HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN 1

TS. Đặng Minh Tuấn

Word2Vec - Word Embedding

Hà Nội – 2019

Mục lục

Mở đầu

Mô hình không gian vector được sử dụng rất nhiều để trích rút thông tin từ văn bản, khi chuyển các văn bản thành vector để so sánh độ tương đồng bằng độ đo cosine. Mặc dù ý tưởng vector hóa cho văn bản đã ra đời từ rất lâu rồi, nhưng ý tưởng vector hóa các từ thì mới xuất hiện gần đây do Tomas Mikolov đề xuất vào năm 2013.

Word embedding hay wordToVector (word2vec) là một kỹ thuật biểu diễn các từ (word) trong một bộ từ vựng thành các vector trong một không gian có số chiều nhỏ hơn kích thước của bộ từ vựng rất nhiều (50-300). Cả hai mô hình CBOW và skip-gram đều do Tomas Mikolov người Czech, trước làm việc tại Google hiên nay chuyển sang làm việc cho Facebook phát triên vào năm 2013 [6].

Mikolov chỉ ra rằng mô hình word2vec cho phép xác định được nhiều mức độ tương đồng giữa các từ. Trong [4], Mikolov chỉ ra mức độ tương đồng ngữ nghĩa và cú pháp có thể thu được bằng các phép toán với vector, ví dụ ‘‘Anh’’-‘‘Đàn ông’’+‘‘Phụ nữ’’=‘‘Chị".

Word2vec có thể được sử dụng làm cơ sở nền tảng cho nhiều bài toán xử lý ngôn ngữ tự nhiên: phân loại từ, phân loại câu, phân loại văn bản, tóm tắt văn bản, và là đầu vào cho nhiều bài toán sử dụng mạng neuron như CNN, RNN...

Tài liệu này được trình bày đầy đủ và chi tiết nhất có thể về các ý tưởng và kiến thức nền tảng về mạng Neuron được sử dụng trong word2vec. Trong tài liệu có nhiều phần chứng minh công thức cũng như mô tả về phần mềm thực hiện các thuật toán liên quan. Tài liệu cung cấp cho người đọc toàn bộ kiến thức cơ bản về vector hóa các từ cũng như cách thức triển khai một mạng neuron.

Một số mô hình có thể cho kết quả tốt hơn như GloVe [8] và Fasttext [1].

Hà Nội, ngày 06 tháng 03 năm 2019

TS.Đặng Minh Tuấn

Chương 1

Mạng Neuron lan truyền ngược Back Propagation

1.1 Thuật toán cho một neuron

Hình 1.1 là một neuron, $\{x_1, \ldots, x_K\}$ là các giá trị đầu vào; $\{w_1, \ldots, w_K\}$ là các trọng số; y là giá trị đầu ra; f là hàm link(activation/decision/transfer). Giá trị đầu ra sẽ được biểu diễn như sau:

$$y = f(u) \tag{1.1}$$

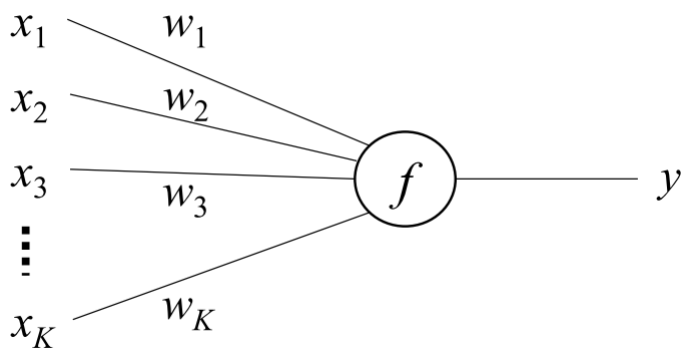
Trong đó u là một giá trị được gọi là net input hay new input của neuron và được định nghĩa như sau:

$$u = \sum_{i=1}^{K} w_i x_i \tag{1.2}$$

Biểu diễn dưới dạng vector như sau:

$$u = \mathbf{w}^T \mathbf{x} \tag{1.3}$$

Ở đây chúng ta bỏ qua bias, muốn thêm giá trị này chúng ta chỉ cần thêm x_0 và có giá trị là hằng số là 1.



Hình 1.1: Mô hình một neuron

1.1.1 Heaviside step function (unit step function)

Hàm link có dạng:

$$f(u) = \begin{cases} 1 & \text{nếu } u > 0 \\ 0 & \text{các trường hợp còn lại} \end{cases} \tag{1.4}$$

Neuron có hàm link dạng này được gọi là perceptron. Công thức cập nhật được định nghĩa như sau:

$$\mathbf{w}^{(new)} = \mathbf{w}^{(old)} - \eta \cdot (y - t) \cdot \mathbf{x} \tag{1.5}$$

Trong đó t là nhãn (gold standard) và η là learning rate ($\eta > 0$). Để ý rằng peceptron là bộ phân loại tuyến tính nghĩa là phần biểu diễn của nó khá hạn chế, chúng ta thường cần sử dụng các hàm phức tạp hơn, và hàm link thường là phi tuyến.

1.1.2 logistic function (sigmoid function)

Hàm link có dạng:

$$\sigma(u) = \frac{1}{1 + e^{-u}} \tag{1.6}$$

Hàm sigmoid có một số thuộc tính sau:

- Giá trị đầu ra y có giá trị trong khoảng [0,1];
- Từ định nghĩa (1.6) suy ra: $\sigma(-u) = 1 - \sigma(u)$;
- Từ định nghĩa (1.6) suy ra: $\frac{d\sigma(u)}{du} = \sigma(u)\sigma(-u)$

Chúng ta sử dụng thuật toán stochastic gradient descent, đầu tiên cần phải định nghĩa hàm sai số (error):

$$E = \frac{1}{2}(y - t)^2 \tag{1.7}$$

Tính đạo hàm của E theo w_i ta có:

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial u} \cdot \frac{\partial u}{\partial w_i} \tag{1.8}$$

$$= (y - t) \cdot y(1 - y) \cdot x_i \tag{1.9}$$

Bởi vì $\frac{\partial y}{\partial u} = y(1 - y)$ vì $y = f(u) = \sigma(u)$. Sau khi có đạo hàm chúng ta có thể tính được stochastic gradient descent:

$$\mathbf{w}^{(new)} = \mathbf{w}^{(old)} - \eta \cdot (y - t) \cdot y(1 - y) \cdot \mathbf{x} \tag{1.10}$$

1.1.3 softmax function

Hàm softmax là một dạng của cross entropy, là đại lượng thường để đo khoảng cách giữa 2 phân bố xác suất được định nghĩa bằng công thức:

$$H(y, p) = -\sum_i y_i log(p_i) \tag{1.11}$$

Cross entropy cũng thường được sử dụng để thay thế phương pháp bình phương nhỏ nhất của sai số, và cũng thường để sử dụng đo sai lệch giữa 2 phân bố xác suất.

$$\text{softmax}(x)_i = p_i = \frac{e^{x_i}}{\sum_{k=1}^{N} e^{x_k}} \tag{1.12}$$

Để hàm softmax có tính ổn định (numerically stable) thường chúng ta sẽ normalize các giá trị trong vector bằng cách nhân cả tử và mẫu với mộ hằng số C:

$$p_i = \frac{e^{x_i}}{\sum_{k=1}^{N} e^{x_k}} = \frac{Ce^{x_i}}{C\sum_{k=1}^{N} e^{x_k}} = \frac{e^{x_i + \log(C)}}{\sum_{k=1}^{N} e^{x_k + \log(C)}}$$

Có thể chọn $\log(C)$ bất kỳ tuy nhiên người ta thường chọn $\log(C) = -\max(x)$.

Một số tính chất của hàm softmax:

- Có giá trị nằm trong khoảng [0,1].
- Tổng các giá trị bằng 1.
- Đạo hàm của hàm softmax:

$$\frac{\partial p_i}{\partial x_j} = \frac{\partial \frac{e^{x_i}}{\sum_{k=1}^{N} e^{x_k}}}{\partial x_j} \tag{1.13}$$

Chúng ta biết rằng với $f(x) = \frac{g(x)}{h(x)}$, ta sẽ có: $f'(x) = \frac{g'(x)h(x) - h'(x)g(x)}{h(x)^2}$. Trong trường hợp này $g(x) = e^{x_i}$ và $h(x) = \sum_{k=1}^{N} e^{x_k}$. Tiếp theo ta cũng có $\frac{\partial e^{x_i}}{\partial x_j} = e^{x_i}$ nếu $i = j$, ngược lại $\frac{\partial e^{x_i}}{\partial x_j} = 0$.

– Với $i = j$:

$$\begin{aligned}\frac{\partial p_i}{\partial x_j} = \frac{\partial \frac{e^{x_i}}{\sum_{k=1}^{N} e^{x_k}}}{\partial x_j} &= \frac{e^{x_i}\sum_{k=1}^{N} e^{x_k} - e^{x_j}e^{x_i}}{\left(\sum_{k=1}^{N} e^{x_k}\right)^2} \\ &= \frac{e^{x_i}\left(\sum_{k=1}^{N} e^{x_k} - e^{x_j}\right)}{\left(\sum_{k=1}^{N} e^{x_k}\right)^2} \\ &= \frac{e^{x_i}}{\left(\sum_{k=1}^{N} e^{x_k}\right)^2} \times \frac{\left(\sum_{k=1}^{N} e^{x_k} - e^{x_j}\right)}{\left(\sum_{k=1}^{N} e^{x_k}\right)^2} \\ &= p_i(1 - p_i)\end{aligned}$$

– Với $i \neq j$:

$$\begin{aligned}\frac{\partial p_i}{\partial x_j} = \frac{\partial \frac{e^{x_i}}{\sum_{k=1}^{N} e^{x_k}}}{\partial x_j} &= \frac{0 - e^{x_j}e^{x_i}}{\left(\sum_{k=1}^{N} e^{x_k}\right)^2} \\ &= \frac{-e^{x_j}}{\sum_{k=1}^{N} e^{x_k}} \times \frac{e^{x_i}}{\sum_{k=1}^{N} e^{x_k}} \\ &= -p_j \cdot p_i\end{aligned}$$

Sử dụng ký hiệu Kronecker delta $\delta_{ij} = \begin{cases} 1 & \text{nếu } i = j \\ 0 & \text{nếu } i \neq j \end{cases}$ Ta sẽ có:

$$\frac{\partial p_i}{\partial x_j} = p_i(\delta_{ij} - p_j) \tag{1.14}$$

- Hàm mất mát khi sử dụng với softmax:

$$E = -\sum_i y_i \log(p_i) \tag{1.15}$$

Đạo hàm hàm E theo x_i sẽ là:

$$\begin{aligned}
\frac{\partial E}{\partial x_i} &= -\sum_k y_k \frac{\partial \log(p_k)}{\partial x_i} \\
&= -\sum_k y_k \frac{\partial \log(p_k)}{\partial p_k} \times \frac{\partial p_k}{\partial x_i} \\
&= -\sum_k y_k \frac{1}{p_k} \times \frac{\partial p_k}{\partial x_i} \\
&= -y_i(1-p_i) - \sum_{k \neq i} y_k \frac{1}{p_k}(-p_k \cdot p_i) \\
&= -y_i + y_i p_i + \sum_{k \neq i} y_k \cdot p_i \\
&= p_i \left(y_i + \sum_{k \neq i} y_k \right) - y_i \\
&= p_i - y_i
\end{aligned} \tag{1.16}$$

Trong biến đổi ở trên chúng ta sử dụng công thức (1.14) và $\left(y_i + \sum_{k \neq i} y_k\right) = \sum_k y_k = 1$.

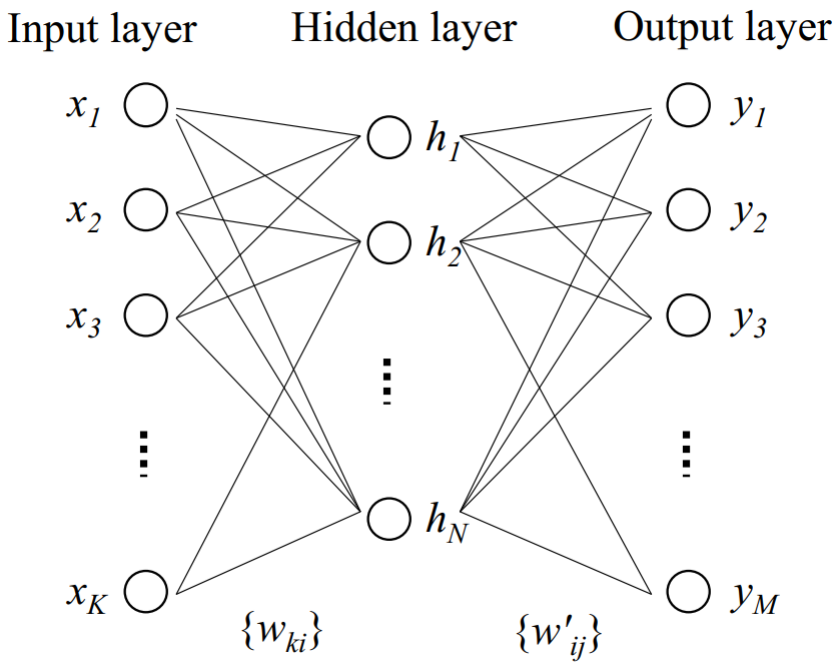
1.2 Thuật toán cho nhiều lớp

Trong hình 1.2, mạng neuron có nhiều lớp với các quy ước:

- $\{x_k\} = \{x_1, \ldots, x_K\}$: giá trị của input layer;
- $\{h_i\} = \{h_1, \ldots, h_N\}$: giá trị của hidden layer;
- $\{y_j\} = \{y_1, \ldots, y_M\}$: giá trị của output layer;
- k, i, j là các chỉ số của lớp input, hidden và output;
- u_i, u'_j lần lượt là net input của lớp hidden và lớp output;
- w_{ki}, w'_{ij} là trọng số của lần lượt giữa input→hidden và hidden→output.

Chúng ta giả thiết hàm link sử dụng sẽ là $\sigma(u)$, như vậy sẽ có giá trị h_i được tính như sau:

$$h_i = \sigma(u_i) = \sigma\left(\sum_{k=1}^{K} w_{ki} x_k\right) \tag{1.17}$$



Hình 1.2: Mô hình mạng neuron nhiều lớp

Tương tự như vậy cho phần tử y_j trong lớp output có đầu ra là:

$$y_j = \sigma(u'_j) = \sigma\left(\sum_{i=1}^{N} w'_{ij} h_i\right) \tag{1.18}$$

Hàm sai số lỗi được định nghĩa như sau:

$$E(\mathbf{x}, \mathbf{t}, \mathbf{W}, \mathbf{W}') = \frac{1}{2}\sum_{j=1}^{M} (y_j - t_j)^2 \tag{1.19}$$

Trong đó:

- $W = \{w_{ki}\}$: ma trận trọng số input-hidden có kích cỡ là $K \times N$;
- $W' = \{w'_{ij}\}$: ma trận trọng số hidden-output có kích cỡ là $N \times M$;
- $\mathbf{t} = \{t_1, \ldots, t_M\}$: vector nhãn (gold-standard labels) của đầu ra có M chiều.

Để có các cập nhật cho w_{ki}, w'_{ij} chúng ta cần tính đạo hàm của hàm lỗi E theo các biến tương ứng. Trình tự tính toán sẽ đi từ lớp output qua bên trái, đến lớp hidden rồi cuối cùng là input, ở mỗi lớp chúng ta sẽ tính theo thứ tự: đạo hàm của hàm lỗi, net input và sau là trọng số tương ứng.

Bắt đầu từ lớp output, ta có đạo hàm lỗi E theo y_j như sau:

$$\frac{\partial E}{\partial y_j} = y_j - t_j \tag{1.20}$$

Tiếp theo tính đạo hàm hàm lỗi theo net input:

$$\frac{\partial E}{\partial u'_j} = \frac{\partial E}{\partial y_j} \cdot \frac{\partial y_j}{\partial u'_j} = (y_j - t_j) \cdot y_j(1 - y_j) := \mathrm{EI}'_j \tag{1.21}$$

Bước thứ 3 là tính đạo hàm hàm lỗi theo trọng số giữa lớp hidden→output:

$$\frac{\partial E}{\partial w'_{ij}} = \frac{\partial E}{\partial u'_j} \cdot \frac{\partial u'_j}{\partial w'_{ij}} = \mathrm{EI}'_j \cdot h_i \tag{1.22}$$

Đến bước này chúng ta có thể tính được cập nhật cho trọng số giữa lớp hidden→output:

$$w'^{(new)}_{ij} = w'^{(old)}_{ij} - \eta \cdot \frac{\partial E}{\partial w'_{ij}} \tag{1.23}$$

$$= w'^{(old)}_{ij} - \eta \cdot \mathrm{EI}'_j \cdot h_i \tag{1.24}$$

Chúng ta lặp lại 3 bước trên cho output của lớp hidden, lưu ý rằng đầu ra của lớp hidden liên quan đến tất cả các thành phần trong lớp output.

$$\frac{\partial E}{\partial h_i} = \sum_{j=1}^{M} \frac{\partial E}{\partial u'_j} \cdot \frac{\partial u'_j}{\partial h_i} = \sum_{j=1}^{M} \mathrm{EI}'_j \cdot w'_{ij} \tag{1.25}$$

Lặp lại bước thứ 2 để tính đạo hàm hàm lỗi đối với net input của lớp hidden ta có:

$$\frac{\partial E}{\partial u_i} = \frac{\partial E}{\partial h_i} \cdot \frac{\partial h_i}{\partial u_i} = \sum_{j=1}^{M} \mathrm{EI}'_j \cdot w'_{ij} \cdot h_i(1 - h_i) := \mathrm{EI}_i \tag{1.26}$$

Lặp lại bước thứ 3, tính đạo hàm hàm lỗi theo trọng số giữa lớp input→hidden:

$$\frac{\partial E}{\partial w_{ki}} = \frac{\partial E}{\partial u_i} \cdot \frac{\partial u_i}{\partial w_{ki}} = \mathrm{EI}_i \cdot x_k \tag{1.27}$$

Cuối cùng chúng ta có cập nhật trọng số giữa lớp input→hidden:

$$w_{ki}^{(new)} = w_{ki}^{(old)} - \eta \cdot \mathrm{EI}_i \cdot x_k \tag{1.28}$$

Có thể để ý rằng EI'_j của lớp này có thể được sử dụng cho lớp trước đó.

Chương 2

Word to Vector (Word embedding)

Deep Learning ảnh hưởng đến các lĩnh vực liên quan tới Machine Learning, và NLP (Natural Language Proccessing) cũng không phải là ngoại lệ. Nhiều bài toán trong NLP như Name Entity Recognition, POS Tagging, Machine Translation, ... đã có được sự phát triển vượt trội nhờ Deep Learning.

Trong mô hình Deep Learning, input đầu vào thường sẽ là một vector (hoặc ma trận) chứa các giá trị số. Đối với NLP, dữ liệu chúng ta có thường là dạng chuỗi ký tự. Con người nhìn chuỗi ký tự này và xử lý nội dung ở dạng là các từ được ghép nối với nhau. Câu hỏi được đặt ra tương tự đối với máy tính. Làm thế nào để biểu diễn một chuỗi ký tự thành các con số để máy tính xử lý, đặc biệt trong các mô hình Deep Learning khi mà dữ liệu đầu vào đóng vai trò cực kỳ quan trọng để xây dựng được mô hình hiệu quả. Cần nhớ rằng dữ liệu training càng phong phú thì khả năng nhận dạng của Deep Learning càng cao và dữ liệu training tồi sẽ dẫn đến mô hình Deep Learning có khả năng nhận dạng kém.

Quay lại với câu hỏi làm sao biểu diễn một chuỗi các ký tự trong text thành input đầu vào của mô hình Deep Learning, chúng ta sẽ cùng nhau xem xét các cách biểu diễn từ đơn giản đến phức tạp. Đối với 1 chuỗi text, thông thường người

ta sẽ phân nhỏ chuỗi text đó ra và biểu diễn từng thành phần đơn vị của chuỗi text đó. Tùy theo cách chọn "đơn vị" mà chúng ta có cách biểu diễn khác nhau:

2.1 Đơn vị là "Ký tự" (Character)

Cách biểu diễn này khá đơn giản bởi vì thứ nhất: các ký tự không có sự liên quan về mặt ngữ nghĩa với nhau, miễn sao chúng ta có thể phân biệt được các ký tự là được. Hơn nữa số lượng ký tự chúng ta có là không lớn cho nên không lo ngại về vấn đề không gian lưu trữ và xử lý. Đơn giản nhất là chúng ta có thể dùng mã trong charecter set để đại diện cho ký tự đó. Trong trường hợp cần biểu diễn mỗi ký tự theo dạng vector thì chúng ta có thể sử dụng **one-hot vector**.

One-hot vector là một vector có toàn bộ giá trị là 0 trừ tại một vị trí đặc biệt nào đó thì giá trị sẽ là 1. Ví dụ như chúng ta chỉ có 4 ký tự: ABCD, chúng ta sẽ có các one-hot vector tương ứng với từng ký tự như sau:

A: [1, 0, 0, 0]
B: [0, 1, 0, 0]
C: [0, 0, 1, 0]
D: [0, 0, 0, 1]

Số chiều của one-hot vector sẽ phụ thuộc vào số lượng phần tử có trong tập hợp mà chúng ta cần biểu diễn. Trong ví dụ trên vì tập hợp chúng ta chỉ có 4 phần tử ('A', 'B', 'C', 'D') nên vector của chúng ta là 4 chiều.

Ngoài ra, chúng ta còn có thể khởi tạo vector của mỗi ký tự có các giá trị là random, khi đó xác suất 2 ký tự có vector biểu diễn giống nhau là gần như bằng 0. Điều quan trọng nhất cần để ý đó là: giữa các ký tự không có mối liên hệ ngữ nghĩa nào rõ ràng nên miễn sao vector biểu diễn khác nhau là được.

2.2 Đơn vị là "Token" (Word)

Tương tự như đối với ký tự, chúng ta cũng có thể áp dụng các cách biểu diễn đã được nêu ra ở trên.

Sử dụng 1 con số để dại diện cho Token đó: Cách này đơn giản nhưng không hiệu quả vì giữa các từ nó có mối quan hệ ngữ nghĩa (đồng nghĩa, khác nghĩa, biến thể, ...), nếu chỉ dùng 1 con số nguyên thì không có biểu diễn được mối quan hệ đó. Hơn nữa các mô hình Deep Learning thường yêu cầu dạng input đầu vào dạng vector thì không dùng cách này được.

*** Sử dụng one-hot vector:** Tương tự như ký tự, chúng ta có thể xem mỗi token là một phần tử trong tập hợp toàn bộ token có thể có của một ngôn ngữ,

ví dụ tiếng Anh có khoảng 1 triệu words, mỗi word sẽ được biểu diễn bằng một one-hot vector có 1 triệu chiều. Nhược điểm của phương pháp này là số lượng chiều của một vector rất lớn nên ảnh hưởng đến việc xử lý cũng như lưu trữ. Ví dụ như tiếng Anh khoảng 1 triệu từ, mỗi từ là vector 1 triệu chiều. Giả sử đoạn text tiếng Anh khoảng 1000 từ thì chúng ta đã phải sử dụng tới ma trận 1000 dòng $\times$ 1 triệu cột để biểu diễn cho input. Hơn nữa, biểu diễn theo dạng one-hot vector như thế này vẫn không giải quyết được việc biểu diễn mối liên hệ giữa các từ với nhau.

*** Sử dụng các vector random:** Nếu dùng vector random thì số chiều chúng ta cần dùng sẽ ít hơn nhiều so với dùng one-hot vector. Ví dụ như các bạn có 1 triệu từ thì chỉ cần trong không gian 3 chiều chúng ta cũng đã có thể biểu diễn được tất cả các từ đó, mỗi từ là một điểm trong không gian 3 chiều. Như one-hot vector, chúng ta cũng không hiểu mối liên hệ giữa các từ thông qua vector random như thế này.

*** Dùng Word Embedding:** Đây được xem là một cách biểu diễn tốt nhất cho các token trong text. Kỹ thuật này này không những biểu diễn mỗi token bằng một vector với số chiều thấp mà còn cho thấy được sự liên hệ ngữ nghĩa giữa các vector đó. Bằng việc sử dụng các vector này như là input cho mô hình Deep Learning, mô hình sẽ có khả năng học tốt hơn và khả năng nhận dạng cũng tăng lên.

Word Embedding (hay Word Vector) được sinh ra như thế nào?. Mọi chuyện bắt đầu vào khoảng năm 2000 khi ông Bengio viết series bài báo Neural probabilistic language models để nhằm giảm số chiều của vector biểu diễn một word theo ngữ cảnh bằng cách dùng machine learning. Sau đó thì nhiều nhà khoa học khác cũng đề xuất nhiều kỹ thuật khác nhau để tìm ra cách biểu diễn tốt nhất cho word. Tuy nhiên trong giai đoạn trước 2010 thì không có nhiều điểm bứt phá vì tốc độ để học các word vector là rất lâu trong khi performance không cải thiện nhiều. Tuy nhiên, năm 2013 thì nhóm nghiên cứu của Tomas Mikolov đến từ Google đã cung cấp bộ công cụ word2vec [6] với khả năng học word vector rất nhanh cũng như performance khá tốt, khi đó thì word embedding càng được sử dụng rộng rãi hơn nhiều. Có nhiều cách học word embedding nhưng mình sẽ giới thiệu một ví dụ sử dụng mạng Neural để học word embedding của các từ.

Đối với bài toán tìm ra word vector, dữ liệu đầu vào sẽ là một văn bản, xem như là tập hợp các từ (word). Đầu tiên, tương ứng với mỗi word thì chúng ta sẽ khởi tạo một vector random với số chiều được chỉ định (giả sử 100). Sau khi đã có vector random, việc tiếp theo là thực hiện quá trình điều chỉnh vector của các từ này để sao cho chúng có thể biểu diễn được liên hệ giữa các từ có quan hệ với nhau.

Giả sử chúng ta có câu văn sau: *Con mèo trèo cây cau.* Tương ứng với mỗi từ trong câu này, chúng ta sẽ khởi tạo một vector random với số chiều được quy định trước (ví dụ số chiều = 50). Người ta sử dụng một mang neuron và dùng mạng neuron này để điều chỉnh dần dần các vector của các từ sao cho chúng thỏa mãn

một số điều kiện nào đó. Câu hỏi đặt ra ở đây: Điều kiện đó là gì?

Để trả lời câu hỏi này thì trước hết chúng ta cần quan tâm tới một đặc điểm của ngôn ngữ, đó là những từ có mối liên hệ với nhau thường sẽ xuất hiện trong những ngữ cảnh khác nhau. Ví dụ từ "trái" và "phải" có thể xem là có mối liên quan nào đó với nhau vì nó đều dùng chỉ phương hướng và nó thường xuất hiện trong những mẫu câu giống nhau. Ví dụ mình có các câu:

"Chạy xe phía bên trái", "Chạy ở bên phải", "Bên trái có vẻ rộng hơn", "Bên phải có một ngôi nhà".

Bạn có để ý thấy các từ nằm xung quanh của từ "trái" và "phải" đều khá là giống nhau không? Đó chính là nguyên tắc học của word2vec. Nó dựa vào những từ xung quanh của một từ nào đó để điều chỉnh vector của từ đó sao cho hợp lý.

Quay trở lại với ví dụ ban đầu: Con mèo trèo cây cau. Chúng ta sử dụng 1 mạng Neural để xem câu này có hợp lệ hay không. Giả sử thay từ "trèo" bằng từ "ngủ", rõ ràng chúng ta sẽ có 1 câu hoàn toàn vô nghĩa và hầu như không bao giờ xuất hiện trong văn bản bình thường: "con mèo ngủ cây cau". Bằng cách thay từ "trèo" bằng từ "ngủ" và nói cho mạng Neural biết rằng câu mới sinh ra là không hợp lệ, mạng Neural sẽ phải điều chỉnh các tham số trong mạng của nó một cách hợp lý để đưa ra được output đúng như chúng ta mong muốn (tức là "không hợp lệ"). Thông thường thì input vào mạng Neural sẽ không phải là nguyên một câu mà chỉ là 1 cụm từ của câu có độ dài dựa theo một tham số gọi là window size. Ví dụ window size = 3 thì chúng ta sẽ có các cụm từ: "con mèo trèo", "mèo trèo cây", "trèo cây cau". Với mỗi windows size thì chúng ta có thể thay 1 từ nào đó bằng 1 từ random khác để có các cụm câu vô nghĩa dùng để train mạng Neural (bởi vì khi train mạng Neural thì phải vừa cho input với nhãn "hợp lệ" và cũng phải có input với nhãn "không hợp lệ" nhằm giúp cho mạng Neural đó phân biệt cho đúng).

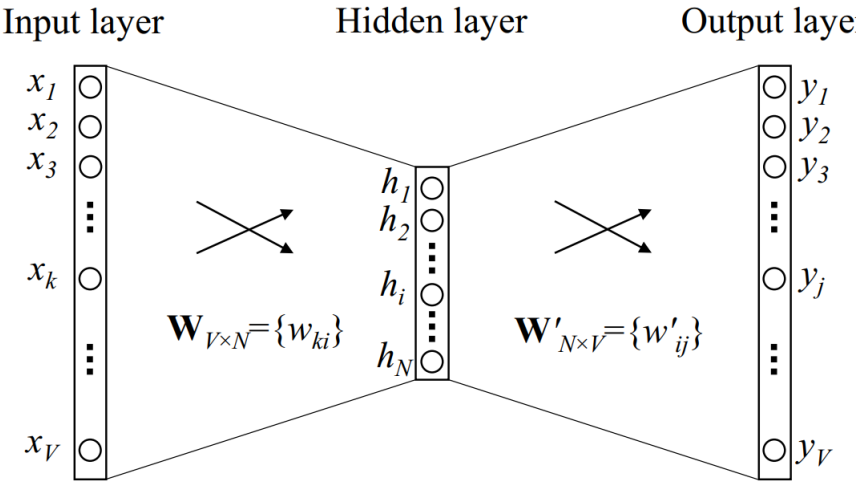
Nhờ việc train mạng Neural trên một số lượng text cực lớn, ví dụ như toàn bộ text trên vnexpress chẳng hạn, thì vector của mỗi từ sẽ được điều chỉnh càng chính xác và những từ có liên quan nhau cũng sẽ xuất hiện ở gần nhau hơn. Khi đó giữa các từ có mối liên hệ với nhau rất thú vị. Chẳng hạn chúng ta lấy vector của từ "king" đem cộng với vector của từ "man" rồi trừ đi vector của từ "women" thì chúng ta sẽ được một vector mà từ gần giống vector đó nhất là "queen".

2.3 Continuous Bag-of-Word Model (CBOW)

2.3.1 Mô hình một từ đơn

Bắt đầu bằng mô hình một từ đơn CBOW do Mikolov và đồng nghiệp đề xuất năm 2013 [5]. Mô hình cho phép đoán word có xác suất xuất hiện cao nhất trong ngữ cảnh (context) cho trước, hoặc cho ngữ cảnh cụ thể thì có thể đoán xác suất

tốt nhất có thể phù hợp với ngữ cảnh đã cho. Mô hình CBOW không quan tâm thứ tự của các từ, chỉ qua tâm số từ xuất hiện, khác với mô hình n-gram là mô hình quan tâm đặc biệt thứ tự của các từ.



Hình 2.1: Mô hình một từ đơn

* *Ký hiệu*

- V: Kích cỡ từ vựng.
- N: Kích cỡ của lớp ẩn.
- $\{x_1, \ldots, x_v\}$: đầu vào là *one-hot vector*, chỉ có từ đãng xét thì tại vị trí của vector có giá trị 1, các vị trị khác có giá trị 0.
- $\mathbf{W}$: Ma trận trọng số có kích cỡ là $V \times N$.
- $\mathbf{v}_w$: là một hàng có cỡ là N của ma trận $\mathbf{W}$ là vector tương ứng với từ (word) của lớp đầu vào (input). Hàng i của $\mathbf{W}$ là $\mathbf{v}_w^T$. Với từ đã cho có $x_k = 1$ và $x_{k'} = 0$ với $k \neq k'$ ta có.

$$\mathbf{h} = \mathbf{W}^T\mathbf{x} = \mathbf{W}^T_{(k,\cdot)} := \mathbf{v}^T_{w_I} \tag{2.1}$$

- $\mathbf{W}' = \{w'\}$: Ma trận trong số từ lớp ẩn (hiden) đến lớp đầu ra (output) có kích thước là $N \times V$.
- u_j: điểm (score) hay còn gọi là net input cho từng từ trong từ vựng được tính như sau:

$$u_j = \mathbf{v}'^T_{w_j}\mathbf{h} \tag{2.2}$$

- $\mathbf{v}'^T_{w_j}$ là cột j của ma trận $\mathbf{W}'$.

- $p(w_j|w_I)$: Xác suất xuất hiện của word w_j đối với input word w_I còn gọi là $word_{context=}$.

- y_j: đầu ra của phần thứ j ở lớp đầu ra (output layer), đầu ra là xác suất xuất hiện của từ trong ngữ cảnh (context) đã cho. Sử dụng hàm softmax, mô hình log-linear classification có thể tính được đầu ra chính là phân bố của từ:

$$y_j = p(w_j|w_I) = \frac{e^{u_j}}{\sum_{j'=1}^{V} e^{u_{j'}}} = \frac{\exp(u_j)}{\sum_{j'=1}^{V} \exp(u'_j)} \tag{2.3}$$

  Thay (2.1) và (2.2) vào (2.3) ta có:

$$p(w_j|w_I) = \frac{\exp\left(\mathbf{v}'^{T}_{w_j}\mathbf{v}_{w_I}\right)}{\sum_{j'=1}^{V} \exp\left(\mathbf{v}'^{T}_{w'_j}\mathbf{v}_{w_I}\right)} \tag{2.4}$$

- $p(w_i)$: Xác suất của từ w_i trong training set (vocab) được định nghĩa như sau [3] (trong đó $f_r(w_i)$ là tần xuất xuất hiện bằng số lần xuất hiện trong tổng số các từ, t_h là tham số ngưỡng thường là 10^{-5}):

$$p(w_i) = 1 - \left(\sqrt{\frac{t_h}{f_r(w_i)}}\right) \tag{2.5}$$

Để ý rằng $\mathbf{v}_w$ và $\mathbf{v}'_w$ là hai vector biểu diễn cho từ w, $\mathbf{v}_w$ là hàng của ma trận $\mathbf{W}$ là ma trận trọng số từ lớp input→hiden, $\mathbf{v}'_w$ là cột của ma trận $\mathbf{W}'$ là ma trận trọng số từ lớp hiden→output. Nên $\mathbf{v}_w$ được gọi là vector đầu vào và $\mathbf{v}'_w$ vector đầu ra.

2.3.1.1 Cập nhật trọng số từ lớp hidden→output

Đối tượng để học là tối đa hóa (2.4) xác suất của từ (word) đầu ra w_O, chỉ số của lớp đầu ra tương ứng là j^* với từ đầu vào là w_I:

$$\max p(w_O|w_i) = \max y_{j^*} \tag{2.6}$$

$$= \max \log y_{j^*} \tag{2.7}$$

$$= u_{j^*} - \log \sum_{j'=1}^{V} \exp\left(u_{j'}\right) := -E, \tag{2.8}$$

Ở đây $E = -\log p(w_O|w_i)$ là hàm mất mát, chúng ta cần cực tiểu hóa E. Hàm mất mát có thể coi là trường hợp đặc biệt của độ đo cross-entropy giữa hai phân bố. Tính đạo hàm của E theo u_j ta có:

$$\frac{\partial E}{\partial u_j} = y_j - t_j := e_j \tag{2.9}$$

$t_j = 1(j = j^*)$ khi phần tử thứ j chính là từ đầu ra, và khác 0 trong các trường hợp khác. e_j là lỗi giữa đầu ra và nhãn được gán ứng với bộ dữ liệu đầu vào. Tiếp theo lấy đạo hàm theo w'_{ij} để có độ dốc trên các trọng số weight→output:

$$\frac{\partial E}{\partial w'_{ij}} = \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial w'_{ij}} = e_j \cdot h_i \tag{2.10}$$

Với learning rate: $\eta \leq 0$, cập nhật lại hệ số sẽ được tính như sau:

$$w'^{(\text{new})}_{ij} = w'^{(\text{old})}_{ij} - \eta \cdot e_j \cdot h_i \tag{2.11}$$

Hay là:

$$\mathbf{v}'^{(\text{new})}_{w_j} = \mathbf{v}'^{(\text{old})}_{w_j} - \eta \cdot e_j \cdot \mathbf{h} \quad \text{với} \quad j = 1, 2, \ldots, V \tag{2.12}$$

Trong đó vector $\mathbf{v}'_{w_j}$ là vector đầu ra của w_j.

2.3.1.2 Cập nhật trọng số từ lớp input→hidden

Sau khi cập nhật W', sau đó cập nhật tiếp W. Xét đạo hàm của E theo đầu ra cho lớp hiden.

$$\frac{\partial E}{\partial h_i} = \sum_{j=1}^{V} \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial h_i} = \sum_{j=1}^{V} e_j \cdot w'_{ij} := \text{EH}_i \tag{2.13}$$

Trong đó: h_i đầu ra thứ i của hiden layer.

$$h_i = \sum_{k=1}^{V} x_k \cdot w_{ki} \tag{2.14}$$

EH là vector N-chiều là tổng của các vector đầu ra của tất cả các word trong từ vựng. Tính đạo hàm của E theo từng phần tử của W, theo chứng minh ở (1.16) ta có:

$$\frac{\partial E}{\partial w_{ki}} = \frac{\partial E}{\partial h_i} \cdot \frac{\partial h_i}{\partial w_{ki}} = \text{EH}_i \cdot x_k \tag{2.15}$$

Có thể viết theo cách khác:

$$\frac{\partial E}{\partial \mathbf{W}} = \mathbf{x} \otimes \text{EH} = \mathbf{x} \cdot \text{EH}^T \tag{2.16}$$

$$\mathbf{v}^{(\text{new})}_{w_I} = \mathbf{v}^{(\text{old})}_{w_I} - \eta \cdot \text{EH}^T \tag{2.17}$$

Trong đó $\mathbf{v}_{w_I}$ là một hàng của $\mathbf{W}$ là input vector của từ đang xét và duy nhất hàng này cũng khác 0.

2.3.2 Mô hình đa từ

Trong mô hình đa từ của CWOB thay vì lấy nguyên input vector thì sẽ lấy giá trị trung bình của các từ đang xét.

$$\mathbf{h} = \frac{1}{C}\mathbf{W}^T(x_1 + x_2 + \ldots + x_C) \tag{2.18}$$

$$= \frac{1}{C}(\mathbf{v}_{w_1} + \mathbf{v}_{w_2} + \ldots + \mathbf{v}_{w_C})^T \tag{2.19}$$

Trong đó C là số từ đang xét, $w_1, \ldots, w_C$ là các từ trong tập đang xét. $\mathbf{v}_w$ là vector đầu vào của word w. Hàm mất mát sẽ là:

$$E = -\log p(w_O|w_{I,1}, \ldots, w_{I,C}) \tag{2.20}$$

$$= -u_{j^*} + \log \sum_{j'=1}^{V} \exp(u_{j'}) \tag{2.21}$$

$$= \mathbf{v'_{w_O}}^T \cdot \mathbf{h} + \log \sum_{j'=1}^{V} \exp(\mathbf{v'_{w_{j'}}}^T \cdot \mathbf{h}) \tag{2.22}$$

Cập nhật cho vector trọng số từ lớp hiden→output tưởng tự như với (2.12)

$$\mathbf{v'_{w_j}}^{(\text{new})} = \mathbf{v'_{w_j}}^{(\text{old})} - \eta \cdot e_j \cdot \mathbf{h} \quad \text{với} \quad j = 1, 2, \ldots, V \tag{2.23}$$

Lưu ý rằng cần phải áp dụng cho mỗi phần tử thuộc ma trận trọng số hiden-output cho mỗi lần học. Để cập nhật trọng số input→hiden, áp dụng tương tự biểu thức (2.17), ngoại trừ rằng cần áp dụng phương trình sau cho từng word khi đang xét:

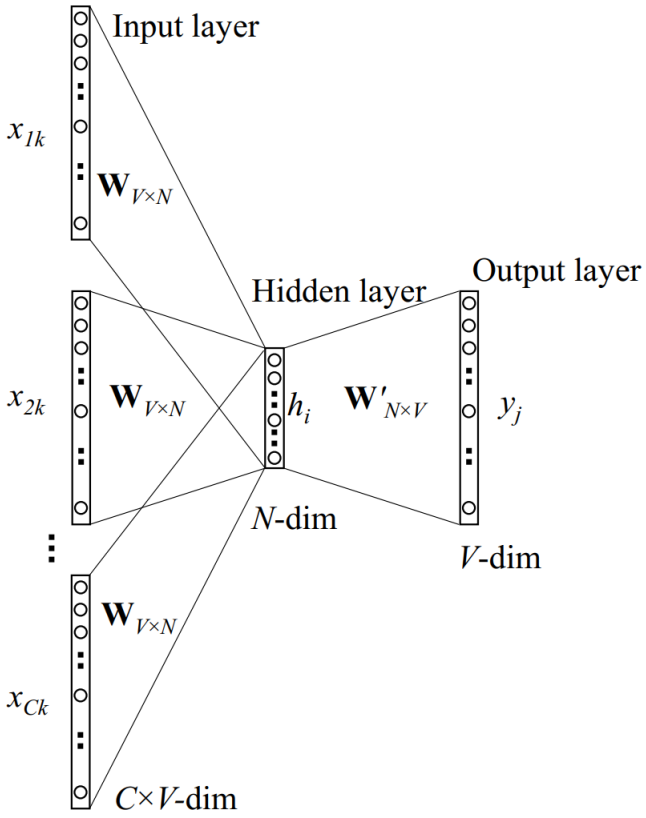
$$\mathbf{v}_{w_{I,C}}{}^{(\text{new})} = \mathbf{v}_{w_{I,C}}{}^{(\text{old})} - \frac{1}{C} \cdot \eta \cdot \text{EH}^T \tag{2.24}$$

Trong đó $\mathbf{v}_{w_{I,C}}$ là intput vector của từ thứ C trong chuỗi đang xét. $\text{EH} = \frac{\partial E}{\partial h_i}$ được mô tả tương tư như (2.13).

2.4 Skip-Gram Model

Mô hình này được giới thiệu bởi Mikolov và đồng nghiệp năm 2013 [6]. Đây là mô hình ngược lại với CBOW. Từ (word) đích nằm ở lớp input, còn các từ đang xét lại nằm ở lớp output. Chúng ta vẫn dùng ký hiệu $\mathbf{v}_{w_I}$ là input vector của một từ ở input layer. Điều đó có nghĩa là $\mathbf{h}$ chỉ đơn thuần là copy (và chuyển vị) hàng của ma trận trọng số input-hiden, $\mathbf{W}$, tương ứng với word đầu vào w_I. Chúng ta sẽ có cùng một định nghĩa với đầu ra của lớp hiden như biểu thức (2.1):

$$\mathbf{h} = \mathbf{W}^T_{(k,\cdot)} := \mathbf{v}^T_{w_I} \tag{2.25}$$



Hình 2.2: Mô hình CBOW

Ở lớp đầu ra thay vì đầu ra chỉ một từ như ở CBOW thì chúng ta sẽ cho một C phân bố đầu ra. Mỗi đầu ra được tính cùng một ma trận trọng số hiden-output.

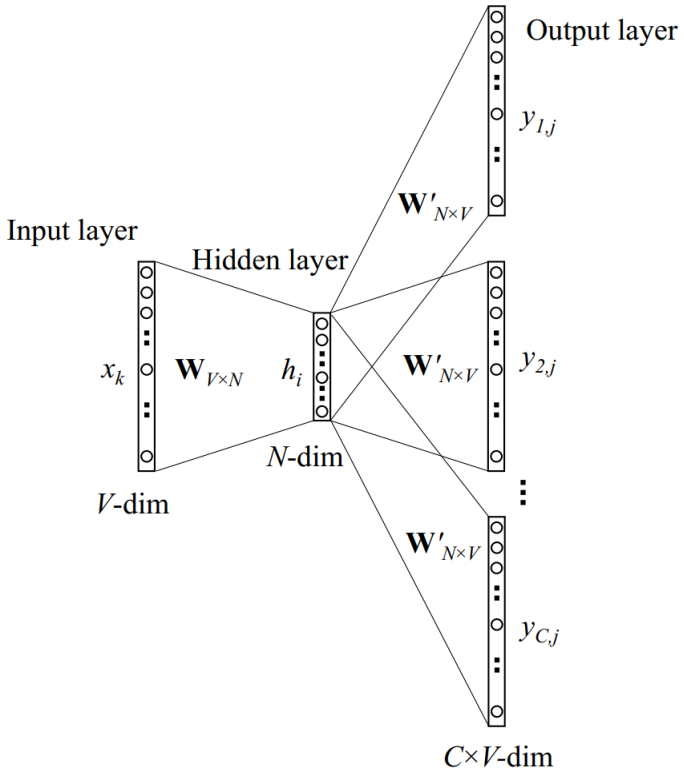
$$y_{c,j} = p(w_{c,j} = w_{O,c}|w_I) = \frac{e^{u_{c,j}}}{\sum_{j'=1}^{V} e^{u_{j'}}} = \frac{\exp(u_{c,j})}{\sum_{j'=1}^{V} \exp(u'_j)} \tag{2.26}$$

Trong đó ký hiệu:

- $w_{c,j}$ là từ thứ j trong panel thứ c của lớp output.
- $w_{O,c}$ là word thứ c trong chuỗi từ đầu ra đang xem xét.
- w_I là input word.
- $y_{c,j}$ là output của phần thứ j trong panel thứ c của lớp output.
- $u_{c,j}$ là net input của phần thứ j trong panel thứ c của lớp output.

Bơi vì các panel đầu ra có cùng ma trận trọng số do đó:

$$u_{c,j} = u_j = {\mathbf{v}'_{w_j}}^T \cdot \mathbf{h} \quad \text{với} \quad c = 1, 2, \ldots, C \tag{2.27}$$



Hình 2.3: Mô hình Skip-Gram

Trong đó, $\mathbf{v}'_{w_j}$ là output vector của word thứ j, w_j trong bộ từ vựng, và $\mathbf{v}'_{w_j}$ cũng được lấy ra từ một cột trong ma trận trọng số của lớp hiden→output, $\mathbf{W}'$. Đạo hàm của các tham số cũng tương tự như mô hình một từ đơn. Ký hiệu j_c^* là index của word ở đầu ra thứ c khi đó hàm mất mát sẽ có dạng:

$$E = -\log p(w_{O,1}, w_{O,2}, \ldots, w_{O,C}|w_I) \tag{2.28}$$

$$= -\log \prod_{c=1}^{C} \frac{\exp(u_{c,j_c^*})}{\sum_{j'=1}^{V} \exp(u'_j)} \tag{2.29}$$

$$= -\sum_{c=1}^{C} u_{c,j_c^*} + C \cdot \log \sum_{j'=1}^{V} \exp(u'_j) \tag{2.30}$$

Lấy đạo hàm của hàm mất mát E đối với net input của từng thành phần trên từng panel của output layer, $u_{c,j}$ ta sẽ có tương tự như trong biểu thức (2.9), chứng minh xem (1.16):

$$\frac{\partial E}{\partial u_{c,j}} = y_{c,j} - t_{c,j} := e_{c,j} \tag{2.31}$$

Chúng ta định nghĩa tiếp vector có V-chiều $\text{EI} = \{\text{EI}_1, \ldots, \text{EI}_V\}$ như là tổng các lỗi dự đoán trên tất cả các word:

$$\text{EI}_j = \sum_{c=1}^{C} e_{c,j} \tag{2.32}$$

Tiếp theo lấy đạo hàm theo ma trận trọng số hidden→output, W' và chúng ta có:

$$\frac{\partial E}{\partial w'_{ij}} = \sum_{c=1}^{C} \frac{\partial E}{\partial u_{c,j}} \cdot \frac{\partial u_{c,j}}{\partial w'_{ij}} = \text{EI}_j \cdot h_i \tag{2.33}$$

Biểu thức để cập nhật ma trận $\mathbf{W}'$ của lớp hidden→output là:

$$w'^{(\text{new})}_{ij} = w'^{(\text{old})}_{ij} - \eta \cdot \text{EI}_j \cdot h_i \tag{2.34}$$

hay có thể viết cách khác:

$$\mathbf{v}'^{(\text{new})}_{w_j} = \mathbf{v}'^{(\text{old})}_{w_j} - \eta \cdot \text{EI}_j \cdot \mathbf{h} \quad \text{với} \quad j = 1, 2, \ldots, V \tag{2.35}$$

Công thức này ý nghĩa cũng tương tự như công thức (2.12), chỉ khác rằng sai số được tính cho tất cả các từ đang được xem xét trong ngữ cảnh (context) ở lớp output.

Với phần cập nhật ma trận trọng số lớp input→hidden công thức sẽ hoàn toàn giống như các công thức từ (2.13) đến (2.17) với một điều khác là thay giá trị sai dố e_j sẽ được thay thế bằng vector EI_j chúng ta có công thức cập nhật là:

$$\mathbf{v}_{w_I}{}^{(\text{new})} = \mathbf{v}_{w_I}{}^{(\text{old})} - \eta \cdot \text{EH}^T \tag{2.36}$$

Trong đó EI là vector N chiều, mỗi phần tử được định nghĩa như sau:

$$\text{EI}_i = \sum_{j=1}^{V} \text{EI}_j \cdot w'_{ij} \tag{2.37}$$

2.5 Tối ưu tính toán

Trong tất cả các mô hình mỗi word trong bộ từ vựng có 2 vector: input vector $\mathbf{v}_w$, output vector $\mathbf{v}'_w$. Phần input vector tính toán khá rẻ nhưng output vector đòi hỏi tài nguyên tính toán rất nhiều, trong các công thức (2.23), (2.34) đòi hỏi chúng ta phải tính cho từng word w_j trong bộ từ vựng, net input u_j, xác suất phân bố y_j và $y_{c,j}$ đối với skip-gram, sai số dự đoán e_j và EI_j đối với skip-gram và cuối cùng là output vector ra của word đó $\mathbf{v}'_j$ nên sẽ đòi hỏi nhiều tài nguyên tính toán nhất là đối với lượng từ vựng lớn. Trong phần này chủ yếu tối ưu cho tính toán output vector và tập chung vào phần tính giá trị E, $\frac{\partial E}{\partial \mathbf{v}'_w}$, $\frac{\partial E}{\partial \mathbf{h}}$.

2.5.1 Hierarchical Softmax

Hierarchical softmax là phương pháp tối ưu được đề xuất trong [7, 2]. Mô hình sử dụng cây nhị phân để biểu diễn các word trong bộ từ vựng. Trong mô hình này không có dạng biểu diễn output vector cho các word thay vào đó mỗi phần tử trung gian của $V-1$ sẽ có output vector là $\mathbf{v}'_{n(w,j)}$. Xác suất của một từ có thể là một output word được định nghĩa như sau:

$$p(w = w_O) = \prod_{j=1}^{L(w)-1} \sigma\left([n(w, j+1) = ch(n(w, j))] \cdot {\mathbf{v}'_{n(w,j)}}^T \mathbf{h}\right) \tag{2.38}$$

Trong đó có các ký hiệu:

- ch(n) là nhánh trái (left child) của phần n;
- $\mathbf{v}'_{n(w,j)}$ là phần biểu diễn của output vector của phần $n(w, j)$.
- $\mathbf{h}$: là giá trị đầu ra của lớp hidden, trong mô hình skip-gram $\mathbf{h} = \mathbf{v}_{w_I}$, còn trong mô hình CBOW thì $\mathbf{h} = \frac{1}{C}\sum_{c=1}^{C} \mathbf{v}_{w_c}$;
- $[\![x]\!]$ là một hàm đặc biệt được định nghĩa.

$$[\![x]\!] = \begin{cases} 1 & \text{nếu } x \text{ là đúng} \\ -1 & \text{nếu ngược lại} \end{cases}$$

Ở mỗi nút trung gian cần phải gán một xác suất đi bên trái và xác suất đi bên phải.

$$p(n, \text{left}) = \sigma({\mathbf{v}'_n}^T \cdot \mathbf{h}) \tag{2.39}$$

$$p(n, \text{right}) = 1 - \sigma({\mathbf{v}'_n}^T \cdot \mathbf{h}) = \sigma(-{\mathbf{v}'_n}^T \cdot \mathbf{h}) \tag{2.40}$$

Trong hình 2.4, xác suất của từ w_2 được tính như sau:

$$p(w_2 = w_O) = p(n(w_2, 1), \text{left}) \cdot p(n(w_2, 2), \text{left}) \cdot p(n(w_2, 3), \text{right}) \tag{2.41}$$

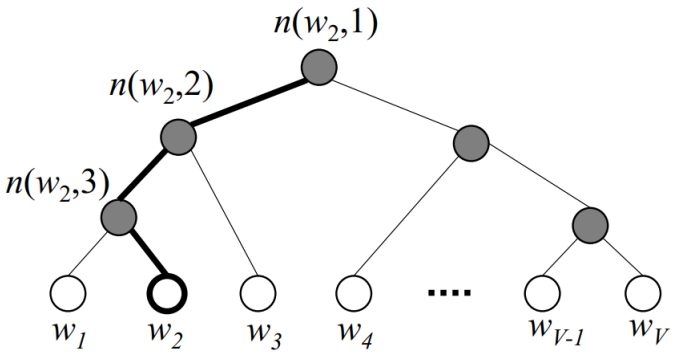
$$= \sigma({\mathbf{v}'_{n(w_2,1)}}^T \cdot \mathbf{h}) \cdot \sigma({\mathbf{v}'_{n(w_2,2)}}^T \cdot \mathbf{h}) \cdot \sigma(-{\mathbf{v}'_{n(w_2,3)}}^T \cdot \mathbf{h}) \tag{2.42}$$

Dễ dàng thấy rằng:

$$\sum_{i=1}^{V} p(w_i = w_O) = 1 \tag{2.43}$$

Để ngắn gọn chúng ta quy ước:

$$[\![\cdot]\!] := [\![n(w, j+1) = \text{ch}(n(w, j))]\!] \tag{2.44}$$



Hình 2.4: Softmax theo dạng cây, những phần mầu trắng là các word trong từ vựng, các phần tối hơn không nằm trong từ vựng, là các từ trung gian. Đường dẫn w_2 được tô đậm.

$$\mathbf{v}'_j := \mathbf{v}'_{n_{w,j}} \tag{2.45}$$

Cho việc học, hàm sai số được định nghĩa như sau:

$$E = -\log p(w = w_O|w_I) = -\sum_{j=1}^{L(w)-1} \log \sigma\left([\![\cdot]\!]\mathbf{v}'^T_j \mathbf{h}\right) \tag{2.46}$$

Lấy đạo hàm của E theo $\mathbf{v}'^T_j \mathbf{h}$ chúng ta sẽ có:

$$\frac{\partial E}{\partial \mathbf{v}'^T_j \mathbf{h}} = \left(\sigma([\![\cdot]\!]\mathbf{v}'^T_j \mathbf{h}) - 1\right)[\![\cdot]\!] \tag{2.47}$$

$$= \begin{cases} \sigma(\mathbf{v}'^T_j \mathbf{h}) - 1 & ([\![\cdot]\!] = 1) \\ \sigma\left(\mathbf{v}'^T_j \mathbf{h}\right) & ([\![\cdot]\!] = -1) \end{cases} \tag{2.48}$$

$$= \sigma\left(\mathbf{v}'^T_j \mathbf{h}\right) - t_j \tag{2.49}$$

Trong đó $t_j = 1$ nếu $[\![\cdot]\!] = 1$ và $t_j = 0$ trong các trường hợp còn lại. Lấy đạo hàm E theo vector biểu diễn phần tử trung gian $n(w, j)$ chúng ta sẽ thu được:

$$\frac{\partial E}{\partial \mathbf{v}'_j} = \frac{\partial E}{\partial \mathbf{v}'^T_j \mathbf{h}} \cdot \frac{\partial \mathbf{v}'^T_j \mathbf{h}}{\partial \mathbf{v}'_j} = \left(\sigma(\mathbf{v}'^T_j \mathbf{h}) - t_j\right) \cdot \mathbf{h} \tag{2.50}$$

Từ đó chúng ta có vector cập nhật sẽ là:

$$\mathbf{v}'^{(new)}_j = \mathbf{v}'^{(old)}_j - \eta\left(\sigma(\mathbf{v}'^T_j \mathbf{h}) - t_j\right) \cdot \mathbf{h} \quad \text{với} \quad j = 1, 2, \ldots, L(w) - 1 \tag{2.51}$$

Chúng ta có thể coi rằng $\sigma(\mathbf{v}'^T_j \mathbf{h}) - t_j$ như là sai số dự đoán cho mỗi phần tử trung gian $n(w, j)$. Nhiệm vụ của mỗi phần tử trung gian là dự đoán liệu nó phải đi sang

nhánh trái hay nhánh bên phải, $t_j = 1$ cần rẽ sang bên trái và $t_j = 0$ là rẽ sang bên phải.

Để lan truyền ngược lỗi (sai số) để cập nhật lại trọng số lớp input→hidden ta cần phai lấy đạo hàm E theo đầu ra của lớp hidden:

$$\frac{\partial E}{\partial \mathbf{h}} = \frac{\partial E}{\partial {\mathbf{v}'_j}^T \mathbf{h}} \cdot \frac{\partial {\mathbf{v}'_j}^T \mathbf{h}}{\partial \mathbf{h}} = \sum_{j=1}^{L(w)-1} \left(\sigma({\mathbf{v}'_j}^T \mathbf{h}) - t_j\right) \cdot \mathbf{v}'_j := \text{EH} \tag{2.52}$$

Có giá trị EH, có thể tính được vector cập nhật theo công thức (2.24) với mô hình CBOW. Trong mô hình skip-gram cần tính giá trị EH cho từng word đang xét và thay vào công thức (2.36). Vơi cách tiếp cận nay độ phức tạp của thuật toán đã giảm đi từ $O(V)$ xuống còn $O(\log(V))$.

2.5.2 Negative Sampling

Để giải quyết vấn đề khó khăn là phải làm việc với quá nhiều output vectors cần phải cập nhật, chúng ta chỉ cập nhật một số trong số đó mà thôi. Output word (trong trường hợp này là ground truth, hoặc positive sample) cần phải được cập nhật, chúng ta cần lấy mẫu một số từ gọi là mẫu âm (negative sample). Phân bố cần thiết được lấy mẫu còn được gọi là phân bố nhiễu và ký hiệu là $P_n(w)$.

Mikolov tin rằng sử dụng hàm đối tượng dưới dây sẽ cho chất lượng word embedding tốt hơn:

$$E = -\log \sigma\left({\mathbf{v}'_{w_O}}^T \mathbf{h}\right) - \sum_{w_j \in \mathcal{W}_{\text{neg}}} \log \sigma\left(-{\mathbf{v}'_{w_j}}^T \mathbf{h}\right) \tag{2.53}$$

Trong đó có các ký hiệu:

- w_O là output word (positive sample).
- $\mathbf{v}'_{w_O}$ là output vector của w_O.
- $\mathbf{h}$: là giá trị đầu ra của lớp hidden, trong mô hình skip-gram $\mathbf{h} = \mathbf{v}_{w_I}$, còn trong mô hình CBOW thì $\mathbf{h} = \frac{1}{C}\sum_{c=1}^{C} \mathbf{v}_{w_c}$;
- $\mathcal{W}_{\text{neg}} = \{w_j | j = 1, \ldots, K\}$ là tập các word được lấy mẫu trên cơ sở $P_n(w)$ hay còn gọi là negative sample.

Lấy đạo hàm của E theo ${\mathbf{v}'_j}^T \mathbf{h}$ chúng ta sẽ có:

$$\frac{\partial E}{\partial {\mathbf{v}'_{w_j}}^T \mathbf{h}} = \begin{cases} \sigma({\mathbf{v}'_{w_j}}^T \mathbf{h}) - 1 & \text{nếu } w_j = w_O \\ \sigma({\mathbf{v}'_{w_j}}^T \mathbf{h}) & \text{nếu } w_j \in \mathcal{W}_{\text{neg}} \end{cases} \tag{2.54}$$

$$= \sigma({\mathbf{v}'_{w_j}}^T \mathbf{h}) - t_j \tag{2.55}$$

Trong đó t_j là nhãn của từ w_j. $t_j = 1$ khi w_j la positive sample và $t_j = 0$ trong trường hợp ngược lại. Tiếp theo lấy đạo hàm của E theo $\mathbf{v}'_{w_j}$ chúng ta sẽ có:

$$\frac{\partial E}{\partial \mathbf{v}'_{w_j}} = \frac{\partial E}{\partial \mathbf{v}'^{T}_{w_j}\mathbf{h}} \cdot \frac{\partial \mathbf{v}'^{T}_{w_j}\mathbf{h}}{\partial \mathbf{v}'_{w_j}} = \left(\sigma(\mathbf{v}'^{T}_{w_j}\mathbf{h}) - t_j\right) \cdot \mathbf{h} \tag{2.56}$$

Kết quả dẫn đến công thức cập nhật cho output vector:

$$\mathbf{v}'^{(new)}_{w_j} = \mathbf{v}'^{(old)}_{w_j} - \eta \left(\sigma(\mathbf{v}'^{T}_{w_j}\mathbf{h}) - t_j\right) \cdot \mathbf{h} \tag{2.57}$$

Trong đó cần phải áp dụng cho word $w_j \in w_O \cup \mathcal{W}_{\text{neg}}$ thay vì phải tính cho tất cả các word trong bộ từ vựng. Đó là lý do cách này giảm được khối lượng tính toán đáng kể. Ý nghĩa của công tức này tương tự như công thức (2.12). Công thức này áp dụng cho cả hai mô hình CBOW và skip-gram, đối với skip-gram chúng ta chỉ tính cho một từ đang xét một lần.

Để lan truyền ngược lỗi (sai số) để cập nhật lại trọng số lớp input→hidden ta cần phai lấy đạo hàm E theo đầu ra của lớp hidden:

$$\frac{\partial E}{\partial \mathbf{h}} = \sum_{w_j \in w_O \cup \mathcal{W}_{\text{neg}}} \frac{\partial E}{\partial \mathbf{v}'^{T}_{w_j}\mathbf{h}} \cdot \frac{\partial \mathbf{v}'^{T}_{w_j}\mathbf{h}}{\partial \mathbf{h}} \tag{2.58}$$

$$= \sum_{w_j \in w_O \cup \mathcal{W}_{\text{neg}}} \left(\sigma(\mathbf{v}'^{T}_{w_j}\mathbf{h}) - t_j\right) \cdot \mathbf{v}'_{w_j} := \text{EH} \tag{2.59}$$

Với mô hình CBOW chúng ta chỉ cần thay EH vào công thức (2.24) để tính input vector. Đối với mô hình skip-gram chúng ta phải tính cho từng word đang xét và lắp EH vào công thức (2.36).

2.6 Phân tích mã nguồn

Mã nguồn do Tomas Mikolov người Czech làm việc cho google và hiện cho Facebook phát triển: https://code.google.com/p/word2vec/, sau này dự án Gensim cũng đã chuyển mã nguồn bằng ngôn ngữ C sang ngôn ngữ python và thêm nhiều thư viện khác nữa.

```
1 struct vocab_word {
2 long long cn;  // word count
3 int  *point;   //
4 char *word,    //
5 char *code,    // ma
```

```
6 char codelen;  // do dai cua ma
7 };
8 // Maximum 30 * 0.7 = 21M words in the vocabulary
```

Listing 2.1: Cấu trúc của bộ từ vựng vocab_word

Algorithm 1: Chương trình chính: int **main**(int argc, char **argv)

Input: file chứa bảng dữ liệu các từ trong từ vựng, mỗi từ có kèm theo số lần xuất hiện trong từ vựng.

Output: file chứ vector của các từ.

1 Xử lý tham số dòng lệnh;

2 Khởi tạo bộ nhớ cho bảng từ vựng **vocab** và bảng index dùng hàm băm **vocab_hash**;

3 Tạo mảng chứa các giá trị của hàm softmax: Precompute f(x) = exp(x) / (exp(x) + 1);

4 **TrainModel();** `/* Hàm chính để tính vector */`

5 return 0;

Thuật toán chính được thực hiện trong hàm **TrainModel()**.

Các biến sử dụng trong phần mềm:

- **syn0[i * layer1_size + a]**: vector của w_i, a: là thành phần thứ a trong vector. Đây cũng chính là lưu các giá trị của vector $\mathbf{v}_{w_i}$ từ lớp input→hidden. các giá trị ban đầu được lấy là các giá trị ngẫu nhiên, sau đó qua quá trình học máy bằng mạng neuron sẽ được điều chỉnh lại một cách tối ưu.
- **syn1**[ ] là vector trọng số đầu ra $\mathbf{v}'_{w_i}$ từ lớp hidden→output.
- **neu1**[ ]: mảng chứa giá trị **h**. Ban đầu được tính là $\mathbf{h} = \frac{1}{C}(\mathbf{v}_{w_1}+\mathbf{v}_{w_2}+\ldots+\mathbf{v}_{w_C})^T$.
- **neu1e**[ ]: mảng chứa lượng phạt sai số cho trọng số đầu vào **syn0**[ ].
- **alpha**: là learning rate: η.
- «`f += neu1[c] * syn1[c + l2];`»: Ở đây **f** chính là $u_j = \mathbf{v}'^T_{w_j} \cdot \mathbf{h}$.
- «`f = expTable[(int)((f + MAX_EXP) * (EXP_TABLE_SIZE / MAX_EXP / 2))];`» trong đó **f** là giá trị đầu ra y_j.
- «`g = (1 - vocab[word].code[d] - f) * alpha;`»: **g** là gradient nhân với learning rate: $-\eta \cdot e_j$.

Algorithm 2: Hàm **TrainModel**()

- Đọc file từ vựng thông qua hàm **ReadVocab()**
  - Đọc từng word một qua hàm **ReadWord(word, fin)**,
    - mỗi từ được phân cách bởi dấu cách, dấu tab, hoặc ngắt dòng, sau đó lưu word vào bảng từ vựng **vocab**[**vocab_size**]**.word**.
    - tăng dần **vocab_size++**;
    - tính giá trị băm của word ghi bảo bảng hash **vocab_hash**[**hash**] = **vocab_size - 1;**
    - đọc word count và lưu vào **vocab**[**a**]**.cn**.
  - Sắp xếp bảng từ vựng bằng hàm **SortVocab();**: sắp xếp bằng hàm qsort theo từ tần suất xuất hiện **vocab_word->cn**.
  - tính lại giá trị của chỉ số hash trong bảng **vocab_hash**[].
- Gọi hàm **InitNet()**
  - Khởi tạo các giá trị ban đầu là 0 cho các bảng **syn1**[], **syn1neg**[], kích thước là $vocab\_size \times layer1\_size$, trong đó $layer1\_size$ là số chiều vector của word.
  - khởi tạo bảng **syn0**[] với giá trị ban đầu là số ngẫu nhiên.
  - Tạo cây nhị phân Huffman bằng cách tính số lần xuất hiện của word qua việc gọi hàm **CreateBinaryTree()**;
- Chạy vòng lặp với **num_threads** lần để khởi tạo **num_threads** luồng thực thi việc xây dựng mô hình qua hàm **TrainModelThread()**.
- Ghi xuống file kết quả là vector của các từ, dòng đầu tiên ghi số lượng từ trong từ vựng là **vocab_size** và kích thước vector của từ là **layer1_size** cách nhau 1 dấu cách xong xuống dòng.
- Chạy vòng lặp ghi tất cả các word xuống file với word đó **vocab**[**i**]**.word** và các giá trị của từng giá trị trong vector **syn0**[**i * layer1_size + j**]. Nếu file là dạng binary thì dùng hàm c chuẩn **fwrite()**, nếu là file text thì dùng hàm **fprintf()**, trong dạng text thì mỗi giá trị cách nhau 1 khoảng trắng.
- Xây dựng mảng giá trị phân cụm cho từng word bằng phương pháp *k-means*.
- Ghi tiếp xuống file kết quả mỗi word là 1 dòng, trong mỗi dòng đầu tiên ghi word trong **vocab**[**i**]**.word**, sau một dấu cách là giá trị chỉ số cụm được lưu trong **cl**[**i**] được tạo ra ở bước trước.

Algorithm 3: Hàm chính **TrainModelThread()**()

- Chuyển vị trí đọc file đến vùng xử lý cho từng thread tùy theo số lượng num_threads.
- Đọc từng từ bằng hàm «`word = ReadWordIndex(fi);`»để tạo thành câu có độ dài là `MAX_SENTENCE_LENGTH`. Thực ra câu này là bộ đệm để không phải đọc đi đọc lại các word từ file.
- Duyệt từng word trong câu «`sen[ ]`».
  - Với mô hình CBOW thực hiện tranning theo các bước sau:
    * Đọc từ bộ đệm ra «`window * 2 + 1`» từ (word) và tính

$$\mathbf{h} = \frac{1}{C}(\mathbf{v}_{w_1} + \mathbf{v}_{w_2} + \ldots + \mathbf{v}_{w_C})^T \tag{2.60}$$

      bằng lệnh «`for (c = 0; c < layer1_size; c++) neu1[c] += syn0[c + last_word * layer1_size]/cw;`»
    * Tính giá trị net inout hay là score:

$$u_j = \mathbf{v}'^T_{w_j}\mathbf{h} \tag{2.61}$$

      bằng câu lệnh `f` là u_j: `for (c = 0; c < layer1_size; c++) f += neu1[c] * syn1[c + l2];` $\mathbf{v}'^T_{w_j}$ là `syn1[ ]` giá trị ban đầu là 0.
    * Tính giá trị đầu ra:

$$y_j = p(w_j|w_I) = \frac{\exp(u_j)}{\sum_{j'=1}^{V}\exp(u'_j)} \tag{2.62}$$

      bằng câu lệnh: `f = expTable[(int)((f + MAX_EXP) * (EXP_TABLE_SIZE / MAX_EXP / 2))];`
    * Tính giá trị: $\mathtt{g} = -\eta \cdot e_j$, bằng câu lệnh `g = (1 - vocab[word].code[d] - f) * alpha;`
    * Cập nhật ma trận trọng số đầu ra:

$$\mathbf{v}'_{w_j}{}^{(\text{new})} = \mathbf{v}'_{w_j}{}^{(\text{old})} - \eta \cdot e_j \cdot \mathbf{h} = \mathbf{v}'_{w_j}{}^{(\text{old})} + \mathtt{g} \cdot \mathbf{h} \tag{2.63}$$

    * Cập nhật ma trận đầu vào:

$$\mathbf{v}_{w_{I,C}}{}^{(\text{new})} = \mathbf{v}_{w_{I,C}}{}^{(\text{old})} - \frac{1}{C} \cdot \eta \cdot \mathrm{EH}^T \tag{2.64}$$

      bằng câu lệnh: `for (c = 0; c < layer1_size; c++) syn1[c + l2] += g * neu1[c];`
    * Cập nhật trọng số đầu vào: bằng câu lệnh: `for (c = 0; c < layer1_size; c++) syn0[c + last_word * layer1_size] += neu1e[c];` Đây chính là các vector của word.

Tài liệu tham khảo

[1] Piotr Bojanowski et al. (2016), "Enriching Word Vectors with Subword Information", *arXiv*, URL: http://arxiv.org/abs/1607.04606.

[2] Frederic Morin and Yoshua Bengio (2005), "Hierarchical Probabilistic Neural Network Language Model", *AISTATS*, 5, pp. 246–252.

[3] Palash Goyal, Sumit Pandey, and Karan Jain (2018), *Deep Learning for Natural Language Processing*, Apress, p. 290.

[4] Tomas Mikolov, Wen-Tau Yih, and Geoffrey Zweig (2013), "Linguistic Regularities in Continuous Space Word Representations", *Proceedings ofNAACL-HLT 2013*, pp. 746–751.

[5] Tomas Mikolov et al. (2013), "Efficient Estimation of Word Representations in Vector Space", *arXiv*, pp. 1–12, URL: http://arxiv.org/abs/1301.3781.

[6] Tomas Mikolov et al. (2013), "Distributed Representations of Words and Phrases and their Compositionality", *CrossRef Listing of Deleted DOIs*, 1, pp. 1–9.

[7] A Mnih and G Hinton (2009), "A Scalable Hierarchical Distributed Language Model", *BT - Advances in Neural Information Processing Systems*, 21, pp. 1081–1088.

[8] Jeffrey Pennington, Richard Socher, and Christopher Manning (2014), "Glove: Global Vectors for Word Representation", *Proceedings ofthe 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, ISSN: 10495258.