Text Data Dimension Reduction:

Word Embedding in Dream of the Red Chamber

By Hongyu Mao, Yushi Ye

1. Introduction

Word Embedding is a technique developed by Bengio from 2000. Now it has become the foundation in Natural Language Processing (NLP) field. Among all concepts of Word Embedding, word2vec is the most important and most frequently used method which transforms words from combination of letters into high-dimension vectors that computer can understand and process.

2. Word Splitting and Data Pre-processing

For human beings, we understand text from at least 3 dimension: words, sentences and paragraphs. When computers are dealing with text input, there is no natural way for computers to understand the inner connections between words, sentences and paragraphs. In order to change the form of texts into something that computer can handle, we start from the smallest components of text, which are words.

For English/French and all the other Latin languages, word splitting usually is just split sentences by spaces and commas. But for Chinese it is much more difficult to do so since the language has no natural signs for splitting. Usually we use word splitting packages to split sentences into words. Packages like jieba or NLTK combines word/phrases search and HMM methods to identify the best combinations of characters in a sentence. In this project we use jieba to split the text from Dream of the Read Chamber.

After word splitting, stops words, which are commonly used auxiliary words, and words that are mentioned less than 10 times are excluded. Each of all remaining words is given a specific number as label. Then we replace sentences with lists of remaining words, which means each sentence is transformed into a list of labels with the same word sequence of the original text.

3. One-hot Representation

There have different problems to deal with in NLP. Problems like Pos Identification, text content classification are well solved by One-hot Representation, which represents each word with a long and sparse vector. Entries of the nth word are all 0 but 1 on nth entry, so that the word is transformed into numerical form.

One-hot Representation works well under many scenarios, but it has its own limitations. The position of 1-entry only depends on the sequence of words, which is random. So one-hot vector cannot tell us inner connection between words and leave this part of word to NLP models. At the same time, the dimension of word vector is too high since the vector is too sparse, which makes calculation very difficult when NLP model becomes deeper and more complicated. Then it is easy to understand why One-hot Representation will fail when we want computers to understand the structure of sentences and text.

4. Word2vec implementation scenario

Words in a sentence have special sequence to make the sentence become a human language. We want to identify is a given sentence, or a sequence of words, belongs to human language by building probability model:

$$p(w_1, w_2, \dots, w_n) = p(w_1)p(w_2|w_1) \dots p(w_n|w_{n-1} \dots w_2 w_1)$$

Which can be simplified into:

$$p(w_1, w_2, \dots, w_n) = \prod_{i=1}^{T} p(w_i|Content_i)$$

And the original form of $p(w_i|Content_i)$ is just:

$$p(w_i|Content_i) = \frac{Count(w_i|Content_i)}{\sum_{v=1}^{V} Count(w_{i,v}|Content_i)}$$

Higher probability means higher frequency in training text, which makes the combination more natural and has larger chance to be regarded as natural language.

However, since the number of words is too big, it is almost impossible for us to count and calculate conditional possibilities for all possible combinations of words and contents. In order to make the probability calculation possible for current computers, we need to design and build models to mimic the probability function.

In word2vec, we define the probability function by introducing the concept of energy function. The probability of certain word under certain content is defined to be:

$$P(w|C) = \frac{e^{-E(w,C)}}{\sum_{v=1}^{V} e^{-E(w_v,C)}}$$

Where

$$E(w, C) = -w * C$$

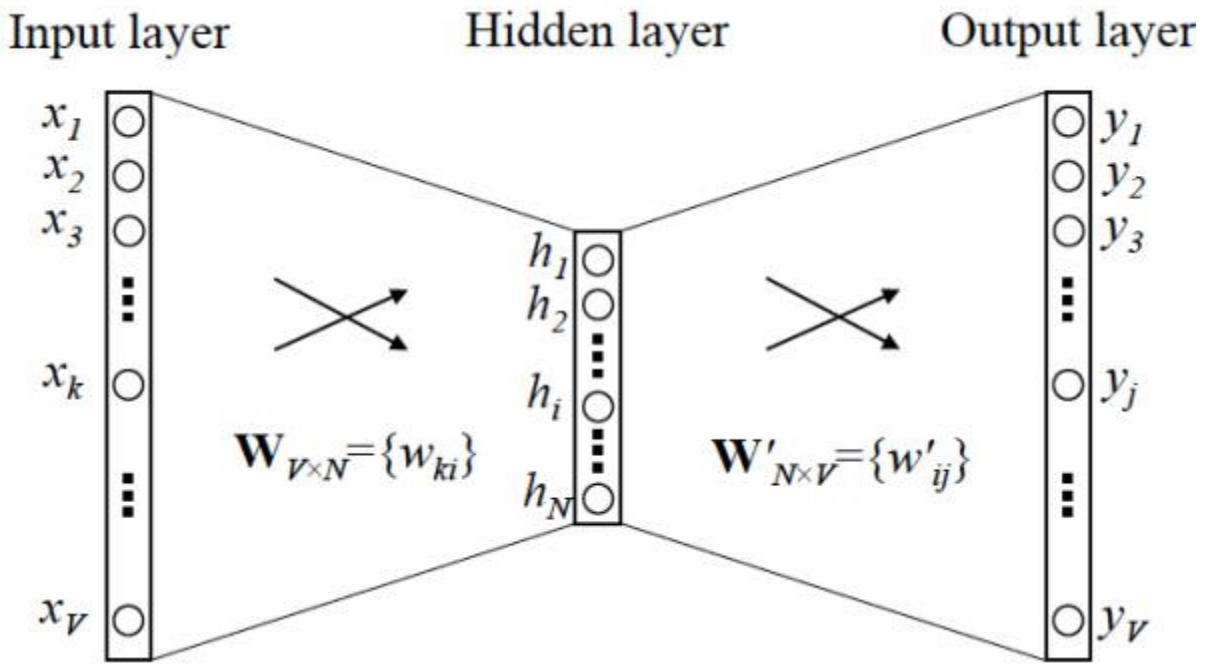
The probability function is just the ratio between energy of given combination and all possible combination. Both w and C are vectors that are long enough so that enough information can be stored. This formula gives us the concept of word2vec.

We can see that word2vec method compresses the one-hot vector into more dense form and forces the word vectors to have inner connections so that the energy function can be evaluated properly, which is the key idea of Word Embedding.

By changing words into vectors, the problem of counting numbers now becomes to how to get reasonable vector values for each words. Algorithms are developed to solve this question.

5. Training models of word2vec: Skip-gram and CBOW

We can see that $P(w|C) = \frac{e^{-E(w,C)}}{\sum_{v=1}^{V} e^{-E(w_v,C)}} = \frac{e^{w*C}}{\sum_{v=1}^{V} e^{w_v*C}}$ is the softmax function of $\{w * C, e^{w_1*C}, \dots, e^{w_V*C}\}$, so that we can build the following model to produce above sequence:

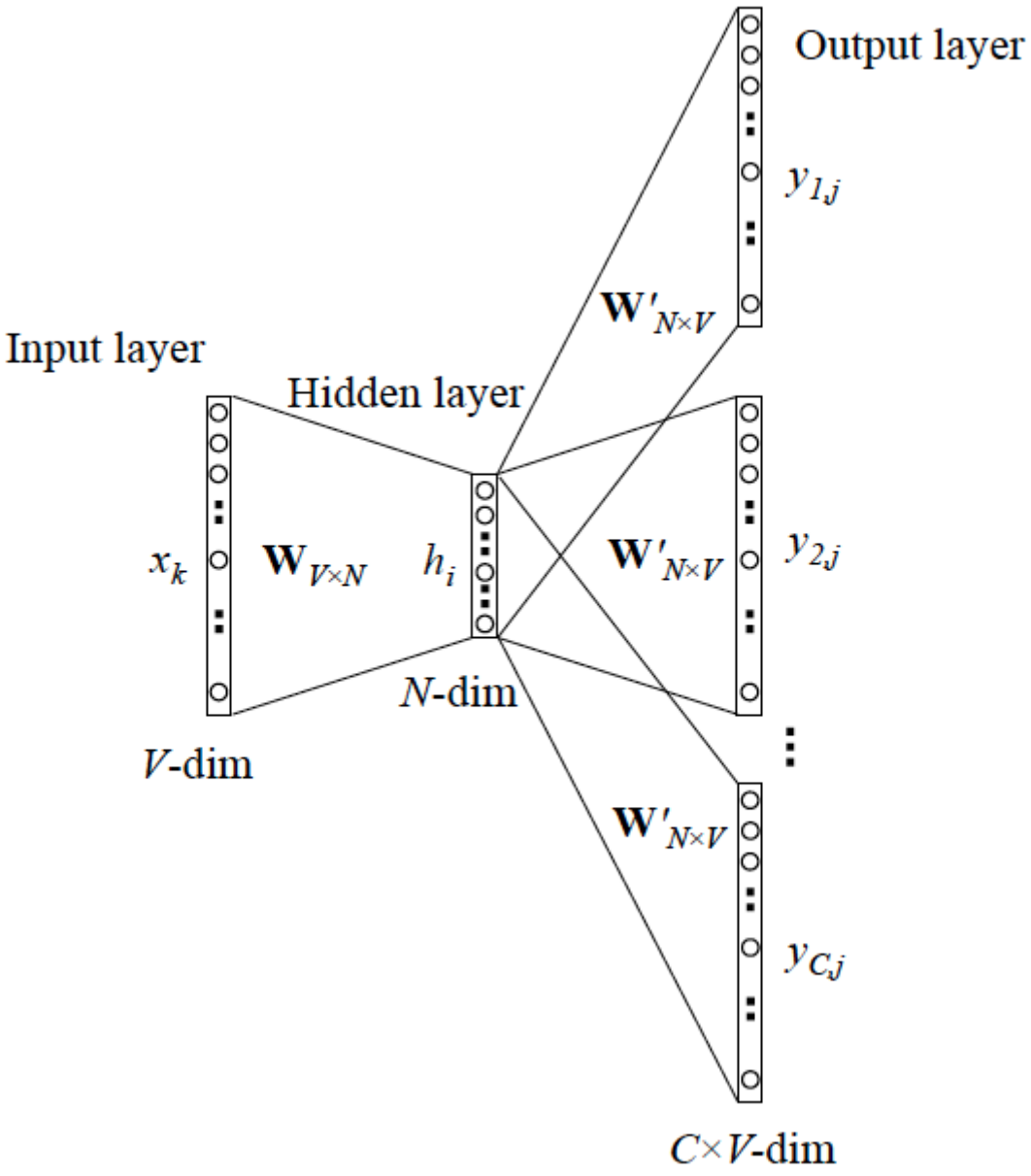


Model I

The input layer is the one-hot vector mentioned in 3., and the hidden layer is nothing about the word vector matrix with its n-th row as the n-th word vector. The output layer is the probability of each word given the existence of input word. By using softmax function before output layer, it is obvious that the model fits the conditional probability function we set.

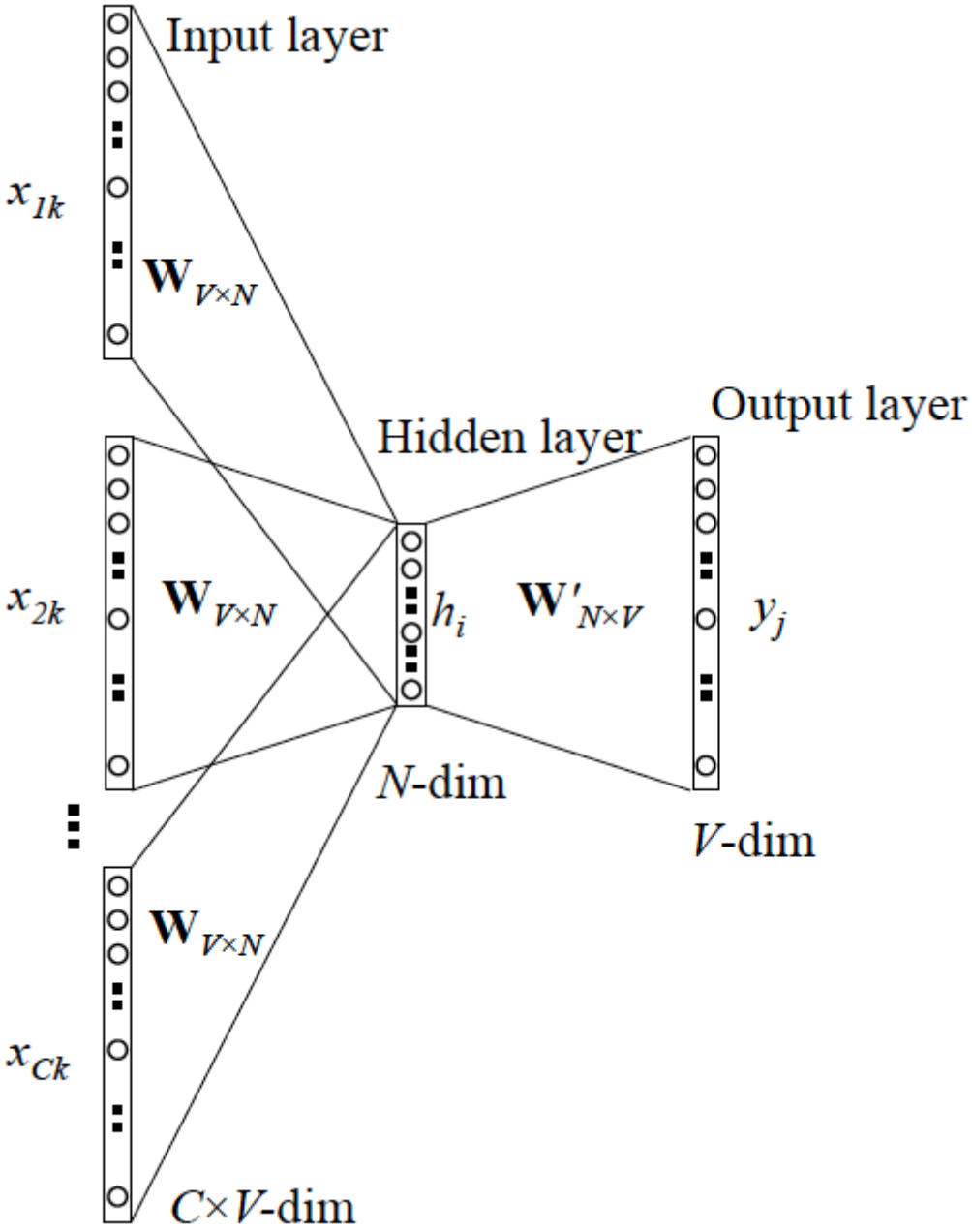
Since there are different ways to define word and context, we can use the word to predict the probability of its surrounding words, or we can use the surrounding words to predict the word lies in the middle. The first way is called Skip-gram and the second way is called CBOW (Continue Bag of Words).

If we only want to predict the last or next word, Model I will be good enough. But if we want to predict a few words around one word, then we need to use:



Model II

As for CBOW mode, the structure is:



Model III

Once traning model structure is set, we are able to calculate the gradients use Back-Propagation to train the inner layer, which is exactly what we want – the word vector matrix.

6. Accelerating algorithms: Hierarchical softmax and Negative Sampling

As stated before, we want to set the word vectors properly so that they can maximize the likelihood of each sentence in training set, which is:

$$l = logL = \frac{1}{N_s}\sum_{i=1}^{T} \log p(w_i|Content_i)$$

But still, usually the log likelihood l above cannot be solved by brutal force. Because the total amount of multiplication is too large for us to do iterations. There are two algorithm to reduce the calculation. One is Hierarchical Softmax and the other one is Negative Sampling.

I. Hierarchical Softmax

For the i-th word in the sentence, if we split all words into two Groups, and insert the i-th word into Group1, the likelihood function can be transformed into:

$$P(w|C) = P(w|G_1, C)P(G_1|C)$$

Now we only need to solve $P(w|G_1, C)$ and $P(G_1|C)$. Since w is in Group1, then:

$$P(w|G_1, C) = \frac{e^{-E(w,C)}}{\sum_{all\ w_v\ in\ G_1} e^{-E(w_v,C)}}$$

And

$$P(G_1|C) = \frac{e^{-E(G_1,C)}}{e^{-E(G_1,C)} + e^{-E(G_2,C)}}$$
$$= \frac{1}{1 + e^{-E(G_2 - G_1,C)}}$$

So that

$$P(w|C) = \frac{e^{-E(w,C)}}{\sum_{all\ w_v\ in\ G_1} e^{-E(w_v,C)}} * \frac{1}{1 + e^{-E(G_2 - G_1,C)}}$$
$$= \frac{e^{-E(w,C)}}{\sum_{all\ w_v\ in\ G_1} e^{-E(w_v,C)}} * \sigma(n_1, C)$$

Where $n_1 = G_2 - G_1$

It is obvious that we can repeat this procedure until there is only word w in the last group, which gives us:

$$P(w|C) = \prod_{k=1}^{K} (\sigma(n_k, C)^{1-d_k} + (1 - \sigma(n_k, C))^{d_k})$$

d_k is the coefficient vector at the k-th knot of the word in the words tree. Huffman Tree is used here to build up the words tree, since on average it can get to any words with the least steps.

Since the original form of energy function can be regarded as the softmax function of N-word classification problem, now we have simplified the problem from n-classification into log_2^N-classification by using this hierarchical structure.

Gradient of log-likelihood function under hierarchical structure is just adding up gradients of all components, so that we can use SGD and iterations to make word vectors converge.

II. Negative Sampling

Negative Sampling is another way to reduce the calculation. We define the negative log likelihood of word probability function as objective, which is:

$$J = -log \frac{e^{-E(w,C)}}{\sum_{v=1}^{V} e^{-E(w_v,C)}}$$

$$= -w * C + log \sum_{v=1}^{V} e^{-w_v * C}$$

The gradient of above equation w.r.t. parameter θ can be shown in the following form:

$$\nabla_\theta J = \nabla_\theta(-w * C) + \nabla_\theta log \sum_{v=1}^{V} e^{w_v * C}$$

$$= \nabla_\theta(-w * C) + \frac{1}{\sum_{v=1}^{V} e^{-w_v * C}} \nabla_\theta \sum_{v=1}^{V} e^{w_v * C}$$

$$= \nabla_\theta(-w * C) + \frac{1}{\sum_{v=1}^{V} e^{-w_v * C}} \sum_{v=1}^{V} \nabla_\theta e^{w_v * C}$$

$$= \nabla_\theta(-w * C) + \sum_{v=1}^{V} \frac{\nabla_\theta e^{-w_v * C}}{\sum_{v=1}^{V} e^{-w_v * C}} * \nabla_\theta(w_v * C)$$

We can see that $\frac{\nabla_\theta e^{-w_v * C}}{\sum_{v=1}^{V} e^{-w_v * C}}$ is exactly the softmax probability $P(w_v|C)$. Bengio et al. noticed this and they treat the gradient as the sum of gradient of correct classified sample and expectation of gradients of wrongly classified samples, which reforms the above formula into:

$$\nabla_\theta J = \nabla_\theta(-w * C) + E_{w_v \sim P} * \nabla_\theta(w_v * C)$$

Where $w_v \sim P$ and P is the probability distribution that w_v will be chosen.

Obviously, the chosen of P will greatly influence the final result. In practice, usually we use the natural frequency of words in training text as P, which simplifies the model quite a lot and has proven to be effective in word2vec training.

7. Word2vec on Dream of the Red Chamber

As stated in 2., we split the text by paragraph. Each paragraph is transformed into a word list. We use genism, a famous and efficient package for NLP, to train the word2vec matrix. The training process is very quick and can be done within several seconds.

We test the combinations of hierarchical softmax/negative sampling with Skip-gram/CBOW, which give us 4 combinations in total. In each text we check the word vector of key characters and use clusters to rebuild their relationships so that we can compare them with our own knowledge about the famous novel. All four training methods give almost the same result, we keep the default model result in the Project.ipynb notebook.

Reference

[1] Xin Rong, *Word2vec Parameter Learning Explained*

[2] Frederic Morin and Yoshua Bengio, *Hierarchical probabilistic neural network language model*

[3] Y. Bengio, R. Ducharme, P. Vincent, *A neural probabilistic language model*

[4] Mikolov, *Distributed Representations of Sentences and Documents*

[5] Mikolov, *Efficient estimation of word representations in vector space*

[6] Yoav Goldberg, *Word2vec Explained- Deriving Mikolov et al.'s Negative-Sampling Word-Embedding Method*