Word embeddings: word2vec

Daniil Gaidamashko
National Researh University "Higher School of Economics"

***Abstract*—In this paper we attempt to describe the details of a word embedding toolkit named word2vec, developed by T. Mikolov and al. at Google Inc. In the introduction a brief overview of word embedding research field is given. In the main part the key features of word2vec are described.**

INTRODUCTION

Word embedding is a natural language modeling technique, which involves mapping words and phrases from a large corpus of text to corresponding vectors of real numbers. Such transformation is necessary for further application of different machine learning algorithms, including neural networks, which are relied on mostly now. As the major purpose is discovering associations between words that emanate from the same or similar context, the corresponding vectors for such pairs are located in close proximity to one another in the space.

Word2vec is a word embedding toolkit, created by Thomas Mikolov and al. at Google Inc. which exceeds the previous approaches, such as *latent semantic analysis (LSA)*, invented in the late 1980s. The latter is based on the construction of a term-text "occurrence" matrix (rows represent unique words and columns correspond to texts; the cells contain the number of repetions on the term in the text). Then the reduction of its dimension via SVD follows, while preserving the similarity structure among columns. Then words are compared by taking cosine of angle between the two vectors formed by two rows. The closer to one the value appears, the more similar the words turn out to be.

As for word2vec, it is based on neural networks, which perform significantly better than LSA for preserving linear regularities among words, according to Mikilov and al [1]. It has become possible to compute high dimensional vectors on much bigger datasets more accurately. Furthermore, applications of neural network based vectors in many NLP tasks like sentiment analysis and paraphrase detection are considered to benefit from these methods [1].

An analysis of this solution is based on publications [2] and [4], which provide a detailed explanation of word2vec outstanding models and their update equations, as well as advanced optimization techniques.

MAIN PART

The word2vec toolkit includes two different learning models as approach to learn the word embedding. These are:

- Continuous Bag-of-Word model (CBOW);
- Continuous Skip-gram model (SG);

We shall study them separately.

Continuous Bag-of-Word Model

The CBOW model learns the embedding by predicting the current word based on its context. The implementation of the model is 2-layer neural network **??**.

The input layer consists of the one-hot encoded input context words $x_1, \ldots, x_C$ for a word window of size C and vocabulary of size V. The hidden layer is N-dimensional vector h. Finally, the output layer is also one-hot encoded. On the position i in the input vector stands 1 if i-th word in the vocabulary is present in the current context and 0 otherwise. As for the output vector, 1 stands if i-th word is predicted and 0 otherwise. The input layer vectors are connected to hidden layer vector via weight $V \times N$ matrix W; hidden layer is connected to the output layer via $N \times V$ matrix W$'$. The rows v_j^T of W and columns v'_j of W$'$ are corresponding representations of word w_j from vocabulary.

The general purpose is minimizing the loss function E, such that:

$$E = -\frac{1}{C} \sum_i p(w_i | w_1, \ldots w_{i-1}, w_{i+1}, \ldots, w_C)$$

Then *softmax*, a log-linear model, is used for estimating the probability:

$$p(w_i | w_1, \ldots w_{i-1}, w_{i+1}, \ldots, w_C) = \frac{\exp s_i^T \cdot v_i}{\sum_j \exp s_j^T \cdot v_j}$$

where

$$s_i = \frac{1}{C} \sum_{j=1, j \neq i}^{C} v'_j.$$

The Skip-gram model

The continuous skip-gram model learns by predicting the surrounding words given a current word. The training objective of the Skip-gram model is to find word representations that are useful for predicting the surrounding words in a sentence or a document. More formally, given a sequence of training words w_1 , w_2 , w_3, ... , w_T , the objective of the Skip-gram model is to maximize the average log probability

$$\frac{1}{T} \sum_{t=1}^{T} \sum_{-c \leqslant j \leqslant c, j \neq 0} \log p(w_{t+j} | w_t)$$

where c is the size of the training context (which can be a function of the center word w_t). Larger c results in more training examples and thus can lead to a higher accuracy, at the expense of the training time.

The basic Skip-gram formulation defines $p(w_{t+j}|w_t)$ using the softmax function:

$$p(w_{t+j}|w_t) = \frac{\exp(v'^T_{w_O} v_{w_I})}{\sum_{w=1}^{V} exp(\exp(v'^T_w)}$$

where v_w and v'_w are the "input" and "output" vector representations of w, and V is the number of words in the vocabulary.

[1]

Optimising Computational Efficiency

It is clear, that both described approaches require a lot of computations in order to get words' net input, probability prediction and prediction error. The solution is limiting the number of output vectors per training instance. These are *hierarchical softmax* and *negative sampling* methods.

Hierarchical Softmax: A computationally efficient approximation of the full softmax is the hierarchical softmax. It uses a binary tree representation of the output layer with the W words as its leaves and, for each node, explicitly represents the relative probabilities of its child nodes. So a random walk is defined. It assigns probabilities to words. Each word w can be reached by path from the root. Firstly, let $n(w, j)$ be the j-th node on the path from the root to w, and let $L(w)$ be the length of this path, so $n(w, 1) = root$ and $n(w, L(w)) = w$. Secondly, for any inner node n, let $ch(n)$ be an arbitrary fixed child of n and let $[[x]]$ be 1 if x is true and -1 otherwise. Then the hierarchical softmax defines $p(w_O|w_I)$:

$$p(w_O|w_I) = \prod_{j=1}^{L(w)-1} \sigma\left([[n(w, j+1) = ch(n(w, j))]] \cdot v'^T_{n(w,j)} v'_{w_I}\right)$$

where $\sigma(x) = 1/(1 + \exp(-x))$. This means that the cost of computing $\log p(w_O|w_I)$ and $\nabla \log p(w_O|w_I)$ is proportional to $L(w)$ and does not surpass $\log W$. Also, while the original softmax assigns v_w and v'_w to each word w, in current situation a representation v_w is required for word w and a representation v'_w is required for every inner node of the binary tree. [3]

Negative Sampling: Negtive sampling is defines by the objective:

$$\log \sigma(v'^T_{w_O} v'_{w_I}) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w)} \left[\log \sigma(-v'^T_{w_i} v_{w_I})\right]$$

which is used to replace every $\log P(w_O, w_I)$. All we need is distinguishing the target word w_O from draws from the noise distribution $P_n(w)$ using logistic regression with k negative samples for each data sample. T. Mikolov and al. indicated acceptable values of k in the range 5-20 for small training datasets, and 2-5 for large ones. [3]

CONCLUSION

To sum up, the work of T. Mikotov and al. is an essential breakthrough in word embedding field. It allows many researchers to practice with word2vec a lot in their projects. However, the understanding of parameter learning process still remains complex for understandig and challenging for the further development in the sphere.

REFERENCES

[1] Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean, *"Learning while searching in constraint-satisfaction problems"*, *CoRR*, vol. abs/1301.3781, 2013. [Online]. Available: http://arxiv.org/abs/1301.3781

[2] Xin Rong, *"word2vec Parameter Learning Explained"*, *CoRR*, vol. abs/1411.2738, 2014. [Online]. Available: http://arxiv.org/abs/1411.2738

[3] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, Jeffrey Dean, *"Distributed Representations of Words and Phrases and their Compositionality"*, *CoRR*, vol. abs/1310.4546, 2013. [Online]. Available: http://arxiv.org/abs/1310.4546

[4] Yoav Goldberg, Omer Levy, *"word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method"*, *CoRR*, vol. abs/1402.3722, 2014. [Online]. Available: http://arxiv.org/abs/1402.3722