ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

WORD EMBEDDING

Sinh viên thực hiện:		
STT	Họ tên	MSSV
1	Nguyễn Lộc Linh	19521754
2	Trần Minh Đạt	19521352
3	Nguyễn Ngọc An	19521182

TP. HỒ CHÍ MINH – 1/2021

1. SƠ LƯỢC WORD EMBEDDING

1.1. Khái niệm word embedding

Word Embedding là tên gọi chung của mô hình ngôn ngữ và các phương pháp học trong lĩnh vực Xử lý ngôn ngữ tự nhiên (NLP), ở đó các từ hoặc cụm từ được ánh xạ sang các vector số (thường là số thực) trong không gian, trong không gian này các từ có mối liên hệ về ngữ nghĩa sẽ có vị trí gần nhau. Đây là một công cụ có vai trò khá cần thiết đối với đa số các thuật toán, kiến trúc Machine Learning, Deep Learning trong việc xử lý Input ở dạng text, do chúng chỉ có thể hiểu được Input ở dạng là số, từ đó mới thực hiện các công việc phân loại, hồi quy,vv…

1.2. Tầm quan trọng

Đầu tiên, tại sao ta cần phải Vector hóa văn bản. Bình thường thì máy tính không thể hiểu được ý nghĩa của các từ ngữ. Nhưng để có thể xử lý được ngôn ngữ tự nhiên, ta cần có phương pháp để biểu diễn văn bản dưới dạng mà máy tính có thể hiểu được. Dù chúng ta có thể biểu diễn từ ngữ bằng mã ascii, thì nó cũng cần lượng lớn bộ nhớ máy tính. Do đó phương pháp tối ưu hơn để biểu diễn văn bản đó là biểu diễn các văn bản theo vector. Trong đó, các từ trong tập tài liệu ngôn ngữ được ánh xạ thành những vector trên hệ không gian số thực. Một kỹ thuật đơn giản được sử dụng là One hot vector.

Để chuyển đổi ngôn ngữ tự nhiên về dạng 1-of-N (one hot vector), ta thực hiện các bước như sau:

- Xây dựng một bộ từ vựng.
- Mỗi vector đại diện cho một từ có số chiều bằng số từ trong bộ từ vựng. Tùy theo bộ dữ liệu mà số chiều sẽ khác nhau. Trong đó, mỗi vector chỉ có một phần tử duy nhất khác 0(bằng 1) tại vị trí tương ứng với vị trí từ đó trong bộ từ vựng.

Tuy nhiên one hot vector vẫn còn chiếm quá nhiều bộ nhớ, trong khi vẫn chưa thể hiện rõ hơn ý nghĩa của từ. Do đó, có nhiều phương pháp khác được đưa ra để thể hiện ngôn ngữ dưới dạng vector gọi chung là Word Embedding.

2. NỘI DUNG

Để có thể biểu diễn được Word Embedding có 2 phương pháp chủ yếu được hay dùng là **Word Embedding cổ điển** và **Neural Embedding**. Cả hai cách này đều dựa trên một giả thuyết rằng những từ nào xuất hiện trong cùng một văn cảnh, một ngữ nghĩa sẽ có vị trí gần nhau trong không gian mới được biến đổi

2.1. Word Embedding cổ điển

Phương pháp này tính toán mức liên quan về mặt ngữ nghĩa giữa các từ bằng cách thống kê số lần đồng xuất hiện của một từ so với các từ khác. Ví dụ bạn có hai câu như sau:

Gà ăn cám

Gà ăn thóc

=> Ta xây dựng được ma trận đồng xuất hiện của các từ như sau và nhận thấy cơm và cá có ý nghĩa tương đồng nhau nên nó sẽ có vị trí gần nhau trong không gian vector biểu diễn

	gà	cám	thóc	ăn
gà	0	1	0	2
cám	1	0	0	1
thóc	1	0	0	1
ăn	2	1	1	0

Tuy nhiên nhược điểm lớn nhất mà phương pháp này gặp phải đó là khi dữ liệu của ta lớn, một số từ có tần suất xuất hiện lớn nhưng lại không mang nhiều thông tin. Và nếu chúng ta thống kê cả số lượng data này thì tần suất của các từ này sẽ làm mờ đi giá trị của những từ mang nhiều thông tin nhưng ít gặp hơn.

Và để giải quyết vấn đề, có một giải pháp là chúng ta đánh lại trọng số (re-weight) cho dữ liệu sao cho phù hợp với bài toán của mình. Một vài thuật toán rất hay dùng để giải quyết vấn đề này, đó chính là TF_IDF transform, singular value decomposition, ...

2.2. Neural Embedding

2.2.1. Word2Vec

Nếu như phương pháp **Word Embedding cổ điển** tính toán mức liên quan về mặt ngữ nghĩa giữa các từ bằng số lần xuất hiện liền kề của một từ với những từ khác, thì **Neural Embedding** tính toán sự tương đồng ngữ nghĩa giữa các từ để dự đoán từ tiếp theo bằng cách đưa qua một mạng neural network, mạng neural này hoạt động dựa trên input là các từ xung quanh. Ví dụ cũng với hai câu ở ví dụ trên, ban đầu hai từ “thóc” và từ “cá” có thể được khởi tạo ở hai vị trí khá xa với nhau, nhưng khi xét về ngữ cảnh xung quanh, cả hai từ này đều ở gần từ “gà” và “ăn”, có cách sử dụng khá tương đồng (đều là tân từ) thì vị trí của hai từ cơm và cá trong không gian vector phải gần nhau. Có 2 phương pháp predictive method phổ biến đó chính là :

- Continuous Bag-of-Words (CBOW)
- Skip-gram

Hai phương pháp này gọi chung là word2vec.

a) Continuous Bag-of-Words (CBOW)

CBOW model : Cách hoạt động của cbow chính là dự đoán một từ (target word) trong một ngữ cảnh nào đó thông qua một tầng neural đơn giản. Nhờ việc tính toán độ lỗi của output với target word ở dạng one-hot vector, mô hình có thể điều chỉnh weight, học được vector biểu diễn cho target word. Ngữ cảnh ở đây có thể là một hay nhiều từ để dự đoán một từ. Để cho đơn giản thì ta chỉ lấy một cho ngữ cảnh dùng dự đoán một từ. Ví dụ ta có một câu tiếng anh như sau: "I love you". Ta có:

- Input context word: love

- Output target word: you

Input sẽ được biến đổi thành dạng one-hot vector đi qua một tầng hidden layer và thực hiện softmax phân loại để dự đoán từ kế tiếp là gì. Và model training sẽ được như hình dưới.

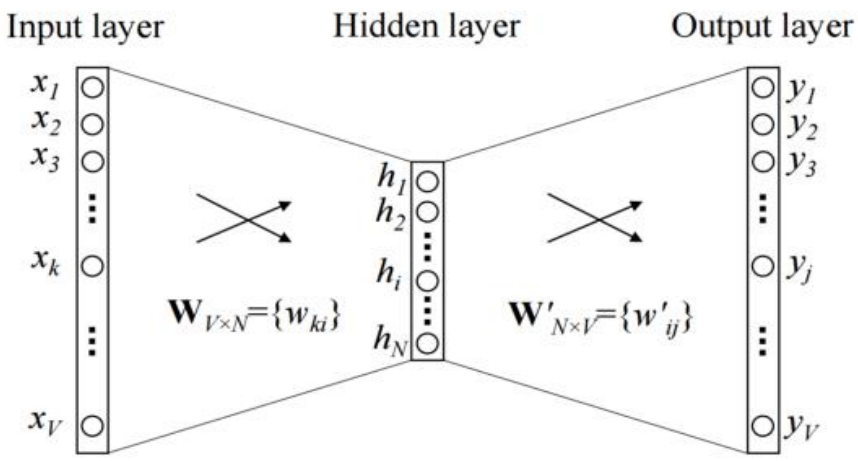


Figure 1: A simple CBOW model with only one word in the context

Kiến trúc mô hình CBOW như hình trên, các vector từ của nhiều từ ngữ cảnh được tính trung bình. Mô hình cố gắng dự đoán từ đích bằng cách cố gắng hiểu ngữ cảnh của các từ xung quanh. Hãy xem xét câu tương tự như trên, 'Đó là một ngày dễ chịu' Mô hình chuyển đổi câu này thành các cặp từ theo mẫu (từ ngữ theo ngữ cảnh, từ khóa mục tiêu). Người dùng sẽ phải đặt kích thước cửa sổ. Nếu cửa sổ cho từ ngữ cảnh là 2 thì các cặp từ sẽ trông như thế này: ([nó, a], là), ([là, dễ chịu], a), ([a, ngày], dễ chịu). Với các cặp từ này, mô hình cố gắng dự đoán từ đích được coi là các từ ngữ cảnh.

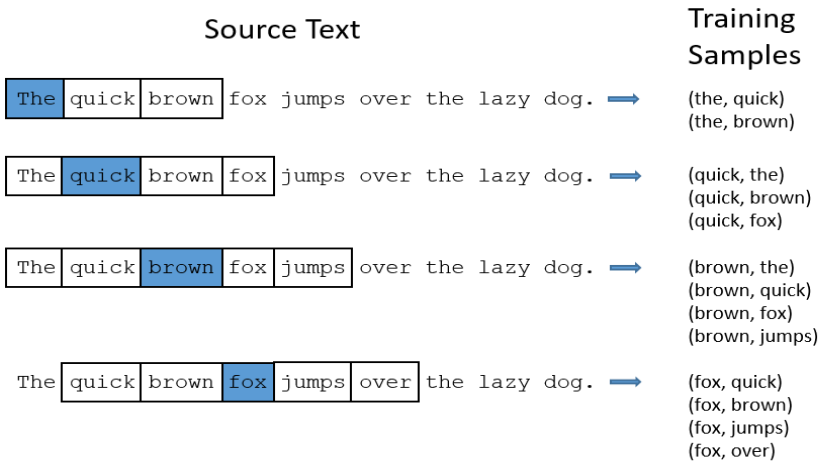
b) Skip-gram

Ngược lại với **CBOW**, mô hình **Skip-gram** có input là một từ target word và cố gắng dự đoán ra các từ lân cận với nó. Số lượng từ xung quanh của target word là tham số window size. Ví dụ có một câu sau: "Tôi thích ăn cua hoàng đế", và target word ban đầu là từ “cua”. Với kích thước window size = 2, ta sẽ có các từ lân cận là (thích, ăn, hoàng, đế). Chúng ta sẽ có bốn cặp input-output sau: (cua, thích), (cua, ăn), (cua, hoàng), (cua, đế).

Chúng ta sẽ train một neural network đơn giản chỉ có một hidden layer để thực hiện một công việc gì đó, nhưng ta sẽ ko thực hiện công việc này sử dụng neural network. Thay vào đó, ta chỉ cần tìm trọng số của hidden layer, chính những trọng số này là những vector đại diện cho từ mà mình cần học.

Ta sẽ train một mạng nơ-ron làm những việc sau. Cho một từ cụ thể ở giữa câu (input), tìm những từ xung quanh và chọn ngẫu nhiên. Mạng nơ-ron này sẽ cho t biết tỉ lệ của mỗi từ trong tập dữ liệu sẽ trở thành từ ở gần từ input. Từ này có thể ở phía trước hay ở sau từ input trong khoảng cách nhất định đặt là window size. vd: nếu window size = 3 thì những từ lân cận từ input sẽ nằm trong khoảng 3 từ đứng trước và 3 từ đứng sau, tổng cộng là 6 từ. Giả sử ta chọn từ “Soviet” làm input thì tỉ lệ xuất hiện từ ở gần nó là từ “Union” hay “Russiz” sẽ cao hơn tỉ lệ xuất hiện từ “dưa hấu”.

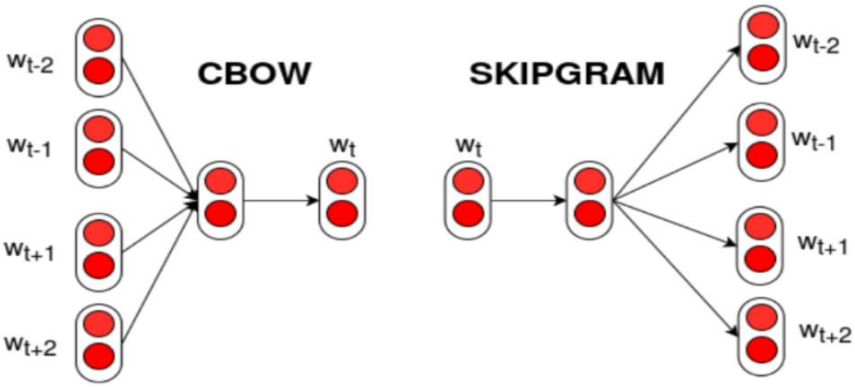
Khi train neural network, ta đưa vào những cặp từ ngữ lân cận xuất hiện trong tài liệu. Ví dụ với câu “the quick brown fox jumps over the lazy dog”, window size = 2, từ màu xanh là input. Như vậy số lần xuất hiện các cặp lân cận nhiều hơn dẫn đến tỉ lệ các cặp từ lân cận sẽ xuất hiện nhiều hơn.



Model training

Chúng ta không thể cứ thế đưa các từ đó vào neural network được, mà phải chuyển những từ này thành thứ mà neural network có thể hiểu được đó là những con số và vector. Những từ này sẽ được biểu diễn bằng one-hot vector. Giả sử ta sẽ tạo ra một từ điển với 10000 từ, và ta muốn biểu diễn 1 từ ví dụ như là “fox” như thế này: nó sẽ là 1 vector 10000 chiều với tất cả giá trị trong đó là 0 ngoại trừ vị trí của “fox” trong từ điển sẽ có giá trị 1.

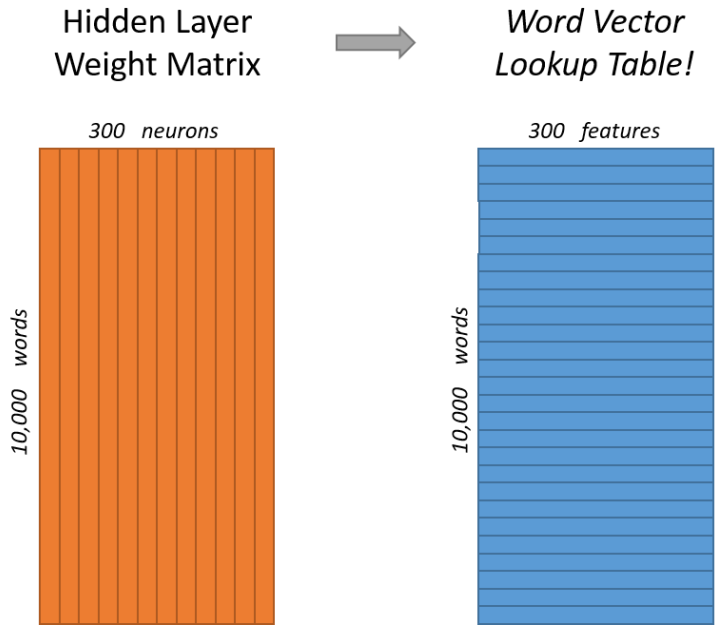
Đầu ra của neural network là một vector được chọn ngẫu nhiên trong tổng số 10000 vector của từ điển với tỉ lệ là từ ở gần từ input.



$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0} logp(w_t|w_{t+j}) \qquad \frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0} logp(w_{t+j}|u$$

The hidden layer

Nếu ta học “word vector” với 300 đặc điểm, thì hidden layer sẽ được biểu diễn bằng ma trận 10000 dòng (tương ứng từng từ trong từ điển) và 300 cột (tương ứng 300 nơ-ron). Con số này do mình tự chọn (giống google) cho phù hợp với bộ dữ liệu.



Nếu để ý thì mỗi dòng sẽ là một word vector mà mình cần tìm. Việc cần làm là học những trọng số trong hidden layer này thôi là xong.

Tuy những one-hot vector gần như chỉ toàn số 0, nhưng nếu ta nhân một vector 1x10000 với một ma trận 10000x300, thì nó sẽ chọn ra dòng tại nơi mà vị trí của nó trong vector bằng 1. Để dễ hình dung thì xem ảnh sau

$$[0 \quad 0 \quad 0 \quad 1 \quad 0] \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = [10 \quad 12 \quad 19]$$

Như vậy hidden layer này đóng vai trò như một bảng tra cứu. Output chỉ là word vector cho input.

The output layer

Vector 1x300 cho từ “fox” sẽ được đưa đến output layer. Output layer này sử dụng thuật toán softmax regression classifier. Đại khái của thuật toán này là mỗi nơ-ron output (1 nơ-ron tương ứng 1 từ trong từ điển) sẽ đưa ra một con số trong khoảng từ 0 đến 1, và tổng tất cả output này là bằng 1.

Cụ thể như thế này, ban đầu khởi tạo hai ma trận trọng số Input-Hidden Weights và Hidden-Output Weights ngẫu nhiên, Input sẽ được nhân với ma trận trọng số Input-Hidden ra được một kết quả gọi là Hidden Activation, kết quả này sẽ được nhân tiếp với một ma trận trọng số nữa là ma trận trọng số Hidden-Output và cuối cùng được đưa vào một hàm softmax để ra được Output là 1 vector xác suất.

$$p(w_k|w_j) = \frac{exp(v'_k . v_j)}{\Sigma_{w' \epsilon |v|} exp(v'_w . v_j)}$$

Output này sẽ được so sánh với Output mong muốn và sau đó tính toán độ lỗi, ta có nhiều độ lỗi do có nhiều vector Output, các độ lỗi này sẽ được tổng hợp lại thành 1 độ lỗi cuối cùng để lan truyền ngược trở lại cập nhật các trọng số. Vector biểu diễn của các từ chính là các trọng số của ma trận Input-Hidden Weights sau khi đã học xong.

Ta da, và thế là xong rồi. Bạn có biết word2vec model còn có thể áp dụng với các dữ liệu khác ngoài chuỗi kí tự ra không? Nó còn được dùng trong công nghiệp quảng cáo và giới thiệu nữa, thay vì học vector qua các câu từ, thì có thể học vector qua chuỗi các hành động trên mạng.

2.2.2. Glove (Global Vector)

Là một phương pháp mới để vector hóa các từ. Sử dụng các vector được huấn luyện bởi **Glove** các mô hình có thể tận dụng thông tin về mối quan hệ ngữ nghĩa giữa các từ tốt hơn. Việc huấn luyện cho **Glove** bao gồm những bước sau.

- Thu thập số liệu thống kê về sự xuất hiện của từ dưới dạng một ma trận vuông đối xứng cụ thể là **co-occurrence matrix**. Mỗi phần tử $\boldsymbol{Xij}$ của ma trận đại diện cho tần suất của xuất hiện của từ i trong ngữ cảnh của từ j hay sự liên quan ngữ nghĩa giữa hai từ i và j. Mối quan hệ của những từ này có thể được kiểm tra bằng cách nghiên cứu tỷ lệ xác suất đồng xuất hiện của chúng với các từ thăm dò khác nhau **Glove** sẽ biểu thị xác suất của từ qua công thức: $\frac{Pik}{Pjk}$

  Trong đó: Pik biểu thị xác suất từ i và từ k xuất hiện cùng nhau.Với cách chia Xik là số lần từ i và từ k xuất hiện cùng nhau cho Xi là tổng số lần từ i xuất hiện trong toàn văn bản. Ví dụ hai từ i và j là "mèo" và "hoa", với từ k là "chó" thì tỉ số $\frac{Pik}{Pjk}$ cao (lớn hơn 1) do từ k có nghĩa gần với i hơn j. Ngược lại tỉ số $\frac{Pik}{Pjk}$ sẽ nhỏ (nhỏ hơn 1) với từ k là "hương thơm" do k gần nghĩa j hơn i. Trong các trường hợp khác thì tỉ số $\frac{Pik}{Pjk}$xấp xỉ 1 do từ k gần nghĩa hoặc không gần nghĩa với cả hai từ i, j.

  Ta sẽ có công thức ràng buộc mềm giữa các cặp từ:

$$w_i^T w_j + b_i + bj = log(X_{ij})$$

Một nhược điểm chính của mô hình này là nó cân bằng tất cả các lần đồng xuất hiện, đồng đều những điều đó hiếm khi xảy ra hoặc không bao giờ. Cho nên ta sẽ đề xuất một mô hình hồi quy bình phương nhỏ nhất có trọng số mới để giải quyết những vấn đề này.

$$J=\sum_{i=1}^{V} \quad \sum_{j=1}^{V} \quad f(X_{ij})(w_i^T w_j + b_j + b_i - logX_{ij})^2$$

Trong đó: V là kích thước của từ vựng. Hàm trọng số phải tuân theo các thuộc tính sau:

1. f (0) = 0. Nếu xem f là một liên tục hàm, nó sẽ biến mất dưới dạng x → 0 nhanh chóng đủ sao cho limx → 0 f (x) log2x là hữu hạn.

2. f (x) không giảm nên rất hiếm các đồng xuất hiện không bị quá tải.

3. f (x) phải tương đối nhỏ đối với các giá trị lớn của x, do đó các trường hợp đồng xuất hiện thường xuyên là không quá cân.Một số lượng lớn các chức năng đáp ứng các thuộc tính, nhưng một lớp hàm mà chúng tôi đã tìm thấy để hoạt động tốt có thể được tham số hóa là

Và cuối cùng để giảm thiểu việc xuất hiện các cặp từ cực kì phổ biến. Một hàm cũng được nhiều chuyên gia lựa chọn là:

$$f(X_{ij}) = \{\left\{(\frac{X_{ij}}{X_{max}})^a, X_{ij} < XMAX\right\} \ 1 \quad , \ otherwise$$

- Về cơ bản, GloVe là một mô hình log-song tuyến tính với mục tiêu bình phương nhỏ nhất có trọng số. Rõ ràng, nó là một phương pháp kết hợp sử dụng học máy dựa trên ma trận thống kê, và đây là sự khác biệt chung giữa GloVe và Word2Vec.

2.3. Tổng kết:

So sánh giữa hai phương pháp **Embedding cổ điển** và **Neural Embedding**, khi chúng ta huấn luyện một bộ dữ liệu lớn, thì **Embedding cổ điển** cần tương đối nhiều bộ nhớ hơn so với **Neural Embedding** do phải xây dựng một ma trận đồng xuất hiện khổng lồ. Tuy nhiên, do nó được xây dựng trên việc thống kê các từ nên khi số lượng dữ liệu của bạn đủ lớn, bạn có thể

train thêm nhiều nhiều dữ liệu nữa mà không lo tăng kích thước của ma trận đồng xuất hiện trong khi tăng độ chính xác của mô hình.

Minh họa Word Embedding với tập dữ liệu

Giả sử cho tập dữ liệu nhỏ như sau:

	text
1	text
2	The future king is the prince
3	Daughter is the princess
4	Son is the prince
5	Only a man can be a king
6	Only a woman can be a queen
7	The princess will be a queen
8	Queen and king rule the realm
9	The prince is a strong man
10	The princess is a beautiful woman
11	The royal family is the king and queen and their children
12	Prince is only a boy now
13	A boy will be a man

Ta sẽ loại bỏ những từ ko cần thiết như là “a”, “an”, “be”, “is”, “are”,.... Giữ lại những từ có ý nghĩa rõ ràng hơn để xử lý.

```
for sentences in corpus:
    tokens = sentences.split()
    for i, token in enumerate(tokens):
        if(i == 0):
            X[words.index(token), words.index(tokens[i + 1])] += 1
        elif(i == len(tokens) - 1):
            X[words.index(token), words.index(tokens[i - 1])] += 1
        else:
            X[words.index(token), words.index(tokens[i + 1])] += 1
            X[words.index(token), words.index(tokens[i - 1])] += 1
```

Sau đó ta, xây dựng nên ma trận đồng xuất hiện của các từ trong dữ liệu và sử dụng phân tách *SVD (Singular Value Decomposition)* để giảm kích thước của vector từ nhằm giúp cho biểu diễn của từ được rõ ràng hơn.

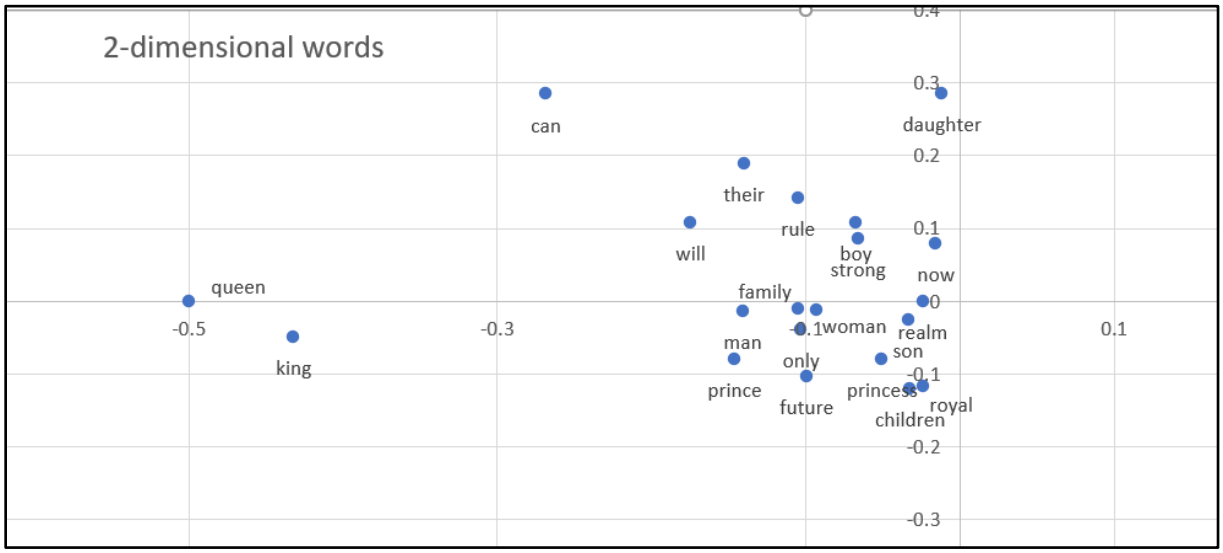
```
U, s, Vh = la.svd(X, full_matrices=False)
```

```
plt.xlim(-0.7, 0,5)
plt.ylim(-0.7, 0.75)

for i in range(len(words)):
    plt.scatter(U[i, 0],U[i, 1])
    plt.text(U[i, 0], U[i, 1], words[i])

plt.show()
```

Sau đó sử dụng ma trận thứ nhất để biểu diễn dữ liệu bởi mỗi dòng của ma trận là một vector biểu diễn được những từ ngữ. Ở đây ta lấy 2 chiều của vector để thể hiện lên biểu đồ.



Từ biểu đồ ta thấy được vài ví dụ hợp lý như: queen ở gần king, son ở gần prince và children, …

3. KẾT LUẬN

Word embedding là một lĩnh vực nghiên cứu cách thể hiện từ ngữ bằng những con số. Đây là bước khởi đầu để đưa vào các mô hình tính toán làm việc với ngôn ngữ tự nhiên. Ngày nay, để thể hiện rõ hơn những từ ngữ, các từ ngữ này sẽ được biểu diễn bằng số lượng lớn vector nhiều chiều hơn, các mô hình được sử dụng ngày càng phức tạp hơn rất nhiều. Tìm hiểu word embedding cho phép ta thực hiện các thao tác với từ ngữ như là tìm từ đồng nghĩa, từ trái nghĩa, thậm chí có thể tính toán cộng trừ nhân chia với các từ ngữ. Hơn nữa đây cũng là tiền đề cho trí tuệ nhân tạo, học sâu,... thực hiện các mục đích phức tạp hơn như truy xuất thông tin, phân loại tài liệu, nhận dạng đồ vật qua tên và còn nhiều ứng dụng thú vị hơn nữa.

TÀI LIỆU THAM KHẢO

[1] https://www.quora.com/How-is-GloVe-different-from-word2vec

[2] https://viblo.asia/p/so-luoc-word-embedding-gDVK2RAeKLj

[3] http://ufldl.stanford.edu/tutorial/supervised/SoftmaxRegression/

[4] https://web.stanford.edu/~jurafsky/li15/lec3.vector.pdf

[5] http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

[6] https://nlp.stanford.edu/pubs/glove.pdf

[7] https://trituenhantao.io/kien-thuc/word-embeddings-cac-phuong-phap-vector-hoa-van-ban/

[8] http://mccormickml.com/2017/01/11/word2vec-tutorial-part-2-negative-sampling/