Chương 1

Giới Thiệu

Trong những năm gần đây, phương pháp học có giám sát của máy học đã được áp dụng để giải quyết nhiều bài toán trong thị giác máy tính, xử lý âm thanh, xử lý ngôn ngữ tự nhiên, ... Thông thường, từ dạng biểu diễn thô ban đầu (ví dụ như ma trận pixel), ta cần phải chuyển sang một dạng biểu diễn tốt hơn, có nhiều ngữ nghĩa hơn – gọi là *biểu diễn đặc trưng* (feature representation); rồi mới đưa dạng biểu diễn đặc trưng này vào một thuật toán học có giám sát (ví dụ như SVM). Việc xác định cách biểu diễn đặc trưng đóng vai trò rất quan trọng để thuật toán học giám sát có thể thực hiện hiệu quả.

Để xác định cách biểu diễn đặc trưng, hướng tiếp cận truyền thống là *thiết kế đặc trưng một cách thủ công* (hand-designed features). Nghĩa là với một lĩnh vực cụ thể, sẽ có những nhóm nghiên cứu gồm các chuyên gia trong lĩnh vực đó cùng ngồi xuống, phân tích, thiết kế, "thử và sai" các cách biểu diễn đặc trưng từ dạng biểu diễn thô ban đầu. Ví dụ về các đặc trưng được thiết kế thủ công trong thị giác máy tính là SIFT, HOG, SURF, ...; trong xử lý âm thanh là Spectrogram, MFCC, Spectral rolloff, ... Tuy nhiên, nhược điểm của hướng tiếp cận này là tốn thời gian và tốn sức lao động; đồng thời cũng thiếu tính tổng quát hóa: các đặc trưng này chỉ sử dụng được cho một loại dữ liệu cụ thể (ví dụ, đặc trưng SIFT chỉ sử dụng được cho một số loại ảnh cụ thể trong thị giác máy tính). Ngoài ra, việc thiết kế đặc trưng một cách thủ công như trên cũng cho thấy điểm yếu của các thuật toán máy học hiện nay: thiếu khả năng tự động rút trích các thông tin có ích trực tiếp từ dữ liệu thô ban đầu.

Do đó, thay vì thiết kế các trưng một cách thủ công, ta mong muốn có một thuật toán có thể *tự động học các đặc trưng* từ dữ liệu thô ban đầu. Hơn nữa, ta mong muốn

Hình 1.1: Minh họa về một biểu diễn đặc trưng tốt: phân tách được các yếu tố giải thích ẩn bên dưới (chó, mèo, cây, ...)

tìm được một thuật toán mà có thể áp dụng tổng quát cho nhiều loại dữ liệu (hình ảnh, âm thanh, . . .). Ngoài ra, ta muốn học các đặc trưng từ tập dữ liệu không có nhãn (unlabeled data) vì dữ liệu không có nhãn có rất nhiều; trong khi đó, dữ liệu có nhãn không có nhiều và phải tốn chi phí để có thể có thêm. Hướng nghiên cứu này được gọi là *học đặc trưng không giám sát* (unsupervised feature learning). Đây là một hướng nghiên cứu mới trong máy học và đang thu hút được rất nhiều sự quan tâm trong thời gian gần đây.

Để giải quyết bài toán học đặc trưng không giám sát, câu hỏi lớn và mang tính định hướng nghiên cứu 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* [1]. Hình 1.1 minh họa cho điểm này; ở đây, chó, mèo, cây, ... là các yếu tố giải thích cho bức ảnh và một biểu diễn đặc trưng tốt cần tìm ra được các yếu tố giải thích này.

Cho đến nay, vẫn chưa có một thuật toán học đặc trưng mà có thể phân tách tốt được các yếu tố giải thích ẩn bên dưới. Để có thể phân tách được các yếu tố giải thích ẩn, ta cần có *các sự hiểu biết trước (priors)* về các yếu tố này. 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, ...). Dưới đây là một số sự hiểu biết trước như vậy [1]:

- **Các yếu tố giải thích được tổ chức một cách phân cấp**: 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 ... *Học sâu (deep learning)* sử dụng sự hiểu biết này: học nhiều tầng biểu diễn đặc trưng với độ trừu tượng tăng dần.

- **Gom cụm tự nhiên (natural clustering)**: 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.

- **Tính thưa (sparsity)**: với mỗi mẫu dữ liệu (ví dụ, với mỗi bức ảnh), chỉ có một số ít các khái niệm (hay các yếu tố giải thích) trong tập các khái niệm. Do đó, với mỗi mẫu dữ liệu, ta muốn tìm một véc-tơ biểu diễn *thưa*, nghĩa là hầu hết các phần tử của véc-tơ này có giá trị bằng 0 (ứng với các khái niệm không liên quan trong tập khái niệm).

Trong luận văn này, chúng tôi sẽ tập trung nghiên cứu về tính thưa. Việc tích hợp tất cả các hiểu biết trước ở trên (cũng như là tìm ra thêm các hiểu biết trước mới) vào trong cùng một mô hình sẽ có thể giúp phân tách các yếu tố giải thích ẩn tốt hơn, nhưng đây là một điều không đơn giản và chúng tôi để lại như là một định hướng cho việc nghiên cứu trong tương lai. Tính thưa lần đầu tiên được đề xuất trong thuật toán "Sparse Coding" [11] để mô hình vùng vỏ não thị giác V1 (là vùng đầu tiên xử lý tín hiệu thị giác từ võng mạc mắt). Và điểm thú vị là "Sparse Coding" có thể học được những đặc trưng tương tự như những đặc trưng của vùng V1 (có dạng các cạnh ở các vị trí khác nhau và với các hướng khác nhau).

"Sparse Auto-Encoders" (SAEs) có thể học được các đặc trưng giống với "Sparse Coding". Tuy nhiên, so với "Sparse Coding", SAEs có những điểm lợi như sau:

- Việc huấn luyện SAEs có thể được thực hiện một cách hiệu quả với thuật toán lan truyền ngược (back-propagation).

- Sau khi đã được huấn luyện, với một véc-tơ đầu vào mới, SAEs có thể tính ra véc-tơ đặc trưng tương ứng rất nhanh; trong khi đó, "Sparse Coding" vẫn phải tiến hành tối ưu hóa.

- Sau khi đã học đặc trưng không giám sát, SAEs có thể cho phép điều chỉnh lại

(fine-tune) các đặc trưng này với các mẫu huấn luyện có nhãn. Nhìn chung, việc điều chỉnh này thường sẽ cho kết quả tốt hơn so với việc không điều chỉnh.

Tuy có những lợi điểm trên, nhưng trong thực tế thì không dễ để làm cho SAEs "hoạt động". Để làm cho SAEs "hoạt động", có hai điểm cần phải làm rõ: (i) ràng buộc thưa, và (ii) ràng buộc trọng số. Sử dụng chuẩn L1 để ràng buộc tính thưa của véc-tơ đặc trưng là một cách tự nhiên (vì L1 được dùng trong "Sparse Coding") và đơn giản (trong trường hợp véc-tơ đặc trưng có giá trị dương, L1 đơn giản là bằng tổng của các phần tử của véc-tơ này), nhưng L1 lại thường không được dùng trong SAEs với lý do vẫn còn chưa rõ ràng [1]. Thay vì dùng L1, các nghiên cứu liên quan đến SAEs thường ràng buộc thưa bằng cách ép giá trị đầu ra trung bình của mỗi nơ-ron ẩn (trong SAEs, mỗi nơ-ron ẩn ứng với một đặc trưng) về một giá trị cố định gần 0 [6][4][3]. Tuy nhiên, giá trị cố định này lại thêm một siêu tham số (hyper-parameter, là tham số mà ta phải chọn trước khi huấn luyện) vào danh sách các siêu tham số vốn đã rất nhiều của SAEs; điều này sẽ làm cho quá trình chọn lựa các siêu tham số trở nên rất "phiền phức" và tốn thời gian. Về vấn đề ràng buộc trọng số của SAEs, có một số cách khác nhau đã được sử dụng. [3] ràng buộc các trọng số của bộ mã hóa (encoder) giống với các trọng số của bộ giải mã (decoder). Cách ràng buộc trọng số này cũng được dùng cho các loại "Auto-Encoders" khác như "Denoising Auto-Encoders" [14] và "Contractive Auto-Encoders" [13][12]. [6][4] dùng một cách ràng buộc trọng số khác là "weight decay" (phạt tổng bình phương các trọng số); cách này lại làm xuất hiện thêm một siêu tham số nữa. [16] ràng buộc các véc-tơ trọng số của tầng giải mã (mỗi véc-tơ ứng với các trọng số đi ra ở mỗi nơ-ron ẩn) có độ dài bằng một. Tuy nhiên, trong số các cách ràng buộc trọng số này, không rõ là nên sử dụng cách nào cũng như là tại sao nên ràng buộc các trọng số như vậy.

Như vậy, có hai câu hỏi cần phải được trả lời: (i) tại sao chuẩn L1 lại thường không được dùng để ràng buộc thưa trong SAEs?; (ii) liệu có cách nào tốt hơn và hợp lý hơn để ràng buộc trọng số của SAEs không? Trong luận văn này, chúng tôi sẽ cố gắng trả lời hai câu hỏi này. Cụ thể là:

- Chúng tôi cố gắng hiểu khó khăn của việc huấn luyện SAE với ràng buộc thưa dùng chuẩn L1. Từ đó, chúng tôi đề xuất một phiên bản điều chỉnh của thuật toán tối ưu hóa "Stochastic Gradient Descent" (SGD), gọi là "Sleep-Wake Stochastic Gradient Descent" (SW-SGD), để giải quyết khó khăn này. Ở đây, chúng tôi tập

trung nghiên cứu SAEs với hàm kích hoạt ở tầng ẩn là hàm "rectified linear" ($f(x) = \max(0, x)$) bởi vì hàm này tính nhanh và cho tính thưa thật sự (đúng bằng 0). Chúng tôi gọi SAEs với hàm kích hoạt này là *"Sparse Rectified Auto-Encoders"* (SRAEs).

- Hơn nữa, chúng tôi cũng đề xuất một cách hợp lý để ràng buộc trọng số của SRAEs.

Với hai thành phần trên (SW-SGD và cách ràng buộc trọng số mà chúng tôi đề xuất), kết quả thí nghiệm trên bộ dữ liệu MNIST (bộ dữ liệu chữ số viết tay từ 0 đến 9) cho thấy SRAEs có thể học được những đặc trưng có ích và những đặc trưng này cho kết quả phân lớp tốt khi so sánh với các loại "Auto-Encoders" khác.

Phần còn lại của luận văn được trình bày như sau:

- Chương 2 trình bày kiến thức nền tảng về "Sparse Coding" và "Sparse Auto-Encoders".
- Chương 3 trình bày về "Sparse Rectified Auto-Encoders" (SRAEs); đây là phần chính của luận văn. Trong phần này gồm có hai phần nhỏ:
  - Ràng buộc thưa: chúng tôi giải thích về vấn đề gặp phải khi huấn luyện SRAEs với chuẩn L1 và đưa ra giải pháp để giải quyết vấn đề này.
  - Ràng buộc trọng số: chúng tôi trình bày về cách ràng buộc trọng số đề xuất cho SRAEs.
- Chương 4 trình bày về các thí nghiệm và các phân tích về kết quả đạt được.
- Cuối cùng, kết luận và hướng phát triển được trình bày ở chương 5.

Chương 2

Kiến Thức Nền Tảng

Trong chương này, đầu tiên chúng tôi trình bày về thuật toán học đặc trưng không giám sát "Sparse Coding". Sau đó, chúng tôi trình bày về "Sparse Auto-Encoders" (SAEs) và so sánh với "Sparse Coding" để thấy được sự kết nối giữa SAEs và "Sparse Coding" cũng như là những điểm lợi của SAEs so với "Sparse Coding"; những điểm lợi này là lý do để chúng tôi tập trung nghiên cứu SAEs. Ngoài ra, chúng tôi cũng trình bày về "Softmax Regression" - mô hình phân lớp mà chúng tôi sẽ sử dụng để đánh giá các đặc trưng học được, và thuật toán "Stochastic Gradient Descent" - thuật toán mà chúng tôi sẽ sử dụng để cực tiểu hóa hàm chi phí của SAEs cũng như của "Softmax Regression". Chương này, đặc biệt là phần về "Sparse Coding" và SAEs, cung cấp những kiến thức nền tảng để có thể hiểu rõ về những đề xuất của chúng tôi ở chương kế tiếp.

2.1 "Sparse Coding"

"Sparse Coding" được đề xuất lần đầu tiên trong lĩnh vực khoa học nơ-ron (neuroscience) để mô hình vùng vỏ não thị giác V1 [11]. "Sparse Coding" sử dụng *sự hiểu biết trước (prior) về tính thưa của các yếu tố giải thích ẩn* (với mỗi mẫu dữ liệu quan sát được, chỉ có một số ít các yếu tố giải thích trong tập lớn các yếu tố giải thích) để phân tách chúng. Mục tiêu của "Sparse Coding" là tìm ra tập các véc-tơ cơ sở (tập các yếu tố giải thích) sao cho mỗi mẫu dữ liệu có thể được "giải thích" bởi một số ít các véc-tơ cơ sở (chỉ có một số ít các véc-tơ cơ sở có hệ số khác 0, hay nói một cách khác,

véc-tơ hệ số "thưa"). Lưu ý là số chiều của không gian đặc trưng (số lượng véc-tơ cơ sở tìm được) có thể lớn hơn số chiều của không gian ban đầu; tính chất này được gọi là "over-complete".

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 $N_x \times N_h$ (N_x là số chiều của không gian ban đầu, N_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 $N_h \times 1$. 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 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 $||Wh - x||_2^2$).

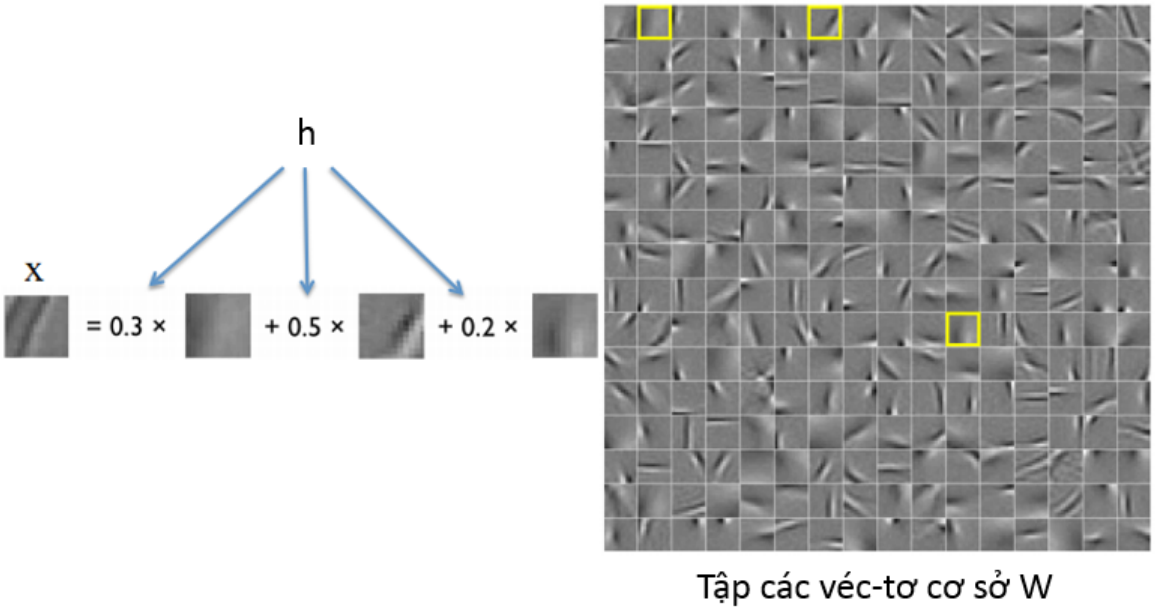
- Thưa (bằng cách tối thiểu hóa chuẩn L1 $||h||_1$).

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 [9]. 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 [15]. 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 $f(\cdot)$.

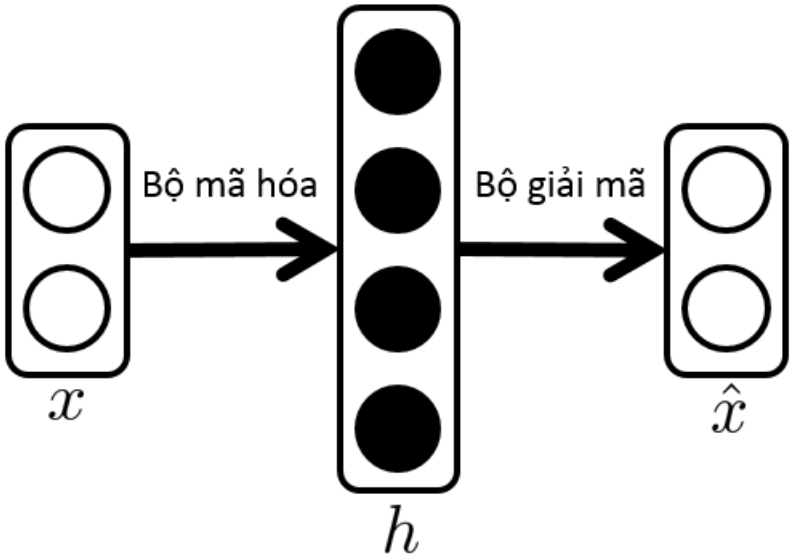
- Phần thứ hai, được gọi là *bộ giải mã* (decoder), cố gắng tái tạo lại véc-tơ đầu vào x ban đầu từ véc-tơ biểu diễn ẩn h:

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

  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):

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

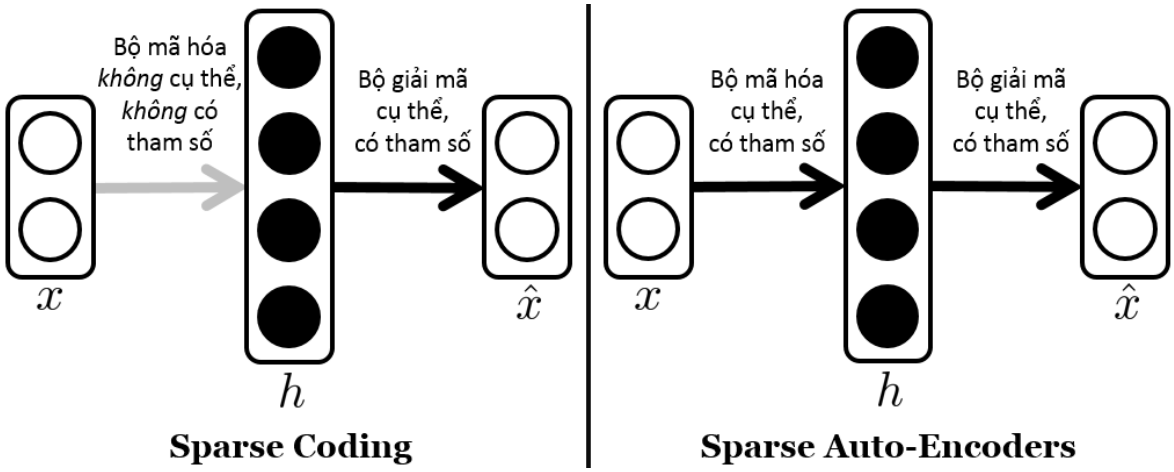
Trong đó, $\hat{x}$ là véc-tơ tái tạo (với $\hat{x} = W^{(d)}h + b^{(d)}$ và $h = f(W^{(e)}x + b^{(e)})$); $s(\cdot)$ là một hàm nào đó mà làm cho véc-tơ biểu diễn ẩn h thưa (ví dụ, $s(\cdot)$ có thể là chuẩn L1 như ở "Sparse Coding"); và λ là siêu tham số điều khiển "sự thỏa hiệp" giữa độ lỗi tái tạo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

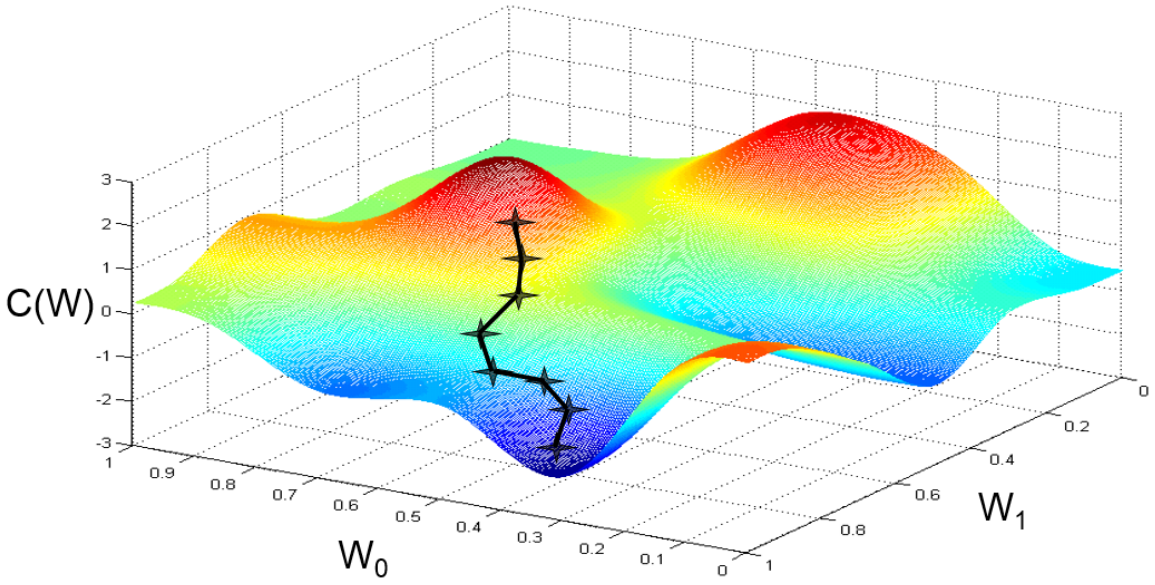
$$C(W) = \frac{1}{N}\sum_{i=1}^{N} C^{(i)}(W) \tag{2.8}$$

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.

Từng bước của thuật toán BGD như sau:

1. Khởi tạo ngẫu nhiên cho W.
2. Lặp cho đến khi thỏa điều kiện dừng:

$$W = W - \alpha \nabla C(W)$$

($\alpha > 0$ là hệ số học)

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.

Từng bước của thuật toán SGD như sau:

1. Khởi tạo ngẫu nhiên cho W.

2. Lặp cho đến khi thỏa điều kiện dừng:

- Xáo trộn ngẫu nhiên thứ tự của các mẫu trong tập huấn luyện (thường sẽ giúp SGD hội tụ nhanh hơn).
- Với $b = 1, 2, \ldots, \frac{N}{B}$ (N là số lượng mẫu trong tập huấn luyện, B là kích thước của "mini-batch"), cập nhật W theo công thức:

$$W = W - \alpha \frac{1}{B} \sum_{i=(b-1)B+1}^{bB} \nabla C^{(i)}(W)$$

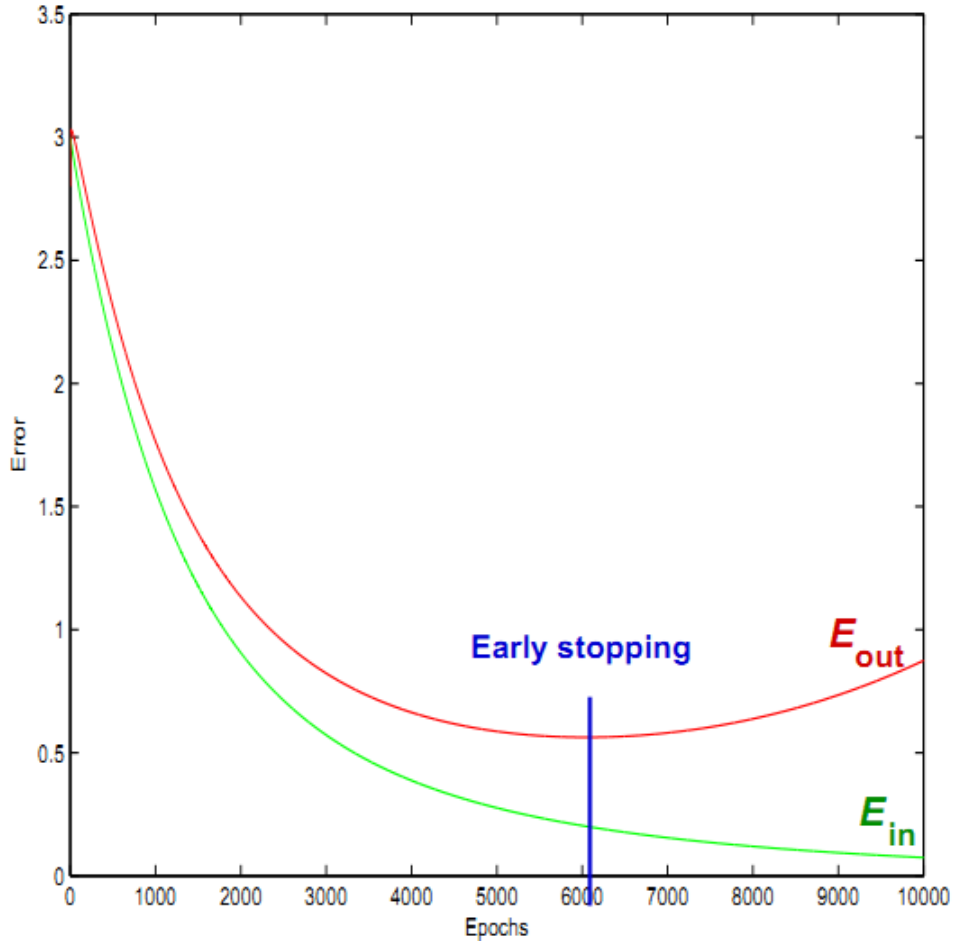
($\alpha > 0$ là hệ số học)

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

Sparse Rectified Auto-Encoders

Chương này trình bày về những đóng góp của luận văn. Ở đây, chúng tôi tập trung nghiên cứu "Sparse Auto-Encoders" với hàm kích hoạt "rectified linear" ($f(x) = max(0,x)$) ở tầng ẩn vì hàm này tính nhanh và có thể cho tính thưa thật sự (đúng bằng 0). Chúng tôi gọi "Sparse Auto-Encoders" với hàm kích hoạt như vậy là "Sparse Rectified Auto-Encoders" (SRAEs). Đóng góp của chúng tôi là làm rõ SRAEs ở hai điểm:

- *Ràng buộc thưa: chúng tôi cố gắng hiểu khó khăn của việc huấn luyện SRAEs khi dùng chuẩn L1 để ràng buộc thưa; từ đó, chúng tôi đề xuất một phiên bản hiệu chỉnh của thuật toán "Stochastic Gradient Descent" (SGD), gọi là "Sleep-Wake Stochastic Gradient Descent" (SW-SGD), để giải quyết khó khăn này.*
- *Ràng buộc trọng số: chúng tôi cũng đưa ra một cách ràng buộc trọng số hợp lý cho SRAEs.*

3.1 "Sparse Rectified Auto-Encoders" (SRAEs)

Các hàm kích hoạt thường được sử dụng trong mạng nơ-ron là:

- hàm sigmoid: $f(x) = \frac{1}{1+e^{-x}}$
- và hàm tanh: $f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$

Gần đây, cộng đồng nghiên cứu mạng nơ-ron phát hiện ra một hàm kích hoạt mới hoạt động rất tốt là hàm "rectified linear" [10][5][16]: $f(x) = max(0,x)$.

Hàm kích hoạt "rectified linear" rất phù hợp với "Sparse Auto-Encoders" (SAEs) bởi vì hàm này vốn dĩ đã tạo ra véc-tơ đặc trưng thưa (với một véc-tơ đầu vào, hàm "rectified linear" vốn dĩ đã tạo ra một véc-tơ đặc trưng với khoảng 50% số phần tử bằng 0). Khác với hàm sigmoid là khi véc-tơ đầu vào không chứa đặc trưng tương tự với đặc trưng của bộ lọc (ở đây bộ lọc ám chỉ véc-tơ gồm các trọng số đi vào một nơ-ron ẩn), hàm "rectified linear" thường sẽ cho giá trị đúng bằng 0; trong khi đó, hàm sigmoid thường vẫn cho một giá trị dương nhỏ. Ngoài ra, hàm "rectified linear" tính nhanh hơn hàm sigmoid và hàm tanh bởi vì hàm này chỉ phải thực hiện phép max chứ không phải thực hiện phép lũy thừa và phép chia như ở hai hàm kia. Cuối cùng, hàm "rectified linear" có tiềm năng để có thể huấn luyện đồng thời nhiều tầng biểu diễn đặc trưng một cách không giám sát (thay vì phải huấn luyện từng tầng biểu diễn đặc trưng một) bởi vì hàm này đã được dùng để huấn luyện thành công nhiều tầng biểu diễn đặc trưng trong ngữ cảnh có giám sát [5][16]. Với những điểm trên, trong luận văn này, chúng tôi sẽ tập trung nghiên cứu SAEs với hàm kích hoạt ở tầng ẩn là hàm "rectified linear". Chúng tôi gọi SAEs với hàm kích hoạt như vậy là "Sparse Rectified Auto-Encoders" (SRAEs).

3.2 Ràng buộc thưa trong SRAEs

Cách thường được dùng để ràng buộc thưa trong "Sparse Auto-Encoders" (SAEs) là ép giá trị đầu ra trung bình $\bar{h}_j$ của nơ-ron ẩn thứ j về một giá trị cố định gần không ρ [6][4][3] (giá trị đầu ra trung bình này được tính với toàn bộ các mẫu huấn luyện, hoặc nếu dùng "mini-batch" thì sẽ được tính với các mẫu huấn luyện trong một "mini-batch" nhưng kích thước của "mini-batch" cần phải tương đối lớn). Trong trường hợp giá trị đầu ra của nơ-ron ẩn thuộc $[0,1]$ (ví dụ, dùng hàm kích hoạt sigmoid), việc ép thưa có thể được thực hiện bằng cách cực tiểu hóa sự sai biệt Kullback-Leibler (KL):

$$\sum_j KL(\rho||\bar{h}_j) = \sum_j \left(\rho \log \frac{\rho}{\bar{h}_j} + (1-\rho) \log \frac{(1-\rho)}{(1-\bar{h}_j)} \right) \tag{3.1}$$

Khi $\bar{h}_j$ càng gần với ρ thì hàm $KL(\rho||\bar{h}_j)$ sẽ có giá trị càng nhỏ và sẽ đạt cực tiểu (bằng 0) khi $\bar{h}_j = \rho$; khi $\bar{h}_j$ càng xa với ρ ($\bar{h}_j$ tiến về phía 0 hoặc phía 1) thì hàm $KL(\rho||\bar{h}_j)$ sẽ có giá trị càng lớn và tiến tới ∞ khi $\bar{h}_j$ tiến tới 0 hoặc tiến tới 1. TODO:

hình minh họa. Trong trường hợp dùng hàm kích hoạt "rectified linear", ta có thể dùng độ lỗi bình phương để ép $\bar{h}_j$ về ρ:

$$\sum_j (\bar{h}_j - \rho)^2 \tag{3.2}$$

Lưu ý là cách ràng buộc thưa này (ép các giá trị đầu ra trung bình của các nơ-ron ẩn về một giá trị cố định gần không) không trực tiếp làm thưa véc-tơ đặc trưng (véc-tơ gồm các giá trị đầu ra ở tầng ẩn khi đưa vào SAEs một véc-tơ đầu vào), mà làm thưa véc-tơ chứa các giá trị của một đặc trưng (véc-tơ gồm các giá trị đầu ra của một nơ-ron ẩn khi đưa vào SAEs các véc-tơ đầu vào khác nhau). Tuy nhiên, cách ràng buộc này làm thưa véc-tơ đặc trưng một cách gián tiếp. Để hình dung rõ hơn về điểm này, ta xét một ví dụ đơn giản sau. Giả sử tập huấn luyện của ta gồm có 5 mẫu huấn luyện và SAEs của ta gồm có 3 nơ-ron ẩn (ứng với 3 đặc trưng). Như vậy ta sẽ có ma trận đặc trưng có kích thước 3×5, trong đó mỗi cột là một véc-tơ đặc trưng ứng với một mẫu huấn luyện (một véc-tơ đầu vào), mỗi dòng là một véc-tơ gồm 5 giá trị của một đặc trưng ứng với 5 mẫu huấn luyện. Cách ràng buộc thưa ở trên sẽ làm thưa các véc-tơ dòng của ma trận này; giả sử mỗi véc-tơ dòng này chỉ có một phần tử khác không và bốn phần tử còn lại bằng không. Bởi vì từ mỗi véc-tơ cột ta cần phải tái tạo lại véc-tơ đầu vào tương ứng, nên ta cần phân bố các phân tử khác 0 trải đều trên toàn các véc-tơ cột, cố gắng sao cho mỗi véc-tơ cột đều có phần tử khác 0 (nếu ta tập trung các phần tử khác 0 trên cùng một véc-tơ cột thì chỉ có một véc-tơ đầu vào tương ứng được tái tạo tốt, các véc-tơ đầu vào còn lại sẽ không được tái tạo tốt). Và điều này dẫn đến các véc-tơ cột cũng sẽ thưa.

Tuy nhiên, cách ràng buộc thưa như trên đưa thêm một siêu tham số (giá trị đầu ra trung bình mong muốn ρ) vào danh sách các siêu tham số vốn đã có rất nhiều của SAEs (hệ số "thỏa hiệp" giữa độ lỗi tái tạo và độ thưa, số lượng node ẩn, hệ số học, kích thước "mini-batch", ...). Điều này sẽ làm cho quá trình chọn lựa các siêu tham số trở nên "phiền phức" hơn và tốn thời gian hơn.

Tại sao lại không dùng chuẩn L1 để ràng buộc thưa trong SAEs? Đây là một cách tự nhiên vì chuẩn L1 đã được dùng để ràng buộc thưa trong Sparse Coding. Hơn nữa, chuẩn L1 không đưa thêm siêu tham số nào. Ngoài ra, cách tính chuẩn L1 cũng rất đơn giản; trong trường hợp dùng hàm kích hoạt "rectified linear", chuẩn L1 của véc-

tơ đặc trưng h đơn giản là bằng tổng giá trị các phần tử của h. Trong phần dưới đây, chúng tôi sẽ giải thích về khó khăn gặp phải khi huấn luyện SAEs, cụ thể là SRAEs, với chuẩn L1.

3.2.1 Khó khăn của việc huấn luyện SRAEs với chuẩn L1

Vấn đề gặp phải khi huấn luyện SRAEs với chuẩn L1 là trong quá trình tối ưu hóa hàm chi phí, chuẩn L1 có thể đẩy véc-tơ gồm các trọng số đi vào một nơ-ron ẩn vào trạng thái mà ở đó nơ-ron ẩn luôn luôn không kích hoạt (có giá trị đầu ra bằng 0 với tất cả các mẫu trong tập huấn luyện). Và một khi véc-tơ trọng số đi vào này đã rơi vào trạng thái nói trên, nó sẽ bị mắc kẹt ở đó mãi mãi và không bao giờ được cập nhật nữa; véc-tơ trọng số đi ra của nơ-ron ẩn này cũng không bao giờ được cập nhật nữa. Một cách cụ thể, xét một nơ-ron ẩn j: có trọng số $W^{(e)}_{ji}$ nối với nơ-ron đầu vào i, và có trọng số $W^{(d)}_{kj}$ nối với nơ-ron đầu ra k. Các đạo hàm riêng của hàm chi phí C (hàm chi phí của một mẫu huấn luyện) ở công thức (2.4) (với hàm ép thưa $s(\cdot) = ||\cdot||_1$) theo $W^{(e)}_{ji}$ và $W^{(d)}_{kj}$ có thể được tính bằng thuật toán lan truyền ngược như sau:

$$\frac{\partial C}{\partial W^{(d)}_{kj}} = 2(\hat{x}_k - x_k)h_j \tag{3.3}$$

$$\frac{\partial C}{\partial W^{(e)}_{ji}} = (\lambda + \sum_{k'} W^{(d)}_{k'j} \frac{\partial C}{\partial \hat{x}_{k'}}) f'(a_j) x_i \tag{3.4}$$

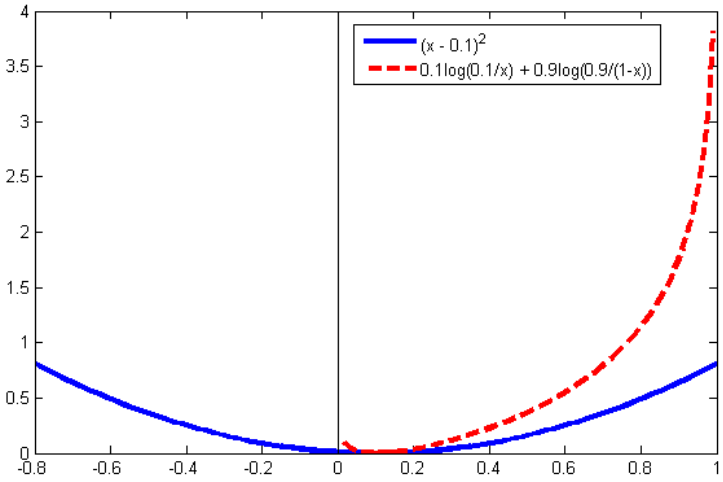
Trong đó:

- x_k và $\hat{x}_k$ lần lượt là phần tử thứ k của véc-tơ đầu vào x và của véc-tơ tái tạo $\hat{x}$.
- h_j là phần tử thứ j của véc-tơ đặc trưng h (véc-tơ đầu ra ở tầng ẩn).
- f' là đạo hàm của hàm kích hoạt tại nơ-ron ẩn và a_j là giá trị trước khi áp dụng hàm kích hoạt ở nơ-ron ẩn j ($a_j = W^{(e)}_{j.} x + b^{(e)}$ với $W^{(e)}_{j.}$ là véc-tơ dòng thứ j của ma trận $W^{(e)}$).
- k là chỉ số của nơ-ron đầu ra đang xét, còn k' là chỉ số chạy dùng để duyệt hết tất cả các nơ-ron đầu ra.

Từ công thức (3.3) và (3.4), ta có thể dễ thấy rằng, trong quá trình tối ưu hóa hàm chi phí, nếu một khi nơ-ron ẩn j đã rơi vào trạng thái có giá trị đầu ra h_j bằng 0 đối với tất cả các mẫu huấn luyện thì các đạo hàm riêng $\frac{\partial C}{\partial W_{kj}^{(d)}}$ và $\frac{\partial C}{\partial W_{ji}^{(e)}}$ sẽ có giá trị bằng 0 đối với tất cả các mẫu huấn luyện ($\frac{\partial C}{\partial W_{kj}^{(d)}} = 0$ vì $h_j = 0$, và $\frac{\partial C}{\partial W_{ji}^{(e)}} = 0$ vì khi $h_j = f(a_j) = 0$ thì $f'(a_j) = 0$ trong trường hợp $f(\cdot)$ là hàm "rectified linear"); và do đó, các trọng số của nơ-ron ẩn này sẽ không bao giờ được cập nhật nữa (trong trường hợp sử dụng hàm kích hoạt sigmoid, h_j thường không đúng bằng 0 mà là một giá trị dương rất nhỏ gần 0 và đạo hàm tương ứng $f'(a_j)$ cũng không đúng bằng 0 mà là một giá trị dương rất nhỏ gần 0; do đó, sự cập nhật trọng số vẫn có thể xảy ra nhưng sẽ rất chậm). Chúng tôi gọi những nơ-ron ẩn như vậy là những *nơ-ron "ngủ"*. Đặc biệt, tính chất "dễ cho giá trị 0" của hàm "rectified linear" làm cho vấn đề này dễ xảy ra hơn (so với hàm sigmoid) trong quá trình tối ưu hóa.

Vấn đề nơ-ron "ngủ" nêu trên có thể giúp lý giải cho việc tại sao các nghiên cứu lại thường không dùng chuẩn L1 trong SAEs mà thay vào đó là ép giá trị đầu ra trung bình của một nơ-ron ẩn về một giá trị cố định gần 0 (nhưng không bằng 0!); cách làm này có thể giúp ngăn nơ-ron ẩn rơi vào tình trạng "không kích hoạt" với tất cả các mẫu huyện và sau đó các trọng số của nó không thể được cập nhật nữa. Với hàm kích hoạt sigmoid, việc ép giá trị đầu ra trung bình của một nơ-ron ẩn về một giá trị cố định gần 0 có thể được thực hiện bằng cách sử dụng hàm sai biệt KL như ở công thức (3.1), và nơ-ron ẩn này sẽ không thể "ngủ" bởi vì nếu như vậy thì giá trị đầu ra trung bình sẽ bằng 0 và hàm sai biệt KL sẽ cho giá trị phạt là ∞. Với hàm kích hoạt "rectified linear", ta không thể sử dụng hàm sai biệt KL bởi vì miền giá trị đầu ra của hàm kích hoạt này không thuộc $[0, 1]$. Hàm lỗi bình phương như ở công thức (3.2) có thể được dùng thay thế, nhưng thí nghiệm của chúng tôi cho thấy rằng vấn đề nơ-ron "ngủ" vẫn xảy ra. Đó là vì khi giá trị đầu ra trung bình bằng 0, khác với hàm sai biệt KL, hàm lỗi bình phương chỉ cho một giá trị phạt rất nhỏ. Hình 3.1 so sánh hai hàm này với giá trị đầu ra trung bình mong muốn $\rho = 0.1$.

Mặc dù "Sparse Coding"sử dụng chuẩn L1 để ràng buộc thưa, nhưng ta thấy rõ ràng là "Sparse Coding" sẽ không mắc phải vấn đề nơ-ron ngủ ở trên vì "Sparse Coding" không có bộ mã hóa cụ thể như SAEs.



Hình 3.1: So sánh giữa hàm sai biệt KL và hàm lỗi bình phương với giá trị đầu ra trung bình mong muốn $\rho = 0.1$. Khi giá trị đầu ra trung bình của một nơ-ron ẩn bằng 0, hàm sai biệt KL cho giá trị phạt bằng ∞, trong khi đó hàm lỗi bình phương chỉ cho một giá trị phạt rất nhỏ.

3.2.2 Thuật toán "Sleep-Wake Stochastic Gradient Descent"

Để khắc phục khó khăn của việc huấn luyện SRAEs với chuẩn L1, chúng tôi đề xuất một phiên bản điều chỉnh của thuật toán "Stochastic Gradient Descent" (SGD), gọi là "Sleep-Wake Stochastic Gradient Descent" (SW-SGD). Ý tưởng là trong mỗi "epoch" của SGD (một "epoch" ứng một lần duyệt qua tất cả các mẫu trong tập huấn luyện), ta tính tổng giá trị đầu ra của mỗi nơ-ron ẩn; và sau mỗi "epoch", ta kiểm xem có nơ-ron nào "ngủ" không (có tổng trị đầu ra bằng không) và "đánh thức" chúng bằng cách khởi tạo lại véc-tơ trọng số đi vào. Mặc dù cách làm này rất đơn giản, nhưng thí nghiệm của chúng tôi cho thấy nó có thể giúp SRAEs học được thành công các đặc trưng mà không có đặc trưng nào "ngủ".

Một cách cụ thể, cho tập huấn luyện không có nhãn $\{x^{(1)}, x^{(2)}, \ldots, x^{(N)}\}$, thuật toán SW-SGD dùng để cực tiểu hóa hàm chi phí sau của SRAEs:

$$\begin{aligned} C(W) &= \frac{1}{N}\sum_{i=1}^{N} C^{(i)}(W) \\ &= \frac{1}{N}\sum_{i=1}^{N}\left(\|x^{(i)} - \hat{x}^{(i)}\|_2^2 + \lambda\|h^{(i)}\|_1\right) \end{aligned} \tag{3.5}$$

Trong đó:

- $h^{(i)}$ là véc-tơ đầu ra ở tầng ẩn tương ứng với véc-tơ đầu vào $x^{(i)}$:

$$h^{(i)} = \max\left(0, W^{(e)}x^{(i)} + b^{(e)}\right)$$

- $\hat{x}^{(i)}$ là véc-tơ tái tạo của véc-tơ đầu vào $x^{(i)}$:

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

- $W = \{W^{(e)}, b^{(e)}, W^{(d)}, b^{(d)}\}$ là các tham số của SRAEs.

Từng bước của thuật toán SW-SGD như sau (những chỗ thay đổi so với thuật toán SGD ban đầu được *in nghiêng*):

1. Khởi tạo ngẫu nhiên cho W.
2. Lặp cho đến khi thỏa điều kiện dừng:
   - *Khởi tạo véc-tơ s gồm có D_h phần tử (với D_h là số lượng nơ-ron ẩn của SRAEs), trong đó mỗi phần tử có giá trị bằng* 0 *(phần tử s_j của véc-tơ s dùng để lưu tổng giá trị đầu ra của nơ-ron ẩn j với tất cả các mẫu trong tập huấn luyện sau một "epoch").*
   - Xáo trộn ngẫu nhiên thứ tự của các mẫu trong tập huấn luyện (thường sẽ giúp hội tụ nhanh hơn).
   - Với "mini-batch" thứ $b = 1, 2, \ldots, \frac{N}{B}$ (N là số lượng mẫu trong tập huấn luyện, B là kích thước của "mini-batch"):
     - Với mẫu huấn luyện thứ $i = (b-1)B+1, (b-1)B+2, \ldots, bB$:
       * Lan truyền tiến với véc-tơ đầu vào $x^{(i)}$.
       * *Cập nhật véc-tơ s: $s = s + h^{(i)}$ (với $h^{(i)}$ là véc-tơ đầu ra ở tầng ẩn tương ứng với véc-tơ đầu vào $x^{(i)}$).*
       * Lan truyền ngược và tính véc-tơ đạo hàm riêng $\nabla C^{(i)}(W)$.

– Cập nhật W:

$$W = W - \alpha \frac{1}{B} \sum_{i=(b-1)B+1}^{bB} \nabla C^{(i)}(W)$$

($\alpha > 0$ là hệ số học)

- *Kiểm xem có nơ-ron ẩn nào "ngủ" (có tổng đầu ra $s_j = 0$) và "đánh thức" bằng cách khởi tạo lại véc-tơ trọng số đi vào nơ-ron ẩn này.*

3.3 Ràng buộc trọng số trong SRAEs

Bên cạnh ràng buộc thưa, ràng buộc trọng số cũng là một thành phần quan trọng để làm cho SAEs "hoạt động". Tại sao cần phải ràng buộc trọng số? Ví dụ, trong Sparse Coding, ta cần phải ràng buộc các véc-tơ cơ sở được chuẩn hóa (có chiều dài bằng 1); nếu không thì sẽ xảy ra trường hợp là giá trị của hàm chi phí ở công thức (2.1) có thể được làm giảm xuống một cách "tầm thường" bằng cách chia hệ số cho một số lớn tùy ý và nhân véc-tơ cơ sở tương ứng với cùng số lớn đó (làm như vậy sẽ làm độ thưa giảm xuống tùy ý, còn độ lỗi tái tạo thì giữ nguyên). Các véc-tơ cơ cở trong "Sparse Coding" tương ứng với các cột của ma trận trọng số $W^{(d)}$ của bộ giải mã của SAEs (mỗi cột của $W^{(d)}$ ứng với véc-tơ trọng số đi ra tại mỗi nơ-ron ẩn). Như vậy, trong SAEs, ta cũng có thể ràng buộc mỗi cột của $W^{(d)}$ được chuẩn hóa (có chiều dài bằng 1) giống như ở Sparse Coding. Nhưng còn ma trận trọng số $W^{(e)}$ của bộ mã hóa của SAEs? Ta nên ràng buộc $W^{(e)}$ như thế nào cho hợp lý?

Dưới đây là một số cách đã được đề xuất để ràng buộc trọng số của SAEs:

- **Ràng buộc** $W^{(d)} = (W^{(e)})^T$: bộ trọng số được dùng chung cho cả bộ mã hóa và bộ giải mã (cụ thể là $W^{(d)}$ và $W^{(e)}$ là chuyển vị của nhau) [3]. Cách ràng buộc trọng số này cũng được dùng trong các loại "Auto-Encoders" khác như "Denoising Auto-Encoders" và "Contractive Auto-Encoders" [14][13][12]. Lưu ý là tất cả [3][14][13][12] đều dùng hàm kích hoạt sigmoid ở tầng ẩn. Trong trường hợp dùng hàm kích hoạt tuyến tính ($f(x) = x$) ở tầng ẩn, ràng buộc $W^{(d)} = (W^{(e)})^T$ sẽ có xu hướng làm cho các véc-tơ cơ sở (các dòng của $W^{(e)}$

hay các cột của $W^{(d)}$) trực giao với nhau và được chuẩn hóa [7]; nhưng trong trường hợp dùng hàm kích hoạt sigmoid ở tầng ẩn, ta không rõ chuyện gì đang xảy ra. Một điểm lợi của việc dùng chung bộ trọng số là tiết kiệm bộ nhớ lưu trữ; điều này sẽ có ích khi cài đặt song song trên GPU (Graphical Processing Units).

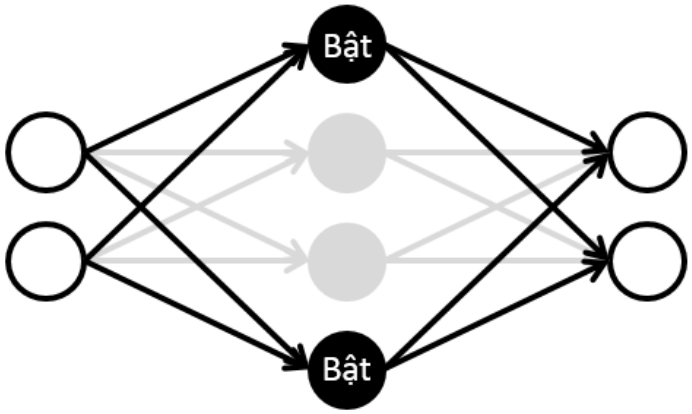
- **Ràng buộc $W^{(d)}$ được chuẩn hóa**: các cột của $W^{(d)}$ được ràng buộc là có độ dài bằng 1 [16]. Ràng buộc này tương tự như ở "Sparse Coding" và giúp ngăn chặn việc hàm chi phí có thể bị làm giảm xuống một cách "tầm thường" như đã nói ở trên. Nhưng còn bộ trọng số $W^{(e)}$ của bộ mã hóa? Chẳng hạn, để công bằng giữa các đặc trưng, ta cũng nên ràng buộc các véc-tơ dòng của $W^{(e)}$ (ứng với các véc-tơ trọng số đi vào các nơ-ron ẩn; các véc-tơ này đóng vai trò như các bộ lọc đặc trưng) có cùng độ dài.

- **Ràng buộc các trọng số có giá trị bình phương nhỏ (weight decay)**: các trọng số của cả bộ mã hóa và bộ giải mã đều được ràng buộc là có độ lớn nhỏ bằng cách phạt tổng bình phương của chúng [6][4]. Cách ràng buộc này vốn ban đầu được dùng trong mạng nơ-ron học có giám sát để tránh vấn đề quá khớp. Khi áp dụng cho SAEs, ta có hiểu nó là một phiên bản "mềm" của cách ràng buộc $W^{(d)}$ được chuẩn hóa ở trên và nhờ đó cũng sẽ giúp cho SAEs tránh khỏi tình trạng hàm chi phí bị giảm xuống một cách "tầm thường"; ngoài ra, nó còn ràng buộc thêm là các véc-tơ dòng của $W^{(e)}$ (ứng với các bộ lọc đặc trưng) có độ dài xấp xỉ bằng nhau (đều nhỏ). Tuy nhiên, cách ràng buộc này lại làm xuất hiện thêm một siêu tham số; ta không muốn điều này.

3.3.1 Cách ràng buộc trọng số đề xuất cho SRAEs

Với SRAEs (SAEs sử dụng hàm kích hoạt "rectified linear" ở tầng ẩn), không rõ là ta nên sử dụng cách ràng buộc trọng số nào trong những cách ở trên. Trong phần này, chúng tôi đề xuất một cách ràng buộc trọng số mới và hợp lý cho SRAEs. Cách ràng buộc này không đưa thêm siêu tham số nào. Cụ thể là, cách ràng buộc trọng số của chúng tôi bao gồm đồng thời hai ràng buộc:

- Thứ nhất, chúng tôi ràng buộc ma trận trọng số $W^{(e)}$ của bộ mã hóa và ma trận



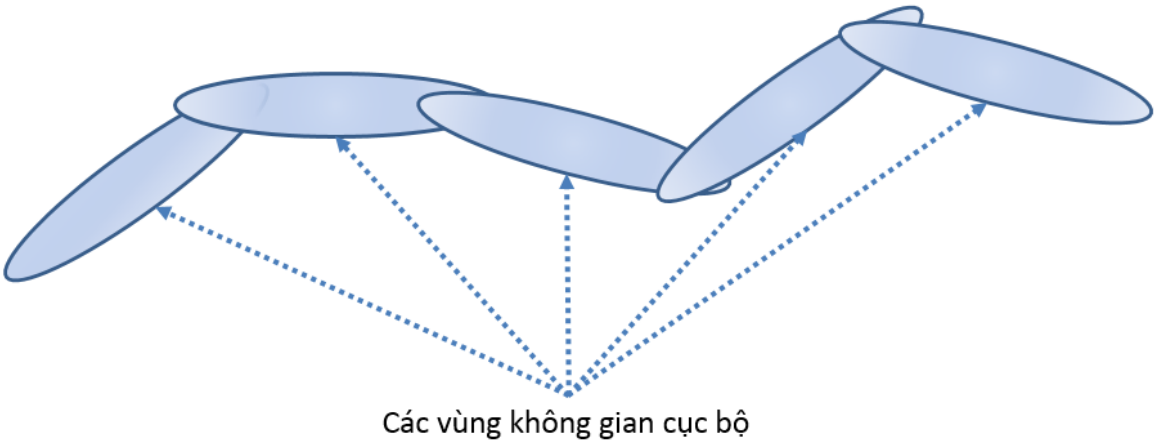
Hình 3.2: Minh họa SRAEs. Với một véc-tơ đầu vào, nếu ta chỉ chú ý đến các nơ-ron được "bật" (có giá trị đầu ra khác không) ở tầng ẩn thì đây là một hệ thống tuyến tính.

trọng số $W^{(d)}$ của bộ giải mã là chuyển vị của nhau: $W^{(d)} = (W^{(e)})^T$.

- Thứ hai, chúng tôi đồng thời cũng ràng buộc là các véc-tơ dòng của $W^{(e)}$ và các véc-tơ cột của $W^{(d)}$ được chuẩn hóa (có độ dài bằng 1). Ở đây, mỗi véc-tơ dòng của $W^{(e)}$ ứng với véc-tơ trọng số đi vào ở mỗi nơ-ron ẩn, và mỗi véc-tơ cột của $W^{(d)}$ ứng với véc-tơ trọng số đi ra ở mỗi nơ-ron ẩn.

Với một véc-tơ đầu vào x, nếu ta chỉ chú ý đến các nơ-ron được "bật" (có giá trị đầu ra khác không) ở tầng ẩn thì đây là một hệ thống tuyến tính (minh họa ở hình 3.2). Do đó, với hai ràng buộc ở trên, SRAEs sẽ chiếu véc-tơ đầu vào x xuống một hệ trục tọa độ cục bộ sao cho từ hệ trục tọa độ này có thể tái tạo được tốt véc-tơ x ban đầu; hệ trục tọa độ cục bộ này bao gồm một số ít các véc-tơ cơ sở (đã được chuẩn hóa) được chọn lựa bởi hàm "rectified linear" từ tập lớn các véc-tơ cơ sở. Ta có thể hiểu tập con các véc-tơ cơ sở này biểu diễn vùng không gian PCA cục bộ xung quanh x. Như vậy, SRAEs (với ràng buộc trọng số đề xuất của chúng tôi) có thể học được mặt phi tuyến mà ở đó dữ liệu tập trung bằng cách ghép nhiều mặt tuyến tính cục bộ lại với nhau (minh họa ở hình 3.3). Mỗi mặt tuyến tính cục bộ được phụ trách bởi một tập con các véc-tơ cơ sở. Điểm lợi ở đây là các véc-tơ cơ sở có thể được dùng chung giữa các mặt tuyến tính cục bộ láng giềng nhau.

Như vậy, ta cần phải tối thiểu hóa hàm chi phí của SRAEs với hai ràng buộc trọng số ở trên. Trong khi ràng buộc thứ nhất ($W^{(d)} = (W^{(e)})^T$) có thể được tích hợp dễ dàng vào thuật toán tối ưu hóa "Stochastic Gradient Descent" (SGD), ràng buộc thứ hai (các dòng của $W^{(e)}$ và các cột của $W^{(d)}$ được chuẩn hóa) thoạt nhìn khó có thể tích



Hình 3.3: Với một véc-tơ đầu vào x, chỉ có một tập con các nơ-ron ẩn được bật. Tập con các véc-tơ cơ sở tương ứng với tập con các nơ-ron ẩn này biểu diễn một vùng không gian cục bộ xung quanh x (giống như vùng không gian PCA cục bộ). Như vậy, SRAEs (với ràng buộc trọng số đề xuất của chúng tôi) có thể học được mặt phi tuyến mà ở đó dữ liệu tập trung bằng cách ghép nhiều mặt tuyến tính cục bộ lại với nhau. Mỗi mặt tuyến tính cục bộ được phụ trách bởi một tập con các véc-tơ cơ sở. Điểm lợi ở đây là các véc-tơ cơ sở có thể được dùng chung giữa các mặt tuyến tính cục bộ láng giềng nhau.

hợp vào thuật toán SGD và có thể ta cần phải sử dụng đến các phương pháp tối ưu hóa phức tạp hơn. Để giải quyết vấn đề này, chúng tôi thay đổi công thức lan truyền tiến của SRAEs như sau:

$$h = \max(0, \hat{W}^{(e)} x + b^{(e)}) \tag{3.6}$$

$$\hat{x} = (\hat{W}^{(e)})^T h + b^{(d)} \tag{3.7}$$

Trong đó, ma trận $\hat{W}^{(e)}$ là ma trận $W^{(e)}$ với các dòng đã được chuẩn hóa (bằng cách lấy mỗi phần tử trên một dòng của $W^{(e)}$ chia cho căn bậc hai của tổng bình phương của tất cả các phần tử trên dòng đó). Ở đây, các tham số được học vẫn là $W^{(e)}$, $b^{(e)}$, và $b^{(d)}$. Bằng cách này, ta vẫn có thể sử dụng thuật toán SGD như bình thường. Khi đưa thêm bước chuẩn hóa vào công thức lan truyền tiến như vậy, ta cũng cần phải tính lại các đạo hàm riêng của hàm chi phí theo các tham số (sẽ phức tạp hơn so với công thức lan truyền tiến ban đầu). Chúng tôi sử dụng ngôn ngữ lập trình là Theano [2]; nhờ tính năng tính đạo hàm một cách tự động của Theano, ở đây ta sẽ không cần phải tính toán cụ thể công thức của các đạo hàm riêng này.

Chương 4

Các Kết Quả Thí Nghiệm

Trong chương này, chúng tôi trình bày các kết quả thí nghiệm để đánh giá các đề xuất đã được nói ở chương trước. Bộ dữ liệu được dùng để tiến hành các thí nghiệm là bộ MNIST (bộ ảnh chữ số viết tay gồm các chữ số từ 0 đến 9). Các kết quả thí nghiệm cho thấy khi huấn luyện "Sparse Rectified Auto-Encoders" (SRAEs) với chuẩn L1 sẽ gặp phải vấn đề nơ-ron "ngủ", và chiến lược "ngủ - đánh thức" trong thuật toán "Sleep-Wake Stochastic Gradient Descent" (SW-SGD) của chúng tôi có thể giúp khắc phục vấn đề này. Các kết quả thí nghiệm cũng cho thấy cách ràng buộc trọng số đề xuất của chúng tôi cho kết quả tốt nhất trong số các cách ràng buộc trọng số có thể áp dụng cho SRAEs. Cuối cùng, thí nghiệm cũng cho thấy SRAEs với hai đề xuất trên của chúng tôi (SW-SGD và cách ràng buộc trọng số) có thể học được những đặc trưng cho kết quả phân lớp tốt khi so sánh với các loại "Auto-Encoders" khác.

4.1 Các thiết lập thí nghiệm

Chúng tôi tiến hành các thí nghiệm trên bộ dữ liệu MNIST [8]; bộ dữ liệu này gồm các ảnh xám (có kích thước 28×28) của mười chữ số viết tay từ 0 đến 9. Ở hình 4.1 là một số ảnh mẫu của bộ dữ liệu này. Dữ liệu được tiến hành tiền xử lý bằng cách lấy mỗi giá trị điểm ảnh chia cho 255 để đưa về đoạn $[0, 1]$. Chúng tôi sử dụng cách phân chia thường được sử dụng cho bộ dữ liệu này: 50000 ảnh dùng để huấn luyện, 10000 ảnh dùng để chọn các siêu tham số (validation), và 10000 ảnh dùng để kiểm tra (test).

Hình 4.1: Một số ảnh mẫu của bộ dữ liệu MNIST

Chúng tôi sử dụng ngôn ngữ lập trình Theano [2] bởi vì ngôn ngữ này cho phép dễ dàng cài đặt các thuật toán và dễ dàng sử dụng GPU (Graphical Processing Units) để tính toán song song. Loại GPU mà chúng tôi sử dụng là NVIDIA GTX 560.

Sau khi tiến hành xong bước học đặc trưng không giám sát, chúng tôi đánh giá các đặc trưng học được bằng cách sử dụng chúng để huấn luyện mô hình phân lớp "Softmax Regression" và đo độ lỗi phân lớp. Một cách cụ thể, cho tập huấn luyện $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})\}$ với $x^{(i)} \in \mathbb{R}^{D_x}$ là véc-tơ điểm ảnh và $y^{(i)} \in \{0, \ldots, 9\}$ là nhãn lớp. Sau khi "Auto-Encoder" đã được huấn luyện trên tập không có nhãn $\{x^{(1)}, \ldots, x^{(N)}\}$, ta lần lượt đưa từng véc-tơ $x^{(i)}$ vào "Auto-Encoder" và thu được ở tầng ẩn véc-tơ đặc trưng tương ứng $h^{(i)}$; bằng cách này, ta có được tập huấn luyện mới $\{(h^{(1)}, y^{(1)}), \ldots, (h^{(N)}, y^{(N)})\}$. Kế đến, tập huấn luyện mới này được sử dụng để huấn luyện "Softmax Regression". Để dự đoán nhãn lớp cho một véc-tơ đầu vào mới x_{test}, đầu tiên ta sử dụng "Auto-Encoder" đã được huấn luyện để tính véc-tơ đặc trưng tương ứng h_{test}; sau đó đưa h_{test} này vào "Softmax Regression" đã được huấn luyện để tính giá trị nhãn lớp dự đoán.

Trong cả hai giai đoạn học không giám sát và có giám sát, chúng tôi sử dụng

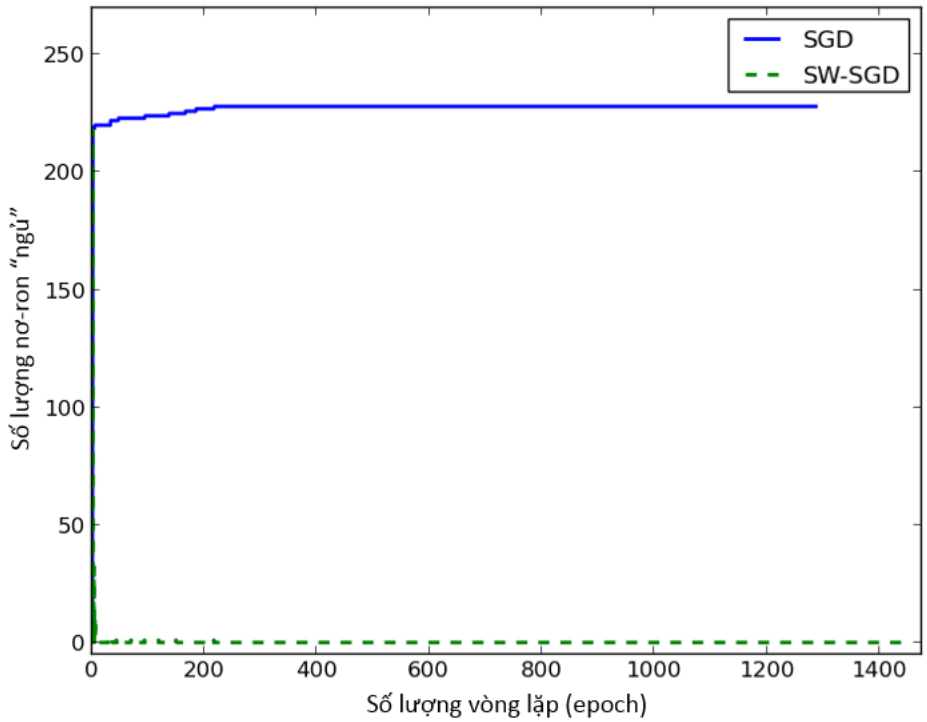
thuật toán để để cực tiểu hóa hàm chi phí là Stochastic Gradient Descent (SGD) với kích thước của "mini-batch" là 100 mẫu huấn luyện. Chiến lược "dừng sớm" (early stopping) được sử dụng để quyết định số vòng lặp (epoch) của SGD cũng như là để chống vấn đề quá khớp (trong giai đoạn học không giám sát, chúng tôi dừng quá trình tối ưu hóa dựa vào giá trị của hàm chi phí trên tập "validation"; còn trong giai đoạn học có giám sát, chúng tôi dựa vào độ lỗi phân lớp trên tập "validation"). Trong tất cả các thí nghiệm dưới đây, chúng tôi dùng SRAEs với 1000 nơ-ron ẩn, tham số "thỏa hiệp" giữa độ lỗi tái tạo và độ thưa λ bằng 0.25, hệ số học khi học không giám sát bằng 0.05, và hệ số học khi học có giám sát bằng 1 (số lượng nơ-ron ẩn được chọn theo [12], các siêu tham số còn lại được chọn dựa vào thực nghiệm).

4.2 SGD và SW-SGD

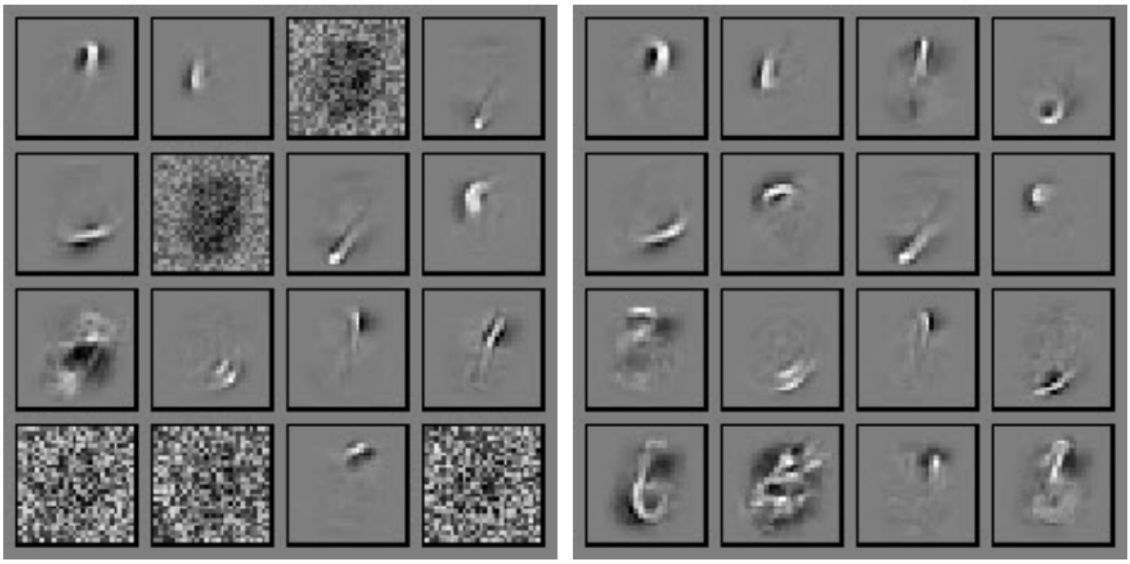
Để thấy được vấn đề gặp phải khi huấn luyện SRAEs với ràng buộc thưa bằng chuẩn L1 cũng như là tác dụng của chiến lược "ngủ - đánh thức" của chúng tôi, trong phần này chúng tôi so sánh việc huấn luyện SRAEs bằng thuật toán "Stochastic Gradient Descent" (SGD) và phiên bản điều chỉnh của chúng tôi, "Sleep-Wake Stochastic Gradient Descent" (SW-SGD). Trong thí nghiệm này, cách ràng buộc trọng số đề xuất của chúng tôi được sử dụng ($W^{(d)} = (W^{(e)})^T$, và các dòng của $W^{(e)}$ và các cột của $W^{(d)}$ được chuẩn hóa).

Hình 4.2 thể hiện số lượng nơ-ron "ngủ" của SRAEs trong khi thực hiện quá trình tối ưu hóa hàm chi phí với SGD và SW-SGD. Vấn đề gặp phải khi huấn luyện SRAEs với chuẩn L1 là trong quá trình tối ưu hóa, chuẩn L1 có thể đẩy các véc-tơ trọng số đi vào các nơ-ron ẩn vào trạng thái "ngủ" (nghĩa là, nơ-ron ẩn tương ứng luôn cho giá trị đầu ra bằng 0 với tất cả các mẫu huấn luyện) và sau đó, chúng sẽ không bao giờ còn được cập nhật nữa. Như có thể thấy từ hình 4.2, khi sử dụng SGD, số lượng nơ-ron "ngủ" tăng dần trong quá trình tối ưu hóa, đặc biệt là trong những vòng lặp đầu tiên, khi mà quá trình tối ưu hóa vẫn còn chưa ổn định. Vấn đề nơ-ron "ngủ" này của chuẩn L1 có thể được khắc phục một cách đơn giản bằng chiến lược "ngủ - đánh thức" của chúng tôi; quá trình tối ưu hóa của SW-SGD kết thúc mà không có nơ-ron "ngủ" nào cả.

Ở hình 4.3 là một số bộ lọc (một bộ lọc tương ứng với véc-tơ trọng số đi vào một



Hình 4.2: Số lượng nơ-ron "ngủ" của SRAEs trong khi thực hiện quá trình tối ưu hóa với SGD và với SW-SGD. Quá trình tối ưu hóa của SGD kết thúc với 228 nơ-ron "ngủ" trong tổng số 1000 nơ-ron; trong khi đó, SW-SGD kết thúc mà không có nơ-ron nào "ngủ". (Hai quá trình tối ưu hóa của SGD và SW-SGD kết thúc sau các số lượng vòng lặp khác nhau là do chiến lược "dừng sớm".)



(a) SGD (b) SW-SGD

Hình 4.3: Ở hình (a) là một số bộ lọc (một bộ lọc tương ứng với véc-tơ trọng số đi vào một nơ-ron ẩn) học được bởi SGD; ta có thể thấy có 5 bộ lọc nhìn vô nghĩa tương ứng với 5 nơ-ron "ngủ". Còn ở hình (b) là các bộ lọc học được bởi SW-SGD; tất cả các bộ lọc đều nhìn có nghĩa, mỗi bộ lọc dò tìm một đường nét nào đó của chữ số.

Bảng 4.1: Giá trị hàm chi phí của SRAEs trên tập huấn luyện và độ lỗi phân lớp (với "Softmax Regression") trên tập kiểm tra khi huấn luyện SRAEs với SGD và với SW-SGD.

	SGD	SW-SGD
Giá trị hàm chi phí của SRAEs trên tập huấn luyện	9.84	**9.48**
Độ lỗi phân lớp trên tập kiểm tra (%)	1.70	**1.62**

nơ-ron ẩn) học được bởi SGD và SW-SGD. Như ta có thể thấy, với SGD, có 5 nơ-ron "ngủ"; các bộ lọc của chúng nhìn vô nghĩa. Với SW-SGD, không có nơ-ron nào "ngủ"; tất cả các bộ lọc đều nhìn có nghĩa, mỗi bộ lọc dò tìm một đường nét nào đó của chữ số.

Nhờ sử dụng hết tất cả các nơ-ron ẩn, SW-SGD tìm được giá trị cực tiểu của hàm chi phí của SRAEs trên tập huấn luyện tốt hơn so với SGD; và các đặc trưng học được của SW-SGD cũng cho kết quả phân lớp (với "Softmax Regression") trên tập kiểm tra tốt hơn so với SGD (bảng 4.1).

4.3 Cách ràng buộc trọng số đề xuất của chúng tôi và các cách ràng buộc trọng số khác

Trong thí nghiệm thứ hai này, cách ràng buộc trọng số đề xuất cho SRAEs của chúng tôi được so sánh với các cách ràng buộc trọng số khác mà có thể áp dụng cho SRAEs. Cụ thể ở đây, chúng tôi so sánh với các cách ràng buộc trọng số sau:

- $W^{(d)}$ **được chuẩn hóa:** các véc-tơ cột của $W^{(d)}$ được ràng buộc là chuẩn hóa (có độ dài bằng 1); mỗi véc-tơ cột của $W^{(d)}$ tương ứng với véc-tơ trọng số đi ra ở mỗi nơ-ron ẩn.

- $W^{(e)}$ **và** $W^{(d)}$ **được chuẩn hóa:** các véc-tơ dòng của $W^{(e)}$ và các véc-tơ cột của $W^{(d)}$ được ràng buộc là chuẩn hóa (có độ dài bằng 1); mỗi véc-tơ dòng của $W^{(e)}$ và mỗi véc-tơ cột của $W^{(d)}$ lần lượt tương ứng với véc-tơ trọng số đi vào và véc-tơ trọng số đi ra ở mỗi nơ-ron ẩn.

- $W^{(d)} = (W^{(e)})^T$**:** $W^{(e)}$ và $W^{(d)}$ được ràng buộc là chuyển vị của nhau.

Cách ràng buộc trọng số của chúng tôi là kết hợp của hai ràng buộc: $W^{(e)}$ và $W^{(d)}$ được chuẩn hóa, và $W^{(d)} = (W^{(e)})^T$. Trong thí nghiệm này, chúng tôi dùng SW-SGD để huấn luyện SRAEs.

Như có thể thấy ở bảng 4.2, trong số các cách ràng buộc trọng số, cách ràng buộc của chúng tôi giúp SRAEs học được những đặc trưng cho kết quả phân lớp (với "Softmax Regression") tốt nhất trên tập kiểm tra. Ngoài ra, bảng 4.2 cũng so sánh thời gian huấn luyện SRAEs trên một vòng lặp (ứng với một lần duyệt qua toàn bộ các mẫu huấn luyện) với các cách ràng buộc trọng số khác nhau này (do chiến lược "dừng sớm", quá trình huấn luyện SRAEs với các cách ràng buộc khác nhau có thể kết thúc sau các số lượng vòng lặp khác nhau; do đó, để chính xác, ta nên so sánh theo thời gian huấn luyện xét trên một vòng lặp hơn là tổng thời gian huấn luyện). Các cách ràng buộc trọng số được sắp xếp theo thứ tự thời gian huấn luyện (trên một vòng lặp) tăng dần là: $W^{(d)} = (W^{(e)})^T$ (2 giây), $W^{(d)}$ được chuẩn hóa (3 giây), cách ràng buộc trọng số của chúng tôi (4 giây), $W^{(e)}$ và $W^{(d)}$ được chuẩn hóa (5 giây). Thứ tự này là hợp lý:

- Ràng buộc $W^{(d)} = (W^{(e)})^T$ có thời gian huấn luyện nhanh nhất vì SRAEs không phải thực hiện bước chuẩn hóa.

- Ràng buộc $W^{(d)}$ được chuẩn hóa có thời gian huấn luyện lâu hơn vì bộ giải mã của SRAEs phải thực hiện bước chuẩn hóa khi lan truyền tiến; và do đó, khi lan truyền ngược, việc tính toán các đạo hàm riêng theo các tham số của bộ giải mã cũng sẽ tốn thời gian hơn bình thường.

- Ở cách ràng buộc trọng số của chúng tôi, khi lan truyền tiến, mặc dù cần phải thực hiện bước chuẩn hóa ở cả bộ mã hóa và bộ giải mã, nhưng nhờ vào ràng buộc $W^{(d)} = (W^{(e)})^T$, ta chỉ cần phải thực hiện bước chuẩn hóa cho bộ trọng số của bộ mã hóa, rồi sau đó dùng lại bộ trọng số đã được chuẩn hóa này cho bộ giải mã. Thời gian huấn luyện của cách ràng buộc này lâu hơn cách ràng buộc $W^{(d)}$ được chuẩn hóa ở trên vì khi lan truyền ngược, ngoài việc tính toán các đạo hàm riêng theo các tham số của bộ giải mã đã được chuẩn hóa, ta cũng cần phải tính toán các đạo hàm riêng theo các tham số của bộ mã hóa đã được chuẩn hóa (khi bộ mã hóa hay bộ giải mã phải thực hiện bước chuẩn hóa khi lan truyền tiến thì việc tính toán các đạo hàm riêng theo các tham số của chúng khi lan truyền ngược sẽ lâu hơn so với khi không thực hiện bước chuẩn hóa).

- Ràng buộc $W^{(e)}$ và $W^{(d)}$ được chuẩn hóa có thời gian huấn luyện lâu nhất vì khi lan truyền tiến, ta phải thực hiện bước chuẩn hóa riêng cho bộ mã hóa và bộ giải mã; và khi lan truyền ngược, ta phải tính toán các đạo hàm riêng theo các tham số của bộ giải mã và bộ mã hóa đã được chuẩn hóa.

Mặc dù thời gian huấn luyện (trên một vòng lặp) của cách ràng buộc trọng số của chúng tôi là khá cao khi so sánh với cách ràng buộc trọng số khác, nhưng nhìn chung nó vẫn nhanh (nhờ vào việc sử dụng GPU để tính toán song song). Tổng thời gian huấn luyện là khoảng 2.5 giờ.

Bảng 4.2: So sánh giữa cách ràng buộc trọng số cho SRAEs của chúng tôi với các cách ràng buộc trọng số khác mà có thể áp dụng cho SRAEs. Cách ràng buộc trọng số của chúng tôi giúp SRAEs học được những đặc trưng mà cho kết quả phân lớp (với "Softmax Regression") tốt nhất trên tập kiểm tra. Ngoài ra, thời gian huấn luyện trên một vòng lặp của SRAEs với các cách ràng buộc trọng số khác nhau cũng được trình bày ở cột cuối cùng của bảng.

Cách ràng buộc trọng số	**Độ lỗi phân lớp trên tập kiểm tra (%)**	**Thời gian huấn luyện của một vòng lặp (giây)**
$W^{(d)}$ được chuẩn hóa	3.28	3
$W^{(e)}$ & $W^{(d)}$ được chuẩn hóa	2.51	5
$W^{(d)} = (W^{(e)})^T$	2.04	2
Cách ràng buộc của chúng tôi	**1.62**	4

4.4 SRAEs và các loại "Auto-Encoders" khác

Cuối cùng, chúng tôi cũng so sánh SRAEs (sử dụng cách ràng buộc trọng số của chúng tôi và dùng SW-SGD để huấn luyện) với các loại "Auto-Encoders" khác, bao gồm:

- **"Denoising Auto-Encoders" (DAEs)** [14]: DAEs muốn học được các đặc trưng "bền vững" bằng cách làm nhiễu véc-tơ đầu vào rồi sau đó cố gắng tái tạo lại véc-tơ đầu vào ban đầu từ véc-tơ đã bị làm nhiễu này (véc-tơ đầu vào đã bị làm nhiễu $\rightarrow$ véc-tơ đặc trưng $\rightarrow$ cố gắng tái tạo lại véc-tơ đầu vào không bị nhiễu).

- **"Contractive Auto-Encoders" (CAEs)** [13]: DAEs muốn học được các đặc trưng thỏa hai tính chất: (i) có thể tái tạo tốt véc-tơ đầu vào ban đầu, và (ii) bất biến đối với sự thay đổi nhỏ của véc-tơ đầu vào (bằng cách phạt chuẩn Frobenius của ma trận Jacobian của véc-tơ đặc trưng đối với véc-tơ đầu vào).

- **"Higher Order Contractive Auto-Encoders" (HCAEs)** [12]: HCAEs là mở rộng của CAEs; bên cạnh độ lỗi tái tạo và chuẩn Frobenius của ma trận Jacobian, HCAEs còn phạt thêm chuẩn Frobenius của ma trận Hessian.

Bảng 4.3 so sánh các đặc trưng học được (theo độ lỗi phân lớp trên tập kiểm tra) của SRAEs với các loại "Auto-Encoders" trên. Với DAEs, CAEs, HCAEs, [12] dùng 1000 nơ-ron ẩn, hàm kích hoạt sigmoid ở cả tầng ẩn và tầng đầu ra, độ lỗi tái tạo

Bảng 4.3: So sánh giữa SRAEs (sử dụng cách ràng buộc trọng số của chúng tôi và dùng SW-SGD để huấn luyện) với các loại "Auto-Encoders" khác, bao gồm: "Denoising Auto-Encoders" (DAEs), "Contractive Auto-Encoders" (CAEs), "Higher Order Contractive Auto-Encoders" (HCAEs).

Thuật toán học đặc trưng	**Độ lỗi phân lớp trên tập kiểm tra (%)**
DAEs [12]	2.05
CAEs [12]	1.82
SRAEs	1.62
HCAEs [12]	**1.20**

"cross-entropy", và ràng buộc $W^{(e)}$ và $W^{(d)}$ là chuyển vị của nhau. Như có thể thấy, các đặc trưng học được bởi SRAEs cho kết quả phân lớp (với "Softmax Regression") trên tập kiểm tra tốt hơn DAEs và CAEs, nhưng không tốt bằng HCAEs. Tuy nhiên, để ý là HCAEs phức tạp hơn nhiều so với SRAEs của chúng tôi với rất nhiều siêu tham số cần phải lựa chọn.

TÀI LIỆU THAM KHẢO

[1] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," 2013. 2, 4

[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)*, Jun. 2010, oral Presentation. 32, 34

[3] A. Coates, "Demystifying unsupervised feature learning," Ph.D. dissertation, Stanford University, 2012. 4, 23, 29

[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*, 2011, pp. 215–223. 4, 23, 30

[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*, vol. 15, 2011, pp. 315–323. 22, 23

[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*, 2009, pp. 646–654. 4, 23, 30

[7] Q. V. Le, A. Karpenko, J. Ngiam, and A. Y. Ng, "Ica with reconstruction cost for efficient overcomplete feature learning," in *NIPS*, 2011, pp. 1017–1025. 30

[8] Y. LeCun, "The MNIST database," http://yann.lecun.com/exdb/mnist/. 33

[9] 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

[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)*, 2010, pp. 807–814. 22

[11] B. A. Olshausen *et al.*, "Emergence of simple-cell receptive field properties by learning a sparse code for natural images," *Nature*, vol. 381, no. 6583, pp. 607–609, 1996. 3, 6

[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*, pp. 645–660, 2011. 4, 29, 35, 40, 41

[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)*, 2011, pp. 833–840. 4, 29, 40

[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*. ACM, 2008, pp. 1096–1103. 4, 29, 40

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

[16] 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. 4, 22, 23, 30