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

PHẠM ĐÌNH VƯƠNG

LÊ HOÀNG TRUNG

DỊCH MÁY BẰNG MÔ HÌNH HỌC LSTM-ATTENTION

Chuyên ngành: Khoa Học Máy Tính

Mã số chuyên ngành: 60 48 01

KHÓA LUẬN TỐT NGHIỆP: KHOA HỌC MÁY TÍNH

NGƯỜI HƯỚNG DẪN KHOA HỌC:

ThS Trần Trung Kiên

Tp. Hồ Chí Minh, Năm 2018

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

DANH MỤC HÌNH ẢNH

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*.

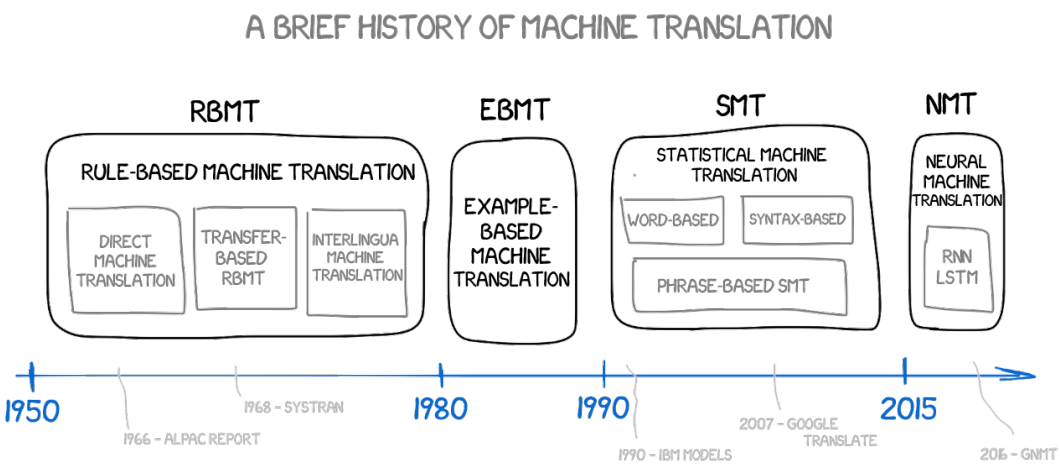
Dịch máy là quá trình chuyển đổi văn bản/tiếng nói từ ngôn ngữ này sang dạng tương ứng của nó trong một ngôn ngữ khác, được thực hiện bởi một chương trình máy tính nhằm mục đích cung cấp bản dịch tốt nhất mà không cần sự trợ giúp của con người.

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 [5]. 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 ngôn ngữ tự động (Automatic Language Processing Advisory) của Hoa Kỳ, tuyên bố



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

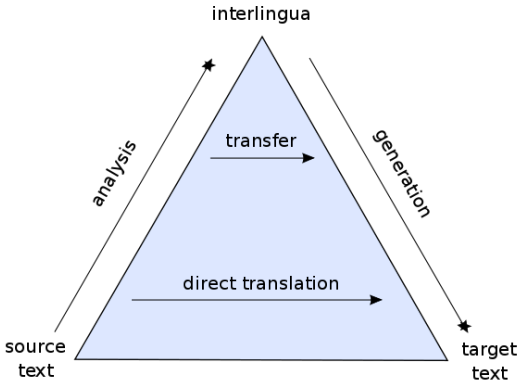
rằng dịch máy là đắt tiền, không chính xác và không mang lại kết quả hứa hẹn [5]. Thay vào đó, họ đề nghị tập trung vào phát triển các từ điển, điều này đã loại bỏ các nhà nghiên cứu Mỹ ra khỏi cuộc đua trong gần một thập kỷ.

1.1 Các phương pháp Dịch máy

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



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)

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 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 [**?**] và Systrans [**?**]. 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, dịch máy dựa trên ngữ liệu bao gồm hai phương pháp: dịch máy dựa trên ví dụ và dịch máy thống kê.

Dịch máy dựa trên ví dụ (Example-based Machine Translation - EBMT):

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.

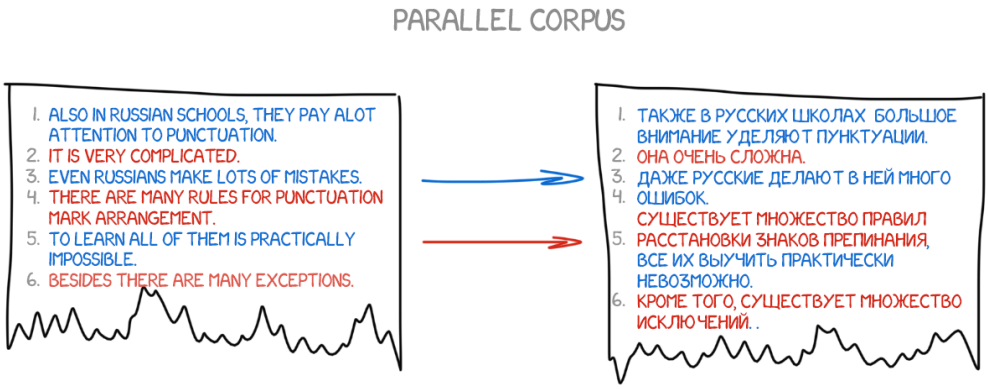
Để hiểu rõ hơn về dịch máy thống kê, xét một ví dụ: ta cần dịch một câu f trong tiếng Pháp sang dạng tiếng Anh e của nó. Có nhiều bản dịch có thể có cuả f trong tiếng Anh, việc cần làm là chọn e sao cho nó là bản dịch "tốt nhất" của f. Chúng ta có thể mô hình hóa quá trình này bằng một xác suất có điều 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



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

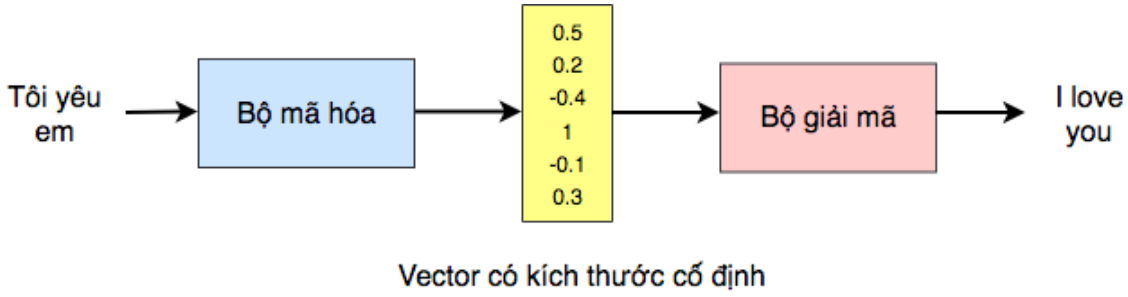
cũng là một nhược điểm của SMT. Mô hình ngôn ngữ là một chủ đề quan trọng và sẽ được chúng tôi đề cập lại một lần nữa trong chương Kiến thức nền tảng.

1.2 Dịch máy Nơ-ron

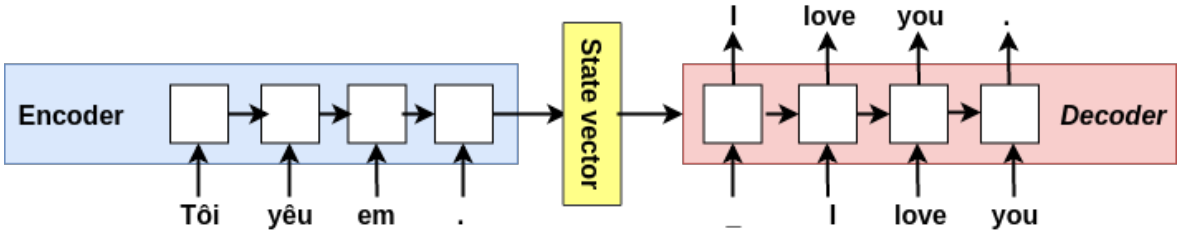
Mặc dù trên thực tế đã có nhiều hệ thống dịch máy được phát triển dựa trên dịch máy thống kê thời bấy giờ, tuy nhiên nó không hoạt động thực sự tốt bởi một số nguyên nhân. Một là việc những từ hay đoạn được dịch cục bộ và quan hệ của chúng với những từ cách xa trong câu nguồn thường bị bỏ qua. Hai là 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 trong dịch máy trong những năm gần đây được đề xuất đầu tiên bởi [7], [14], [1]. 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) ([14], [1]). 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



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

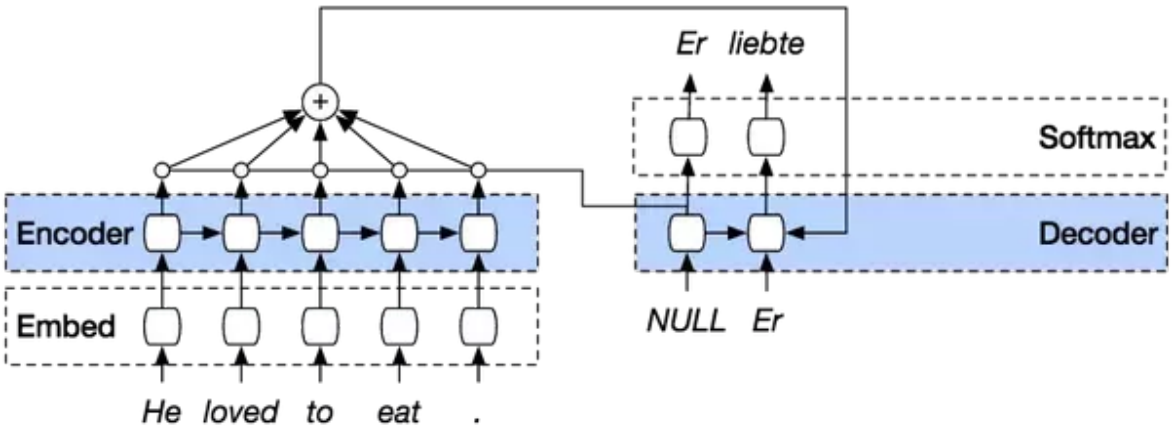


Hình 1.5: Kiến trúc bộ mã hóa - bộ giải mã được xây dựng trên mạng nơ-ron hồi quy

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 1.2.

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 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 1.3 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 2, 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



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

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 [12]. Để giải quyết vấn đề này, thay vì dùng mạng nơ-ron hồi quy thuần, người ta sử dụng các biến thể của nó quy 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 khi đưa vào bộ mã hóa tạo ra kết quả tốt hơn một cách đáng kể [14] 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.

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 các từ trong bản dịch, bộ giải mã sẽ "nhìn" 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.

1.3 Cấu trúc của khóa luận

Trong khóa luận này, chúng tôi quyết định tập trung nghiên cứu về dịch máy nơ-ron và cơ chế Attention dựa trên nghiên cứu của nhóm tác giả tại đại học Stanford bao gồm Minh-Thang Luong, Hieu Pham, Christopher Manning trong bài báo *Effective Approaches to Attention-based Neural Machine Translation* [10]. 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.
- Kết luận và hướng phát triển của luận văn.

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, mô hình ngôn ngữ nơ-ron và mô hình dịch máy 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. Một phiên bản cải tiến của RNN là *Long short-term memory* cũng được chúng tôi trình bày, phiên bản này giúp cho việc huấn luyện RNN trở nên dễ dàng hơn. Sau đó, dựa trên những kiến thức về mạng nơ-ron hồi quy, chúng tôi nói về khái niệm *mô hình ngôn ngữ* với chức năng tạo ra từ trong bộ giải mã, là bước quan trọng trong dịch máy nơ-ron. Cuối cùng, chúng tôi cũng trình bày về mô hình dịch máy nơ-ron theo kiến trúc bộ mã hóa - bộ giải mã với RNN và mô hình ngôn ngữ hồi quy là những thành phần nền tảng.

2.1 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).

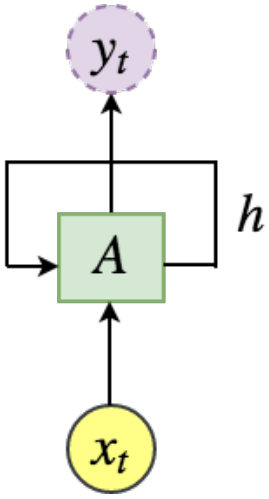
Mạng nơ-ron hồi quy (recurrent neural network) [3] gọi tắt là *RNN* là một nhánh của nạng nơ-ron nhân tạo được thiết kế đặc biệt cho việc mô hình hóa dữ liệu chuỗi. Khác với những mô hình đã đề cập giả định sự phụ thuộc chỉ xảy ra trong một vùng có chiều dài cố định. RNN, trên lý thuyết, có khả năng 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.

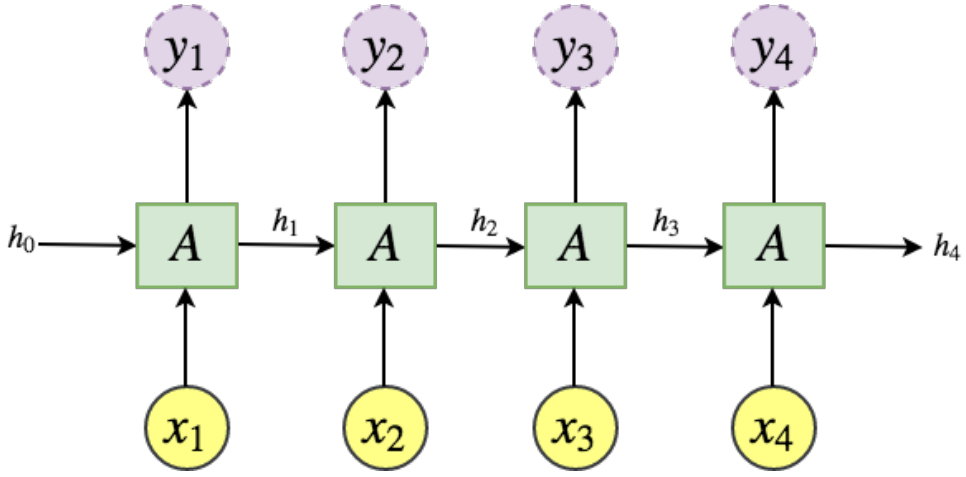
RNN với công thức cụ thể như sau:

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

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.



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

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 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.

2.1.1 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:

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

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

$$y_t = softmax(s_t) \tag{2.6}$$

Ta xác định hàm độ lỗi độ lỗi giữa đầu ra dự đoán $\hat{y}_t$ và đầu ra thật sự y_t. Gọi V 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 V. 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*:

$$E_t = -y_t \log(\hat{y}_t) \tag{2.7}$$

trong đó E_t là độ lỗi tại một bước thời gian t. Độ lỗi của toàn bộ quá trình học E là tổng của độ lỗi tại của tất cả các bước thời gian.

$$E = \sum_t E_t = -\sum_t y_t \log(\hat{y}_t) \tag{2.8}$$

Mục tiêu của quá trình học là cực tiểu hóa độ lỗi tổng hợp E. Thuật toán "back-propagation" 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.

Mục tiêu của việc học là tìm bộ tham số $\theta = \left(W_{hy}, W_{hh}, W_{xh}, b_y, b_h\right)$ sao cho cực tiểu hóa hàm độ lỗi E. Theo thuật toán "gradient descent", bộ tham số được cập nhật theo công thức :

$$\theta \leftarrow \theta - \eta \frac{\partial E}{\partial \theta} \tag{2.9}$$

Ở đây, $\frac{\partial E}{\partial \theta}$ là "gradient" của hàm độ lỗi ứng với các tham số của mô hình. 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 các bộ số học $\theta = \left(W_{hy}, W_{hh}, W_{xh}, b_y, b_h\right)$. Bộ tham số này có thể được chia làm hai nhóm, bao gồm: $\left(W_{hy}, b_y\right)$ và (W_{hh}, W_{xh}, b_h).

Gradient theo W_{hy} và b_y

Bởi vì W_{hy} và b_y chỉ hiện diện trong hàm $\hat{y}$. Với $s_t = W_{hy}h_t + b_y$ và $\hat{y}_t = softmax(s_t)$, sử dụng công thức nhân trong tính đạo hàm ta được:

$$\frac{\partial E_t}{\partial W_{hy}} = \frac{\partial E_t}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial s_t} \frac{\partial s_t}{\partial W_{hy}} \tag{2.10}$$

Từ công thức 2.7 ta có:

$$\frac{\partial E_t}{\partial \hat{y}} = -\frac{y_t}{\hat{y}_t} \tag{2.11}$$

Hàm $\hat{y}$ là một hàm softmax nên nó có đạo hàm:

$$\frac{\partial \hat{y}_t}{\partial s_t} = \begin{cases} -\hat{y}_{t_k}\hat{y}_{t_l}, & k \neq l \\ \hat{y}_{t_k}\left(1 - \hat{y}_{t_k}\right), & k = l \end{cases}. \tag{2.12}$$

Kết hợp 2.11 và 2.12 ta có được:

$$-\frac{y_{t_l}}{\hat{y}_{t_l}}\hat{y}_{t_l}\left(1 - \hat{y}_{t_l}\right) + \sum_{k \neq l}\left(-\frac{y_{t_k}}{\hat{y}_{t_k}}\right)\left(-\hat{y}_{t_k}\hat{y}_{t_l}\right) = -y_{t_l} + y_{t_l}\hat{y}_{t_l} + \sum_{k \neq l} y_{t_k}\hat{y}_{t_l} \tag{2.13a}$$

$$= -y_{t_l} + \hat{y}_{t_l}\sum_k y_{t_k}. \tag{2.13b}$$

Lưu ý rằng y_t là "one-hot" vec-tơ nên tổng trong công thức trên bằng 1, cho nên:

$$\frac{\partial E_t}{\partial s_t} = \hat{y}_t - y_t \tag{2.14}$$

Bởi vì W_{hy} được chia sẻ trên toàn bộ chuỗi, do đó đạo hàm hàm độ lỗi tổng hợp E theo W_{hy} sẽ là tổng đạo hàm của các E_t theo W_{hy}. Từ công thức 2.14 ta có được:

$$\frac{\partial E}{\partial W_{hy}} = \sum_t \frac{\partial E_t}{\partial s_t}\frac{\partial s_t}{\partial W_{hy}} = \sum_t (\hat{y}_t - y_t) \otimes h_t \tag{2.15}$$

trong đó $\otimes$ là "outer-product" của hai vec-tơ.

Tương tự với đạo hàm của E theo b_y, ta cũng có:

$$\frac{\partial E}{\partial b_y} = \sum_t \frac{\partial E_t}{\partial s_t}\frac{\partial s_t}{\partial b_y} = \sum_t \hat{y}_t - y_t \tag{2.16}$$

Gradient theo W_{hh}, W_{xh} và b_h

Tham số W_{hh} tồn tại ở cả trạng thái ẩn h_t và đầu ra dự đoán $\hat{y}_t$, để tính "gradient" theo W_{hh}. Chúng ta cũng để ý rằng $\hat{y}_t$ cũng dựa trên W_{hh} trực tiếp và gián tiếp (thông qua h_{t-1}). Đặt $p_t = W_{hx}x_t + W_{hh}h_{t-1}$ và $h_t = \tanh(p_t)$:

$$\frac{\partial E_t}{\partial W_{hh}} = \frac{\partial E_t}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial s_t} \frac{\partial s_t}{\partial h_t} \frac{\partial h_t}{\partial W_{hh}} \tag{2.17}$$

Trong công thức trên, ta đã biết $\frac{\partial E_t}{\partial \hat{y}}$ và $\frac{\partial \hat{y}}{\partial s_t}$ trong phần trước, công thức tính $\frac{\partial s_t}{\partial h_t}$ khá đơn giản:

$$\frac{\partial s_t}{\partial h_t} = W_{hy} \tag{2.18}$$

Cuối cùng, để tính được $\frac{\partial h_t}{\partial W_{hh}}$ ta có quan sát rằng có một sự phụ thuộc giữa h_t và W_{hh} thông qua trạng thái ẩn trước đó h_{t-1}. Ta biết rằng nếu $f(x,y)$ với $x,y \in \mathbb{R}^N$, giả sử x,y là những hàm số của r sao cho $x = x(r); y = y(r)$ thì ta có:

$$\frac{\partial f}{\partial r} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial r} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial r} \tag{2.19}$$

Áp dụng công thức trên để tính $\frac{\partial h_t}{\partial W_{hh}}$ ta được:

$$\frac{\partial h_t}{\partial W_{hh}} = \frac{\partial h_t}{\partial W_{hh}} + \frac{\partial h_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial W_{hh}} \tag{2.20}$$

Tuy nhiên, ta có thể áp dụng công thức trên một lần nữa với $\frac{\partial h_{t-1}}{\partial W_{hh}}$:

$$\frac{\partial h_t}{\partial W_{hh}} = \frac{\partial h_t}{\partial W_{hh}} + \frac{\partial h_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial W_{hh}} + \frac{\partial h_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial h_{t-2}}\frac{\partial h_{t-2}}{\partial W_{hh}} \tag{2.21}$$

Quá trình này tiếp tục cho đến khi chúng kết thúc ở h_0 là trạng thái ẩn khởi tạo. Có thể thấy

$$\frac{\partial h_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial h_{t-2}}\frac{\partial h_{t-2}}{\partial W_{hh}} = \frac{\partial h_t}{\partial h_{t-2}}\frac{\partial h_{t-2}}{\partial W_{hh}} \tag{2.22}$$

và:

$$\frac{\partial h_t}{\partial W_{hh}} = \frac{\partial h_t}{\partial h_t}\frac{\partial h_t}{\partial W_{hh}} \tag{2.23}$$

Như vậy, có thể rút gọn 2.21 thành một công thức duy nhất:

$$\frac{\partial h_t}{\partial W_{hh}} = \sum_{r=0}^{t} \frac{\partial h_t}{\partial h_r} \frac{\partial h_r}{\partial W_{hh}} \tag{2.24}$$

Kết hợp các công thức từ 2.17 suy ra:

$$\frac{\partial E_t}{\partial W_{hh}} = (\hat{y}_t - y_t) W_{hy} \sum_{r=0}^{t} \frac{\partial h_t}{\partial h_r} \frac{\partial h_r}{\partial W_{hh}} \tag{2.25}$$

Đạo hàm hàm độ lỗi tổng hợp E theo W_{hh} sẽ là tổng đạo hàm của các E_t theo W_{hh}

$$\frac{\partial E}{\partial W_{hh}} = \sum_{t=0}^{T} \sum_{r=0}^{t} (\hat{y}_t - y_t) W_{hy} \frac{\partial h_t}{\partial h_r} \frac{\partial h_r}{\partial W_{hh}} \tag{2.26}$$

Cũng giống như W_{hh}, trong công thức tính h_t, W_{hh} cũng có liên hệ với h_t một cách trực tiếp và với h_{t-1} một cách gián tiếp. Ta có:

$$\frac{\partial E_t}{\partial W_{xh}} = \frac{\partial E_t}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial s_t} \frac{\partial s_t}{\partial h_t} \frac{\partial h_t}{\partial W_{hx}} \tag{2.27}$$

Ta chỉ cần tính $\frac{\partial h_t}{\partial W_{hx}}$, theo cách tương tự như đã làm với W_{hh} ,ta được:

$$\frac{\partial h_t}{\partial W_{xh}} = \sum_{r=0}^{t} \frac{\partial h_t}{\partial h_r} \frac{\partial h_r}{\partial W_{xh}} \tag{2.28}$$

Như vậy cuối cùng ta được:

$$\frac{\partial E_t}{\partial W_{hh}} = (\hat{y}_t - y_t) W_{hy} \sum_{r=0}^{t} \frac{\partial h_t}{\partial h_r} \frac{\partial h_r}{\partial W_{xh}} \tag{2.29}$$

Điểm khác biệt giữa $\frac{\partial E_t}{\partial W_{hh}}$ và $\frac{\partial E_t}{\partial W_{xh}}$ là ở cách tính đạo hàm $\frac{\partial h_r}{\partial W_{hh}}$ và $\frac{\partial h_r}{\partial W_{xh}}$

Cuối cùng đạo hàm hàm độ lỗi tổng hợp E theo W_{hh} sẽ là tổng đạo hàm của các E_t theo W_{hh}

$$\frac{\partial E}{\partial W_{xh}} = \sum_{t=0}^{T} \sum_{r=0}^{t} (\hat{y}_t - y_t) W_{hy} \frac{\partial h_t}{\partial h_r} \frac{\partial h_r}{\partial W_{xh}} \tag{2.30}$$

Với những lập luận tương tự, ta cũng có đạo hàm hàm độ lỗi tổng hợp E theo b_h:

$$\frac{\partial E}{\partial b_h} = \sum_{t=0}^{T} \sum_{r=0}^{t} (\hat{y}_t - y_t) \frac{\partial h_t}{\partial h_r} \frac{\partial h_r}{\partial W_{xh}} \tag{2.31}$$

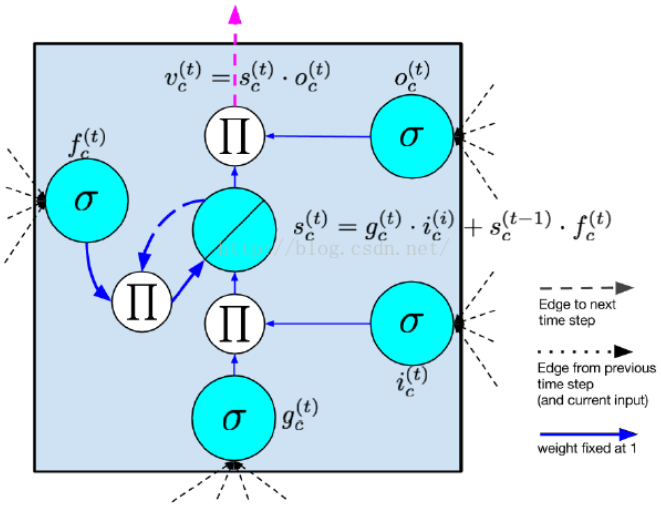
2.1.2 Khó khăn trong việc huấn luyện RNN

Trong công thức 2.31

Mặc dù "gradient" của RNN dễ tính toán, nhưng RNN cơ bản là khó huấn luyện. Những vấn đề này bao gồm gradient bùng nổ (exploiting gradients) và gradient biến mất (vanishing gradients) được đề cập trong các nghiên cứu [**?**] [**?**]. Nếu gradient bùng nổ, mô hình không thể học được. Nếu gradient biến mất, việc học những phụ thuộc dài hạn trở nên khó khăn.

2.2 Long short-term memory

Hochreiter và Schmidhuber [1997] đã giới thiệu mô hình LSTM chủ yếu ở để khắc phục vấn đề biến mất gradient. Mô hình này tương tự với mô hình RNN một lớp ẩn, nhưng mỗi "RNN cell" (được ký hiệu "A" trong hình 2.2) được thay thế bằng một "memory cell" (Hình 2.3). Giống như "RNN cell", "memory cell" cũng chứa một kết nối hồi quy nhằm để kết nối thông tin từ các bước thời gian với nhau. Tuy nhiên, trong "LSTM cell" kết nối hồi quy này có trọng số cố định là một. Điều này nhằm bảo đảm "gradient" được luân chuyển qua các bước thời gian mà không bị bùng nổ hay biến mất.



Hình 2.3: 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.

2.3 Mô hình ngôn ngữ

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. Cụ thể hơn, mô hình ngôn ngữ là một phân phối xác suất trên một chuỗi các từ. Cho trước một chuỗi $w_1, w_2, \ldots, w_m$ (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. Sử dụng quy tắc dây chuyền (chain rule) trong xác suất, ta có:

$$P(w_{1:n}) = P(w_1)P(w_2|w_1)P(w_3|w_{1:2})P(w_4|w_{1:3})...P(w_n|w_{1:n-1}) \tag{2.32}$$

Chương 3

Dịch máy bằng mô hình học LSTM-Attention

Chương này trình bày về việc tổ chức các LSTM theo kiến trúc Bộ mã hóa-Bộ giải mã cùng với sử dụng cơ chế Attention. Ở đây, chúng tôi tập trung tìm hiểu về LSTM trong kiến trúc Bộ mã hóa-Bộ giải mã và các phiên bản của cơ chế Attention, sau đó giải thích chúng dựa trên cơ sở Toán học. Cụ thể, về LSTM, chúng tôi sử dụng bi-LSTM và uni-LSTM, về Attention, chúng tôi tìm hiểu về hai phiên bản Toàn cục (Global) và Cục bộ (Local):

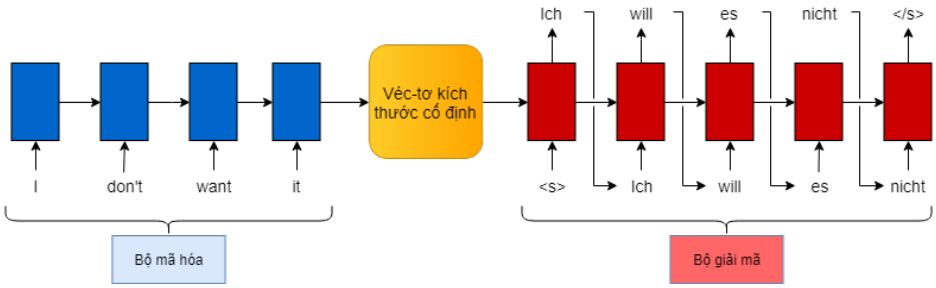
- *LSTM: chúng tôi tổ chức các bi-LSTM và uni-LSTM theo kiến trúc Bộ mã hóa-Bộ giải mã để tránh được hạn chế về số lượng từ ở ngôn ngữ nguồn và ngôn ngữ đích.*
- *Attention: chúng tôi tìm hiểu được sự hạn chế hiện có của kiến trúc Bộ mã hóa-Bộ giải mã khi thực hiện dịch những câu dài. Do vậy, chúng tôi sử dụng cơ chế Attention để giải quyết hạn chế đó bằng cách sử dụng thêm các trạng thái ẩn của bộ mã hóa trong quá trình giải mã.*
  - *Toàn cục: phiên bản này sử dụng tất cả trạng thái ẩn trong bộ mã hóa trong quá trình giải mã.*
  - *Cục bộ: phiên bản này chỉ sử dụng một số trạng thái ẩn trong bộ mã hóa trong quá trình giải mã.*

3.1 LSTM với kiến trúc Bộ mã hóa-Bộ giải mã

Hạn chế của việc sử dụng một mô hình LSTM khi áp dụng vào trong bài toán Dịch máy là ràng buộc chiều dài của câu đầu vào (ngôn ngữ nguồn) phải bằng chiều dài của câu đầu ra (ngôn ngữ đích). Bởi vì tại mỗi thời điểm, LSTM nhận đầu vào là một từ ở ngôn ngữ nguồn, sau đó thực hiện dự đoán một từ ở ngôn ngữ đích. Do vậy, một mô hình LSTM chỉ có thể dịch ổn trên những cặp ngôn ngữ rất giống nhau (ví dụ như Trung-Việt) nhưng chất lượng dịch vẫn rất khó đảm bảo vì thậm chí giữa cặp ngôn ngữ này dù rất giống nhau nhưng vẫn có cách diễn đạt, sử dụng từ ngữ khác nhau. // TODO: Đưa ra ví dụ. Dễ nhận thấy rằng đây là một giả định phi thực tế, dẫu là con người dịch thì cũng rất khó và phải là người am hiểu về cả 2 ngôn ngữ đó thì mới có thể dịch sang ngôn ngữ đích mà vẫn truyền đạt được đầy đủ ý nghĩa ở ngôn ngữ nguồn. Đối với ràng buộc chiều dài ở 2 câu phải bằng nhau thì chỉ phù hợp với khi dịch thơ có luật ràng buộc về số từ như thơ lục bát, thơ Đường v.v... Tuy nhiên, dù đã thỏa ràng buộc về số lượng từ nhưng vẫn khó đảm bảo về mặt ngữ nghĩa. Trong thơ văn, người ta sử dụng rất nhiều biện pháp tu từ, văn phong và ngữ pháp thì rất đa dạng, cách hành văn rất khác. Hơn nữa, một câu có thể biểu diễn bởi rất nhiều cách khác nhau. Minh chứng là một bài thơ tiếng Trung có thể có rất nhiều bản dịch khác nhau. Từ những lí do kể trên, sử dụng một mô hình LSTM để giải quyết bài toán Dịch máy là không phù hợp với thực tế.

Để giải quyết những hạn chế của đó, nhóm tác giả Sutskever, 2014 [14] đã đề xuất một kiến trúc tên gọi là Chuỗi tới Chuỗi (Sequence to Sequence - Seq2Seq). Tuy nhiên, mọi người thường gọi kiến trúc này là kiến trúc Bộ mã hóa-Bộ giải mã (Encoder-Decoder). Trong khóa luận này, chúng tôi sử dụng kiến trúc này để giải quyết bài toán Dịch máy và gọi nó là Bộ mã hóa-Bộ giải mã. Ý tưởng của kiến trúc này rất đơn giản. Kiến trúc này tận dụng các đặc tính của các mô hình LSTM và tổ chức chúng một cách hợp lí sao cho phù hợp với các yêu cầu của bài toán Dịch máy.

Kiến trúc Bộ mã hóa-Bộ giải mã như tên gọi, gồm có 2 bộ phận chính là bộ mã hóa và bộ giải mã. Mỗi một bộ phận sẽ là một mô hình học được lựa chọn sao cho phù hợp với bài toán và có thể hoạt động tốt với nhau. Đối với bài toán Dịch máy và trong phạm vi khóa luận này, mỗi bộ phận sẽ là một mạng nơ-ron hồi qui, cụ thể là các LSTM.



Hình 3.1: Minh họa kiến trúc Bộ mã hóa-Bộ giải mã. Kiến trúc có 2 bộ phận. Trong đó, bộ mã hóa là một bi-LSTM có nhiệm vụ nhận câu nguồn làm đầu vào và xuất ra một véc-tơ có kích thước cố định chứa thông tin cần thiết để dịch của toàn bộ câu nguồn. Bộ giải mã là một uni-LSTM có đầu vào là véc-tơ có kích thước cố định của bộ mã háo và đầu ra là câu đích.

Hình 3.1 minh họa kiến trúc Bộ mã hóa-Bộ giải mã trong việc giải quyết bài toán Dịch máy. Chúng tôi sẽ trình bày cụ thể về cách hoạt động của 2 bộ phận này trong các phần tiếp theo.

3.1.1 Bộ mã hóa

Bộ mã hóa là một bi-LSTM (LSTM 2 chiều). Bi-LSTM này nhận đầu vào là một câu ở ngôn ngữ nguồn $\boldsymbol{x} = \{x_0, ..., x_{S-1}\}$. Đầu ra là một véc-tơ có kích thước cố định. Véc-tơ này chứa các thông tin cần thiết để tiến hành dịch sang câu ở ngôn ngữ đích. Nhiệm vụ của bộ mã hóa (Bi-LSTM) là thực hiện mã hóa toàn bộ câu nguồn thành véc-tơ có chứa các thông tin hữu ích. Véc-tơ kích thước cố định này rất quan trọng, nó là tiền đề để việc dịch sang câu đích đạt chất lượng tốt.

Ở đây, chúng tôi sử dụng véc-tơ trạng thái ẩn h_t làm véc-tơ có kích thước cố định. Lí do bởi vì trong Bi-LSTM, trạng thái ẩn của một từ thứ t (thời điểm t) chứa những thông tin, mối quan hệ giữa từ này và những từ lân cận. Hơn nữa, nó còn chứa thông tin của toàn bộ những từ ở đầu câu cho tới từ thứ t. Vậy nên trạng thái ẩn của từ cuối cùng của câu sẽ chứa đựng thông tin của toàn bộ những từ trong câu. Và điều đó phù hợp với ý tưởng của véc-tơ có kích thước cố định của bộ mã hóa.

Trong thực tế, đầu ra của bộ mã hóa có thể là bất kì véc-tơ có tính chất như thế nào. Trong một số công trình sử dụng thêm một phép biến đổi trên trạng thái ẩn cuối cùng của bi-LSTM để tạo ra một véc-tơ có kích thước cố định. Đối với bài toán như Phát sinh câu miêu tả cho ảnh, đầu ra của bộ mã hóa là một véc-tơ kích thước cố định

chứa đặc trưng hữu ích của toàn bộ bức ảnh.

3.1.2 Bộ giải mã

Bộ giải mã là một uni-LSTM (LSTM 1 chiều). Uni-LSTM này nhận đầu vào là một véc-tơ kích thước cố định chứa thông tin cần thiết của câu cần dịch. Véc-tơ này là véc-tơ kích thước cố định từ đầu ra của bộ mã hóa. Đầu ra của bộ giải mã là câu đã được dịch sang ngôn ngữ đích $\boldsymbol{y} = \{y_0, ..., y_{S-1}\}$. Lí do chúng tôi sử dụng uni-LSTM mà không phải là bi-LSTM bởi vì trong quá trình giải mã (dịch) mô hình chỉ biết có được những từ đã được dịch ở trước đó (thông tin trong quá khứ) mà không biết được những từ đằng sau (thông tin trong tương lai) (vì chưa được dịch). Do đó, việc sử dụng lợi thế của bi-LSTM trong bộ giải mã là không thể và có thể gây ảnh hưởng tới chất lượng dịch của mô hình.

Một điều đáng lưu ý trong việc huấn luyện uni-LSTM trong bộ giải mã là đầu vào của uni-LSTM trong mỗi thời điểm t. Hình 3.1 minh họa bộ giải mã trong quá hình kiểm thử. Từ được dự đoán ở thời điểm $t-1$ sẽ là đầu vào của uni-LSTM ở thời điểm t. Tuy nhiên, việc như vậy sẽ là hạn chế lớn nếu sử dụng nó trong quá trình huấn luyện. Tại mỗi thời điểm t, khi thực hiện dự đoán từ đích tiếp theo, mô hình sẽ dựa vào những từ ở phía trước rồi sau đó sẽ tính độ lỗi dựa trên từ đúng được cung cấp. Do vậy, việc dự đoán tại thời điểm t có tốt hay không còn cần phải xem những từ đã được dự đoán ở phía trước có tốt hay không nữa. Nếu những từ ban đầu dự đoán không tốt, tất cả những được dự đoán ở phía sau sẽ càng kém hơn nữa. Để mô hình có thể học được cách dịch một câu dài với quá trình huấn luyện như trên thì phải tốn nhiều thời gian. Vì vậy, chúng tôi sử dụng phương pháp huấn luyện mạng nơ-ron hồi quy hiệu quả là *teacher forcing* // TODO: dịch. Phương pháp này đơn giản là tại mỗi thời điểm t, từ đúng tại t sẽ làm đầu vào của mạng tại thời điểm $t+1$. // TODO: thêm hình minh họa. Với phương pháp này, mô hình hội tụ nhanh hơn so với việc sử dụng cách huấn luyện ở trên. Bởi vì khi dịch những câu dài, lúc dự đoán một từ thì không phải phụ thuộc hoàn toàn vào những từ đã được dự đoán trước đó mà dựa vào những từ đúng đã được cung cấp. Khi thực hiện quá trình kiểm thử thì hiển nhiên sẽ phải dùng kết quả của những lần dự đoán phía trước để dự đoán từ tiếp theo hiện tại.

3.2 Cơ chế Attention

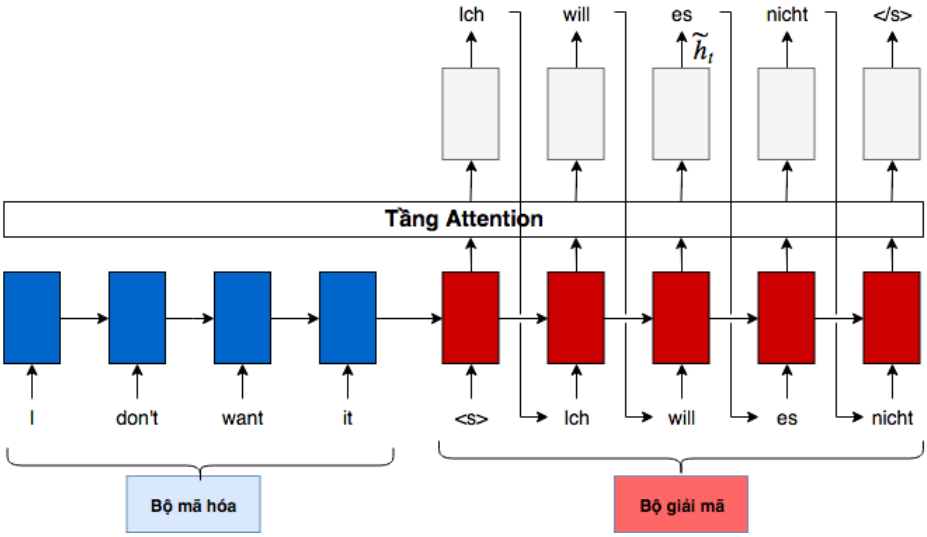
Ở phần trước, chúng tôi đã trình bày về kiến trúc Bộ mã hóa-Bộ giải mã cùng với những điểm mạnh của nó trong việc giải quyết bài toán Dịch máy. Tuy nhiên, kiến trúc này vẫn còn tồn tại hạn chế về việc dịch những câu dài do những thông tin được mã hóa của câu nguồn bị mất dần theo các thời điểm về sau. Lí do mà vấn đề này tồn tại thực chất là bởi vì các mô hình LSTM được sử dụng trong Bộ mã hóa và Bộ giải mã. Bản thân mô hình LSTM chưa thật sự giải quyết hoàn toàn vấn đề "sự phụ thuộc dài hạn". Để có thể vẫn tận dụng được các mô hình LSTM mà vẫn nâng cao được chất lượng dịch, chúng tôi sử dụng cơ chế Attention.

Trước khi đi vào cách hoạt động của cơ chế Attention, chúng tôi điểm qua một chút về nguồn cảm hứng và lịch sử của cơ chế này. Cơ chế Attention được lấy cảm hứng trên cơ chế đặt sự chú ý khi quan sát sự vật, hiện tượng của thị giác con người. Khi con người quan sát một sự vật, hiện tượng nào đó bằng mắt, con người chỉ có thể tập trung vào một vùng nhất định trên sự vật, hiện tượng được quan sát để ghi nhận thông tin. Sau đó, khi cần ghi nhận thêm thông tin khác, con người sẽ di chuyển vùng tập trung lên vật thể của mắt sang vị trí khác. Những vùng lân cận xung quanh vùng tập trung sẽ bị "mờ" hơn so với vùng tập trung. Cơ chế Attention đã được ứng dụng trong lĩnh vực Thị giác máy tính từ khá lâu [9] [2]. Vào những năm gần đây, cơ chế Attention được sử dụng cho các kiến trúc mạng nơ-ron hồi quy trên bài toán Dịch máy và đã đạt được những kết quả ấn tượng.

Cơ chế Attention được sử dụng trong đề tài này là một cơ chế sử dụng thông tin trong các trạng thái ẩn của RNN trong bộ mã hóa khi thực hiện quá trình giải mã. Cụ thể là:

- Trong quá trình giải mã, trước khi dự đoán đầu ra, bộ giải mã nhìn vào các thông tin nằm trong các trạng thái ẩn của RNN ở bộ mã hóa.
- Ở mỗi phần tử đầu ra tại thời điểm t, bộ giải mã dựa vào trạng thái ẩn tại thời điểm t hiện tại và quyết định sử dụng các thông tin trong trạng thái ẩn ở bộ mã hóa như thế nào.

2 phiên bản Toàn cục và Cục bộ mà trong khóa luận này chúng tôi trình bày là 2 cách mà cơ chế Attention sử dụng các trạng thái ẩn của RNN trong bộ mã hóa. Để làm rõ



Hình 3.2: Minh họa cơ chế Attention. Một tầng Attention được đặt ở trước bước dự đoán đầu ra của bộ giải mã.

hơn về ý tưởng của cơ chế Attention, dưới đây chúng tôi sẽ trình bày chi tiết về nền tảng Toán học của nó. Attention sử dụng thêm một số đại lượng:

- a_t: trọng số gióng hàng, a_t được tính theo công thức dưới đây:

$$a_t = \text{align}(h_t, \bar{h}_s) = \frac{\exp\left(\text{score}(h_t, \bar{h}_s)\right)}{\sum_{s'} \exp\left(\text{score}(h_t, \bar{h}_{s'})\right)} \tag{3.1}$$

  a_t là một véc-tơ chứa các điểm số giữa trạng thái ẩn ở thời điểm t h_t và các trạng thái ẩn ở câu nguồn $\bar{h}_s$. Hàm điểm số score mà chúng tôi sử dụng là gồm 2 hàm:

$$\text{score}(h_t, \bar{h}_s) = \begin{cases} h_t^T \bar{h}_s & dot \\ h_t^T W_a \bar{h}_s & general \end{cases} \tag{3.2}$$

  Đối với hàm score là hàm *dot*, mô hình chỉ đơn giản là thực hiện tính tích vô hướng giữa 2 trạng thái ẩn. Giá trị của hàm score này đạt cao nhất khi 2 véc-tơ trạng thái ẩn hoàn toàn giống nhau. Ưu điểm của hàng *dot* này là chi phí tính toán thấp nên thời gian huấn luyện và dự đoán nhanh. Đối với hàm score là hàm *general*, hàm này có sự tinh tế hơn hàm *dot*. Hàm *dot* thực hiện tính sự tương đồng lên tất cả cặp phần tử trong 2 véc-tơ, trong khi đó hàm *general* sử dụng thêm một bộ trọng số W_a, do đó những thông tin giữa hai trạng thái ẩn sẽ được

tính một cách chọn lọc hơn. Tuy nhiên, đổi lại thì hàm này sẽ có thời gian thực thi chậm hơn hàm *dot* một chút. Trong thực tế, không có minh chứng rõ ràng nào cho thấy rằng hàm nào sẽ tốt hơn, do vậy cần phải thực nghiệm cẩn thận để có được sự lựa chọn chính xác nhất.

- c_t: véc-tơ ngữ cảnh tại thời điểm t, là trung bình có trọng số của các trạng thái ẩn ở câu nguồn:

$$c_t = \sum_s a_{ts} h_s \tag{3.3}$$

  Véc-tơ c_t cho mô hình biết thông tin rằng với trạng thái ẩn hiện tại (chứa thông tin của quá trình dịch trước đó) thì ngữ cảnh hiện của thời điểm t hiện tại là gì. Ngữ cảnh đó được thể hiện thông qua những thông tin của các trạng thái ẩn h_s của câu nguồn mà được lựa chọn một cách có chọn lọc (có trọng số). Véc-tơ ngữ cảnh c_t là một cách biểu diễn ngữ cảnh của ngôn ngữ đích bằng ngữ cảnh của ngôn ngữ nguồn. Trong quá trình dịch, bộ giải mã cần phải dự đoán từ tiếp theo của câu dịch. Để dự đoán được chính xác, mô hình cần phải biết được ngữ cảnh hiện tại của câu là gì. Để đảm bảo ngữ cảnh mà mô hình nhận được chính xác, mô hình không thể chỉ dựa vào các trạng thái ẩn của bộ giải mã ở các thời điểm trước đó. Do vậy, mô hình sử dụng thêm các trạng thái ẩn của các từ ở câu nguồn để thể hiện ngữ cảnh một cách chính xác hơn.

- $\tilde{h}_t$, véc-tơ attention tại thời điểm t, được tính như sau:

$$\tilde{\boldsymbol{h}}_{\boldsymbol{t}} = \tanh(\boldsymbol{W_c}[\boldsymbol{c_t}; \boldsymbol{h_t}]) \tag{3.4}$$

  Véc-tơ attention chứa thông tin gióng hàng và trạng thái ẩn của thời điểm t hiện tại. Nhờ đó, mô hình nắm giữ được nhiều thông tin hơn để có thể dự đoán tốt hơn.

Bước dự đoán đầu ra không thay đổi ngoài trạng thái ẩn $\boldsymbol{h_t}$ được thay thế bởi véc-tơ attention $\tilde{\boldsymbol{h}}_{\boldsymbol{t}}$. $\tilde{\boldsymbol{h}}_{\boldsymbol{t}}$ được đưa qua tầng softmax để cho ra phân bố xác suất dự đoán trên các từ:

$$p(y_t | y_{<t}, x) = \text{softmax}(\boldsymbol{W_s}\tilde{\boldsymbol{h}}) \tag{3.5}$$

Nói một cách đơn giản, mục tiêu của cơ chế Attention là xoay quanh việc tìm véc-tơ

ngữ cảnh c_t một cách hiệu quả. Tiếp theo, chúng tôi trình bày chi tiết hơn về 2 phiên bản Toàn cục và Cục bộ. 2 phiên bản này chỉ khác nhau về cách suy ra véc-tơ ngữ cảnh $\boldsymbol{c_t}$, còn các bước còn lại giống nhau. Quy trình tính toán của cơ chế Attention: $h_t -> a_t -> c_t -> \tilde{h}_t$

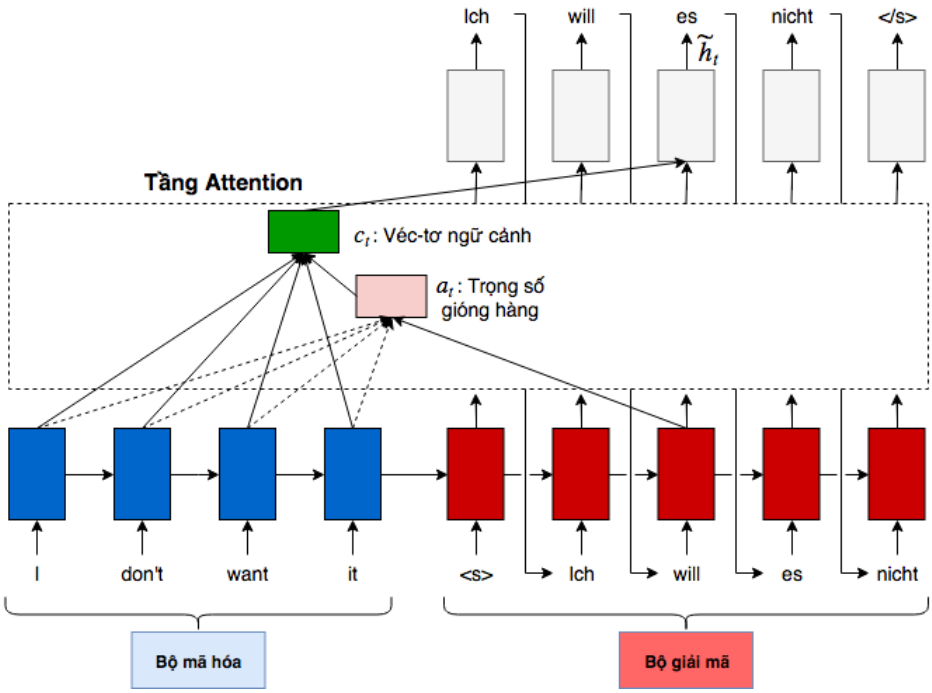
3.3 Attention Toàn cục

Ý tưởng của Attention toàn cục là nhìn vào toàn bộ các vị trí nguồn (các trạng thái ẩn của RNN ở bộ mã hóa) khi thực hiện giải mã. Khi đó trọng số gióng hàng a_t là một véc-tơ có kích thước thay đổi và bằng số trạng thái ẩn (số từ) ở câu nguồn: $\text{len}(a_t) = S$.

$$a_t = \text{align}(h_t, \bar{h}_s) = \frac{\exp\left(\text{score}(h_t, \bar{h}_s)\right)}{\sum_{s'=1}^{S} \exp\left(\text{score}(h_t, \bar{h}_{s'})\right)} \tag{3.6}$$

Hình 3.3 minh họa cơ chế Attention Toàn cục cho thấy rằng tại thời điểm t, trước khi thực hiện dự đoán từ tiếp theo, mô hình đặt "sự chú ý" lên toàn bộ các trạng thái ẩn ở câu nguồn hay bộ mã hóa.

Ưu điểm của phương pháp này là ý tưởng đơn giản, dễ cài đặt nhưng vẫn đạt được hiệu quả tốt (sẽ được trình bày ở phần thực nghiệm). Tuy nhiên, ý tưởng này vẫn còn chưa thực sự tự nhiên và còn hạn chế. Khi dịch một từ thì không cần phải đặt "sự chú ý" lên toàn bộ câu nguồn, chỉ cần đặt "sự chú ý" lên một số từ cần thiết. Mặc dù khi mô hình Attention Toàn cục được huấn luyện tốt thì hoàn toàn có thể chỉ đặt "sự chú ý" lên một số từ thật sự cần thiết. Trong thực tế, để đạt được độ chính xác như thế thì phải tiêu tốn nhiều tài nguyên cho việc huấn luyện mô hình như tài nguyên về tập dữ liệu đủ lớn, đủ tốt hay thời gian huấn luyện phải đủ lâu. Nhưng dễ thấy rằng dù bản thân mô hình đã học được cách đặt "sự chú ý" thật tốt nhưng vẫn phải tiêu tốn chi phí cho việc tính toán trọng số gióng hàng a_t cho những vị trí không cần thiết. Để giải quyết hạn chế trên của Attention Toàn cục, chúng tôi đã tìm hiểu và sử dụng phiên bản tinh tế hơn, đó là mô hình Attention Cục bộ. Ở phần tiếp theo, chúng tôi sẽ trình bày về mô hình này.



Hình 3.3: Minh họa cơ chế Attention Toàn cục. Tại thời điểm t, bộ giải mã nhìn vào toàn bộ trạng thái ẩn ở các vị trí nguồn. Trọng số gióng hàng và véc-tơ ngữ cảnh được tính dựa trên những trạng thái ẩn được "nhìn" bởi bộ giải mã. Sau đó véc-tơ ngữ cảnh sẽ được nối với trạng thái ẩn ở bộ giải mã ở thời điểm hiện tại để tạo thành véc-tơ attention. Sau đó mô hình sẽ dự đoán từ tiếp theo dựa trên véc-tơ attention này.

3.4 Attention Cục bộ

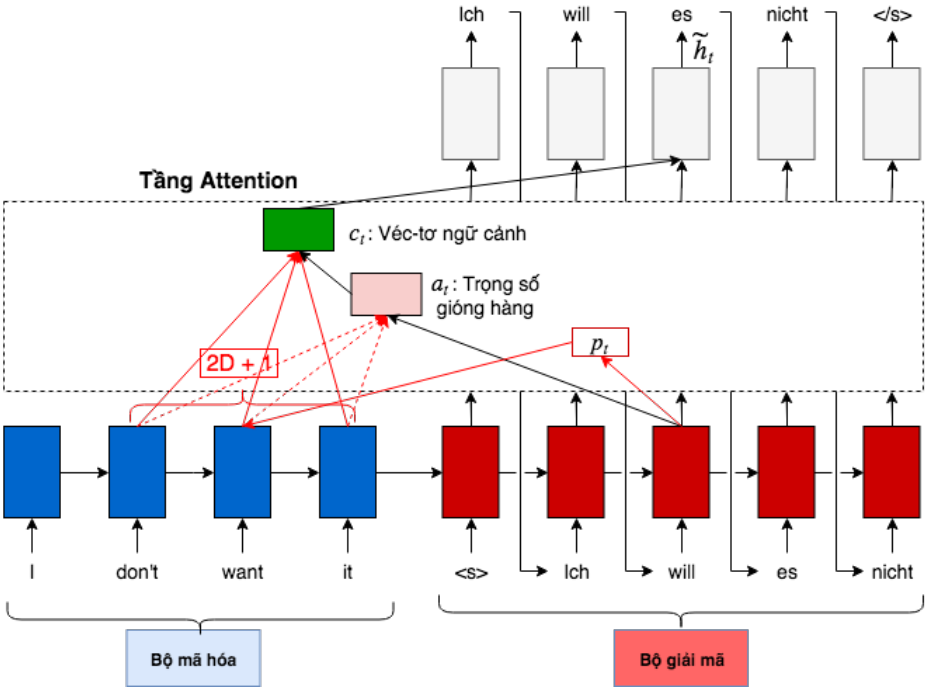
Như đã nêu ở phần trước, Attention Toàn cục có một hạn chế là đặt "sự chú ý" lên toàn bộ các từ ở câu nguồn khi dịch từng từ ở câu đích. Điều này gây tiêu tốn chi phí tính toán và có thể tạo ra những câu dịch không thực tế khi dịch những câu dài như trong các đoạn văn hay trong một tài liệu. Attention Cục bộ ra đời để giải quyết hạn chế này. Lưu ý, ý tưởng của cơ chế Attention Cục bộ này dựa vào đặc điểm của 2 cặp ngôn ngữ tiếng Anh và tiếng Đức. Do đó, sự hiệu quả của cơ chế này không đảm bảo cho các cặp ngôn ngữ khác.

Khi dịch mỗi từ ở câu đích, Attention Cục bộ chỉ đặt "sự chú ý" lên một số từ gần nhau ở câu nguồn. Mô hình này lấy cảm hứng từ sự đánh đổi giữa 2 mô hình "soft attention" và "hard attention" được đề xuất trong công trình Show, Attend and Tell [15] để giải quyết bài toán Phát sinh câu miêu tả cho ảnh (Image Captioning). Trong công trình [15], Attention Toàn cục tương ứng với "soft attention", "sự chú ý" được đặt trên toàn bộ bức ảnh. Còn "hard attention" thì đặt "sự chú ý" lên một số phần của bức ảnh.

Dễ thấy, với cách hoạt động chỉ tập trung một số các từ gần nhau ở câu nguồn, mô hình hoạt động gần với cách con người tập trung vào một sự vật, hiện tượng nào đó. Chi phí cho huấn luyện và dự đoán sẽ được giảm bớt bởi vì chúng ta chỉ thực hiện tính véc-tơ trọng số gióng hàng a_t cho những từ mà mô hình đặt "sự chú ý" lên.

Để làm rõ hơn về cách thức hoạt động của mô hình Attention Cục bộ, chúng tôi sẽ trình bày cụ thể hơn về nền tảng Toán học của mô hình này. Bên cạnh những đại lượng đã có ở mô hình Attention Toàn cục, Attention Cục bộ có thêm và thay đổi một số đại lượng như sau:

- p_t: vị trí đã được gióng hàng. Tại mỗi thời điểm t, mô hình sẽ phát sinh một số thực p_t. Số thực này có giá trị nằm trong đoạn $[0, S]$ với ý nghĩa rằng đây là vị trí đã được gióng hàng của với từ ở câu nguồn tại thời điểm t hiện tại. Hay nói cách khác, "sự chú ý" được đặt trên từ có vị trí p_t này. Để ý thấy rằng có sự không tự nhiên khi p_t là một số thực, do vậy p_t không thể cho biết được chính xác từ nào sẽ được đặt "sự chú ý" lên. Thực tế, với miền giá trị số thực, p_t có tác dụng là dùng để làm vị trí trung tâm cho các từ lân cận. Để làm rõ hơn về vấn đề này, chúng tôi sẽ trình bày rõ ràng hơn ở sau.



Hình 3.4: Minh họa cơ chế Attention Cục bộ. Tại thời điểm t, bộ giải mã nhìn vào một số trạng thái ẩn ở các vị trí nguồn nằm trong phạm vi của cửa sổ có kích thước $2D+1$. Trọng số gióng hàng và véc-tơ ngữ cảnh được tính dựa trên những trạng thái ẩn được bộ giải mã đặt sự chú ý.

- Đối quá trình tính véc-tơ ngữ cảnh c_t có sự thay đổi rằng mô hình xét các vị trí ở câu nguồn mà nằm xung quanh vị trí p_t một đoạn D. D là một đại lượng với miền số nguyên lớn hơn 0 và được gọi là kích thước cửa sổ. Cụ thể:

$$c_t = \sum_{x \in [p_t - D, p_t + D]} a_{tx} \tilde{h}_x \tag{3.7}$$

  D là một siêu tham số của mô hình. Việc lựa chọn giá trị của D là dựa vào thực nghiệm. Theo đề xuất của [10], chúng tôi lựa chọn $D = 10$.

Hình 3.4 minh họa cơ chế Attention Cục bộ cho thấy cách hoạt động của Attention Cục bộ cùng với sự khác biệt giữa Attention Cục bộ và Attention Toàn cục.

Mô hình Attention Cục bộ có 2 biến thể:

- Gióng hàng đều (monotonic alignment - local-m): vị trí được gióng hàng được phát sinh một cách đơn giản bằng cách cho $p_t = t$ tại mỗi thời điểm t. Ta giả định rằng các từ ở câu nguồn và các từ ở câu đích được gióng hàng đều nhau

theo từng từ.

- Gióng hàng dự đoán (predictive alignment - local-p): giả định rằng tất cả từ ở câu nguồn và câu đích đều được gióng hàng đều nhau không thực tế vì giữa 2 ngôn ngữ có ngữ pháp riêng và trật tự từ khác nhau. Do vậy, mô hình sẽ phát sinh vị trí được gióng hàng p_t một cách tự nhiên hơn cho phù hợp đặc điểm của ngôn ngữ. Cụ thể mô hình sẽ phát sinh vị trí p_t tại mỗi thời điểm t như sau:

$$p_t = S \cdot \text{sigmoid}(v_p^T \tanh(W_p h_t)) \tag{3.8}$$

Trong đó, v_p và W_p là 2 tham số mới của mô hình dùng cho việc dự đoán vị trí p_t. Mô hình cần học 2 tham số này để có thể dự đoán vị trí p_t được chính xác. Miền giá trị của $p_t \in [0, S]$. Để "ưu tiên" các vị trí được gióng hàng p_t, mô hình thêm vào trọng số gióng hàng của những từ lân cận đó một lượng có giá trị bằng giá trị của phân phối chuẩn (Gauss) mà đã được đơn giản hóa (chưa được chuẩn hóa) với giá trị kì vọng p_t và độ lệch chuẩn $\sigma = \frac{D}{2}$:

$$p_t = \text{align}(h_t, \bar{h}_s) \exp\left(-\frac{(s - p_t)^2}{2\sigma^2}\right) \tag{3.9}$$

Mô hình sử dụng hàm gióng hàng như các phiên bản trước. s là giá trị số nguyên thể hiện các vị trí nằm xung quanh p_t mà nằm trong cửa sổ D.

Đối với những vị trí s nằm ngoài câu (cửa sổ D vượt qua các biên của câu) thì mô hình sẽ bỏ qua những vị trí s nằm ngoài và chỉ xem xét những vị trí s nằm trong biên của câu.

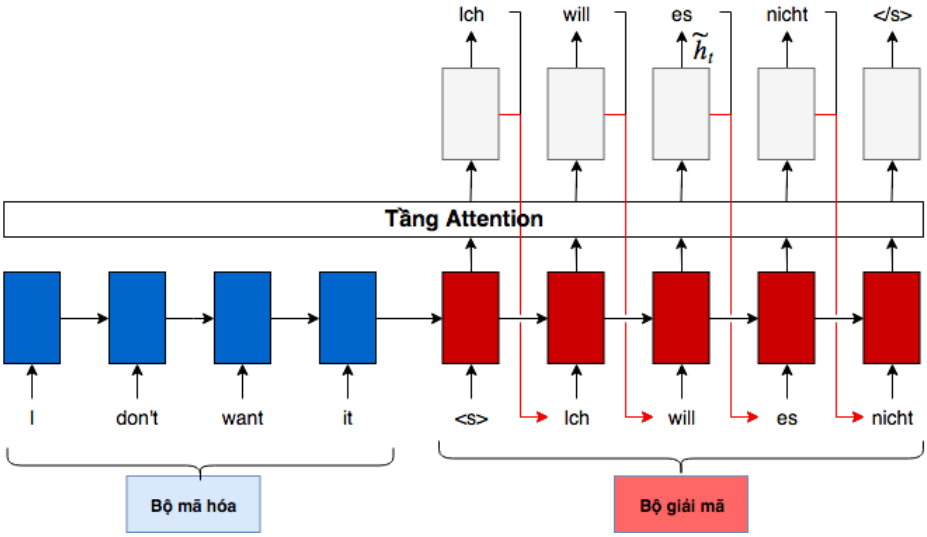
Véc-tơ trọng số gióng hàng a_t ở Attention Cục bộ có kích thước cố định $\in \mathbb{R}^{2D+1}$ và thường ngắn hơn a_t ở Attention Toàn cục. Local-p và local-m giống nhau chỉ khác rằng local-p tính vị trí p_t một cách linh hoạt và sử dụng một phân phối chuẩn đã được đơn giản hóa để điều chỉnh các trọng số gióng hàng $\text{align}(h_t, \bar{h}_s)$. Việc sử dụng thêm phân phối chuẩn để khuyến khích mô hình đặt "sự chú ý" lên vị trí p_t và phân chia dần cho các vị trí lân cận. Nếu không có việc sử dụng phân phối chuẩn này, mô hình có thể sẽ đặt "sự chú ý" hoàn toàn lên các từ lân cận xung quanh p_t mà không phải là vị trí p_t. Điều này không phù hợp với ý tưởng ban đầu của việc phát sinh vị trí p_t do vị trí này thể hiện ý nghĩa rằng bộ giải mã đang tập trung vào những từ ở vị trí p_t. Với

cơ chế được trình bày cụ thể như trên, mô hình Attention Cục bộ hoạt động tự nhiên hơn, phù hợp với ý tưởng về cách con người đặt "sự chú ý" khi quan sát sự vật, hiện tượng. Bên cạnh đó, Attention Cục bộ giảm chi phí tính toán của mô hình khi chỉ thực hiện tính trên những từ được chú ý nằm trong phạm vị cửa sổ nhất định khi câu nguồn dài.

3.5 Phương pháp Input feeding

Trong quá trình dịch, các mô hình được đề cập ở trên như Attention Toàn cục hay Cục bộ, đều vẫn còn một hạn chế về cách đặt "sự chú ý" hay gióng hàng lên các vị trí nguồn. Ở mỗi thời điểm t khi dịch một từ ở câu đích, việc đặt "sự chú ý" của thời điểm t độc lập hoàn toàn với việc đặt "sự chú ý" ở các thời điểm trước đó. Việc quyết định gióng hàng như thế nào (véc-tơ a_t) hoàn toàn phụ thuộc vào điểm số (giá trị của hàm score) giữa trạng thái ẩn h_t hiện tại và các trạng thái ẩn $\bar{h}_s$ ở câu nguồn. Trong thực tế, khi dịch, một từ ở câu nguồn chỉ tương ứng với một vài từ ở câu đích. Do vậy, mô hình cần phải theo dõi xem là những từ nào ở câu nguồn đã được dịch trước đó thì hạn chế đặt "sự chú ý" lên lại những từ đó. Việc không có cơ chế kiểm soát những từ nào đã được dịch sẽ khiến cho mô hình sẽ rơi vào 2 trường hợp "được dịch quá nhiều" (over-translated) hoặc "được dịch quá ít" (under-translated). Tức là có một số từ ở câu nguồn sẽ được đặt "sự chú ý" lên quá nhiều lần dẫn tới bỏ qua những từ quan trọng khác hoặc là một số từ quan trọng được đặt "sự chú ý" lên quá ít dẫn tới việc bỏ qua thông tin của từ đó trong quá trình dịch. Dù là trường hợp nào thì cũng gây giảm chất lượng dịch của mô hình và cho ra những câu dịch không thực tế.

Trong Dịch máy Thống kê, Koehn et al. 2003 [8] đã đề xuất một mô hình dịch dựa trên cụm từ (phrase-based) mà có cơ chế để giải quyết vấn đề trên. Cơ chế này rất đơn giản và trực quan. Trong quá trình dịch, bộ giải mã duy trì một véc-tơ bao phủ (coverage vector) để chỉ ra rằng từ ở câu nguồn nào đã được dịch hoặc chưa được dịch. Quá trình dịch được hoàn thành khi toàn bộ từ ở câu nguồn được "bao phủ" hay đã được dịch. Trong khi đó, các mô hình Dịch máy nơ-ron hiện nay chỉ kết thúc quá trình dịch khi và chỉ khi gặp kí tự kết thúc câu hoặc vượt quá số lượng từ cho trước. Việc này dễ dẫn đến trường hợp "được dịch quá nhiều" khi kí hiệu kết thúc câu xuất hiện trễ hay ngược lại dẫn đến trường hợp "được dịch quá ít" khi kí hiệu kết thúc câu



Hình 3.5: Minh họa phương pháp Input feeding. Tại thời điểm t, bộ giải mã nhận đầu vào gồm véc-tơ attention ở thời điểm trước đó $t-1$ và từ hiện tại x_t.

xuất hiện sớm. Ngoài ra còn bị ảnh hưởng bởi số lượng từ quy định khi dịch.

Công trình [10] đề xuất một cơ chế góp phần giải quyết vấn đề ở trên: (tạm dịch là "cho đầu vào ăn" // TODO: dịch khác). Ý tưởng và cách thực hiện của Input feeding rất đơn giản. Nhận thấy véc-tơ attention $\tilde{h}_{t-1}$ lưu giữ thông tin gióng hàng của thời điểm $t-1$ trước đó, mô hình thực hiện truyền véc-tơ $\tilde{h}_{t-1}$ vào đầu vào x_t của thời điểm t hiện tại. Bằng cách như vậy, mô hình có thể nắm được thông tin gióng hàng trước đó từ $\tilde{h}_{t-1}$. Cụ thể, véc-tơ $\tilde{h}_{t-1}$ được nối với véc-tơ đầu vào của thời điểm t là x_t:

$$x_t^{'} = [x_t, \tilde{h}_t] \tag{3.10}$$

Tuy nhiên, phương pháp này chưa thực sự giải quyết triệt để vấn đề "được dịch quá nhiều" hay "được dịch quá ít". Vì mô hình chỉ nhận được thông tin gióng hàng từ các thời điểm trước đó nhưng lại không được hướng dẫn hay có ràng buộc cụ thể nào mà để giải quyết vấn đề này. Việc giải quyết vấn đề trên hoàn toàn phụ thuộc vào quyết định của mô hình. Mặc dù chưa thực sự giải quyết triệt để, nhưng lại cho mô hình tăng thêm tính mềm dẻo trong việc sử dụng thông tin gióng hảng trước đó. Trong thực tế, phương pháp này đã cải thiện chất lượng dịch lên đáng kể và đơn giản về mặt cài đặt.

Ngoài ra, phương pháp này giúp cho mô hình phức tạp hơn nhờ vào việc đưa véc-tơ attention $\tilde{h}_t$ vào đầu vào của thời điểm tiếp theo, do đó làm tăng khả năng học của mô hình.

3.6 Kĩ thuật thay thế từ hiếm

Trong quá trình dịch thuật, có rất nhiều hạn chế gây ảnh hưởng tới chất lượng của bản dịch. Trong phần này, chúng tôi đề cập tới một vấn đề quan trọng mà dù là con người hay máy tính đều gặp phải và rất khó giải quyết. Đó là vấn đề về những "từ hiếm" (unknown words).

Mỗi ngôn ngữ có muôn hình vạn trạng các từ ngữ khác nhau. Số lượng từ ngữ trong một ngôn ngữ là không có định. Trong quá trình hình thành và phát triển ngôn ngữ, theo thời gian số lượng từ ngữ sẽ tăng lên hoặc mất đi (bị lãng quên hay không dùng nữa) tùy thuộc vào hoàn cảnh, môi trường sử dụng của ngôn ngữ đó. Nhưng thường đối với những ngôn ngữ phổ biển hiện nay thì số lượng từ ngữ tăng lên lớn hơn nhiều so với số lượng từ ngữ mất đi. Khi xã hội phát triển, nhu cầu giao tiếp giữa các dân tộc, quốc gia, nền văn hóa khác nhau cũng tăng theo. Mỗi nơi lại có cách sử dụng ngôn ngữ khác nhau, do đó bộ từ vựng của mỗi ngôn ngữ cũng phải thay đổi sao cho phù hợp với nhu cầu giao tiếp. Khoa học kĩ thuật phát triển kèm theo đó là những khám phá về thế giới tự nhiên. Những sự vật, hiện tượng mới được phát hiện ngày càng nhiều. Và không phải sự vật, hiện tượng nào cũng có thể được mô tả, thể hiện bằng những vốn từ vựng vốn có của một số ngôn ngữ. Ngoài ra còn có nhiều lí do làm cho bộ từ vựng của các ngôn ngữ thay đổi theo thời gian.

Với tốc độ phát triển của ngôn ngữ là như vậy nhưng khả năng của con người là hữu hạn. Một người dù có thông thạo một ngôn ngữ tới đâu thì cũng không thể nào biết được hết tất cả từ vựng của ngôn ngữ đó. Theo thống kê, số lượng từ ngữ cần để giao tiếp hàng ngày trong tiếng Anh chỉ khoảng từ 2000-3000 từ, đối với lĩnh vực chuyên ngành thì khoảng 5000-6000 từ. Nhưng theo kích thước của một số bộ từ điển thịnh hành trong tiếng Anh thì số lượng từ vựng của những bộ từ điển đó khoảng 60000 từ. Tức là đa số mọi người chưa biết hết được 10% từ vựng của tiếng Anh. Do vậy khi thực hiện việc dịch thuật giữa các ngôn ngữ với nhau, mọi người chỉ có thể dịch tốt khi văn bản, hội thoại cần dịch thuộc về chủ đề mà họ quen thuộc. Mọi người sẽ gặp khó khăn khi gặp những từ nằm ngoài bộ từ vựng của bản thân (out-of-vocabulary words - OOV words) vì không biết phải dịch như thế nào.

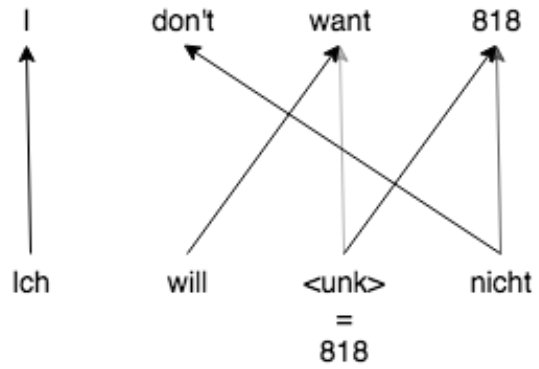
Khi huấn luyện một mô hình Dịch máy thì cần phải có một bộ từ vựng cố định cho mô hình đó trong suốt quá trình huấn luyện và dự đoán. Kích thước của bộ từ vựng

này bị hạn chế với số lượng nhất định. Sự hạn chế về kích thước này xuất phát từ nhiều lí do như giới hạn về dữ liệu huấn luyện, khả năng học của mô hình, tài nguyên tính toán (phần cứng), v.v... Do vậy việc quyết định xem những từ nào sẽ được đưa vào bộ từ vựng của mô hình cũng rất quan trọng. Thông thường có 2 chiến thuật để xây dựng bộ từ vựng này. Cách đầu tiên phù hợp cho việc phát triển các ứng dụng là lấy các từ vựng có trong dữ liệu huấn luyện làm bộ từ vựng và lọc ra những từ nào có tần số xuất hiện trong dữ liệu huấn luyện thấp hơn một ngưỡng nhất định (ví dụ: lọc ra những từ vựng nào có tần số xuất hiện ít hơn 10). Cách thứ 2 thường phù hợp cho việc nghiên cứu, đó là lựa chọn số lượng từ vựng nhất định mà có tần số xuất hiện cao nhất (ví dụ: lấy 50000 từ có tần số xuất hiện cao nhất). Do đó có những từ xuất hiện trong dữ liệu huấn luyện nhưng vì có tần số xuất hiện thấp nên bị coi là từ nằm ngoài bộ từ vựng (OOV). Đó là lí do chúng tôi gọi đây là vấn đề "từ hiếm".

Có nhiều cách để giải quyết vấn đề này, cách mà mọi người hay sử dụng nhất là thêm từ mới đó vào bộ từ vựng. Cách thứ 2 là giữ nguyên từ đó và đưa nó vào vị trí thích hợp trong câu ở ngôn ngữ đích. Trong khóa luận này chúng tôi sẽ sử dụng cách thứ 2 để giải quyết vấn đề các từ nằm ngoái bộ từ vựng.

Kĩ thuật thay thế từ hiếm mà chúng tôi trình bày sau đây là một phương pháp dựa trên kết quả của cơ chế Attention. Do vậy, hiệu quả của phương pháp này phụ thuộc lớn vào độ chính xác của cơ chế Attention. Kĩ thuật này chúng tôi sử dụng từ bài báo của Jean et al., 2015 [6] về sử dụng cơ chế Attention trong mô hình Dịch máy nơ-ron. Nếu mô hình không sử dụng cơ chế Attention thì cũng không sử dụng được phương pháp thay thế từ hiếm này. Kĩ thuật này chỉ được sử dụng trong quá trình dự đoán, trong quá trình huấn luyện thì không sử dụng. Cách hoạt động của phương pháp này rất đơn giản. Sau khi mô hình đã dự đoán (dịch) xong một câu, mô hình sẽ thực hiện xử lý những từ nào mà được dự đoán là từ hiếm (unknown words) trong câu đã được dự đoán (những từ hiếm được ký hiệu là <*unk*>). Đối với mỗi từ hiếm, mô hình sẽ thực hiện dịch lại từ đó bằng cách chọn một từ phù hợp trong câu nguồn rồi thực hiện sao chép từ được chọn vào từ hiếm hiện tại. Cách mà mô hình lựa chọn từ phù hợp là dựa vào véc-tơ trọng số gióng hàng a_t. Mô hình sẽ lựa chọn từ nào có trọng số cao nhất.

Trong thực tế, kĩ thuật này giúp cho mô hình có thể dịch được chính xác những câu có chữ số, số, tên riêng, tên địa danh v.v... Bởi vì những từ này rất ít xuất hiện



Hình 3.6: Minh họa phương pháp thay thế từ hiếm. Khi gặp một từ hiếm (được kí hiệu là <unk>) trong kết quả dự đoán, mô hình sẽ tìm một từ ở câu nguồn có trọng số gióng hàng từ kết quả cơ chế Attention cao nhất và thực hiện sao chép từ đó thay cho từ hiếm hiện tại. (Mũi tên càng đậm thì trọng số gióng hàng càng cao). Kết quả dự đoán được cập nhật với từ hiếm đã được thay thế.

trong tập dữ liệu so với những từ khác. Hơn nữa, những loại từ như thế này rất đa dạng (số lượng số có thể có rất lớn hay tên riêng, tên địa danh có rất nhiều). Điều đặc biệt rằng những từ này thường không cần phải dịch, chúng ta chỉ cần sao chép chính xác chúng lại qua câu ở ngôn ngữ đích với một vị trí phù hợp do những từ này dù ở ngôn ngữ nào thì cũng đều có một cách biểu diễn

Với kĩ thuật đơn giản là tận dụng ý nghĩa của kết quả của cơ chế Attention, kĩ thuật này đã cải thiện kết quả dịch lên một cách rõ rệt (sẽ được trình bày ở trong phần thực nghiệm).

Chương 4

Các Kết Quả Thực Nghiệm

Trong chương này, chúng tôi trình bày các kết quả thí nghiệm để đánh giá các mô hình được tìm hiểu mà đã trình bày ở chương trước. Bộ dữ liệu được dùng để tiến hành các thí nghiệm là bộ WMT'14 English-German (bộ dữ liệu tiếng Anh-tiếng Đức của cuộc thi Dịch máy WMT năm 2014). Các kết quả thí nghiệm cho thấy khi huấn luyện mô hình mà không sử dụng cơ chế Attention thì kết quả đạt được rất thấp. Các kết quả cũng cho thấy rằng các mô hình Attention Toàn cục, Attention Cục bộ, phương pháp Input feeding cho kết quả được cải thiện một cách rõ rệt.

4.1 Các thiết lập thực nghiệm

Chúng tôi tiến hành các thực nghiệm trên bộ dữ liệu WMT' 14 English-German được cung cấp trên trang chủ của Nhóm Xử lý Ngôn ngữ Tự nhiên Đại học Stanford [4]. Bộ dữ liệu này gồm các cặp câu được viết dưới dạng ngôn ngữ tự nhiên ở 2 ngôn ngữ là tiếng Anh và tiếng Đức. Tất cả mô hình sẽ được huấn luyện trên tập dữ liệu này. Tập dữ liệu có khoảng 4,5 triệu cặp câu (trong đó có khoảng 116 triệu từ tiếng Anh và khoảng 110 triệu từ tiếng Đức). Chúng tôi thực hiện thiết lập thực nghiệm giống với các thiết lập của bài báo của Luong et al., 2015 [10].

Dữ liệu được tiến hành tiền xử lý bằng cách thực hiện tách từ đối với mỗi câu. Bộ từ vựng cho mỗi ngôn ngữ được sử dụng cho các mô hình là bộ từ vựng có 50.000 từ xuất hiện nhiều nhất (có tần số lớn nhất) trong dữ liệu huấn luyện của mỗi ngôn ngữ đó. Những từ nào không nằm trong bộ từ vựng sẽ được gán cho kí hiệu $< unk >$.

Trong quá trình huấn luyện, chúng tôi lọc bỏ những cặp câu mà một trong 2 câu thuộc cặp đó có chiều dài hơn 50 từ. Chúng tôi thực hiện sắp xếp tất cả câu theo chiều dài của câu giảm dần (những câu nào có chiều dài lớn nhất thì đứng đầu), sau đó lấy ngẫu nhiên các mini-batches từ những câu đã được sắp xếp. Với việc sắp xếp như vậy, tốc độ huấn luyện của mô hình được cải thiện và mô hình học được tốt hơn.

Chúng tôi sử dụng các mô hình LSTM với mỗi LSTM có 4 tầng. Mỗi tầng LSTM có kích thước trạng thái ẩn là 1000 (sử dụng Bi-LSTM nên mỗi chiều sẽ có kích thước trạng thái ẩn là 500) và số chiều của word embedding là 1000. Các tham số của mô hình được khởi tạo ngẫu nhiên với phân phối đều trong đoạn $[-0,1;0,1]$. Thuật toán để cực tiểu hóa hàm chi phí là Stochastic Gradient Descent (SGD) với kích thước của mini-batch là 128 mẫu huấn luyện. Cách lập lịch cho hệ số học: huấn luyện 12 epochs; hệ số học ban đầu là 1,0; sau 8 epochs, hệ số học sẽ giảm đi 1 nửa sau mỗi epoch tiếp theo. Gradient của các tham số sẽ được chuẩn hóa nếu norm của chúng vượt quá 5,0. Mô hình còn sử dụng cơ chế Dropout với xác suất tắt các nơ-ron $p = 0.2$. Mỗi câu ở ngôn ngữ nguồn khi được đưa vào mô hình thì sẽ được đảo ngược trật tự. Đối với các mô hình Attention Cục bộ, kích thước cửa sổ $D = 10$.

Chúng tôi sử dụng ngôn ngữ lập trình Python và framework PyTorch dành cho Học sâu [13]. PyTorch hỗ trợ việc cài đặt các thuật toán một cách thân thiện, tự nhiên giống như Python và còn hỗ trợ xử lí tính toán song song trên GPU (Graphical Processing Units) rất mạnh mẽ. GPU mà chúng tôi sử dụng để thực hiện các thực nghiệm là NVIDIA GeForce GTX 1080 Ti. Để có thể huấn luyện một mô hình, cần đến 3-5 ngày.

Để đánh giá chất lượng dịch của các mô hình đã được huấn luyện, chúng tôi sử dụng tập dữ liệu kiểm thử *newstest_2014.en* và *newstest_2014.de* của cuộc thi WMT'14 và độ đo được sử dụng để đánh giá là BLEU (BiLingual Evaluation Understudy) [11] cùng với Perplexity. Dữ liệu validation được sử dụng là tập dữ liệu kiểm thử *newstest_2013.en* và *newstest_2013.de* của cuộc thi WMT'13.

Để đánh giá độ hiệu quả của cơ chế Attention , chúng tôi tiến huấn luyện một mô hình cơ bản (Baseline) mà không sử dụng cơ chế Attention (chỉ dùng kiến trúc Bộ mã hóa-Bộ giải mã với các LSTM). Các mô hình có sử dụng cơ chế Attention sẽ được so sánh với mô hình cơ bản này.

Mô hình	Perplexity	BLEU
Cơ bản		15.04
Cơ bản + global (dot)		19.02 (+3.98)

Bảng 4.1: So sánh giữa mô hình sử dụng cơ chế Attention và mô hình không sử dụng cơ chế Attention.

4.2 Kết quả thực nghiệm

Các kết quả của các mô hình được ghi trong bảng 4.3:

4.2.1 Không sử dụng Attention và có sử dụng Attention

Chúng tôi thực hiện đánh giá độ hiệu quả của cơ chế Attention bằng cách so sánh với mô hình không sử dụng cơ chế Attention và các mô hình có sử dụng cơ chế Attention.

Đối với mô hình không sử dụng cơ chế Attention, chúng tôi sử dụng:

- Kiến trúc Bộ mã hóa-Bộ giải mã với bộ mã hóa là bi-LSTM và bộ giải mã là uni-LSTM.
- Đảo ngược trật tự từ trong câu.
- Dropout.

Chúng tôi gọi đó là mô hình cơ bản. Đối với mô hình sử dụng cơ chế Attention, chúng tôi thiết lập mô hình như mô hình căn bản cộng với sử dụng cơ chế Attention Toàn cục với hàm tính điểm là hàm *dot*.

Bảng 4.1 cho thấy kết quả giữa 2 mô hình. Với cơ chế Attention, chất lượng dịch của mô hình được cải thiện rất lớn. Điểm BLEU tăng từ 15.04 đến 19.02 (+3.98). Đây là bước tiến lớn trong Dịch máy nơ-ron khi Dịch máy Thống kê đang dần chạm tới giới hạn. Kết quả này chứng minh được rằng cơ chế Attention rất hiệu quả trong việc giải quyết các hạn chế của kiến trúc Bộ mã hóa-Bộ giải mã với LSTM ban đầu và cũng rất phù hợp với bài toán Dịch máy.

Mô hình	Perplexity	BLEU	
		Trước khi	Sau khi dùng thay thế từ hiếm
Global (dot)		19.02	
Global (dot) + input feed		19.78	22.35 (+2.57)
Local-p + input feed		20.37	22.75 (+2.38)

Bảng 4.2: So sánh giữa các mô hình Attention.

4.2.2 Giữa các mô hình Attention với nhau

Để đánh giá độ hiệu quả của các mô hình Attention với nhau, chúng tôi thực hiện huấn luyện các mô hình Attention và đánh giá chúng trên độ đo Perplexity và BLEU.

Từ bảng kết quả cho thấy mô hình Attention cục bộ với Gióng hàng dự đoán (Local-p) có chất lượng dịch tốt nhất với điểm BLEU là 20.37 (22.75 khi dùng thay thế từ hiếm). Mô hình Attention Toàn cục với hàm tính điểm dot cộng với phương pháp Input feeding (Global (dot) + input feed) có độ tăng điểm BLEU cao nhất khi dùng kĩ thuật thay thế từ hiếm. Kết quả này chứng minh được rằng mô hình Local-p có ý tưởng phù hợp cho cặp ngôn ngữ Anh-Đức và hoạt động hiệu quả như mong đợi trong thực tế.

Kết quả của các mô hình Attention được thể hiện trong bảng 4.2. Kết quả cho thấy rằng với 2 phương pháp Input feeding và thay thế từ hiếm mà chúng tôi đã trình bày ở phần trước đều góp phần tăng hiệu quả của mô hình lên rất đáng kể. Cụ thể, phương pháp Input feeding tăng BLEU lên trung bình khoảng 1.0. Kĩ thuật thay thế từ hiếm tăng trung bình hơn 2.25 BLEU. Các kết quả này chứng minh được rằng những ý tưởng, lý thuyết chúng tôi trình bày ở phần trước đều hoạt động tốt như mong đợi. Những phương pháp, kĩ thuật này vừa rõ ràng về mặt lý thuyết, vừa hiệu quả trong thực tế.

Về phương pháp Input feeding, có một chút hạn chế của phương pháp này về chi phí tính toán. Input feeding làm kích thước đầu vào ở các thời điểm của uni-LSTM của bộ giải mã tăng lên theo kích thước của véc-tơ attention. Hơn nữa, về mặt cài đặt, chúng ta không tận dụng được LSTM đã được hỗ trợ cài đặt bởi NVIDIA. Do vậy, tốc độ huấn luyện và trong kiểm thử của mô hình khi có sử dụng cơ chế Input feeding sẽ giảm đi đáng kể, bên cạnh đó kích thước của mô hình cũng tăng lên theo.

Về kĩ thuật thay thế từ hiếm, từ kết quả cho thấy kĩ thuật này rất mạnh mẽ. Các mô

Model	BLEU	
	Ours	Paper
Baseline	15.04	14.0
Baseline + global (general)	20.25	17.3
Baseline + global (dot)	19.02	18.6
Baseline + global (dot) + input feed	20.23	
Baseline + global (dot) + input feed + unk rpl	22.71	
Baseline + local-p (genenral) + input feed	20.75	

Bảng 4.3: Kết quả của các mô hình trên tập dữ liệu WMT'14 English-German.

hình Attention được sử dụng cùng với kĩ thuật này đều được tăng điểm BLEU hơn 2. Mô hình này chỉ có hạn chế nhỏ là phải phụ thuộc vào kết quả của cơ chế Attention có tốt hay không. Còn lại kĩ thuật này rất hiệu quả và tiện lợi. Thay thế từ hiếm không ảnh hưởng tới quá trình huấn luyện, do vậy mô hình không phải tốn chi phí tính toán cho kĩ thuật này trong quá trình huấn luyện và tốc độ huấn luyện không bị ảnh hưởng.

Nhìn chung, kết quả cho thấy kĩ thuật thay thế từ hiếm tốt hơn phương pháp Input feeding, nhưng Input feeding giúp tạo ra một mô hình Attention tốt hơn để tạo điều kiện cho kĩ thuật thay thế từ hiếm phát huy sự hiệu quả.

Từ bảng kết quả cho thấy việc sử dụng cơ chế Attention giúp cải thiện kết quả rất lớn. Mô hình Baseline có kết quả trên độ đo BLEU của chúng tôi là 15,04. Khi sử dụng cơ chế Attention, mô hình cho khoảng chênh lệch nhỏ nhất giữa mô hình Baseline và các mô hình Attention là mô hình Attention Toàn cục với hàm score là dot với BLEU bằng 19,02 (chênh lệch 3,98 BLEU). Khi sử dụng phương pháp Input feeding, kết quả tăng 1,21 BLEU thành 20.23 BLEU. Kết quả còn được cải thiện hơn nữa với việc sử dụng kĩ thuật thay thế từ hiếm (unk rpl), chúng tôi đạt được điểm BLEU là 22,71 (tăng 2.43 BLEU). Kết quả của chúng tôi có một chút chênh lệch so với bài báo của Luong et al. [10] do yếu tố ngẫu nhiên của việc huấn luyện mô hình.

Các kết quả trên cho thấy những cơ chế mà chúng tôi tìm hiểu và sử dụng trong khóa luận này thực sự hiệu quả. Các cơ chế này vừa rõ ràng về lý thuyết vừa có hiệu năng tốt trong thực tế. Đặc biệt với cơ chế thay thế từ hiếm dựa vào kết quả của cơ chế Attention đã tăng kết quả của mô hình lên rất đáng kể (tăng 2,43 BLEU). Điều này cũng cho thấy tiềm năng của cơ chế Attention trong việc giải quyết bài toán Dịch máy. Không chỉ bản thân của cơ chế này nâng cao chất lượng dịch mà còn là nền tảng để những cơ chế khác sử dụng và tiếp tục nâng cao chất lượng dịch.

Chương 5

Kết Luận Và Hướng Phát Triển

5.1 Kết luận

Trong khóa luận này, chúng tôi nghiên cứu bài toán Dịch máy nơ-ron bằng mô hình Attention-LSTM. Mô hình Attention-LSTM học được cách dịch giữa 2 ngôn ngữ (trong khóa luận này là Anh-Đức) như các mô hình với kiến trúc Bộ mã hóa-Bộ giải mã với bộ mã hóa và bộ giải mã là các LSTM. Tuy nhiên cơ chế Attention đem lại nhiều lợi thế mà những mô hình không sử dụng Attention không có được:

- Các mô hình sử dụng Attention tận dụng được các trạng thái ẩn trên bộ mã hóa để hạn chế vấn đề "sự phụ thuộc dài" của các mô hình RNNs. Trong quá trình dự đoán, bộ giải mã sử dụng các trạng thái ẩn của bộ mã hóa bằng cách đặt "sự chú ý" lên một số trạng thái ẩn cần thiết và sử dụng nó để suy ra ngữ cảnh hiện tại của câu, sau đó mô hình dự đoán từ tiếp theo dựa vào ngữ cảnh đó.

- Cơ chế có cách dịch giống với ý tưởng về cách con người dịch nhìn sự vật, hiện tượng. Con người thường chỉ tập trung vào những phần quan trọng mà cung cấp những thông tin cần thiết, phù hợp với mục đích quan sát của sự vật, hiện tượng.

- Có thể sử dụng kết quả của cơ chế Attention để phát triển thêm các phương pháp, kĩ thuật khác:

  - Phương pháp Input feeding: giúp mô hình có thể biết được thông tin gióng hàng trong những thời điểm trước đó thông qua véc-tơ attention $\tilde{h}_t$. Từ đó giúp mô hình có thể hạn chế vấn đề "đươc dịch quá nhiều" hoặc "được

dịch quá ít", tránh được những câu dịch không thực tế như những câu dịch lặp lại một từ nhiều lần.

- Kĩ thuật thay thế từ hiếm: giúp mô hình giải quyết được vấn đề hạn chế về kích thước của bộ từ vựng. Do bộ từ vựng không thể chứa hết tất cả các từ có thể có trong quá trình dịch, vì vậy chất lượng dịch của mô hình bị giảm đáng kể nếu gặp những từ hiếm đó. Đặc biệt là khi mô hình gặp những câu có chứa các số, tên riêng, tên các địa danh, v.v...

Các kết quả thực nghiệm trên bộ dữ liệu WMT'14 English-German cho thấy rằng:

- Cơ chế Attention cải thiện chất lượng dịch của mô hình rất cao so với những mô hình không sử dụng cơ chế Attention.
- Những phương pháp, kĩ thuật sử dụng kết quả của cơ chế Attention để giải quyết những vấn đề còn tồn tại khi sử dụng mô hình dịch máy nơ-ron cũng cải thiện chất lượng dịch của mô hình lên đáng kể.

Cơ chế Attention đã mở ra một không gian rộng lớn để phát triển cho việc cải tiến mô hình các dịch máy nơ-ron.

// TODO Chúng tôi đạt được trong khóa luận này:

-

Chương 6

Kết Luận và Hướng Phát Triển

6.1 Kết luận

Trong luận văn này, chúng tôi nghiên cứu về bài toán học đặc trưng không giám sát bằng "Sparse Auto-Encoders" (SAEs). SAEs có thể học được những đặc trưng tương tự như "Sparse Coding", nhưng điểm lợi là quá trình huấn luyện SAEs có thể được thực hiện một cách hiệu quả thông qua thuật toán lan truyền ngược, và với một véc-tơ đầu vào mới, SAEs có thể tính được véc-tơ đặc trưng tương ứng rất nhanh. Tuy nhiên, trong thực tế, không dễ để có thể làm SAEs "hoạt động"; có hai điểm ta cần phải làm rõ: (i) ràng buộc thưa, và (ii) ràng buộc trọng số. Đóng góp của luận văn là làm rõ SAEs ở hai điểm này. Cụ thể như sau:

- Về ràng buộc thưa, mặc dù chuẩn L1 là cách tự nhiên (vì L1 được dùng trong Sparse Coding) và đơn giản để ràng buộc tính thưa của véc-tơ đặc trưng, nhưng L1 lại thường không được dùng trong SAEs với lý do vẫn còn chưa rõ ràng. Thay vì dùng L1, các bài báo về SAEs thường ràng buộc thưa bằng cách ép giá trị đầu ra trung bình của mỗi nơ-ron ẩn về một giá trị cố định gần 0. Nhưng giá trị cố định này lại thêm một siêu tham số vào danh sách các siêu tham số vốn đã có rất nhiều của SAEs; điều này sẽ làm cho quá trình chọn lựa các siêu tham số trở nên "phiền phức" hơn và tốn thời gian hơn. *Trong luận văn, chúng tôi cố gắng hiểu khó khăn gặp phải khi huấn luyện SAEs với chuẩn L1; từ đó, đề xuất một phiên bản hiệu chỉnh của thuật toán "Stochastic Gradient Descent" (SGD), gọi là "Sleep-Wake Stochastic Gradient Descent" (SW-SGD), để khắc phục khó khăn gặp phải này. Ở đây, chúng tôi tập trung nghiên cứu SAEs với*

hàm kích hoạt "rectified linear" ở tầng ẩn vì hàm này tính nhanh và có thể cho tính thưa thật sự (đúng bằng 0); chúng tôi gọi SAEs với hàm kích hoạt này là "Sparse Rectified Auto-Encoders" (SRAEs).

- Về ràng buộc trọng số, có một số cách đã được đề xuất để ràng buộc trọng số của SAEs, nhưng không rõ là tại sao ta lại nên ràng buộc trọng số như vậy. Liệu có cách ràng buộc trọng số nào tốt hơn? *Trong luận văn, chúng tôi đề xuất một cách ràng buộc trọng số mới và hợp lý cho SRAEs.*

Các kết quả thí nghiệm trên bộ dữ liệu MNIST (bộ ảnh chữ số viết tay từ 0 đến 9) cho thấy:

- Khi huấn luyện SRAEs với chuẩn L1 sẽ gặp phải vấn đề nơ-ron "ngủ" và chiến lược "ngủ - đánh thức" đề xuất của chúng tôi trong thuật toán SW-SGD có thể giúp khắc phục vấn đề này.

- Cách ràng buộc trọng số đề xuất của chúng tôi giúp SRAEs học được những đặc trưng cho kết quả phân lớp tốt nhất so với các cách ràng buộc trọng số khác mà có thể áp dụng cho SRAEs.

- SRAEs với SW-SGD và cách ràng buộc trọng số của chúng tôi có thể học được những đặc trưng cho kết quả phân lớp tốt so với các loại "Auto-Encoders" khác.

6.2 Hướng phát triển

Thật ra, luận văn mới chỉ giải quyết được một phần nhỏ và mang tính kỹ thuật (làm cho SAEs hoạt động) của bài toán học đặc trưng không giám sát. Câu hỏi lớn và mang tính định hướng dài hạn là: *Thế nào là một biểu diễn đặc trưng tốt?* Theo GS. Yoshua Bengio, một trong những nhà nghiên cứu tiên phong trong lĩnh vực học biểu diễn đặc trưng, thì: *Một biểu diễn đặc trưng tốt cần* ***phân tách (disentangle)*** *được các yếu tố giải thích ẩn bên dưới*. Để phân tách được các yếu tố giải thích ẩn, ta cần có sự hiểu biết trước (prior) về các yếu tố ẩn. Ở đây, ta quan tâm đến các sự hiểu biết trước mang tính tổng quát, có thể áp dụng để học đặc trưng trong nhiều bài toán liên quan đến trí tuệ nhân tạo (thị giác máy tính, xử lý ngôn ngữ tự nhiên, ...). Định hướng phát triển

của luận văn là tích hợp thêm các hiểu biết trước khác vào SAEs nhằm phân tách tốt hơn các yếu tố giải thích ẩn. Dưới đây là một số hiểu biết trước mà có thể tích hợp vào SAEs:

- **Học sâu**: thế giới xung quanh ta có thể được mô tả bằng một kiến trúc phân cấp; cụ thể là, các yếu tố hay các khái niệm (concept) trừu tượng (ví dụ như con mèo, cái cây, ...) bao gồm các khái niệm ít trừu tượng hơn; các khái niệm ít trừu tượng hơn này lại bao gồm các khái niệm ít trừu tượng hơn nữa ... Do đó, ta muốn học nhiều tầng biểu diễn đặc trưng với độ trừu tượng tăng dần. Mặc dù, SRAEs có thể được dùng để học từng tầng đặc trưng một, nhưng mục tiêu mà chúng tôi hướng đến là: học *đồng thời* nhiều tầng biểu diễn đặc trưng một cách không giám sát.

- **Gom cụm tự nhiên**: các mẫu thuộc các lớp khác nhau nằm trên các đa tạp (manifold) khác nhau và các đa tạp này được phân tách tốt với nhau bởi các vùng có mật độ thấp; hơn nữa, số chiều của các đa tạp này nhỏ hơn rất nhiều so với số chiều của không gian ban đầu. Ta thấy rằng sự gom cụm tự nhiên này sẽ dẫn đến tính thưa. Cụ thể là, các đa tạp khác nhau (ứng với các lớp khác nhau) sẽ được mô tả bởi các hệ trục tọa độ khác nhau. Với một véc-tơ đầu vào x thì chỉ có hệ trục tọa độ của đa tạp ứng với lớp mà x thuộc về được kích hoạt. Nếu ta hiểu véc-tơ đặc trưng h của x chứa các hệ số của các hệ trục tọa độ này thì h sẽ thưa bởi vì chỉ có các hệ số của hệ trục tọa độ được kích hoạt là có giá trị khác 0. Do đó, thay vì ràng buộc tính thưa một cách đơn thuần bằng chuẩn L1, ta có thể tìm cách để ràng buộc tính thưa từ góc nhìn gom cụm tự nhiên nói trên.

Phụ Lục: Các Công Trình Đã Công Bố

Hội nghị quốc tế:

- **K. Tran** and B. Le, "Demystifying Sparse Rectified Auto-Encoders," in *Proceedings of the Fourth Symposium on Information and Communication Technology*, ser. SoICT'13. New York, NY, USA: ACM, 2013, pp. 101–107. [Online]. Available: http://doi.acm.org/10.1145/2542050.2542065

PROCEEDINGS OF

THE FOURTH SYMPOSIUM ON INFORMATION AND COMMUNICATION TECHNOLOGY

SoICT 2013

Da Nang, Vietnam
December 5-6, 2013

ISBN: 978-1-4503-2454-0

Symposium on Information and Communication Technology 2013

SoICT 2013

Table of Contents

Demystifying Sparse Rectified Auto-Encoders

Kien Tran
Department of Computer Science
Faculty of Information Technology
Vietnam University of Science - HCM
ttkien@fit.hcmus.edu.vn

Bac Le
Department of Computer Science
Faculty of Information Technology
Vietnam University of Science - HCM
lhbac@fit.hcmus.edu.vn

ABSTRACT

Sparse Auto-Encoders can learn features similar to Sparse Coding, but the training can be done efficiently via the back-propagation algorithm as well as the features can be computed quickly for a new input. However, in practice, it is not easy to get Sparse Auto-Encoders working; there are two things that need investigating: sparsity constraint and weight constraint. In this paper, we try to understand the problem of training Sparse Auto-Encoders with L1-norm sparsity penalty, and propose a modified version of Stochastic Gradient Descent algorithm, called Sleep-Wake Stochastic Gradient Descent (SW-SGD), to solve this problem. Here, we focus on Sparse Auto-Encoders with rectified linear units in the hidden layer, called Sparse Rectified Auto-Encoders (SRAEs), because such units compute fast and can produce true sparsity (exact zeros). In addition, we propose a new reasonable way to constrain SRAEs' weights. Experiments on MNIST dataset show that the proposed weight constraint and SW-SGD help SRAEs successfully learn meaningful features that give excellent performance on classification task compared to other Auto-Encoder variants.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning—*connectionism and neural nets, concept learning, parameter learning*; I.2.10 [**Artificial Intelligence**]: Vision and Scene Understanding—*representation, data structures, and transforms*; I.4.7 [**Image Processing and Computer Vision**]: Feature Measurement—*feature representation*

General Terms

Algorithms, Design, Experimentation

Keywords

unsupervised feature learning, deep learning, sparse coding, sparse auto-encoders, rectified linear units

SoICT'13, December 05 - 06 2013, Danang, Viet Nam

http://dx.doi.org/10.1145/2542050.2542065.

1. INTRODUCTION

Recently, unsupervised feature learning and deep learning have attracted a lot of interest from various fields such as computer vision, audio processing, text processing, and so on. The idea is that instead of designing features manually, one lets the learning algorithms automatically learn features from unlabeled data; and deep learning means learning multiple levels of features with increasing abstraction. Auto-Encoders (AEs) and Restricted Boltzmann Machines (RBMs) are two main groups of algorithms that have been used in unsupervised feature learning and deep learning [1]. AEs belong to the non-probabilistic group while RBMs belong to the probabilistic group. One big disadvantage of RBMs compared to AEs is that the objective function of RBMs is intractable. For this reason, here we will focus on the study of AEs.

Several criteria have been proposed to guide AEs to learn useful representation. They include: sparsity criterion [6], denoising criterion [14], and contraction criterion [13, 12]. Among them, sparsity is an interesting and promising one (here sparsity means forcing the majority of elements of the feature vector to be zeros). The first reason is that it has the inspiration from biology. In the brain, there is a very small fraction of neurons active simultaneously. Sparsity was first introduced in Sparse Coding and interestingly, it helped learn features similar to the primary visual cortex [11]. AEs with sparsity criterion, called Sparse Auto-Encoders (SAEs), can learn features much like Sparse Coding, but unlike Sparse Coding, the training can be done efficiently via the back-propagation algorithm, and with a new input, the features can be computed quickly. Secondly, sparsity can help learn high-level features - concepts. The intuitive justification is that there are only a few concepts per example; therefore, sparsity can help learn a dictionary of concepts and each example will be explained just by a small number of concepts. Thirdly, sparsity can potentially help speed up the training of SAEs. With each example, in the forward propagation phase, there is only a small fraction of neurons active; and hence, in the backward propagation phase, there is only a small fraction of parameters (corresponding to active neurons) updated. This point can be made use of to speed up the training process. It is important because if the training is fast, the model can be scaled up (i.e. increase the number of features); in unsupervised feature learning and deep learning, large-scale is a key factor to get good performance [4, 7].

Despite above advantages, it is not easy to get SAEs working in practice. To make SAEs work, there are two things

that need investigating: sparsity constraint and weight constraint. Although L1-norm is a natural (because it is used in Sparse Coding) and simple (in case the feature vector has positive values, it is just simply sum of them) way to constrain sparsity, it is not often used in SAEs for reasons that remain to be understood [1]. Instead of L1-norm, people often constrain sparsity in SAEs by pushing the average output of a hidden neuron (e.g. over a minibatch) to a fixed target (close to zero) [6, 4, 3]. But this fixed target adds one more hyper-parameter to the list of SAEs' hyper-parameters which already has many ones. As a result, the process of tuning hyper-parameters will become more tedious and more time-consuming. Regarding weight constraint, many different ways were used in the literature. [3, 14, 13, 12] tied the weights of encoder and decoder together. [6, 4] used weight decay; this way even adds one more hyper-parameter. [15] constrained the weights of decoder to have unit norm. However, it is not clear which way should be used as well as why weights should be constrained like those.

Two questions remain to be answered: (i) why is L1-norm sparsity penalty not often used in SAEs?; (ii) is there a better and more reasonable way to constrain SAEs' weights? In this paper, we try to understand the problem of training SAEs with L1-norm sparsity penalty. Then, we propose a modified version of Stochastic Gradient Descent algorithm (SGD), called Sleep-Wake Stochastic Gradient Descent (SW-SGD), to remedy this problem. Here we focus on SAEs with rectified linear units (ReLUs) in the hidden layer because such units compute fast and can produce true sparsity (exact zeros) [10, 5, 15]. We call these Sparse Rectified Auto-Encoders (SRAEs). Furthermore, we propose a new reasonable way to constrain SRAEs' weights. With these two ingredients, our proposed weight constraint and SW-SGD, our experiments show that SRAEs can successfully learn meaningful features that give excellent classification performance on MNIST dataset compared to other Auto-Encoder variants.

The rest of the paper is organized as follows. We start by reviewing Sparse Coding and Sparse Auto-Encoders (SAEs) to see advantages of SAEs compared to Sparse Coding. Then, Section 3 presents Sparse Rectified Auto-Encoders (SRAEs): Subsection 3.1 explains the problem of training SRAEs with L1-norm sparsity penalty and describes our remedy for this problem; Subsection 3.2 presents our proposed weight constraint for SRAEs. Experiment and analysis are shown in Section 4 followed by the conclusion in Section 5.

2. REVIEW OF SPARSE CODING AND SPARSE AUTO-ENCODERS

2.1 Sparse Coding

Sparse Coding was first introduced in neuroscience to model the primary visual cortex [11]. The goal is to find an over-complete set of basic vectors so that each input can be explained just by a small number of basis vectors (i.e. the feature vector is sparse). Specifically, given the unlabeled data $\{x^{(1)}, \dots, x^{(N)}\}$ with $x^{(n)} \in \mathbb{R}^D$, Sparse Coding solves the following optimization problem:

$$\begin{aligned} \underset{\phi,a}{\text{minimize}} \quad & \sum_{n=1}^{N} \left(\|x^{(n)} - \sum_{k=1}^{K} a_k^{(n)} \phi^{(k)}\|_2^2 + \lambda \|a^{(n)}\|_1 \right) \\ \text{subject to} \quad & \|\phi^{(k)}\|_2^2 = 1, \forall k = 1, \dots, K \end{aligned} \quad (1)$$

Here, the optimization variables are *the basis vectors* $\phi = \{\phi^{(1)}, \dots, \phi^{(K)}\}$ with each $\phi^{(k)} \in \mathbb{R}^D$, and *the coefficient vectors* (the feature vectors) $a = \{a^{(1)}, \dots, a^{(N)}\}$ with each $a^{(n)} \in \mathbb{R}^K$; $a_k^{(n)}$ is the coefficient of basic $\phi^{(k)}$ for input $x^{(n)}$. With this optimization problem, we want to learn a representation having the following properties:

- Preserving information about the input (by minimizing the reconstruction error).
- Being sparse (by minimizing the L1-norm of the feature vector).

λ is the hyper-parameter controlling the trade-off between reconstruction error and sparsity penalty.

The problem (1) can be solved by iteratively optimizing over a and ϕ alternately while holding the other set of variables fixed [9]. However, this process often takes a long time to converge. Furthermore, after training, to find the feature vector for a new input, we still have to do optimization (with fixed ϕ).

2.2 Sparse Auto-Encoders

An Auto-Encoder (AE) is a feed-forward neural network with two layers. The first layer, called *encoder*, maps the input x to the hidden representation a: $a = f\left(W^{(e)}x + b^{(e)}\right)$ where $f(\cdot)$ is some activation function (e.g. sigmoid), $W^{(e)}$ and $b^{(e)}$ are parameters of the encoder. The second layer, called *decoder*, then tries to reconstruct the input from the hidden representation a: $\hat{x} = W^{(d)}a + b^{(d)}$ where $\hat{x}$ is the reconstructed input, $W^{(d)}$ and $b^{(d)}$ are parameters of the decoder. In this way, we hope that the hidden representation can capture the structure of the input.

In Sparse Auto-Encoders (SAEs), besides reconstruction error, we also constrain the representation to be sparse (i.e. with a input, there are only a few hidden neurons active). Specifically, given the unlabeled data $\{x^{(1)}, \dots, x^{(N)}\}$ with $x^{(n)} \in \mathbb{R}^D$, SAEs minimize the following objective function:

$$J(W^{(e)}, b^{(e)}, W^{(d)}, b^{(d)}) = \sum_{n=1}^{N} \|x^{(n)} - \hat{x}^{(n)}\|_2^2 + \lambda s(a^{(n)}) \quad (2)$$

where: $a^{(n)} = f\left(W^{(e)}x^{(n)} + b^{(e)}\right)$; $\hat{x}^{(n)} = W^{(d)}a^{(n)} + b^{(d)}$; $s(\cdot)$ is some function that encourages the feature vector $a^{(n)}$ to be sparse; and λ is the hyper-parameter controlling the trade-off between reconstruction error and sparsity penalty.

Similar to Sparse Coding, SAEs aim at learning a representation that both preserves information about the input and is sparse. The difference between them is that SAEs have an explicit parametric encoder, while Sparse Coding has an implicit non-parametric encoder. This point helps training SAEs be more efficient than Sparse Coding; it can be done via the back-propagation algorithm. In addition, with a new input, SAEs can compute the corresponding feature vector very quickly just by one step.

3. SPARSE RECTIFIED AUTO-ENCODERS

The typical activation functions have been used in neural networks are the sigmoid function and the tanh function. Recently, a new activation function which have been found to work very well is the rectified linear function [10, 5, 15]:

$f(x) = \max(0, x)$. Units with such activation function are called rectified linear units (ReLUs).

ReLUs fit well with SAEs because such units naturally produce a sparse feature vector. Unlike logistic units that give small positive values when the input is not aligned with the filters (the incoming weight vectors of hidden units), ReLUs often give exact zeros. Furthermore, ReLUs compute faster than logistic or tanh units because they do not involve exponentiation and division; they just have to compute the max operation. Finally, ReLUs can potentially help jointly train multi-layers of features (instead of training layer by layer in greedy fashion) because ReLUs have been used to train supervised deep networks successfully [5, 15]. Therefore, here we will focus on SAEs with ReLUs (in the hidden layer). We call them Sparse Rectified Auto-Encoders (SRAEs).

3.1 Sparsity Constraint in SRAEs

The typical way that have been used to constrain sparsity in Sparse Auto-Encoders (SAEs) is pushing the average output $\bar{a}_j$ of hidden neuron j (over a minibatch) to some fixed target ρ (a value close to zero) [6, 4, 3]. In case the hidden neuron's output $\in [0, 1]$ (e.g. sigmoid unit), this can be done through the Kullback-Leibler (KL) divergence: $\sum_j \mathrm{KL}(\rho \| \bar{a}_j) = \sum_j \rho \log \frac{\rho}{\bar{a}_j} + (1-\rho) \log \frac{(1-\rho)}{(1-\bar{a}_j)}$. In case using ReLUs, the squared error can be used: $\sum_j (\bar{a}_j - \rho)^2$. Note that this way does not directly encourage the feature vector (corresponding to a example) to be sparse, but encourage the values of a feature (the outputs of a hidden neuron) over examples to be sparse. It, however, indirectly leads to a sparse feature vector because the reconstruction error tends to make learned features differ from each other; therefore, with an example, if some feature is active (having a non-zero value), the majority of the rest will be inactive (having a zero value).

This way, however, adds one more hyper-parameter (the fixed target ρ) to the list of SAEs' hyper-parameters which already has many ones (the trade-off parameter λ, the number of features, learning rate, minibatch size, and so on). As a result, the process of tuning hyper-parameters will become more annoying and more time-consuming. Why do not use L1-norm? It is natural because L1-norm is used in Sparse Coding. In addition, it doesn't have any extra hyper-parameter. It is also very simple; in case using ReLUs, it is just the sum of elements of the feature vector a. In the following section, we will explain the problem of training SAEs, in particular SRAEs, with L1-norm.

3.1.1 The Difficulty of Training SRAEs with L1-norm

The problem of training SAEs with L1-norm is that during the optimization process, L1-norm can drive the incoming weight vector of a hidden neurons to the state in which the hidden neuron is always inactive (produce zero with all examples in the dataset). And once the incoming weight vector has been in such a state, it will be stuck there forever and never get updated; the outgoing weight vector of this hidden neuron will also never get updated. Formally, let's consider a hidden neuron j which have a weight $W^{(e)}_{ji}$ connecting to an input neuron i and a weight $W^{(d)}_{kj}$ connecting to an output neuron k. The gradients of the objective function J in equation (2) (with the sparsity function $s(\cdot) = \|\cdot\|_1$) with respect to $W^{(e)}_{ji}$ and $W^{(d)}_{kj}$ are:

$$\frac{\partial J}{\partial W^{(d)}_{kj}} = \sum_{n=1}^{N} 2(\hat{x}^{(n)}_k - x^{(n)}_k) a^{(n)}_j \tag{3}$$

$$\frac{\partial J}{\partial W^{(e)}_{ji}} = \sum_{n=1}^{N} (\epsilon^{(n)}_j + \lambda) f'(a^{(n)}_j) x^{(n)}_i \tag{4}$$

where:

- $x^{(n)}_k$ and $\hat{x}^{(n)}_k$ are respectively the k^{th} element of the input vector $x^{(n)}$ and the reconstructed input vector $\hat{x}^{(n)}$.
- $a^{(n)}_j$ is the j^{th} element of the feature vector $a^{(n)}$.
- $\epsilon^{(n)}_j$ is the "error" that the hidden neuron j receives from the output layer (corresponding to the input $x^{(n)}$).

From equations (3) and (4), one can easily see that, during the optimization, if once the hidden neuron j has been in the state having a_j equal zero with all examples, the gradients $\frac{\partial J}{\partial W^{(d)}_{kj}}$ and $\frac{\partial J}{\partial W^{(e)}_{ji}}$ will be zeros with all examples (in case $f(\cdot)$ is the rectified linear function, the derivative $f'(0)$ equals 0) and the weights of this neuron will never get updated anymore. We call such neurons *"sleep" neurons*. Especially, the "easy to get exact zeros" property of ReLUs can make this problem easier to happen during the optimization.

The above problem may explain why people often don't use L1-norm in SAEs but instead, push the average output of a hidden neuron to a fixed target close to zero (but not zero!); this way may prevent the hidden neuron from the situation in which it is inactive for all examples and then never get updated. With sigmoid units, the KL divergence can be used and the average output cannot be zero because if so, the KL divergence will give an infinite penalty. With ReLUs, the KL divergence cannot be used because the outputs of ReLUs are not in $[0, 1]$. The squared error can be used instead but we found experimentally that the "sleep" neuron problem still happens. It is because with a zero average output, unlike the KL divergence, the squared error still gives a very small penalty. See Figure 1 for a comparison of them with the fixed target ρ of 0.1.

Although using L1-norm, Sparse Coding clearly doesn't have this problem because the encoder of Sparse Coding is implicit.

3.1.2 Sleep-Wake Stochastic Gradient Descent

To remedy the problem of training SRAEs with L1-norm, we propose a modified version of Stochastic Gradient Descent algorithm (SGD), called Sleep-Wake Stochastic Gradient Descent (SW-SGD). The idea is that during each epoch of SGD, we track the average outputs of hidden neurons. Then, after each epoch, we check if there are any "sleep" neurons (having the average output equal zero), and we will "wake-up" them by simply re-initializing their incoming weight vectors (including the biases). Despite its simplicity, our experiments showed that this strategy can help SRAEs successfully learn meaningful features without any "sleep" features.

3.2 Weight Constraint in SRAEs

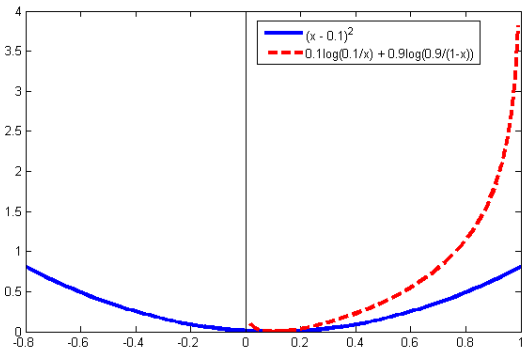


Figure 1: Comparison of KL divergence to squared error with the fixed target ρ of 0.1. When the average output of a hidden neuron is zero, KL divergence gives an infinite penalty while squared error still gives a very small penalty.

Besides sparsity constraint, weight constraint is also a key ingredient to get SAEs working. There are several ways have been used to constrain SAEs' weights:

- **Tied weights**: the weights of encoder and decoder are tied together ($W^{(d)} = (W^{(e)})^T$) [3]. This way was also used in other Auto-Encoder variants such as Denoising Auto-Encoders and Contractive Auto-Encoders [14, 13, 12]. Note that all [3, 14, 13, 12] used sigmoid units in the hidden layer. There is a trivial descent direction of SAEs' objective function in which the hidden neuron's output a_j is scaled down (by scaling down the incoming weight vector of this hidden neuron) and the outgoing weight vector of this hidden neuron is scaled up by some large constant; as a result, the sparsity penalty can decrease arbitrary while the reconstruction error is unchanged. Tied weights can help prevent from this trivial direction, but it is not clear what is going on when the encoder's weights and the decoder's weights are tied together, especially in case using sigmoid units.

- $W^{(d)}$ **norm constraint**: [15] constrained the basis vectors of the decoder (the outgoing weight vectors of hidden neurons) to have unit norm. This constraint is similar to Sparse Coding and also helps prevent from the scale problem. But how about the encoder's weights? For example, to be fair between features, the incoming weight vectors of hidden neurons should have the same norm.

- **Weight decay**: weights of the encoder and decoder are kept small by penalizing the sum of squares of them [6, 4]. As two previous ways, this way prevents SAEs from the scale problem too. It can be interpreted as a "soft" way to constrain the norms of the incoming weights vector of hidden units to be approximately equal to each other and the norms of the outgoing weight vectors of hidden units to be approximately equal to each other. However, this way introduces one more hyper-parameter; it's annoying.

3.2.1 *Our Proposed Weight Constraint for SRAEs*

In this section, we propose a reasonable way to constrain SRAEs' weights. It also doesn't introduce any extra hyper-parameter. Concretely, our way consists of two constraints:

- First, we tie the encoder's weights and the decoder's weights together: $W^{(d)} = \left(W^{(e)}\right)^T$

- Second, we also constrain the incoming weight vectors as well as the outgoing weight vectors of hidden units to have unit norm.

With an example x, if one just pays attention to non-zero rectified linear units, the whole system is a linear system. Therefore, with two above constraints, the encoder will project linearly the input vector x onto a few normalized basis vectors (in the whole set of normalized basis vectors) corresponding to non-zero hidden units; and then, the decoder will reconstruct the input vector from these basis vectors: $\hat{x} = W^T W x$ where x is a column vector and rows of W corresponds to normalized basis vectors selected by ReLUs (here, we just ignore the biases for simplicity). In other words, with above constraints, SRAEs will learn a set of normalized basis vectors such that different inputs can be explained by different small subsets of basis vectors (by projecting linearly the input onto the subset of basis vectors selected by ReLUs and then reconstructing the input from this subset).

The second constraint, however, cannot be enforced by gradient-based methods. To overcome this problem, we change the forward propagation formula of SRAEs as follows:

$$\hat{x} = (\hat{W}^{(e)})^T \max(0, \hat{W}^{(e)} x + b^{(e)}) + b^{(d)} \qquad (5)$$

where $\hat{W}^{(e)}$ is a row-normalized matrix of $W^{(e)}$ (each row of $W^{(e)}$ corresponds to an unnormalized basis vector). Here, the learned parameters are still $W^{(e)}$, $b^{(e)}$, and $b^{(d)}$. In this way, gradient-based methods can be used as usual.

Finally, the first constraint, tied weights, also helps save about haft of memory compared to untied weights. It will be beneficial when using GPU (for parallel computing).

4. EXPERIMENTS

4.1 Setup

We experimented on the MNIST dataset which composes of grayscale images (28×28 pixels) of 10 hand-written digits (from 0 to 9) [8]. Figure 2 shows some examples of this dataset. The images were preprocessed by scaling to $[0, 1]$. We used the usual split: $50,000$ examples for training, $10,000$ examples for validation, and $10,000$ examples for test.

We conducted all experiments using the Python Theano library [2], which allows for quick development and easy use of GPU (for parallel computing). We used a single NVIDIA GTX 560 GPU.

After the unsupervised feature learning phase, we evaluated the learned features by feeding them to a softmax regression and measuring the classification error. Concretely, given the training set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})\}$ where $x^{(i)} \in \mathbb{R}^D$ is the image vector and $y^{(i)} \in \{0, \ldots, 9\}$ is the class label, we fed $x^{(i)}$ to the trained Auto-Encoder (the Auto-Encoder was trained on the unlabeled data $\{x^{(1)}, \ldots, x^{(N)}\}$)

Figure 2: Some examples of MNIST dataset

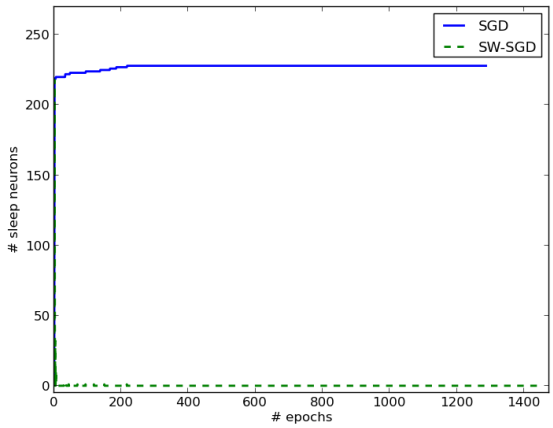


Figure 3: The number of "sleep" hidden neurons of SRAEs during the optimization process with SGD and SW-SGD. The optimization of SGD ended up with 228/1000 "sleep" neurons while SW-SGD ended up without any "sleep" neurons. (These two optimizations terminated after different number of epochs because of the early stopping strategy.)

to get the corresponding feature vector $f^{(i)}$; by this way, we got the new training set $\{(f^{(1)}, y^{(1)}), \ldots, (f^{(N)}, y^{(N)})\}$. Then, we used this new training set to train a softmax regression. With a test example x, we first used the trained Auto-Encoder to compute the feature vector f; then, we fed f to the trained softmax regression to get the class prediction.

In both unsupervised and supervised phase, we used Stochastic Gradient Descent as the optimization algorithm with mini-batch size 100 and early stopping (in the unsupervised phase, we stopped the optimization based on the objective value on the validation set; in the supervised phase, we based on the classification error on the validation set). In all experiments, we used SRAEs with 1000 hidden units, a trade-off parameter λ of 0.25, an unsupervised learning rate of 0.05, and a supervised learning rate of 1.

4.2 SGD versus SW-SGD

To see the problem of training SRAEs with L1-norm sparsity penalty and the effect of our "sleep-wake" strategy, we compared training SRAEs with ordinary Stochastic Gradient Descent (SGD) and our modified version, Sleep-Wake Stochastic Gradient Descent (SW-SGD). In this experiment, we used our proposed weight constraint (tied weights + $W^{(e)}$ norm constraint + $W^{(d)}$ norm constraint).

Figure 3 shows the number of "sleep" hidden neurons of SRAEs during the optimization process with SGD and with SW-SGD. The problem of training SRAEs with L1-norm sparsity penalty is that during the optimization, L1 penalty can push the incoming weight vectors of hidden neurons to "sleep" states (meaning that the corresponding hidden neurons always give zero outputs with all examples in the dataset) and then, they will never get updated anymore; as can be seen from the figure, with ordinary SGD, the number of "sleep" neurons increased during the optimization, especially during the first epochs when the optimization had not stable yet. The SGD optimization finally ended up with 228/1000 "sleep" neurons. This problem of L1 penalty can be remedied by our simple "sleep-wake" strategy; the SW-SGD optimization ended up without any "sleep" neurons.

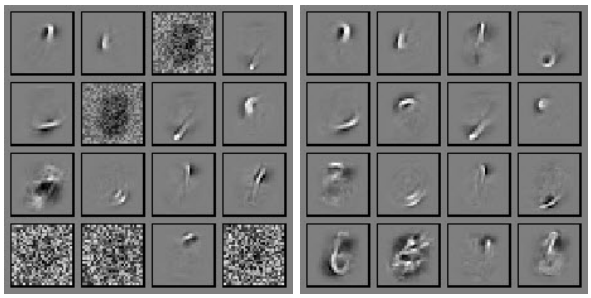
Figure 4 visualizes some example filters (the incoming weight vectors of hidden neurons) learned by SGD and SW-SGD. With SGD, there are five "sleep" filters; they look meaningless. With SW-SGD, there are not any "sleep" filters; all of them look meaningful, like "pen stroke" detectors.

Making use of all filters, SW-SGD achieved better training unsupervised objective value and better test classification performance (with softmax regression) than SGD (Table 1).

4.3 Our Proposed Weight Constraint versus Other Weight Constraints

In this second experiment, we compared our proposed weight constraint for SRAEs to other weight constraints that are possible to be applied to SRAEs. Concretely, we considered the following weight constraints:

- $W^{(d)}$ **norm constraint**: the outgoing weight vectors of hidden units (the columns of $W^{(d)}$) are constrained to have unit norm.



(a) SGD (b) SW-SGD

Figure 4: Figure (a) shows example filters learned by SGD; one can recognize there are five "sleep" filters looking meaningless. Figure (b) shows example filters learned by SW-SGD; all filters look meaningful, like "pen stroke" detectors.

Table 1: Unsupervised objective value on the training set and classification error (with softmax regression) on the test set when training SRAEs with SGD and with SW-SGD

	SGD	SW-SGD
Train Unsupervised Objective Value	9.84	**9.48**
Test Classification Error (%)	1.70	**1.62**

- $W^{(e)}$ **&** $W^{(d)}$ **norm constraint**: both the incoming and outgoing weight vectors of hidden units (the rows of $W^{(e)}$ and the columns of $W^{(d)}$ respectively) are constrained to have unit norm.
- **Tied weights**: the encoder's weights and the decoder's weights are tied together ($W^{(d)} = (W^{(e)})^T$).

Our weight constraint combines both $W^{(e)}$ **&** $W^{(d)}$ **norm constraint** and **tied weights**. In this experiment, we used SW-SGD to train SRAEs. As can be seen from Table 2, our weight constraint gave the best test classification performance (with softmax regression). In the last column, we also show the (approximate) training time per epoch of SRAEs with these different weight constraints (because of the early stopping strategy, the training processes of SAREs with different weight constraints can terminate after different number of epochs; therefore, it will be more accurate to compare them in term of the training time per epoch rather than the total training time). Weight constraints sorted from lowest to highest training time per epoch are: tied weights (2 seconds), $W^{(d)}$ norm constraint (3 seconds), our weight constraint (4 seconds), and $W^{(e)}$ & $W^{(d)}$ norm constraint (5 seconds). This order is reasonable because:

- In tied weights, SRAE doesn't have to do normalization in the forward propagation phase.
- In $W^{(d)}$ norm constraint, SRAE's decoder has to do normalization in the forward propagation phase; and because of this, in the back-propagation phase, the computation of derivatives with respect to the decoder's parameters will also become more expensive than usual.
- In our weight constraint, although we have to do normalization in both the encoder and decoder, we just have to compute the encoder's normalized weights and use them for the decoder thanks to the tied weights constraint. Its epoch time is higher than $W^{(d)}$ norm constraint above because in the back-propagation phase, the computation of derivatives with respect to both the encoder's parameters and the decoder's parameters is more expensive than usual.
- In $W^{(e)}$ & $W^{(d)}$ norm constraint, the training time per epoch is highest because SRAE has to do normalization in the encoder and decoder separately and the computation of derivatives with respect to both the encoder's parameters and the decoder's parameters is more expensive than usual.

Although the training time per epoch of our weight constraint is pretty high compared to other weigh constraints, it's still fast (thanks to the use of GPU). Its total training time is roughly 2.5 hours.

Table 2: Comparison of our weight constraint to other possible weight constraints. Our weight constraint gave the best classification performance (with softmax regression) on the test set. The last column shows the training time per epoch (roughly) of SRAEs with these different weight constraints.

Weight Constraint	Test Error (%)	Epoch Time (sec)
$W^{(d)}$ norm constraint	3.28	3
$W^{(e)}$ & $W^{(d)}$ norm constraint	2.51	5
Tied weights	2.04	2
Our weight constraint	**1.62**	4

Table 3: Comparison of SRAEs (with our weight constraint and SW-SGD) to other Auto-Encoder variants, including: Denoising Auto-Encoders (DAEs), Contractive Auto-Encoders (CAEs), and Higher Order Contractive Auto-Encoders (HCAEs), in term of classification error (with softmax regression) on the test set

Feature Learning Algorithm	Test Error (%)
DAEs [12]	2.05
CAEs [12]	1.82
SRAEs	1.62
HCAEs [12]	**1.20**

4.4 SRAEs versus Other Auto-Encoder Variants

Finally, we also compared SRAEs (with our weight constraint and SW-SGD) to other Auto-Encoder variants, including:

- **Denoising Auto-Encoders (DAEs)** [14]: want to learn robust features by making the input corrupted and trying to reconstruct the "clean" input from this corrupted version.
- **Contractive Auto-Encoders (CAEs)** [13]: want to learn features robust to small changes of the input by besides the reconstruction error, penalizing the Frobenius norm of the Jacobian of the feature vector with respect to the input vector.
- **Higher Order Auto-Encoders (HCAEs) [12]**: are the extension of CAEs; besides the reconstruction error and the Jacobian norm, HCAEs also penalize the approximated Hessian norm.

Table 3 compares the test classification performance of SRAEs to these Auto-Encoder variants. Note that with DAEs, CAEs, and HCAEs, [12] used 1000 hidden units, the sigmoid activation function in the hidden and output layer, the cross-entropy reconstruction error, and tied weights. Our SRAEs were better in term of test classification performance than DAEs and CAEs but worse than HCAEs. However, HCAEs are more complicated than our SRAEs with many hyper-parameters which need to be tuned.

5. CONCLUSION

In this paper, we have investigated SRAEs and in particular, two key ingredients to get SRAEs working: spar-

sity constraint and weight constraint. We have tried to understand the optimization problem when training SRAEs with L1-norm sparsity penalty and proposed a simple modified version of SGD, called SW-SGD, to remedy this problem. We have also proposed a reasonable weight constraint for SRAEs. Our experiments on the MNIST dataset have shown that our weight constraint and SW-SGD work well with SRAEs and can help SRAEs learn meaningful features that give excellent classification performance compared to other Auto-Encoder variants.

Our future work will include:

- Making use of sparsity to speed up the training.
- Unsupervised deep learning: SRAEs can be used to learn multiple layers of representation in greedy fashion but the interesting question is how to jointly learn multiple layers of representation?

6. REFERENCES

[1] Y. Bengio, A. C. Courville, and P. Vincent. Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR*, abs/1206.5538, 2012.

[2] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation.

[3] A. Coates. *Demystifying Unsupervised Feature Learning*. PhD thesis, Stanford University, 2012.

[4] A. Coates, A. Y. Ng, and H. Lee. An analysis of single-layer networks in unsupervised feature learning. In *International Conference on Artificial Intelligence and Statistics*, pages 215–223, 2011.

[5] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier networks. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics. JMLR W&CP Volume*, volume 15, pages 315–323, 2011.

[6] I. Goodfellow, H. Lee, Q. V. Le, A. Saxe, and A. Y. Ng. Measuring invariances in deep networks. In *Advances in neural information processing systems*, pages 646–654, 2009.

[7] Q. Le, M. Ranzato, R. Monga, M. Devin, K. Chen, G. Corrado, J. Dean, and A. Ng. Building high-level features using large scale unsupervised learning. In J. Langford and J. Pineau, editors, *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, ICML '12, pages 81–88, New York, NY, USA, July 2012. Omnipress.

[8] Y. LeCun. The MNIST database. `http://yann.lecun.com/exdb/mnist/`.

[9] H. Lee, A. Battle, R. Raina, and A. Ng. Efficient sparse coding algorithms. In *Advances in neural information processing systems*, pages 801–808, 2006.

[10] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

[11] B. A. Olshausen et al. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996.

[12] S. Rifai, G. Mesnil, P. Vincent, X. Muller, Y. Bengio, Y. Dauphin, and X. Glorot. Higher order contractive auto-encoder. *Machine Learning and Knowledge Discovery in Databases*, pages 645–660, 2011.

[13] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 833–840, 2011.

[14] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

[15] M. Zeiler, M. Ranzato, R. Monga, M. Mao, K. Yang, Q. Le, P. Nguyen, A. Senior, V. Vanhoucke, J. Dean, et al. On rectified linear units for speech processing. ICASSP, 2013.

TÀI LIỆU THAM KHẢO

[1] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder–decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, October 2014, pp. 1724–1734. [Online]. Available: http://www.aclweb.org/anthology/D14-1179 7

[2] M. Denil, L. Bazzani, H. Larochelle, and N. de Freitas, "Learning where to attend with deep architectures for image tracking," *CoRR*, vol. abs/1109.3737, 2011. [Online]. Available: http://arxiv.org/abs/1109.3737 26

[3] J. L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, no. 2, pp. 179 – 211, 1990. [Online]. Available: http://www.sciencedirect.com/science/article/pii/036402139090002E 12

[4] T. S. N. L. P. Group. (2015) Neural machine translation. [Online]. Available: https://nlp.stanford.edu/projects/nmt/ 39

[5] W. J. Hutchins, "Machine translation: A concise history," 2007. 2, 3

[6] S. Jean, K. Cho, R. Memisevic, and Y. Bengio, "On using very large target vocabulary for neural machine translation," *CoRR*, vol. abs/1412.2007, 2014. [Online]. Available: http://arxiv.org/abs/1412.2007 37

[7] N. Kalchbrenner and P. Blunsom, "Recurrent continuous translation models," in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Seattle, Washington, USA: Association for Computational

Linguistics, October 2013, pp. 1700–1709. [Online]. Available: http://www.aclweb.org/anthology/D13-1176 7

[8] P. Koehn, F. J. Och, and D. Marcu, "Statistical phrase-based translation," in *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, ser. NAACL '03. Stroudsburg, PA, USA: Association for Computational Linguistics, 2003, pp. 48–54. [Online]. Available: https://doi.org/10.3115/1073445.1073462 34

[9] H. Larochelle and G. E. Hinton, "Learning to combine foveal glimpses with a third-order boltzmann machine," in *Advances in Neural Information Processing Systems 23*, J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, Eds. Curran Associates, Inc., 2010, pp. 1243–1251. [Online]. Available: http://papers.nips.cc/paper/4089-learning-to-combine-foveal-glimpses-with-a-third-order-boltzmann-machine.pdf 26

[10] M. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *CoRR*, vol. abs/1508.04025, 2015. [Online]. Available: http://arxiv.org/abs/1508.04025 10, 32, 35, 39, 43

[11] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: A method for automatic evaluation of machine translation," in *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ser. ACL '02. Stroudsburg, PA, USA: Association for Computational Linguistics, 2002, pp. 311–318. [Online]. Available: https://doi.org/10.3115/1073083.1073135 40

[12] R. Pascanu, T. Mikolov, and Y. Bengio, "Understanding the exploding gradient problem," *CoRR*, vol. abs/1211.5063, 2012. [Online]. Available: http://arxiv.org/abs/1211.5063 9

[13] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017. 40

[14] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” *CoRR*, vol. abs/1409.3215, 2014. [Online]. Available: http://arxiv.org/abs/1409.3215 7, 9, 23

[15] K. Xu, J. Ba, R. Kiros, K. Cho, A. C. Courville, R. Salakhutdinov, R. S. Zemel, and Y. Bengio, “Show, attend and tell: Neural image caption generation with visual attention,” *CoRR*, vol. abs/1502.03044, 2015. [Online]. Available: http://arxiv.org/abs/1502.03044 31