**Math in a Nutshell #MiaN**

CONTINUOUS BAG OF WORDS

***Disclaimer***

As suggested in the title, this note explains the mathematical fundamentals without considering any optimization techniques such as regularization, negative sampling or hierarchical softmax. They would require modifications in the formulas but the core ideas remain.

This note does not aim to explain what CBOW (or even word embedding) is all about, but instead I do expect readers to be familiar with these concepts. I also assume background knowledge in calculus, linear algebra, machine learning and neural network.

At any point will you have trouble comprehending the concepts, feel free to pause and go through the following materials, which contribute to deepening my understanding. It is totally normal to go back and forth among these papers / blogs.

**Reference**

(Original paper)

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Wang, B., Wang, A., Chen, F., Wang, Y., & Kuo, C. C. J. (2019). Evaluating word embedding models: Methods and experimental results. *APSIPA transactions on signal and information processing*, *8*.

Rong, X. (2014). word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738*.

Lecture Notes 1 – Stanford CS 224D: Deep Learning for NLP

<https://cs224d.stanford.edu/lecture_notes/LectureNotes1.pdf>

|  |
| --- |
|  |
|  |

**Data**

A corpus of text.

If you have a dataset storing multiple documents, concatenate them into a giant corpus, but don’t mess up the order as the model takes it into account.

All the words in the corpus form a vocabulary.

**Notations**

Vocabulary size:

Embedding (or hidden layer) size:

Window size:

Learning rate:

A word at jth position in vocabulary:

is the kth element of any vector size (N x 1) with k: 1 🡪

is the ith element of any vector size (n x 1) with i: 1 🡪

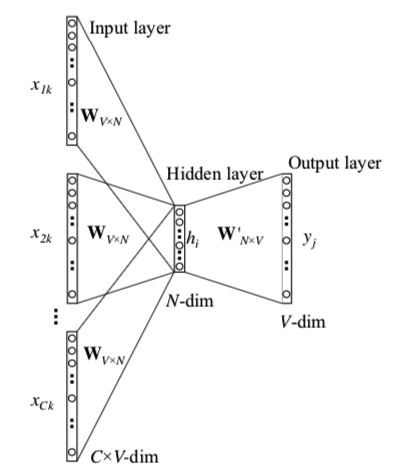
**Goal**

To predict a center word wj given its context words

For a given center word , the number of its context words (both sides) is

Throughout the optimization process, we also aim to get the vector representations of each word in vocabulary.

**A stochastic network of CBOW model for a single center word**



**Inputs / Input layer**

Each word is represented by a one-hot vector size (N x 1) with jth unit = 1. The word has context words, each of which is also represented by a one-hot vector size (N x 1) with its corresponding index = 1\*\*\*

**Weights**

* Input weights V size (n x N)
* Output weights U size (N x n)

Matrix U contains numeric representations of all N words in vocabulary.

**Outputs / Output layer**

One-hot vector representing actual center word j to be predicted size (N x 1) with jth unit = 1.

We also denote as the predicted (probability) center word vector. Values of its elements are between 0 and 1. The predicted word is the one at which the probability is the highest.

**\*\*\*Input vectors**

As the center word j has multiple context words, I present two ways to look at how the input vector is derived – one from the original paper and the other from my implementation.

*Original paper version*

1. Get embedded word vectors for each context word, size (N x 1) each
2. Average these vectors to get hidden layer vector size (n x 1)

*My version*

1. Average context vectors to get size (N x 1)
2. Pass this vector through the network as in single context model

**In general, the model goes as follow**

|  |
| --- |
| **🡪 🡪** |

🡪

With , is the kth element of vector and

Softmax is used to make sure values represent probabilities (between 0 and 1).

**Loss function / Cross entropy**

Recall that is a one-hot vector with only jth unit = 1. Hence,

Note that L is a scalar.

**Optimization**

The objective is to find the optimal set of weights that minimize the loss function, which basically measures the discrepancy between the actual and predicted output – the error. The lower the cost, the closer the predicted output approximates the actual.

**Backward Propagation**

***Update weights for hidden 🡪 output weights U***

For each component of matrix U, we update

(2) shows that changes in affect L through only.

By the Chain Rule,

We have

Since L also depends on (jth component of vector )

which is equivalent to

with the kth component of actual ouput vector since is one-hot.

Also, through (2) we know that . So

***Update weights for input 🡪 hidden weights V***

For each component of matrix U, we update

Similarly, (1) shows that changes in affects only. So,

It can be easily derived that

From (2), we know that changes in affects by affecting all components of Thus changes in with respect to is the summation of changes in all with from 1 to .

After all, we have

**As for implementation,** we wish to update all the weights at once through matrix manipulation. The updates for each element of and canbe translated to

with and is the transpose of

and

with is the transpose of and is the transpose of