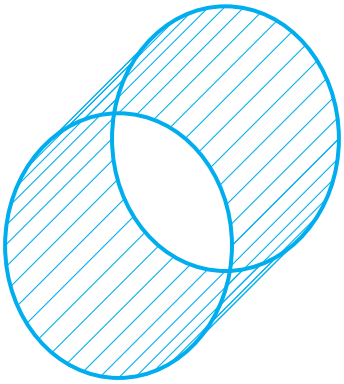
ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA TOÁN - TIN HỌC

———————oOo———————

REPORT

Autoencoder versus Kernel PCA and Non-negative Matrix Factorization



Khoa Toán - Tin học
Fac. of Math. & Computer Science

Supervisor:	**Ngô Minh Mẫn**	
Group of students:	**Lê Hoàng Đức**	**- 18110075**
	Nguyễn Duy Thanh	**- 18110013**
	Đinh Anh Huy	**- 18110103**

May 25, 2021, Ho Chi Minh, Viet Nam

Mục lục

1 Kernel Principal Components Analysis - KPCA

1.1 Constructing the Kernel Matrix

Principal Components Analysis (PCA) là một phép biến đổi cơ sở để lập đường chéo ước tính ma trận hiệp phương sai của dữ liệu $\boldsymbol{x}_k, k = 1, \ldots, m, \boldsymbol{x}_k \in \mathbb{R}^D, \sum_{k=1}^{m} \boldsymbol{x}_k = 0$, định nghĩa là :

$$C = \frac{1}{m} \sum_{i=1}^{m} \boldsymbol{x}_i \boldsymbol{x}_i^T$$

Các tọa độ mới trong cơ sở "Eigenvector", tức là các phép chiếu trực giao lên "Eigenvector", được gọi là các thành phần chính (Principal Components).

Giả sử dữ liệu của chúng ta là phi tuyến. Khi đó , chúng ta sẽ ánh xạ dữ liệu phi tuyến vào một không gian đặc trưng (feature space) $\mathbb{R}^N, D < N$ bằng cách :

$$\begin{aligned} \Phi \quad : \quad & \mathbb{R}^D \longrightarrow \mathbb{R}^N \\ & \boldsymbol{x} \longmapsto \Phi(\boldsymbol{x}) \end{aligned}$$

Sau đó , chúng ta sẽ thực hiện PCA trên không gian đặc trưng này. Cách làm như trên được gọi là Kernel Principal Components Analysis (KPCA).

Giả sử dữ liệu khi được ánh xạ vào không gian đặc trưng thõa tính chất sau : $\sum_{i=1}^{m} \Phi(\boldsymbol{x}_i) = 0$. Hay trung bình của dữ liệu trong không gian đặ trưng là :

$$\mu = \frac{1}{m} \sum_{i=1}^{m} \Phi(\boldsymbol{x}_i) = 0$$

Ma trận hiệp phương sai :

$$\underset{(N \times N)}{\boldsymbol{C}} = \frac{1}{m} \sum_{i=1}^{m} \Phi(\boldsymbol{x}_i) \Phi(\boldsymbol{x}_i)^T$$

Chúng ta sẽ đi tìm trị riêng $\lambda_i \geq 0$ và vector riêng $\boldsymbol{v}_i \in \mathbb{R}^N, \forall i = 1, 2, \ldots, N$ thõa :

$$\boldsymbol{C}\boldsymbol{v}_i = \lambda_i \boldsymbol{v}_i$$

Từ (**??**), ta thấy các nghiệm $\boldsymbol{v}$ đều nằm trong không gian sinh bởi $\Phi(\boldsymbol{x}_1), \ldots, \Phi(\boldsymbol{x}_m)$. Điều đó có nghĩa là ứng với mỗi $k, k = 1, 2, \ldots, N$ tồn tại các hệ số $\alpha_{k1}, \ldots, \alpha_{km}$ thõa

$$\boldsymbol{v}_k = \sum_{i=1}^{m} \alpha_{ki} \Phi(\boldsymbol{x}_i), \quad \forall k = 1, 2, \ldots, N.$$

Ta đặt :

$$\boldsymbol{K}(\boldsymbol{x}_i, \boldsymbol{x}_k) = \Phi(\boldsymbol{x}_i)^T \Phi(\boldsymbol{x}_k)$$

Bằng các phép biến đổi đơn giản , ta có kết quả sau :

$$\boldsymbol{K}^2 \boldsymbol{\alpha}_k = m\lambda_k \boldsymbol{K} \boldsymbol{\alpha}_k \tag{1}$$

trong đó $\boldsymbol{\alpha}_k$ là vector cột với các phần tử $\alpha_{k1}, \ldots, \alpha_{km}$. Để tìm nghiệm thõa (1), ta đi tìm các trị riêng khác 0 thõa

$$\boldsymbol{K}\boldsymbol{\alpha}_k = m\lambda_k \boldsymbol{\alpha}_k, \quad \forall k = 1, 2, \ldots, N. \tag{2}$$

Sử dụng điều kiện chuẩn tắc của các vector riêng

$$\boldsymbol{v}_k^T \boldsymbol{v}_k = 1 \quad \Rightarrow \boldsymbol{\alpha}_k^T \boldsymbol{K} \boldsymbol{\alpha}_k = 1 \quad \Rightarrow m\lambda_k \boldsymbol{\alpha}_k^T \boldsymbol{\alpha}_k = 1, \quad \forall k = 1, 2, \ldots, N. \tag{3}$$

Với mỗi điểm $\boldsymbol{x}$ mới , hình chiếu của nó lên thành phần chính $\boldsymbol{v}_k$ ở không gian feature là

$$y_k(\boldsymbol{x}) = \Phi(\boldsymbol{x})^T \boldsymbol{v}_k = \sum_{i=1}^{m} \alpha_{ki} \Phi(\boldsymbol{x})^T \Phi(\boldsymbol{x}_i) = \sum_{i=1}^{m} \alpha_{ki} \boldsymbol{K}(\boldsymbol{x}, \boldsymbol{x}_i), \quad \forall k = 1, 2, \ldots, N. \tag{4}$$

Ngoài ra, chúng ta có thể giới hạn số lượng thành phần $K < N$ bằng cách chọn K giá trị α_i ứng với K trị riêng lớn nhất.
Nếu tập dữ liệu $\Phi(\boldsymbol{x}_i)$ không có trung bình bằng 0. Khi đó, ta đặt:

$$\widetilde{\Phi}(\boldsymbol{x}_k) = \Phi(\boldsymbol{x}_k) - \frac{1}{m} \sum_{i=1}^{m} \Phi(\boldsymbol{x}_i) \tag{5}$$

Khi đó , các phần tử của ma trận kernel tương ứng

$$\widetilde{\boldsymbol{K}}(\boldsymbol{x}_k, \boldsymbol{x}_j) = \widetilde{\Phi}(\boldsymbol{x}_k)^T \widetilde{\Phi}(\boldsymbol{x}_j)$$

Qua các phép biến đổi đại số , ta được

$$\widetilde{\boldsymbol{K}} = \boldsymbol{K} - 2\mathbf{1}_{1/m} \boldsymbol{K} + \mathbf{1}_{1/m} \boldsymbol{K} \mathbf{1}_{1/m}$$

trong đó $\mathbf{1}_{1/m}$ là ma trận kích thước $m \times m$ với tất cả các phần tử đều là $1/m$.

1.2 Reconstruction

Ta định nghĩa một toán tử P_K để reconstruct $\Phi(\boldsymbol{x})$ từ phép chiếu của chính nó lên $\boldsymbol{v}_k$ thông qua K vector thành phần chính ở không gian feature:

$$P_K\Phi(\boldsymbol{x}) = \sum_{k=1}^{K} y_k(\boldsymbol{x})\boldsymbol{v}_k = \sum_{k=1}^{K}\left(y_k(\boldsymbol{x})\sum_{j=1}^{K}\alpha_{kj}\Phi(\boldsymbol{x}_j)\right)$$

trong đó $\boldsymbol{v}_k$ là vector riêng của ma trận $\boldsymbol{C}$.
Nếu K đủ lớn , ta có :

$$P_K\Phi(\boldsymbol{x}) \approx \Phi(\boldsymbol{x})$$

Tuy nhiên, ta mong muốn xấp xỉ tính toán của các phép ánh xạ thông qua không gian dữ liệu gốc hơn là tính toán ở không gian feature, do đó chuyển sang tìm một giá trị gần đúng $\boldsymbol{z}$ sao cho

$$\Phi(\boldsymbol{z}) \approx P_K\Phi(\boldsymbol{x})$$

Điều này có thể được xấp xỉ bằng cách "minimize" :

$$\rho(\boldsymbol{z}) = \|\Phi(\boldsymbol{z}) - P_K\Phi(\boldsymbol{x})\|^2$$

Hoặc, nếu gọi $P_K\Phi(\boldsymbol{x})$ là toán tử reconstruct, ta có thể viết lại hàm mục tiêu cho toàn bộ dữ liệu dưới dạng

$$min\sum_{i}^{m}\|\Phi(\boldsymbol{x}_i) - \boldsymbol{W}_K\boldsymbol{W}_K^T\Phi(\boldsymbol{x}_i)\|^2$$

trong đó, W_K là ma trận tạo thành tử K vector riêng v_k.

Nếu trường hợp $P_K\Phi(\boldsymbol{x}) \neq \Phi(\boldsymbol{x})$, ta không thể đảm bảo việc reconstruct là chính xác hoàn toàn, nghĩa là ρ có thể khác 0. Do vậy, một cách tiếp cận để tối ưu hàm ρ là sử dụng gradient methods thông qua biến $\boldsymbol{z}$. Để dễ dàng thấy rõ, ta viết lại ρ dưới dạng:

$$\rho(\boldsymbol{z}) = \boldsymbol{K}(\boldsymbol{z},\boldsymbol{z}) - 2\sum_{k=1}^{K} y_k \sum_{i=1}^{m}\alpha_{ki}\boldsymbol{K}(\boldsymbol{x_i},\boldsymbol{z}) + C$$

2 Non-Negative Matrix Factorization - NMF

2.1 Problem definition

Cho ma trận không âm $\boldsymbol{V}$, tìm các ma trận nhân tố không âm $\boldsymbol{W}, \boldsymbol{H}$ sao cho:

$$\boldsymbol{V} \approx \boldsymbol{W}\boldsymbol{H}$$

Hay nói cách khác, NMF là phương pháp *xấp xỉ tuyến tính, không âm* cho dữ liệu mô tả. Giả sử, ta có tập dữ liệu với các vector n chiều. Trong đó, các vector này được xem là các cột của ma trận $\boldsymbol{V}$ có số chiều là $n \times m$ với m là số lượng quan sát của tập dữ liệu.

Dữ liệu sau khi được phân tích thành tích của hai ma trận $\boldsymbol{W}$ có số chiều là $n \times r$ và $\boldsymbol{H}$ có số chiều $r \times m$ (thông thường r được chọn sao cho $r \leq max(n, m)$). Trong đó, các cột của ma trận $\boldsymbol{W}$ được gọi là các vector cơ sở $\boldsymbol{w}_i$ và các dòng của ma trận $\boldsymbol{H}$ được gọi là các vector hệ số. Từ đây ta cũng có thể biểu diễn bài toán dưới dạng tổ hợp tuyến tính của mỗi vector cột v của $\boldsymbol{V}$

$$\boldsymbol{v}_i \approx \boldsymbol{W}\boldsymbol{h}_i$$

Kết quả này được xem là phiên bản *nén* của dữ liệu ban đầu.

2.2 Loss functions

Có rất nhiều dạng *objective/loss functions* khác nhau. Báo cáo này sử dụng hàm *Bình phương sai số* có dạng:

$$\mathcal{L}(\boldsymbol{W}, \boldsymbol{H}) = \|\boldsymbol{V} - \boldsymbol{W}\boldsymbol{H}\|^2 = \sum_{i,j} (\boldsymbol{V}_{ij} - (\boldsymbol{W}\boldsymbol{H})_{ij})^2$$

2.3 Algorithms

Có rất nhiều thuật toán giúp NMF tìm được ma trận $\boldsymbol{W}, \boldsymbol{H}$ sao cho $\min \mathcal{L}(\boldsymbol{W}, \boldsymbol{H})$. Một trong số đó là một phương pháp nổi tiếng được đề ra bởi Lee & Seungs, được gọi là *Lee and Seung's multiplicative update rule*:

2.4 Uniqueness

Phép phân tích ma trận trên là không duy nhất. Ví dụ:

$$\boldsymbol{W}\boldsymbol{H} = \boldsymbol{W}\boldsymbol{B}\boldsymbol{B}^{-1}\boldsymbol{H}$$

Với $\boldsymbol{B}$ là ma trận không âm, khi đó đặt $\widetilde{\boldsymbol{W}} = \boldsymbol{W}\boldsymbol{B}$ và $\widetilde{\boldsymbol{H}} = \boldsymbol{B}^{-1}\boldsymbol{H}$. Từ đây ta kết luận phép phân tích trên là không duy nhất.

Algorithm 1 Update rule

procedure UPDATE RULE

loop: *n as an index of the iteration*

Initialize $\boldsymbol{W}, \boldsymbol{H}$ non negative

$$\boldsymbol{H}^{n+1}_{[i,j]} \leftarrow \boldsymbol{H}^{n}_{[i,j]} \frac{((\boldsymbol{W}^n)^T \boldsymbol{V})_{[i,j]}}{((\boldsymbol{W}^n)^T \boldsymbol{W}^n \boldsymbol{H}^n)_{[i,j]}}$$

$$\boldsymbol{W}^{n+1}_{[i,j]} \leftarrow \boldsymbol{W}^{n}_{[i,j]} \frac{(\boldsymbol{V}(\boldsymbol{H}^{n+1})^T)_{[i,j]}}{(\boldsymbol{W}^n \boldsymbol{H}^{n+1}(\boldsymbol{H}^{n+1})^T)_{[i,j]}}$$

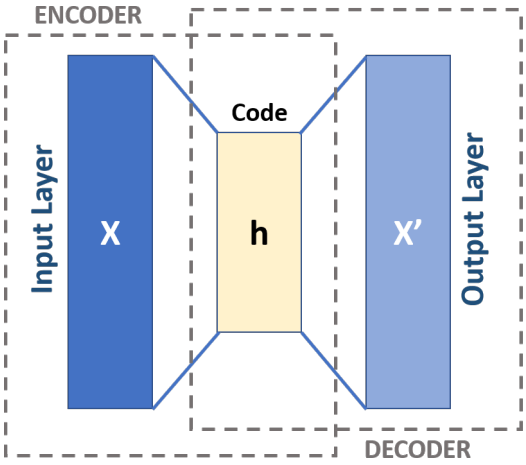
Until $\boldsymbol{W}, \boldsymbol{H}$ are stable

*Note that the updates are done on an element by element basis not matrix multiplication.

3 Autoencoder - AE

3.1 Introduction

Autoencoder (AE) là một dạng neural network, điểm khác so với các kiến trúc mạng khác là mạng này dùng chính input làm ground truth. Hay nói cách khác, AE là phương pháp khôi phục lại dữ liệu gốc (bằng cách tối thiểu hoá khoảng cách giữa input và output) thay vì dự đoán giá trị đầu ra mới từ input đã cho. Autoencoder gồm có 2 phần chính là encoder và decoder. Về cơ bản, mạng sẽ có dạng như sau



Hình 1: Sơ đồ cấu trúc của một Autoencoder đơn giản[1]

[1]Nguồn ảnh: https://en.wikipedia.org/wiki/Autoencoders

3.2 Two-layers Autoencoder

Cho ma trận dữ liệu $\boldsymbol{X} \in \mathbb{R}^d$. Xét trường hợp đơn giản, mô hình chỉ có một hidden layer. Xét các phép biến đổi tương ứng cho 2 quá trình encode và decode như sau

$$\phi : \mathbb{R}^d \to \mathbb{R}^p \quad \text{và} \quad \psi : \mathbb{R}^p \to \mathbb{R}^d$$

Lúc này, đầu ra của quá trình encode là $\boldsymbol{Y} \in \mathbb{R}^p$ được xem là dạng biểu diễn rút gọn (reduced representation) của $\boldsymbol{X}$ thường được gọi là *code*, *latent variables* hay *latent representation* và được biểu diễn như sau

$$\boldsymbol{Y} = \phi(\boldsymbol{X}) = f_\phi(\boldsymbol{W}\boldsymbol{X} + \boldsymbol{b_X}) \tag{6}$$

trong đó f_ϕ là hàm activation, $\boldsymbol{W}$ là ma trận trọng số (weight) và $\boldsymbol{b_X}$ là vectơ sai số (bias). Weights và biases thường được xác định một cách ngẫu nhiên và được cập nhật liên tục trong suốt quá trình training thông qua thuật toán backpropagation.

Quá trình decode ánh xạ hidden representation $\boldsymbol{Y}$ vào reconstruction $\boldsymbol{X'}$ có cùng kích thước với $\boldsymbol{X}$

$$\boldsymbol{X'} = \psi(\boldsymbol{Y}) = f_\psi(\boldsymbol{W'}\boldsymbol{Y} + \boldsymbol{b_Y}) \tag{7}$$

trong đó, f_ψ là hàm activation, ma trận trọng số $\boldsymbol{W'}$ và vectơ sai số $\boldsymbol{b}_\psi$.

Bài toán Autoencoder sẽ đi tìm các hệ số $\theta = (\boldsymbol{W}, \boldsymbol{W'}, \boldsymbol{b_X}, \boldsymbol{b_Y})$ sao cho *reconstruction loss* trên tập dữ liệu đầu vào $\boldsymbol{X}$ đạt giá trị nhỏ nhất và hàm objective được cho như sau

$$\Theta = \min_\theta \mathcal{L}(\boldsymbol{X}, \boldsymbol{X'}) = \min_\theta \mathcal{L}(\boldsymbol{X}, \psi(\phi(\boldsymbol{X}))) \tag{8}$$

Trong trường hợp reconstruction tuyến tính, thì reconstruction loss ($\mathcal{L}_1$) thường sử dụng squared error

$$\mathcal{L}_1 = \sum_{i=1}^{m} \|x_i - x_i'\|^2 = \sum_{i=1}^{m} \|x_i - \psi(\phi(x_i))\|^2 \tag{9}$$

Trong trường hợp reconstruction phi tuyến, thì reconstruction loss ($\mathcal{L}_2$) thường sử dụng cross-entropy

$$\mathcal{L}_2 = -\sum_{i=1}^{m} [x_i \log(y_i) + (1 - x_i) \log(1 - y_i)] \tag{10}$$

trong đó $x_i \in \boldsymbol{X}, x_i' \in \boldsymbol{X'}$ và $y_i \in \boldsymbol{Y}$.

3.3 Multi-layers Autoencoder

Trong trường hợp mở rộng, mô hình autoencoder có nhiều hơn 2 lớp layers. Với $L \geq 3$, xét tập hợp các không gian Hilbert $\mathcal{X}_0, \mathcal{X}_1, ..., \mathcal{X}_L$ với $\mathcal{X}_L = \mathcal{X}_0$. Với $0 \leq \ell \leq L-1$, trong không gian $\mathcal{X}_\ell$, ta định nghĩa phép biến đổi:

$$\psi_\ell : \mathcal{X}_\ell \to \mathcal{X}_{\ell+1}$$

Lúc này, ta mong muốn tối thiểu hoá reconstruction loss ($\mathcal{L}$) trên $\prod_{\ell=1}^{L} \mathcal{X}_\ell$:

$$\min_{\psi_\ell \in \mathcal{X}_\ell} \sum_{i=1}^{m} \|x_i - \psi_L \circ \cdots \circ \psi_1(x_i)\|^2_{\mathcal{X}_0}$$

4 Autoencoder versus Non-negative Matrix Factorization

4.1 Non-negative Sparse Autoencoder (NSAE)

Như ta được biết, hàm mục tiêu (objective function) của *autoencoder* giúp khôi phục hoặc tái dựng lại dữ liệu đầu vào. Trong suốt quá trình huấn luyện, các trọng số của hidden neurons là tổ hợp của các điểm input và trọng số của lớp layer trước đó. Tuy nhiên, với mô hình *Non-negative Autoencoder* cơ bản các trọng số này sẽ càng lớn theo số lượng layers, và dẫn đến tạo ra các đặc trưng phụ thuộc vào cấu trúc của mạng (network) hơn là dữ liệu đầu vào. Để tránh việc này, một biến thể chính là *Non-negative Sparse Autoencoder (NSAE)* sẽ giúp kiểm soát độ lớn của các trọng số đó cùng với ràng buộc về tính không âm.

4.1.1 Definition

Cho ma trận dữ liệu $\boldsymbol{X}$ không âm (nghĩa là: $\forall i, j : X_{ij} \geq 0$). Hàm mục tiêu cho bài toán reconstruction có dạng:

$$\mathcal{L}(\boldsymbol{W}, \boldsymbol{H}) = \|\boldsymbol{X} - \boldsymbol{W}\boldsymbol{H}\|^2 + \lambda \sum_{i,j} f(\boldsymbol{H}_{ij}) \tag{5-1}$$

sao cho ma trận trọng số $\boldsymbol{W}$ và $\boldsymbol{H}$ thỏa mãn: $\forall ij : \boldsymbol{W}_{ij} \geq 0, \boldsymbol{H}_{ij} \geq 0$ và $\forall i : \|\boldsymbol{w}_i\| = 1$, với $\boldsymbol{w}_i$ là các vector cột của ma trận $\boldsymbol{W}$. Và điều kiện dương cũng được áp dụng cho λ ($\lambda \geq 0$).

Chú ý rằng, nếu ta sử dụng một *linear activation penalty* cho phép đo tính *thưa* (nghĩa là: $f(H) = H$), hàm mục tiêu lúc này sẽ được viết lại dưới dạng:

$$\mathcal{L}(\boldsymbol{W}, \boldsymbol{H}) = \|\boldsymbol{X} - \boldsymbol{W}\boldsymbol{H}\|^2 + \lambda \sum_{i,j} \boldsymbol{H}_{ij} \tag{5-2}$$

thông thường, với cách chọn linear activation penalty như trên, sẽ giúp hàm mục tiêu trở thành dạng "quadratic form" theo $\boldsymbol{H}$. Điều này sẽ có lợi cho việc hội tụ của thuật toán để tìm các hidden components $\boldsymbol{H}$.

4.1.2 Estimating the hidden components

Như đã đề cập trên, hàm mục tiêu có dạng "quadratic form" nên ta hy vọng sẽ không có điểm cực trị địa phương nào tồn tại hoặc nói cách khác, ta mong muốn tìm được trực trị toàn cục để tối ưu hàm mục tiêu. Bằng cách sử dụng các thuật toán gradient descent có thể giúp ta giải quyết bài toán.
Theorem Hàm mục tiêu sẽ là hàm không tăng với luật cập nhật như sau:

$$\boldsymbol{H}^{t+1} = \boldsymbol{H}^t * (\boldsymbol{W}^T\boldsymbol{X})/(\boldsymbol{W}^T\boldsymbol{W}\boldsymbol{H}^t + \lambda)$$

trong đó, phép * và phép / chỉ phép nhân element-wise và chia tương ứng, và phép cộng hằng số λ là cộng với toàn bộ phần tử của ma trận $\boldsymbol{W}^T\boldsymbol{W}\boldsymbol{H}^t$.

4.1.3 Learning the basis

Để tối ưu hàm mục tiêu, ta cần tối ưu với ma trận $\boldsymbol{W}$ và cả ma trận $\boldsymbol{H}$. Trước hết, ta cố định $\boldsymbol{H}$ để đi cập nhật $\boldsymbol{W}$. Ta có quy tắc cập nhật cho $\boldsymbol{W}$ như sau.

1. $\boldsymbol{W}^{'} = \boldsymbol{W}^t - \mu(\boldsymbol{W}^t\boldsymbol{H} - \boldsymbol{X})\boldsymbol{H}^T$
2. Gán tất cả giá trị âm của ma trận $\boldsymbol{W}^{'}$ bằng 0
3. Chuẩn hóa các cột của ma trận $\boldsymbol{W}^{'}$ về dạng unit norm, sau đó gán $\boldsymbol{W}^{t+1} = \boldsymbol{W}^{'}$

Từ mục 5.1.2 và 5.1.3, ta có thuật toán cho NSAE như sau:

1. Khởi tạo $\boldsymbol{W}^0, \boldsymbol{H}^0$ là các ma trận dương (ngặt) và chuẩn hóa các cột của ma trận $\boldsymbol{W}$ về dạng unit norm. Gán t = 0.
2. Lặp đến khi hội tụ:
   a) $\boldsymbol{W}^{'} = \boldsymbol{W}^t - \mu(\boldsymbol{W}^t\boldsymbol{H} - \boldsymbol{X})\boldsymbol{H}^T$
   b) Gán tất cả giá trị âm của ma trận $\boldsymbol{W}^{'}$ bằng 0

c) Chuẩn hóa các cột của ma trận $\boldsymbol{W}^{'}$ về dạng unit norm, sau đó gán $\boldsymbol{W}^{t+1} = \boldsymbol{W}^{'}$

d) $\boldsymbol{H}^{t+1} = \boldsymbol{H}^{t} * (\boldsymbol{W}^{T}\boldsymbol{X})/(\boldsymbol{W}^{T}\boldsymbol{W}\boldsymbol{H}^{t} + \lambda)$

e) tăng t

4.2 Non-negative symmetric encoder-decoder

Mở rộng một dạng khác của bài toán NSAE. Thay vì sử dụng ràng buộc như hàm mục tiêu (5-2), một mô hình encoder-decoder sử dụng tính chất đối xứng thông qua việc sử dụng cùng ma trận $\boldsymbol{W}$ cho cả hai quá trình encode và decode, cộng với một số ràng buộc về tính không âm, cũng đã đáp ứng được cho bài toán reconstruction.

Được phân biệt với các thuật toán Auto-Encoder khác thông qua ba tính chất:

(1) Ma trận cơ sở $\boldsymbol{W}$ không âm

(2) Ma trận mã hóa $\boldsymbol{H}$ không âm

(3) Encoder và Decoder là đối xứng, nghĩa là: ma trận cơ sở được sử dụng thông qua cả hai quá trình encode và decode.

4.2.1 Decoder

Như đã trình bày ở mục 3. Phương thức giải mã (decode) thực hiện việc khôi phục dữ liệu gốc thông qua ma trận cơ sở $\boldsymbol{W} \in \mathbb{R}_{+}^{n \times r}$ và ma trận mã hóa $\boldsymbol{H} \in \mathbb{R}_{+}^{r \times m}$ sao cho $\boldsymbol{X} \approx \boldsymbol{W}\boldsymbol{H}$. Như vậy, từ đây ta có thể viết lại hàm mục tiêu để tiện cho việc so sánh với mô hình NMF về sau, dưới dạng:

$$\mathcal{L} = \|\boldsymbol{X} - \boldsymbol{W}\boldsymbol{H}\|^2$$

4.2.2 Encoder

Phương thức mã hóa (encode) biến đổi dữ liệu gốc sang dạng latent representation thông qua ma trận cơ sở $\boldsymbol{W}$, nghĩa là: $\boldsymbol{H} = \boldsymbol{W}^{T}\boldsymbol{X}$. Một cách ngầm hiểu, ta mong muốn $\boldsymbol{W}$ có khả năng tiết lộ về cấu trúc tổng quát của dữ liệu.

Để ma trận $\boldsymbol{W}$ thỏa mãn yêu cầu của bài toán, cần có một số ràng buộc. Đầu tiên, giá trị của $\boldsymbol{W}$ nên là các giá trị dương. Thứ hai, các giá trị trên dòng của ma trận $\boldsymbol{W}$ phải *thưa* (sparse). Để thực hiện các tính chất này, một cách đơn giản là phương thức mã hóa và giải mã được sử dụng chung ma trận cơ sở $\boldsymbol{W}$. Một cách

tự nhiên, ràng buộc về tính đối xứng này đảm bảo được tính trực giao yếu (soft orthogonality) của ma trận $\boldsymbol{W}$.

Đặc biệt, khi hàm mục tiêu được tối ưu, ta có:

$$\boldsymbol{X} \approx \boldsymbol{W}\boldsymbol{H} \tag{1}$$
$$\boldsymbol{H} \approx \boldsymbol{W}^T\boldsymbol{X} \tag{2}$$

Từ (1) và (2), ta được

$$\boldsymbol{X} \approx \boldsymbol{W}\boldsymbol{W}^T\boldsymbol{X}$$

Từ đây, dẫn đến ràng buộc về tính trực giao của $\boldsymbol{W}$ được thỏa mãn. Đây cũng là hàm ý quan trọng nhất đối với cấu trúc đối xứng như NSAE. Tóm lại, ràng buộc về tính đối xứng giữa hai quá trình mã hóa và giải mã cùng với ràng buộc không âm đã đảm bảo được ràng buộc về độ thưa (sparsity) đối với ma trận $\boldsymbol{W}$.

4.2.3 Loss function and Opminization

Hàm mục tiêu có thể viết dưới dạng

$$\mathcal{L}(\boldsymbol{W}, \boldsymbol{H}) = \|\boldsymbol{X} - \boldsymbol{W}\boldsymbol{H}\|^2 + \|\boldsymbol{H} - \boldsymbol{W}^T\boldsymbol{X}\|^2 \tag{5-3}$$
$$\text{s.t} \quad \boldsymbol{W} \geq 0, \boldsymbol{H} \geq 0$$

Với $\|.\|^2$ là bình phương của chuẩn L_2 (hoặc có thể sử dụng chuẩn Frobenius. Chúng ta có quy tắc cập nhật ma trận cơ sở $\boldsymbol{W}$ dưới dạng

$$\boldsymbol{W} \leftarrow \boldsymbol{W} * \frac{\boldsymbol{X}\boldsymbol{Z}^T}{\boldsymbol{W}\boldsymbol{Z}\boldsymbol{Z}^T + \boldsymbol{X}\boldsymbol{X}^T\boldsymbol{W}}$$

Tương tự, với ma trận $\boldsymbol{H}$ được cập nhật dưới dạng

$$\boldsymbol{Z} \leftarrow \boldsymbol{Z} * \frac{\boldsymbol{W}^T\boldsymbol{X}}{\boldsymbol{W}^T\boldsymbol{W}\boldsymbol{H} + \boldsymbol{H}}$$

Ta dừng thuật toán khi $\boldsymbol{W}, \boldsymbol{H}$ không có sự thay đổi lớn sau một số lần lặp.

4.3 Connection with Non-negative Matrix Factorization

Phương pháp encoder-decoder vừa được trình bày ở mục 5.2 được gọi là một dạng của Matrix Factorization (MF). Từ đây, ta cần làm rõ một vài điểm khác biệt giữa các thuật toán Matrix Factorization bao gồm dạng cơ bản và cả dạng

Non-negative Matrix Factorization (NMF).

Trước hết, MF là một thuật toán hiệu quả đối với việc thể hiện dữ liệu bằng phương thức reconstruct dữ liệu gốc. Sự khác biệt giữa các thuật toán matrix factorization nằm ở hai ma trận cơ sở $\boldsymbol{W}$ và ma trận mã hóa $\boldsymbol{H}$. NMF được phân biệt bằng các điều kiện ràng buộc các ma trận trên là ma trận có các phần tử không âm. Ràng buộc này dẫn đến khả năng học đặc trưng từng phần (part-based representation) của thuật toán, vì chúng cho phép các giá trị dương. Tuy nhiên, ràng buộc về tính không âm vẫn chưa đảm bảo được tính *thưa* của các giá trị trên $\boldsymbol{W}$. Thông thường, ta phải thêm các ràng buộc về tính thưa cho NMF để thỏa yêu cầu bài toán như là L_1 *regularization.*

Như vậy, với các thuật toán NSAE đã trình bày ở mục 5.1 hoặc 5.2. Việc xấp xỉ thuật toán NMF nằm ở chỗ ràng buộc về hàm mục tiêu, nếu (5-2) ta bỏ đi phần chuẩn hóa $\lambda \sum_{ij} \boldsymbol{H}_{ij}$ thì lúc này, thuật toán hoạt động gần như là khá giống với NMF.

5 References

Tài liệu

[1] D.D.Lee & H. Seung, *Algorithms for Non-negative Matrix Factorization*, Proceedings of the 13th International Conference on Neural Information Processing SystemsJanuary 2000.

[2] Patrik O. Hoye, *Non-negative Matrix Factorization with Sparseness Constraints*, Journal of Machine Learning Research 5 (2004) 1457–1469.

[3] D.D.Lee & H. Seung, *Learning the parts of objects by non-negative matrix factorization*, Computer Science, Medicine, Physics - Nature DOI:10.1038/44565, 1999.

[4] Andre Lemme, René Felix Reinhart, Jochen Jakob Steil, *Online learning and generalization of parts-based image representations by Non-Negative Sparse Autoencoders*, Neural networks: the official journal of the International Neural Network Society 33:194-203, May 2012.

[5] B.Sun, H.Shen, J.Gao, W.Ouyang, X.Cheng, *A Non-negative Symmetric Encoder-Decoder Approach for Community Detection*, Proceedings of the 2017

ACM on Conference on Information and Knowledge ManagementNovember 2017.

[6] Andre Lemme, René Felix Reinhart, Jochen Jakob Steil, *Efficient online learning of a non-negative sparse autoencoder*, ESANN 2010, 18th European Symposium on Artificial Neural Networks, Bruges, Belgium, April 28-30, 2010, Proceedings, January 2010.

[7] Andrew Ng, *Sparse Autoencoder - Lecture notes*, Standford University.

[8] Patrik O. Hoyer, *NON-NEGATIVE SPARSE CODING*, IEEE Workshop on Neural Networks for Signal Processing) Martigny, Switzerland, 2002.

[9] Qinxue Meng, Daniel Catchpoole, David Skillicorn and Paul J. Kennedy, *"Relational Autoencoder for Feature Extraction"*

[10] Sebastian Mika, Bernhard Scholkopf, Alex Smola Klaus-Robert Muller, Matthias Scholz, Gunnar Riitsch, *"Kernel PCA and De-Noising in Feature Spaces"* GMD FIRST, Rudower Chaussee 5, 12489 Berlin, Germany

[11] Autoencoder - Wikipedia, `https://en.wikipedia.org/wiki/Autoencoder`

[12] Lecture of Machine Learning course of Doina Precup, *Dimensionality reduction. PCA. Kernel PCA.*
`https://www.cs.mcgill.ca/ dprecup/courses/ML/Lectures/ml-lecture13.pdf`