UỶ BAN NHÂN DÂN THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC SÀI GÒN

♦♦♦

TIỂU LUẬN

MÔN: XỬ LÝ NGÔN NGỮ TỰ NHIÊN

ĐỀ TÀI: CÀI ĐẶT VÀ HUẤN LUYỆN MÔ HÌNH SKIP-GRAM TRÊN NGỮ LIỆU TIẾNG VIỆT

Giảng viên hướng dẫn: PGS. TS. Nguyễn Tuấn Đăng

Thành viên nhóm: Huỳnh Lê Trung Hiếu - 3121411070

TP. HỒ CHÍ MINH, THÁNG 12 NĂM 2024

Mục lục

LỜI CẢM ƠN **3**

Danh mục hình ảnh **4**

Chương 1: Giới thiệu **5**

1.1 Lời mở đầu **5**

1.2 Mục tiêu của đề tài **6**

1.3 Tổng quan về mô hình Skip-gram **7**

1.3.1 Kiến trúc mô hình Skip-gram **7**

1.3.2 Cách hoạt động của mô hình Skip-gram **8**

1.3.3 Ưu điểm của mô hình Skip-gram **9**

1.3.4 Hạn chế của mô hình Skip-gram **9**

1.3.5 Ứng dụng của mô hình Skip-gram **9**

Chương 2: Tiền xử lý dữ liệu **10**

2.1 Thu thập và chuẩn bị dữ liệu tiếng Việt **10**

2.2 Các bước tiền xử lý **10**

Chương 3: Thiết kế mô hình Skip-gram **12**

3.1 Lớp nhúng và ma trận nhúng **12**

3.2 Lớp đầu ra và ma trận trọng số **12**

3.3 Hàm mất mát và phương pháp tính toán **13**

Chương 4: Huấn luyện mô hình **15**

4.1 Các tham số huấn luyện **15**

4.2 Quá trình huấn luyện **16**

Chương 5: Đánh giá và phân tích embedding vector **19**

5.1 Phương pháp đánh giá: Tính toán cosine similarity **19**

5.2 Đánh giá Embedding Vector với các từ đồng nghĩa và trái nghĩa **20**

5.3 Trực quan hóa vector embedding **22**

Chương 6: Phân tích kết quả huấn luyện **24**

6.1 Biểu đồ mất mát theo thời gian **24**

6.2 Phân tích kết quả huấn luyện **24**

Chương 7: Khó khăn và bài học kinh nghiệm **26**

7.1 Các thách thức trong việc triển khai mô hình **26**

7.2 Bài học rút ra **27**

Chương 8: Kết luận **28**

8.1 Tóm tắt **28**

8.2 Đề xuất hướng phát triển trong tương lai **29**

Tài liệu tham khảo **30**

Phụ lục A: Source code chương trình **31**

LỜI CẢM ƠN

Em xin gửi lời cảm ơn chân thành và sâu sắc nhất đến thầy Nguyễn Tuấn Đăng, người đã dành nhiều tâm huyết, kiến thức và sự chu đáo để giảng dạy môn Xử lý ngôn ngữ tự nhiên cho chúng em. Trong suốt thời gian học tập, em đã được thầy chỉ dẫn một cách tận tình, giải đáp mọi thắc mắc và khơi dậy niềm đam mê với lĩnh vực này.

Thầy không chỉ truyền đạt kiến thức một cách khoa học mà còn luôn quan tâm đến sự tiến bộ của từng sinh viên, tạo điều kiện để chúng em có thể tự tin và phát triển bản thân. Những bài giảng của thầy không chỉ là những kiến thức chuyên môn mà còn là nguồn cảm hứng để em luôn cố gắng học tập và trau dồi khả năng của mình.

Đặc biệt, trong quá trình làm tiểu luận, em đã nhận được sự hướng dẫn và động viên từ thầy, giúp em có thêm động lực để hoàn thành tốt nhiệm vụ. Em rất biết ơn vì những giờ học và những lời khuyên bổ ích của thầy, đó là những kỷ niệm quý giá mà em sẽ luôn ghi nhớ.

Một lần nữa, em xin chân thành cảm ơn thầy vì tất cả những gì thầy đã dành cho chúng em. Hy vọng trong tương lai, em sẽ có cơ hội được học hỏi thêm từ thầy và góp phần xây dựng những dự án thú vị trong lĩnh vực Xử lý ngôn ngữ tự nhiên.

Danh mục hình ảnh

Hình 1. Kiến trúc mô hình Skip-gram 7
Hình 2. Xác định cặp từ trung tâm và ngữ cảnh 8
Hình 3. Tiền xử lý dữ liệu 11
Hình 4. Tạo từ điển từ vựng 11
Hình 5. Khởi tạo ma trận nhúng 12
Hình 6. Khởi tạo ma trận trọng số 13
Hình 7. Hàm softmax 13
Hình 8. Hàm mất mát 14
Hình 9. Tính toán và cập nhật gradient 15
Hình 10. Các tham số huấn luyện 16
Hình 11. Khởi tạo mô hình 16
Hình 12. Hàm forward 17
Hình 13. Tính toán lỗi giữa dự đoán và thực tế 18
Hình 14. Tính toán gradient cho ma trận trọng số 18
Hình 15. Cập nhật trọng số W1 và W2 theo gradient descent 19
Hình 16. Huấn luyện mô hình 19
Hình 17. Hàm tính toán cosine similarity giữa hai vector 20
Hình 18. Kiểm tra độ tương đồng giữa các cặp từ 21
Hình 19. Trực quan hoá vector nhúng 22
Hình 20. Biểu đồ trực quan hoá PCA 23
Hình 21. Biểu đồ mất mát theo thời gian 24

Chương 1: Giới thiệu

1.1 Lời mở đầu

Trong thời đại số hóa như hiện nay, xử lý ngôn ngữ tự nhiên (Natural Language Processing - NLP) đã trở thành một trong những lĩnh vực nghiên cứu quan trọng và có ảnh hưởng sâu rộng đến nhiều lĩnh vực khác nhau như công nghệ, giáo dục, y tế, và kinh doanh. Một trong những thách thức lớn nhất của NLP là cách biểu diễn ngữ nghĩa của từ ngữ trong một ngôn ngữ cụ thể. Để giải quyết vấn đề này, các mô hình embedding từ vựng đã được phát triển, trong đó mô hình Skip-gram là một trong những phương pháp tiên phong và hiệu quả.

Mô hình Skip-gram, được giới thiệu bởi Mikolov và cộng sự vào năm 2013, nhằm mục đích học các vector biểu diễn từ (word embeddings) từ một tập dữ liệu văn bản lớn. Bằng cách sử dụng các vector này, các từ có ý nghĩa tương tự sẽ được biểu diễn gần nhau trong không gian vector, giúp cải thiện hiệu suất của các tác vụ NLP như phân loại văn bản, dịch máy, và tóm tắt văn bản. Tuy nhiên, việc áp dụng mô hình Skip-gram trên ngữ liệu tiếng Việt vẫn còn nhiều thách thức, bởi ngôn ngữ này có đặc trưng riêng về ngữ pháp, từ vựng và cách sử dụng.

Đề tài "Cài đặt và huấn luyện mô hình Skip-gram trên ngữ liệu tiếng Việt" nhằm mục đích tìm hiểu và triển khai mô hình Skip-gram từ đầu, mà không dựa vào các thư viện có sẵn. Thông qua đề tài này, chúng ta hy vọng sẽ không chỉ hiểu rõ hơn về cơ chế hoạt động của mô hình Skip-gram mà còn có thể đánh giá được chất lượng của các vector embedding được học trên ngữ liệu tiếng Việt. Đây là một bước quan trọng trong việc phát triển các ứng dụng NLP tiếng Việt, từ đó góp phần thúc đẩy sự phát triển của công nghệ xử lý ngôn ngữ tự nhiên trong nước.

Trong phần tiếp theo của tiểu luận, chúng ta sẽ trình bày chi tiết về các bước tiền xử lý dữ liệu, thiết kế và huấn luyện mô hình Skip-gram, cũng như đánh giá kết quả đạt được. Hy vọng rằng, nghiên cứu này sẽ là một tài liệu tham khảo hữu ích cho những ai quan tâm đến lĩnh vực NLP và muốn áp dụng các mô hình tiên tiến vào ngôn ngữ tiếng Việt.

1.2 Mục tiêu của đề tài

Mục tiêu chính của đề tài "Cài đặt và huấn luyện mô hình Skip-gram trên ngữ liệu tiếng Việt" là xây dựng và triển khai mô hình Skip-gram. Thông qua đề tài này, chúng ta mong muốn đạt được những mục tiêu cụ thể sau:

1. **Hiểu rõ cơ chế hoạt động của mô hình Skip-gram:**
    - Tìm hiểu về kiến trúc mô hình Skip-gram, bao gồm cách thức biểu diễn từ vựng thành vector nhúng (embedding) và cách dự đoán các từ ngữ cảnh xung quanh từ trung tâm.
    - Nắm vững các thành phần của mô hình như lớp nhúng, ma trận trọng số, và hàm mất mát cross-entropy.
2. **Áp dụng mô hình Skip-gram trên ngữ liệu tiếng Việt:**
    - Xây dựng mô hình Skip-gram hoàn toàn từ đầu bằng ngôn ngữ lập trình Python, mà không sử dụng các thư viện có sẵn.
    - Huấn luyện mô hình trên một tập dữ liệu văn bản tiếng Việt, đảm bảo rằng mô hình có thể học được các vector embedding phù hợp với ngữ cảnh ngôn ngữ tiếng Việt.
3. **Đánh giá chất lượng của vector embedding:**
    - Sử dụng phương pháp tương đồng từ vựng (word similarity) để đánh giá chất lượng của các vector embedding được học.
    - Tính toán cosine similarity giữa các cặp từ đồng nghĩa, trái nghĩa, và từ không liên quan để kiểm tra tính hợp lý của các vector nhúng.
4. **Rút ra bài học và đề xuất hướng phát triển:**
    - Phân tích các thách thức gặp phải trong quá trình triển khai mô hình và đưa ra các giải pháp để cải thiện hiệu suất của mô hình.
    - Đề xuất các hướng phát triển tiếp theo như mở rộng tập dữ liệu, tối ưu hóa tham số, hoặc áp dụng các kỹ thuật nâng cao khác.

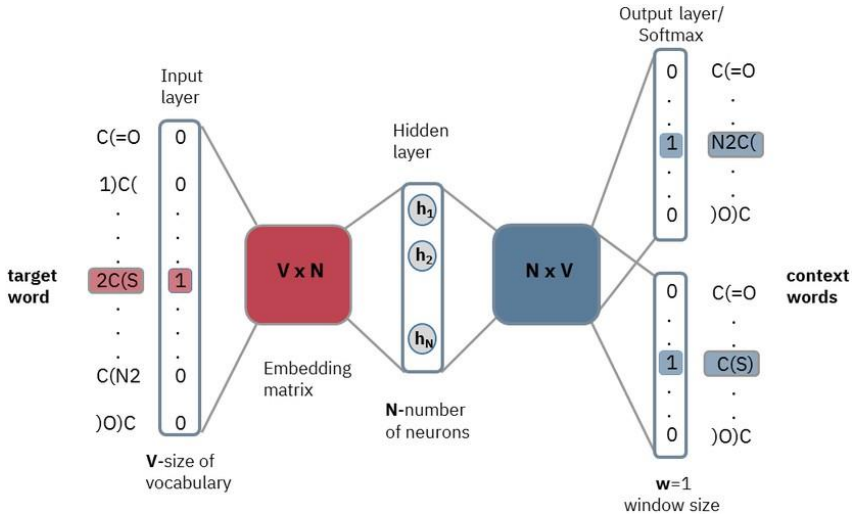
1.3 Tổng quan về mô hình Skip-gram

Mô hình Skip-gram là một trong những phương pháp tiên phong trong việc học các vector biểu diễn từ (word embeddings) từ dữ liệu văn bản. Được giới thiệu bởi Mikolov và cộng sự vào năm 2013, mô hình Skip-gram nhanh chóng trở thành một trong những công cụ quan trọng trong lĩnh vực xử lý ngôn ngữ tự nhiên (NLP). Skip-gram hoạt động dựa trên ý tưởng rằng các từ xuất hiện trong cùng một ngữ cảnh có xu hướng có ý nghĩa tương tự nhau. Vì vậy, mô hình này cố gắng dự đoán các từ ngữ cảnh (context words) xung quanh một từ trung tâm (center word) trong một cửa sổ ngữ cảnh (context window).

1.3.1 Kiến trúc mô hình Skip-gram

Mô hình Skip-gram là một mạng nơ-ron nhân tạo đơn giản, bao gồm hai lớp chính:

1. **Lớp nhúng (Embedding Layer):**
    - Mỗi từ trong từ điển (vocabulary) được biểu diễn bằng một vector nhúng (embedding vector) có kích thước cố định (ví dụ: 100 hoặc 300 chiều).
    - Ma trận nhúng (embedding matrix) có kích thước [V x d], trong đó V là kích thước từ điển (số lượng từ trong từ điển) và d là kích thước của vector nhúng.
2. **Lớp đầu ra (Output Layer):**
    - Lớp đầu ra sử dụng một ma trận trọng số có kích thước [d x V] để dự đoán xác suất của các từ ngữ cảnh xung quanh từ trung tâm.
    - Hàm softmax được áp dụng để chuyển đổi các giá trị dự đoán thành xác suất, với tổng xác suất của tất cả các từ trong từ điển bằng 1.

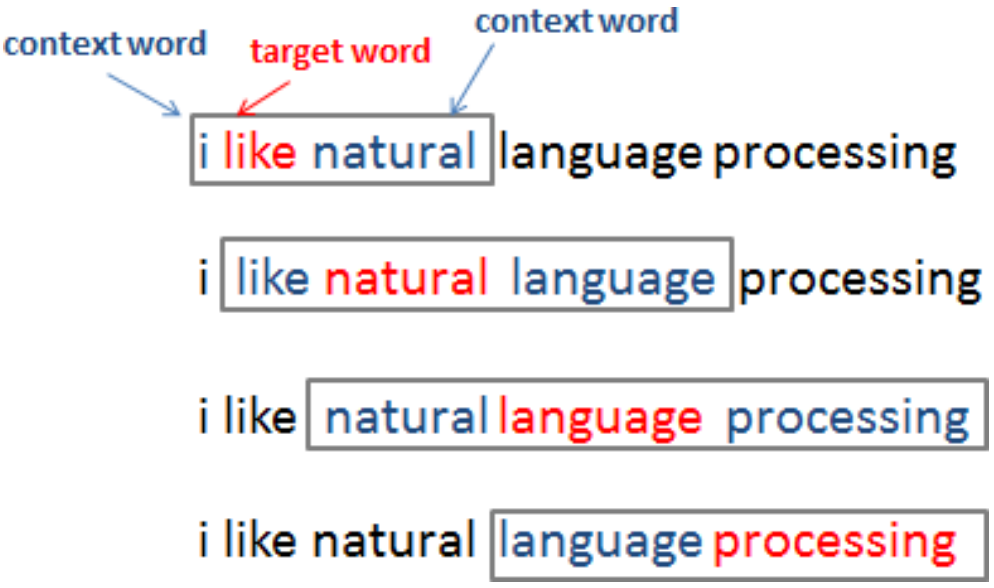


Hình 1. Kiến trúc mô hình Skip-gram

1.3.2 Cách hoạt động của mô hình Skip-gram

Mô hình Skip-gram hoạt động theo các bước sau:

1. **Chọn từ trung tâm và ngữ cảnh:**
    - Với mỗi từ trung tâm trong văn bản, mô hình sẽ xác định các từ ngữ cảnh trong một cửa sổ ngữ cảnh cố định (ví dụ: cửa sổ kích thước 2 có nghĩa là xem xét các từ trước và sau từ trung tâm).
2. **Dự đoán từ ngữ cảnh:**
    - Mô hình sử dụng vector nhúng của từ trung tâm để dự đoán xác suất của các từ ngữ cảnh trong từ điển.
3. **Tối ưu hàm mất mát:**
    - Hàm mất mát (loss function) được sử dụng là cross-entropy, đo lường sự khác biệt giữa xác suất dự đoán và xác suất thực tế của các từ ngữ cảnh.
    - Mô hình sử dụng thuật toán lan truyền ngược (backpropagation) để cập nhật các trọng số của ma trận nhúng và ma trận đầu ra.



Hình 2. Xác định cặp từ trung tâm và ngữ cảnh

1.3.3 Ưu điểm của mô hình Skip-gram

- **Hiệu quả trong việc học embedding:** Skip-gram có khả năng học các vector biểu diễn từ tốt, đặc biệt là trong các tập dữ liệu lớn.
- **Khả năng xử lý từ hiếm (rare words):** Mô hình có thể học được các vector biểu diễn cho các từ hiếm mà không cần quá nhiều dữ liệu huấn luyện.
- **Đơn giản và dễ triển khai:** Skip-gram là một mô hình đơn giản, dễ hiểu và dễ triển khai, đặc biệt là khi không sử dụng các thư viện có sẵn.

1.3.4 Hạn chế của mô hình Skip-gram

- **Tính toán phức tạp:** Việc tính toán xác suất cho tất cả các từ trong từ điển (softmax toàn cục) có thể tốn nhiều tài nguyên, đặc biệt là với các từ điển lớn.
- **Không xử lý tốt các cụm từ (phrases):** Skip-gram chỉ làm việc với các từ đơn lẻ và không thể học được các cụm từ hoặc cấu trúc ngữ pháp phức tạp.

1.3.5 Ứng dụng của mô hình Skip-gram

Mô hình Skip-gram được áp dụng rộng rãi trong nhiều tác vụ NLP như:

- **Phân loại văn bản:** Sử dụng các vector nhúng để biểu diễn văn bản và phân loại chúng.
- **Dịch máy:** Cải thiện chất lượng dịch máy bằng cách sử dụng các vector nhúng để biểu diễn các từ trong các ngôn ngữ khác nhau.
- **Tóm tắt văn bản:** Giúp tóm tắt thông tin từ các văn bản dài bằng cách hiểu ngữ nghĩa của các từ.

Chương 2: Tiền xử lý dữ liệu

2.1 Thu thập và chuẩn bị dữ liệu tiếng Việt

Dữ liệu tiếng Việt được sử dụng trong đề tài này là một tập văn bản được lưu trữ trong file data.txt, chứa khoảng 10.898 từ tiếng Việt và nội dung thiên về lịch sử. Tập dữ liệu này được thu thập từ trang Wikipedia. Mục tiêu của việc thu thập dữ liệu là để cung cấp một nền tảng đầy đủ và đa dạng cho việc huấn luyện mô hình Skip-gram.

Tập dữ liệu data.txt được sử dụng để huấn luyện mô hình Skip-gram, với mục đích học các vector biểu diễn từ (word embeddings) phù hợp với ngữ cảnh ngôn ngữ tiếng Việt. Để đảm bảo chất lượng của mô hình, dữ liệu cần được tiền xử lý một cách cẩn thận, bao gồm các bước như chuyển đổi chữ thường, loại bỏ dấu câu và ký tự đặc biệt, tách từ, và tạo từ điển từ vựng.

2.2 Các bước tiền xử lý:

Tiền xử lý dữ liệu là một bước quan trọng trong quá trình xây dựng mô hình Skip-gram. Các bước tiền xử lý được thực hiện nhằm chuẩn hóa dữ liệu và chuẩn bị cho quá trình huấn luyện mô hình. Dưới đây là các bước tiền xử lý cụ thể:

1. Chuyển đổi chữ thường

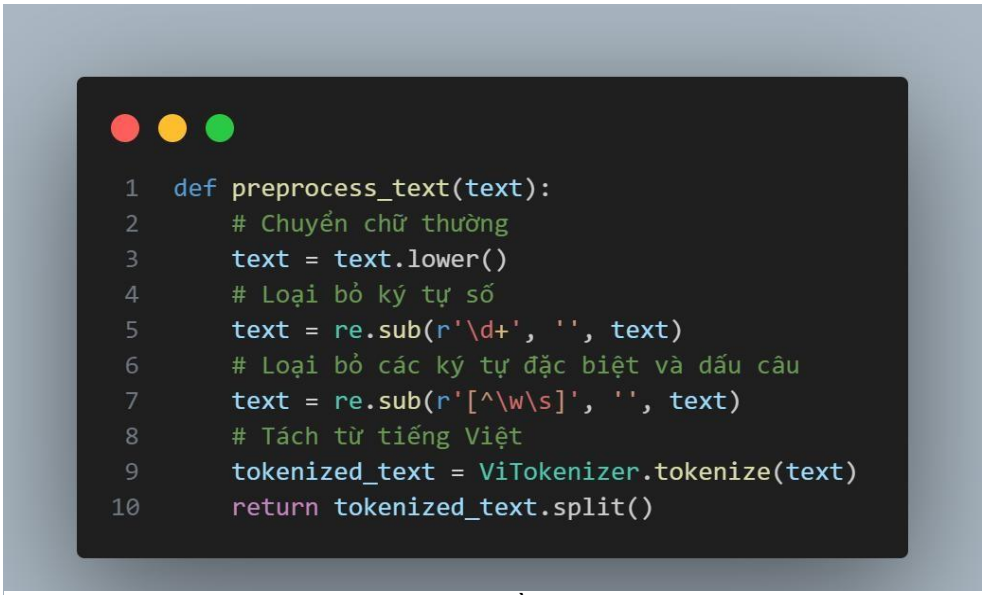
Để đảm bảo tính nhất quán trong việc xử lý dữ liệu, tất cả các từ trong tập dữ liệu được chuyển đổi thành chữ thường. Việc này giúp loại bỏ sự khác biệt giữa chữ hoa và chữ thường, đồng thời giảm kích thước từ điển và tăng hiệu suất của mô hình.

2. Loại bỏ ký tự đặc biệt, dấu câu và ký tự số

Các dấu câu và ký tự đặc biệt không cung cấp thông tin hữu ích cho việc học các vector biểu diễn từ. Do đó, chúng được loại bỏ khỏi tập dữ liệu để giảm nhiễu và tăng chất lượng của dữ liệu đầu vào.

3. Tách từ

Trong tiếng Việt, việc tách từ là một bước quan trọng để chuyển đổi văn bản thành các từ riêng lẻ. Để thực hiện tách từ thì thư viện pyvi, một công cụ mạnh mẽ hỗ trợ tách từ tiếng Việt thường được sử dụng

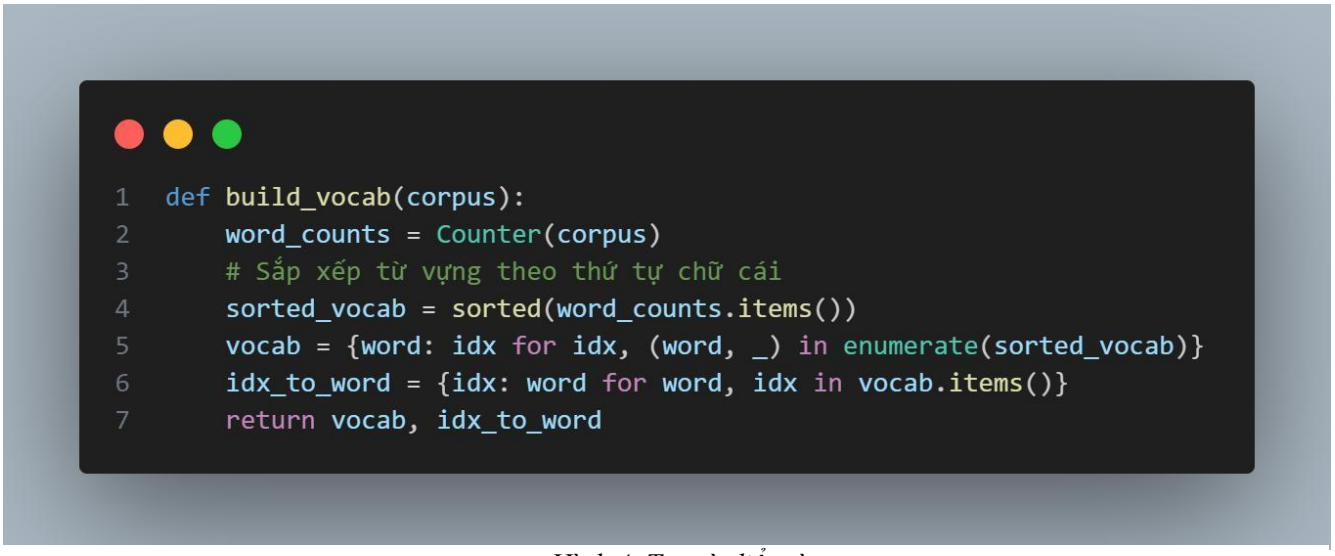


```python
def preprocess_text(text):
    # Chuyển chữ thường
    text = text.lower()
    # Loại bỏ ký tự số
    text = re.sub(r'\d+', '', text)
    # Loại bỏ các ký tự đặc biệt và dấu câu
    text = re.sub(r'[^\w\s]', '', text)
    # Tách từ tiếng Việt
    tokenized_text = ViTokenizer.tokenize(text)
    return tokenized_text.split()
```

Hình 3. Tiền xử lý dữ liệu

4. Tạo từ điển từ vựng

Từ điển từ vựng (vocabulary) là một tập hợp các từ duy nhất trong tập dữ liệu. Từ điển này được sử dụng để ánh xạ mỗi từ với một chỉ số duy nhất, giúp cho việc xử lý và huấn luyện mô hình được hiệu quả hơn. Từ điển được tạo bằng cách đếm số lần xuất hiện của mỗi từ và sắp xếp chúng theo thứ tự chữ cái.



```python
def build_vocab(corpus):
    word_counts = Counter(corpus)
    # Sắp xếp từ vựng theo thứ tự chữ cái
    sorted_vocab = sorted(word_counts.items())
    vocab = {word: idx for idx, (word, _) in enumerate(sorted_vocab)}
    idx_to_word = {idx: word for word, idx in vocab.items()}
    return vocab, idx_to_word
```

Hình 4. Tạo từ điển từ vựng

Sau khi tạo từ điển, từ điển này được lưu vào file vocab.txt để sử dụng trong các bước tiếp theo của quá trình huấn luyện.

Chương 3: Thiết kế mô hình Skip-gram

3.1 Lớp nhúng và ma trận nhúng

Lớp nhúng là thành phần quan trọng nhất của mô hình Skip-gram. Mỗi từ trong từ điển (vocabulary) được biểu diễn bằng một vector nhúng (embedding vector) có kích thước cố định d. Ma trận nhúng W1 có kích thước [V x d], trong đó V là kích thước từ điển (số lượng từ trong từ điển) và d là kích thước của vector nhúng.

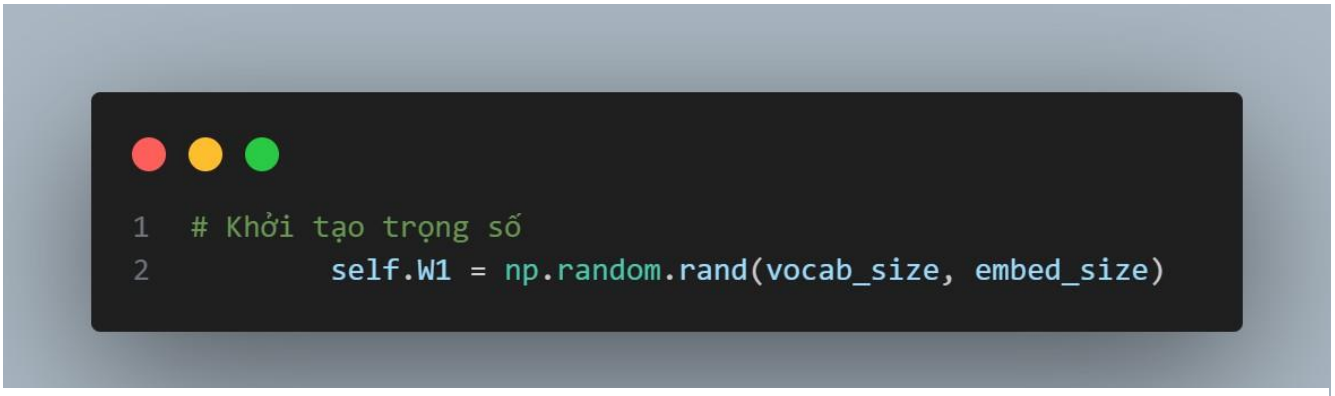
Công thức tính vector nhúng của từ trung tâm w_c là:

$$h = W1[wc]$$

Trong đó:

- w_c là chỉ số của từ trung tâm trong từ điển.
- h là vector nhúng của từ trung tâm.

Trong mã nguồn, ma trận nhúng W1 được khởi tạo ngẫu nhiên và cập nhật trong quá trình huấn luyện:



Hình 5. Khởi tạo ma trận nhúng

3.2 Lớp đầu ra và ma trận trọng số

Lớp đầu ra của mô hình Skip-gram sử dụng ma trận trọng số W2 có kích thước [d x V] để dự đoán xác suất của các từ ngữ cảnh. Công thức tính giá trị đầu ra u là:

$$u = h \times W2$$

Trong đó:

- h là vector nhúng của từ trung tâm.
- u là vector kết quả sau khi nhân ma trận nhúng với ma trận trọng số.

Hàm softmax được áp dụng để chuyển đổi vector u thành xác suất của các từ trong từ điển:

$$y_{pred} = softmax(u)$$

Trong đó:

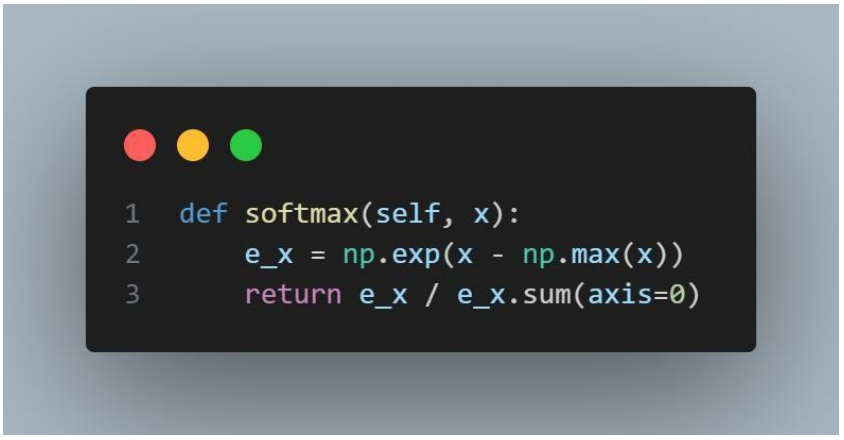
- y_{pred} là xác suất dự đoán của các từ ngữ cảnh.

Trong mã nguồn, ma trận trọng số W2 cũng được khởi tạo ngẫu nhiên và cập nhật trong quá trình huấn luyện:

```
# Khởi tạo trọng số
        self.W2 = np.random.rand(embed_size, vocab_size)
```

Hình 6. Khởi tạo ma trận trọng số

Hàm $softmax$ được triển khai như sau:



```
def softmax(self, x):
    e_x = np.exp(x - np.max(x))
    return e_x / e_x.sum(axis=0)
```

Hình 7. Hàm softmax

3.3 Hàm mất mát và phương pháp tính toán

Hàm mất mát (loss function) được sử dụng trong mô hình Skip-gram là hàm cross-entropy. Hàm này đo lường sự khác biệt giữa xác suất dự đoán và xác suất thực tế của các từ ngữ cảnh. Công thức tính hàm mất mát là:

$$Loss = -\sum_{i=1}^{V} y_{true}[i] log(y_{pred}[i])$$

Trong đó:

- y_{true} là xác suất thực tế của các từ ngữ cảnh (với giá trị 1 cho từ ngữ cảnh đúng và 0 cho các từ khác).
- y_{pred} là xác suất dự đoán của các từ ngữ cảnh.

Trong mã nguồn, hàm mất mát được tính như sau:

Hình 8. Hàm mất mát

Để cập nhật trọng số, thuật toán lan truyền ngược (backpropagation) đã được sử dụng. Công thức tính gradient của hàm mất mát theo các trọng số W1 và W2 là:

$$\frac{\partial Loss}{\partial W_2} = h \times error$$

$$\frac{\partial Loss}{\partial W_1} = error \times W_2^T$$

Trong đó:

- $error$ là vector lỗi được tính bằng cách trừ xác suất dự đoán với xác suất thực tế.

Trong mã nguồn, gradient được tính và cập nhật như sau:

```python
def backward(self, center_word_idx, context_word_idx, y_pred, h):
    error = y_pred
    error[context_word_idx] -= 1  # Cập nhật lỗi cho từ ngữ cảnh

    # Gradient cho W2 và W1
    dW2 = np.outer(h, error)
    dW1 = np.dot(error, self.W2.T)

    # Cập nhật trọng số
    self.W2 -= self.lr * dW2
    self.W1[center_word_idx] -= self.lr * dW1
```

Hình 9. Tính toán và cập nhật gradient

Chương 4: Huấn luyện mô hình

4.1 Các tham số huấn luyện

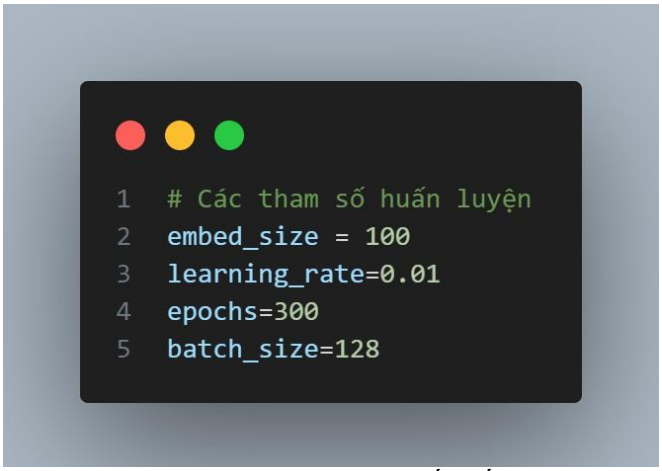
Trong quá trình huấn luyện mô hình Skip-gram, các tham số sau được sử dụng để điều chỉnh hiệu suất của mô hình:

1. **Batch size**: Kích thước của mỗi batch dữ liệu được sử dụng để cập nhật trọng số mô hình. Trong code, batch size được đặt là 128. Batch size ảnh hưởng đến tốc độ huấn luyện và độ chính xác của mô hình. Kích thước batch lớn hơn có thể tăng tốc độ huấn luyện nhưng cũng yêu cầu nhiều bộ nhớ hơn.

2. **Epochs**: Số lần lặp lại quá trình huấn luyện trên toàn bộ tập dữ liệu. Trong code, số lượng epochs được đặt là 300. Số lượng epochs ảnh hưởng đến khả năng hội tụ của mô hình. Quá nhiều epochs có thể dẫn đến hiện tượng overfitting, trong khi quá ít epochs có thể khiến mô hình chưa hội tụ hoàn toàn.

3. **Learning rate**: Tốc độ học (learning rate) quyết định mức độ cập nhật trọng số mô hình trong mỗi bước huấn luyện. Trong code, learning rate được đặt là 0.01. Learning rate quá lớn có thể khiến mô hình không hội tụ, trong khi learning rate quá nhỏ có thể kéo dài thời gian huấn luyện.

4. **Embed size**: Kích thước của vector nhúng (embedding vector) được sử dụng để biểu diễn từ. Trong code, embed size được đặt là 100. Kích thước vector nhúng ảnh hưởng đến khả năng biểu diễn ngữ nghĩa của các từ. Kích thước lớn hơn có thể cải thiện độ chính xác nhưng cũng làm tăng độ phức tạp của mô hình.

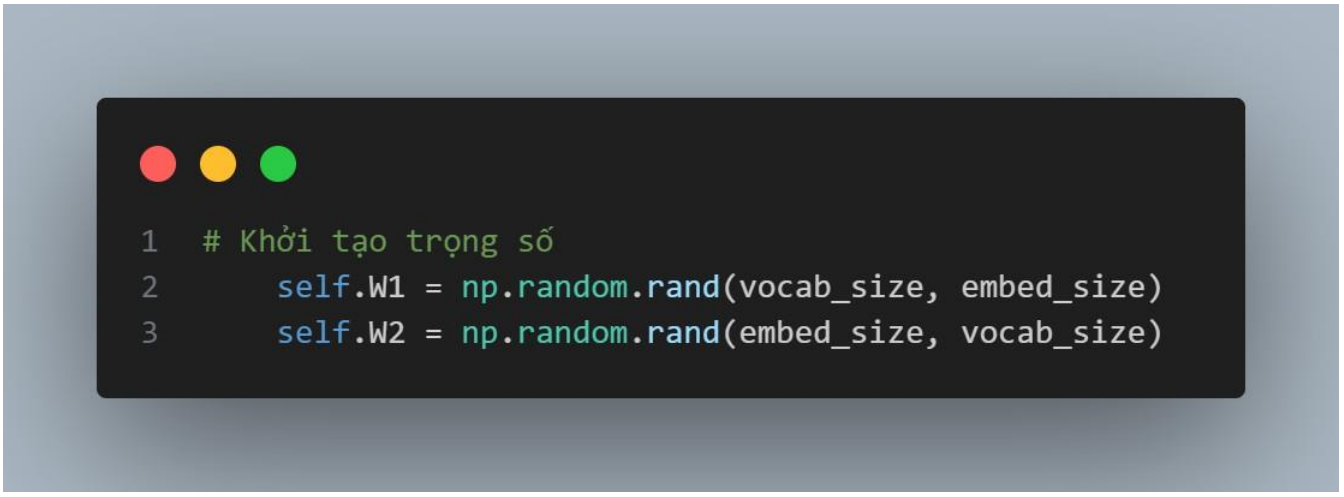


Hình 10. Các tham số huấn luyện

4.2 Quá trình huấn luyện

Quá trình huấn luyện mô hình Skip-gram được thực hiện qua các bước sau:

1. **Khởi tạo mô hình**:
   - Ma trận nhúng W_1 và ma trận trọng số đầu ra W_2 được khởi tạo ngẫu nhiên với kích thước tương ứng là $[V \times d]$ và $[d \times V]$, trong đó V là kích thước từ vựng và d là kích thước vector nhúng:



Hình 11. Khởi tạo mô hình

2. **Forward pass**:

- Với mỗi từ trung tâm w_c, vector nhúng h được lấy từ ma trận W_1:

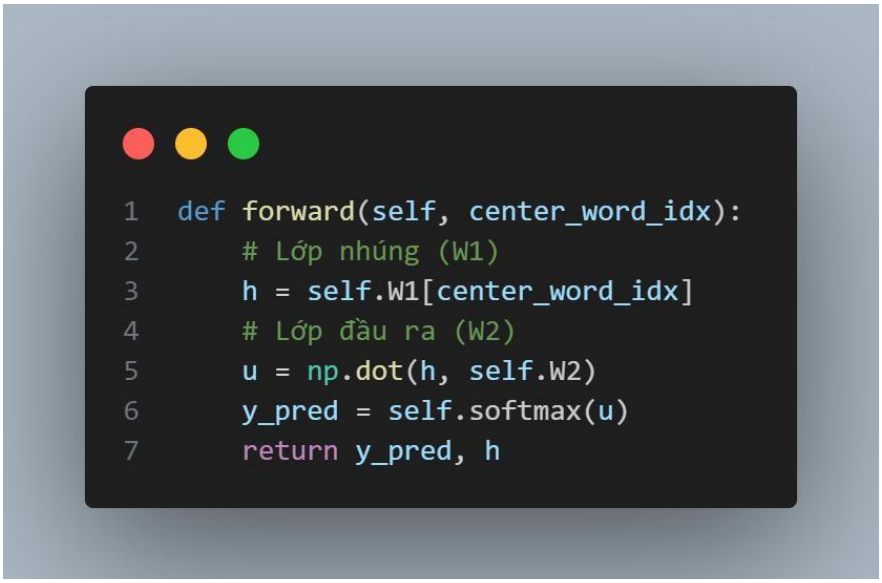
$$h = W_1[w_c]$$

- Tính toán đầu ra dự đoán y_{pred} bằng cách nhân vector nhúng h với ma trận trọng số đầu ra W_2:

$$u = h \times W_2$$

- Áp dụng hàm $softmax$ để chuyển đổi đầu ra uu thành xác suất:

$$y_{pred} = softmax(u)$$



Hình 12. Hàm forward

3. **Backward pass**:

- Tính toán lỗi giữa đầu ra dự đoán y_{pred} và nhãn thực tế y_{true}:

$$error = y_{pred} - y_{true}$$

```
error = y_pred
error[context_word_idx] -= 1  # Cập nhật lỗi cho từ ngữ cảnh
```

Hình 13. Tính toán lỗi giữa dự đoán và thực tế

- Tính toán gradient cho ma trận trọng số W_2 và W_1:

$$\frac{\partial Loss}{\partial W2} = h \times error$$

$$\frac{\partial Loss}{\partial W1} = error \times W^T_2$$

```
# Gradient cho W2 và W1
dW2 = np.outer(h, error)
dW1 = np.dot(error, self.W2.T)
```

Hình 14. Tính toán gradient cho ma trận trọng số

- Cập nhật trọng số W_2 và W_1 theo gradient descent:

$$W_2 \leftarrow W_2 - \eta \times \frac{\partial Loss}{\partial W_2}$$

$$W_1 \leftarrow W_1 - \eta \times \frac{\partial Loss}{\partial W_1}$$

```
# Cập nhật trọng số
self.W2 -= self.lr * dW2
self.W1[center_word_idx] -= self.lr * dW1
```

Hình 15. Cập nhật trọng số W1 và W2 theo gradient descent

4. **Lặp lại quá trình**:
   - Quá trình forward và backward được lặp lại qua từng batch dữ liệu cho đến khi hoàn thành số lượng epochs.

```
# Khởi tạo mô hình
embed_size = 100
skipgram_model = SkipGram(len(vocab), embed_size, learning_rate=0.01)

# Huấn luyện
losses = train_skipgram(skipgram_model, pairs, epochs=300, batch_size=128)
```

Hình 16. Huấn luyện mô hình

Chương 5: Đánh giá và phân tích embedding vector

5.1 Phương pháp đánh giá: Tính toán cosine similarity

Cosine similarity là một phương pháp được sử dụng để đo lường sự tương đồng giữa hai vector trong không gian vector. Công thức tính cosine similarity giữa hai vector u và v là:

$$cosine\_similarity(u, v) = \frac{u \cdot v}{\| u \| \cdot \| v \|}$$

Trong đó:

- $u \cdot v$ là tích vô hướng của hai vector.
- $\| u \|$ và $\| v \|$ là độ dài (chuẩn) của hai vector.

Ý nghĩa công thức:

- Cosine similarity nằm trong khoảng [−1,1][−1,1].
- Giá trị 1 cho thấy hai vector hoàn toàn tương đồng.
- Giá trị -1 cho thấy hai vector hoàn toàn đối lập.
- Giá trị 0 cho thấy hai vector không có mối liên hệ gì với nhau.

Trong code, hàm tính cosine similarity được triển khai như sau:

```python
# Tính toán cosine similarity giữa hai vector
def cosine_similarity(vec1, vec2):
    return np.dot(vec1, vec2) / (np.linalg.norm(vec1) * np.linalg.norm(vec2))
```

Hình 17. Hàm tính toán cosine similarity giữa hai vector

5.2 Đánh giá Embedding Vector với các từ đồng nghĩa và trái nghĩa

Để đánh giá chất lượng của các vector embedding, chúng ta sử dụng cosine similarity để so sánh các cặp từ đồng nghĩa và trái nghĩa. Các cặp từ được kiểm tra bao gồm:

- **Cặp từ đồng nghĩa**: "đầu_tiên" và "ban_đầu", "bố" và "cha".
- **Cặp từ trái nghĩa**: "lên" và "xuống", "lớn" và "nhỏ".

Cách hoạt động trong code:

- Với mỗi cặp từ, chúng ta lấy vector embedding tương ứng từ ma trận nhúng W1$W1$ của mô hình.
- Sử dụng hàm cosine_similarity để tính toán độ tương đồng giữa hai vector.
- In ra kết quả để đánh giá mối quan hệ ngữ nghĩa giữa các cặp từ.

```python
# Kiểm tra tương đồng cosine cho các cặp từ
word_pairs = [("lên", "xuống"), ("lớn", "nhỏ"), ("đầu_tiên", "ban_đầu"),("bố", "cha")]
for word1, word2 in word_pairs:
    idx1 = vocab[word1]
    idx2 = vocab[word2]
    vec1 = skipgram_model.W1[idx1]
    vec2 = skipgram_model.W1[idx2]
    similarity = cosine_similarity(vec1, vec2)
    print(f"Cosine similarity between '{word1}' and '{word2}': {similarity}")
```

Hình 18. Kiểm tra độ tương đồng giữa các cặp từ

Kết quả đánh giá:

- Cosine similarity giữa 'lên' và 'xuống': 0.31786933940767154
- Cosine similarity giữa 'lớn' và 'nhỏ': 0.2839457355948672
- Cosine similarity giữa 'đầu_tiên' and 'ban_đầu': 0.6744831777445114
- Cosine similarity giữa 'bố' và 'cha': 0.5994154669414063

Phân tích kết quả:

- **Cặp từ trái nghĩa**: Các cặp từ như "lên" và "xuống", "lớn" và "nhỏ" có độ tương đồng cosine thấp (0.317 và 0.283), phù hợp với kỳ vọng rằng các từ trái nghĩa sẽ có mối quan hệ ngữ nghĩa xa nhau.
- **Cặp từ đồng nghĩa**: Các cặp từ như "đầu_tiên" và "ban_đầu", "bố" và "cha" có độ tương đồng cosine cao hơn (0.674 và 0.599), cho thấy mô hình đã học được mối quan hệ ngữ nghĩa tương đồng giữa các từ này.

Kết luận:

- Mô hình đã học được các mối quan hệ ngữ nghĩa giữa các từ, với các cặp từ đồng nghĩa có độ tương đồng cao hơn so với các cặp từ trái nghĩa.
- Tuy nhiên, độ tương đồng giữa các cặp từ đồng nghĩa có thể được cải thiện hơn nữa để phản ánh chính xác hơn mối quan hệ ngữ nghĩa.

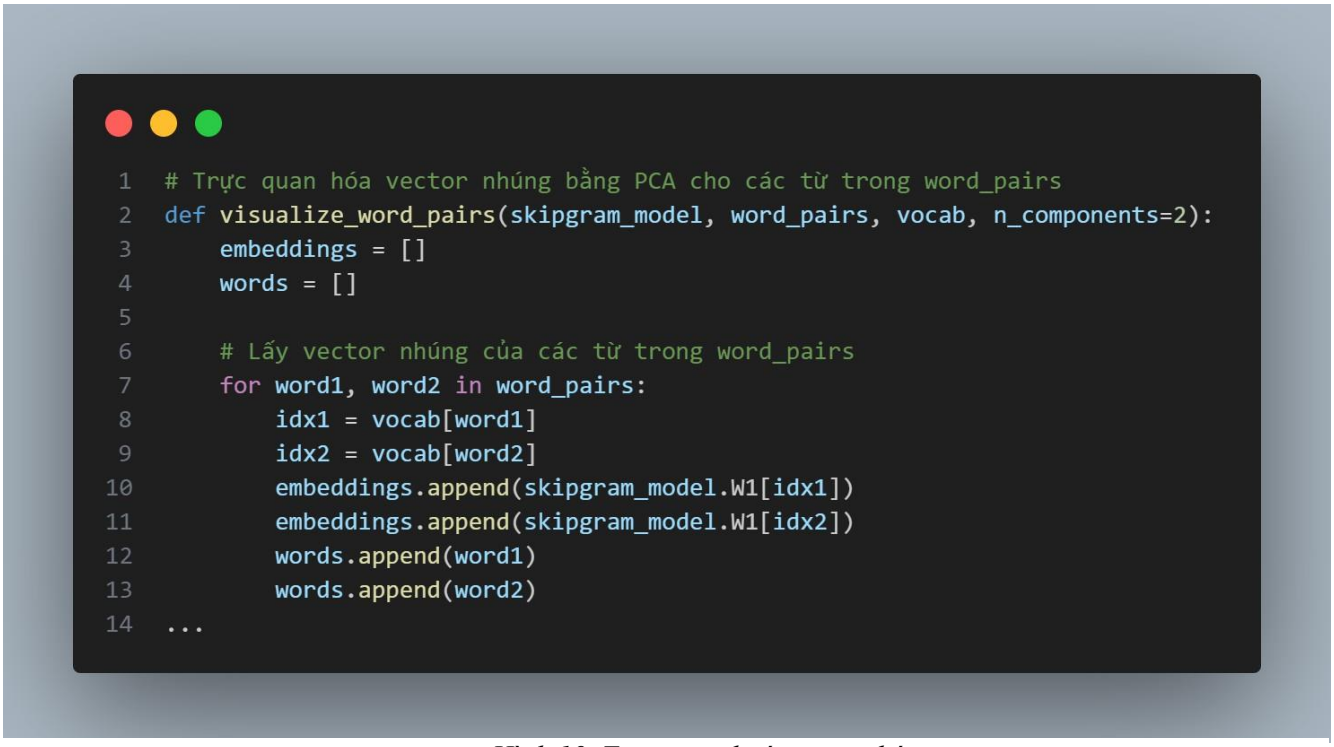
5.3 Trực quan hóa vector embedding

Để trực quan hóa các vector embedding, chúng ta sử dụng phương pháp **Phân tích Thành phần Chính (PCA)** để giảm chiều dữ liệu từ d chiều (kích thước vector nhúng) xuống 2 chiều. PCA giúp chúng ta biểu diễn các vector embedding trên một biểu đồ 2D, giúp dễ dàng quan sát mối quan hệ giữa các từ.

Cách đánh giá:

- Lấy các vector embedding của các từ cần trực quan hóa.
- Sử dụng PCA để giảm chiều dữ liệu xuống 2 chiều.
- Vẽ biểu đồ 2D để trực quan hóa các từ.

Triển khai trong code:



```python
# Trực quan hóa vector nhúng bằng PCA cho các từ trong word_pairs
def visualize_word_pairs(skipgram_model, word_pairs, vocab, n_components=2):
    embeddings = []
    words = []

    # Lấy vector nhúng của các từ trong word_pairs
    for word1, word2 in word_pairs:
        idx1 = vocab[word1]
        idx2 = vocab[word2]
        embeddings.append(skipgram_model.W1[idx1])
        embeddings.append(skipgram_model.W1[idx2])
        words.append(word1)
        words.append(word2)
...
```

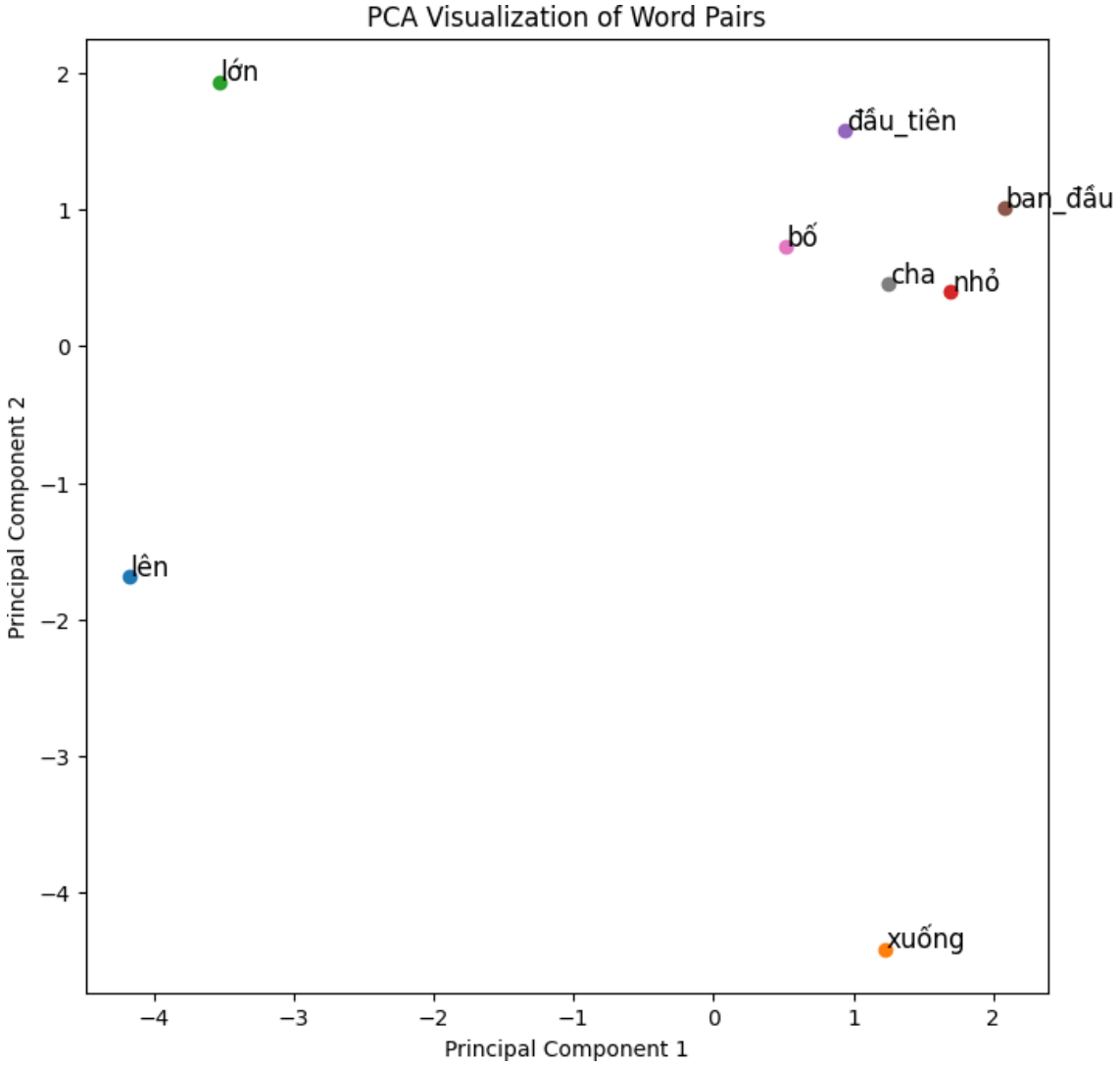
Hình 19. Trực quan hoá vector nhúng

Cách hoạt động:

- **Lấy vector embedding**: Lấy vector embedding của các từ cần trực quan hóa từ ma trận nhúng $W1$.
- **PCA**: Sử dụng PCA để giảm chiều dữ liệu từ d chiều xuống 2 chiều.
- **Vẽ biểu đồ**: Vẽ biểu đồ 2D với các điểm tương ứng với các từ.

Kết quả:

- Biểu đồ PCA cho thấy các từ đồng nghĩa nằm gần nhau, trong khi các từ trái nghĩa nằm xa nhau.
- Ví dụ: "bố" và "cha" nằm gần nhau, trong khi "lớn" và "nhỏ" nằm ở hai phía đối diện của đồ thị.



Hình 20. Biểu đồ trực quan hoá PCA

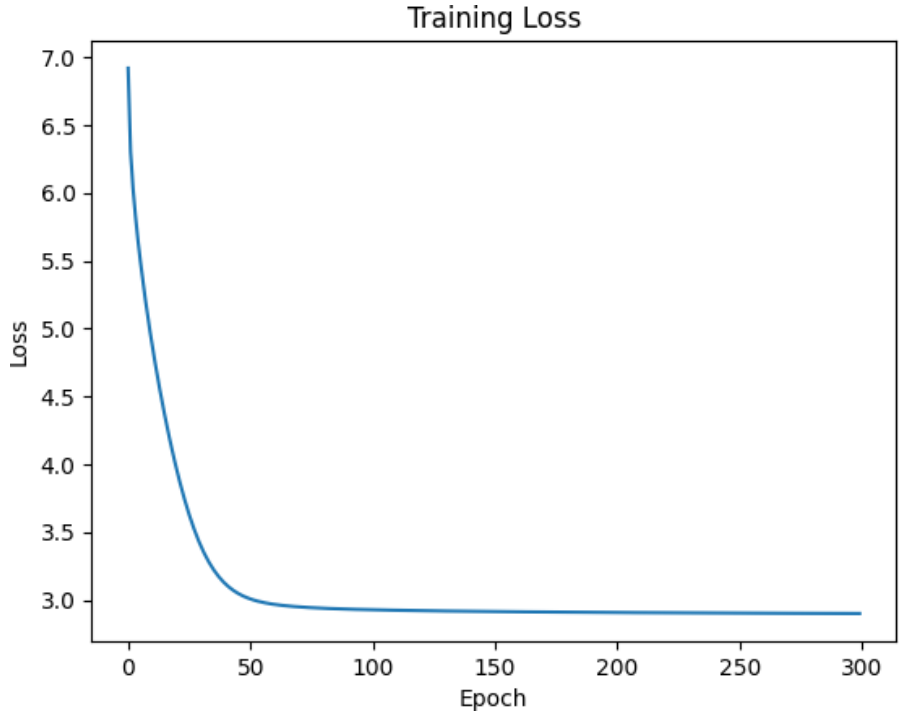
Kết luận:

- Biểu đồ PCA cho thấy mô hình đã học được các mối quan hệ ngữ nghĩa giữa các từ.
- Các từ đồng nghĩa nằm gần nhau, trong khi các từ trái nghĩa nằm xa nhau, phù hợp với kỳ vọng.

Chương 6: Phân tích kết quả huấn luyện

6.1 Biểu đồ mất mát theo thời gian

Biểu đồ Training Loss là một công cụ quan trọng để đánh giá hiệu suất của mô hình trong quá trình huấn luyện. Biểu đồ này thể hiện sự thay đổi của hàm mất mát (loss) theo số lượng epoch.



Hình 21. Biểu đồ mất mát theo thời gian

Biểu đồ cho thấy mô hình đã học tốt và hội tụ sau 300 epoch.

Tuy nhiên, mất mát không giảm đến 0, cho thấy mô hình có thể chưa đạt được độ chính xác tối ưu vẫn cần thiết cải tiến.

6.2 Phân tích kết quả huấn luyện

Dựa trên biểu đồ mất mát, chúng ta có thể rút ra một số nhận xét:

- **Hiệu suất huấn luyện**: Mô hình đã học được các mối quan hệ ngữ cảnh giữa các từ, nhưng mất mát vẫn còn cao, cho thấy mô hình có thể chưa đạt được độ chính xác tối ưu.
- **Sự hội tụ**: Mô hình đã hội tụ sau khoảng 300 epoch, nhưng mất mát không giảm đến 0, cho thấy có thể có nhiều yếu tố ảnh hưởng đến kết quả, chẳng hạn như kích thước tập dữ liệu, tham số huấn luyện, hoặc cấu trúc mô hình.

Kết quả mất mát qua các epoch:

Epoch 1, Loss: 6.918767927080753

Epoch 2, Loss: 6.302576513977667

Epoch 3, Loss: 6.033612249244593

Epoch 4, Loss: 5.829111951376145

Epoch 5, Loss: 5.654151890109314

Epoch 6, Loss: 5.497281712847372

Epoch 7, Loss: 5.353264139407696

Epoch 8, Loss: 5.218942178176883

Epoch 9, Loss: 5.092411750769834

Epoch 10, Loss: 4.972540670271273

...

Epoch 297, Loss: 2.900514312598685

Epoch 298, Loss: 2.9004627637915736

Epoch 299, Loss: 2.900411551076946

Epoch 300, Loss: 2.9003606714241683

Phân tích:

- **Ban đầu**: Mất mát giảm nhanh trong những epoch đầu tiên, cho thấy mô hình học được nhiều thông tin từ dữ liệu.
- **Sau đó**: Mất mát giảm chậm dần và tiến tới một giá trị ổn định, cho thấy mô hình đã bắt đầu hội tụ.
- **Cuối cùng**: Mất mát đạt đến một giá trị ổn định (khoảng 2,9) và không giảm nhiều nữa, cho thấy mô hình đã học được hầu hết các đặc trưng của dữ liệu.

Kết luận:

- Mô hình đã học được các mối quan hệ ngữ cảnh giữa các từ, nhưng mất mát vẫn còn cao, cho thấy mô hình có thể chưa đạt được độ chính xác tối ưu.

Chương 7: Khó khăn và bài học kinh nghiệm

7.1 Các thách thức trong việc triển khai mô hình

Trong quá trình triển khai mô hình Skip-gram, chúng ta đã gặp phải một số thách thức sau:

1. **Tiền xử lý dữ liệu phức tạp**:
    - Tiếng Việt là ngôn ngữ có nhiều đặc trưng riêng như dấu thanh, từ ghép, và cách tách từ phức tạp. Việc tiền xử lý dữ liệu đòi hỏi sử dụng các công cụ tách từ tiếng Việt như pyvi hoặc vncorenlp, nhưng các công cụ này có thể không hoàn toàn chính xác trong một số trường hợp.
    - **Khó khăn**: Đảm bảo dữ liệu được tiền xử lý chính xác để mô hình có thể học được các mối quan hệ ngữ cảnh giữa các từ.
2. **Khởi tạo và điều chỉnh tham số**:
    - Việc khởi tạo các ma trận trọng số W_1 và W_2 cần được thực hiện cẩn thận để tránh trường hợp mô hình không hội tụ hoặc hội tụ chậm.
    - **Khó khăn**: Lựa chọn giá trị khởi tạo phù hợp và điều chỉnh các tham số như learning rate, batch size, và số lượng epoch để đạt được kết quả tốt nhất.
3. **Tính toán gradient và lan truyền ngược**:
    - Việc tính toán gradient và cập nhật trọng số trong quá trình huấn luyện đòi hỏi sự chính xác và hiệu quả. Lỗi trong tính toán gradient có thể dẫn đến mô hình không hội tụ hoặc hội tụ về một giải pháp không tối ưu.
    - **Khó khăn**: Đảm bảo rằng các phép tính toán học (forward pass, backward pass) được triển khai chính xác.
4. **Đánh giá embedding vector**:
    - Đánh giá chất lượng của các vector embedding đòi hỏi sử dụng các phương pháp như cosine similarity và trực quan hóa PCA. Tuy nhiên, việc lựa chọn các cặp từ để đánh giá cũng như giải thích kết quả đòi hỏi sự hiểu biết sâu về ngữ nghĩa của các từ.
    - **Khó khăn**: Đảm bảo rằng các phương pháp đánh giá phản ánh chính xác chất lượng của vector embedding.

7.2 Bài học rút ra

Qua quá trình triển khai mô hình Skip-gram, chúng ta rút ra một số bài học kinh nghiệm quan trọng:

1. **Tiền xử lý dữ liệu là bước quan trọng**:
    - Tiền xử lý dữ liệu chính xác là điều kiện tiên quyết để mô hình học được các mối quan hệ ngữ cảnh giữa các từ. Việc sử dụng các công cụ tách từ tiếng Việt như pyvi hoặc vncorenlp là cần thiết, nhưng cần kiểm tra và điều chỉnh để đảm bảo chất lượng dữ liệu.
2. **Lựa chọn tham số phù hợp**:
    - Việc lựa chọn các tham số như learning rate, batch size, và số lượng epoch có ảnh hưởng lớn đến hiệu suất của mô hình. Thử nghiệm và điều chỉnh các tham số này là cần thiết để đạt được kết quả tốt nhất.
3. **Tính toán gradient chính xác**:
    - Việc tính toán gradient và cập nhật trọng số trong quá trình huấn luyện đòi hỏi sự chính xác. Lỗi trong tính toán gradient có thể dẫn đến mô hình không hội tụ hoặc hội tụ về một giải pháp không tối ưu.
4. **Đánh giá chất lượng embedding vector**:
    - Đánh giá chất lượng của các vector embedding bằng các phương pháp như cosine similarity và trực quan hóa PCA là cần thiết để đánh giá hiệu suất của mô hình. Việc lựa chọn các cặp từ để đánh giá cũng như giải thích kết quả đòi hỏi sự hiểu biết sâu về ngữ nghĩa của các từ.
5. **Cải thiện mô hình**:
    - Để cải thiện hiệu suất của mô hình, có thể thử nghiệm với các phương pháp khác như tăng kích thước tập dữ liệu, sử dụng các kỹ thuật nâng cao như negative sampling, hoặc thử nghiệm với các mô hình embedding khác như GloVe hoặc FastText.

Chương 8: Kết luận

8.1 Tóm tắt

Trong đề tài này, chúng ta đã triển khai và huấn luyện mô hình Skip-gram để học các vector embedding từ một tập dữ liệu văn bản tiếng Việt. Quá trình này bao gồm các bước chính sau:

1. **Tiền xử lý dữ liệu**:
    - Dữ liệu được tiền xử lý bằng cách chuyển đổi chữ thường, loại bỏ dấu câu và ký tự đặc biệt, tách từ bằng công cụ pyvi, và tạo từ điển từ vựng.
    - Các cặp từ trung tâm-ngữ cảnh được tạo ra để huấn luyện mô hình.
2. **Thiết kế mô hình Skip-gram**:
    - Mô hình Skip-gram bao gồm hai lớp chính: lớp nhúng (embedding layer) và lớp đầu ra (output layer).
    - Hàm mất mát được tính bằng cross-entropy, và quá trình huấn luyện sử dụng gradient descent để cập nhật trọng số.
3. **Huấn luyện mô hình**:
    - Mô hình được huấn luyện trong 300 epoch với các tham số như batch size, learning rate, và kích thước vector nhúng.
    - Kết quả huấn luyện cho thấy mất mát giảm dần và mô hình đã hội tụ sau 300 epoch.
4. **Đánh giá embedding vector**:
    - Chất lượng của vector embedding được đánh giá bằng cách tính cosine similarity giữa các cặp từ đồng nghĩa và trái nghĩa.
    - Kết quả cho thấy mô hình đã học được các mối quan hệ ngữ nghĩa giữa các từ, với các cặp từ đồng nghĩa có độ tương đồng cao hơn so với các cặp từ trái nghĩa.
5. **Trực quan hóa vector embedding**:
    - Các vector embedding được trực quan hóa bằng phương pháp PCA, cho thấy các từ đồng nghĩa nằm gần nhau, trong khi các từ trái nghĩa nằm xa nhau.

Kết luận chung:

- Mô hình Skip-gram đã học được các mối quan hệ ngữ nghĩa giữa các từ trong tập dữ liệu tiếng Việt.
- Tuy nhiên, mất mát vẫn còn cao, cho thấy mô hình có thể chưa đạt được độ chính xác tối ưu.

8.2 Đề xuất hướng phát triển trong tương lai

Để cải thiện hiệu suất của mô hình và mở rộng ứng dụng, chúng ta có thể xem xét các hướng phát triển sau:

1. **Tăng kích thước tập dữ liệu**:
    - Sử dụng tập dữ liệu lớn hơn và đa dạng hơn để mô hình học được nhiều mối quan hệ ngữ nghĩa hơn.
    - Tích hợp các tập dữ liệu từ nhiều nguồn khác nhau để tăng độ phong phú của dữ liệu.
2. **Sử dụng kỹ thuật negative sampling**:
    - Negative sampling là một kỹ thuật giúp tăng tốc độ huấn luyện và cải thiện chất lượng của vector embedding.
    - Áp dụng negative sampling để giảm số lượng từ cần dự đoán trong quá trình huấn luyện.
3. **Thử nghiệm với các mô hình embedding khác**:
    - Ngoài Skip-gram, có thể thử nghiệm với các mô hình embedding khác như GloVe hoặc FastText để so sánh hiệu suất.
    - Các mô hình này có thể cung cấp các vector embedding tốt hơn cho các từ trong tập dữ liệu tiếng Việt.
4. **Mở rộng ứng dụng:**
    - Sử dụng các vector embedding đã học được để xây dựng các ứng dụng thực tế như tìm kiếm thông tin, gợi ý từ đồng nghĩa, hoặc phân tích cảm xúc.
    - Tích hợp các vector embedding vào các mô hình học sâu như RNN, LSTM, hoặc Transformer để xử lý các tác vụ NLP phức tạp hơn.

Tài liệu tham khảo

1. Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). *Distributed Representations of Words and Phrases and their Compositionality*. Truy cập tại https://arxiv.org/abs/1310.4546
2. Nguyễn Trường Long. (n.d.). *Mô hình Skip-gram (Continuous Skip-gram)*. Truy cập tại https://nguyentruonglong.net/mo-hinh-skip-gram.html
3. Pyvi. (n.d.). *Vietnamese Natural Language Toolkit*. Truy cập tại https://pypi.org/project/pyvi/
4. Tiep. (n.d.). *Word2vec — Machine Learning cho dữ liệu dạng bảng*. Truy cập tại https://machinelearningcoban.com/tabml_book/ch_embedding/word2vec.html
5. Jurafsky, D., & Martin, J. H. (2020). *Speech and Language Processing*. Stanford University. Truy cập tại https://web.stanford.edu/~jurafsky/slp3/
6. Scikit-learn Documentation. (2023). *PCA - Principal Component Analysis*. Truy cập tại https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

Phụ lục A: Source code chương trình

```
import re
import numpy as np
from collections import Counter
from pyvi import ViTokenizer
from sklearn.decomposition import PCA

# Đọc và tiền xử lý dữ liệu
 def preprocess_text(text):
    # Chuyển chữ thường
    text = text.lower()
    # Loại bỏ ký tự số
    text = re.sub(r'\d+', '', text)
    # Loại bỏ các ký tự đặc biệt và dấu câu
    text = re.sub(r'[^\w\s]', '', text)
    # Tách từ tiếng Việt
    tokenized_text = ViTokenizer.tokenize(text)
    return tokenized_text.split()

# Tạo từ điển từ vựng
def build_vocab(corpus):
    word_counts = Counter(corpus)
    # Sắp xếp từ vựng theo thứ tự chữ cái
    sorted_vocab = sorted(word_counts.items())
    vocab = {word: idx for idx, (word, _) in enumerate(sorted_vocab)}
    idx_to_word = {idx: word for word, idx in vocab.items()}
    return vocab, idx_to_word

# Tạo cặp từ trung tâm-ngữ cảnh
def generate_skipgram_pairs(corpus, vocab, window_size=1):
    pairs = []
    for i, word in enumerate(corpus):
        center_word_idx = vocab[word]
        for j in range(-window_size, window_size + 1):
            if j == 0 or i + j < 0 or i + j >= len(corpus):
                continue
            context_word_idx = vocab[corpus[i + j]]
            pairs.append((center_word_idx, context_word_idx))
    return pairs

# Đọc và tiền xử lý dữ liệu từ file
with open('./data/data.txt', 'r', encoding='utf-8') as f:
    text = f.read()

corpus = preprocess_text(text)
vocab, idx_to_word = build_vocab(corpus)
pairs = generate_skipgram_pairs(corpus, vocab)\
```

```
# Lưu từ vựng vào file
with open('./data/vocab.txt', 'w', encoding='utf-8') as f:
    for word, idx in vocab.items():
        f.write(f"{word}\t{idx}\n")
print(vocab)

class SkipGram:
    def __init__(self, vocab_size, embed_size, learning_rate=0.01):
        self.vocab_size = vocab_size
        self.embed_size = embed_size
        self.lr = learning_rate

        # Khởi tạo trọng số
        self.W1 = np.random.rand(vocab_size, embed_size)
        self.W2 = np.random.rand(embed_size, vocab_size)

    def softmax(self, x):
        e_x = np.exp(x - np.max(x))
        return e_x / e_x.sum(axis=0)

    def forward(self, center_word_idx):
        # Lớp nhúng (W1)
        h = self.W1[center_word_idx]
        # Lớp đầu ra (W2)
        u = np.dot(h, self.W2)
        y_pred = self.softmax(u)
        return y_pred, h

    def backward(self, center_word_idx, context_word_idx, y_pred, h):
        error = y_pred
        error[context_word_idx] -= 1  # Cập nhật lỗi cho từ ngữ cảnh

        # Gradient cho W2 và W1
        dW2 = np.outer(h, error)
        dW1 = np.dot(error, self.W2.T)

        # Cập nhật trọng số
        self.W2 -= self.lr * dW2
        self.W1[center_word_idx] -= self.lr * dW1

def train_skipgram(model, pairs, epochs=100, batch_size=64):
    losses = []
    for epoch in range(epochs):
        total_loss = 0
        # Chia pairs thành các batch
        num_batches = len(pairs) // batch_size
        for batch_idx in range(num_batches):
```

```
            batch = pairs[batch_idx * batch_size : (batch_idx + 1) * batch_size]
            batch_loss = 0
            for center_word_idx, context_word_idx in batch:
                y_pred, h = model.forward(center_word_idx)
                #cross-entrophy
                loss = -np.log(y_pred[context_word_idx])
                batch_loss += loss
                model.backward(center_word_idx, context_word_idx, y_pred, h)
            total_loss += batch_loss / len(batch)  # Tính loss trung bình của batch
        losses.append(total_loss / num_batches)  # Tính loss trung bình của tất cả
các batch
        print(f"Epoch {epoch+1}, Loss: {losses[-1]}")
    return losses

# Khởi tạo mô hình
embed_size = 100
skipgram_model = SkipGram(len(vocab), embed_size, learning_rate=0.01)

# Huấn luyện
losses = train_skipgram(skipgram_model, pairs, epochs=50, batch_size=128)

# Vẽ đồ thị mất mát
import matplotlib.pyplot as plt
plt.plot(losses)
plt.title("Training Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.show()

# Tính toán cosine similarity giữa hai vector
def cosine_similarity(vec1, vec2):
    return np.dot(vec1, vec2) / (np.linalg.norm(vec1) * np.linalg.norm(vec2))

# Cặp đồng nghĩa: ("đầu_tiên", "ban_đầu"),("bố", "cha")
# Cặp trái nghĩa: ("lên", "xuống"), ("lớn", "nhỏ")

# Kiểm tra tương đồng cosine cho các cặp từ
word_pairs = [("lên", "xuống"), ("lớn", "nhỏ"), ("đầu_tiên", "ban_đầu"),("bố",
"cha")]
for word1, word2 in word_pairs:
    idx1 = vocab[word1]
    idx2 = vocab[word2]
    vec1 = skipgram_model.W1[idx1]
    vec2 = skipgram_model.W1[idx2]
    similarity = cosine_similarity(vec1, vec2)
    print(f"Cosine similarity between '{word1}' and '{word2}': {similarity}")

# Trực quan hóa vector nhúng bằng PCA cho các từ trong word_pairs
```

```
def visualize_word_pairs(skipgram_model, word_pairs, vocab, n_components=2):
    embeddings = []
    words = []

    # Lấy vector nhúng của các từ trong word_pairs
    for word1, word2 in word_pairs:
        idx1 = vocab[word1]
        idx2 = vocab[word2]
        embeddings.append(skipgram_model.W1[idx1])
        embeddings.append(skipgram_model.W1[idx2])
        words.append(word1)
        words.append(word2)

    # Áp dụng PCA để giảm chiều
    pca = PCA(n_components=n_components)
    reduced_embeddings = pca.fit_transform(np.array(embeddings))

    # Vẽ đồ thị trực quan hóa PCA
    plt.figure(figsize=(8, 8))
    for i, (x, y) in enumerate(reduced_embeddings):
        plt.scatter(x, y)
        plt.text(x + 0.01, y + 0.01, words[i], fontsize=12)

    plt.title("PCA Visualization of Word Pairs")
    plt.xlabel("Principal Component 1")
    plt.ylabel("Principal Component 2")
    plt.show()

# Gọi hàm trực quan hóa cho các cặp từ
visualize_word_pairs(skipgram_model, word_pairs, vocab)
```