Đại học Khoa học tự nhiên

Môn học: Khoa Học Dữ Liệu

Báo cáo đồ án cuối kỳ

Word2Vec

Giảng viên
Nguyễn Ngọc Đức

Sinh viên
20424008 - Dương Mạnh Cường

Ngày 28 tháng 2 năm 2022

Mục lục

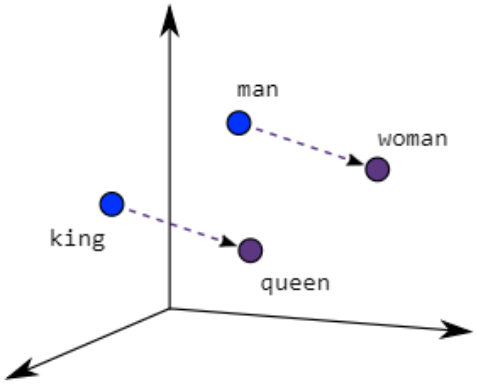
Neural network yêu cầu đầu vào ở dạng **numeric**. Cho nên, khi ta có dữ liệu dạng **text**, ta cần chuyển đổi chúng thành dữ liệu dạng numeric.

Có nhiều phương pháp khác nhau để chuyển đổi dữ liệu dạng text sang numeric mà phổ biến nhất là:

- Term frequency-inverse document frequency (TF-IDF).
- Bag of words (BOW).

Tuy nhiên, điểm yếu của hai phương pháp trên là chúng **không nắm bắt được ngữ nghĩa của từ**, nói cách khác là chúng không hiểu được ý nghĩa của từ.

Có nhiều cách khắc phục nhược điểm này, mà một trong những cách đó là sử dụng **Word2Vec** bằng cách đại diện cho từng từ bằng một **vector** trong không gian m chiều. Lúc này, các từ có nghĩa tương đồng nhau sẽ nằm gần nhau.



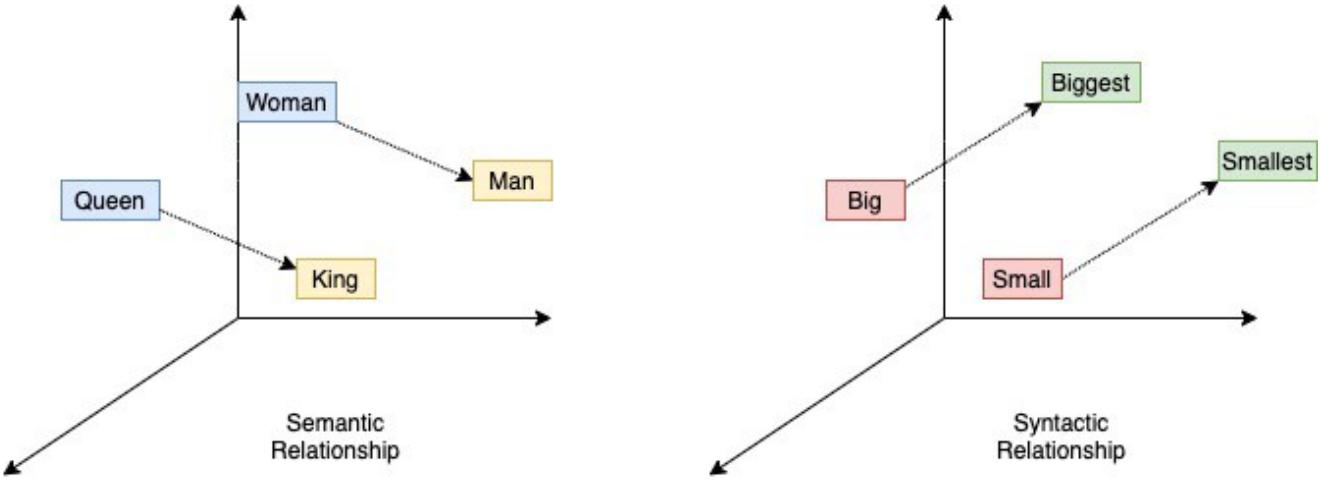
Hình 1: Các từ có ý nghĩa tương đồng nhau nằm gần nhau.

1 Word2Vec model

Word2Vec là một trong những phương pháp **word embedding** được sử dụng phổ biến.

Word embedding là cách ta biểu diễn các **word vector** trong không gian vector.

Các word vector được tạo ra bởi Word2Vec model có khả năng nắm bắt được các **semantic** *(ngữ nghĩa)* và **syntactic** *(ý nghĩa cú pháp)* của từ.



Hình 2: Ví dụ về **semantic** và **syntactic**.

Ví dụ có câu: *"Archie used to live in New York, he then moved to Santa Clara. He loves apples and strawberries."*.

Word2Vec model sẽ phát sinh các vector cho từng từ trong văn bản. Nếu chúng ta trực quan các vector này trong không gian vector tương ứng, chúng ta có thể thấy các từ tương tự nhau sẽ nằm gần nhau.

Hình 3: Các từ trong câu ví dụ được biểu diễn trong không gian vector

Nhận xét

- Ở đây các cặp từ như *apples - strawberries*, *New York - Santa Clara* có ý nghĩa tương đồng nhau nên nằm gần nhau.

Do đó, với Word2Vec model có thể học cách biểu diễn các vector giúp cho neural network có thể hiểu được ý nghĩa của từ tương ứng với vector đó.

Và vì chúng ta có thể hiểu được semantic và syntactic của từ điều này giúp ta tận dụng các vector này vào các bài toán như **text summarization** *(tóm tắt văn bản)*, **sentiment analysis** *(phân tích tình cảm)*, **text generation** *(tạo văn bản)*,...

Có hai cách để xây dựng Word2Vec model:

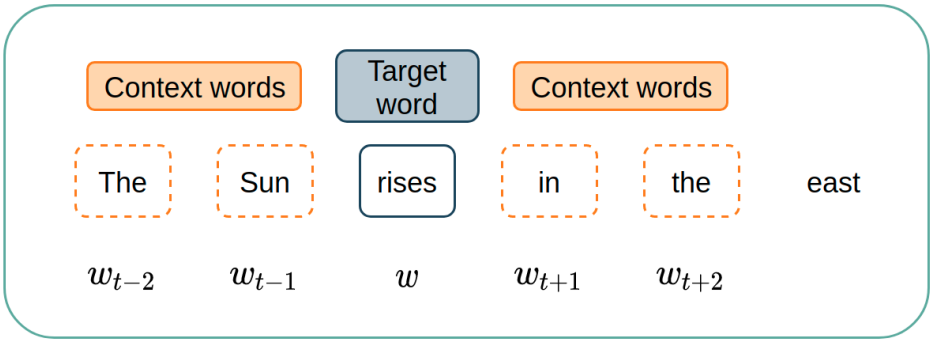
1. **CBOW model**.
2. **Skip-gram model**.

1.1 CBOW model

Giả sử chúng ta có một neural network bao gồm: **một input layer**, **một hidden layer** và **một output layer**. Mục đích của network này là dự đoán ra **một từ** dựa vào **các từ xung quanh nó**. Từ mà chúng ta cố gắng dự đoán được gọi là **target word** và các từ xung quanh nó được gọi là **context word**.

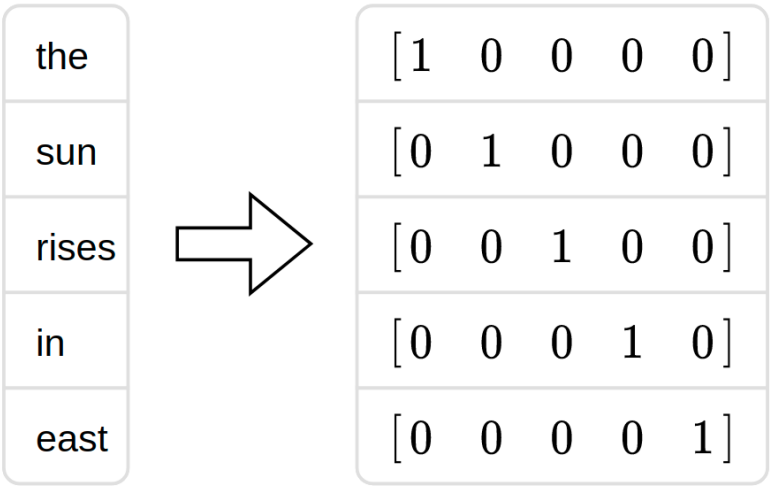
Vậy chúng ta cần bao nhiêu context word để dự đoán ra target word? Chúng ta sẽ sử dụng một **window** *(cửa sổ)* có kích thước là n để chọn các context word. Nếu $n = 2$ thì chúng ta sẽ sử dụng hai từ **phía trước** và **phía sau** của target word làm các context word.

Xem xét câu sau: *"The Sun rises in the east."* với *rises* là target word. Nếu chúng ta xét kích thước của window là 2 thì chúng ta sẽ lấy hai từ phía trước là *The*, *Sun* và hai từ phía sau là *in*, *the* của target word làm các context word.



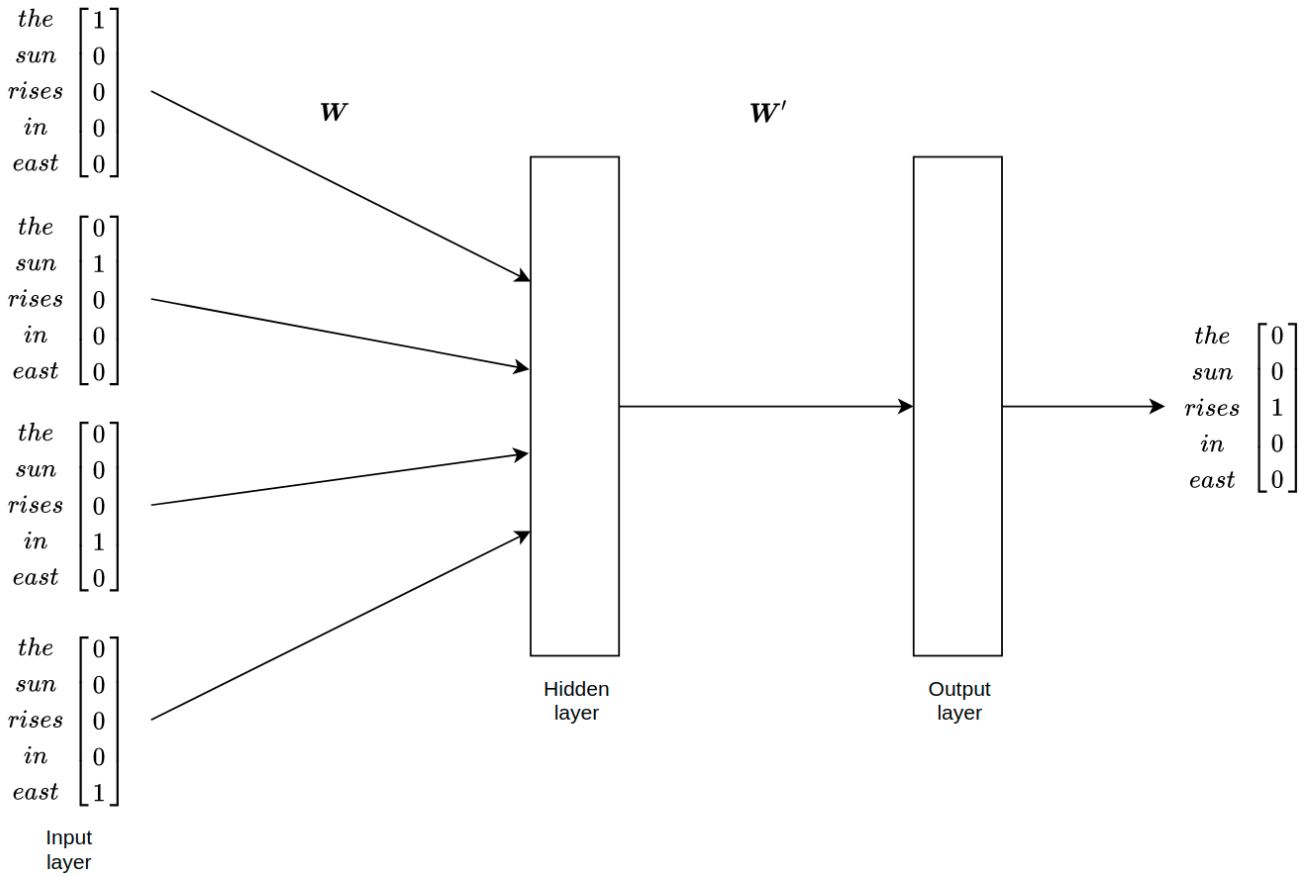
Hình 4: Target word và context word trong CBOW model.

Lúc này input của network là các context word và output của network là target word. Và vì neural network chỉ chấp nhận input là numeric data nên chúng ta sẽ sử dụng kỹ thuật **one-hot encoding** để chuyển đổi các text data thành numeric data.



Hình 5: One-hot encoding cho text data.

Kiến trúc của CBOW model được thể hiện dưới hình sau. Ở đây các context word là: *the*, *sun*, *in* và *east* được dùng làm đầu vào cho network và dự đoán ra target word là *rises* ở đầu ra.



Hình 6: Kiến trúc của CBOW model 1.

Ở vài lần lặp đầu tiên, network không thể dự đoán target word một cách chính xác. Nhưng sau một loạt các vòng lặp bằng cách sử dụng **gradient descent**, các **weight** *(trọng số)* của network được cập nhật và tìm ra được **optimal weight** *(trọng số thích hợp)* để dự đoán ra target word một cách chính xác.

Vì chúng ta có một input layer, một hidden layer và một output layer, nên chúng ta sẽ có hai weight:

1. Weight từ input layer đến hidden layer - $\boldsymbol{W}$.
2. Weight từ hidden layer đến output layer - $\boldsymbol{W'}$.

Trong quá trình đào tạo network, các weight sẽ được cập nhật trong quá trình back propagation nhằm tìm ra optimal weight cho hai bộ $\boldsymbol{W}$ và $\boldsymbol{W'}$.

Về sau, weight set giữa input layer và hidden layer $\boldsymbol{W}$ được cập nhật và tối ưu tạo thành các vector đại diện cho các từ của input layer.

Sau khi kết thúc quá trình đào tạo, chúng ta chỉ cần loại bỏ output layer và lấy ra weight set giữa input layer và hidden layer và gán chúng cho các từ tương ứng.

Dưới đây là các vector tương ứng cho các từ của $\boldsymbol{W}$. Word embedding tương ứng cho từ *sun* là $\begin{bmatrix}0.0 & 0.3 & 0.3 & 0.6 & 0.1\end{bmatrix}$.

$$\boldsymbol{W} = \begin{array}{c} \\ the \\ sun \\ rises \\ in \\ east \end{array} \begin{bmatrix} \\ 0.01 & 0.02 & 0.1 & 0.5 & 0.37 \\ 0.0 & 0.3 & 0.3 & 0.6 & 0.1 \\ 0.4 & 0.34 & 0.11 & 0.61 & 0.43 \\ 0.1 & 0.11 & 0.1 & 0.17 & 0.369 \\ 0.33 & 0.4 & 0.3 & 0.17 & 0.1 \end{bmatrix}$$

1.1.1 CBOW model với một context word

CBOW model cần một số lượng context word C nhất định để dự đoán target word. Ở phần này chúng ta sẽ xem xét trường hợp chỉ sử dụng duy nhất một context word, tức $C = 1$. Lúc này network nhận vào một context word ở đầu vào và trả về một target word ở đầu ra.

Trước tiên, chúng ta cần làm quen với một vài khái niệm. Tất cả các từ duy nhất nằm trong **corpus** *(kho ngữ liệu)* của chúng ta được gọi là các **vocabulary** *(từ vựng)*. Trong ví dụ trước đó của chúng ta, chúng ta có năm từ là: *the*, *sun*, *rises*, *in* và *east* - toàn bộ năm từ này chính là vocabulary trong corpus của chúng ta.

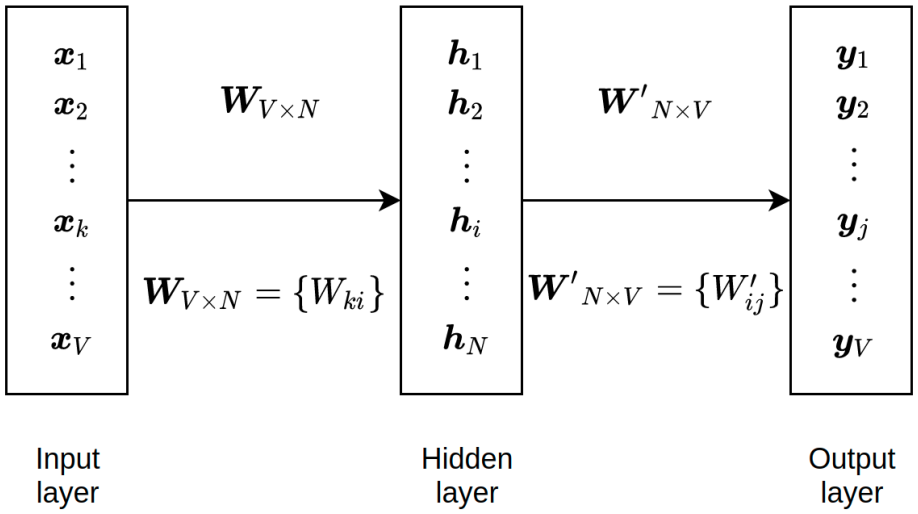
Đặt V là số lượng vocabulary và N là số neuron của hidden layer. Vì neural network của chúng ta có ba layer, cụ thể:

- Input layer được đại diện bởi $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, ..., \boldsymbol{x}_k, ..., \boldsymbol{x}_V\}$. Khi chúng ta nói $\boldsymbol{x}_k$, tức ta đang đề cập đến từ thứ k trong corpus của chúng ta.
- Hidden layer được đại diện bởi $\boldsymbol{H} = \{\boldsymbol{h}_1, \boldsymbol{h}_2, \boldsymbol{h}_3, ..., \boldsymbol{h}_i, ..., \boldsymbol{h}_N\}$. Khi chúng ta nói $\boldsymbol{h}_i$, tức ta đang đề cập đến neuron thứ i trong hidden layer.

- Output layer được đại diện bởi $\boldsymbol{Y} = \{\boldsymbol{y}_1, \boldsymbol{y}_2, \boldsymbol{y}_3, ..., \boldsymbol{y}_j, ..., \boldsymbol{y}_V\}$. Khi chúng ta nói $\boldsymbol{y}_j$, tức ta đang đề cập đến từ thứ j trong output layer.

Số chiều của $\boldsymbol{W}$ là $V \times N$, tức *số vocabulary* $\times$ *số neuron trong hidden layer*. W_{ki} đại diện cho một phần tử trong ma trận $\boldsymbol{W}$ giữa $\boldsymbol{x}_k$ trong input layer và $\boldsymbol{h}_i$ trong hidden layer.

Số chiều của $\boldsymbol{W'}$ là $N \times V$, tức *số neuron trong hidden layer* $\times$ *số vocabulary*. W'_{ij} đại diện cho một phần tử trong ma trận $\boldsymbol{W'}$ giữa $\boldsymbol{h}_i$ trong hidden layer và $\boldsymbol{y}_j$ trong output layer.



Hình 7: Kiến trúc của CBOW model 2.

1.1.1.1 Forward propagation

Để dự đoán target word từ các context word, chúng ta sẽ thực hiện quá trình forward propagation. Trước tiên, chúng ta nhân $\boldsymbol{X}$ cho $\boldsymbol{W}$.

$$\boldsymbol{H} = \boldsymbol{X}\boldsymbol{W}^T$$

Chúng ta biết rằng input là các one-hot vector, nên khi ta nhân $\boldsymbol{X}$ với $\boldsymbol{W}$ thì $\boldsymbol{H}$ lúc này đơn giản là:

$$\boldsymbol{H} = \boldsymbol{W}_{(k,.)}$$

$\boldsymbol{W}_{(k,.)}$ về cơ bản là biểu diễn vector của input word. Đặt vector đại diện cho input word w_I bằng Z_{w_I}. Lúc này phương trình trên có thể được viết thành.

$$\boldsymbol{H} = Z_{w_I} \tag{1}$$

Bây giờ chúng ta đang ở hidden layer, tiếp theo chúng ta cần tính xác suất cho từng từ trong corpus để nó là một target word.

Đặt $\boldsymbol{u}_j$ là điểm số để từ thứ j trong corpus là target word - được tính bằng cách nhân $\boldsymbol{H}$ cho $\boldsymbol{W'}$. Và vì chúng ta đang tính toán điểm số cho từ j trong corpus, nên ta chỉ nhân cột j trong $\boldsymbol{W'}$ cho $\boldsymbol{H}$.

$$\boldsymbol{u}_j = \boldsymbol{W'}_{ij}^T \cdot \boldsymbol{H}$$

Lúc này, cột j của $\boldsymbol{W'}_{ij}$ đại diện cho vector của từ j trong corpus. Đặt vector đại diện cho từ thứ j là $Z'_{w_j^T}$. Phương trình trên có thể được viết lại thành:

$$\boldsymbol{u}_j = Z'_{w_j^T} \cdot \boldsymbol{H} \tag{2}$$

Thế phương trình (1) vào phương trình (2), ta được:

$$\boldsymbol{u}_j = Z'_{w_j^T} \cdot Z_{w_I}$$

Việc tính toán dot product giữa $Z'_{w_j^T}$ và Z_{w_I} nói cho chúng ta biết độ tương đồng giữa target word với các context word đầu vào. Cho nên nếu điểm số của từ j trong corpus cao thì có nghĩa có sự tương đồng cao giữa các context word và target word j này và ngược lại.

Cuối cùng, ta sẽ áp dụng softmax activation để chuyển đổi các điểm số này sang xác suất:

$$\boldsymbol{y}_j = \frac{\exp\left(\boldsymbol{u}_j\right)}{\sum_{j'=1}^{V} \exp\left(\boldsymbol{u}'_j\right)} \tag{3}$$

Lúc này, $\boldsymbol{y}_j$ cho chúng ta biết xác suất của từ j là target word. Chúng ta sẽ tính xác suất cho tất cả các từ trong corpus và chọn từ mà có xác suất cao nhất là target word.

Tiếp theo, chúng ta sẽ tính toán hàm loss. Đặt $\boldsymbol{y}_j^*$ là xác suất để từ j chính xác là target word. Chúng ta cần tối đa xác suất này:

$$\max \boldsymbol{y}_j^*$$

Và để giảm phạm vị khi tính hàm loss ta sẽ tính tối đa hàm log của xác suất.

$$\max \log\left(\boldsymbol{y}_j^*\right)$$

Nhưng vì chúng ta sẽ áp dụng gradient descent để tìm ra optimal weight nên thay vì tối đa hóa ta sẽ tìm tối thiểu hóa công thức trên như sau:

$$\min(-\log\left(\boldsymbol{y}_j^*\right))$$

Vậy lúc này loss function của chúng ta sẽ thành:

$$\mathfrak{L} = -\log\left(\boldsymbol{y}_j^*\right) \tag{4}$$

Thế phương trình (3) vào phương trình (4) ta thu được:

$$\begin{aligned}
\mathfrak{L} &= -\log\left(\frac{\exp\left(u_j\right)}{\sum_{j'=1}^{V} \exp\left(u'_j\right)}\right) \\
&= -\left(\log\left(\exp\left(u_j\right)\right) - \log\left(\sum_{j'=1}^{V} \exp\left(u'_j\right)\right)\right) \\
&= -\log\left(\exp\left(u_j\right)\right) + \log\left(\sum_{j'=1}^{V} \exp\left(u'_j\right)\right) \\
&= -u_j + \log\left(\sum_{j'=1}^{V} \exp\left(u'_j\right)\right)
\end{aligned}$$

1.1.1.2 Backward propagation

Để tối thiểu hóa loss function, chúng ta sử dụng gradient descent. Chúng ta sẽ tính gradient của loss function để cập nhật weight.

Chúng ta có hai weight set là $\boldsymbol{W}$ và $\boldsymbol{W'}$. Trong quá trình backward propagration, chúng ta có thể tiến hành cập nhật weight cho chúng. Cụ thể như sau:

$$\boldsymbol{W} = \boldsymbol{W} - \alpha \frac{\partial \mathfrak{L}}{\partial \boldsymbol{W}}$$

$$\boldsymbol{W'} = \boldsymbol{W'} - \alpha \frac{\partial \mathfrak{L}}{\partial \boldsymbol{W'}}$$

Trước tiên, trong quá trình forward propagation chúng ta có các phương trình sau:

$$\boldsymbol{H} = \boldsymbol{X}\boldsymbol{W}^T$$

$$\boldsymbol{u}_j = \boldsymbol{W'}_{ij}^T \cdot \boldsymbol{H}$$

$$\mathfrak{L} = -\boldsymbol{u}_j + \log\left(\sum_{j'=1}^{V} \exp\left(\boldsymbol{u'}_j\right)\right)$$

Trước tiên, chúng ta sẽ tính gradient của loss function cho $\boldsymbol{W'}$. Chúng ta không thể tính gradient của loss function đối với $\boldsymbol{W'}$ trực tiếp. Cho nên chúng ta phải áp dụng **chain rule** như sau:

$$\frac{\partial \mathfrak{L}}{\partial \boldsymbol{W'}_{ij}} = \frac{\partial \mathfrak{L}}{\partial \boldsymbol{u}_j} \cdot \frac{\partial \boldsymbol{u}_j}{\partial \boldsymbol{W'}_{ij}}$$

Ta có đạo hàm của $\dfrac{\partial \mathfrak{L}}{\partial \boldsymbol{u}_j}$ như sau:

$$\frac{\partial \mathfrak{L}}{\partial \boldsymbol{u}_j} = \boldsymbol{e}_j \tag{5}$$

với $\boldsymbol{e}_j$ là sự khác biệt giữa actual word và predicted word.

Tiếp theo, chúng ta sẽ tính đạo hàm cho $\dfrac{\partial \boldsymbol{u}_j}{\partial \boldsymbol{W'}_{ij}}$, và bởi vì chúng ta biết $\boldsymbol{u}_j = \boldsymbol{W'}_{ij}^T \cdot \boldsymbol{H}$ nên:

$$\frac{\partial \boldsymbol{u}_j}{\partial \boldsymbol{W'}_{ij}} = \boldsymbol{H}$$

Như vậy, gradient của loss function đối với $\boldsymbol{W'}$ bằng:

$$\frac{\partial \mathfrak{L}}{\partial \boldsymbol{W'}_{ij}} = \boldsymbol{e}_j \cdot \boldsymbol{H}$$

Tiếp theo, chúng ta cần tính gradient của loss function đối với $\boldsymbol{W}$, và chúng ta cũng không thể tính gradient trực tiếp nên chúng ta phải áp dụng chain rule.

$$\frac{\partial \mathfrak{L}}{\partial \boldsymbol{W}_{ki}} = \frac{\partial \mathfrak{L}}{\partial \boldsymbol{h}_i} \cdot \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{W}_{ki}}$$

Để tính đạo hàm của $\frac{\partial \mathfrak{L}}{\partial \boldsymbol{h}_i}$, chúng ta áp dụng chain rule:

$$\frac{\partial \mathfrak{L}}{\partial \boldsymbol{h}_i} = \sum_{j=1}^{V} \frac{\partial \mathfrak{L}}{\partial \boldsymbol{u}_j} \cdot \frac{\partial \boldsymbol{u}_j}{\partial \boldsymbol{h}_i}$$

Thế phương trình (5) vào phương trình trên, ta được:

$$\frac{\partial \mathfrak{L}}{\partial \boldsymbol{h}_i} = \sum_{j=1}^{V} \boldsymbol{e}_j \cdot \frac{\partial \boldsymbol{u}_j}{\partial \boldsymbol{h}_i}$$

Và vì chúng ta biết $\boldsymbol{u}_j = \boldsymbol{W'}_{ij}^T \cdot \boldsymbol{H}$, tiếp tục thế vào phương trình trên chúng ta được:

$$\frac{\partial \mathfrak{L}}{\partial \boldsymbol{h}_i} = \sum_{j=1}^{V} \boldsymbol{e}_j \cdot \boldsymbol{W'}_{ij} = \mathfrak{L}\boldsymbol{H}^T$$

Bây giờ chúng ta sẽ tính đạo hàm cho $\frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{W}_{ki}}$. Và vì chúng ta biết $\boldsymbol{H} = \boldsymbol{X}\boldsymbol{W}^T$ nên:

$$\frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{W}_{ki}} = \boldsymbol{X}$$

Cuối cùng, gradient của loss function đối với $\boldsymbol{W}$ là:

$$\frac{\partial \mathfrak{L}}{\partial \boldsymbol{W}_{ki}} = \boldsymbol{L}\boldsymbol{H}^T \cdot \boldsymbol{X}$$

Như vậy, các set weight $\boldsymbol{W}$ và $\boldsymbol{W'}$ của chúng ta lần lượt được cập nhật trong quá trình backward propagation như sau:

$$\boldsymbol{W} = \boldsymbol{W} - \alpha \boldsymbol{L}\boldsymbol{H}^T \cdot \boldsymbol{X}$$

$$\boldsymbol{W'} = \boldsymbol{W'} - \alpha \boldsymbol{e}_j \cdot \boldsymbol{H}$$

Chúng ta thực hiện cập nhật $\boldsymbol{W}$ và $\boldsymbol{W'}$ đến cuối quá trình đào tạo network, $\boldsymbol{W}$ của chúng ta lúc này đạt được trạng thái optimal weight, ta có thể lấy nó ra làm các word vector đại diện cho các vocabulary của chúng ta trong corpus.

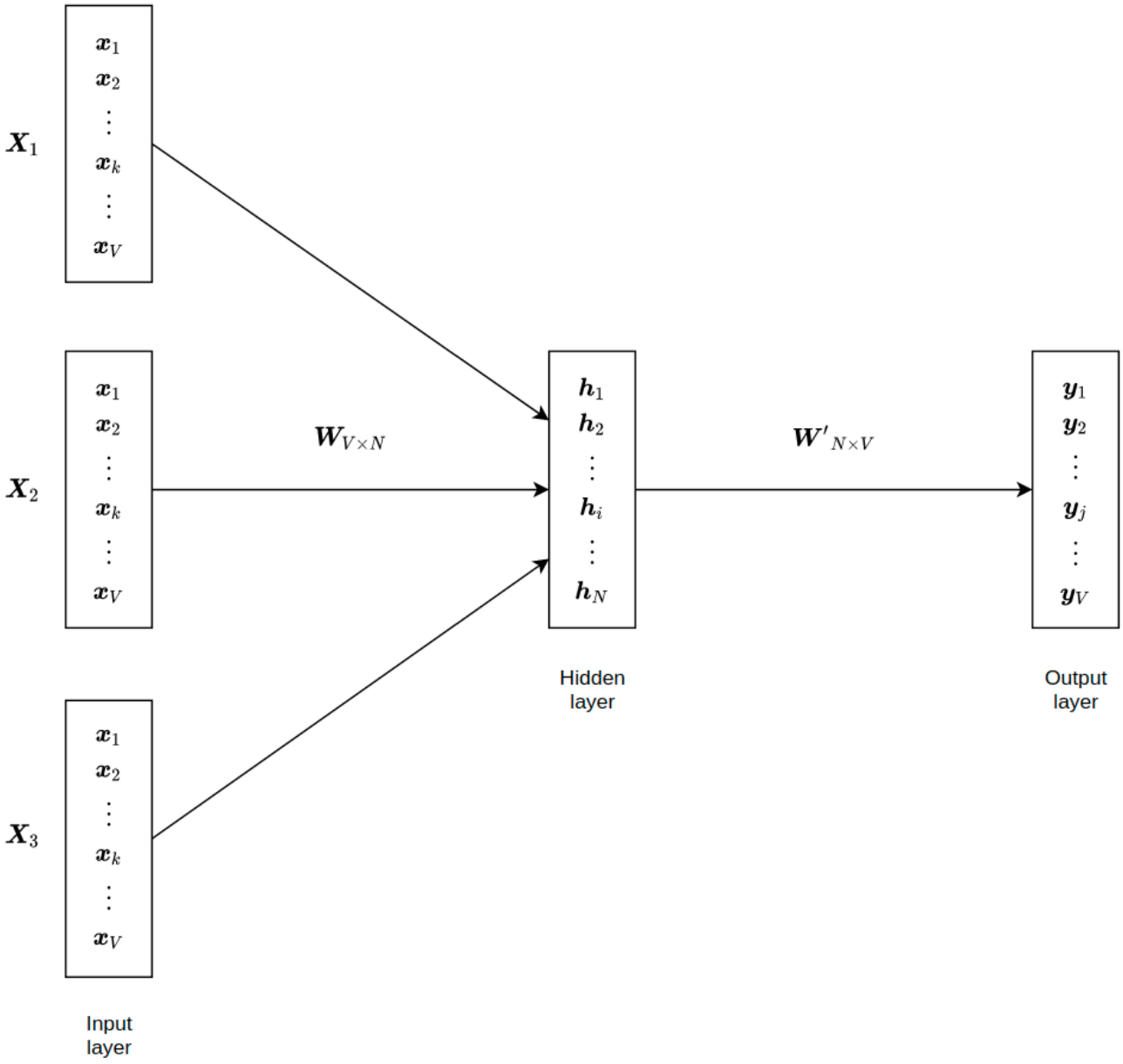
Dưới đây là mã nguồn Python để tìm kiếm các word vector.

```
def single_context_CBOW(x, label, W1, W2, loss):
  # forward propagation
  h = np.dot(W1.T, x)
  u = np.dot(W2.T, h)
  y_pred = softmax(u)

  # error giữa actual value và predicted value + tính backward propagation
  e = label - y_pred
  dW2 = np.outer(h, e)
  dW1 = np.outer(x, np.dot(W2.T, e))

  # cập nhật các weight set
  W1 = W1 - lr * dW1
  W2 = W2 - lr * dW2
```

```
15
16  # loss function
17  loss += -float(u[label] == 1) + np.log(np.sum(np.exp(u)))
18
19  return W1, W2, loss
```

1.1.2 CBOW model với multiple context words

Bây giờ, chúng ta sẽ tìm hiểu cách CBOW model với multiple context words hoạt động. Kiến trúc của CBOW model với multiple context words như sau:



Hình 8: Kiến trúc của CBOW model với multiple context words.

Không có nhiều sự khác biệt giữa single context word và multiple context words. Sự khác biệt là với multiple context words, chúng ta lấy giá trị trung bình của tất cả các context word đầu vào.

Cụ thể với C context word $\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_C$, chúng ta thực hiện:

$$\begin{aligned}\boldsymbol{H} &= \frac{(\boldsymbol{X}_1 + \boldsymbol{X}_2 + \ldots + \boldsymbol{X}_C)}{C}\boldsymbol{W}^T \\ &= \frac{1}{C}(\boldsymbol{X}_1\boldsymbol{W}^T + \boldsymbol{X}_1\boldsymbol{W}^T + \ldots + \boldsymbol{X}_C\boldsymbol{W}^T)\end{aligned}$$

Tương tự như CBOW model với single context word, $\boldsymbol{X}_1\boldsymbol{W}^T$ đại diện cho vector của input context word w_1. Chúng ta đại diện Z_{w_1} cho input context word w_1, cho nên:

$$\boldsymbol{H} = \frac{1}{C}(Z_{w_1} + Z_{w_2} + \ldots + Z_{w_C}) \tag{6}$$

Ở đây, C đại diện cho số lượng context word, để tính toán giá trị của $\boldsymbol{u}_j$ sẽ tương tự như cách ta tính toán cho single context word.

$$\boldsymbol{u}_j = Z'_{w_j^T} \cdot \boldsymbol{H} \tag{7}$$

Ở đây, $Z'_{w_j^T}$ là vector đại diện cho từ thứ j trong corpus.

Thế phương trình (6) vào phương trình (7), ta được:

$$\boldsymbol{u}_j = Z'_{w_j^T} \cdot \frac{1}{C}(Z_{w_1} + Z_{w_2} + \ldots + Z_{w_C})$$

Tiếp theo, chúng ta sẽ tính toán cho loss function - nó tương tự như những gì chúng ta đã làm cho loss function ở single context word.

$$\mathfrak{L} = -u_j + \log\left(\sum_{j'=1}^{V}\exp\left(u'_j\right)\right)$$

Tiếp theo, là quá trình backward propagation. Cụ thể, quá trình cập nhật cho $\boldsymbol{W}$ sẽ như sau:

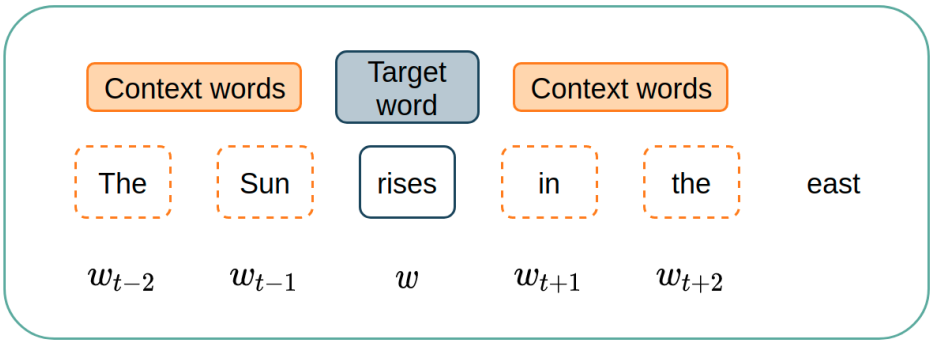
$$\boldsymbol{W} = \boldsymbol{W} - \alpha\mathfrak{L}\boldsymbol{H}^T \cdot \frac{(\boldsymbol{X}_1 + \boldsymbol{X}_2 + \ldots + \boldsymbol{X}_C)}{C}$$

Đối với $\boldsymbol{W'}$ sẽ tương tự như single context word:

$$\boldsymbol{W'} = \boldsymbol{W'} - \alpha\boldsymbol{e}_j \cdot \boldsymbol{H}$$

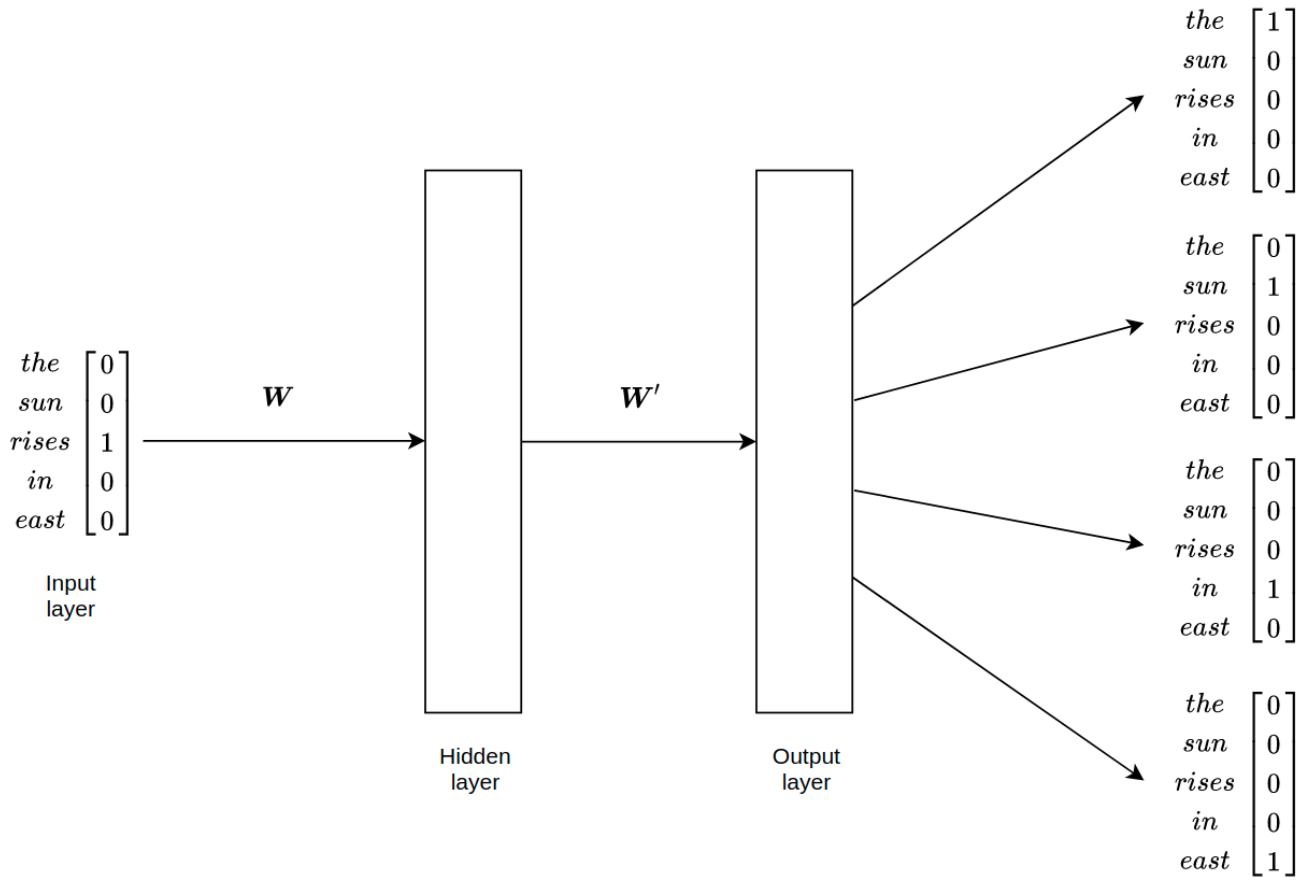
1.2 Skip-gram model

Skip-gram model về cơ bản như là việc làm ngược lại CBOW model. Với Skip-gram model, chúng ta sẽ cố gắng dự đoán các context word dựa vào target word được cung cấp ở đầu vào. Lấy lại ví dụ ban đầu của CBOW model, chúng ta sẽ dùng target word là *rises* để dự đoán các context word là *the*, *sun*, *in* và *east*.



Hình 9: Target word và context word trong Skip-gram model.

Dưới đây là kiến trúc của Skip-gram model:

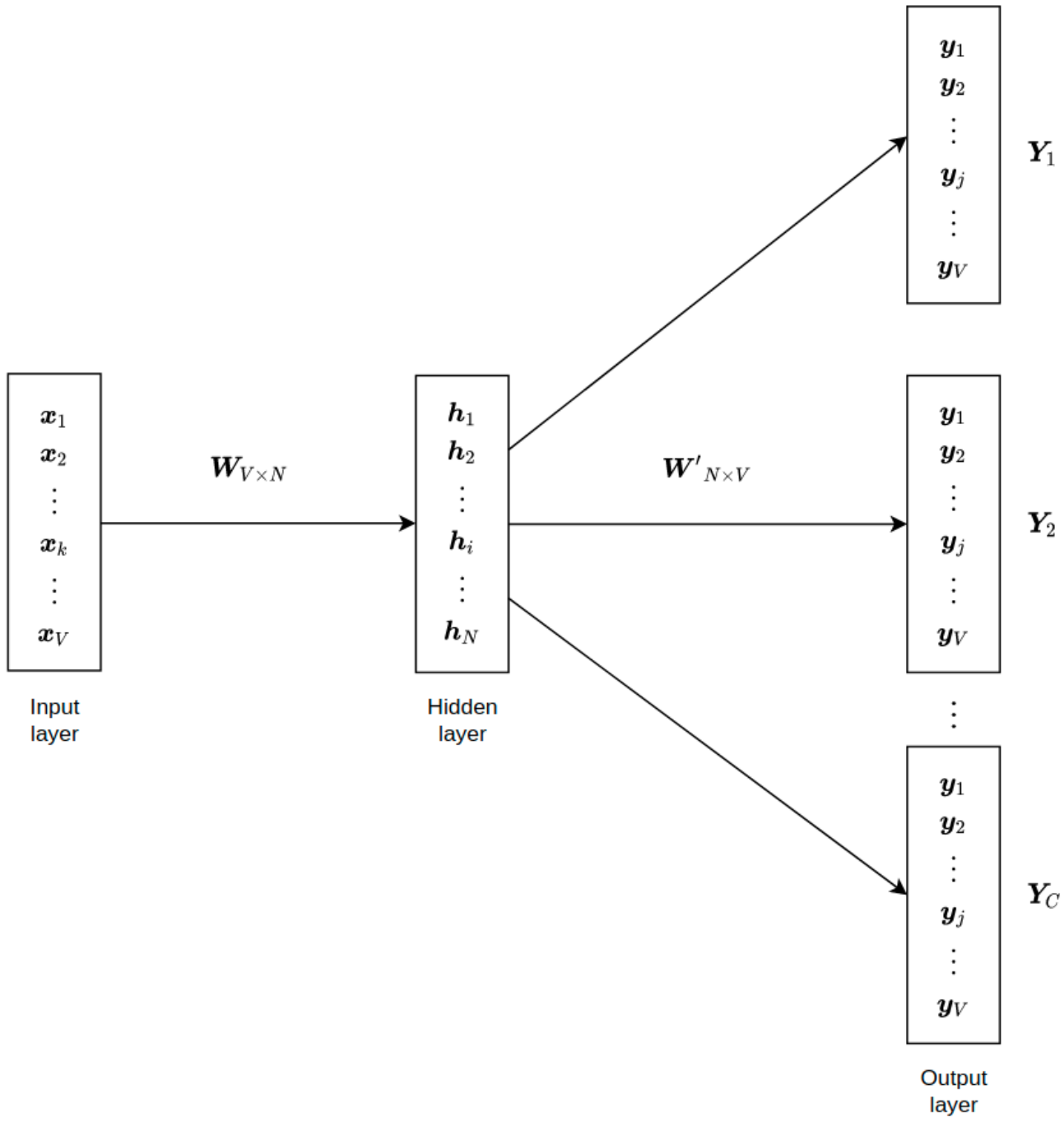


Hình 10: Kiến trúc của Skip-gram model 1.

Trong Skip-gram model chúng ta sẽ cố gắng dự đoán các context word dựa vào target word. Ở đây ta chỉ có một target word và sẽ dự đoán ra C context word. Sau quá trình đào tạo chúng ta sẽ lấy weight set giữa input layer và hidden layer $\boldsymbol{W}$ làm vector đại diện cho các từ. Bây giờ chúng ta sẽ tìm hiểu quá trình forward propagation và backward propagation.

1.2.1 Forward propagation

Hình dưới đây là kiến trúc của Skip-gram model, chúng ta có một target word làm đầu vào $\boldsymbol{X}$ và trả về C context word $\boldsymbol{Y}$ ở đầu ra.



Hình 11: Kiến trúc của Skip-gram model 2.

Tương tự như CBOW model, trong quá trình forward propagation, trước tiên chúng ta nhân $\boldsymbol{X}$ cho hidden layer weight $\boldsymbol{W}$.

$$\boldsymbol{H} = \boldsymbol{X}\boldsymbol{W}^T$$

Tương tự như CBOW model, chúng ta có thể viết lại như sau:

$$\boldsymbol{H} = Z_{w_I}$$

ở đây, Z_{w_I} là vector đại diện cho input word w_I.

Tiếp theo, chúng ta cần tính $\boldsymbol{u}_j$, đây chính là điểm số dùng để đánh giá sự tương đồng giữa từ j trong corpus và target word. Cũng giống với CBOW model, $\boldsymbol{u}_j$ được tính:

$$\begin{aligned}\boldsymbol{u}_j &= \boldsymbol{W'}_{ij}^T \cdot \boldsymbol{H} \\ &= Z'_{w_j^T} \cdot \boldsymbol{H}\end{aligned}$$

với $Z'_{w_j^T}$ là vector đại diện cho từ j.

Nhưng không giống với CBOW model chúng ta chỉ cần dự đoán duy nhất một target word. Ở đây, chúng ta sẽ dự đoán tra C context word. Nên để cho tường minh chúng ta có thể ghi lại phương trình trên thành:

$$\boldsymbol{u}_{c,j} = Z'_{w_j^T} \cdot \boldsymbol{H} \text{ với } c = 1, 2, 3, \ldots, C$$

Vì $\boldsymbol{u}_{c,j}$ là điểm số đại diện cho từ thứ j là context word c. Cho nên:

- $\boldsymbol{u}_{1,j}$ là điểm số cho từ thứ j sẽ là context word đầu tiên.
- $\boldsymbol{u}_{2,j}$ là điểm số cho từ thứ j sẽ là context word thứ hai.
- $\boldsymbol{u}_{c,j}$ là điểm số cho từ thứ j sẽ là context word thứ c.

Tiếp theo, chúng ta chuyển toàn bộ các điểm số trên thành xác suất bằng hàm softmax và đại diện bởi $\boldsymbol{y}_{c,j}$.

$$\boldsymbol{y}_{c,j} = \frac{\exp\left(\boldsymbol{u}_{c,j}\right)}{\sum_{j'=1}^{V} \exp\left(\boldsymbol{u}_{j'}\right)} \tag{8}$$

Ở đây, $\boldsymbol{y}_{c,j}$ là xác suất cho từ thứ j trong corpus sẽ là context word thứ c.

Bây giờ chúng ta sẽ tìm hiểu cách tính hàm loss function. Đặt $\boldsymbol{y}^*_{c,j}$ là xác suất để từ được chọn đúng là context word. Chúng ta cần tối đa hóa xác suất này.

$$\max \boldsymbol{y}^*_{c,j}$$

Và cũng tương tự như CBOW model, thay vì tối đa hóa hàm trên, ta sẽ đi tối thiểu hóa hàm *log* và lấy âm.

$$\min\left(-\log\left(\boldsymbol{y}^*_{c,j}\right)\right)$$

Thế phương trình (8) vào phương trình trên ta được:

$$\mathfrak{L} = -\log \frac{\exp\left(\boldsymbol{u}_{c,j}\right)}{\sum_{j'=1}^{V} \exp\left(\boldsymbol{u}_{j'}\right)}$$

Và vì chúng ta có C context word, nên chúng ta sẽ lấy tích các xác suất này với nhau.

$$\begin{aligned}\mathfrak{L} &= -\log \prod_{c=1}^{C} \frac{\exp\left(\boldsymbol{u}_{c,j}\right)}{\sum_{j'=1}^{V} \exp\left(\boldsymbol{u}_{j'}\right)} \\ &= -\sum_{c=1}^{C} \boldsymbol{u}_{c,j} + C \cdot \log \sum_{j'=1}^{V} \exp\left(\boldsymbol{u}_{j'}\right)\end{aligned}$$

1.2.2 Backward propagation

Để tối thiểu hóa loss function, chúng ta áp dụng thuật toán gradient descent. Trong quá trình backward propagation trên network, chúng ta tính toán gradient cho loss function và cập nhật cho weight.

Trước tiên, chúng ta tính toán gradient cho $\boldsymbol{W'}$ bằng chain rule như sau:

$$\frac{\partial \mathfrak{L}}{\partial \boldsymbol{W'}_{ij}} = \sum_{c=1}^{C} \frac{\partial \mathfrak{L}}{\partial \boldsymbol{u}_{c,j}} \cdot \frac{\partial \boldsymbol{u}_{c,j}}{\partial \boldsymbol{W'}_{ij}}$$

Tương tự như CBOW model, đạo hàm của $\dfrac{\partial \mathfrak{L}}{\partial \boldsymbol{u}_{c,j}}$ chính là sự khác biệt giữa actual value và predicted value:

$$\frac{\partial \mathfrak{L}}{\partial \boldsymbol{u}_{j}} = \boldsymbol{e}_{c,j}$$

Tuy nhiên, do chúng ta có nhiều context word, nên chúng ta phải cộng tất cả các $\boldsymbol{e}_{c,j}$ này lại:

$$\frac{\partial \mathfrak{L}}{\partial \boldsymbol{u}_{j}} = \sum_{c=1}^{C} \boldsymbol{e}_{c,j} = E$$

Bây giờ, chúng ta sẽ tính đạo hàm cho $\dfrac{\partial \boldsymbol{u}_j}{\partial \boldsymbol{W'}_{ij}}$, và bởi vì chúng ta biết $\boldsymbol{u}_j = \boldsymbol{W'}_{ij}^{T} \cdot \boldsymbol{H}$, cho nên:

$$\frac{\partial \boldsymbol{u}_j}{\partial \boldsymbol{W'}_{ij}} = \boldsymbol{H}$$

Cho nên, gradient của loss function đối với $\boldsymbol{W'}$ như sau:

$$\frac{\partial \boldsymbol{u}_j}{\partial \boldsymbol{W'}_{ij}} = E \cdot \boldsymbol{H}$$

Bây giờ, chúng ta sẽ tính gradient của loss function đối với $\boldsymbol{W}$. Nó về cơ bản cũng tương tự như CBOW model.

$$\frac{\partial \mathfrak{L}}{\partial \boldsymbol{W}_{ki}} = \frac{\partial \mathfrak{L}}{\partial \boldsymbol{h}_i} \cdot \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{W}_{ki}}$$

Cho nên gradient của loss function so với $\boldsymbol{W}$ và $\boldsymbol{W'}$ như sau:

$$\boldsymbol{W'} = \boldsymbol{W'} - \alpha \boldsymbol{E} \cdot \boldsymbol{H}$$

$$\boldsymbol{W} = \boldsymbol{W} - \alpha \boldsymbol{L} \boldsymbol{H}^{T} \cdot \boldsymbol{X}$$

Kết thúc quá trình đào tạo, chúng ta sẽ tiến hành cập nhật cho các weight set trên network. Sau cùng $\boldsymbol{W}$ đạt trạng thái optimal weight và trở thành word vector cho vocabulary trong corpus.

2 Các chiến lược cải thiện hiệu suất

Để tiến hành cải thiện hiệu suất của quá trình đào tạo model, người ta sẽ sử dụng kèm theo một số kỹ thuật như:

- Sử dụng **Hierarchical softmax** thay vì softmax truyền thống.
- **Negative sampling**.
- **Subsampling frequent words**.

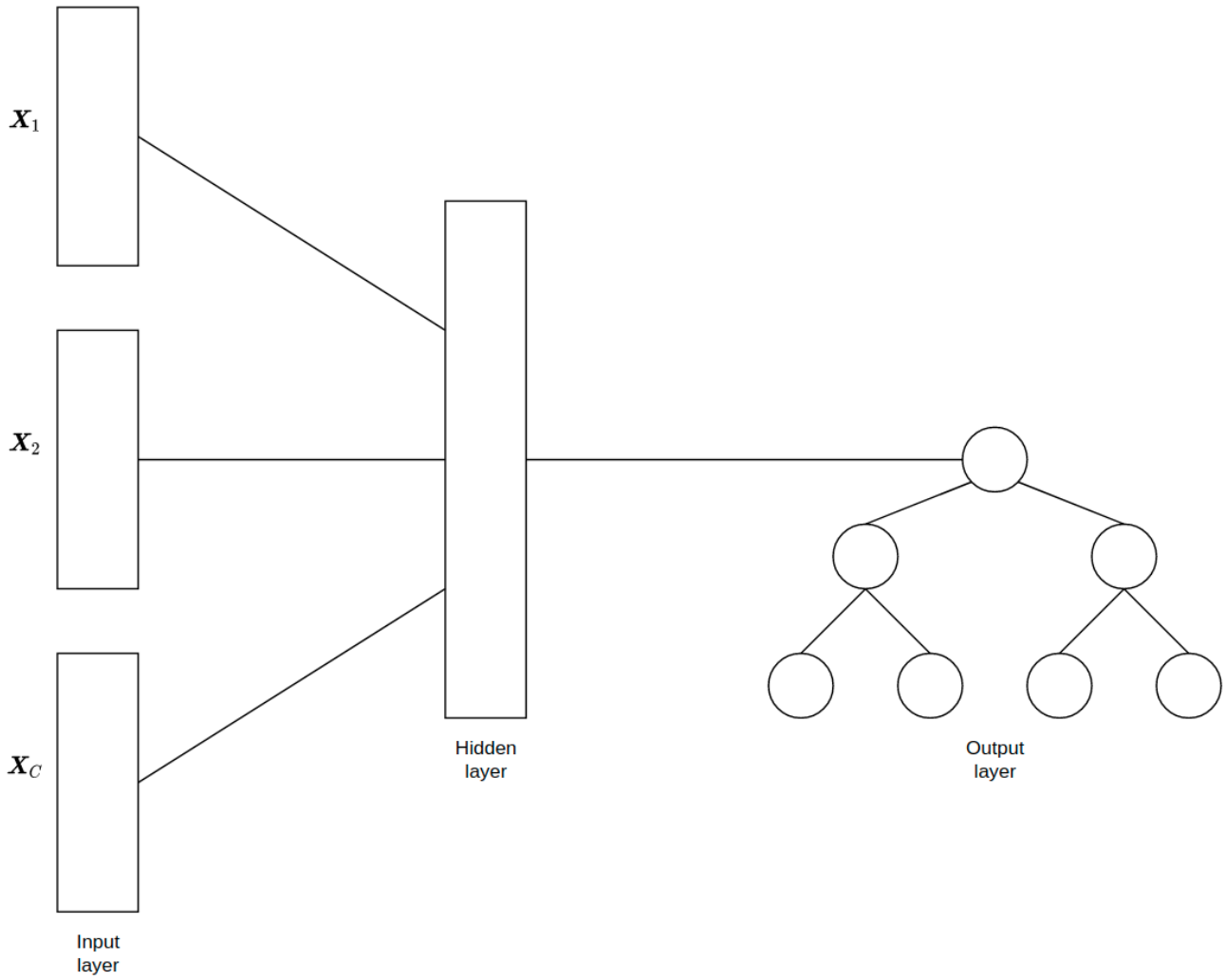
2.1 Hierarchical softmax

Trong cả CBOW model và Skip-gram model ban đầu, chúng ta đều sử dụng hàm softmax để tính toán xác suất để một từ là đầu ra chính xác. Nhưng việc tính toán xác suất bằng hàm softmax rất tốn kém về mặt tài nguyên tính toán. Khi chúng ta đào tạo một CBOW model, chúng ta cần tính toán xác suất của vocabulary thứ j để nó là một target word chính xác:

$$\boldsymbol{y}_j = \frac{\exp\left(u_j\right)}{\sum_{j'=1}^{V} \exp\left(u'_j\right)}$$

Nhìn vào phương trình trên, cơ bản là chúng ta đang tính toán hàm mũ cho tất cả các từ $\boldsymbol{u'}_j$ trong corpus của chúng ta. Độ phức tạp của thuật toán lúc này là $O(V)$. Khi trong corpus của chúng ta có đến hàng triệu từ nó sẽ chắc chắn xảy ra tốn kém về mặt tính toán. Vì vậy, để khắc phục vấn đề này, thay vì tính toán hàm softmax một cách truyền thống người ta thực hiện tính toán softmax trên cây.

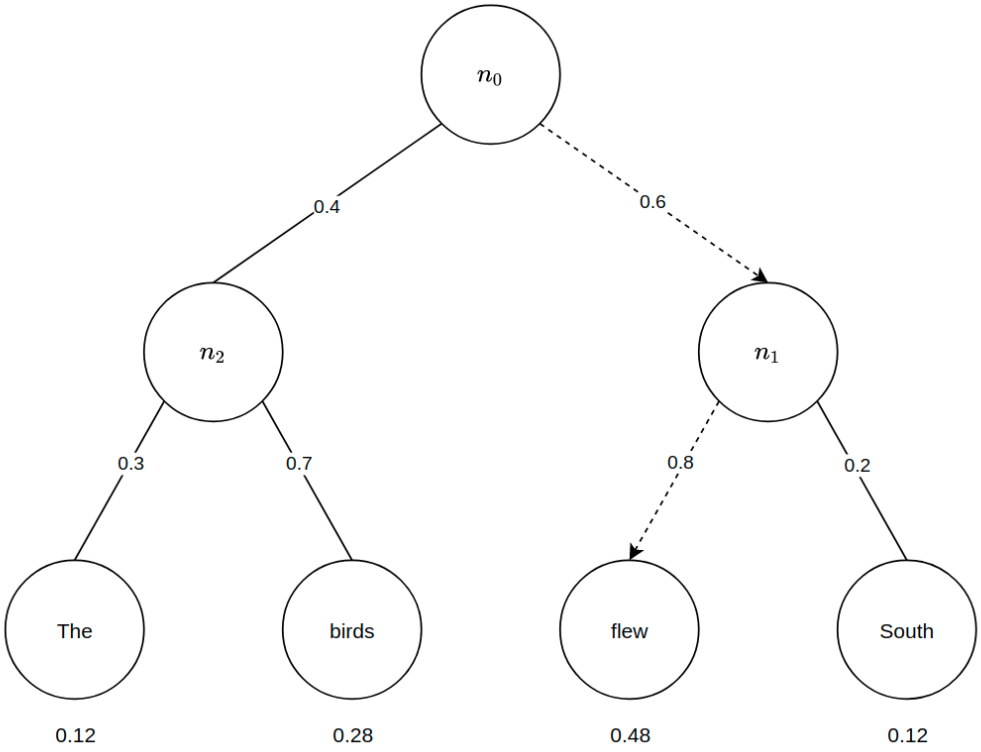
Hierarchical softmax sử dụng **Huffman binary search tree** *(cây tìm kiếm nhị phân Huffman)* để giảm độ phức tạp về mặt tính toán xuống $O(\log_2(V))$. Lúc này kiến trúc mạng có sự thay đổi đáng kể ở output layer - chúng ta sẽ thay output layer bằng một **binary search tree**.



Hình 12: Thay thế output layer bằng Hierarchical softmax trong CBOW model.

Mỗi node lá bây giờ trong cây đại diện cho một từ trong corpus và tất cả các node trung gian đại diện cho **relative probability** *xác suất tương đối* của tất cả các node con của chúng.

Như vậy, làm sao để chúng ta tính toán xác suất của một target word khi được cung cấp context word. Chúng ta chỉ cần duyệt cây và đưa ra quyết định nên duyệt cây bên trái hay bên phải tại một node nhất định. Như thể hiện trong hình dưới đây, chúng ta tính xác suất của từ *flew* là target word dựa vào c context word. Chúng ta lấy tích của tất cả các xác suất dọc theo con đường đến target word.



Hình 13: Tính xác suất của target word trên Hierarchical softmax.

$$p(\textit{flew}|c) = p_{n_0}(left|c) \times p_{n_1}(right|c) = 0.6 \times 0.8 = 0.12$$

2.2 Negative sampling

Xem xét câu: *"Birds are flying in the sky."* với target word là *flying* và còn lại là các context word. Chúng ta cần cập nhật lại các weight trong network mỗi khi nó dự đoán một target word không chính xác. Vì vậy, ngoại trừ từ *flying*, nếu một từ khác được dự đoán là target word thì chúng ta sẽ cập nhật network.

Hãy xem xét trường hợp chúng ta có hàng triệu vocabulary trong corpus. Trong trường hợp này, chúng ta cần phải thực hiện hàng triệu lần cập nhật weight cho đến khi đạt được optimal weight - lúc này nó tốn rất nhiều tài nguyên và chi phí.

Để khắc phục điều này, chúng ta sẽ đánh dấu correct target word là positive class và lấy mẫu một vài từ còn lại thuộc negative class. Về cơ bản, chúng ta đang chuyển về bài toán phân loại nhị phân. Lúc này, xác suất để một từ được chọn là negative word.

$$p(w_i) = \frac{frequency(w_i)^{\frac{3}{4}}}{\sum_{j=0}^{n} frequency(w_j)^{\frac{3}{4}}}$$

2.3 Subsampling frequent words

Trong corpus, sẽ có một số từ xuất hiện rất thường xuyên chẳng hạn như *the*, *is*,... và có một số từ lại xuất hiện không thường xuyên cho lắm. Để duy trì sự cân bằng giữa hai yếu tố, người ta

sử dụng kỹ thuật **subsampling** *(lấy mẫu con)*. Vậy nên, chúng ta sẽ loại bỏ một số từ xuất hiện thường xuyên hơn một **threshold** nhất định với xác suất p được tính như sau:

$$p(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$$

ở đây t là threshold và $f(w_i)$ là frequency của tử i.

3 Mã nguồn

3.1 Xây dựng English Word2Vec model bằng Gensim

Ở đây, em sử dụng Gensim package để xây dựng Word2Vec model với data là file data/text.csv. Gensim là một package được sử dụng rộng rãi để xây dựng các Word2Vec model. Trước tiên cần cài đặt Gensim package.

```
# bash
pip install gensim
```

Import các thư viện cần thiết.

```
import warnings
warnings.filterwarnings('ignore')

# data processing
import pandas as pd
import re
from nltk.corpus import stopwords
stopWords = stopwords.words('english')

# modelling
from gensim.models import Word2Vec
from gensim.models import Phrases
from gensim.models.phrases import Phraser
```

Đọc dữ liệu được lưu trong file data/text.csv, có kích thước khoảng 23 MB.

```
data = pd.read_csv('data/text.csv', header=None)
```

Xem qua vài quan sát trong dữ liệu.

```
data.head()
```

	0
0	room kind clean strong smell dogs. generally a...
1	stayed crown plaza april april . staff friendl...
2	booked hotel hotwire lowest price could find. ...
3	stayed husband sons way alaska cruise. loved h...
4	girlfriends stayed celebrate th birthdays. pla...

Định nghĩa hàm dùng để tiền xử lí dữ liệu.

```
def pre_process(text):
  '''
  Lowercase text '''
  text = str(text).lower()

  '''
  Remove các kí tự đặc biệt, số, khoảng trăng thừa '''
  text = re.sub(r'[^A-Za-z0-9\s.]', r'', text)

  '''
  Remove new line characters '''
  text = re.sub(r'\n',r' ',text)

  '''
  Remove stopwords '''
  text = " ".join([word for word in text.split() if word not in stopWords])

  return text
```

Thử áp dụng hàm `pre_process()` lên một mẫu dữ liệu.

```
pre_process(data[0][50])
```

```
'agree fancy. everything needed. breakfast pool hot tub nice shuttle airport later
checkout time. noise issue tough sleep through. awhile forget noisy door nearby noisy
guests. complained management later email credit compd us amount requested would
return.'
```

Pre-processing trên toàn bộ data.

```
data[0] = data[0].map(lambda x: pre_process(x))
```

Xem lại dữ liệu sau khi pre-processing.

```
data[0].head()
```

```
0    room kind clean strong smell dogs. generally a...
1    stayed crown plaza april april . staff friendl...
2    booked hotel hotwire lowest price could find. ...
3    stayed husband sons way alaska cruise. loved h...
4    girlfriends stayed celebrate th birthdays. pla...
Name: 0, dtype: object
```

Gensim yêu cầu input có dạng list of lists. Mỗi quan sát trong dữ liệu là một set of sentences. Cho nên ta cần tách các sentence bằng dấu '.' sau đó bỏ chúng vào list. Ví dụ: text = [[word1, word2, word3], [word1, word2, word3]]

```
data[0][1].split('.')[:5]
```

```
['stayed crown plaza april april ',
 ' staff friendly attentive',
 ' elevators tiny ',
 ' food restaurant delicious priced little high side',
 ' course washington dc']
```

Chúng ta đã có một data là một list of sentence. Bây giờ chúng ta cần tách các sentence thành list of words dựa vào kí tự `' '`.

```
corpus = []
for line in data[0][1].split('.'):
  words = [x for x in line.split()]
  corpus.append(words)
```

Bây giờ, input của chúng ta đã có dạng chuẩn của Gensim yêu cầu là list of lists.

```
corpus[:2]
```

```
[['stayed', 'crown', 'plaza', 'april', 'april'],
 ['staff', 'friendly', 'attentive']]
```

Các đoạn code trên chúng ta chỉ thử trên một sentence, bây giờ chúng ta sẽ chuyển đổi toàn bộ dataset.

```
data = data[0].map(lambda x: x.split('.'))

corpus = []
for i in (range(len(data))):
  for line in data[i]:
    words = [x for x in line.split()]
    corpus.append(words)

corpus[:2]
```

```
[['room', 'kind', 'clean', 'strong', 'smell', 'dogs'],
 ['generally', 'average', 'ok', 'overnight', 'stay', 'youre', 'fussy']]
```

Giả sử chúng ta có từ `'new york'`, chúng ta sẽ thêm dấu gạch `'_'` để thay cho khoảng trắng. Chúng ta thiết lập `min_count=25`, tức ta bỏ qua tất cả các từ có tần số xuất hiện dưới 25.

```
phrases = Phrases(sentences=corpus, min_count=25, threshold=50)
bigram = Phraser(phrases)

for index,sentence in enumerate(corpus):
  corpus[index] = bigram[sentence]
```

Kiểm tra dữ liệu.

```
corpus[111]
```

```
['connected', 'rivercenter', 'mall', 'downtown', 'san_antonio']
```

Bây giờ chúng ta sẽ build model word2vec, trước tiên chúng ta cần xác định một vài hyper-params.

- `vector_size`: kích thước của embedding vector. Tùy chỉnh dựa vào kích thước của dataset mà ta có.
- `window_size`: kích thước của cửa sổ context words.
- `min_count`: tần số thấp nhất của từ trong dataset.
- `workers`: chỉ định số nhân CPU dùng để training model.
- `sg`: bằng 1 sẽ dùng skip-gram method để training, nếu 0 thì chỉ định CBOW method để training.

```
vector_size = 100
window_size = 2
epochs = 100
min_count = 2
workers = 4
sg = 1
```

Training model.

```
  model = Word2Vec(corpus, sg=sg, window=window_size, vector_size=vector_size, min_count=
    ↪ min_count, workers=workers, epochs=epochs)
```

Lưu lại Word2Vec model vào folder `model`.

```
model.save('./model/word2vec.model')
```

Đánh giá Word2Vec model. Gensim cung cấp phương thức `most_similar()` sẽ cho ta biết những từ nào tương đồng với một từ được cung cấp. Dưới đây từ `san_deigo` được cung cấp là đầu vào, ta cần tìm các từ mà có sự tương đồng với từ này.

```
model.wv.most_similar('san_diego')
```

```
[('san_francisco', 0.7817285656929016),
 ('san_antonio', 0.7652156352996826),
 ('phoenix', 0.7618695497512817),
 ('seattle', 0.7534513473510742),
 ('memphis', 0.7474998235702515),
 ('dallas', 0.7412389516830444),
 ('sd', 0.7398099303245544),
 ('austin', 0.7396041750907898),
 ('boston', 0.7347566485404968),
 ('sf', 0.7248415946960449)]
```

Có thể áp dụng các phép toán số học trên vector, ví dụ:

$$woman + king - man = queen$$

```
model.wv.most_similar(positive=['woman', 'king'], negative=['man'], topn=1)
```

```
[('queen', 0.7189985513687134)]
```

Chúng ta có thể tìm hiểu xem từ nào là không phù hợp với đa số các từ được cung cấp trong list.

```
text = ['los_angeles','indianapolis', 'holiday', 'san_antonio','new_york']
model.wv.doesnt_match(text)
```

```
'holiday'
```

Sử dụng tensorboard để trực quan hóa word2vec. Import các thư viện cần thiết.

```
import warnings
warnings.filterwarnings(action='ignore')


import tensorflow as tf
from tensorflow.contrib.tensorboard.plugins import projector
tf.logging.set_verbosity(tf.logging.ERROR)

import numpy as np
import gensim
import os
```

Load model Word2Vec.

```
file_name = "./model/word2vec.model"
model = gensim.models.keyedvectors.KeyedVectors.load(file_name)
```

Sau khi load model thành công, chúng ta cần lưu lại số từ vựng có trong model.

```
max_size = len(model.wv.key_to_index.keys())-1
```

Chúng ta biết rằng số chiều của các word vector là $\boldsymbol{V} \times \boldsymbol{N}$. Tức số từng vựng $\boldsymbol{V}\times$ số neuron trong hidden layer $\boldsymbol{N}$. Bây giờ chúng ta sẽ tạo ma trận $\boldsymbol{W}$ và lưu vào biến `w2v`.

```
w2v = np.zeros((max_size, model.layer1_size))
```

Tiếp theo, chúng ta sẽ lưu các từ cùng các word vectors vào file `metadata.tsv`.

```
if not os.path.exists('projections'):
  os.makedirs('projections')

with open("projections/metadata.tsv", 'w+') as file_metadata:
  for i, word in enumerate(model.wv.index_to_key[:max_size]):
    # lưu embedding vector
    w2v[i] = model.wv[word]

    # lưu word tương ứng
    file_metadata.write(word + '\n')
```

Khởi tạo TensorFlow session.

```
sess = tf.InteractiveSession()
```

Tạo biến `embeddings` để chứa các từ nhúng và các thiết lập cần thiết cho TensorBoard.

```
with tf.device("/gpu:0"):
  embedding = tf.Variable(w2v, trainable=False, name='embedding')

# cấu hình cần thiết
tf.global_variables_initializer().run()
saver = tf.train.Saver()
writer = tf.summary.FileWriter('projections', sess.graph)

config = projector.ProjectorConfig()
embed = config.embeddings.add()
```

Chỉ định tập tin `metadata.tsv` để trực quan hóa.

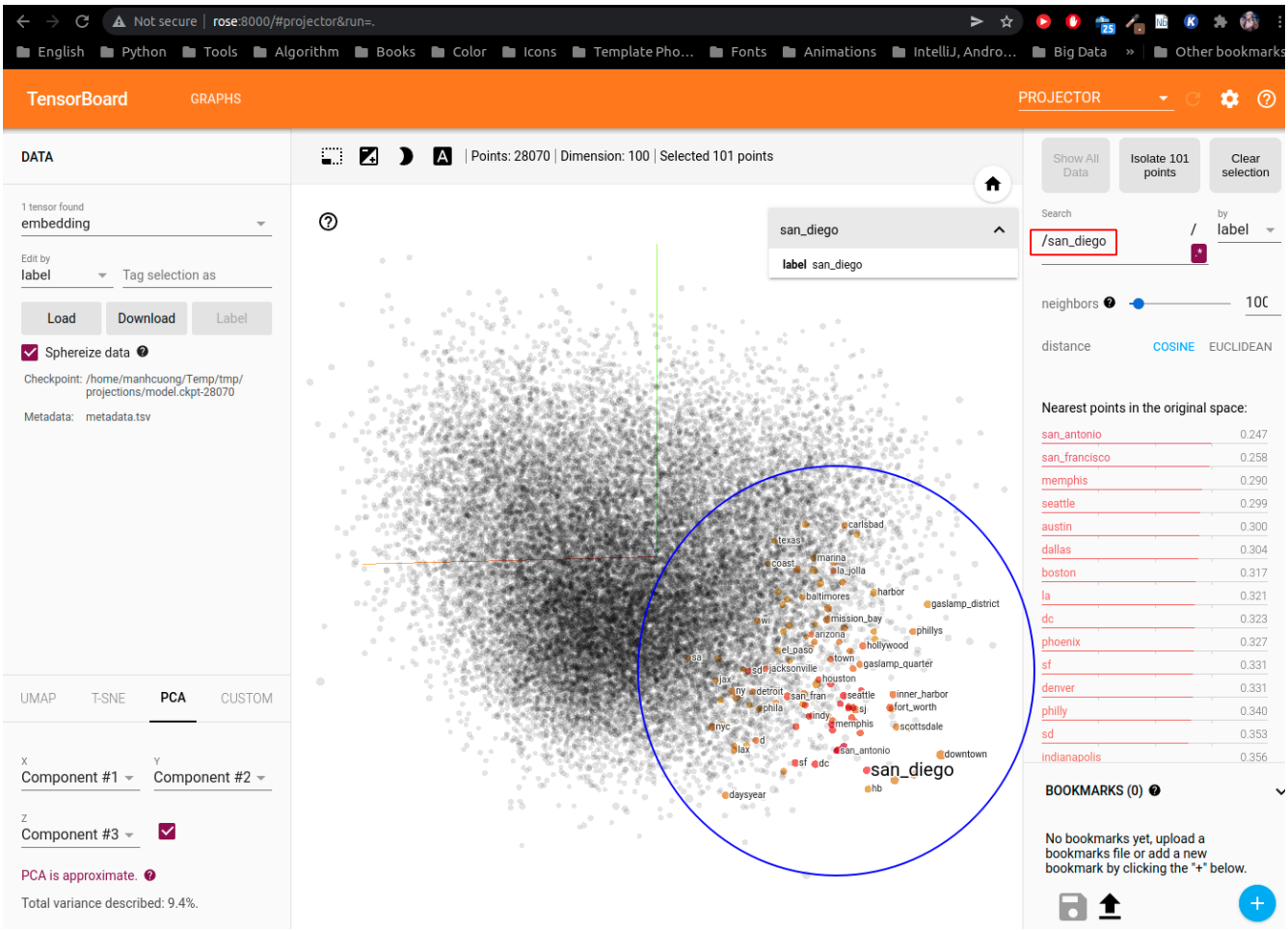
```
embed.tensor_name = 'embedding'
embed.metadata_path = 'metadata.tsv'
```

Lưu lại model dùng để visualization của TensorBoard.

```
projector.visualize_embeddings(writer, config)
saver.save(sess, 'projections/model.ckpt', global_step=max_size)
```

Mở terminal và nhập lệnh dưới đây để xem kết quả trực quan.

```
# bash
tensorboard --logdir=projections --port=8000
```



Nhận xét

- Các từ này được giảm chiều từ 100 xuống 3 bằng PCA và biểu diễn trong không gian 3 chiều.
- Khi ta tìm từ `'san_diego'` ta sẽ thấy được các từ được cho là tương tự với từ này trong TensorBoard (các từ nằm trong vùng xanh).

3.2 Xây dựng Word2Vec model sử dụng build-in model của Báo Mới

Import các package cần thiết.

```
import warnings
warnings.filterwarnings('ignore')

# data processing
import pandas as pd
```

```
import re
from nltk.corpus import stopwords
stopWords = stopwords.words('english')

# modelling
from gensim.models import Word2Vec, KeyedVectors
from gensim.models import Phrases
from gensim.models.phrases import Phraser
```

Load Vietnamese Word2Vec model.

```
  model = KeyedVectors.load_word2vec_format('./model/baomoi.vn.model.bin', binary=True)
```

Đánh giá Word2Vec model. Gensim cung cấp phương thức `most_similar()` sẽ cho ta biết những từ nào tương đồng với một từ được cung cấp. Dưới đây từ `san_deigo` được cung cấp là đầu vào, ta cần tìm các từ mà có sự tương đồng với từ này.

```
model.most_similar('khoa_học_tự_nhiên')
```

```
[('khtn', 0.752849817276001),
 ('khoa_học_cơ_bản', 0.6600980162620544),
 ('khoa_học_xã_hội', 0.6187462210655212),
 ('khxh&nv', 0.6083912253379822),
 ('khxhnv', 0.5912861227989197),
 ('khxh', 0.5885123610496521),
 ('toán_học', 0.5577161908149719),
 ('khoa_học_máy_tính', 0.5532575249671936),
 ('sư_phạm', 0.5512712001800537),
 ('y_sinh_học', 0.5470149517059326)]
```

Có thể áp dụng các phép toán số học trên vector, ví dụ:

$$\text{toán} + \text{khoa\_học} - \text{kinh\_tế} = \text{toán\_tin}$$

```
  model.most_similar(positive=['toán', 'khoa_học'], negative=['kinh_tế'], topn=10)
```

```
[('toán_tin', 0.4689362347126007),
 ('anh_văn', 0.4674587845802307),
 ('tin_học', 0.46048691868782043),
 ('vật_lý', 0.456289678812027),
 ('y_sinh_học', 0.4422585964202881),
 ('môn_sinh_học', 0.4390445649623871),
 ('khoa_học_tự_nhiên', 0.4379250109195709),
 ('Đại_số', 0.43606090545654297),
 ('môn_toán', 0.4359796345233917),
 ('khoa_học_máy_tính', 0.43592217564582825)]
```

Chúng ta có thể tìm hiểu xem từ nào là không phù hợp với đa số các từ được cung cấp trong list.

```
  text = ['máy_tính','khoa_học_máy_tính', 'công_nghệ_phần_mềm', 'kinh_tế', 'đại_số']
model.doesnt_match(text)
```

```
'kinh_tế'
```

Data visualization cho Word2Vec model của Báo Mới. Import các package cần thiết.

```
import warnings
warnings.filterwarnings(action='ignore')


import tensorflow as tf
from tensorflow.contrib.tensorboard.plugins import projector
tf.logging.set_verbosity(tf.logging.ERROR)

import numpy as np
import gensim
import os
```

Load Word2Vec model

```
  file_name = "./model/baomoi.vn.model.bin"
  model = gensim.models.keyedvectors.KeyedVectors.load_word2vec_format(file_name, binary=True)
```

Sau khi load model thành công, chúng ta cần lưu lại số từ vựng có trong model.

```
  max_size = len(model.key_to_index.keys())-1
```

Chúng ta biết rằng số chiều của các word vector là $\boldsymbol{V} \times \boldsymbol{N}$. Tức số từng vựng $\boldsymbol{V}\times$ số neuron trong hidden layer $\boldsymbol{N}$. Bây giờ chúng ta sẽ tạo ma trận $\boldsymbol{W}$ và lưu vào biến `w2v`.

```
  w2v = np.zeros((max_size, model.vector_size))
```

Tiếp theo, chúng ta sẽ lưu các từ cùng các word vectors vào file `metadata.tsv`.

```
  if not os.path.exists('projections_vi'):
  os.makedirs('projections_vi')

  with open("projections_vi/metadata.tsv", 'w+') as file_metadata:
  for i, word in enumerate(model.index_to_key[:max_size]):
  # lưu embedding vector
  w2v[i] = model[word]

  # lưu word tương ứng
  file_metadata.write(word + '\n')
```

Khởi tạo TensorFlow session.

```
sess = tf.InteractiveSession()
```

Tạo biến `embeddings` để chứa các từ nhúng và các thiết lập cần thiết cho TensorBoard.

```
  with tf.device("/gpu:0"):
  embedding = tf.Variable(w2v, trainable=False, name='embedding')

  # run TensorFlow session and set up some configuration
  tf.global_variables_initializer().run()
  saver = tf.train.Saver()
  writer = tf.summary.FileWriter('projections_vi', sess.graph)

  config = projector.ProjectorConfig()
  embed = config.embeddings.add()
```

Chỉ định tập tin để `metadata.tsv` để trực quan hóa bằng TensorBoard.

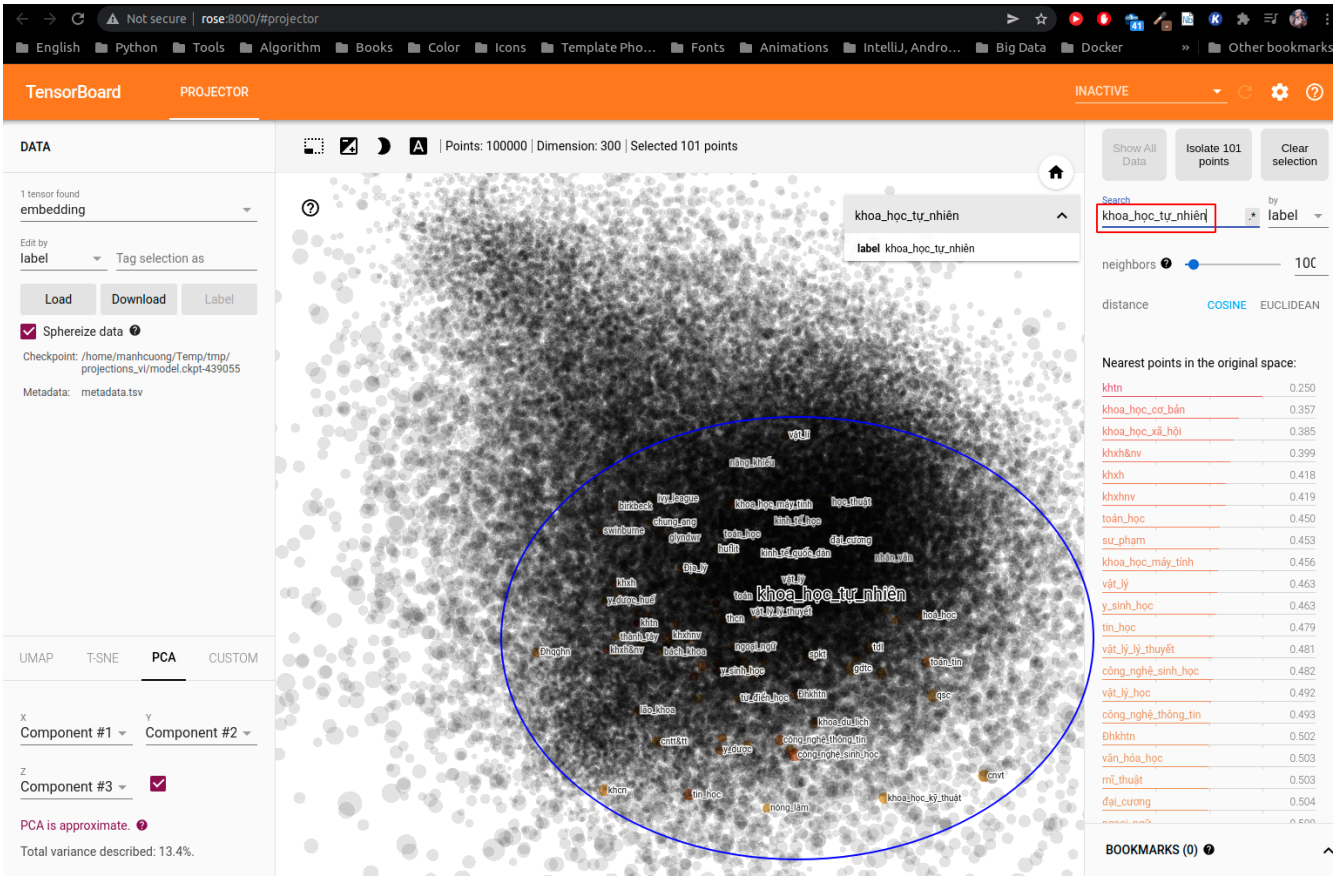
```
  embed.tensor_name = 'embedding'
  embed.metadata_path = 'metadata.tsv'
```

Lưu lại model dùng để visualization của TensorBoard.

```
projector.visualize_embeddings(writer, config)
saver.save(sess, 'projections_vi/model.ckpt', global_step=max_size)
```

Mở terminal và nhập lệnh dưới đây để trực quan hóa.

```
  # bash
  tensorboard --logdir=projections_vi --port=8000
```



Nhận xét

- Các từ này được giảm chiều từ 300 xuống 3 bằng PCA và biểu diễn trong không gian 3 chiều.
- Khi ta tìm từ `'khoa_học_tự_nhiên'` ta sẽ thấy được các từ được cho là tương tự với từ này trong TensorBoard (các từ nằm trong vùng xanh).

4 Tài nguyên tham khảo

- **Hands-On Deep Learning Algorithms with Python** - *(chương 7)*, link sách tại https://www.amazo Deep-Learning-Algorithms-Python/dp/1789344158.
- **Natural Language Processing in Action**, link sách tại https://www.amazon.com/Natural-Language-Processing-Action-Understanding/dp/1617294632.