Natual Language Processsing
Spring 2025

Assignment 1

Name Ke Li **Student No.** 2024210837

1 Word2Vec

1.1 Word2Vec Implementation

The task there is to implement a simple version of Word2Vec, which is a popular model for learning word embeddings. The model is trained on a small subset of the English Wikipedia. Before training, the text is processed. I implement a Word2Vec model based on Continuous Bag of Words (CBOW), with or without Negative sampling. Below is the implementation of the Word2Vec model.

1.1.1 Data Preprocessing

I use a subset of the English Wikipedia as the training data, which is part of the `enwiki-20241201-pages-articles-multistream1.xml-p1p41242.bz2`. Because the original data is too large, I only use the first **1000** lines of the data for training.

The data preprocessing steps are as follows:

1. Tokenization: I use the default tokenizer in the `gensim.corpora.wikicorpus` package to tokenize the text, which is based on the regular expression `r'^[\w\'"]+$'`. The minimum length of the token is set to 2, and the maximum length is set to 15.
2. Lowercasing: I convert all the tokens to lowercase.
3. Remove stopwords: I remove the stopwords from the tokens. The English stopwords are defined in the `nltk.corpus.stopwords` package.

The code for data preprocessing is `cbow/create_corpus.py`.

1.1.2 Word2Vec Model

The model is based on the Continuous Bag of Words (CBOW) architecture. The basic idea of CBOW is to predict the target word based on the context words. The model architecture is as follows:

So the model has two `Embedding` layers, one for the input words and the other for the output words. The input words are the context words, and the output word is the target word. The model is trained to minimize the negative log-likelihood of the target word given the context words.

To show the principle of the model, let me denote the input words (context words) as $w_{m-k}, w_{m-(k-1)}, \dots, w_{m-1}, w_{m+1}, w_2, \dots, w_{m+k}$, and the target word as w_m. The vocabulary size is $V > m$, and the embedding dimension is d. The model parameters are the input embedding matrix $\boldsymbol{W}_{\text{in}} \in \mathbb{R}^{V \times d}$, the output embedding matrix $\boldsymbol{W}_{\text{out}} \in \mathbb{R}^{V \times d}$. After the embedding layers, the input words are averaged to get the input vector $v_m = \frac{1}{2k} \sum_{i=-k, i \neq 0}^{k} \boldsymbol{W}_{\text{in}}[w_{i+m}]$, and the output embedding vector is $u_n = \boldsymbol{W}_{\text{out}}[w_n], n = 1, 2, \dots, V$. The model is trained to minimize the negative log-likelihood of the target word given the context words, which is defined as:

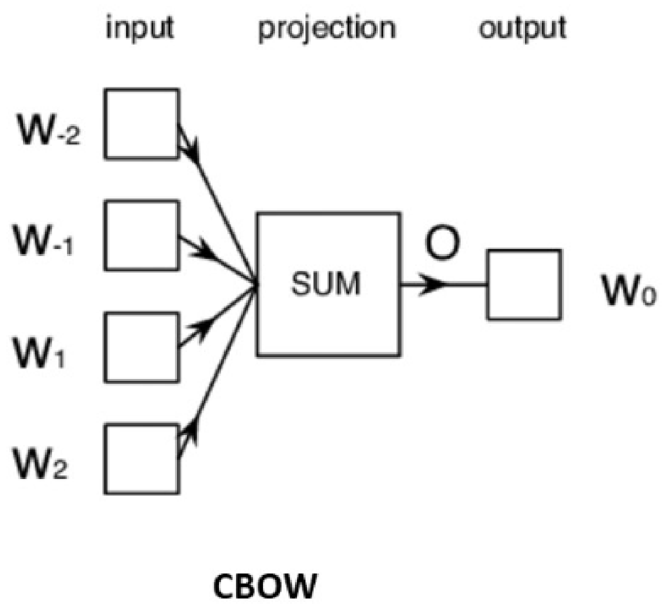

Figure 1 CBOW Model Architecture

$$\begin{aligned}\mathcal{L}(\theta) &= -\log p(w_m | w_{m-k}, w_{m-(k-1)}, \dots, w_{m-1}, w_{m+1}, w_2, \dots, w_{m+k}; \theta) \\ &= -\log \frac{\exp(u_m^T v_m)}{\sum_{n=1}^{V} \exp(u_n^T v_m)} \\ &= -u_m^T v_m + \log \sum_{n=1}^{V} \exp(u_n^T v_m)\end{aligned} \tag{1}$$

To implement the model above, we need to create the dataset based on the context words and the target words. In each sentence, we use the context words to predict the target word. The context words are the input, and the target word is the output. The window size is set to 5 (namely the context size k is 2). For the beginning and the end of the sentence, we pad the sentence with the special token `<UNK>`. The code for creating the dataset is `cbow/create_dataset.py`.

And the training code is `cbow/train_cbow.py`. The model is trained with the Adam optimizer, and the learning rate is set to 0.002. The number of epochs is set to 10. The model is trained on the first 1000 lines of the English Wikipedia, and the vocabulary size is set to 126317. The embedding dimension is set to 100, 300, and 500, respectively. The performance of the model is discussed in the next section.

Negative Sampling The drawback of the softmax function is that it is computationally expensive when the vocabulary size is large. We must update all the output weights for each training sample, which is time-consuming. To address this issue, we can use the negative sampling technique. The idea of negative sampling is to sample a small number of negative samples (usually 5-20) for each positive sample. The model is trained to distinguish the positive sample from the negative samples. The negative sampleing method is a simplified version of Noise Contrastive Estimation (NCE).

The negative samples are sampled from the noise distribution, which is the unigram distribution of the vocabulary. The probability of sampling a word w is defined as $P(w) = \frac{f(w)^{0.75}}{\sum_{w'} f(w')^{0.75}}$, where $f(w)$ is the frequency of the word w in the training data. The negative samples are sampled from the noise distribution. The code for negative sampling is `cbow/train_cbow_neg.py`.

For theroetical details, we denote the negative samples as $w_{m,1}, w_{m,2}, \dots, w_{m,N}$, where N is the number of negative samples. The loss function is defined as:

$$\begin{aligned}\mathcal{L}(\theta) &= -\log p(w_m | w_{m-k}, w_{m-(k-1)}, \dots, w_{m-1}, w_{m+1}, w_2, \dots, w_{m+k}; \theta) \\ &= -\log \sigma(u_m^T v_m) - \sum_{n=1}^{N} \log \sigma(-u_n^T v_m)\end{aligned} \tag{2}$$

where $\sigma(x) = \frac{1}{1+\exp(-x)}$ is the sigmoid function. The model is trained to minimize the negative log-likelihood of the target word and the negative samples. We can see that the computing cost is reduced from $O(V)$ to $O(N)$, which is much more efficient.

However, I generate the negative samples for all the target words in the dataset, which is quite time-consuming. So I change the negative sampling method to sample the negative samples for each positive sample in the training process. The code for negative sampling is `cbow/create_dataset_neg.py`.

1.2 Word2Vec Improvements

There are three improvements in the instructions for the Word2Vec model:

- Incorporating word sense disambiguation: there are different senses for the same word, so one word may have different embeddings to learn.
- Leveraging external word knowledge sources: we can use the external knowledge sources to improve the word embeddings, such as WordNet, BabelNet, etc.
- Evaluation character-level embeddings: we can evaluate the character-level embeddings to improve the word embeddings.

I implement the second improvement by incorporating the external knowledge source WordNet. WordNet is a lexical database of English, which is used to improve the word embeddings. The theroetical details are as follows: we only consider the sense from the context words, while we ignore the relationship among different target words. We use the external knowledge to find the synonyms and hypernyms of the target word, and we use the synonyms and hypernyms to improve the word embeddings. We add the synonyms and hypernyms loss to the original loss function. The loss function is defined as:

$$\mathcal{L}(\theta) = \mathcal{L}_{\text{CBOW}} + \lambda_{syn} MSE(\boldsymbol{v}_m, \boldsymbol{v}_{\text{syn}}) + \lambda_{hyp} \text{cos\_sim}(\boldsymbol{v}_m, \boldsymbol{v}_{\text{hyp}}) \tag{3}$$

where $\mathcal{L}_{\text{CBOW}}$ is the original loss function, $\boldsymbol{v}_m$ is the original word embedding, $\boldsymbol{v}_{\text{syn}}$ is the synonym embedding, $\boldsymbol{v}_{\text{hyp}}$ is the hypernym embedding, λ_{syn} and λ_{hyp} are the hyper-parameters, $MSE(\cdot)$ is the mean squared error, and $\text{cos\_sim}(\cdot)$ is the cosine similarity. The code for incorporating WordNet is `cbow/train_enhanced_cbow.py`.

2 Pretrained Model Embedding Generation

In this section, I generate the word embeddings based on the pretrained Word2Vec model in Huggingface. I use the `openbmb/MiniCPM-1B-sft-bf16` model. The methods of extracting the word embeddings are implemented in the `cbow/evaluate_sentences.py`.

3 Evaluation

In this section, I evaluate the performance of the Word2Vec model based on the following metrics:

- Word Similarity Task: I use the WordSim-353 dataset to evaluate the word similarity task. The WordSim-353 dataset contains 353 word pairs, and each word pair is annotated with a similarity score. The similarity score is in the range of 0 to 10. The evaluation metric is the Spearman correlation coefficient between the predicted similarity score and the annotated similarity score.

- Paraphrase Detection Task: I use the MRPC dataset to evaluate the paraphrase detection task. The MRPC test dataset contains 1726 sentence pairs, and each sentence pair is annotated with a label (0 for non-paraphrase and 1 for paraphrase). The evaluation metric is the accuracy of the model.

The code for evaluation is `cbow/test_similarity.py` and `cbow/evaluate_sentences.py`.

3.1 Word Similarity Task

I compute the Spearman correlation coefficient between the predicted similarity score and the annotated similarity score. The results are shown in Table 1.

The Spearman correlation shows the performance of the Word2Vec model on the word similarity task. It ranges from -1 to 1, where 1 means the model has a perfect correlation with the annotated similarity score, and -1 means the model has a perfect negative correlation with the annotated similarity score. The results are shown in Table 1. (The 500-dim results are not available for the CBOW model and the CBOW-Enhanced model, and the 300-dim of the CBOW-Enhanced model.The former is due to the memory issue, and the latter is due to the time issue.)

We can draw the following conclusions from the results:

- The CBOW model has the best performance on the word similarity task. Because it uses the full negative samples (all vocabulary) for each positive sample, which can capture the word similarity better. The enhanced CBOW with negative sampling has a better performance than the CBOW model with negative sampling. This shows that the external knowledge source can improve the word embeddings.

- The performance of the model decreases as the embedding dimension increases. This is because the training data I use is two small, the model is overfitting when the embedding dimension is too large. The model has a better generalization ability when the embedding dimension is small.

Table 1 Spearman Correlation of Different CBOW Models with Various Embedding Dimensions

Model Type	**100-dim**	**300-dim**	**500-dim**
CBOW	0.38	0.32	-
CBOW-Neg	0.21	0.20	0.15
CBOW-Enhanced	0.22	-	-

3.2 Paraphrase Detection Task

For the paraphrase detection task, because we get the word embeddings from the pre-trained model, we need to use the sentence embeddings to represent the sentence. Word embeddings are not quite suitable for the sentence-level task. To acquire the sentence embeddings, I use the average of the word embeddings in the sentence. Another method is to use the BERT score, which calculates the similarity between two sentences based on the cosine similarity of the word embeddings. The code for the BERT score is `cbow/evaluate_sentences.py`.

Table 2 Model Performance Comparison (Accuracy and F1 Scores)

Model	**Accuracy**		**F1 Score**		**BERT-Style**
	Base	BERT	Base	BERT	Improvement
MiniCPM-1B-sft-bf16	0.599	0.629	0.644	0.688	+4.8%
CBOW-100	0.648	0.649	0.706	0.692	+0.1%
CBOW-300	0.671	0.641	0.737	0.679	-4.5%
CBOW-Neg-100	0.659	0.652	0.721	0.697	-1.1%
CBOW-Neg-300	0.627	0.659	0.661	0.708	+5.1%
CBOW-Neg-500	0.660	0.657	0.714	0.706	-0.5%
CBOW-Enhanced-100	0.640	0.680	0.692	0.736	+6.3%

The results is shown in Table 2. An interesting observation is that the pre-trained model has a worse performance than the CBOW model on the paraphrase detection task. Our model is trained on the small subset of the English Wikipedia, which is not enough to capture the sentence-level semantics. But the CBOW model has a better performance than the pre-trained model, which shows that sentence embeddings which are based on the word embeddings can not capture the sentence-level semantics well. We can also see this from the accuracy.

4 Disscussion

During the implementation of the Word2Vec model, I encounter some issues:

- The training process is time-consuming. The negative sampling method is quite time-consuming, which is not efficient. I need to sample the negative samples for each positive sample, which is time-consuming. So I using the `gensim.models.Word2Vec` package to train the Word2Vec model. I find that it is much more efficient than the negative sampling method I implemented.

- The paragraph detection task is quite challenging. Since I think the Word2Vec model is not suitable for the sentence-level task, I don't think the results can reflect the performance of the model. As the results show, the pre-trained model has a worse performance than the CBOW model, which is not reasonable. May be I can use BERT to get the sentence embeddings, which can capture the sentence-level semantics better.

The models are uploaded to the `Tsinghua Cloud`.