem như bộ nhớ được luân chuyển trong RNN. Chú ý rằng đường nét đứt ở đầu ra thể hiện rằng tại một thời điểm t, RNN có thể có hoặc không có một đầu ra.

RNN với công thức cụ thể như sau:

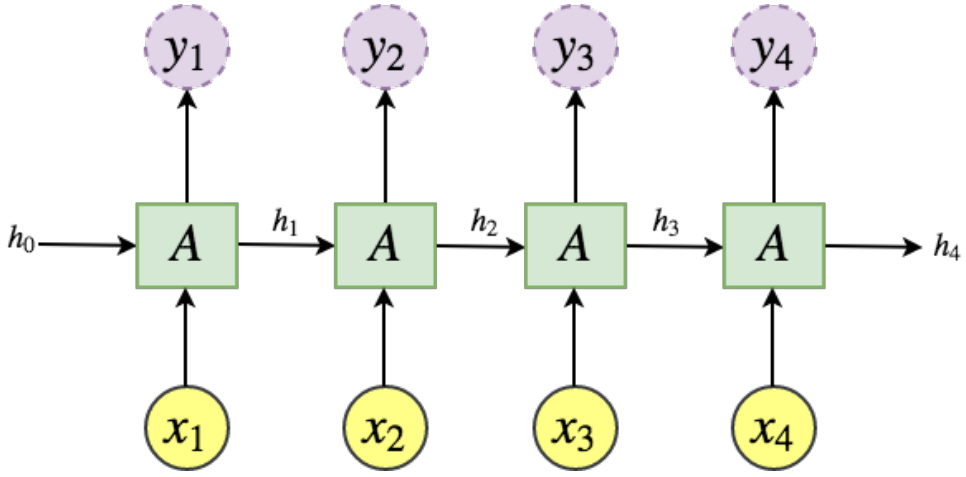
$$h_t = \phi\left(W_{xh}x_t + W_{hh}h_{t-1} + b_h\right) \tag{2.2}$$

Trong đó:

- ϕ là một hàm kích hoạt (ví dụ: sigmoid, tanh hay ReLU).
- $h_t \in \mathbb{R}^n$ là trạng thái ẩn tại bước thời gian hiện tại.
- $x_t \in \mathbb{R}^m$ là đầu vào hiện tại.
- $h_{t-1} \in \mathbb{R}^n$ là trạng thái ẩn tại bước thời gian trước đó.
- $W_{xh} \in \mathbb{R}^{m\times n}, W_{hh} \in \mathbb{R}^{n\times n}$ và $b_h \in \mathbb{R}^n$ lần lượt là hai ma trận trọng số và vec-tơ "bias".

Ma trận W_{xh} là làm nhiệm vụ kết nối giữa đầu vào và trạng thái ẩn, W_{hh} kết nối trạng thái ẩn với chính nó trong các bước thời gian liền kề. Vec-tơ b_h dùng để điều chỉnh giá trị của h_t. Tại thời điểm bắt đầu, trạng thái ẩn h_0 có thể được khởi tạo bằng 0 hoặc là một vector chứa tri thức có sẵn như trường hợp của bộ giải mã như chúng tôi đã đề cập trong chương 1.

Tại mỗi bước thời gian t, tùy vào mục tiêu cụ thể của quá trình học mà RNN có thể có thêm một đầu ra y_t. Trong ngữ cảnh bài toán dịch máy nơ-ron, đầu ra của RNN



Hình 2.2: Mô hình RNN được dàn trải (unrolled), ví dụ trong 4 bước thời gian.

trong quá trình giải mã chính là một từ trong ngôn ngữ đích hay nói chung là một đầu ra dạng rời rạc. Với mục tiêu đó, đầu ra dự đoán của RNN $\hat{y}_t$ sẽ có dạng là một phần phối xác suất trên tập các các lớp ở đầu ra. Phân phối này nhằm dự đoán vị trí xuất hiện của $\hat{y}_t$.

$$\hat{y}_t = \text{softmax}(W_{hy} h_t + b_y) \tag{2.3}$$

Trong đó:

- softmax là một hàm kích hoạt với $\text{softmax}(v_j) = \frac{e^{v_j}}{\sum_{k=1}^{K} e^{v_k}}$, $j = 1, ..., K$, K là độ dài của vec-tơ v.
- $h_t \in \mathbb{R}^n$ là trạng thái ẩn tại bước thời gian hiện tại.
- $W_{hy} \in \mathbb{R}^{L \times n}$ và $b_y \in \mathbb{R}^L$ lần lượt là hai ma trận trọng số và vec-tơ "bias". L là số lượng lớp cần phân biệt ở đầu ra.

Trong công thức trên, hàm softmax đóng vai trò là một hàm chuẩn hóa để $\hat{y}_t$ thể hiện một phân phối xác suất trên các lớp ở đầu ra. Ma trận W_{hy} kết nối đầu ra với trạng thái ẩn, b_y dùng để điều chỉnh giá trị của kết quả tính toán trước khi đưa qua hàm softmax.

Để ý rằng các ma trận trọng số W_{xh}, W_{hh}, W_{hy} và các vector bias b_h, b_y là các tham số học của mô hình và chúng là duy nhất. Có nghĩa là khi những tham số này được học, bất kỳ một đầu vào nào cũng đều sử dụng chung một bộ tham số. Điều này chính là sự chia sẻ tham số (parameters sharing) trong mạng nơ-ron hồi quy. Chia sẻ tham

số khiến cho mô hình học dễ dàng hơn, nó giúp cho RNN có thể xử lý chuỗi đầu vào với độ dài bất kỳ mà không làm tăng độ phức tạp của mô hình. Quan trọng hơn, nó giúp ích cho việc tổng quát hóa. Đây chính là điểm đặc biệt của RNN so với mạng nơ-ron truyền thẳng.

Với một số lượng hữu hạn các bước thời gian, mô hình RNN trên hình 2.1 có thể được dàn trải ra (unrolled). Dạng dàn trải này được miêu tả trực quan như trên hình 2.2. Với cách thể hiện này, RNN có thể được hiểu như là một mạng nơ-ron sâu với mỗi bước thời gian là một mạng nơ-ron một tầng ẩn và các tham số học được chia sẻ giữa các mạng nơ-ron đó. Dạng dàn trải cũng thể hiện rằng RNN có thể được huấn luyện qua nhiều bước thời gian bằng thuật toán lan truyền ngược (backpropagation). Thuật toán này được gọi là "Backpropagation through time" (BPTT) [**?**]. Thực chất đây là chỉ thuật toán "Backpropagation" khi áp dụng cho RNN dưới dạng dàn trải để tính "gradient" cho các tham số ở từng bước thời gian. Hầu hết cả các mạng nơ-ron hồi quy phổ biến ngày nay đều áp dụng thuật toán này vì tính đơn giản và hiệu quả của nó.

Huấn luyện mạng nơ-ron hồi quy

Xét một chuỗi đầu vào $x = x_1, x_2, ..., x_n$ với đầu ra tương ứng $y = y_1, y_2, ..., y_n$. Trong quá trình lan truyền tiến, tại mỗi bước thời gian t ứng mẫu dữ liệu (x_t, y_t), công thức tính toán đầu ra dự đoán có dạng:

$$h_t = \phi\left(W_{xh}x_t + W_{hh}h_{t-1} + b_h\right) \tag{2.4}$$

$$s_t = W_{hy}h_t + b_y \tag{2.5}$$

$$\hat{y}_t = softmax(s_t) \tag{2.6}$$

Ta cần định nghĩa hàm độ lỗi giữa đầu ra dự đoán $\hat{y}_t$ và đầu ra thật sự y_t. Với L là số lượng lớp của y, lúc này có thể thấy $\hat{y}_t$ là một vec-tơ phân phối xác suất có độ dài L. Để so sánh với $\hat{y}_t$, y_t được chuẩn hóa thành một vec-tơ dạng "one hot" có nghĩa là một vec-tơ với độ dài V có giá trị bằng 0 trừ vị trí ứng với lớp của y_t có giá trị 1. Như vậy để so sánh hai phân phối xác suất y và $\hat{y}$ ta sử dụng hàm độ lỗi *negative log-likelihood*

hay còn gọi là *cross entropy*, gọi L_t là độ lỗi tại một bước thời gian t, ta có:

$$L_t = -y_t^T \log(\hat{y}_t) \tag{2.7}$$

Gọi θ là một tham số của mô hình, ta biết rằng θ được chia sẻ trong quá trình học, tức là ở mọi bước thời gian, chúng đều có giá trị bằng nhau:

$$\theta_t = \theta_k \tag{2.8}$$

Với t, k là những bước thời gian ($t \neq k$). Tại thời điểm bắt đầu với $t = k = 0$ mọi θ_i đều có giá trị bằng nhau nên trong quá trình học, ta cần:

$$\frac{\partial L}{\partial \theta_t} = \frac{\partial L}{\partial \theta_k} \tag{2.9}$$

Với L là độ lỗi tổng hợp tại của tất cả các bước thời gian. Như vậy để $\theta_t = \theta_k, \forall t; k$, ta chỉ đơn giản xem L là tổng độ lỗi ở tất cả các bước thời gian. Với cách làm này, tham số luôn bằng nhau sau mỗi lần cập nhật.

$$L = \sum_t L_t = -\sum_t y_t^T \log(\hat{y}_t) \tag{2.10}$$

Mục tiêu của việc học là cực tiểu hóa độ lỗi tổng hợp L. Thuật toán "backpropagation" với *gradient descent* sẽ được áp dụng để huấn luyện RNN. Trên thực tế, người ta sẽ sử dụng một phiên bản của "gradient descent" là "mini-batch gradient descent" cho việc huấn luyện. Tập dữ liệu ban đầu sẽ được chia thành nhiều "mini-batch", mỗi "mini-batch" là một tập con với số lượng khoảng vài chục đến vài trăm mẫu thuộc tập dữ liệu ban đầu. Với mỗi lần duyệt (iteration), việc tính toán gradient để cập nhật các tham số học của mô hình được thực hiện lần lượt trên tất cả các mini-batch này.

Ta cần tìm bộ tham số $\theta \in \{W_{hy}, W_{hh}, W_{xh}, b_y, b_h\}$ sao cho cực tiểu hóa hàm độ lỗi L. Theo thuật toán "gradient descent", bộ tham số được cập nhật theo công thức:

$$\theta \leftarrow \theta - \eta \nabla_\theta L \tag{2.11}$$

Ở đây, $\nabla_\theta L$ là "gradient" của hàm độ lỗi ứng với tham số θ. η được gọi là hệ số học

(learning rate) là một siêu tham số quyết định rằng θ nên thay đổi nhiều bao nhiêu khi "gradient" ứng với tham số thay đổi. Trong phần dưới đây, chúng tôi sẽ trình bày việc tính toán "gradient" của hàm độ lỗi theo bộ tham số học $\theta \in \{W_{hy}, W_{hh}, W_{xh}, b_y, b_h\}$.

Với $s_t = W_{hy}h_t + b_y$ và $\hat{y}_t = softmax(s_t)$, sử dụng "chain rule" trong tính đạo hàm ta được:

$$\nabla_\theta L_t = \frac{\partial L_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial s_t}\frac{\partial s_t}{\partial h_t}\frac{\partial h_t}{\partial \theta} = \frac{\partial L_t}{\partial s_t}\frac{\partial s_t}{\partial h_t}\frac{\partial h_t}{\partial \theta} \tag{2.12}$$

Và ta có (công thức này được chứng minh trong phần mở rộng):

$$\nabla_{s_t} L_t = \frac{\partial L_t}{\partial s_t} = \hat{y}_t - y \tag{2.13}$$

Ta cũng dễ dàng tính được:

$$\frac{\partial s_t}{\partial h_t} = W_{hy}^T \tag{2.14}$$

Đến đây, ta chỉ cần tìm $\frac{\partial h_t}{\partial \theta}$ với $\theta \in \{W_{hy}, W_{hh}, W_{xh}, b_y, b_h\}$. Ta có thể chia các tham số thành hai nhóm. Nhóm đầu tiên là các tham số liên quan đến quá trình tính toán đầu ra, bao gồm $\theta^{(1)} \in \{W_{hy}, b_y\}$. Những tham số này không tham gia vào hàm hồi quy h_t cho nên "gradient" ứng với chúng được tính một cách dễ dàng:

$$\nabla_{W_{hy}} L_t = \frac{\partial L_t}{\partial s_t}\frac{\partial s_t}{\partial W_{hy}} = (\hat{y}_t - y)h_t^T \tag{2.15}$$

$$\nabla_{b_y} L_t = \frac{\partial L_t}{\partial s_t}\frac{\partial s_t}{\partial b_y} = \hat{y}_t - y \tag{2.16}$$

Nhóm thứ hai gồm những tham số tham gia vào quá trình hồi quy, bao gồm $\theta^{(2)} \in \{W_{hh}, W_{xh}, b_h\}$. Để tính được "gradient" ứng với $\theta^{(2)}$, ta cần tính được $\frac{\partial h_t}{\partial \theta^{(2)}}$

Vì h_t là một hàm hồi quy được xây dựng dựa trên $\theta^{(2)}$ và h_{t-1} nên "gradient" cuả h_t tương ứng với $\theta^{(2)}$ được tính dựa trên quy tắc "total derivative". Quy tắc này nói rằng nếu $f(x,y)$ với $x, y \in \mathbb{R}^M$, giả sử x, y là những hàm số của r sao cho $x = x(r); y = y(r)$ thì ta có:

$$\frac{\partial f}{\partial r} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial r} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial r} \tag{2.17}$$

Áp dụng vào trường hợp của $\frac{\partial h_t}{\partial \theta^{(2)}}$ ta được:

$$\frac{\partial h_t}{\partial \theta^{(2)}} = \frac{\partial h_t}{\partial \theta^{(2)}} + \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial \theta^{(2)}} \tag{2.18}$$

Tuy nhiên, ta có thể khai triển công thức trên một lần nữa với cách làm tương tự cho $\frac{\partial h_{t-1}}{\partial \theta^{(2)}}$:

$$\frac{\partial h_t}{\partial \theta^{(2)}} = \frac{\partial h_t}{\partial \theta^{(2)}} + \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial \theta^{(2)}} + \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \frac{\partial h_{t-2}}{\partial \theta^{(2)}} \tag{2.19}$$

Khai triển trên sẽ kéo dài cho đến khi gặp $\frac{\partial h_0}{\partial \theta^{(2)}}$. Và để ý rằng:

$$\frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \frac{\partial h_{t-2}}{\partial \theta^{(2)}} = \frac{\partial h_t}{\partial h_{t-2}} \frac{\partial h_{t-2}}{\partial \theta^{(2)}} \tag{2.20}$$

$$\frac{\partial h_t}{\partial \theta^{(2)}} = \frac{\partial h_t}{\partial h_t} \frac{\partial h_t}{\partial \theta^{(2)}} \tag{2.21}$$

Với cách khai triển như vậy, ta có công thức tổng quát cho $\frac{\partial h_t}{\partial \theta^{(2)}}$, đó là:

$$\frac{\partial h_t}{\partial \theta^{(2)}} = \sum_{r=0}^{t} \frac{\partial h_t}{\partial h_r} \frac{\partial^{\dagger} h_r}{\partial \theta^{(2)}} \tag{2.22}$$

trong đó $\frac{\partial^{\dagger} h_r}{\partial \theta^{(2)}}$ là đạo hàm "lập tức", có nghĩa là khi lấy đạo hàm h_r theo $\theta^{(2)}$, h_{r-1} được coi là hằng số.

Lần lượt thay thế $\theta^{(2)}$ bằng W_{hh}, W_{xh}, b_h ta có "gradient" ứng với từng tham số là:

$$\nabla_{W_{hh}} L_t = \frac{\partial L_t}{\partial s_t} \frac{\partial s_t}{\partial h_t} \frac{\partial h_t}{\partial W_{hh}} = (\hat{y}_t - y) W_{hy}^T \sum_{r=0}^{t} \frac{\partial h_t}{\partial h_r} \frac{\partial h_r}{\partial W_{hh}} \tag{2.23}$$

$$\nabla_{W_{xh}} L_t = \frac{\partial L_t}{\partial s_t} \frac{\partial s_t}{\partial h_t} \frac{\partial h_t}{\partial W_{xh}} = (\hat{y}_t - y) W_{hy}^T \sum_{r=0}^{t} \frac{\partial h_t}{\partial h_r} \frac{\partial h_r}{\partial W_{xh}} \tag{2.24}$$

$$\nabla_{b_h} L_t = \frac{\partial L_t}{\partial s_t} \frac{\partial s_t}{\partial h_t} \frac{\partial h_t}{\partial b_h} = (\hat{y}_t - y) W_{hy}^T \sum_{r=0}^{t} \frac{\partial h_t}{\partial h_r} \frac{\partial h_r}{\partial b_h} \tag{2.25}$$

Để tính $\frac{\partial h_t}{\partial h_r}$ ta áp dụng "chain rule" từ bước thời gian thứ t đến r

$$\frac{\partial h_t}{\partial h_r} = \prod_{i=r}^{t} \frac{\partial h_i}{\partial h_{i-1}} \tag{2.26}$$

với $h_{i+1}, h_i \in \mathbb{R}^n$ nên $\frac{\partial h_i}{\partial h_{i-1}}$ là một ma trận "Jacobian"

$$\frac{\partial h_i}{\partial h_{i-1}} = \left[\frac{h_i}{h_{i-1,1}}, ..., \frac{h_i}{h_{i-1,n}} \right] \tag{2.27}$$

$$= \begin{bmatrix} \frac{h_{i,1}}{h_{i-1,1}} & \cdots & \frac{h_{i,1}}{h_{i-1,n}} \\ \vdots & \ddots & \vdots \\ \frac{h_{i,n}}{h_{i-1,1}} & \cdots & \frac{h_{i,n}}{h_{i-1,n}} \end{bmatrix} = W_{hh}^T \text{diag}\left(\phi'(h_i)\right) \tag{2.28}$$

Như vậy, ta có "gradient" của các tham số qua tất cả các bước thời gian sẽ là:

$$\nabla_{b_y} L = \sum_t \hat{y}_t - y \tag{2.29}$$

$$\nabla_{b_h} L = \sum_t \sum_{r=0}^{t} (\hat{y}_t - y) W_{hy}^T \frac{\partial h_t}{\partial h_r} \frac{\partial h_r}{\partial b_h} \tag{2.30}$$

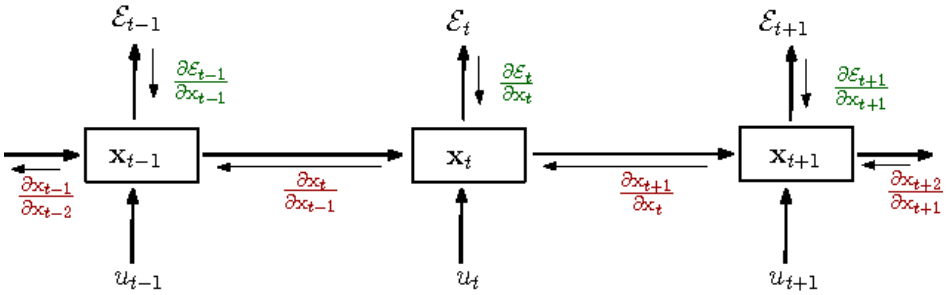
$$\nabla_{W_{hy}} L = \sum_t (\hat{y}_t - y) h_t^T \tag{2.31}$$

$$\nabla_{W_{hh}} L = \sum_t \sum_{r=0}^{t} (\hat{y}_t - y) W_{hy}^T \frac{\partial h_t}{\partial h_r} \frac{\partial h_r}{\partial W_{hh}} \tag{2.32}$$

$$\nabla_{W_{xh}} L_t = \sum_t \sum_{r=0}^{t} (\hat{y}_t - y) W_{hy}^T \frac{\partial h_t}{\partial h_r} \frac{\partial h_r}{\partial W_{xh}} \tag{2.33}$$

Thách thức trong việc học các phụ thuộc dài hạn

Theo thuật toán BPTT, ta biết rằng gradient theo một tham số hồi quy ở mỗi bước thời gian $\nabla_{W_{hh}} L_t$ là tổng gradient của L_t theo tham số đó ở tất cả trạng thái ẩn tại các bước thời gian trước đó $1, ..., t$ (công thức 2.32). Chính bởi cơ chế tính toán gradient như vậy, trên lý thuyết RNN có khả năng học được những phụ thuộc dài hạn với độ dài



Hình 2.3: Tại mỗi bước thời gian t, đội lỗi không chỉ được lan truyền qua các tầng ở bước thời gian hiện tại mà còn phải lan truyền thông tin độ lỗi qua tất cả thời điểm trước đó. Chính sự lan truyền qua thời gian này gây ra hiện tượng bùng nổ hoặc biến mất gradient.

bất kỳ. Tuy nhiên, trong thực tế, việc học các phụ thuộc dài hạn là một vấn đề lớn của RNN được gây ra bởi hai nguyên nhân: sự biến mất gradient (vanishing gradient) và sự bùng nổ gradient (exploding gradient). Nói một cách đơn giản, gradient của một hàm mục tiêu ứng với một tham số nói lên rằng hàm số kia sẽ thay đổi bao nhiêu khi tham số thay đổi. Sự biến mất gradient xảy ra khi gradient trở nên cực kỳ nhỏ; nó khiến cho hay sự thay đổi của tham số không ảnh hưởng đến sự thay đổi của hàm mục tiêu. Ngược lại sự bùng nổ gradient khiến gradient trở nên lớn một cách đột ngột; điều này khiến cho một sự thay đổi nhỏ trong tham số cũng ảnh hưởng mạnh đến hàm mục tiêu, hoặc đơn giản là gradient lớn đến mức không thể tính toán được. Sự biến mất và bùng nổ gradient được phát hiện và trình bày trong những nghiên cứu [**?**], [**?**], [**?**]. Trong mục này, chúng tôi sẽ trình bày lại nguyên nhân và mô tả một số giải pháp cho hai vấn đề này. Những chứng minh của chúng tôi dựa trên chứng minh trong nghiên cứu [**?**].

Nhắc lại rằng trong RNN, gradient của hàm lỗi tại bước thời gian t theo tham số hồi quy $\theta \in \{W_{hh}, W_{xh}, b_h\}$ là:

$$\nabla_\theta L = \frac{\partial L_t}{\partial s_t} \frac{\partial s_t}{\partial h_t} \sum_{r=0}^{t} \frac{\partial h_t}{\partial h_r} \frac{\partial h_r}{\partial \theta} \tag{2.34}$$

và

$$\frac{\partial h_t}{\partial h_r} = \prod_{i=r}^{t} W_{hh}^T \text{diag}\left(\phi'(h_i)\right) \tag{2.35}$$

Đầu tiên, để cho đơn giản, ta hãy xem ϕ là hàm một hàm đồng nhất (identity

function) với $\phi(x) = x$ như vậy theo công thức trên, ta có:

$$\frac{\partial h_t}{\partial h_r} = \prod_{i=r}^{t} W_{hh}^T = \left(W_{hh}^T\right)^{t-r} \tag{2.36}$$

Với W_{hh} là một ma trận vuông, nó có thể được phân tích thành dạng:

$$W_{hh} = \Sigma \text{diag}\,(\lambda)\,\Sigma^{-1} \tag{2.37}$$

Với Σ, λ lần lượt là ma trận vec-tơ riêng và vec-tơ trị riêng của ma trận W_{hh}. Ta biết rằng lũy thừa của W_{hh} cũng chính là lũy thừa của vec-tơ trị riêng của nó λ. Khi số mũ của phép lũy thừa lớn, những vec-tơ riêng ứng với trị riêng $\lambda_i < 1$ sẽ giảm theo hàm mũ, những vec-tơ riêng ứng với trị riêng $\lambda_i > 1$ sẽ tăng theo hàm mũ. Nói cách khác, *điều kiện đủ* để xảy ra "gradient" biến mất là trị riêng lớn nhất của ma trận kết nối hồi quy λ_{max} có giá trị nhỏ hơn 1. Để xảy ra "gradient" bùng nổ, *điều kiện cần* là $\lambda_{max} > 1$.

Trong trường hợp tổng quát với một hàm kích hoạt ϕ bất kỳ, với ϕ' bị chặn trên bởi $\gamma \in \mathbb{R}$ và do đó $||diag(\phi'(h_k))|| \leq \gamma$. Với $\lambda_{max} < \frac{1}{\gamma}$, hiện tượng "gradient" biến mất sẽ xảy ra. Theo công thức 2.28:

$$\forall i, \left\| \frac{\partial h_{i+1}}{\partial h_i} \right\| \leq \left\| W_{hh}^T \right\| \left\| diag(\phi'(h_k)) \right\| < \frac{1}{\gamma}\gamma < 1 \tag{2.38}$$

Đặt $\beta \in \mathbb{R}$ sao cho $\forall i, \left\| \frac{\partial h_{i+1}}{\partial h_i} \right\| \leq \beta < 1$, như vậy ta có thể thấy được rằng:

$$\lim_{t-r\to\infty} \frac{\partial h_t}{\partial h_r} = \prod_{i=r}^{t} W_{hh}^T \text{diag}\left(\phi'(h_i)\right) \leq \beta^{t-r} = 0 \tag{2.39}$$

Bằng cách đảo ngược chứng minh này ta được *điều kiện cần* để xảy ra "gradient" bùng nổ là trị riêng lớn nhất λ_{max} lớn hơn $\frac{1}{\gamma}$.

Chứng minh trên cho thấy rằng, khi $t - r$ lớn, tức là khoảng cách giữa từ đang xét và một từ trong quá khứ là lớn thì $\nabla_{\theta_r} L_t$ sẽ hoặc rất bé nếu $\lambda_{max} < \frac{1}{\gamma}$ hoặc có khả năng trở nên rất lớn nếu $\lambda_{max} > \frac{1}{\gamma}$. Trong thực tế, với ϕ là hàm **tanh** ta có $\gamma = 1$ trong khi với ϕ là hàm sigmoid, ta có $\gamma = 1/4$. Nếu ta khởi tạo tham số θ nhỏ hoặc dùng hàm kích hoạt có γ nhỏ như hàm **sigmoid** thì sự biến mất gradient sẽ dễ rất dễ xảy ra; nó

khiến cho mô hình chỉ học được những phụ thuộc cục bộ. Ngược lại, nếu θ được khởi tạo với giá trị lớn, gradient tại các bước thời gian ở xa sẽ bùng nổ và kết quả là mô hình không thể học được [**?**].

Một kỹ thuật để đối phó với sự bùng nổ gradient được đề là chuẩn hóa gradient về một giá trị nếu nó vượt quá một ngưỡng nào đó. Kỹ thuật này gọi là "gradient clipping" được đề xuất bởi Thomas Mikolov và trình bày lại trong [**?**]. Cụ thể, ta đặt một ngưỡng là chặn trên cho gradient, bất cứ khi nào độ lớn của gradient lớn hơn ngưỡng này ta sẽ chuẩn hóa trở về một giá trị nhỏ hơn. Giải pháp này được áp dụng trong thực tế để ngăn chặn các giá trị **NaN** (Not a Number) trong gradient và cho phép quá trình huấn luyện tiếp tục. "Gradient clipping" được trình bày trong thuật toán 2.1.

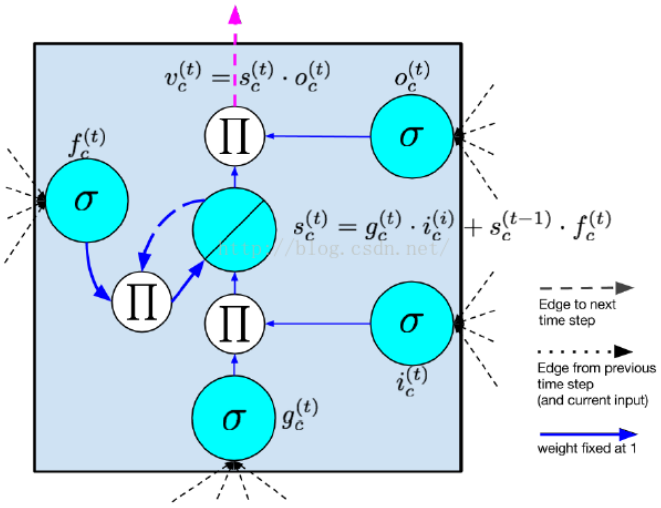
Thuật toán 2.1 Gradient clipping

Đầu vào: Gradient ứng với tham số θ
Đầu ra: Gradient ứng với tham số θ được chuẩn hóa về một ngưỡng *threshold*
Thao tác:
1: $\hat{g} \leftarrow \frac{\partial E}{\partial \theta}$
2: **if** $\|\hat{g}\| \geq threshold$ **then**
3: $\quad \hat{g} \leftarrow \frac{threshold}{\|\hat{g}\|}\hat{g}$
4: **end if**

Long short-term memory (LSTM)

Hochreiter và Schmidhuber [**?**] đã giới thiệu mô hình *Long short-term memory* (LSTM) chủ yếu ở để khắc phục vấn đề biến mất gradient trong RNN. Nhớ lại rằng trong RNN, chính việc mô hình hóa phụ thuộc thời gian dựa vào ma trận trọng số W_{hh} đã gây ra hiện tượng gradient biến mất. Giả sử gọi S_t là trạng thái liên kết hồi quy (trong RNN, nó chính là trạng thái ẩn). Ý tưởng của LSTM là thay vì tính toán S_t từ S_{t-1} với một phép nhân ma trận theo sau là hàm kích hoạt phi tuyến, LSTM trực tiếp tính toán một ΔS_t sau đó nó được cộng với S_{t-1} để tạo ra S_t. Thoạt nhìn, sự khác biệt này có thể không đáng kể khi mà chúng ta đều đạt được S_t trong cả hai cách. Tuy nhiên, với cách làm này, các "gradient" của LSTM tính toán ΔS_t sẽ không bị biến mất.

Thuật ngữ "Long short-term memory" xuất phát từ nhận định sau. Mạng RNN đơn giản có "long term memory" (bộ nhớ dài hạn) dưới dạng các ma trận trọng số.



Hình 2.4: Mô hình RNN đơn giản với kết nối vòng, *h* được xem như bộ nhớ được luân chuyển trong RNN. Chú ý rằng đường nét đứt ở đầu ra thể hiện rằng tại một thời điểm *t*, RNN có thể có hoặc không có một đầu ra.

Những ma trận trọng số này thay đổi một cách chậm rãi trong quá trình học nhằm mã hóa kiến thức về dữ liệu. RNN cũng có "short-term memory" (bộ nhớ ngắn hạn) dưới dạng các kích hoạt tạm thời, được truyền từ mỗi bước thời gian sang các bước thời gian sau đó. "Long short-term memory" tạm dịch là "bộ nhớ ngắn hạn dài" cho phép mở rộng bộ nhớ ngắn hạn bằng cách thêm vào một loại lưu trữ trung gian gọi là trạng thái lưu giữ (cell state). Trạng thái lưu giữ này có khả năng lưu giữ các thông tin cần thiết một cách lâu dài dưới dạng một bộ nhớ ngắn hạn. Để làm được điều này, LSTM sử dụng một cơ chế gọi là "cổng", các "cổng" giúp được huấn luyện để chọn lọc thông tin nào là cần thiết để tác động lên trạng thái lưu trữ. Với cách làm này, trạng thái lưu trữ sẽ lưu được nhiều thông tin hơn, vì chỉ những thông tin quan trọng mới tồn tại trong nó.

Về cấu tạo, một LSTM tương tự như một RNN một lớp ẩn, nhưng mỗi "RNN cell" (ký hiệu "A" trong hình 2.2) được thay thế bằng một "memory cell" (hình 2.4). Giống như "RNN cell", "memory cell" nhận một đầu vào bên ngoài và phát sinh một đầu ra cũng như là truyền đi một trạng thái ẩn sang "memory cell" ở bước thời gian kế tiếp. Tuy nhiên, trong "memory cell" còn có thêm một trạng thái lưu giữ cũng được truyền đi như một trạng thái ẩn. Cấu tạo chi tiết của LSTM sẽ được trình bày trong phần dưới đây, cấu tạo này dựa trên phiên bản LSTM của [**?**].

- *Nút đầu vào (input node)*: Đơn vị này được ký hiệu là g, là một mạng nơ-ron một tầng ẩn. Nút đầu vào có nhiệm vụ mô hình hóa đầu vào tại mỗi bước thời gian. Nó nhận tham số là đầu vào tại bước thời gian hiện tại x_t và trạng thái ẩn tại thời điểm trước đó h_{t-1}. Cụ thể, tại mỗi bước thời gian nút đầu vào có công thức:

$$g_t = \phi\left(W_{gx}x_t + W_{gh}h_{t-1} + b_g\right) \tag{2.40}$$

- *Cổng vào (input gate)*: "Cổng" như đã nói, là một cơ chế đặc biệt của LSTM. Cổng vào cũng được cấu tạo giống như nút đầu vào, nó nhận tham số là x_t và h_{t-1}. Sau đó được đưa qua hàm kích hoạt sigmoid để tạo ra giá trị trong khoảng $(0,1)$. Sở dĩ đơn vị này được gọi là "cổng vào" vì giá trị của nó sẽ được sử dụng để nhân với giá trị của nút đầu vào. Giá trị của nó thể hiện lượng thông tin mà nút đầu vào được phép truyền đi. Nếu cổng vào bằng 0, nút đầu vào sẽ truyền đi với giá trị 0. Nếu cổng vào bằng 1, nút đầu vào sẽ truyền đi với giá trị ban đầu. Cụ thể hơn, ta có công thức của cổng vào, được ký hiệu là i, tại bước thời gian t:

$$i_t = \sigma\left(W_{ix}x_t + W_{ih}h_{t-1} + b_i\right) \tag{2.41}$$

- *Trạng thái lưu giữ (cell state)*: Trái tim của LSTM chính là trạng thái lưu trữ, là một mạng nơ-ron với hàm kích hoạt tuyến tính. Khá giống với trạng thái ẩn trong RNN, trạng thái lưu trữ s_t cũng có một kết nối hồi quy với trạng thái lưu trữ trước đó s_{t-1}. Tuy nhiên, trọng số kết nối hồi quy luôn có giá trị cố định là 1. Bởi vì kết nối hồi quy này qua nhiều bước đều có trọng số không đổi nên khi tính toán, "gradient" của độ lỗi không bị bùng nổ hay biến mất. Tại mỗi bước thời gian, trạng thái lưu trữ được tính như sau:

$$s_t = s_{t-1} + g_t \odot i_t \tag{2.42}$$

- *Cổng quên (forget gate)*: Cổng quên là một đề xuất của [**?**] so với bài báo LSTM gốc. Thay vì kiểm soát lượng thông tin để đưa vào trạng thái lưu giữ như cổng vào, cổng quên cung cấp khả năng tẩy đi một lượng thông tin trong trạng thái lưu giữ. Cụ thể, cổng quên với giá trị thuộc khoảng $(0,1)$ sẽ được nhân với s_{t-1} trong công thức 2.42. Tại mỗi bước thời gian, giá trị của cổng quên f_t được tính

như sau:

$$f_t = \sigma\left(W_{fx}x_t + W_{fh}h_{t-1} + b_f\right) \tag{2.43}$$

Công thức của trạng thái lưu trữ được sửa lại khi có cổng quên:

$$s_t = s_{t-1} \odot f_t + g_t \odot i_t \tag{2.44}$$

- *Cổng ra (output gate)*: Giá trị đầu ra v_t của "memory cell" tại mỗi bước thời gian chính là tích giá trị của trạng thái lưu trữ s_t với giá trị của cổng ra o_t. Trong một số phiên bản của LSTM, hàm kích hoạt ϕ có thể là hàm **tanh** hoặc **sigmoid** hoặc không sử dụng hàm kích hoạt nào. Tại mỗi bước thời gian, giá trị của cổng ra o_t được tính như sau:

$$o_t = \sigma\left(W_{ox}x_t + W_{oh}h_{t-1} + b_o\right) \tag{2.45}$$

Đầu ra của "memory cell" cũng chính là trạng thái ẩn h_t có giá trị:

$$h_t = \phi(s_t) \odot o_t \tag{2.46}$$

LSTM xếp chồng (Stacked LSTM)

LSTM hai chiều (Bidirectional LSTM)

Mô hình ngôn ngữ

Mô hình ngôn ngữ *n*-gram

Như chúng tôi đã đề cập trong chương 1, *mô hình ngôn ngữ* (language model) là một bộ phận quan trọng trong cả dịch máy thống kê và dịch máy nơ-ron. Nó giúp ta dự đoán được từ tiếp theo trong một chuỗi từ cho trước. Cụ thể hơn, mô hình ngôn ngữ là một phân phối xác suất trên một chuỗi các từ. Cho trước một chuỗi $w_1, w_2, ..., w_n$ (ký hiệu $w_{1:n}$), mô hình ngôn ngữ gán cho nó một xác suất $p(w_{1:n})$ đại diện cho độ "trơn

tru" của chuỗi đó. Theo công thức xác suất có điều kiện, ta có:

$$\begin{aligned} p(w_{1:n}) &= p(w_1)p(w_2|w_1)p(w_3|w_{1:2})...p(w_n|w_{1:n-1}) \\ &= \prod_{t=1}^{n} p(w_t|w_{1:t-1}) \end{aligned} \tag{2.47}$$

Trong công thức trên, mỗi thành phần $p(w_i|w_{1:i})$ là xác suất có điều kiện của từ w_i biết những từ trước đó $w_{1:i}$, những từ này còn được gọi là ngữ cảnh đối với từ đang xét. Ta thấy rằng để tính được xác suất $p(w_i|w_{1:i})$ ta phải xét đến tất cả những từ đứng trước nó. Điều này làm cho chi phí tính toán trở nên rất lớn. Để giảm chi phí tính toán, người ta sử dụng giả định *markov* (markov-assumption); giả định này nói rằng các từ tiếp theo chỉ liên quan đến từ hiện tại và độc lập với các từ trước đó. Chính xác hơn, một giả định markov bậc k nói rằng từ tiếp theo trong một chuỗi chỉ phụ thuộc vào k từ cuối cùng trong chuỗi.

$$p(w_{i+1}|w_{1:i}) \approx p(w_{i+1}|w_{i-k:i}) \tag{2.48}$$

Công thức 2.48 theo giả định markov trở thành:

$$p(w_{1:n}) = \prod_{t=1}^{n} \approx p(w_{i-k}|w_{i-k:i-1}) \tag{2.49}$$

Mặc dù giả định markov bậc k rõ ràng là sai với k bất kỳ (một câu có thể có phụ thuộc dài hạn với chiều dài bất kỳ như câu bắt đầu bằng từ *what* và kết thúc bằng dấu *?*), tuy nhiên nó vẫn tạo ra những mô hình ngôn ngữ mạnh mẽ với với các giá trị tương đối nhỏ của k ($k = 3$ hoặc $k = 5$), và là phương pháp chủ yếu cho bài toán mô hình hóa ngôn ngữ trong hàng thập kỷ.

Cách tiếp cận truyền thống cho bài toán mô hình hóa ngôn ngữ là sử dụng mô hình ngôn ngữ n-gram. Sở dĩ nó có tên như vậy là vì nó sử dụng giả định markov bậc n - 1 để ước lượng xác suất $p\left(w_{i+1} = m|w_{1:i}\right) \approx p\left(w_{i+1} = m|w_{i-(n-1):i}\right)$. Theo mô hình ngôn ngữ n-gram, xác suất để từ m theo sau chuỗi các từ $w_{1:i}$ là:

$$p\left(w_{i+1} = m|w_{i-(n-1):i}\right) = \frac{\#\left(w_{i-(n-1):i+1}\right)}{\#\left(w_{i-(n-1):i}\right)} \tag{2.50}$$

trong đó $\#(w_{i:j})$ là số lần xuất hiện của chuỗi $w_{i:j}$ trong tập ngữ liệu. Các mô hình n-gram thông thường sử dụng $n = 2$ và $n = 1$ được gọi lần lượt là mô hình *trigram* và mô hình *bigram*. Trong trường hợp mô hình không sử dụng ngữ cảnh để ước lược lượng xác suất của một từ ($n = 0$), thì mô hình này được gọi là mô hình *unigram*.

Bên cạnh sự hiệu quả, phương pháp này có một khuyết điểm lớn: nếu chuỗi $w_{i-(n-1):i+1}$ chưa từng xuất hiện trong tập ngữ liệu, tức $\#\left(w_{i-(n-1):i+1}\right) = 0$ thì xác suất ước lượng sẽ có giá trị bằng 0. Điều này dẫn đến xác suất của toàn bộ chuỗi cũng bằng 0 do phép nhân trong công thức tính của nó (công thức 2.49). Việc xác suất bằng 0 xảy ra khá thường xuyên do sự giới hạn của tập ngữ liệu.

Một cách để tránh việc xảy ra các sự kiện xác suất bằng không là sử dụng những kỹ thuật *làm mịn* (smoothing techniques). Làm mịn bảo đảo tất cả mọi chuỗi đều có một xác suất xuất hiện (mặc dù nhỏ). Kỹ thuật làm mịn đơn giản nhất là làm mịn thêm α (add-α smoothing) [**?**]; nó bảo đảm bất kỳ chuỗi nào cũng xuất hiện ít nhất α lần. Với làm mịn thêm α ta được:

$$p_{add-k}\left(w_{i+1} = m | w_{i-k:i}\right) = \frac{\#\left(w_{i-(n-1):i+1}\right) + \alpha}{\#\left(w_{i-(n-1):i}\right) + \alpha |V|} \tag{2.51}$$

trong đó $|V|$ là số lượng từ vựng trong tập ngữ liệu.

Một kỹ thuật khác bên cạnh làm mịn là kỹ thuật *truy hồi* (back-off): nếu chuỗi n-gram chưa từng xuất hiện trong tập ngữ liệu thì chúng ta sẽ tính xác suất của chuỗi dựa trên (n - 1)-gram. Chúng ta tiếp tục truy hồi cho đến khi chúng ta gặp một chuỗi mà tần xuất của nó khác không. Ví dụ với trường hợp *trigram*:

$$\begin{aligned} p_{bo}\left(w_{i+1} = m | w_{i-1}, w_{i-2}\right) = \lambda_1 p\left(w_{i+1} = m | w_{i-1}, w_{i-2}\right) \\ + \lambda_2 p\left(w_{i+1} = m | w_{i-2}\right) \\ + \lambda_3 p\left(w_{i+1} = m\right) \end{aligned} \tag{2.52}$$

trong đó các λ có tổng là 1:

$$\sum_i \lambda_i = 1 \tag{2.53}$$