TRƯỜNG ĐẠI HỌC SÀI GÒN
KHOA CÔNG NGHỆ THÔNG TIN

----- ๛ 🕮 ๛ -----

TIỂU LUẬN XỬ LÝ NGÔN NGỮ TỰ NHIÊN

Cài Đặt và Huấn Luyện Mô Hình Skip-Gram Trên Ngữ Liệu Tiếng Việt

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

Sinh viên thực hiện

Họ và tên	*MSSV*
Huỳnh Quang Quân	3122410341

NHẬN XÉT, ĐÁNH GIÁ CỦA GIẢNG VIÊN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

MỤC LỤC

MỞ ĐẦU

Skip-gram, một mô hình thuộc nhóm Word2Vec do Tomas Mikolov giới thiệu, đã chứng minh hiệu quả vượt trội trong việc học biểu diễn từ với độ chính xác cao và chi phí tính toán hợp lý. Mô hình này hoạt động dựa trên ý tưởng đơn giản nhưng mạnh mẽ: sử dụng một từ trung tâm để dự đoán các từ xung quanh, qua đó học được ngữ nghĩa ngầm của từ từ ngữ cảnh của nó.

Tiểu luận này tập trung vào phân tích chi tiết mô hình Skip-gram, từ ý tưởng lý thuyết, cách thức hoạt động đến ứng dụng thực tế. Minh họa cách mô hình học các vector từ, cũng như cách các vector này biểu diễn các mối quan hệ ngữ nghĩa và cú pháp trong ngôn ngữ. Bên cạnh đó, đánh giá hiệu quả của Skip-gram trên các bài toán thực tiễn và so sánh với các phương pháp word embedding.

Bằng cách tiếp cận chặt chẽ và khoa học, hy vọng tiểu luận này sẽ mang đến cái nhìn toàn diện và chi tiết về Skip-gram, góp phần làm sáng tỏ tầm quan trọng của mô hình này trong việc phát triển các ứng dụng xử lý ngôn ngữ tự nhiên hiện đại.

1. GIỚI THIỆU

1.1. Word2Vec

Word2vec là một mô hình đơn giản và nổi tiếng giúp tạo ra các biểu diễn embedding của từ trong một không gian có số chiều thấp hơn nhiều lần so với số từ trong từ điển. Ý tưởng của word2vec đã được sử dụng trong nhiều bài toán với dữ liệu khác xa với dữ liệu ngôn ngữ, được trình bày với nhiều ứng dụng để tạo một mô hình *product2vec* giúp tạo ra các embedding khác nhau cho thực phẩm và đồ gia dụng.

Ý tưởng cơ bản của word2vec có thể được gói gọn trong các ý sau:

- Hai từ xuất hiện trong những văn cảnh giống nhau thường có ý nghĩa gần với nhau.
- Ví dụ, với câu "Hà Nội là … của Việt Nam" thì từ trong dấu ba chấm khả năng cao là "thủ đô". Với câu hoàn chỉnh "Hà Nội là thủ đô của Việt Nam", mô hình word2vec sẽ xây dựng ra embeding của các từ sao cho xác suất để từ trong dấu ba chấm là "thủ đô" là cao nhất.

1.2. Word Embedding

Word embedding là một nhóm các kỹ thuật đặc biệt trong xử lý ngôn ngữ tự nhiên, có nhiệm vụ ánh xạ (chuyển đổi hoặc liên kết) một từ hoặc một cụm từ trong bộ từ vựng tới một vector số thực. Từ không gian một chiều cho mỗi từ tới không gian các vector liên tục. Các vector từ được biểu diễn theo phương pháp word embedding thể hiện được ngữ nghĩa của các từ, từ đó ta có thể nhận ra được mối quan hệ giữa các từ với nhau (tương đồng, trái nghịch,...).

- **Những ứng dụng phổ biến của Word Embedding gồm:**
  - Phân tích ngữ nghĩa văn bản.
  - Hệ thống gợi ý.
  - Mô hình dịch máy.

- **Phương pháp phổ biến hiện nay:**
  - **CBOW (Continuous Bag of Words)**: Dự đoán từ trung tâm từ các từ ngữ cảnh.
  - **Skip-Gram**: Dự đoán từ ngữ cảnh từ từ trung tâm.

1.3. Skip-Gram

Skip-Gram là một phương pháp tạo Word Embedding bằng cách học cách dự đoán các từ ngữ cảnh dựa trên một từ trung tâm, biểu diễn từ trong không gian vector.

- **Tổng quan sơ lược về cách hoạt động**:
  - Mô hình học để dự đoán xác suất xuất hiện các từ ngữ cảnh, cho trước từ trung tâm.
  - Dữ liệu được xử lý theo từng cặp (từ trung tâm, từ ngữ cảnh) trong một cửa sổ ngữ cảnh nhất định.

- **Lợi ích của Skip-Gram**:
  - Hiệu quả trên tập dữ liệu nhỏ hoặc tập dữ liệu ngôn ngữ hiếm.
  - Xử lý tốt các từ ít xuất hiện, vì mô hình học được các quan hệ ngữ cảnh đa chiều.

2. MỤC TIÊU BÀI TIỂU LUẬN

Xây dựng mô hình **Skip-Gram** dựa trên dữ liệu văn bản tiếng Việt.

Tiến hành các bước:

- Tiền xử lý dữ liệu văn bản.
- Thiết kế mô hình Skip-Gram, bao gồm lớp nhúng, lớp đầu ra và hàm mất mát.
- Huấn luyện mô hình Skip-Gram từ dữ liệu qua tiền xử lý cùng với các bài toán.
- Đánh giá kết quả thông qua các phương pháp trực quan hóa vector nhúng, ma trận tương đồng (cosine simularity matrix).

MÔ TẢ MÔ HÌNH

Lưu ý: Xử lý dữ liệu, xây dựng và huấn luyện mô hình được xây dựng ở file python "skipgram_model.py".

1. NGUYÊN LÍ HOẠT ĐỘNG CỦA SKIP-GRAM

- **Mục tiêu chính**:
  Cho trước một từ trung tâm w_t, Skip-Gram tìm cách tối đa hóa xác suất của các từ ngữ cảnh xung quanh $\{w_{t-k}, \cdots w_{t+k}\}$, trong đó k là kích thước cửa sổ ngữ cảnh.

- **Cách hoạt động**:

  1. Từ trung tâm (center word) được chuyển đổi thành một vector nhúng thông qua ma trận nhúng $W1$.
  2. Vector nhúng này được nhân với ma trận trọng số $W2$ để tạo ra đầu ra u, biểu diễn xác suất của tất cả các từ trong từ điển.
  3. Xác suất của các từ ngữ cảnh được tính thông qua **hàm softmax**.

- **Công thức tối ưu hóa (softmax)**:
  Với w_c là từ ngữ cảnh và w_t là từ trung tâm, xác suất $P(w_c, w_t)$ được tính bằng:

$$P(w_c|w_t) = \frac{exp(u_c)}{\sum exp(u_{w'})}, w'\epsilon V$$

trong đó:

- $u_c = W2a^T \cdot h$, với $h = W1 \cdot x$.
- u_c: giá trị logit cho từ ngữ cảnh w_c
- V là toàn bộ từ vựng.

Softmax đảm bảo rằng:

1. Xác suất của tất cả các từ w' trong từ vựng V cộng lại bằng 1.
2. Xác suất càng cao cho từ đúng w_c, mô hình càng chính xác.

- **Hàm mất mát (Loss function)**:
  Để tối ưu hóa mô hình, hàm mất mát **cross-entropy** được sử dụng, tính trung bình trên tất cả các cặp (center word, context word):

$$L = -\sum_{(w_t, w_c)} logP(w_t|w_c)$$

2. QUÁ TRÌNH XÂY DỰNG VÀ HUẤN LUYỆN MÔ HÌNH SKIP-GRAM

2.1. Cài đặt thư viện

- **numpy**: Xử lý mảng, tính toán ma trận.
- **matplotlib**: Trực quan hóa các thống kê, chỉ số kết quả.
- **pyvi**: Xử lý văn bản tiếng Việt.
- **pandas**: Phân tích và thao tác dữ liệu dạng bảng.
- **seaborn**: Thư viện mở rộng cho matplotlib, trực quan hóa biểu đồ đẹp mắt.
- **openpyxl**: Xử lý file excel.
- **json**: Lưu và đọc file json.

2.2. Chuẩn bị dữ liệu

- Tự chọn một đoạn hoặc bài mẫu là tập dữ liệu văn bản tiếng Việt, hoặc sử dụng một tập dữ liệu mẫu được cung cấp trong đề bài.

Dữ liệu: Số lượng từ vựng là 5114 => Dữ liệu lớn, được lấy từ các bài văn mẫu nghị luận xã hội.

- Dữ liệu sẽ được lưu ở file ***text.txt***.

2.3. Xây dựng các hàm xử lý thuật toán dữ liệu cho huấn luyện

a) Hàm xử lý văn bản

```python
# Hàm tiền xử lý văn bản
def preprocess_text(text, custom_phrases):
    text = text.lower()  # Chuyển thành chữ thường
    cleaned_text = ''.join(char for char in text if char.isalnum() or char.isspace())  # Loại bỏ ký tự đặc biệt
    words = ViTokenizer.tokenize(cleaned_text).split()  # Tách từ tiếng Việt
    # Nối cụm từ dựa trên danh sách custom_phrases
    for phrase in custom_phrases:
        phrase_words = phrase.split()  # Tách cụm từ thành danh sách các từ
        i = 0
        while i <= len(words) - len(phrase_words):
            if words[i:i + len(phrase_words)] == phrase_words:
                # Nối các từ trong cụm phrase thành một từ duy nhất
                words[i:i + len(phrase_words)] = ['_'.join(phrase_words)]
            else:
                i += 1
    return words
```

Chức năng: xử lý văn bản thô (raw text) để chuẩn bị dữ liệu đầu vào cho mô hình.

Nguyên lý hoạt động:

Chuyển thành chữ thường: `text = text.lower()`

- Mọi ký tự được chuyển về chữ thường để tránh phân biệt chữ hoa và chữ thường.

Loại bỏ ký tự đặc biệt:

```
cleaned_text = ''.join(char for char in text if char.isalnum() or
char.isspace())
```

- Ký tự đặc biệt như dấu chấm câu, dấu gạch ngang bị loại bỏ.
- Chỉ chấp nhận giữ lại các ký tự chữ cái, số, và khoảng trắng.

Tách từ bằng thư viện PyVi:

```
words = ViTokenizer.tokenize(cleaned_text).split()
```

- **ViTokenize:** Thư viện xử lý tiếng Việt, giúp tách các cụm từ ghép (VD: "trí tuệ nhân tạo" được tách thành "trí_tuệ", "nhân_tạo").
- split(): Chuyển văn bản đã tách từ thành danh sách các từ.

Xử lý từ vựng tùy chỉnh:

- Một số từ vựng thư viện PyVi sẽ không tách từ chính xác.

- Dựa vào từng từ vựng (phrase) trong từ vựng tùy chọn (custom_phrases), từ vựng (phrase) sẽ được tách ra thành cụm từ vựng (phrase_words) để xét điều kiện cho đúng:

```
for phrase in custom_phrases:
    phrase_words = phrase.split()
```

- Nếu gặp phải cụm từ vựng trong câu (words) ứng với của cụm từ vựng đang kiếm (phrase_words), lập tức nối cụm lại thành một từ vựng.

```
if words[i:i + len(phrase_words)] == phrase_words:
    words[i:i + len(phrase_words)] = ['_'.join(phrase_words)]
```

Kết quả trả lại: Danh sách các từ đã được xử lý, sẵn sàng để xây dựng từ điển hoặc huấn luyện.

b) Hàm loại bỏ từ vựng gây nhiễu

```
# Stopword
def remove_stopwords(text, stopwords):
    words = text.split()
    filtered_words = [word for word in words if word.lower() not in stopwords]
    return " ".join(filtered_words)
```

Chức năng: Hàm loại bỏ các từ dừng ra khỏi văn bản đầu vào và trả về văn bản đã được lọc.

Nguyên lý hoạt động:

Chia đoạn văn bản thành các từ: Hàm split() chia văn bản thành danh sách các từ dựa trên khoảng trắng `words = text.split()`

Lọc các từ không nằm trong stopwords:

```
filtered_words = [word for word in words if word.lower() not in stopwords]
```

- Duyệt qua từng từ trong danh sách words và giữ lại những từ không thuộc tập hợp stopwords.

- `word.lower()`: Chuyển từ về dạng chữ thường để so sánh, đảm bảo loại bỏ từ dừng bất kể viết hoa hay thường.

Kết quả trả về: Chuỗi văn bản mới, trong đó các từ dừng đã bị loại bỏ. `join()` sẽ ghép các từ trong danh sách `filtered_words` thành một chuỗi, sử dụng khoảng trắng làm ký tự phân cách.

c) Hàm xây dựng từ điển

```
# Hàm xây dựng từ điển từ vựng
def build_vocab(sentences):
    vocab = {}
    for sentence in sentences:
        for word in sentence:
            if word not in vocab:
                vocab[word] = len(vocab)
    return vocab
```

Chức năng: Tạo một từ điển ánh xạ (chuyển đổi) mỗi từ trong tập dữ liệu thành một số nguyên duy nhất (index).

Nguyên lý hoạt động:

Khởi tạo từ điển rỗng: `vocab = {}`

Duyệt qua từng câu và từ:

```
for sentence in sentences:
    for word in sentence:
```

- sentences: Danh sách các câu, mỗi câu là một danh sách các từ.

Thêm từ vào từ điển:

```
if word not in vocab:
    vocab[word] = len(vocab)
```

- Nếu từ chưa có trong từ điển, gán cho nó một **index** mới dựa trên kích thước hiện tại của từ điển. (Ví dụ: kích thước từ điển là 1, từ "nam" không có trong từ điển, từ điển "nam" sẽ được gán là 1 (kích thước hiện tại của từ điển), từ đó kích thước từ điển không còn là 1, mà là 2).

Kết quả trả lại: Từ điển dạng {word: index}, nơi mỗi từ tương ứng với một chỉ số duy nhất.

d) Hàm Softmax

```
# Hàm tính softmax
def softmax(x):
    e_x = np.exp(x - np.max(x))
    return e_x / e_x.sum(axis=0)
```

Chức năng: Chuyển đổi đầu ra tuyến tính (logits) thành xác suất.

Nguyên lý hoạt động:

Tính giá trị mũ của mỗi phần tử: `e_x = np.exp(x - np.max(x))`

- x - np.max(x): Để tránh số mũ lớn gây tràn số (overflow), trừ giá trị lớn nhất trong x khỏi mọi phần tử.

Chuẩn hóa giá trị mũ thành xác suất: `e_x / e_x.sum(axis=0)`

- Chia từng phần tử cho tổng toàn bộ, đảm bảo tổng các xác suất bằng 1.

Kết quả đầu ra: Danh sách xác suất, trong đó mỗi giá trị là xác suất dự đoán cho một lớp.

e) Hàm mất mát cross_entropy

```
# Hàm tính cross-entropy loss
def cross_entropy_loss(y_true, y_pred):
    return -np.sum(y_true * np.log(y_pred + 1e-9))
```

Chức năng: Tính toán độ mất mát giữa phân phối xác suất thực tế (y_true) và dự đoán (y_pred).

Nguyên lý hoạt động:

Công thức Cross-Entropy: `-np.sum(y_true * np.log(y_pred + 1e-9))`

- y_true: Vector one-hot đại diện cho nhãn thực.

- y_pred: Xác suất dự đoán từ hàm Softmax.
- +1e-9: Thêm một số nhỏ để tránh log(0), gây lỗi.

Ý nghĩa:

Hàm này đo lường sự khác biệt giữa 2 phân phối xác suất:

- y_true: Phân phối mục tiêu (đúng).
- y_pred: Phân phối do mô hình dự đoán.

Giá trị nhỏ hơn thể hiện dự đoán gần đúng hơn với thực tế.

f) Hàm tạo dữ liệu huấn luyện cặp từ trung tâm và ngữ cảnh

```python
# Hàm tạo cặp từ trung tâm và ngữ cảnh
def generate_training_data(sentences, vocab, window_size):
    data = []
    for sentence in sentences:
        sentence_len = len(sentence)
        for idx, word in enumerate(sentence):
            word_idx = vocab[word]
            context_indices = list(range(max(0, idx - window_size), min(sentence_len, idx + window_size + 1)))
            context_indices.remove(idx)
            for context_idx in context_indices:
                context_word = sentence[context_idx]
                context_word_idx = vocab[context_word]
                data.append((word_idx, context_word_idx))
    return data
```

Chức năng: Tạo ra các cặp từ trung tâm và từ ngữ cảnh dựa trên cửa sổ ngữ cảnh xung quanh một từ trong câu.

Nguyên lí hoạt động:

Khởi tạo danh sách lưu dữ liệu: data = []

Lặp qua từng câu và xác định chiều dài câu

Lặp qua từng từ trong câu: `for idx, word in enumerate(sentence):`

Lấy chỉ số từ trung tâm: Chuyển từ trung tâm sang chỉ số tương ứng trong từ điển vocab `word_idx = vocab[word].`

Xác định các chỉ số ngữ cảnh:

```
context_indices = list(range(max(0, idx - window_size),
                  min(sentence_len, idx + window_size + 1)))
context_indices.remove(idx)
```

- `max(0, idx - window_size)`: Đảm bảo cửa sổ không vượt quá đầu câu.
- `min(sentence_len, idx + window_size + 1)`: Đảm bảo cửa sổ không vượt quá cuối câu.
- `remove(idx)`: Loại bỏ vị trí của từ trung tâm ra khỏi danh sách ngữ cảnh.

Tạo cặp từ trung tâm và ngữ cảnh:

```
for context_idx in context_indices:
    context_word = sentence[context_idx]
    context_word_idx = vocab[context_word]
    data.append((word_idx, context_word_idx))
```

Với mỗi chỉ số ngữ cảnh trong context_indices:

1. Lấy từ tương ứng tại chỉ số đó (context_word).
2. Chuyển từ ngữ cảnh sang chỉ số (context_word_idx).
3. Thêm cặp (word_idx, context_word_idx) vào danh sách data.

Kết quả trả về: Một danh sách data chứa các các cặp từ (từ trung tâm, từ ngữ cảnh)

g) Hàm huấn luyện Skip-Gram

```python
# Hàm huấn luyện Skip-gram
def train_skipgram(sentences, vocab, epochs, learning_rate):
    window_size = 8
    vocab_size = len(vocab)
    embedding_dim = 10
    batch_size = 64
    print(f'Batch_size: {batch_size}\nEmbedding_dim: {embedding_dim}\n')

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

    # Chuẩn bị dữ liệu huấn luyện
    training_data = generate_training_data(sentences, vocab, window_size)

    # Theo dõi mất mát qua các epoch
    loss_history = []

    for epoch in range(epochs):
        total_loss = 0

        # Chia dữ liệu thành các batch
        np.random.shuffle(training_data)
        batches = [training_data[i:i + batch_size] for i in range(0, len(training_data), batch_size)]

        for batch in batches:
            # Khởi tạo gradient tổng cho mỗi batch
            grad_W1 = np.zeros_like(W1)
            grad_W2 = np.zeros_like(W2)

            for word_idx, context_word_idx in batch:
                # Forward pass
                h = W1[word_idx]  # Nhúng từ trung tâm
                u = np.dot(W2.T, h)  # Ma trận đầu ra
                y_pred = softmax(u)

                # Tạo vector nhãn one-hot
                y_true = np.zeros(vocab_size)
                y_true[context_word_idx] = 1

                # Tính loss
                loss = cross_entropy_loss(y_true, y_pred)
                total_loss += loss

                # Backpropagation
                e = y_pred - y_true
                grad_W2 += np.outer(h, e)
                grad_W1[word_idx] += np.dot(W2, e)

            # Cập nhật trọng số sau mỗi batch
            W2 -= learning_rate * grad_W2
            W1 -= learning_rate * grad_W1

        # Ghi nhận mất mát
        loss_history.append(total_loss)
        print(f"Epoch {epoch + 1}/{epochs}, Loss: {total_loss}")
    print("Huấn luyện hoàn tất!\n")
    return W1, W2, loss_history
```

Chức năng: Hàm huấn luyện mô hình Skip-gram bằng cách tối ưu hóa biểu diễn vector cho các từ trong một tập từ vựng

Nguyên lý hoạt động:

1) **Khởi tạo**:

   Các tham số:

   - **epochs**: Số lần lặp qua toàn bộ tập dữ liệu để huấn luyện.
   - **learning_rate**: Tốc độ học dùng để cập nhật trọng số.
   - **vocab_size**: Tổng số từ trong từ vựng.
   - **embedding_dim**: Kích thước vector nhúng mà mô hình sẽ học cho mỗi từ.
   - **batch_size**: Số lượng cặp từ (từ trung tâm, từ ngữ cảnh) được xử lý trong một lượt cập nhật.
   - **window_size**: Số từ xung quanh từ trung tâm được coi là ngữ cảnh

   Trọng số (W1**,** W2**)**:

   - d là số chiều nhúng (embedding_dim) và $|V|$ là kích thước từ vựng (vocab_size)

   - W1: Ma trận nhúng (kích thước $|V| \times d$).
   - W2: Ma trận ánh xạ (chuyển đổi) từ không gian nhúng trở lại từ vựng (kích thước $d \times |V|$)

2) **Chuẩn bị dữ liệu huấn luyện:**

```
training_data = generate_training_data(sentences, vocab, window_size)
```

- Tạo danh sách các cặp dùng làm dữ liệu huấn luyện.

3) **Tạo biến danh sách mất mát (loss):** `loss_history = []`

4) **Vòng lặp huấn luyện qua các epoch và xử lý từng batch**

5) **Forward pass**:

Vector nhúng h: `h = W1[word_idx]`

- Truy xuất vector nhúng của từ trung tâm.

Logits u: `u = np.dot(W2.T, h)`

- Dự đoán logits bằng cách nhân h với W2.

Softmax y_{pred} : `y_pred = softmax(u)`

6) **Loss và Backpropagation (Lan truyền ngược)**:

Tính loss:

```
y_true = np.zeros(vocab_size)
y_true[context_word_idx] = 1
loss = cross_entropy_loss(y_true, y_pred)
```

- Loss đo lường sự khác biệt giữa dự đoán và nhãn thực tế.
- Cập nhật mất mát: `total_loss += loss`

Gradient ngược (Backward Pass):

```
e = y_pred - y_true
grad_W2 += np.outer(h, e)
grad_W1[word_idx] += np.dot(W2, e)
```

- **Gradient** e: Sai số giữa dự đoán và thực tế.
- Cập nhật gradient cho cả hai ma trận trọng số.

7) **Cập nhật trọng số:**

```
W2 -= learning_rate * grad_W2
W1 -= learning_rate * grad_W1
```

- Trọng số được cập nhật bằng phương pháp gradient descent.

8) **Theo dõi mất mát**: `loss_history.append(total_loss)`

Ghi lại tổng mất mát cho mỗi epoch.

Kết quả trả lại:

- *W1*: Ma trận vector nhúng của từ.
- *W2*: Ma trận ánh xạ sang từ vựng.
- *loss_history*: Danh sách mất mát qua từng epoch.

h) Hàm lưu mô hình huấn luyện

```python
# Hàm lưu mô hình huấn luyện
def save(W1, W2, vocab):
    # Lưu các ma trận trọng số W1, W2
    np.save( file: "W1.npy", W1)
    np.save( file: "W2.npy", W2)

    # Lưu từ điển vocab dưới dạng JSON
    with open("vocab.json", "w", encoding="utf-8") as f:
        json.dump(vocab, f, ensure_ascii=False)
```

Chức năng: Lưu các lớp ma trận đã được huấn luyện và từ điển.

Nguyên lí hoạt động:

- `np.save`, Lưu ma trận nhúng và ma trận đầu ra với
- `json.dump`, lưu từ điển từ dữ liệu qua tiền xử lý, không biến đổi kí tự Unicode, tiếng Việt, thành dạng mã hóa ASCII bằng cách vô hiệu hóa yếu tố `ensure_ascii=False`.

3. THỰC HIỆN TIỀN XỬ LÝ DỮ LIỆU VÀ HUẤN LUYỆN MÔ HÌNH SKIP-GRAM

```python
#-----------------THỰC HIỆN-----------------#
stopwords = [
    "là", "và", "của", "có", "trong", "với", "được", "cho", "như", "cũng", "khi", "ở", "này", "ra", "lại", "về", "đã",
    "những", "rằng", "đó", "nên", "còn", "đi", "đây", "nhiều", "mà", "thì", "hoặc", "hay", "gì", "ai", "sao", "thế", "tại", "nào", "chỉ",
    "cái", "vì", "đâu", "vẫn", "đều", "nếu", "như", "vậy", "này", "ấy", "trên", "dưới", "từ", "tới", "hơn", "cả", "ít", "nhiều", "nữa",
    "nào", "vừa", "mỗi", "một", "mọi", "không", "rất"
]

# Từ vựng tùy chỉnh
custom = open("custom_dict.txt","r",encoding="utf-8")
custom_read = custom.read()
custom.close()
custom_phrases = []
for phrase in custom_read.splitlines():
    custom_phrases.append(phrase)

# Dữ liệu mẫu
file = open("train.txt", "r", encoding='utf-8')
texts = file.read()
```

```
# Tiền xử lý dữ liệu
# Stopword cho dữ liệu
texts = remove_stopwords(texts,stopwords)

# Tách chuỗi dựa trên nhiều dấu ngắt câu
separators = [".", "?", "!"]
for separator in separators:
    texts = texts.replace(separator, "|")  # Thay thế các ký tự phân cách bằng "|"
processed_text = [preprocess_text(text,custom_phrases) for text in texts.split("|")]
vocab = build_vocab(processed_text)
print(f'Số lượng từ: {len(vocab)}')

# Huấn luyện mô hình
W1, W2, loss_history = train_skipgram(processed_text, vocab, epochs=10, learning_rate=0.01)

# Lưu mô hình
save(W1, W2, vocab)
```

- **Đọc dữ liệu mẫu từ file text.txt:** Vì là ngôn ngữ tiếng Việt, nên khi mở file phải có encoding=’utf-8’, đây là giờ Việt Nam, thì khi xử lý sẽ nhận diện được ngôn ngữ Việt.
- **Đọc từ vựng tùy chỉnh từ file custom_dict.txt:** Đôi lúc sẽ có các từ vựng tiếng Việt thư viện Pyvi không thể tách hoặc xử lý được, sử dụng từ vựng tùy chỉnh để hỗ trợ tách từ:
  ”trí tuệ”, “nâng cao”, “nguyễn minh châu”, “nguyên khoa điềm”, “trí_tuệ nhân_tạo”, “hồ chí minh”, “góc phố”, “con đường”, “việt nam”
- **Loại bỏ từ vựng thừa:** `texts = remove_stopwords(texts,stopwords)`
- **Tiền xử lý dữ liệu:**

```
separators = [".", "?", "!"]
for separator in separators:
    texts = texts.replace(separator, "|")
processed_text = [preprocess_text(text,custom_phrases) for text in texts.split("|")]
vocab = build_vocab(processed_text)
print(f'Số lượng từ: {len(vocab)}')
```

Dữ liệu trước khi vào việc xử lý, sẽ được chia câu ra. Không quan trọng dữ liệu có bao nhiêu dòng, chỉ cần là câu được kết thúc bằng “?”, “!”, “.” thì sẽ được tách ra lần lượt để được xử lý.

✓ Từ tiếng Việt sau khi được xử lý (processed_texts)

Ví dụ:

```
[['hà_nội', 'là', 'trái_tim', 'của', 'việt_nam'], ['hà_nội', 'là', 'nơi',
'lưu_giữ', 'những', 'giá_trị', 'văn_hóa', 'lịch_sử', 'lâu_đời'], ['đi', 'dọc',
'36', 'phố_phường', 'du_khách', 'sẽ', 'cảm_nhận', 'được', 'sự', 'kết_hợp',
'hài_hòa', 'giữa', 'nét', 'cổ_kính', 'của', 'phố', 'và', 'vẻ', 'hiện_đại',
```

```
'của', 'đô_thị', 'đang', 'phát_triển'], ['mỗi', 'góc_phố', 'mỗi', 'con_đường',
'mang', 'trong', 'mình', 'một', 'câu_chuyện', 'và', 'dấu_ấn', 'thời_gian'],
[]…]
```

✓ Sau đó, mỗi từ sẽ được cho vào danh sách từ vựng (vocab) ứng với mỗi **index** bắt đầu từ 0:

Ví dụ:

```
{'hà_nội': 0, 'là': 1, 'trái_tim': 2, 'của': 3, 'việt_nam': 4, 'nơi': 5,
'lưu_giữ': 6, 'những': 7, 'giá_trị': 8, 'văn_hóa': 9, 'lịch_sử': 10,
'lâu_đời': 11, 'đi': 12, 'dọc': 13, '36': 14, 'phố_phường': 15, 'du_khách':
16, 'sẽ': 17, 'cảm_nhận': 18, 'được': 19, 'sự': 20, 'kết_hợp': 21, 'hài_hòa':
22, 'giữa': 23, 'nét': 24, 'cổ_kính': 25, 'phố': 26, 'và': 27, 'vẻ': 28,
'hiện_đại': 29, 'đô_thị': 30, 'đang': 31, 'phát_triển': 32, 'mỗi': 33,
'góc_phố': 34, 'con_đường': 35, 'mang': 36, 'trong': 37, 'mình': 38, 'một':
39, 'câu_chuyện': 40, 'dấu_ấn': 41, 'thời_gian': 42,… }
```

➢ **Huấn luyện mô hình Skip-Gram:**

- Gọi hàm, sử dụng processed_texts (dữ liệu đã được xử lý), vocab (từ vựng), sử dụng tham số epochs=10 (số lần lặp), learning_rate=0.01 (tốc độ học).
- Khởi tạo các tham số khác và trọng số, window_size=8, embedding_dim=10, batch_size=64, W1, W2.
- Thực hiện 10 vòng lặp qua toàn bộ dữ liệu, tham số đầu vào để tối ưu hóa trọng số.
- Mỗi câu trong sentences được chia nhỏ thành các từ, mỗi từ được xem là "từ trung tâm" (center word).
- Với mỗi từ trung tâm, xác định các từ ngữ cảnh xung quanh nó trong phạm vi window_size.
- Thực hiện tuyến tính (Forward Pass), áp dụng softmax để tính xác suất.

- Tính mất mát (loss) cross_entropy giữa true và predict.
- Backpropagation (Lan truyền ngược) để tính lỗi, cập nhật ma trận theo gradient descent.
- Ghi nhận mất mát total_loss và lưu lại lịch sử mất mát, ma trận nhúng W1 và ma trận ánh xạ W2 sẽ được cập nhật.

➢ **Lưu mô hình:** `save(W1, W2, vocab)`

❖ **Lí do chọn những tham số đó:**

➢ epoch=10, số lần lặp tương đối mà không khiến cho quá trình huấn luyện bị overfitting.

➢ learning_rate=0.01, sử dụng tốc độ học mặc định thường được sử dụng nhất.

➢ window_size=8, mở rộng phạm vi học để tăng độ đa dạng và mở rộng khả năng nhận diện ngữ cảnh, điều này tăng độ phức tạp trong huấn luyện.

➢ embedding_dim=10, để giảm phạm vi chiều nhúng cho mô hình chạy nhanh hơn đồng thời không bị overfitting vì phạm vi học ngữ cảnh.

➢ batch_size=64, tối ưu hóa cân bằng giữa tốc độ học và cập nhật ma trận trọng số mỗi lần lặp.

KẾT QUẢ

Lưu ý: Kết quả mô hình được xây dựng ở file python "skipgram_model.py"

1. XÂY DỰNG HÀM TRỰC QUAN HÓA ĐỘ MẤT MÁT

```python
# Hàm hiển thị mất mát theo thời gian
def plot_loss(loss_history):
    plt.plot( *args: range(1, len(loss_history) + 1), loss_history)
    plt.xlabel("Epoch")
    plt.ylabel("Loss")
    plt.title("Độ mất mát qua số Epochs")
    plt.savefig("Loss Over Epochs.jpg")
    plt.close()
```

Chức năng: Hàm này nhận vào danh sách loss_history (lịch sử mất mát qua các epoch), vẽ biểu đồ thể hiện độ mất mát theo số epoch và lưu biểu đồ dưới dạng tệp ảnh.

Các bước thực hiện:

Vẽ biểu đồ:

- Sử dụng plt.plot() để tạo biểu đồ tuyến tính.
- Số epoch `(range(1, len(loss_history) + 1))` được làm trục x.
- Giá trị mất mát (loss_history) được làm trục y.

Tùy chỉnh biểu đồ:

- Gán nhãn cho trục x (Epoch), trục y (Loss), và tiêu đề biểu đồ (Độ mất mát qua số Epochs).

Lưu kết quả:

- Dùng plt.savefig() để lưu biểu đồ vào tệp "Loss Over Epochs.jpg".
- Đóng biểu đồ bằng plt.close() để giải phóng bộ nhớ.

2. ĐỘ MẤT MÁT (LOSS)

Trong quá trình huấn luyện sẽ xuất hiện những dòng hiển thị kết quả mất mát qua số Epoch ?/10, giúp theo dõi độ hiệu quả và quá trình huấn luyện của mô hình.

Kết quả mất mát theo số Epoch:

Epoch 1/10, Loss: 1606690.6377885821

Epoch 2/10, Loss: 1559358.0680156685

Epoch 3/10, Loss: 1545140.2812950083

Epoch 4/10, Loss: 1531475.0417595974

Epoch 5/10, Loss: 1517733.2094597458

Epoch 6/10, Loss: 1504414.5508787364

Epoch 7/10, Loss: 1491902.5301976223

Epoch 8/10, Loss: 1480197.237904006

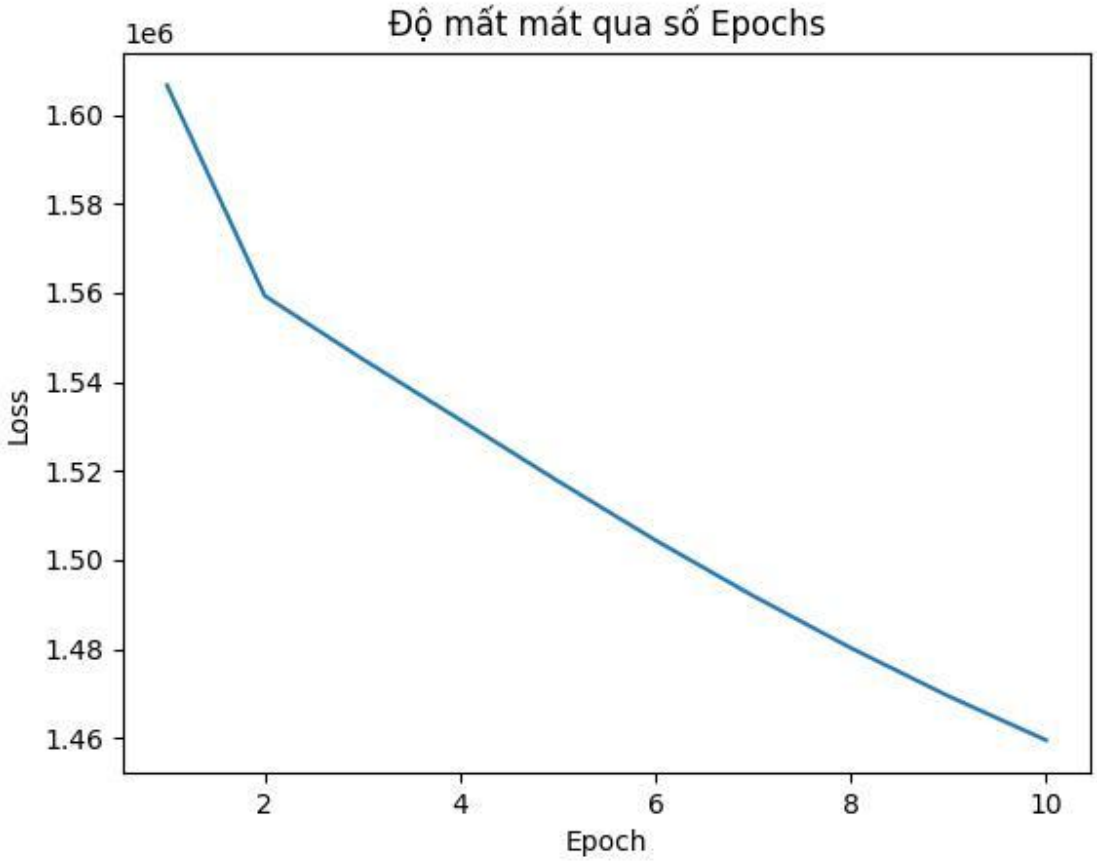
Epoch 9/10, Loss: 1469367.6202339844

Epoch 10/10, Loss: 1459448.6768753936

Kết quả mất mát qua số lần lặp (Loss Over Epochs)

Trực quan hóa mất mát theo số Epoch:

✧ Gọi hàm: `plot_loss(loss_history)`



Biểu đồ mất mát đã được vẽ và lưu dưới tên **"Loss Over Epochs.jpg"**, hiển thị sự giảm dần của độ mất mát qua các epoch.

Nhận xét và giải thích về sự thay đổi mất mát

Dạng tổng quát của đồ thị:

- Đồ thị cho thấy **độ mất mát (Loss)** giảm dần qua số lượng Epochs, từ khoảng **1.6 triệu** ở Epoch 1 xuống còn **1.46 triệu** ở Epoch 10.
- Đây là xu hướng mong muốn trong quá trình huấn luyện mô hình, vì độ mất mát giảm chứng tỏ rằng mô hình đang dần học được cách dự đoán tốt hơn trên dữ liệu huấn luyện.

Giảm mạnh ban đầu: Ở các epoch đầu tiên, trọng số (weights) được khởi tạo ngẫu nhiên, do đó mô hình chưa tối ưu và mất mát cao. Quá trình backpropagation điều chỉnh nhanh chóng, làm giảm đáng kể độ mất mát.

Giảm chậm dần:

- Sau một số epoch, mô hình đã học được nhiều đặc trưng quan trọng từ dữ liệu, dẫn đến việc giảm mất mát chậm lại. Đây là giai đoạn hội tụ dần của mô hình.
- Khi huấn luyện tiếp tục, các trọng số của mô hình tiến gần hơn đến giá trị tối ưu, khiến việc giảm mất mát trở nên chậm lại. Điều này là do gradient trở nên nhỏ dần khi tiến gần đến cực tiểu của hàm mất mát.

Hội tụ ổn định: Ở các epoch cuối, mất mát thay đổi rất ít, cho thấy mô hình đã đạt trạng thái tối ưu hoặc gần tối ưu. Đồ thị không xuất hiện dấu hiệu độ mất mát tăng trở lại, cho thấy mô hình không bị overfitting.

ĐÁNH GIÁ

Lưu ý: Đánh giá kết quả mô hình được xây dựng ở file python "evaluate_similarity.py".

1. XÂY DỰNG HÀM TRỰC QUAN HÓA ĐỘ TƯƠNG ĐỒNG COSIN

a) Hàm tải về dữ liệu của mô hình

```
# Hàm tải lại các trọng số và từ điển
def load():
    W1 = np.load("W1.npy")
    W2 = np.load("W2.npy")
    with open("vocab.json", "r", encoding="utf-8") as f:
        vocab = json.load(f)
    return W1, W2, vocab
```

W1: Ma trận nhúng từ đầu tiên, chứa vector nhúng của từ trung tâm (center word).

W2: Ma trận nhúng từ ngữ cảnh (context word), không được sử dụng trong đoạn mã này.

vocab: Từ điển ánh xạ mỗi từ sang một **chỉ số** (index) trong ma trận nhúng W1.

b) Hàm tính cosine similarity giữa cặp từ

```python
# Hàm tính cosine similarity giữa các vector nhúng
def compute_cosine_similarity(word1, word2, vocab, W1):
    if word1 not in vocab or word2 not in vocab:
        print(f"Không tìm thấy một trong các từ '{word1}' hoặc '{word2}' trong từ điển!")
        return None

    idx1, idx2 = vocab[word1], vocab[word2]
    vec1, vec2 = W1[idx1], W1[idx2]

    # Tính cosine similarity bằng numpy
    dot_product = np.dot(vec1, vec2)
    norm1 = np.linalg.norm(vec1)
    norm2 = np.linalg.norm(vec2)
    similarity = dot_product / (norm1 * norm2)
    return similarity
```

Chức năng: Tính **cosine similarity** giữa hai từ dựa trên vector nhúng của chúng.

Các bước thực hiện:

Kiểm tra sự tồn tại của từ: Nếu một trong hai từ không có trong từ điển (vocab), hàm trả về None và in thông báo lỗi.

Truy cập chỉ số vector:

- Lấy chỉ số của từ từ từ điển vocab.
- Sử dụng chỉ số đó để lấy vector nhúng từ ma trận W1.

Tính toán cosine similarity:

- Tính tích vô hướng giữa hai vector bằng np.dot.
- Tính chuẩn (norm) của từng vector bằng np.linalg.norm.
- Tính cosine similarity theo công thức:

$$similarity = \frac{dotproduct}{norm1 \times norm2}$$

Kết quả trả về: Giá trị cosine similarity giữa hai vector.

c) Hàm trực quan hóa cosine similarity

```python
# Hàm tạo heatmap cho các từ
def plot_similarity_heatmap(word1, word2, vocab, W1):
    # Tạo ma trận tương đồng
    similarity_matrix = np.zeros((len(word1), len(word2)))

    for i, feature1 in enumerate(word1):
        for j, feature2 in enumerate(word2):
            try:
                similarity = compute_cosine_similarity(feature1, feature2, vocab, W1)
                similarity_matrix[i, j] = similarity if similarity is not None else 0
            except:
                similarity_matrix[i, j] = 0

    # Chuyển ma trận thành DataFrame để trực quan hóa
    df_similarity = pd.DataFrame(similarity_matrix, index=word1, columns=word2)

    # Vẽ heatmap
    plt.figure(figsize=(12,8))
    sns.heatmap(df_similarity,annot=True, cmap="coolwarm", fmt=".2f", cbar=True)
    sns.set(font_scale=1)
    plt.title("Cosine Similarity Heatmap")
    plt.xlabel("Từ 2")
    plt.ylabel("Từ 1")
    plt.tight_layout()
    plt.savefig("Cosine Similarity Heatmap.png")
    print("Đã lưu heatmap cosine similarity")
    plt.close()
```

Chức năng: Lưu biểu đồ trực quan hóa **heatmap** của ma trận cosine similarity.

Các bước hoạt động:

Tạo ma trận trống: `similarity_matrix = np.zeros((len(word1), len(word2)))` kích thước mảng hàng (độ dài danh sách 1) và cột (độ dài danh sách 2).

Tính độ tương đồng:

- Duyệt qua từng cặp từ trong word1 và word2.
- Tính độ tương đồng cosine và điền vào ma trận.

Tạo heatmap:

- Sử dụng sns.heatmap để tạo heatmap từ DataFrame cosine similarity.
- Hiển thị giá trị tương đồng trên mỗi ô.

Lưu kết quả:

- Lưu hình ảnh heatmap dưới dạng file PNG.

2. THỰC HIỆN ĐÁNH GIÁ SIMILARITY VECTOR NHÚNG

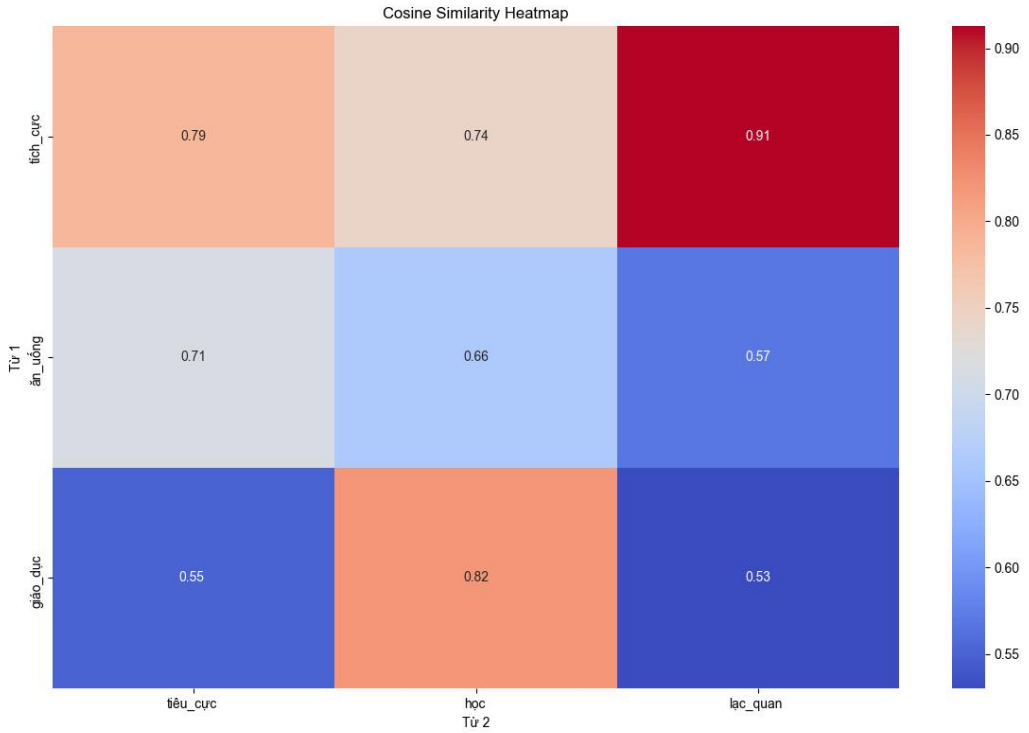
```
#----------------------THỰC HIỆN-------------------------#
# Gọi hàm tải mô hình
W1, W2, vocab = load()

# Kiểm tra cosin của các cặp từ giữa word1 (context) và word2 (center)
word1 = ["tích_cực", "ăn_uống", "giáo_dục"]
word2 = ["tiêu_cực", "học", "lạc_quan"]
if word1.__sizeof__() < word2.__sizeof__():
    print("Danh sách 1 phải có số lượng từ nhiều hơn hoặc bằng số lượng từ của danh sách 2!!!")
    exit()
for feature1 in word1:
    for feature2 in word2:
        try:
            var = compute_cosine_similarity(feature1,feature2,vocab,W1)
            print(f'Độ tương đồng <{feature1}>|<{feature2}>: {var:.2f}')
        except:
            print(f"Không tìm thấy một trong các từ '{feature1}' hoặc '{feature2}' trong từ điển!")

# Gọi hàm lưu heatmap
plot_similarity_heatmap(word1, word2, vocab, W1)
```

- Gọi hàm tải mô hình
- Kiểm tra cosine similarity giữa các từ trong word1 và word2
- Kiểm tra xem word1 có đủ số lượng từ không (**ít nhất phải bằng hoặc nhiều hơn** word2). Nếu không đúng điều kiện, dừng chương trình.
- Tạo heatmap để trực quan hóa độ tương đồng

3. COSINE SIMILARITY



Biểu đồ trực quan hóa heatmap Cosine Similarity

1) Khái niệm

Cosine Similarity (Độ tương đồng cosine) là một phương pháp đo lường độ tương đồng giữa hai vectơ trong không gian nhiều chiều, không phụ thuộc vào độ dài của vectơ. Phổ biến trong xử lý ngôn ngữ tự nhiên (NLP), khai phá dữ liệu văn bản, và các ứng dụng khác khi muốn đo độ tương đồng về **hướng** thay vì độ dài.

2) Thuật toán

Công thức Cosine Simularity được tính như sau:

$$CosineSimilarity = \frac{A \cdot B}{||A||||B||}$$

- A và B là vector nhúng của hai từ.
- $A \cdot B$ là tích vô hướng của hai vector.
- $||A||$ và $||B||$ là độ dài (norm) của từng vector.

Giá trị cosine similarity nằm trong khoảng [0, 1]:

- Giá trị 1 biểu thị sự tương đồng hoàn toàn.
- Giá trị 0 thể hiện sự độc lập hoặc không liên quan.
- Dựa vào biểu đồ, mức độ tương đồng bắt đầu cao lên từ 0.6 trở đi, nó sẽ được chia ra như sau:

  - **Cặp từ có giá trị cosine cao** (>0.7): Các từ có ngữ nghĩa gần gũi, thường xuất hiện trong các ngữ cảnh tương tự.
  - **Cặp từ có giá trị cosine trung bình** (khoảng 0.5-0.7): Các từ có mối liên hệ ngữ nghĩa nhất định nhưng không mạnh mẽ.
  - **Cặp từ có giá trị cosine thấp** (<0.5): Các từ không liên quan hoặc ngữ nghĩa khác biệt.

Cosine Similarity sẽ có giá trị cao khi:

- **Hướng gần giống nhau** trong không gian vector.
- **Ngữ nghĩa tương tự nhau** (tương đồng về ý nghĩa hoặc ngữ cảnh).
- **Tần suất sử dụng gần nhau** trong ngữ liệu huấn luyện.

3) Phân tích:

```
Độ tương đồng <tích_cực>|<tiêu_cực>: 0.79
Độ tương đồng <tích_cực>|<học>: 0.74
Độ tương đồng <tích_cực>|<lạc_quan>: 0.91
Độ tương đồng <ăn_uống>|<tiêu_cực>: 0.71
Độ tương đồng <ăn_uống>|<học>: 0.66
Độ tương đồng <ăn_uống>|<lạc_quan>: 0.57
Độ tương đồng <giáo_dục>|<tiêu_cực>: 0.55
Độ tương đồng <giáo_dục>|<học>: 0.82
Độ tương đồng <giáo_dục>|<lạc_quan>: 0.53
```

Kết quả Cosine Similarity qua Terminal

Cặp từ *tích_cực* | *tiêu_cực* **(0.79)**

- Mặc dù hai từ này mang ý nghĩa trái ngược nhau, chúng vẫn có mức độ tương đồng tương đối cao (0.79). Điều này có thể vì:
    - Cả hai thường xuất hiện trong cùng ngữ cảnh để so sánh hoặc đối lập (ví dụ: phân tích tích cực và tiêu cực trong các bài luận xã hội).
    - Hai từ này có thể thuộc cùng chủ đề (như cảm xúc, thái độ).

Cặp từ *tích_cực* | *học* **(0.74)**

- Tương đồng khá cao (0.74) cho thấy:
    - Từ *tích_cực* có thể được sử dụng để miêu tả thái độ hoặc phương pháp học tập (*học tích cực*).
    - Trong ngữ cảnh giáo dục, các từ này thường xuất hiện gần nhau để nói về động lực học tập hoặc sự phát triển cá nhân.

Cặp từ *tích_cực* | *lạc_quan* **(0.91)**

- Tương đồng rất cao, cho thấy:
    - *Tích_cực* và *lạc_quan* đều thuộc nhóm từ ngữ cảm xúc tích cực.
    - Chúng có thể được sử dụng thay thế nhau trong nhiều ngữ cảnh (ví dụ: *tích cực trong tư duy* cũng có thể đồng nghĩa với *lạc quan*).

Cặp từ *ăn_uống* | *tiêu_cực* **(0.71)**

- Tương đồng tương đối cao, có thể giải thích bởi:
    - Trong các ngữ cảnh thảo luận về sức khỏe hoặc thói quen ăn uống, *ăn_uống* có thể liên quan đến các khía cạnh tiêu cực như thói quen xấu (ăn uống không lành mạnh, chế độ ăn tiêu cực).
    - Điều này đặc biệt phổ biến trong các bài nghị luận về sức khỏe.

Cặp từ *ăn_uống* | *học* **(0.66)**

- Tương đồng trung bình, có thể vì:
    - *Ăn_uống* và *học* có thể liên quan qua các chủ đề chung về lối sống lành mạnh, ví dụ: chế độ ăn uống ảnh hưởng đến khả năng học tập hoặc tập trung.
    - Tuy nhiên, mối liên hệ này không quá trực tiếp, nên giá trị tương đồng thấp hơn các cặp khác.

Cặp từ *ăn_uống* **|** *lạc_quan* **(0.57)**

- Tương đồng thấp hơn, do:
    - *Ăn_uống* và *lạc_quan* ít liên quan trực tiếp về ý nghĩa ngữ cảnh. Tuy nhiên, một mối liên hệ nhỏ có thể là chế độ ăn uống ảnh hưởng đến cảm xúc, tinh thần (ăn uống lành mạnh có thể giúp lạc quan hơn).

Cặp từ *giáo_dục* **|** *tiêu_cực* **(0.55)**

- Tương đồng thấp, có thể vì:
    - *Giáo_dục* thường mang ý nghĩa tích cực, nhưng có thể liên quan đến các khía cạnh tiêu cực (như hệ thống giáo dục chưa hoàn thiện, áp lực học tập).
    - Dù vậy, hai từ này không thường xuất hiện cùng nhau.

Cặp từ *giáo_dục* **|** *học* **(0.82)**

- Tương đồng cao, do:
    - *Giáo_dục* và *học* có quan hệ chặt chẽ về mặt ngữ nghĩa. Học tập là một phần chính trong giáo dục, và chúng thường được dùng chung trong cùng ngữ cảnh.

Cặp từ *giáo_dục* **|** *lạc_quan* **(0.53)**

- Tương đồng thấp, do:
    - *Giáo_dục* và *lạc_quan* không liên quan chặt chẽ. Tuy nhiên, giáo dục có thể thúc đẩy sự phát triển thái độ tích cực (*lạc quan*) trong học sinh, nên vẫn tồn tại một mối liên hệ nhỏ.

4) Nhận xét:

- Đường chéo chính của ma trận luôn có giá trị bằng 1, vì nó biểu diễn độ tương đồng của một từ với chính nó.
- Các từ có liên hệ mạnh (giá trị cao > 0.7) như "giáo_dục" và "học", “tích_cực” và “lạc_quan” có thể là do sự liên kết ngữ nghĩa cao hoặc tần suất sử dụng với nhau nhiều.
- Một số từ như "xấu" và "tiêu_cực" có cosine similarity cao nhưng không mạnh mẽ (giá trị trung bình, 0.5-0.7), có thể là do ngữ cảnh được sử dụng cho nhiều lĩnh vực.

- Cặp từ như “lạc_quan” và “giáo_dục”, cosine similarity rất thấp (giá trị thấp < 0.5) vì ít liên quan với nhau và không được sử dụng thường xuyên với nhau.

5) Giải thích:

- Cosine similarity đo lường góc giữa hai vector nhúng, cho biết mức độ tương đồng về hướng trong không gian vector.

- Những từ có cosine similarity cao thường xuất hiện trong các ngữ cảnh tương tự, còn những từ với giá trị thấp thuộc các ngữ cảnh khác biệt, ít được đề cập với nhau.

KẾT LUẬN

✧ **Tầm quan trọng của Skip-Gram:**

- Skip-Gram là một phương pháp hiệu quả trong Word2Vec, cho phép dự đoán ngữ cảnh từ từ trung tâm và ngược lại, từ đó tạo ra các vector nhúng đại diện ý nghĩa ngữ nghĩa và cú pháp của từ.
- Mô hình thể hiện ưu điểm vượt trội trong việc xử lý dữ liệu ngôn ngữ hiếm hoặc các từ xuất hiện ít, đặc biệt phù hợp với dữ liệu tiếng Việt.

✧ **Kết quả và đánh giá thực nghiệm:**

- Quá trình huấn luyện đã cho thấy độ mất mát giảm dần qua các epoch, đạt trạng thái hội tụ và chứng tỏ mô hình đã học tốt các đặc trưng từ dữ liệu.
- Các vector nhúng phản ánh rõ ràng tương quan ngữ nghĩa giữa các từ trong không gian vector, minh họa qua biểu đồ heatmap độ tương đồng cosine.

✧ **Hạn chế và đề xuất:**

- Mô hình gặp khó khăn trong việc xử lý các từ phổ biến, đặc trưng, do dữ liệu ngữ cảnh chưa đủ phong phú.
- Để cải thiện, cần mở rộng tập dữ liệu huấn luyện và điều chỉnh tham số mô hình, như embedding_dim hoặc window_size, để tăng khả năng nhận diện.

✧ **Đóng góp:**

- Tiểu luận đã minh họa rõ ràng cách xây dựng và triển khai mô hình Skip-Gram trên dữ liệu tiếng Việt, bao gồm các bước tiền xử lý, huấn luyện, trực quan hóa và đánh giá kết quả.
- Điều này khẳng định tiềm năng của Skip-Gram trong phát triển các ứng dụng xử lý ngôn ngữ tự nhiên hiện đại, đặc biệt là đối với ngôn ngữ có dữ liệu hạn chế như tiếng Việt.

LỜI CẢM ƠN

Để hoàn thành tiểu luận này, em xin gửi lời cảm ơn chân thành đến:

Trường Đại Học Sài Gòn và Khoa Công Nghệ Thông Tin vì đã tạo điều kiện về cơ sở vật chất với hệ thống thư viện hiện đại, đa dạng các loại sách, tài liệu thuận lợi cho việc tìm kiếm, nghiên cứu thông tin.

Xin cảm ơn giảng viên bộ môn "Xử lý ngôn ngữ tự nhiên" - Thầy Nguyễn Tuấn Đăng đã giảng dạy tận tình, chi tiết để em có đủ kiến thức và vận dụng chúng vào bài tiểu luận này.

Do chưa có nhiều kinh nghiệm làm tốt bài tiểu luận cũng như có những hạn chế về kiến thức, trong bài tiểu luận chắc chắn sẽ không tránh khỏi những thiếu sót. Rất mong nhận được sự nhận xét, ý kiến đóng góp cũng như sự cảm thông từ phía Thầy.

Lời cuối cùng, em xin kính chúc thầy nhiều sức khỏe, thành công và hạnh phúc.

Em xin chân thành cảm ơn.