Word Embedding for Vietnamese using Viwiki with Word2Vec and FastText

Ta Nguyen Thanh
Institute for Artificial Intelligence
UET-VNU
23020437@vnu.edu.vn

October 8, 2025

1 Introduction

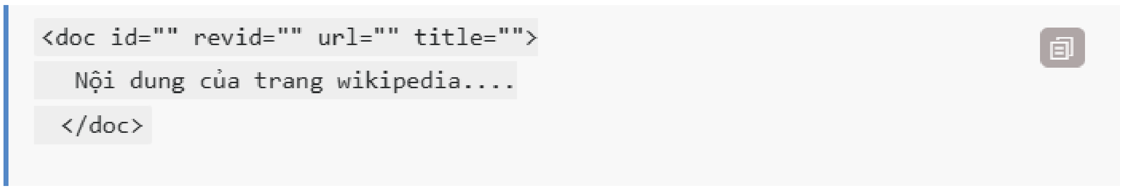
Word embedding is a fundamental technique in natural language processing (NLP) that represents words as continuous numerical vectors, capturing semantic and syntactic relationships between them. Instead of treating words as discrete symbols, embeddings enable models to understand linguistic similarity through vector operations. Among the most widely used approaches are Word2Vec and FastText, both developed by researchers at Facebook and Google.

Word2Vec learns word representations based on their surrounding context using architectures such as Continuous Bag of Words (CBOW) and Skip-gram. It captures meaningful relationships like analogies and word similarities but treats each word as an independent unit. FastText, on the other hand, extends Word2Vec by representing words as combinations of character n-grams, allowing it to handle rare and out-of-vocabulary words more effectively.

In this report, we compare and analyze the use of Word2Vec and FastText for generating word embeddings from Vietnamese corpora, focusing on their training methods, computational efficiency, and quality of the resulting representations. There are also visualizations on the correlations in the words' meaning as a 2x2 grid.

2 Data

The data was collected from Vietnam Wikipedia as a compressed file. After downloading the zipped file locally, we use the wikiextractor library so that the output data has the following format:



```
<doc id="" revid="" url="" title="">
  Nội dung của trang wikipedia....
 </doc>
```

Figure 1: Format of extracted data

For preprocessing, we first used regular expression to remove the HTML tags in the documents. Then we split the sentences simply based on their punctuations. After that, words from each sentence are segmented and normalized by removing all punctuations plus converting into lowercase. Next, from a **stopwords.csv** file, we remove all stopwords within the documents. The cleaned data is finally written into a single text file **datatrain.txt**. After the preprocessing method, we move forward to training the model using **Word2Vec** and **FastText** from the **gensim** library. Due to hardware constraints, only hafl of the original data is taken as the training data.

3 Methodology

3.1 Word2Vec

Word2Vec is a neural embedding model that represents each word w as a continuous vector $\mathbf{v}_w \in \mathbb{R}^d$. It learns these representations by predicting context words surrounding a target word within a window. There are two main architectures: **Continuous Bag of Words (CBOW)** and **Skip-gram**.

CBOW model. The **CBOW** model predicts the target word given its surrounding context:

$$P(w_t \mid w_{t-c}, \dots, w_{t+c}) = \frac{\exp(\mathbf{u}_{w_t}^\top \mathbf{h})}{\sum_{w=1}^{|V|} \exp(\mathbf{u}_w^\top \mathbf{h})}$$

where

$$\mathbf{h} = \frac{1}{2c} \sum_{-c \le j \le c, j \ne 0} \mathbf{v}_{w_{t+j}},$$

$\mathbf{v}$ and $\mathbf{u}$ are the input and output word vectors, and $|V|$ is the vocabulary size.

Skip-gram model. The **Skip-gram** model, in contrast, predicts surrounding context words given the target word:

$$P(w_{t+j} \mid w_t) = \frac{\exp(\mathbf{u}_{w_{t+j}}^\top \mathbf{v}_{w_t})}{\sum_{w=1}^{|V|} \exp(\mathbf{u}_w^\top \mathbf{v}_{w_t})}$$

The objective of **Word2Vec** is to minimize the negative log-likelihood over the corpus:

$$L = -\sum_{t=1}^{T} \sum_{-c \le j \le c, j \ne 0} \log P(w_{t+j} \mid w_t)$$

Because the denominator of the softmax is computationally expensive, practical implementations use approximations such as **Negative Sampling** or **Hierarchical Softmax**.

3.2 FastText

FastText extends **Word2Vec** by representing each word as a bag of character n-grams, which allows it to capture subword information and handle rare or unseen words effectively.

Each word w is represented as a set of subword units G_w, and its vector is computed as the sum of its n-gram embeddings:

$$\mathbf{v}_w = \sum_{g \in G_w} \mathbf{z}_g$$

where $\mathbf{z}_g$ is the vector representation of the subword g.

The **Skip-gram** training objective is then modified to use $\mathbf{v}_w$ instead of a single word vector:

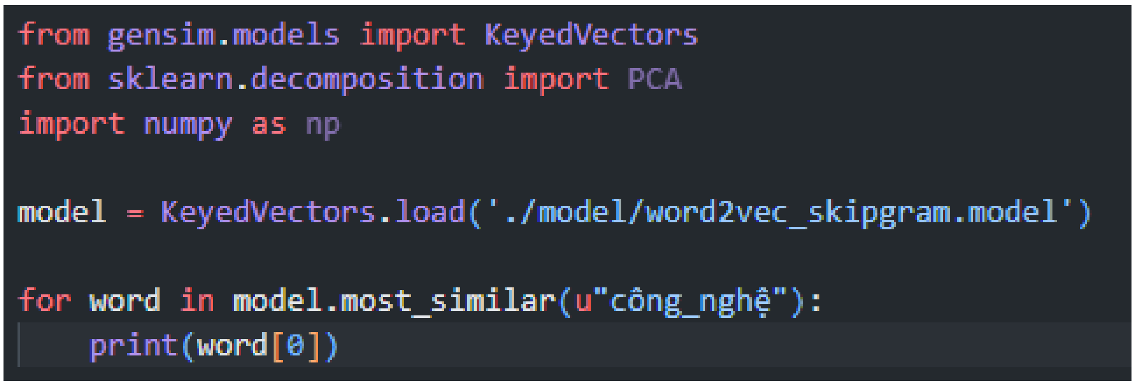
$$P(w_{t+j} \mid w_t) = \frac{\exp(\mathbf{u}_{w_{t+j}}^\top \mathbf{v}_{w_t})}{\sum_{w=1}^{|V|} \exp(\mathbf{u}_w^\top \mathbf{v}_{w_t})}$$

This approach enables **FastText** to generalize better to out-of-vocabulary (OOV) words and morphologically rich languages such as Vietnamese.

3.3 Implementation Details

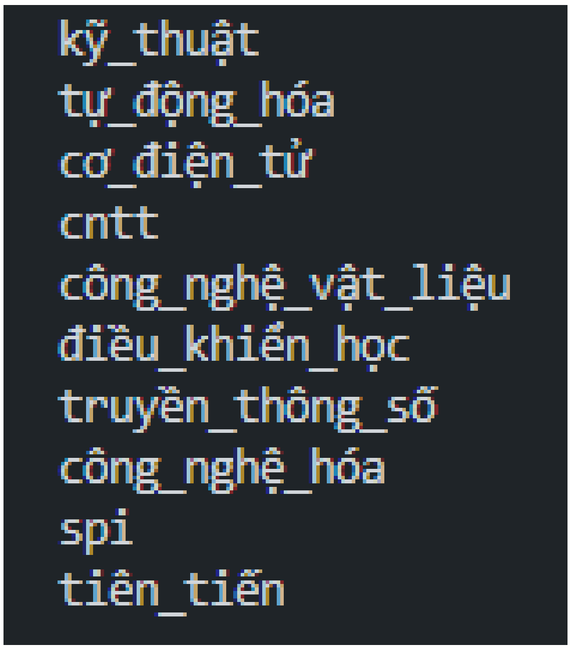
For **Word2Vec**, we experimented with **Skip-gram** training where given the input Vietnamese corpus as the training data, the dimensionality of the word embeddings would be 100. For context window length, we set it at 10 so the model would predict within 5 words before and after the target word. We also removed all words that appeared less than 4 times in the corpus, therefore noise from very rare words was ensured to be reduced. In order to speed up the training process and computation, we utilized 4 CPU threads to handle the data in a parallelized manner.

For **FastText**, with the implementation of character n-gram information, we adopted similar parameters in comparison to the Word2Vec model counterpart. The only difference is that for this model, we trained it for 10 epochs to improve embedding quality and overall accuracy. This would generate embeddings for out-of-vocabulary words and better handle morphologically rich languages.



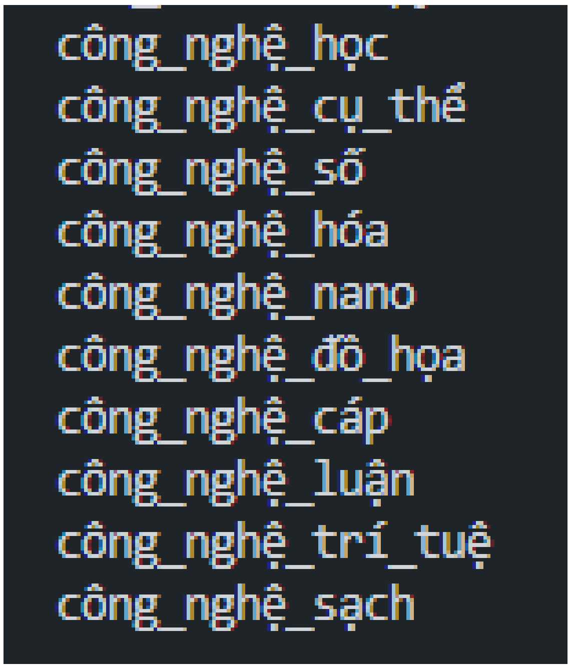
```python
from gensim.models import KeyedVectors
from sklearn.decomposition import PCA
import numpy as np

model = KeyedVectors.load('./model/word2vec_skipgram.model')

for word in model.most_similar(u"công_nghệ"):
    print(word[0])
```

Figure 2: Script to get the most similar words of a target word



```
kỹ_thuật
tự_động_hóa
cơ_điện_tử
cntt
công_nghệ_vật_liệu
điều_khiển_học
truyền_thông_số
công_nghệ_hóa
spi
tiên_tiến
```

(a) Most similar words using Word2Vec



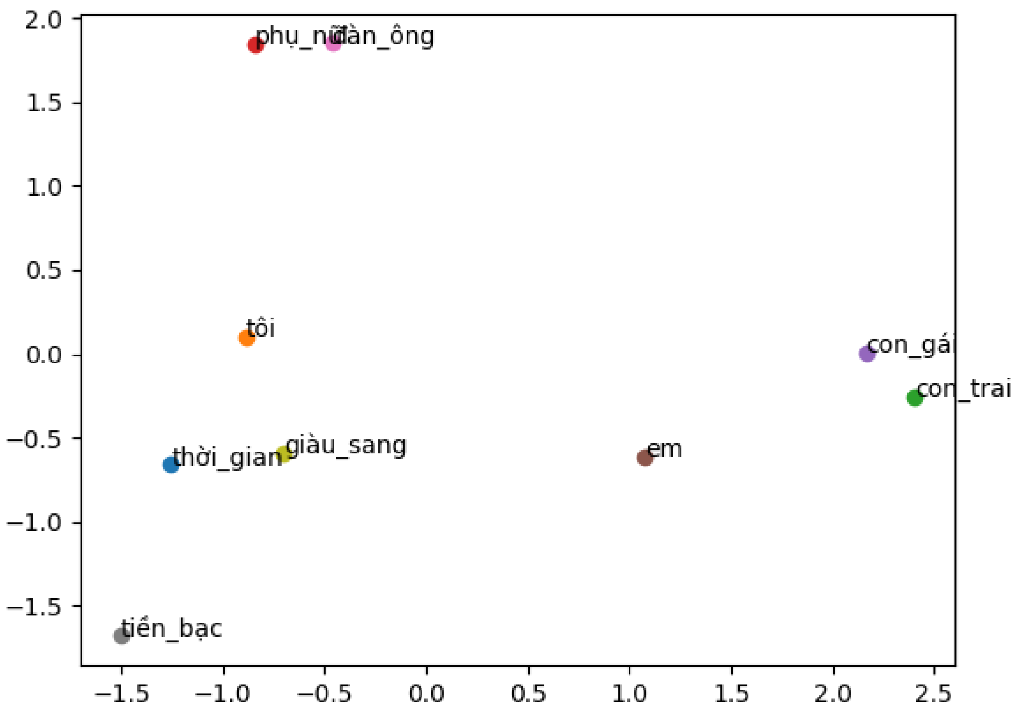
```
công_nghệ_học
công_nghệ_cụ_thể
công_nghệ_số
công_nghệ_hóa
công_nghệ_nano
công_nghệ_đồ_họa
công_nghệ_cáp
công_nghệ_luận
công_nghệ_trí_tuệ
công_nghệ_sạch
```

(b) Most similar words using FastText

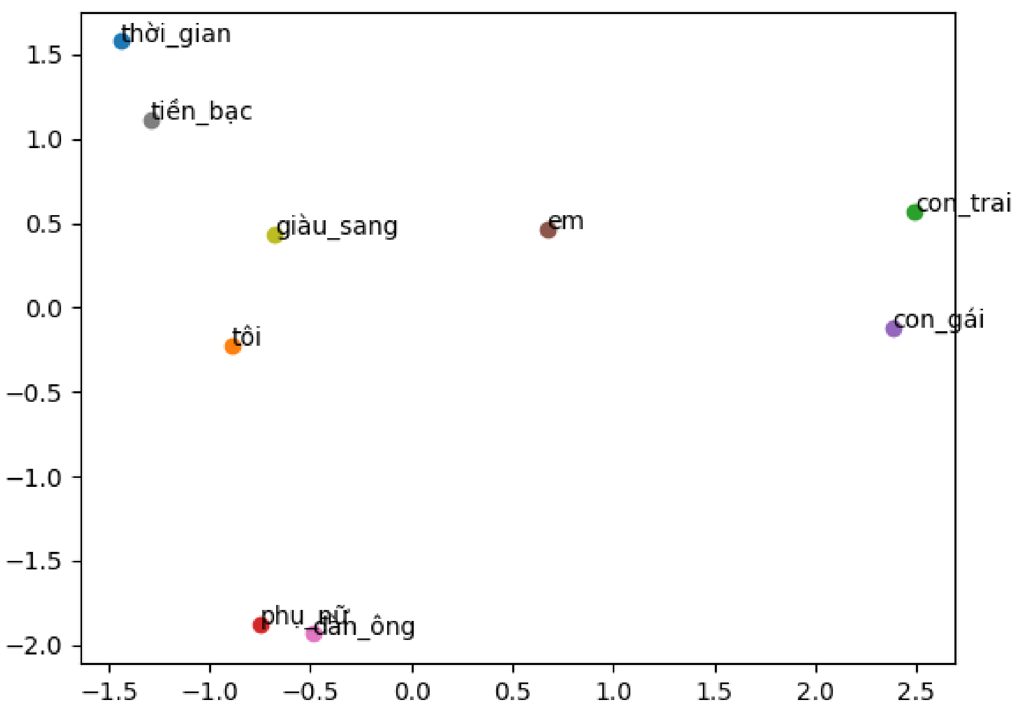
Figure 3: Most similar words evaluation

4 Experiments

All the code for this assignment can be found in this GitHub repository. It took 20 minutes and 25 minutes for the **Word2Vec** model and the **FastText** model to complete their training processes, respectively. After training, the models' visualizations are shown to see the correlation between the words as in Figure 4. Furthermore, by using a script like Figure 2, for a given word, its most similar words are given in Figure 3.



(a) Visualize word correlation with Word2Vec



(b) Visualize word correlation with FastText

Figure 4: Word Correlation of two models

5 Conclusion

Our comparative analysis reveals that **FastText** produces more balanced and centralized embeddings for Vietnamese text compared to **Word2Vec**. While both models capture semantic relationships among family and social terms, **Word2Vec** exhibits greater polarization, particularly in gender-related vocabulary positioning. **FastText**'s subword-aware architecture appears better suited for Vietnamese's agglutinative characteristics, resulting in more stable geometric representations without extreme outliers. These findings suggest **FastText** may offer reduced bias amplification and more reliable embeddings for downstream Vietnamese NLP tasks

References

[1] Viblo, *Xây dựng mô hình không gian vector cho tiếng Việt*, Blog post, 2017. `https://viblo.asia/p/xay-dung-mo-hinh-khong-gian-vector-cho-tieng-viet-GrLZDXr2Zk0`

[2] Quang Pham, *FramgiaBlog*, GitHub repository. `https://github.com/quangph-1686/FramgiaBlog`

[3] Wikimedia Foundation, *Vietnamese Wikipedia Database Dump*, Dataset, 2024. `https://dumps.wikimedia.org/viwiki/latest/viwiki-latest-pages-articles.xml.bz2`

[4] G. Attardi, *WikiExtractor*, GitHub repository. `https://github.com/attardi/wikiextractor`