Notes of Document Embedding

Kaiwen Dong

Department of Statistics, University of Illinois at Urbana-Champaign, Champaign, Illinois, 61801, USA, kaiwend2@illinois.edu

Kai Lu

Department of Statistics, University of Illinois at Urbana-Champaign, Champaign, Illinois, 61801, USA, kailu2@illinois.edu

Abstract

When we deal with nature language processing in machine learning, it is a common problem to find a way to represent the words and sentences in the document. There are many approaches to represent them. One popular and powerful method is to transfer a word or a sentence into a fixed-length continuous feature vector, named word2vec. Inspired by the word2vec method, there is a way to embed the document. The information of a sentence can be represented by a feature vector using document embedding technique. Once the document is transferred into a continuous vector, one can feed these vectors to the conventional machine learning methods. The method is introduced by Quoc Le and Tomas Mikolov, and this paper is an implementation and experiment of this method.

KEYWORDS

Document embedding, word embedding, document classification

Introduction

In nature language processing field, it is always crucial to represent words or sentences as an object which can be handled and processed by the traditional machine learning method. Using vectors to represent them is a common way to achieve it. Bag-of-words (Harris, 1954) is a classical method among them. However, the disadvantage of the method is also obvious. The sparsity of the representation vector makes it high-dimensional. When applying the machine learning method on it, the high dimension harms both the performance and efficiency of the language model. Also, the information about the order of the words is lost, which leads to a confusion about the meaning of a sentence, especially it expresses irony. Bag-of-n-grams (Harris, 1954) is an improvement of the method, but it has higher dimension and sparser data representation.

There are many other word2vec methods and the most popular and successful one might be Skip-gram model, which is introduced by Mikolov et al. (2014) The architecture of the Skip-gram model is shown in Figure 1. The task of the method is to get a collection of fixed-length continuous vectors corresponding to words in the vocabulary. Each vector can be regarded as a feature vector representing the word. In the figure, the input is only one target word and the output is the words surrounding the target word. Both input and output are encoded as a one-hot vector. The projection matrix will be updated while training the model. And the projection matrix gives us the feature vectors.

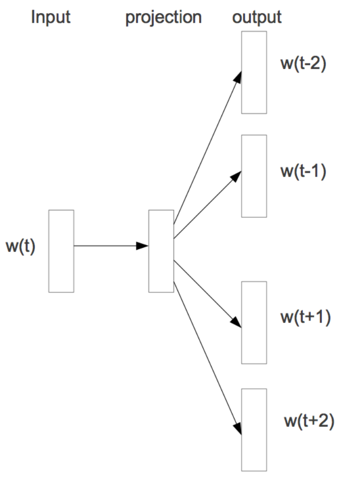


Figure 1. The framework of Skip-gram model.

The distributional hypothesis of Skip-gram is that a word ought to be able to predict its context. After the training step is completed, we will get a feature vector representing a corresponding word. The similar words will have the feature vectors close to each other. The vector also shows some parallel property. If $v(word)$ means the vector of the "word", then $v(\text{"king"}) - v(\text{"}man\text{"}) \approx v(\text{"}queen\text{"}) - v(\text{"}woman\text{"})$
The word will distribute in the space based on its meaning and information.

There is also another method training such model, called Continuous Bag-of-Words, which is similar to Skip-gram model but more like an inversed version. It is illustrated in Figure 2.

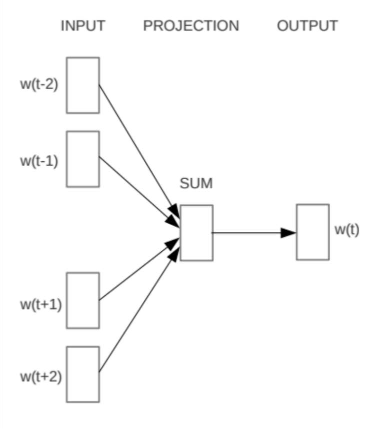


Figure 2. The framework of Continuous Bag-of-Words model.

Opposite to Skip-gram model, the input is the context, the words surrounding the target words, while the output is just the target word. The training process is similar. The distributional hypothesis of CBOW is that a context ought to be able to predict its missing word. The performance is as good as Skip-gram model in general, but it also loses the order of word, making it a little bit worse than Skip-gram model.

Inspired by these word2vec model, Le and Mikolov (2014) introduced a method to represent sentence, paragraph or document as a fixed-length feature vector. The method is similar to word2vec but the task now is to find the document embedding matrix. Once we get the document embedding vector, we can feed these vectors to the conventional machine learning method to do classification or clustering.

The cutting-edge technique we found in text categorization field is the semi-supervised convolutional neural networks (Rie and Tong, 2015), which achieves 6.51% error rate on IMDB dataset. In the document embedding method, Le and Mikolov (2014) claim their model could reach 7.42% error rate.

Our research will implement the document embedding method and experiment different hyperparameters tuning to reproduce the state-of-art results claimed in the original paper. We will choose another dataset to train our model to discover more properties of the document embedding in detail.

Algorithm

The algorithm includes two parts: training the document embedding and feeding them to a neural network. The first part is crucial, which is the main algorithm we want to implement. The second part is just a validation step to verify our document embedding vectors have the property to do classification.

Embedding

Our main task is to get the document embedding vectors. But in fact, the feature matrices of words and documents are trained simultaneously, that is we also need to train word embedding part if we want the document embedding.

We will first introduce the word embedding part, which is the inspiration and guide to train the document embedding.

Word Embedding

There are two famous word embedding methods just as the Figure 1 and Figure 2 illustrate. We will implement both models later, but now we will first introduce the CBOW model which is more similar to our document embedding method.

After cleaning the dataset, like removing punctuation marks and changing all characters to lower case, we can split each document into words by the space or tab. We need to build a vocabulary, which has a fixed-size. If our vocabulary size is $|V|$, the size of our word embedding matrix will be $|V| * M,$ where M is the length of the word embedding feature vector. Because the vocabulary is fixed-size, we will keep the top $|V| - 1$ most frequent words in the dataset, and mark those less frequent words as a special token (We use "UNK" Unknown). In the whole dataset, we only have $|V|$ classes words now.

Finished the preparation, we can build our word embedding model now. In the model, we map each word to a specific row of the word embedding matrix. Because we apply the CBOW model, we use the surrounding words to predict the target word. If we use k as our skip window, there are $2k$ context words for each target word. (The left k words and right k words of the target word) For those context words, we can concatenate these feature vectors corresponding to the words (results in $|V| * 2k$ dimensional vector), or average these feature vectors (results in $|V|$ -dimensional vector). We multiply the feature vector by a weight matrix to get a $|V|$ - dimensional vector, which is the logit value for prediction. Then we can apply softmax classifier to predict the probability of each class (word).

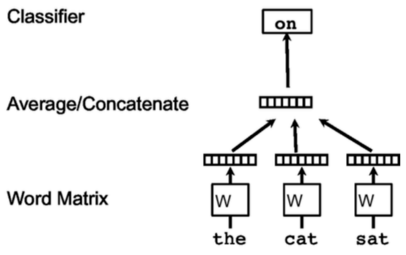


Figure 3. The model of CBOW.

Softmax

In the softmax step, one can find that the computation is quite heavy when calculating the probability of each word, especially when the vocabulary size is large (it is common to set a large size vocabulary). In practice, there are many approaches to speed up the calculation at the cost of little bit worse performance. We have learned Negative Sampling (Mikolov et al., 2012) in class. Negative Sampling will only update part of the weight matrix. The rows of the weight matrix are sampled followed a multinomial distribution. The probability of each class is proportional to the frequency of the word.

We also provide another speed up method called noise contrastive estimation (NCE) (Mnih and Teh, 2012). It is implemented in the TensorFlow tutorial, and it definitely increase the efficiency of computation. One can choose either of the speed up method. During my training, we use Negative Sampling.

In the original paper, the author noticed that they would use hierarchical softmax (Morin & Bengio, 2005; Mnih & Hinton, 2008; Mikolov et al., 2013c) as the speed up method. In TensorFlow, there is no quick way to implement the method. If one wants to use it, you have to build the softmax layer from scratch (without TensorFlow). We recommend readers who are capable of building Huffman tree and encoding the weight matrix implementing the hierarchical one, which might be more powerful than our method.

Document Embedding

Document embedding step is similar to the word embedding. We simply add one more input into our model: the document embedding vector. Compared to the CBOW, the document embedding vector is also corresponding to the specific row of the document embedding matrix. In the same document, the document vector will be the same row of the document embedding matrix. The framework can be shown in the Figure 4.

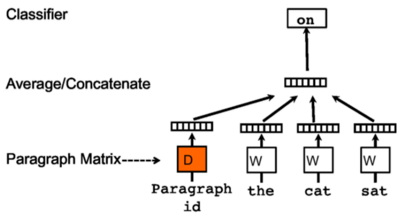


Figure 4. The framework of document embedding

The whole training progress is almost the same as word embedding. One thing to notice is that you can concatenate the document embedding vector and the word embedding vectors. In the original paper, they also suggest one can use averaged vector. However, the average method shouldn't work if the size of word feature vector and of document feature vector is different. We use concatenate in our implementation.

We also find that in the original paper, the document embedding vector is trained simultaneously along with training word embedding vector. We have more discussion about these part in the later section.

Back-propagation

We train the model and update the weight matrix using back-propagation method. We generate a batch of data to feed into the model, use the stochastic gradient descent to update each parameter in the weight matrix. There are two methods we provide in our model: the conventional gradient descent and the Adam optimizer. The conventional gradient descent needs a learning rate to control the training speed. The Adam stochastic gradient descent controls the learning rate based on the convergence speed of the training step. Even though the Adam optimizer is more computation consuming, it performs better to minimize the loss function.

```
Summary
Step 1: Clean the dataset. Remove the punctuation marks and change characters from uppercase to lowercase
Step 2: Build the vocabulary. Keep the frequent words in the vocabulary. Mark all other less frequent word as token "UNK"
Step 3: Train the model. In each step, feed the model a batch of data and use back-propagation to update both the embedding matrix (word and document) and the weight matrix of the softmax layer.
End
```

Neural Network: Evaluation step

In this part, we build a single hidden layer neural network to classify the document. The structure is as below.

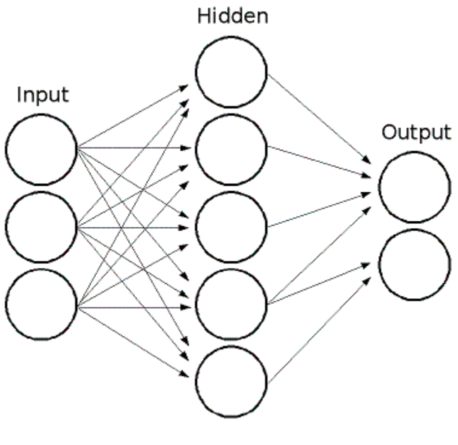


Figure 5. One layer neural network.

Input of the neural network is the document embedding vectors. The output is the document labels. The neural network uses the document feature vectors to predict the label. In the neural network, there are only two hyper parameter we can tune: the number of hidden units and the activation function. In general, we apply an activation function after the output of each layer, to make the network nonlinear. This step is an optional step for our document embedding project. It is just an evaluation step, and a supervised learning. One can also use unsupervised learning method like k-means to classify the document. The choice of the machine learning method to do classification and clustering is out of the range of the project.

Experiment

We will performance sentiment analysis on the Yelp dataset. The dataset contains 560,000 reviews about the restaurant or café. Each document is labelled as positive or negative. If we have dataset without any labeled document, we can first apply document embedding to these documents to get the document vectors, then use unsupervised clustering method to classify these documents. But in this project, we simply use part of the labeled data to train our model, and part of the labeled data to test the result.

Yelp Sentiment dataset

The whole dataset is quite huge to train all of them. There are totally 560,000 reviews in the dataset. Due to the limitation of our computation resource, we plan to use only 100,000 reviews to train our model.

Build the vocabulary

We first build the vocabulary of the dataset. And here is the top 10 frequent words and their frequency:

Word	UNK	the	and	i	to
Count	86195	257371	163000	138019	127363

Word	a	was	of	it	for
Count	127041	89549	75303	65501	59162

One advantage of our method is that we don't need to remove the stop words in our dataset, unlike many other NLP method.

After building the vocabulary, we can assign an integer token to represent each word.

Train the model

We follow the algorithm in the former section to train our model. During the training process, we find many interesting properties of the document embedding method and also modify our algorithm based on what we found in the process.

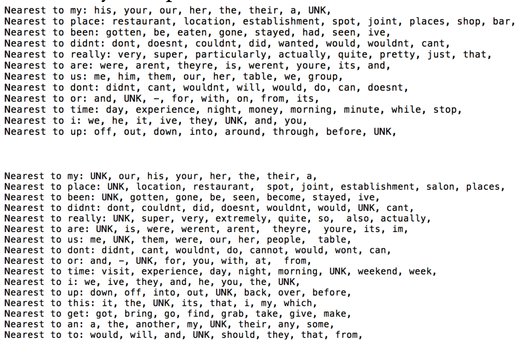
Too much memory to use during training

At first, we tried to train the model based on the whole dataset. Even though the size of word embedding matrix in under control (through two hyper parameters: word embedding size and vocabulary size), we can't control the size of the document embedding matrix (we can only control document embedding size). The size of document embedding matrix is $|D| * m$, where $|D|$ is the number of documents and m is the document embedding size. In the original paper, they use 400 as the document embedding size. Compared to the original paper, we plan to use 50 as the document embedding size to binary classification (positive & negative). We can assume that the higher dimension, the more information the document embedding vector carries, if we have train the model through enough epochs.

Even though we drop to 1/8 embedding size, the memory usage is still quite huge. We have to train only part of the dataset, 100,000 in our model. Our computer can just handle the out of memory problem to shrink the dataset and embedding size to current setting.

Word embedding shrinkage

As we train our document embedding matrix, the word embedding matrix is also being updated. In the early stage of our training (in the 2nd epoch), the words are displayed quite well in the space. We choose some words in our vocabulary, and print the closest words to them.



```
Nearest to my: his, your, our, her, the, their, a, UNK,
Nearest to place: restaurant, location, establishment, spot, joint, places, shop, bar,
Nearest to been: gotten, be, eaten, gone, stayed, had, seen, ive,
Nearest to didnt: dont, doesnt, couldnt, did, wanted, would, wouldnt, cant,
Nearest to really: very, super, particularly, actually, quite, pretty, just, that,
Nearest to are: were, arent, theyre, is, werent, youre, its, and,
Nearest to us: me, him, them, our, her, table, we, group,
Nearest to dont: didnt, cant, wouldnt, will, would, do, can, doesnt,
Nearest to or: and, UNK, -, for, with, on, from, its,
Nearest to time: day, experience, night, money, morning, minute, while, stop,
Nearest to i: we, he, it, ive, they, UNK, and, you,
Nearest to up: off, out, down, into, around, through, before, UNK,


Nearest to my: UNK, our, his, your, her, the, their, a,
Nearest to place: UNK, location, restaurant,  spot, joint, establishment, salon, places,
Nearest to been: UNK, gotten, gone, be, seen, become, stayed, ive,
Nearest to didnt: dont, couldnt, did, doesnt, wouldnt, would, UNK, cant,
Nearest to really: UNK, super, very, extremely, quite, so,  also, actually,
Nearest to are: UNK, is, were, werent, arent,  theyre,  youre, its, im,
Nearest to us: me, UNK, them, were, our, her, people,  table,
Nearest to dont: didnt, cant, wouldnt, do, cannot, would, wont, can,
Nearest to or: and, -, UNK, for, you, with, at,  from,
Nearest to time: visit, experience, day, night, morning, UNK, weekend, week,
Nearest to i: we, ive, they, and, he, you, the, UNK,
Nearest to up: down, off, into, out, UNK, back, over, before,
Nearest to this: it, the, UNK, its, that, i, my, which,
Nearest to get: got, bring, go, find, grab, take, give, make,
Nearest to an: a, the, another, my, UNK, their, any, some,
Nearest to to: would, will, and, UNK, should, they, that, from,
```

Figure 6. The closest words to some of the words. The former one is the early stage result, the latter one is the result from the model after trained 4 epochs.

However, the distribution of the words tends to shrink together after training several epochs. As we can see, all the words tend to be close to "UNK" (the token of all less frequent words) and "the" the most frequent word. We think there are two reasons for it.

Skip window size

We set 4 as our skip window size at first, which is suggested by the original paper. However, after several training epochs, the words tend to be close to each other no matter they have similar meanings or not. We find that if we set 4 as the skip window size, the probability of any two words appearing in the same skip window will increase a lot.

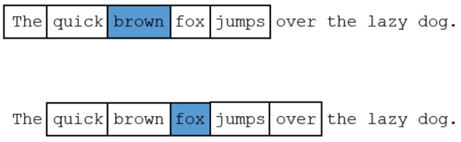

The quick brown fox jumps over the lazy dog.
The quick brown fox jumps over the lazy dog.

Figure 7. The skip window illustration.

It is easy to see that the larger the skip window is, the more likely two unrelated words are to appear in the same window. Thus, all words will tend to come closer to each other and the distribution of different words is collapsed. To fix it, we abandoned the training result from large skip window, and trained the model from beginning and set the skip window size as 2. It seems we wastes nearly 20 hours training time due to inappropriate skip window setting.

Overfitting the model

After we decreased the skip window size, the words tend to shrink again when we trained the 100,000 documents several epochs. We used to think we should stop training process in advance. However, when we fed the document embedding vectors into a supervised classifier to test it performance, we found that only 4 epochs are not enough to make the document embedding vectors distributed evenly in the space. The train error rate is about 40% which is much higher than the estimation.

We found that the training process of word embedding is much faster than the process of document embedding, because the vocabulary size is smaller than the number of documents. In the original paper, it claims that we should train the word embedding and document embedding simultaneously. However, we became doubted about the training process.

Something wrong in the original paper?

We believed that we should end the training process of word embedding in case of words shrinkage, but we still need to train the document embedding further to get better performance.

Our first attempt to fix it is to make the learning rate between word embedding and document embedding different, lower learning rate for word embedding but higher learning rate for document embedding. However, we didn't find a way to make it with TensorFlow package.

Then we planned to fix the word embedding matrix during training process, only to train the document embedding matrix. It is against the algorithm stated in the original paper, but we had to find a way to work it out.

It seems work!

After fixing the word embedding matrix, there are three advantages coming out: 1. The speed of training is faster because we only need to update the document embedding matrix; 2. Avoid the overfitting problem. The word embedding matrix is fixed so it won't change during the training. 3. We can train the document embedding much more times. We don't need to care about whether to stop the training in advance.

Result

Training Time

We trained the model for nearly 80 hours in total. We first spent nearly 20 hours training the model with skip window size 4. It proves to be a waste of time. Then we set the skip window size to only 2 and limited the number of document to be trained. We spent nearly 60 hours to train it. During the first 15 hours, we trained the word embedding matrix and document embedding matrix simultaneously. We only trained the whole 100,000 documents 1 epoch for 15 hours. Then we fixed the word embedding matrix to train the document embedding matrix. The dataset was trained for 10 epochs for these 45 hours. We only have one NVIDIA GTX 1060 GPU to do the computation. We can get better result if we have enough computation resource.

Word Embedding

We already have a look at the nearest words in the former section. Here, we have a plot to project the word embedding vectors to the 2-dimension figure (using Principal Component Analysis).

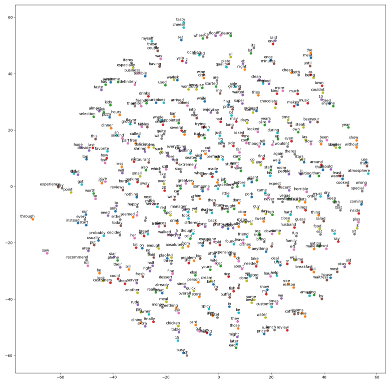


Figure 8. Words distribution

The plot of words distribution might seem not quite well. Some words with similar meanings are not distributed close. It is because the distance shown in the 2-d plot is not the same as the true distance in the M-dimensional space, where M is the length of word feature vector. However, the plot is still quite well in terms of words distribution. The words are distributed in the space based on their meanings. We can assert that our word embedding model is efficient according to the former nearest words list.

Document Embedding

There are 100,000 documents in total whose feature vector is of size 50. We fed these vectors into a single layer neural network. Our activation function is $tanh$ and the number of units in the hidden layer is 20. We divide the 100,000 documents into two parts: first 90,000 as the train data and the last 10,000 as the test data.

After training the neural network classifier on 90,000 documents, we saved the parameters in the network. Then we restore the parameters and fed in the last 10,000 documents.

	Error Rate
Train	14.8 %
Test	19.8 %

We found that in terms of the Yelp sentiment dataset, our error rate is not quite well. Some methods could achieve at most 96.6% accuracy (this is the state-of-art result right now, using region based CNN). We thought that there is some weakness in our model:

1. In the original paper, they didn't test on this Yelp dataset. The performance of model often varies a lot among different datasets.
2. We should train the model with more epochs. We have only trained the model for 10 epochs. In the original paper, they didn't point out the epoch we need to train. Based on our experience, we think 50 epochs are acceptable in such a large dataset.
3. We can enlarge the size of the document embedding vector. We only use 50 as the size. The longer the feature vector is, the more information the vector carries. In the paper, they use 400 as the size, which is 8 times as our feature vector. A longer feature vector definitely outperforms the shorter one, if only we have enough training. And a longer feature vector also requires more memory and computation resource, which we still lack of.

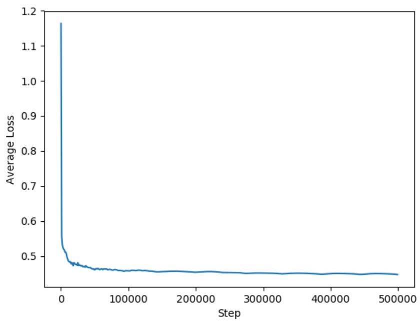


Figure 9. The plot of average loss during training the classifier.

Discussion

When I discussed the project with TA, we learned that in general, Skip-gram model performs better than the CBOW model. In our project, we used the CBOW model, which is suggested in the paper. However, the result of the model is not as good as we imagine. We don't have enough time to implement another Skip-gram model to verify it (Mainly short of computation time). We recommend the reader to use Skip-gram model instead.

We only test the method on a binary classification task. One can also apply the method to do multiple class classification. You just need to enlarge the document embedding size and train enough epoch. Then, you can feed the document embedding vectors into the conventional machine learning method, no matter unsupervised one, like k-means or supervised one, like logistic regression.

The document embedding method is an unsupervised method, which is also able to work when the documents have no label. The training process of document embedding doesn't involve the label of document. This makes it a powerful method to embed document as a vector.

References

Harris, Zellig. Distributional structure. Word, 1954.

Quoc V. Le and Tomas Mikolov, Distributed Representations of Sentences and Documents. CoRR, abs/1405.4053, 2014

Johnson Rie and Zhang Tong, Semi-supervised Convolutional Neural Networks for Text Categorization via Region Embedding, 2015

Mikolov, Tomas, Sutskever, Ilya, Chen, Kai, Corrado, Greg, and Dean, Jeffrey. Distributed representations of phrases and their compositionality. In Advances on Neu- ral Information Processing Systems, 2013c.

Mnih, Andriy and Whye Teh, Yee, A Fast and Simple Algorithm for Training Neural Probabilistic Language Models, In Proceedings of the 29th International Conference on Machine Learning, pages 1751-1758, 2012

Morin, Frederic and Bengio, Yoshua. Hierarchical proba- bilistic neural network language model. In Proceedings of the International Workshop on Artificial Intelligence and Statistics, pp. 246–252, 2005.

Mnih, Andriy and Hinton, Geoffrey E. A scalable hi- erarchical distributed language model. In Advances in Neural Information Processing Systems, pp. 1081–1088, 2008.

Appendix

Code is stored at github: https://github.com/Barcavin/document_embedding