GEORGETOWN UNIVERSITY

ANALYTICS 561 PROJECT

Word Embeddings using Neural Networks

Min Xiao
Hanjing Wang
Pengsen Li
Jiajia Liu

This dataset contains messages from 20 Newsgroups, gathered from 11,270 total articles. We used the Bydate version and take each group as a separate version. In the project, we first represent the skip gram model used to build word embeddings. Then, we apply negative sampling to generate a false corpus when training the model. After the algorithm deduction, we train the word vector space while using tensorboard to monitor the training loss. To visualize the embedding results, we use tSNE to show the distance of words. Finally, in order to use our embedding results on text classification, a simple regression is used and 60% accuracy is achieved for 20 balanced classes of text data.

December 9, 2017

1 Introduction of word embeddings

1.1 Word Embeddings in NLP

Motivation for word embeddings

Natural language processing (NLP) is a field of computer science focused on training computers to understand human languages. There is a variety of current research into NLP systems, with many applications thereof. Common tasks include spell checking, keywords searches, information parsing, machine translation, and semantic analysis, among others. Since many NLP algorithms can only deal with numeric data and not raw text, the most fundamental challenge for various NLP tasks is word representation. More precisely, the text should be converted into numeric values for further analysis. Early research used atomic units to represent words; that is,indexes represent words in a vocabulary. This simple technique turns out to be useful in completing some tasks, but is limited in many other situations. A more recently developed method maps text words to vectors trained by machine learning techniques, i.e., word embeddings in NLP. Vectors are powerful tools because they allow us to compute word similarities using distance measures, such as cosine and Euclidean, to name a few.

Singular Valued Decomposition Based and Iterative Based Word Embeddings

Vector space models (VSMs) represent embedded words in continuous vector space where semantically similar words are mapped to nearby points. In linguistics, VSMs aim to quantify and categorize semantic similarities between linguistic items based on their distributional properties in the large corpus. The simplest representation is one-hot vector with the vocabulary size as its length. However, in the English language there are 13 million tokens, and so this single hot vector method would fall prey to the curse of dimensionality. Because there are some linguistically-related objects in all words, vector spaces with lower but more dense dimensions are better suited. Some commonly used methods for word embeddings include Singular Valued Decomposition(SVD) and iterative based methods. SVD does single value decomposition on the co-occurrence matrix, but will not be covered in this project. Instead, we will explore iterative method skip-gram models' ability to produce effective word vectors.

1.2 Literature review

Vector space models have been in use since the 1990s. Early research developed many models to estimate continuous representation of words, such as Latent Semantic Analysis (LSA) and Latent Dirichlet Allocation (LDA). In 2003, [BDVJ03] invented the term word embeddings and trained the object in a neural language model. They show that the word embedding is a distributed representation of words and greatly reduces the data sparsity problem. [HS06] and [EBC$^+$10] show that neural networks can converge to a better local minima with a suitable unsupervised pre-training procedure. The most significant influence in recent word embeddings research comes from [MCCD13], which shows that pre-trained word embeddings can detect syntactic and semantic regularities. A subsequent article [MSC$^+$13] uses several methods to improve efficiency and the quality of words and phrases. In this project, we will reproduce the core content of these two just-mentioned papers.

1.3 Project Overview

Our data set is the 20Newsgroups[1]. This dataset contains messages from 20 Newsgroups, gathered from 11,270 total articles. The data is organized into 20 different newsgroups, each corresponding to a different topic. Some of the newsgroups are closely related to each other, while others are

[1] http://qwone.com/ jason/20Newsgroups/

unrelated. We used the Bydate version and take each group as a separate version.
In the project, we first represent the skip gram model used to build word embeddings. Then, to reduce the computational complexity, we apply negative sampling (NEG) to generate a false corpus when training the model. There are some papers about word embeddings - for example, [MCCD13] - but the mathematics is not properly presented and the algorithm description is obscure. We reformat the model representation and give a clear and detailed explanation of the process. After the algorithm deduction, we train the word vector space while using tensorboard to monitor the training loss. To visualize the embedding results, we use tSNE[2] to show the distance of words. Finally, in order to use our embedding results on text classification, a simple regression is used and 60% accuracy is achieved for 20 balanced classes of text data.

2 Basic algorithms for word embeddings

2.1 Observations generated from raw text

We have message text and can generate training sets from the raw text. A detailed description can be found in section 4. To make trainable datasets from the raw text, we need to generate observations of central and context words. Central words are words of the most interes. Given a central word, the corresponding context words are the set of C surrounding words. Hyperparameter window size is defined to be the distance from the central word to the furthest context word(s). For example, we have a text sequence ["the quick fox jumped over the lazy dog"]. If we pick "fox" to be the central word, and set the window size as 2, context words for "fox" are: "the", "quick", "jumped", and "over". So we would have the paired word observations: "fox","the", "fox","quick", "fox","jumped", and "fox","over".
More generally, We need a model that aims to make a prediction from every center word wt to its context words in terms of word vectors p(context|wt), which has a loss function. This is the main idea of word2vec, which has two main algorithms for training the model – continuous bag of words (CBOW) and skip-gram. Our project will focus on skip gram.

2.2 The Skip-gram Model: Neural Networks in Word Embeddings

2.2.1 Network Structure

The skip-gram algorithm builds a shallow neural network model for word2vec. The one-hot vectors of length V for all central words are the input, where V represents the size of the vocabulary. For each central word, w_i, the output is the word vectors for words in w_i's context $\{w_{O,1}, \cdots, w_{O,C}\}$, which is defined using a word window of size C. The graph**??** is a description of skip-gram.
In the neural network, there are three layers. The input layer is the one-hot vector x relative to the input word of V-dim. To compute the hidden layer, there is a weight matrix $\mathbf{W}$ of shape V by d, whose i^{th} row represents the weights corresponding to the i^{th} word in the vocabulary. Mathematically, the hidden layer could be defined as

$$\mathbf{h_i} = \mathbf{x}^T\mathbf{W} = \mathbf{W}_{(k,\cdot)} := \mathbf{V}_{\mathbf{w_i}}$$

which will equal to the k^{th} row of $\mathbf{W}$.
Then, the network needs another associated $N \times V$ output matrix $\mathbf{W}'$. In the output layer,

$$\mathbf{U} = \mathbf{h_i}\mathbf{W}'$$

[2]https://lvdmaaten.github.io/tsne/

Also, noting that the input to each of the $C \times V$ output nodes is computed by the weighted sum of its input, the input to the j^{th} node of the c^{th} output word is

$$u_{c,j} = \mathbf{h}\mathbf{V}'_{\mathbf{w_j}}$$

where $\mathbf{V}'_{\mathbf{w_j}}$ represents the column vector of $\mathbf{W}'$ for word w_j.
Finally, noting that each output word shares the same weights, which means $u_{c,j} = u_j$, the softmax function is used to compute the output of the j^{th} node of the c^{th} output word.

$$P(w_{c,j} = w_{o,c}|w_i) = y_{c,j} = \frac{\exp(u_{c,j})}{\sum_{j'=1}^{V} \exp(u_{j'})}$$

The above equation calculates the probability that given the central word w_i, the output of the j^{th} node of the c^{th} output word is the context word $w_{o,c}$.

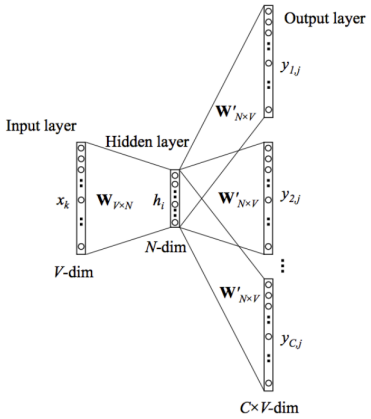

Figure 2.1: Skip Gram

2.2.2 Model Training with Backpropagation and Stochastic Gradient Descent

The skip-gram model learns the weights with backpropagation and stochastic gradient descent. Since we have computed the probability of each output context word given the central word, the loss function is the product of all such probabilities. Since all these probabilities are independent, we could have that

$$L = P(w_{O,1}, w_{O,2}, \cdots, w_{O,C}|w_i) = \Pi_{c=1}^{C} P(w_{O,c}|w_i) = \Pi_{c=1}^{C} \frac{\exp(u_{c,j_c^*})}{\sum_{j'=1}^{V} \exp(u_{j'})}$$

We need to minimize the loss function. Taking the negative log of the likelihood function,

$$E = -\log L = -\log \Pi_{c=1}^{C} \frac{\exp(u_{c,j_c^*})}{\sum_{j'=1}^{V} \exp(u_{j'})} = -\sum_{c=1}^{C} u_{j_c^*} + C \cdot \log \sum_{j'=1}^{V} \exp(u_{j'})$$

where j_c^* is the index of c^{th} output word. Then, we began to establish the update function for $\mathbf{W}, \mathbf{W}'$ using gradient decent, we can derive the derivative with respect to $\mathbf{W}, \mathbf{W}'$ using the chain rule.

$$\frac{\partial E}{\partial w'_{i,j}} = \sum_{c=1}^{C} \frac{\partial E}{\partial u_{c,j}} \cdot \frac{\partial u_{c,j}}{\partial w'_{i,j}} = \sum_{c=1}^{C} (y_{c,j} - t_{c,j}) \cdot h_i$$

Here,

$$t_{c,j} = \begin{cases} 1 & \text{the jth node of the cth output word is equal to 1} \\ 0 & \text{otherwise} \end{cases}$$

Also, we have

$$\begin{aligned} \frac{\partial E}{\partial w_{k,i}} = \frac{\partial E}{\partial h_i} \cdot \frac{\partial h_i}{\partial w_{k,i}} &= (\sum_{j=1}^{V} \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial h_i}) \frac{\partial h_i}{\partial w_{k,i}} \\ &= (\sum_{j=1}^{V} \sum_{c=1}^{C} (u_{c,j} - t_{c,j}) \cdot w'_{i,j}) \frac{\partial h_i}{\partial w_{k,i}} \\ &= \sum_{j=1}^{V} \sum_{c=1}^{C} (y_{c,j} - t_{c,j}) w'_{i,j} \cdot x_k \end{aligned}$$

Finally, the gradient decent update equation for $\mathbf{W}, \mathbf{W}'$ is in the following

$$\begin{aligned} w'^{(new)}_{i,j} &= w'^{(old)}_{i,j} - \eta \sum_{c=1}^{C} (y_{c,j} - t_{c,j}) \cdot h_i \\ w^{(new)}_{i,j} &= w^{(old)}_{i,j} - \eta \sum_{j=1}^{V} \sum_{c=1}^{C} (y_{c,j} - t_{c,j}) w'_{i,j} \cdot x_j \end{aligned} \tag{1}$$

After training the model, $\mathbf{W}$ is the word vectors space for the given vocabulary.

3 Optimization using Negative Sampling

Negative sampling is a practical technique to reduce computational complexity as well as improve the quality of word representation when implementing word embeddings.

3.1 Stochastic Gradient Descent

In an optimization program, the target function is $J(\theta)$, and we intend to mininize it respect to θ for a training data set X, Y. The iteration method of Full Batch Gradient Descent is

$$\theta' = \theta - \alpha \nabla_\theta E(J(\theta; X, Y))$$

where $(x^{(1)}, y^{(1)})$, $(x^{(2)}, y^{(2)})$, $\cdots$ $(x^{(M)}, y^{(M)})$ are the data for SGD. Compared to Full Batch Gradient Descent, SGD can indeed improve optimization performance for large-scale problems, and it will converge faster because SGD can run more examples within the available computation time. In our project, we have a very large data set, so we will use an algorithm based on SGD to reduce the computational complexity.

$$\theta' = \theta - \alpha \sum_{j=1}^{M} \frac{1}{M} \nabla_\theta J(\theta; x^{(i)}, y^{(i)})$$

where $(x^{(1)}, y^{(1)})$, $(x^{(2)}, y^{(2)})$, $\cdots$ $(x^{(M)}, y^{(M)})$ are the data for SGD. Compared to Full Batch Gradient Descent, SGD can indeed improve optimization performance for large-scale problems and converge faster because SGD can run more examples within the available computation time. In our project, we have a very large data set, we will use algorithm based on SGD to reduce the an computation complexity.

3.1.1 Principle of Negative Sampling

To train the model faster, Negative Sampling is used in the calculating process. Next we will show the generating process of the negative sample for a given a word, w_r. The figure 3.1 shows the mechanics of negative sampling.

We define D as the corpus data which contains word vectors(w), $count(w)$ as the frequency of w in D. For every $w \in D$,we also define:

$$\text{len}(w) = \frac{\text{count}(w)}{\sum_{\mu \in D} \text{count}(\mu)}, \qquad l_0 = 0, \quad l_k = \sum_{j=1}^{k} \text{len}(w_j), \quad k = 1, 2, \cdots, N$$

Then we divide the interval [0,1] into $I_i = (l_{i-1}, I_i], \quad i = 1, 2, \cdots, N$. Besides, we also divide the Interval [0,1] into M parts equally, where $M \in N^+ >> N$. In our sampling, we use $M = 10^8$.
Next, we build a mapping relation:

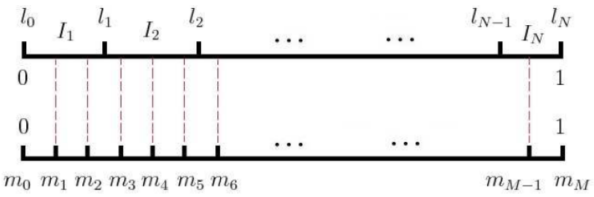

Figure 3.1: Negative Sampling

We pick a series of integers in M so that we can get a negative sample for w^i. In this process, if w^i itself is chosen, it will be ignored. Sampling process then continues to randomly choose other words. From the process above, we can generate the negative sample for w^i [MSC+13].

3.1.2 Negative Corpus Data and Modified Loss Function

Recall the original objective function(loss function) in section is:

$$E = -\sum_{c=1}^{C} u_{j_c^*} + C \cdot \log \sum_{j'=1}^{V} \exp(u_{j'})$$

As we discussed above, the large corpus data size means that skip-gram neural network has millions of weights (V), all of which would be updated slightly by each of our millions of training samples. Negative sampling will update the computing by having each training sample only modify a small percentage of the weights rather than all of them [3].
In this part, we definite a different objective function. Consider a word o, we denote by $P(D = 1|w_o)$ the probability that w_o is in the corpus data, while $P(D = 0|w_o)$ the probability that w_o is not in the corpus data.

$$P(D = 1|w_o) = \frac{1}{1 + \exp(-u_{w_o})}$$

$$P(D = 0|w_o) = 1 - \frac{1}{1 + \exp(-u_{w_o})} = \frac{1}{1 + \exp(u_{w_o})}$$

The motivation is to maximize the probability of a word being in the corpus data if it really is, and also to maximize the probability of a word not being in the corpus data if it indeed is not.

[3] https://towardsdatascience.com/word2vec-skip-gram-model-part-1-intuition-78614e4d6e0b

We assume the word occurrences are completely independent. Therefore, the loss function this time is:

$$\begin{aligned} E' &= -\log(P(w_{O,1}, w_{O,2}, \cdots, w_{O,C}|w_i)) \\ &= \log\{P(D=1|w_O) \prod_{w \in \widetilde{D}} P(D=0|w_O)\} \\ &= \sum_{w_O \in D} \log \frac{1}{1+\exp(-u_{w_O})} + \sum_{w_O \in \widetilde{D}} \log \frac{1}{1+\exp(u_{w_O})} \\ &= \sum_{j=1}^{C} \log \frac{1}{1+\exp(-u_{j_c^*})} + \sum_{k=1}^{K} \log \frac{1}{1+\exp(-u_{j'})} \end{aligned}$$

In the above formulation, $w_O = \{w_{O,1}, w_{O,2}, \cdots, w_{O,C}\}$, and $\{\widetilde{D} : w_k | k = 1, 2, \cdots, K\}$ is a negative sample respect to W_O. Actually, in order to make the best approximation, the process of negative sampling uses the Unigram Model raised to the power of $\frac{3}{4}$. In other words, there is a little change in the definition of $len(w)$:

$$\text{len}(w) = \frac{[\text{count}(w)]^{\frac{3}{4}}}{\sum_{\mu \in D}[\text{count}(\mu)]^{\frac{3}{4}}}$$

As we can see in E, the last term should do a summation V times. However, in E', the number of summation times is K, usually below 50, which is a significant decrease. Therefore, the computation is very efficient.

4 Experiments

4.1 Experiment Settings

- For the English text, we used the NLTK[4] package to tokenize and split the dataset into training(0.9) and test sets(0.1).
- In skip-gram models, we set the window size to 1, batch size to 100, and the number of negative samples in NEG to 50. For a single word vector, the embedding dimension is =100. In the architecture of the shallow neural network, the hidden layer size is $H = 100$. Cross-entropy is used as the loss function and SGD as the optimization algorithm. The model was built and trained using tensorflow, and the codes are adapted from tensorflow word2vec tutorial[5]. We used tSNE[6] to visualize the word vector spaces. Also Principal Component Analysis(PCA) was used in dimension reduction for better visualization.
- To test whether the embedding result if reasonable, the simple logistic regression will be used to do the text classification after word embeddings. Neural networks produce the vocabulary embeddings matrix W. For a single document, we can represent the document by average embedding vectors [LXLZ15] of all words in that document. Then we will have a vector (dimension=100) for each document. (The weighted_embeddings.csv is the weighted embeddings for each documents, in which rows are docs and columns are embed dimensions.) In the logistic regression, the 100 training features are the input while category for each document is the classification label. We use sklearn to implement the algorithm. The dataset will be split to train(0.9) and test(0.1) in sklearn commands. [7].

[4] http://www.nltk.org/
[5] https://www.tensorflow.org/tutorials/word2vec
[6] https://lvdmaaten.github.io/tsne/
[7] http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

4.2 Experiment Results and discussion

Sections 2 and 3 give a detailed description of how neural networks are used in word embeddings. The result is below.

- Monitoring training from the tensorboard
  The shallow neural network gives a good result for our dataset. In figure 4.2, the training loss decreases very quickly when using negative sampling and almost converges after 25,000 iterations. Without negative sampling, the model is not trainable because of our limited hardware. The NEG not only improves the efficiency but also gives better quality for word embeddings. After training, a word embeddings matrix (42634*100, check in attachment) is produced. The 100 dimension of matrix captures the semantic and syntactic regularities for the vocabulary. The dense matrix also solve the sparsity problem using one hot vector, making further analysis more efficient.

- Visualization for word embeddings
  The t-SNE visualization shows that words of similar functionality are more close to each other. Figures below are sub areas of the word vectors visualization using t-SNE; it measures the distance between words. Full plots can be found in the attachment. Clearly, words with the same functionality, such as common modal verbs, are close to each other. Similarly, numbers in the contexts are closer in the word vector space. Intuitively, this result captures the semantics. It should be a reasonable result.

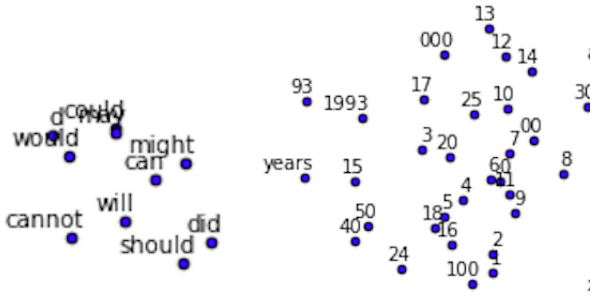

Figure 4.1: sub-area of tSNE visulization of word vectors

- Logistic regression using word embeddings The Logistic Regression (LR) classification accuracy is almost 60% in cross validation. Considering there are 20 classes, that is a fair result. This dataset is trained to have better results in a recent research paper [LXLZ15], in which more advanced methods like recurrent neural networks are used. Another factor may be that the dataset is not perfectly separable because there are similar groups, which is mentioned on the data source webpage. In future works, we can try to improve our models from the aspects listed above.

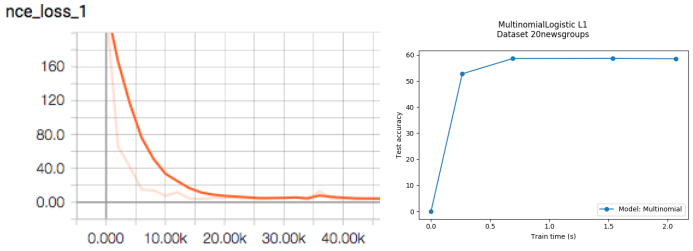

Figure 4.2: training loss(left) and classification accuracy(right)

References

[BDVJ03] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin. A Neural Probalistic Language Model. *JMLR*, 3:1137–1155, 2003.

[EBC+10] D. Erhan, Y. Bengio, A. Courville, P. A. Manzagol, P. Vincent, and S. Bengio. Why does unsupervised pretraining help deep learning? *JMLR*, 11:625–660, 2010.

[HS06] G. e. Hinton and R.R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[LXLZ15] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. Recurrent convolutional neural networks for text classification. pages 2267–2273, 2015.

[MCCD13] T. Mikolov, K. Chen, G. Corrado, and J Dean. Efficient estimation of word representations in vector space. *ICLR Workshop*, 2013.

[MSC+13] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J Dean. Distributed representations of words and phrases and their compositionality. *In NIPS*, pages 3111–3119, 2013.