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

TRẦN TRUNG KIÊN

HỌC ĐẶC TRƯNG KHÔNG GIÁM SÁT BẰNG AUTO-ENCODERS

Chuyên ngành: Khoa Học Máy Tính

Mã số chuyên ngành: 60 48 01

LUẬN VĂN THẠC SỸ: KHOA HỌC MÁY TÍNH

NGƯỜI HƯỚNG DẪN KHOA HỌC:

PGS.TS LÊ HOÀI BẮC

Tp. Hồ Chí Minh, Năm 2014

LỜI CẢM ƠN

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

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

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

TP. Hồ Chí Minh, 3/2014

Trần Trung Kiên

MỤC LỤC

DANH MỤC HÌNH ẢNH

DANH MỤC BẢNG

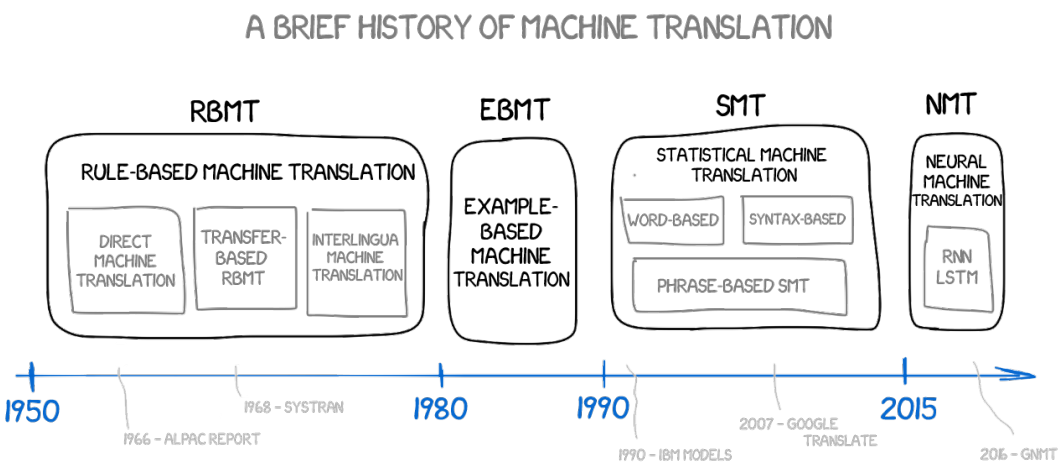
Chương 1

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 17, đã có những ý tưởng về một loại ngôn ngữ mang ý nghĩa phổ quát nhưng mãi đến những năm 1950 những nghiên cứu về dịch máy mới thật sự bắt đầu. Trong thời kì Chiến tranh Lạnh, vào ngày 7 tháng 1 năm 1954, tại trụ sở chính của IBM ở New York, thử nghiệm Georgetown-IBM được tiến hành. Máy tính IBM 701 đã tự động dịch 49 câu tiếng Nga sang tiếng Anh lần đầu tiên trong lịch sử chỉ sử dụng 250 từ vựng và sáu luật ngữ pháp. Thí nghiệm này được xem như là một thành công và mở ra kỉ nguyên cho những nghiên cứu với kinh phí lớn về dịch máy ở Hoa Kỳ. Ở Liên Xô những thí nghiệm tương tự cũng được thực hiện không lâu sau đó.

Trong một thập kỷ tiếp theo, nhiều nhóm nghiên cứu về dịch máy được thành lập. Một số nhóm chấp nhận phương pháp thử và sai, thường dựa trên thống kê với mục



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

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

1.1 Các phương pháp Dịch máy

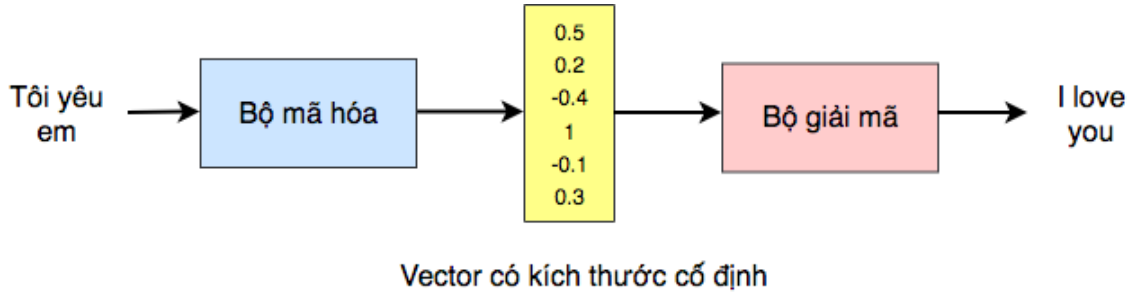
Từ đó đến nay, đã có nhiều hướng tiếp cập đã được sử dụng trong dịch máy với mục tiêu tạo ra bản dịch có độ chính xác cao và giảm thiểu công sức của con người có thể chia làm hai nhóm. Nhóm đầu tiên là những hướng tiếp cận dựa trên *Từ điển* (Dictionary based). Đây là hướng tiếp cận chính cho những nghiên cứu về Dịch máy trong những năm 1950-1960. Những phương pháp dựa trên hướng tiếp cập Từ điển có thể kể đến như *Dịch trực tiếp* (Direct machine translation), *Dịch máy chuyển dịch* (Transfer-based machine translation) hay *Dịch máy ngôn ngữ đại diện* (Interlingual machine translation). Điểm chung của những phương pháp này là dùng một từ điển để dịch các từ từ ngôn ngữ nguồn sang ngôn ngữ đích và sau đó cố gắng chỉnh sửa bản dịch để tạo ra một câu có nghĩa. Nhóm phương pháp này thường yêu cầu một bộ từ điển giữa hai ngôn ngữ cần dịch và một tập các quy tắc ngữ pháp cho mỗi ngôn ngữ. Bản dịch của hướng tiếp cận Từ điển thường có chất lượng kém và không sử dụng được trừ một số trường hợp đặc biệt. Ngoài ra chúng còn đòi hỏi một lượng nhân lực lớn với hiểu biết sâu sắc cho việc xây dựng những bộ từ điển và các quy tắc ngôn ngữ.

Nhóm thứ hai là những hướng tiếp cận dựa trên *Ngữ liệu* (Corpus based). Nhóm này hoạt động dựa trên một tập dữ liệu song song của các cặp câu là bản dịch của nhau trong hai ngôn ngữ gọi là Ngữ liệu và chỉ yêu cầu những tri thức tối thiểu về ngôn ngữ học.

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 trong số đó là việc những từ hay đoạn được dịch cục bộ và quan hệ của chúng với những từ cách xa chúng trong câu nguồn thường bị bỏ qua. Bên cạnh đó, 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, gióng hàng,.. 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

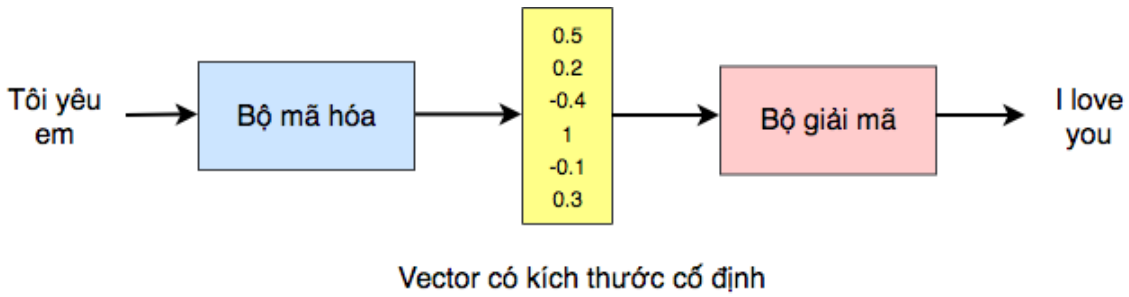


Hình 1.2: Ví dụ về Kiến trúc Bộ mã hóa - bộ giải mã trong dịch máy Nơ-ron

dịch máy trong những năm gần đây được đề xuất đầu tiên bởi Kalchbrenner and Blunsom (2013), Sutskever et al. (2014) and Cho et al. (2014b). 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) (Sutskever et al., 2014; Cho et al., 2014a). Bộ mã hóa thường là một mạng Nơ-ron có tác dụng *"nén"* tất cả thông tin của câu trong ngôn ngữ nguồn vào một vector có kích thước cố định. Bộ giải mã, cũng là một mạng Nơ-ron, sẽ tạo bản dịch trong ngôn ngữ đích từ vector có kích thước cố định kia. Toàn bộ hệ thống bao gồm bộ mã hóa và bộ giải mã sẽ được huấn luyện *"end-to-end"* để tạo ra bản dịch, quá trình này được mô tả như hình 1.2.

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 những từ cách xa nhau 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.



Hình 1.3: Ví dụ về Kiến trúc Bộ mã hóa - bộ giải mã trong dịch máy Nơ-ron

Chương 2

Kiến Thức Nền Tảng

Trong chương này, chúng tôi sẽ trình bày những kiến thức nền tảng trên ba chủ đề chính bao gồm cơ bản về *Mạng nơ-ron hồi quy (Recurrent neural network)* là thành phần xướng sống trong Dịch máy Nơ-ron và thiết kế cụ thể của nó trong Bộ mã hóa và Bộ giải mã. Tiếp theo chúng tôi nói về những vấn đề của mà Mạng nơ-ron hồi quy và *Long short-term memory* bản nâng cấp của của nó với khả năng giải quyết những vấn đề đó. Chúng tôi cũng trình bày về Mô hình dịch máy nơ-ron đã được đề cập đến trong chương giới thiệu. Những kiến thức được trình bày trong chương này cung cấp những nền tảng cũng như các vấn đề mà Mô hình dịch máy nơ-ron gặp phải để đi đến chương tiếp theo về cơ chế *Attention* trong dịch máy Nơ-ron.

2.1 "Mô hình ngôn ngữ (Language modeling)"

Mô hình ngôn ngữ ban đầu được sử dụng trong các hệ thống nhận dạng tiếng nói, ngày nay nói được sử dụng rộng rãi trong nhiều tác vụ khác của xử lý ngôn ngữ tự nhiên. Một mô hình ngôn ngữ được định nghĩa là một phân bố xác suất trên tập các chuỗi từ (câu). Cụ thể hơn, cho trước một từ từ vựng V là tập tất cả các từ khác nhau trong một ngôn ngữ, ví dụ, trong tiếng Việt ta có

$$V = \{\text{tôi, ăn, xe, đẹp, gà, vịt, qua,...}\}$$

Giả sử tập V là hữu hạn, một *câu* S được định nghĩa là chuỗi các từ

$$S = w_1 w_2 ... w_n$$

Trong đó $n \geq 1$ và $w_i \in V$ với $i \in 1...n$. Gọi D là tập bao gồm N câu trong sao cho

$$D = \{S^1, S^2, ..., S^N\}$$

Với mỗi câu S^n mang chiều dài n có dạng:

$$S^n = w_1^n, w_2^n, ..., w_{T^n}^n$$

Một cách cụ thể, với một véc-tơ đầu vào x có kích thước $D_x \times 1$, "Sparse Coding" tối thiểu hóa hàm chi phí sau sau:

$$C(W,h) = ||Wh - x||_2^2 + \lambda ||h||_1 \tag{2.1}$$

với ràng buộc là các véc-tơ cơ sở (ứng với các cột của W) được chuẩn hóa (có độ dài bằng 1).

Ở đây:

- Các biến tối ưu hóa là W và h. Trong đó, W là ma trận chứa các véc-tơ cơ sở (mỗi cột của W ứng với một véc-tơ cơ sở); W có kích thước $D_x \times D_h$ (D_x là số chiều của không gian ban đầu, D_h là số chiều của không gian đặc trưng). h là véc-tơ đặc trưng (véc-tơ hệ số) tương ứng với véc-tơ đầu vào x; h có kích thước $D_h \times 1$. Ma trận các véc-tơ cơ sở W dùng chung cho tất cả các mẫu huấn luyện, còn véc-tơ hệ số h thay đổi theo từng mẫu huấn luyện. Lưu ý là $C(W,h)$ là hàm chi phí cho một mẫu huấn luyện; chúng tôi chỉ viết hàm chi phí cho một mẫu huấn luyện để đơn giản về mặt ký hiệu. Trong thực tế, mục tiêu là tối thiểu hóa chi phí trên toàn bộ tập huấn luyện (bằng trung bình của chi phí của các mẫu huấn luyện) và sự cập nhật các tham số có thể được tiến hành với một mẫu huấn luyện, hoặc với một số mẫu huấn luyện, hoặc với toàn bộ mẫu trong tập huấn luyện.

- $||\cdot||_p$ là ký hiệu của chuẩn p (p-norm) với $||x||_p = (\sum_{i=1}^n |x_i|^p)^{\frac{1}{p}}$, trong đó x_i là

phần tử thứ i của véc-tơ x.

Với hàm mục tiêu trên, "Sparse Coding" muốn tìm ra véc-tơ biểu diễn đặc trưng h thỏa hai tính chất sau:

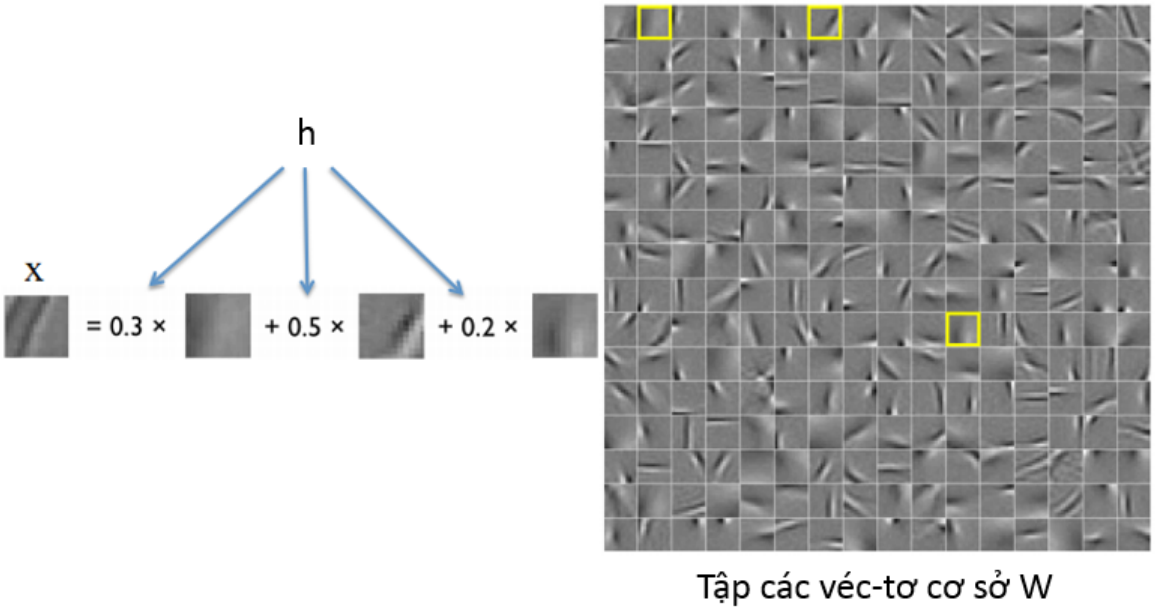
- Có thể tái tạo lại được tốt véc-tơ đầu vào x (bằng cách tối thiểu hóa độ lỗi tái tạo $||Wh - x||_2^2$).
- Thưa (bằng cách tối thiểu hóa chuẩn L1 $||h||_1$).

λ là siêu tham số (hyper-parameter, là tham số phải chọn trước khi huấn luyện) điều khiển "sự thỏa hiệp" giữa khả năng tái tạo và độ thưa. Nếu véc-tơ đặc trưng h càng thưa thì khả năng tái tạo lại véc-tơ đầu vào ban đầu càng thấp và ngược lại. Do đó, để học được các đặc trưng tốt, ta cần phải chọn giá trị λ trung dung sao cho véc-tơ đặc trưng vừa thưa và vừa có thể tái tạo tốt véc-tơ đầu vào ban đầu.

Hàm chi phí (2.1) có thể được tối thiểu hóa bằng cách lặp cho đến khi hội tụ, trong đó ở mỗi vòng lặp, các biến W và h sẽ được tối ưu một cách luân phiên nhau: đầu tiên, cố định h và tối thiểu hóa hàm mục tiêu theo W; sau đó, lại cố định W và tối thiểu hóa hàm mục tiêu theo h [5]. Tuy nhiên, quá trình tối ưu hóa này của "Sparse Coding" thường tốn nhiều thời gian để có thể hội tụ.

Một điểm hạn chế nữa của "Sparse Coding" là sau khi huấn luyện xong, với một véc-tơ đầu vào mới, để tìm ra véc-tơ đặc trưng tương ứng, ta vẫn phải tiến hành tối thiểu hóa hàm chi phí (2.1) với W cố định.

Một kết quả được biết đến phổ biến của "Sparse Coding" là nếu huấn luyện "Sparse Coding" trên ảnh tự nhiên thì các đặc trưng học được (các véc-tơ cơ sở) sẽ có dạng các cạnh ở các vị trí khác nhau và với các hướng khác nhau (minh họa ở hình 2.1); các đặc trưng này tương tự với các đặc trưng quan sát được ở vùng vỏ não thị giác V1.



Hình 2.1: Minh họa các đặc trưng (các véc-tơ cơ sở) học được của "Sparse Coding" khi huấn luyện trên ảnh tự nhiên [12]. Các đặc trưng học được có dạng các cạnh ở các vị trí khác nhau và với các hướng khác nhau. Véc-tơ đầu vào x có thể được tái tạo từ một số ít các đặc trưng trong tập các đặc trưng; nghĩa là, đa số các phần tử của véc-tơ đặc trưng (véc-tơ hệ số) h bằng 0 (trong hình vẽ chỉ thể hiện các phần tử khác 0 của h).

2.2 "Sparse Auto-Encoders"

"Auto-Encoder" đơn giản là một mạng nơ-ron truyền thẳng gồm có hai phần:

- Phần thứ nhất, được gọi là *bộ mã hóa* (encoder), ánh xạ véc-tơ đầu vào $x \in \mathbb{R}^{D_x \times 1}$ sang véc-tơ biểu diễn ẩn $h \in \mathbb{R}^{D_h \times 1}$ theo công thức:

$$h = f(W^{(e)}x + b^{(e)}) \tag{2.2}$$

  Trong đó, $W^{(e)} \in \mathbb{R}^{D_h \times D_x}$ và $b^{(e)} \in \mathbb{R}^{D_h \times 1}$ là các tham số của bộ mã hóa. $f(\cdot)$ là một hàm kích hoạt nào đó; nói rõ hơn là, $f(\cdot)$ nhận đầu vào là một véc-tơ và trả về véc-tơ kết quả có cùng kích thước với véc-tơ đầu vào của $f(\cdot)$, trong đó mỗi phần tử của véc-tơ kết quả có được bằng cách áp dụng hàm kích hoạt (ví dụ, hàm sigmoid) lên phần tử tương ứng của véc-tơ đầu vào của $f(\cdot)$.

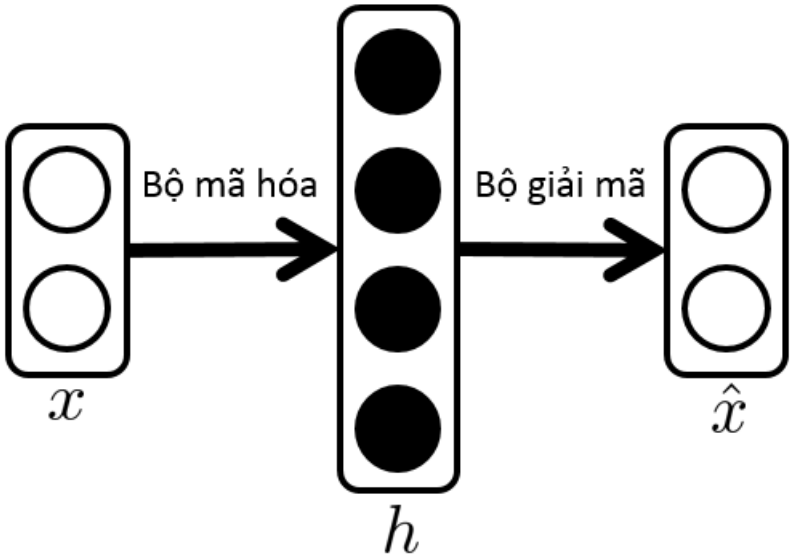
- Phần thứ hai, được gọi là *bộ giải mã* (decoder), cố gắng tái tạo lại véc-tơ đầu vào x ban đầu từ véc-tơ biểu diễn ẩn h:

$$\hat{x} = W^{(d)}h + b^{(d)} \tag{2.3}$$

  Trong đó, $\hat{x} \in \mathbb{R}^{D_x \times 1}$ là véc-tơ tái tạo, $W^{(d)} \in \mathbb{R}^{D_x \times D_h}$ và $b^{(d)} \in \mathbb{R}^{D_x \times 1}$ là các tham số của bộ giải mã.

Như vậy, từ véc-tơ đầu vào, "Auto-Encoder" ánh xạ sang véc-tơ biểu diễn ẩn; rồi từ véc-tơ biểu diễn ẩn này, "Auto-Encoder" cố gắng tái tạo lại véc-tơ đầu vào ban đầu (minh họa ở hình 2.2). Bằng cách này, ta hy vọng có thể thu được ở véc-tơ biểu diễn ẩn những thông tin có ích, giải thích dữ liệu quan sát được (véc-tơ đầu vào).

"Sparse Auto-Encoder" (SAE) là một "Auto-Encoder" trong đó véc-tơ biểu diễn ẩn được ràng buộc thưa (nghĩa là, với một véc-tơ đầu vào, chỉ có một số nơ-ron ẩn kích hoạt). Một cách cụ thể, với một mẫu huấn luyện $x \in \mathbb{R}^{D_x}$, SAEs tối thiểu hóa hàm chi phí sau (tương tự như "Sparse Coding", để đơn giản về mặt ký hiệu, ở đây chúng tôi chỉ ghi hàm chi phí cho một mẫu huấn luyện; trong thực tế, mục tiêu là tối thiểu hóa chi phí trên toàn bộ tập huấn luyện và sự cập nhật các tham số có thể được tiến hành với một mẫu huấn luyện, hoặc với một số mẫu huấn luyện, hoặc với toàn bộ mẫu



Hình 2.2: Minh họa "Auto-Encoders"

trong tập huấn luyện):

$$C(W^{(e)}, b^{(e)}, W^{(d)}, b^{(d)}) = ||x - \hat{x}||_2^2 + \lambda s(h) \tag{2.4}$$

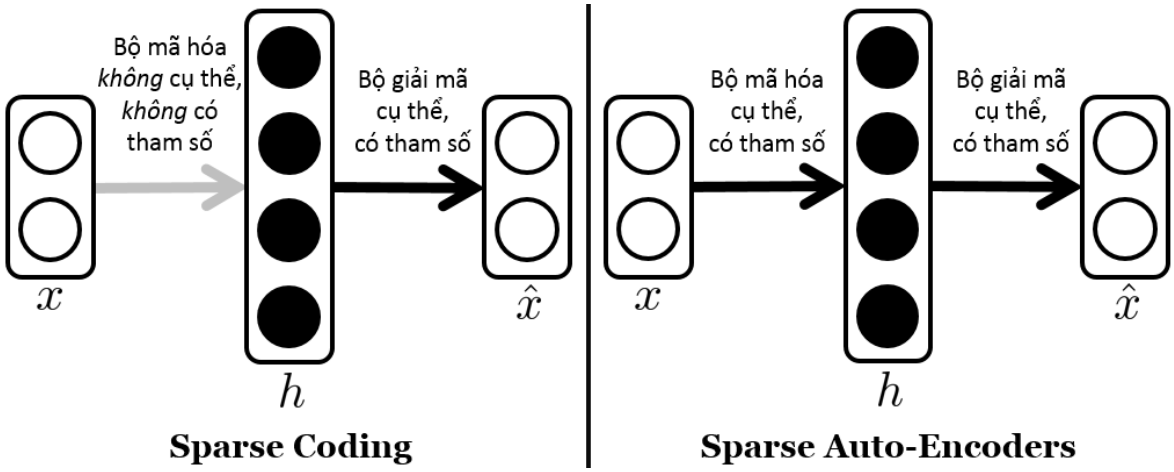
Trong đó, $\hat{x}$ là véc-tơ tái tạo (với $\hat{x} = W^{(d)}h + b^{(d)}$ và $h = f(W^{(e)}x + b^{(e)})$); $s(\cdot)$ là một hàm nào đó mà làm cho véc-tơ biểu diễn ẩn h thưa (ví dụ, $s(\cdot)$ có thể là chuẩn L1 như ở "Sparse Coding"); và λ là siêu tham số điều khiển "sự thỏa hiệp" giữa độ lỗi tái tạo và độ thưa.

Như vậy, ta thấy rằng, mục tiêu của SAEs giống với "Sparse Coding", đó là tìm ra véc-tơ biểu diễn đặc trưng (véc-tơ biểu diễn ẩn) thỏa hai tính chất:

- Có thể tái tạo tốt véc-tơ đầu vào.
- Thưa.

Tuy nhiên, điểm khác biệt giữa chúng là (minh họa ở hình 2.3): SAEs có bộ mã hóa *cụ thể, có tham số* (nghĩa là, có hàm cụ thể ánh xạ từ véc-tơ đầu vào sang véc-tơ đặc trưng); trong khi đó, bộ mã hóa của "Sparse Coding" *không cụ thể, không có tham số* (nghĩa là, không có hàm cụ thể ánh xạ từ véc-tơ đầu vào sang véc-tơ đặc trưng). Điểm khác biệt này giúp cho SAEs có một số lợi thế so với "Sparse Coding":

- Việc huấn luyện SAEs có thể được thực hiện hiệu quả hơn "Sparse Coding" thông qua thuật toán lan truyền ngược.



Hình 2.3: So sánh giữa "Sparse Coding" và SAEs. SAEs có bộ mã hóa *cụ thể, có tham số* ($h = f(W^{(e)}x + b^{(e)})$); trong khi đó, bộ mã hóa của "Sparse Coding" *không cụ thể, không có tham số*.

- Sau khi huấn luyện, với một véc-tơ đầu vào mới, SAEs có thể tính ra được véc-tơ đặc trưng rất nhanh bằng cách lan truyền tiến qua bộ mã hóa; trong khi đó, "Sparse Coding" vẫn phải tiến hành quá trình tối ưu hóa.

2.3 "Softmax Regression"

"Softmax Regression" là mô hình phân K lớp. Trong ngữ cảnh của bài toán học đặc trưng, "Softmax Regression" thường được dùng để đánh giá các đặc trưng học được (bởi mô hình này đơn giản, không có nhiều siêu tham số).

2.3.1 Hàm dự đoán của "Softmax Regression"

Với một véc-tơ đầu vào $x \in \mathbb{R}^{D\times 1}$, hàm dự đoán $h(x)$ của "Softmax Regression" sẽ trả về một véc-tơ gồm có K phần tử (ứng với K lớp), trong đó phần tử thứ k của véc-tơ này cho biết xác suất $p(y = k|x)$ với $y \in \{1,\ldots,K\}$ là nhãn lớp của véc-tơ đầu vào x. Như vậy, ta có thể quyết định x sẽ thuộc về lớp mà có xác suất lớn nhất.

Cụ thể, hàm dự đoán $h(x)$ của "Softmax Regression" như sau:

$$
\begin{aligned}
h(x) &= \begin{bmatrix} p(y=1|x) \\ p(y=2|x) \\ \vdots \\ p(y=K|x) \end{bmatrix} \\
&= \begin{bmatrix} \frac{\exp(W_1^T x+b_1)}{\sum_{k=1}^{K}\exp(W_k^T x+b_k)} \\ \frac{\exp(W_2^T x+b_2)}{\sum_{k=1}^{K}\exp(W_k^T x+b_k)} \\ \vdots \\ \frac{\exp(W_K^T x+b_K)}{\sum_{k=1}^{K}\exp(W_k^T x+b_k)} \end{bmatrix}
\end{aligned}
\tag{2.5}
$$

với $W = \{W_1, \ldots, W_K\}$ ($W_k \in \mathbb{R}^{D\times 1}$) và $b = \{b_1, \ldots, b_K\}$ ($b_k \in \mathbb{R}$) là các tham số của hàm dự đoán. Để ý tổng các phần tử của véc-tơ $h(x)$ bằng 1.

2.3.2 Tìm các tham số của hàm dự đoán của "Softmax Regression"

Cho tập huấn luyện $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})\}$. Để tìm ra được các tham số W và b của hàm dự đoán của "Softmax Regression" ở công thức (2.5), ta sẽ dùng phương pháp "maximum likelihood". Giả sử các mẫu dữ liệu trong tập huấn luyện được phát sinh một cách độc lập với nhau, ta có hàm "likelihood":

$$
\begin{aligned}
L(W,b) &= p(Y|X) \\
&= \prod_{i=1}^{N} p(y^{(i)}|x^{(i)}) \\
&= \prod_{i=1}^{N}\prod_{j=1}^{K}\left(\frac{\exp(W_j^T x+b_j)}{\sum_{k=1}^{K}\exp(W_k^T x+b_k)}\right)^{1\{y^{(i)}=j\}}
\end{aligned}
\tag{2.6}
$$

Trong đó:

- $X = \{x^{(1)}, \ldots, x^{(N)}\}$ và $Y = \{y^{(1)}, \ldots, y^{(N)}\}$.
- Hàm $1\{y^{(i)} = j\}$ sẽ trả về 1 nếu $y^{(i)} = j$ và trả về 0 nếu ngược lại.

Ta tìm W và b sao cho hàm "likelihood" $L(W,b)$ đạt cực đại. Cực đại $L(W,b)$ tương đương với cực tiểu $-\log L(W,b)$ (hàm này được gọi là hàm "negative log-likelihood").

Như vậy, ta sẽ tìm các tham số W và b của hàm dự đoán của "Softmax Regression" sao cho hàm chi phí sau đạt cực tiểu:

$$\begin{aligned} C(W,b) &= -\log L(W,b) \\ &= -\sum_{i=1}^{N}\sum_{j=1}^{K} 1\{y^{(i)}=j\} \log \frac{\exp(W_j^T x + b_j)}{\sum_{k=1}^{K}\exp(W_k^T x + b_k)} \end{aligned} \tag{2.7}$$

Để cực tiểu hóa hàm này, ta có thể sử dụng thuật toán "Gradient Descent" (sẽ được trình bày ở dưới).

2.4 "Gradient Descent"

2.4.1 "Batch Gradient Descent"

Thuật toán "Batch Gradient Descent" (BGD) dùng để cực tiểu hóa hàm chi phí $C(W)$ *trên toàn bộ tập huấn luyện* theo tham số W (ví dụ, $C(W)$ có thể là hàm chi phí trên toàn bộ tập huấn luyện của "Auto-Encoders" hay của "Softmax Regression"). Một cách cụ thể, xét hàm chi phí trên toàn bộ tập huấn luyện của một mô hình học nào đó (ví dụ, "Auto-Encoders" hay "Softmax Regression"):

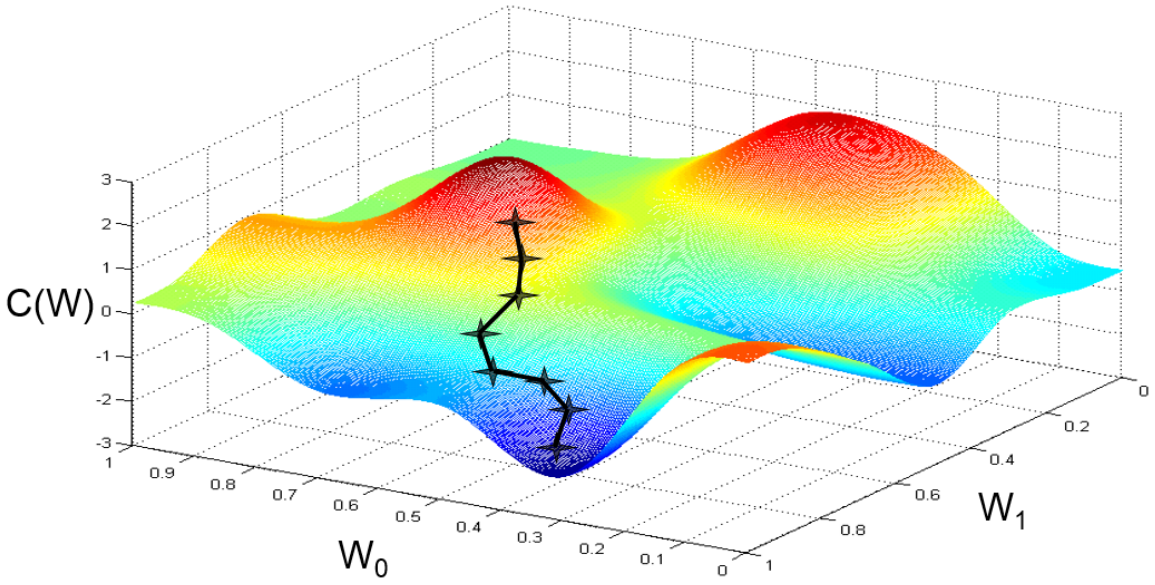
$$C(W) = \frac{1}{N}\sum_{i=1}^{N} C^{(i)}(W) \tag{2.8}$$

Trong đó:

- W là các tham số của mô hình học.
- $C^{(i)}(W)$ là chi phí của mẫu huấn luyện thứ i trong tập huấn luyện.
- N là tổng số mẫu huấn luyện.

Mục tiêu của ta là tìm W để $C(W)$ đạt cực tiểu.

Ý tưởng của BGD là đầu tiên khởi tạo ngẫu nhiên W, rồi nhìn vùng cục bộ xung quanh W và đi (cập nhật W) theo hướng mà làm cho $C(W)$ giảm nhiều nhất; tại W



Hình 2.4: Minh họa quá trình chạy của thuật toán BGD (hình vẽ được điều chỉnh từ hình vẽ lấy từ slide bài giảng của GS. Andrew Ng trong lớp máy học trực tuyến ở trang coursera.org).

mới, ta lại lặp lại qui trình này: nhìn vùng cục bộ xung quanh W và đi theo hướng mà làm cho $C(W)$ giảm nhiều nhất; cứ thế..., ta lặp cho đến khi hội tụ. Hình 2.4 minh họa cho quá trình chạy này của BGD với trường hợp đơn giản là W chỉ gồm có 2 thành phần là W_0 và W_1.

Cụ thể, ở mỗi vòng lặp, ta sẽ cập nhật W theo công thức:

$$W = W + \eta \hat{v} \tag{2.9}$$

Trong đó:

- $\hat{v}$ là véc-tơ đơn vị có cùng kích thước với W cho biết hướng đi (hướng cập nhật W) mà sẽ làm cho $C(W)$ giảm nhiều nhất xét trong vùng cục bộ xung quanh W hiện tại.
- η là hằng số dương điều khiển độ dài của một bước đi.

Ta nên đi theo hướng $\hat{v}$ nào để làm cho $C(W)$ giảm nhiều nhất xét trong vùng cục bộ xung quanh W hiện tại? Xét hiệu sau:

$$\Delta C = C(W + \eta \hat{v}) - C(W) \tag{2.10}$$

Ta cần tìm $\hat{v}$ để làm cho ΔC có giá trị âm nhỏ nhất. BGD xấp xỉ $C(W+\eta\hat{v})$ bằng cách sử dụng khai triển Taylor *đến số hạng ứng với đạo hàm bậc nhất* (để ý $W+\eta\hat{v}$ là điểm lân cận xung quanh W):

$$C(W+\eta\hat{v}) \approx C(W) + \eta\nabla C(W)^T\hat{v} \tag{2.11}$$

với $\nabla C(W)$ là véc-tơ chứa các đạo hàm riêng của C theo W (ở đây, khi nói đến véc-tơ, ta ngầm hiểu là véc-tơ cột). Thế công thức (2.11) vào công thức (2.10) ta được:

$$\begin{aligned}\Delta C &= \eta\nabla C(W)^T\hat{v} \\ &= \eta\|\nabla C(W)\|\|\hat{v}\|\cos(\nabla C(W);\hat{v}) \\ &= \eta\|\nabla C(W)\|\cos(\nabla C(W);\hat{v}) \\ &\geq -\eta\|\nabla C(W)\|\end{aligned} \tag{2.12}$$

Ta thấy ΔC sẽ có giá trị âm nhỏ nhất khi cos của góc tạo bởi hai véc-tơ $\nabla C(W)$ và $\hat{v}$ có giá trị bằng -1; nghĩa là, $\hat{v}$ sẽ có chiều ngược với chiều của $C(W)$. Và vì $\hat{v}$ là véc-tơ đơn vị nên cuối cùng ta có:

$$\hat{v} = -\frac{\nabla C(W)}{\|\nabla C(W)\|} \tag{2.13}$$

Như vậy, ta có công thức cập nhật tham số ở mỗi vòng lặp của BGD như sau:

$$W = W - \eta\frac{\nabla C(W)}{\|\nabla C(W)\|} \tag{2.14}$$

Với công thức cập nhật tham số trên, ở mỗi vòng lặp, BGD sẽ luôn đi một bước có độ dài cố định là η. Tuy nhiên, ta thấy rằng khi $\|\nabla C(W)\|$ lớn (độ dốc lớn), ta muốn đi một bước dài; và khi $\|\nabla C(W)\|$ nhỏ (độ dốc nhỏ, nhiều khả năng gần cực trị), ta muốn đi một bước ngắn. Nghĩa là, thay vì dùng độ dài bước đi η cố định, ta muốn dùng η thay đổi và tỉ lệ thuận với $\|\nabla C(W)\|$:

$$\eta = \alpha\|\nabla C(W)\| \tag{2.15}$$

với α là hằng số dương cho biết mức độ tỉ lệ thuận giữa $\|\nabla C(W)\|$ và η; α được gọi là hệ số học (learning rate). Thế (2.15) vào (2.14) ta được công thức cập nhật tham số

của BGD:

$$W = W - \alpha \nabla C(W) \tag{2.16}$$

Nếu α lớn thì ta sẽ đi được một bước dài nhưng có nguy cơ ra khỏi vùng xấp xỉ cục bộ của khai triển Taylor (nghĩa là không đảm bảo sau khi cập nhật W sẽ làm cho giá trị của hàm chi phí C giảm). Nếu α nhỏ thì sẽ đảm bảo nằm trong vùng xấp xỉ cục bộ của khai triển Taylor nhưng thời gian học sẽ rất lâu (vì mỗi lần cập nhật chỉ đi được một bước ngắn). Do đó, cần chọn giá trị α trung dung.

Tổng thể thuật toán BGD được trình bày ở thuật toán 2.1. Cách xác định điều kiện dừng của thuật toán sẽ được trình bày ở mục 2.4.3.

Thuật toán 2.1 Batch Gradient Descent (BGD)

Đầu vào: Tập huấn luyện, hệ số học $\alpha > 0$
Đầu ra: Bộ tham số W của mô hình học để cho hàm chi phí $C(W)$ đạt cực tiểu
Thao tác:

1: Khởi tạo ngẫu nhiên cho W
2: **while** chưa thỏa điều kiện dừng **do**
3: $\quad W = W - \alpha \nabla C(W)$ %% C là hàm chi phí trên toàn bộ tập huấn luyện
4: **end while**

2.4.2 "Stochastic Gradient Descent"

Thuật toán "Stochastic Gradient Descent" (SGD) là cải tiến của "Batch Gradient Descent" (BGD) để tăng tốc quá trình tối ưu hóa khi phải làm việc với tập dữ liệu lớn. Một cách cụ thể, xét công thức cập nhật tham số (2.16) của BGD, ta thấy để đi một bước (thực hiện một lần cập nhật W), ta cần phải tính véc-tơ đạo hàm riêng $\nabla C(W)$. Từ công thức (2.8) của hàm chi phí $C(W)$ ta có:

$$\nabla C(W) = \frac{1}{N} \sum_{i=1}^{N} \nabla C^{(i)}(W) \tag{2.17}$$

Nghĩa là với BGD, để đi được một bước, ta cần phải duyệt hết toàn bộ tập huấn luyện để tính các véc-tơ đạo hàm riêng $\nabla C^{(i)}(W)$ của hàm chi phí của mỗi mẫu huấn luyện, rồi sau đó lấy trung bình các véc-tơ đạo hàm riêng này để ra được $\nabla C(W)$. Khi mà tập huấn luyện lớn, quá trình này sẽ tốn thời gian và làm cho BGD chạy rất chậm.

SGD khắc phục nhược điểm trên của BGD bằng cách: thay vì phải duyệt tất cả các mẫu trong tập huấn luyện và tính véc-tơ đạo hàm riêng trung bình rồi mới đi được một bước như ở BGD, SGD chỉ duyệt qua *một số mẫu* trong tập huấn luyện, tính véc-tơ đạo hàm riêng trung bình *trên tập con này*, rồi đã đi ngay một bước. Ví dụ, với tập huấn luyện có 1000 mẫu, BGD sẽ duyệt qua hết 1000 mẫu này rồi mới đi được một bước; trong khi đó, với 10 mẫu đầu tiên, SGD đi được một bước, với 10 mẫu kế tiếp, SGD đi được một bước nữa... (ở đây, giả sử số lượng mẫu mà SGD cần duyệt qua để đi được một bước là 10). Như vậy, với một lần quét qua toàn bộ tập huấn luyện, BGD chỉ đi được 1 bước, trong khi đó SGD đi được tới 100 bước. Nguyên 1000 mẫu được gọi là một "batch", còn tập gồm 10 mẫu để SGD đi được một bước gọi là một "mini-batch"; ở đây, ta nói kích thước của "mini-batch" bằng 10. Một lần duyệt qua toàn bộ tập huấn luyện được gọi là một "epoch"; như vậy, SGD sẽ thực hiện nhiều "epoch", trong mỗi "epoch" lại thực hiện nhiều lần cập nhật tham số ứng với các "mini-batch".

Tại sao SGD hoạt động? Ta thấy hướng đi của BGD được tính bằng cách lấy trung bình trên *toàn bộ tập huấn luyện* các véc-tơ đạo hàm riêng $\nabla C^{(i)}(W)$, còn hướng đi của SGD được tính bằng cách lấy trung bình trên *một tập con (một "mini-batch") của tập huấn luyện* các véc-tơ đạo hàm riêng $\nabla C^{(i)}(W)$. Như vậy, tuy hướng đi của SGD không chính xác hoàn toàn với hướng đi của BGD nhưng nó sẽ giao động xung quanh hướng đi của BGD; hay nói một cách khác, hướng đi của SGD xấp xỉ hướng đi của BGD. Nếu ta chọn kích thước của "mini-batch" nhỏ (tối thiểu là bằng 1) thì SGD sẽ chạy nhanh nhưng độ "nhiễu loạn" (độ giao động xung quanh hướng đi của BGD) sẽ tăng; còn nếu ta chọn kích thước của "mini-batch" lớn (tối đa là bằng số lượng mẫu của tập huấn luyện, lúc này SGD trở thành BGD) thì độ "nhiễu loạn" sẽ giảm nhưng SGD sẽ chạy chậm. Do đó, cần chọn kích thước "mini-batch" có giá trị trung dung. Lưu ý là tính "nhiễu loạn" của SGD cũng sẽ thể có lợi khi hàm chi phí có "bề mặt" phức tạp (ví dụ như hàm chi phí của mạng nơ-ron); chẳng hạn, tính "nhiễu loạn" có thể giúp SGD "nhảy" ra khỏi những vùng cực trị cục bộ, hay không bị mắc kẹt ở những vùng "đồng bằng". Ngoài ra, khi chọn kích thước của "mini-batch" > 1, ta sẽ có thể tận dụng được sức mạnh tính toán song song.

Tổng thể thuật toán SGD được trình bày ở thuật toán 2.2. Cách xác định điều kiện dừng của thuật toán sẽ được trình bày ở mục 2.4.3.

Thuật toán 2.2 Stochastic Gradient Descent (SGD)

Đầu vào: Tập huấn luyện gồm N mẫu, hệ số học $\alpha > 0$, kích thước "mini-batch" B
Đầu ra: Bộ tham số W của mô hình học để cho hàm chi phí $C(W)$ đạt cực tiểu
Thao tác:

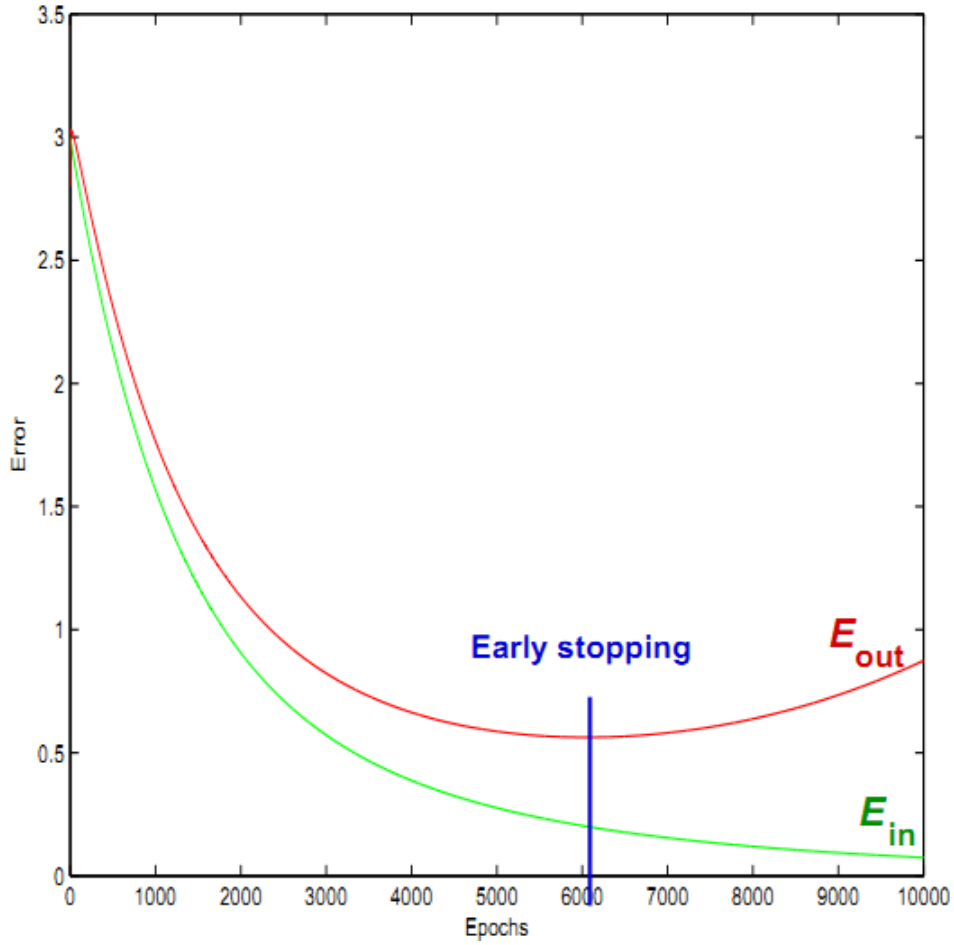
Khởi tạo ngẫu nhiên cho W
while chưa thỏa điều kiện dừng **do** %% Với mỗi "epoch"
    Xáo trộn ngẫu nhiên thứ tự của các mẫu trong tập huấn luyện (thường sẽ giúp SGD hội tụ nhanh hơn)
    for b = 1 : N/B **do** %% Với mỗi "mini-batch"
        $W = W - \alpha \frac{1}{B} \sum_{i=(b-1)B+1}^{bB} \nabla C^{(i)}(W)$
    end for
end while

2.4.3 Chiến lược "dừng sớm"

"Dừng sớm" (early stopping) là một cách "miễn phí" để quyết định số vòng lặp của SGD (hay BGD). Sở dĩ nói "miễn phí" là vì để chọn một siêu tham số, thông thường ta cần phải tiến hành huấn luyện nhiều lần với các giá trị khác nhau của siêu tham số này, và chọn ra giá trị mà cho kết quả tốt nhất trên tập "validation" (tập ngoài tập huấn luyện); ở đây, với chiến lược "dừng sớm", số lượng vòng lặp sẽ được xác định ngay trong quá trình huấn luyện (nghĩa là, chỉ tốn một lần huấn luyện). Ngoài ra, chiến lược "dừng sớm" cũng giúp chống vấn đề quá khớp (overfitting).

Ý tưởng của chiến lược "dừng sớm" đơn giản là trong khi thực hiện các vòng lặp của quá trình tối ưu hóa với SGD (hay BGD), ta sẽ theo dõi "độ lỗi" trên tập "validation" (ở đây, "độ lỗi" được định nghĩa tùy theo ngữ cảnh; ví dụ, nếu dùng SGD để cực tiểu hóa hàm chi phí của "Softmax Regression" thì "độ lỗi" có thể là tỉ lệ phân lớp sai, còn nếu dùng SGD để cực tiểu hóa hàm chi phí của SAEs thì "độ lỗi" có thể là giá trị của hàm chi phí). Khi SGD (hay BGD) càng thực hiện nhiều vòng lặp thì nhìn chung "độ lỗi" trên tập huấn luyện sẽ càng giảm xuống, còn "độ lỗi" trên tập "validation" (ngoài tập huấn luyện) ban đầu sẽ giảm xuống nhưng đến một lúc nào đó sẽ tăng lên, báo hiệu bắt đầu xảy ra sự quá khớp. Do đó, trong quá trình tối ưu hóa với SGD (hay BGD), nếu thấy "độ lỗi" trên tập "validation" tăng lên thì ta sẽ dừng quá trình tối ưu hóa. Ý tưởng này của chiến lược "dừng sớm" được minh họa ở hình 2.5.

Trong thực tế cài đặt, khi thấy "độ lỗi" trên tập "validation" tăng lên, ta không nên dừng ngay quá trình tối ưu hóa mà nên thực hiện thêm một số vòng lặp nữa rồi mới



Hình 2.5: Minh họa chiến lược "dừng sớm" (early stopping). E_{in} là độ lỗi trên tập huấn luyện, còn E_{out} là độ lỗi trên tập "validation" (ngoài tập huấn luyện).

quyết định dừng hay không (bởi vì có thể "độ lỗi" trên tập "validation" chỉ tăng lên một tí rồi sau đó lại giảm xuống).

Chương 3

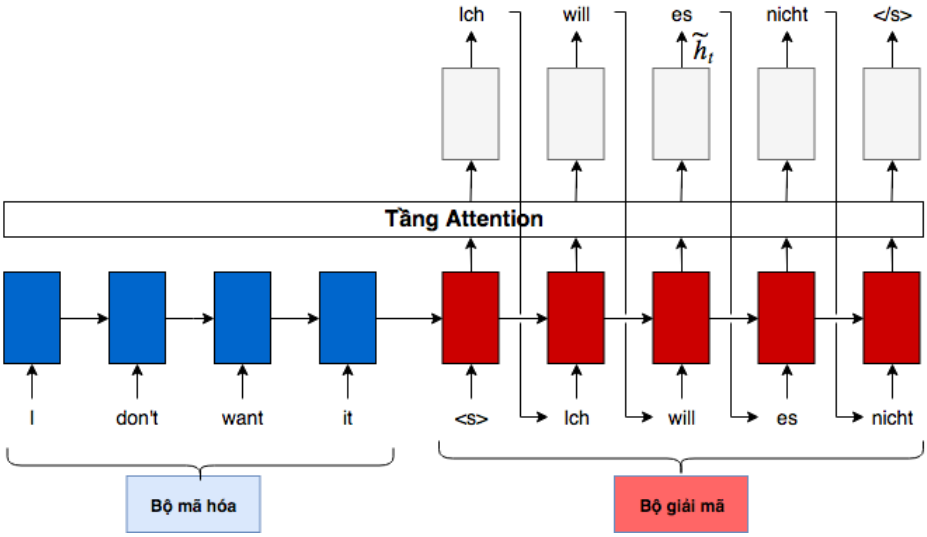
Cơ chế Attention cho mô hình Dịch máy

Chương này trình bày về cơ chế Attention. Ở đây, chúng tôi tập trung tìm hiểu về các phiên bản của cơ chế Attention và đánh giá chúng dựa trên cơ sở Toán học. Cụ thể, chúng tôi tìm hiểu về hai phiên bản Toàn cục (Global) và Cục bộ (Local):

- *Toàn cục: chúng tôi nhận thấy sự hạn chế hiện có của kiến trúc Bộ mã hóa-Bộ mã hóa khi thực hiện dịch những câu dài. Do vậy, chúng tôi sử dụng cơ chế Attention phiên bản Toàn cục để giải quyết vấn đề này.*
- *Cục bộ: chúng tôi quan sát thấy rằng Attention Toàn cục vẫn còn một chút vấn đề về ý tưởng và chi phí tính toán. Với sự quan sát đó, chúng tôi hiệu chỉnh Attention Toàn cục thành phiên bản Attention Cục bộ để giải quyết những hạn chế đó.*

3.1 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



Hình 3.1: Minh họa cơ chế Attention. Một tầng Attention được đặt ở trước bước dự đoán đầu ra của bộ giải mã.

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 [4] [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 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 độ tương đồng giữa 2 trạng thái ẩn. Giá trị của hàm score 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 suy diễn 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:

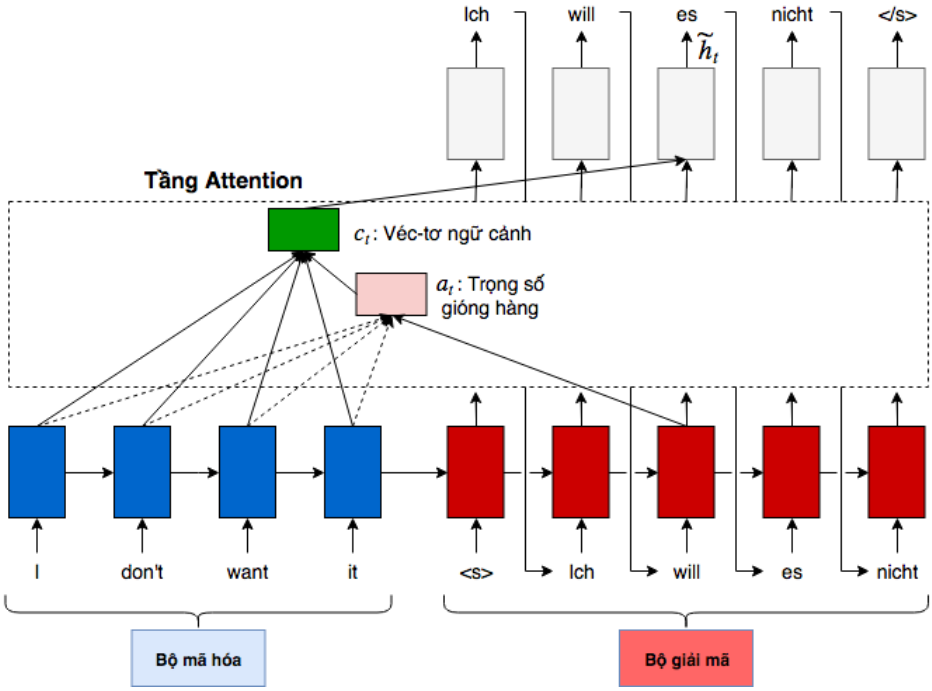
$$\tilde{\boldsymbol{h}}_{\boldsymbol{t}} = \tanh(\boldsymbol{W}_{\boldsymbol{c}}[\boldsymbol{c}_{\boldsymbol{t}}; \boldsymbol{h}_{\boldsymbol{t}}]) \tag{3.4}$$

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 $\boldsymbol{h}_{\boldsymbol{t}}$ 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:

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

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 $\boldsymbol{c}_{\boldsymbol{t}}$, còn các bước còn lại giống nhau. Quy trình tính toán của cơ chế Attention: $h_t -> a_t -> c_t -> \tilde{h}_t$



Hình 3.2: 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.

3.2 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: $\text{len}(a_t) = S$.

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

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, nhưng dễ thấy rằng bản thân mô hình vẫn phải tiêu tốn chi phí cho việc tính toán trọng số gióng hàng a_t cho những vị trí không cần thiết. Đó là trường hợp lý tưởng cho mô hình Attention Toàn cục, nhưng 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. 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.

3.3 Attention Cục bộ

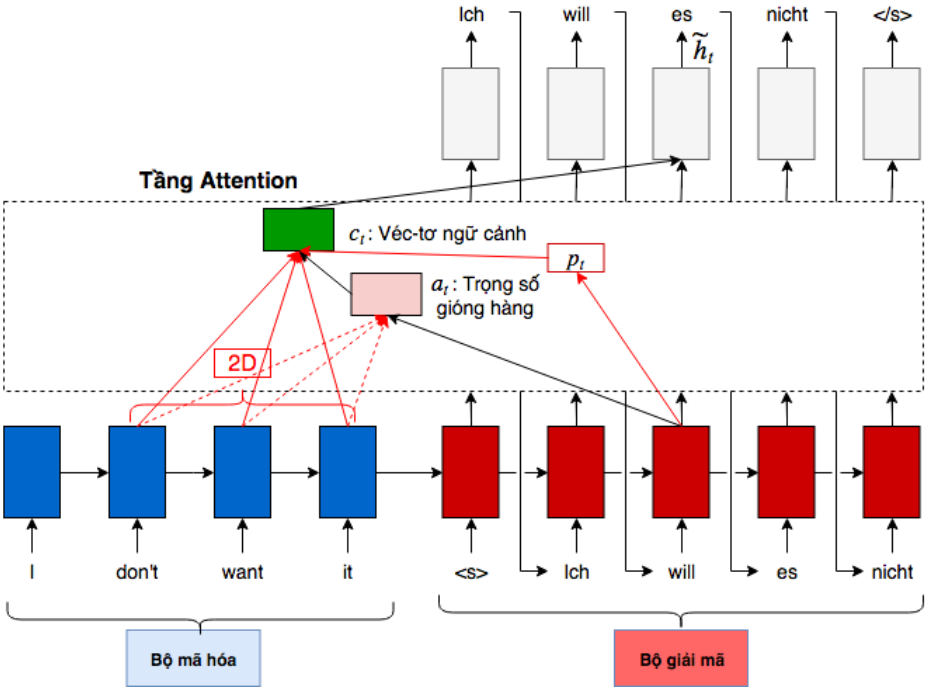
Như đã nêu ở phần trước, Attention Toàn cục có một hạn chế là đặt "sự chú ý" lên toàn bộ các từ ở câu nguồn khi dịch từng từ ở câu đích. Điều này gây tiêu tốn chi phí tính toán và có thể tạo ra những câu dịch không thực tế khi dịch những câu dài như trong các đoạn văn hay trong một tài liệu. Attention Cục bộ ra đời để giải quyết hạn chế này.

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 [11] để giải quyết bài toán Phát sinh câu miêu tả cho ảnh (Image Captioning). Trong công trình [11], 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



Hình 3.3: 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.

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.

- 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 [6], chúng tôi lựa chọn $D = 10$.

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. Chúng tôi sẽ trình bày rõ ràng hơn vào các phần sau. 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 cho việc dự đoán vị trí p_t. Mô hình cần học 2 tham số này. 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 với trung bình 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 gốc $\text{align}(h_t, \bar{h}_s)$. 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

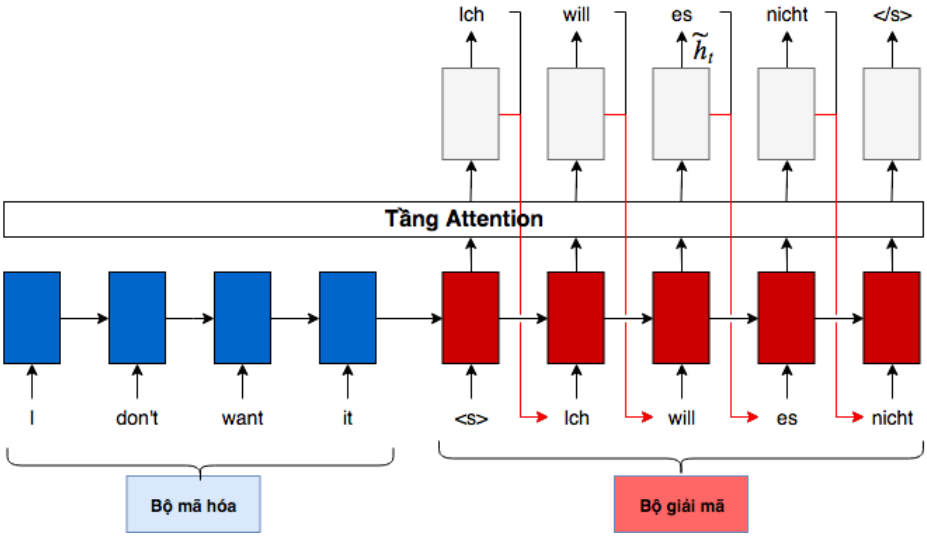
p_t. Với cơ chế được trình bày cụ thể như trên, mô hình Attention Cục bộ hoạt động tự nhiên hơn, phù hợp với cách con người đặt "sự chú ý" khi quan sát sự vật, hiện tượng. Bên cạnh đó, Attention Cục bộ giảm chi phí tính toán của mô hình.

3.4 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.

Trong Dịch máy Thống kê, Koehn et al. 2003 [3] 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 dễ 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 xuất hiện sớm. Ngoài ra còn bị ảnh hưởng bởi số lượng từ quy định khi dịch.

Công trình [6] đề xuất một cơ chế góp phần giải quyết vấn đề ở trên: (tạm dịch là



Hình 3.4: 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.

"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, ràng buộc cụ thể nào mà có thể 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.

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, đồng thời làm tăng khả năng học của mô hình.

3.5 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.

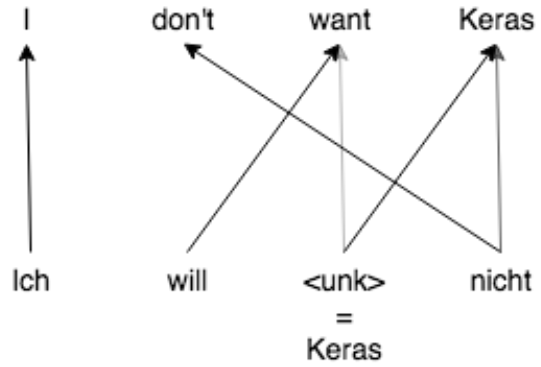
Với tốc độ phát triển của ngôn ngữ là như vậy nhưng khả năng của con người là hữu hạn. Một người dù có thông thạo một ngôn ngữ tới đâu thì cũng không thể nào biết được hết tất cả từ vựng của ngôn ngữ đó. Theo thống kê, số lượng từ ngữ cần để giao tiếp hàng ngày trong tiếng Anh chỉ khoảng từ 2000-3000 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. 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.



Hình 3.5: Minh họa phương pháp thay thế từ hiếm. Khi gặp một từ hiếm (được kí hiệu là <unk>), 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)

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. 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).

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, đối với những câu có độ dài lớn hơn 50 từ, chúng tôi

chỉ lấy 50 từ đầu tiên và bỏ những từ còn lại. 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 của sổ $D = 10$.

Chúng tôi sử dụng ngôn ngữ lập trình Python và framework PyTorch dành cho Học sâu [7]. 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 Titan V. 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 đành giá là BLEU (BiLingual Evaluation Understudy) 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.

Mô hình cơ bản (Baseline) mà chúng tôi sử dụng để so sánh với các mô hình Attention là mô hình với kiến trúc Bộ mã hóa-Bộ giải mã mà không có sử dụng cơ chế Attention.

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

4.2 Kết quả thực nghiệm

Các kết quả của các mô hình được ghi trong bảng 4.1:

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 cơ chế thay thế từ hiếm (unknown replacement - 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. [6] do yếu tố ngẫu nhiên của việc huấn luyện mô hình và do chúng tôi không lọc bỏ những câu có độ dài hơn 50 từ như trong bài báo thực hiện.

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 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.

5.2 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

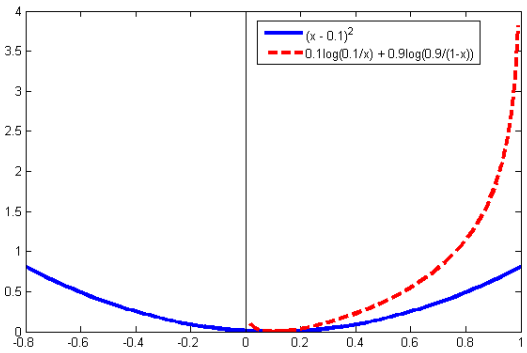


Figure 1: Comparison of KL divergence to squared error with the fixed target ρ of 0.1. When the average output of a hidden neuron is zero, KL divergence gives an infinite penalty while squared error still gives a very small penalty.

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

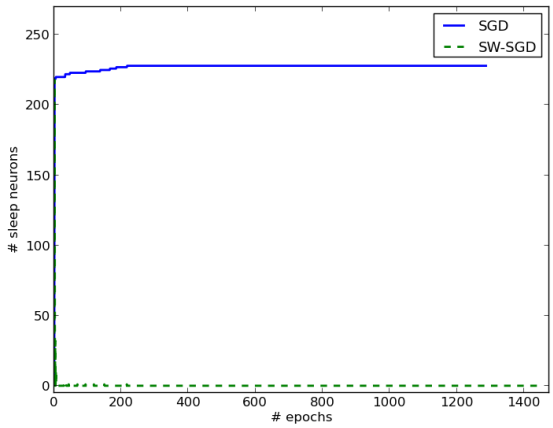


Figure 3: The number of "sleep" hidden neurons of SRAEs during the optimization process with SGD and SW-SGD. The optimization of SGD ended up with 228/1000 "sleep" neurons while SW-SGD ended up without any "sleep" neurons. (These two optimizations terminated after different number of epochs because of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

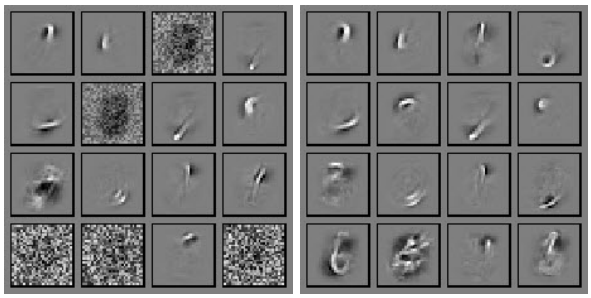
Figure 4 visualizes some example filters (the incoming weight vectors of hidden neurons) learned by SGD and SW-SGD. With SGD, there are five "sleep" filters; they look meaningless. With SW-SGD, there are not any "sleep" filters; all of them look meaningful, like "pen stroke" detectors.

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] 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)*, Jun. 2010, oral Presentation.

[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 23

[3] 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 30

[4] 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 23

[5] H. Lee, A. Battle, R. Raina, and A. Ng, "Efficient sparse coding algorithms," in *Advances in neural information processing systems*, 2006, pp. 801–808. 8

[6] 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 28, 30, 37

[7] 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. 36

[8] 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*, pp. 645–660, 2011.

[9] 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)*, 2011, pp. 833–840.

[10] 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*. ACM, 2008, pp. 1096–1103.

[11] 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 27

[12] M. D. Zeiler, "Hierarchical convolutional deep learning in computer vision," Ph.D. dissertation, New York University, 2014. 9