Advanced Word2Vec

Ivan Dziubenko, Anna Tatarnikova, Bogdan Borzdov, Anna Kuzmina

1 Introduction

Word embeddings have become one of the most popular techniques in natural language processing. A word embedding maps each word in the vocabulary to a low dimensional vector. Several algorithms (e.g., Mikolov et al. (2013); Pennington et al. (2014)) can produce word embedding vectors whose distances or inner-products capture semantic relationships between words. The vector representations are useful for solving many NLP tasks, such as analogy tasks or serving as features for supervised learning problems.

Existing popular methods of word embedding do not store the order in which words occur. Until recently, NNLMs have ignored morphology and word shape. However, including information about word structure in word representations has proven valuable for part of speech analysis (Santos Zadrozny, 2014), word similarity (Luong et al., 2013), and information extraction (Qi et al., 2014). Therefore, a key challenge is how to capture more information about words in word embeddings.

In this project, we try to implement Word2Vec with negative sampling for word embedding and methods which was motivated by it - PENN which models the order in which words occur, DIEM which uses neural embeddings learned at the character level to generate a fixed-length syntactic embedding at the world level useful for syntactic word-analogy tasks, leveraging patterns in the characters as a human might when detecting syntactic features such as plurality, and Gaussian embedding.

2 Related works

Many well-known word embedding methods (e.g., Pennington et al. (2014); Mikolov et al. (2013)) don't explicitly utilize or model syntactic structure within text. Andreas Klein (2014) nd that such syntaxblind word embeddings fail to capture syntactic information above and beyond what a statistical parser can obtain, suggesting that more work is required to build syntax into word embeddings. Several syntax-aware embedding algorithms have been proposed to address this. Levy Goldberg (2014a) propose a syntax-oriented variant of the well-known skip-gram algorithm of Mikolov et al. (2013), using contexts generated from syntactic dependency-based contexts obtained with a parser. Cheng Kartsaklis (2015) build syntax-awareness into a neural network model for word embeddings by indroducing a negative set of samples in which the order of the context words is shufed, in hopes that the syntactic elements which are sensitive to word order will be captured.

3 Partitioned Embedding Neural Network (PENN)

PENN improves upon word2vec by modeling the order in which words occur. It models order by partitioning both the embedding and classifier layers. The first property of PENN is that each word embedding is partitioned. Each partition is trained differently from each other partition based on word order, such that each partition models a different probability distribution. These different probability distributions model different perspectives on the same word. The second property of PENN is that the classifier has different inputs for words from different window positions. The classifier is partitioned with equal partition dimensionality as the embedding. It is possible to have fewer partitions in the classifier than the embedding, such that a greater number of word embeddings are summed/averaged into fewer classifier partitions. This configuration has better performance when using smaller dimensionality feature vectors with large windows as it balances the (embedding partition size) / (window size) ratio. Figure 1 shows PENN structure.

There is a simpler case. Instead of each partition corresponding to a window position, there are only two partitions. One partition corresponds to every positive, forward predicting window position (left of the focus term) and the other partition corresponds to every negative, backward predicting window position (right of the focus term). For each partition, all embeddings corresponding to that partition are summed or averaged when being propagated forward. This is called directional configuration of PENN.

The skip-gram training style under the PENN framework optimizes the following objective function:

$$h \arg\max_\theta \left(\prod_{(w,C)\in d - c \leq j \leq c, j \neq 0} p\left(w_j = 1 | c_j^j; \theta\right) \prod_{(w,C)\in d' - c \leq j \leq c, j \neq 0} p\left(w_j = 0 | c_j^j; \theta\right) \right))$$

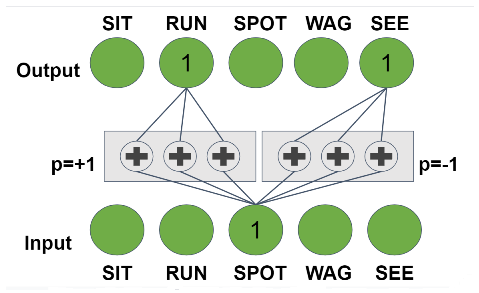

Figure 1: Configuration of PENN.

4 Dense Interpolated Embedding Model (DIEM)

DIEM uses neural embeddings learned at the character level to generate a fixed-length syntactic embedding at the world level useful for syntactic word-analogy tasks, leveraging patterns in the characters as a human might when detecting syntactic features such as plurality.

Generating syntactic embeddings begins by generating character embeddings. Character embeddings are generated using vanilla word2vec by predicting a focus character given its context. This clusters characters in an intuitive way, vowels with vowels, numbers with numbers, and capitals with capitals. In this way, character embeddings represent morphological building blocks that are more or less similar to each other, based on how they have been used. Once character embeddings have been generated, interpolation may begin over a word of length I. The final embedding size must be selected as a multiple M of the character embedding dimensionality C. For each character in a word, its index i is first scaled linearly with the size of the final "syntactic" embedding such that s = M * i / l. Then, for each length C position m (out of M positions) in the final word embedding vm, a squared distance is calculated relative to the scaled index such that distance d = pow(1-(abs(s - j)) / M,2). The character vector for the character at position i in the word is then scaled by d and added elementwise into position m of vector v. A more efficient form of this process caches a set of transformation matrices, which are cached values of di,m for words of varying size. These matrices are used to transform variable length concatenated character vectors into fixed length word embeddings via vector-matrix multiplication. These embeddings are useful for a variety of tasks, including syntactic word-analogy queries. Furthermore, they are useful for syntactic query expansion, mapping sparse edge cases of a word (typos, odd capitalization, etc.) to a more common word and its semantic embedding

Algorithm 1 Dense Interpolated Embedding Pseudocode

```
Input: wordlength $I$, list char embeddings (e.g. the word) $char_i$, multiple $M$, char dim $C$, vector $v_m$
for $i = 0$ to $I - 1$ do
    $s =$ M * $i/$l
    for $m = 0$ to $M - 1$ do
        $d = pow(1 - (abs(s - m)) / M, 2)$
        $v_m = v_m + d * char_i$
    end for
end for
```

5 Gaussian embedding

Each word w is mapped to some dictionary D and context word type c in a dictionary C is mapped to a Gaussian distribution over a latent embedding space, such that linguistic properties of the words are captured by properties of and relationships between the distributions. An element of the dictionary is a word type, and a particular observed token in some context is a word token.

Let the parameter correspond to our learned word representations, and the x and y input-output pairs correspond to word tokens and their contexts. For scoring pairs of inputs x and outputs y, parametrized by is an energy function E(x,y) being used (LeCun et al., 2006). The goal of energy-based learning is to train the parameters of the energy function to score observed positive input-output pairs higher (or lower, depending on sign conventions) than negative pairs. This is accomplished by means of a loss function L which defines which pairs are positive and negative according to some supervision, and provides gradients on the parameters given the predictions of the energy function.

To get relative values of energies rather than absolute, as the loss function was used a ranking-based loss – max-margin ranking objective, similar to that used in Rank-SVM.

$$L_m\left(w, c_p, c_n\right) = \max\left(0, m - E\left(w, c_p\right) + E\left(w, c_n\right)\right)$$

Asymmetric similarity: Kullback–Leibler divergence. To encode vectors context distributions, they were being trained through KL-divergence. It also even incorporates more explicit directional supervision. The goal to optimize following energy function:

$$-E\left(P_i, P_j\right) = D_{KL}\left(\mathcal{N}_j \| \mathcal{N}_i\right) = \int_{x \in R^n} \mathcal{N}\left(x; \mu_i, \Sigma_i\right) \log \frac{\mathcal{N}\left(x; \mu_j, \Sigma_j\right)}{\mathcal{N}\left(x; \mu_i, \Sigma_i\right)} dx$$

$$= \frac{1}{2}\left(tr\left(\Sigma_i^{-1}\Sigma_j\right) + \left(\mu_i - \mu_j\right)^\top \Sigma_i^{-1}\left(\mu_i - \mu_j\right) - d - \log\frac{det\left(\Sigma_j\right)}{det\left(\Sigma_i\right)}\right)$$

6 Experiments

6.1 Implementation

We implemented directional skip-gram PENN with negative sampling, DIEM and Word2Gaussian. In all cases we used Python with PyTorch. We used Adam as an optimizer in PENN and DIEM and Adagrad in Word2Gaussian. For PENN and DIEM custom loss functions from the related paper were used, while for the Word2Gaussian we used max-margin. Following parameters were used for the models:

PENN: embedding size (as the size of the hidden layer) = 500, window size = 5, 5 epoch, initial learning rate 0,001

DIEM character embedding size = 500, word embedding size = 2500, window size = 2, 5 epoch, initial learning rate 0,001

Word2Gaussian: embedding size = 50, C = 2.0, window size = 5, 1 epoch, constant learning rate 0,005

6.2 Methodology

We used the English Wikipedia dump dataset. The dataset was prepossessed before training, in particular stop-words (e.g. articles), very rare words and unwanted symbols were removed. We trained all the models on the same dataset just to be able to show how differently they work on the same data. Most comparisons were made by observing how different models have words with best cosine similarity to some specified words.

6.3 Results

Table. Top cosine similar words. Comparison between word embedding methods.

Word2Vec		PENN		Word2Gaussian	
“anarchism”	similarity	“anarchism”	similarity	“anarchism”	similarity
“capitalist”	0.93298	“individualist”	0.8593	“anarchist”	0.8471
“capitalism”	0.90261	“anarchist”	0.84715	“communist”	0.8331
“anarchist”	0.89451	“rothbard”	0.83162	“historical”	0.7984
“anarcho”	0.89449	“metaphysical”	0.82071	“political”	0.7734
“libertarian”	0.8798	“zionism”	0.81265	“power”	0.7684

Word2Vec		PENN		Word2Gaussian	
“general”	similarity	“general”	similarity	“general”	similarity
“partisan”	0.50995	“leader”	0.5788	“despot”	0.453
“officer”	0.49944	“davi”	0.56246	“officer”	0.441
“naval”	0.49662	“senator”	0.56038	“structure”	0.427
“subordinate”	0.48742	“deputy”	0.55958	“chief”	0.357
“cauchy”	0.41986	“commission”	0.55916	“emperor”	0.342

Word2Vec		PENN		DIEM	
“general”	similarity	“general”	similarity	“general”	similarity
“partisan”	0.50995	“leader”	0.5788	“mineral”	0.9997
“officer”	0.49944	“davi”	0.56246	“cerebral”	0.99958
“naval”	0.49662	“senator”	0.56038	“generous”	0.99951
“subordinate”	0.48742	“deputy”	0.55958	”liberal”	0.9994
“cauchy”	0.41986	“commission”	0.55916	”renewal”	0.9994

Word2Vec		PENN		DIEM	
“boy+girl”	similarity	“boy+girl”	similarity	“boy+girl”	similarity
“thirteen”	0.72937	“aisha”	0.70917	“foul”	0.9987
“teenage”	0.72388	“catherine”	0.70795	“pour”	0.9985
“beautiful”	0.70801	“aphrodite”	0.70564	“paul”	0.9985
“rap”	0.70469	“margaret”	0.69833	”gil”	0.9984
“astro”	0.6931	“wicked”	0.69320	”kirk”	0.9984

Brief overview of compared results shows that the methods PENN, Word2Vec, Word2Gaussian work good in semantics. DIEM does not represent semantics at all, only syntactic, but does it pretty well.

7 Work distribution

Bogdan Borzdov and Anna Kuzmina worked on Gaussian embedding implementation, Ivan Dziubenko and Anna Tatarnikova worked on PENN and DIEM implementations. All of the team members contributed to the presentation and the project report.

8 Conclusion

1. PENN models the order in which words occur and in some cases it shows better results in semantics than Word2Vec. Both methods can learn much faster with negative sampling.

2. Word2Vec and PENN capture syntactic relationships worse than special syntactic methods as DIEM.

3. It appears that Word2Gaussian can provide better word similarity in comparison with Word2Vec and PENN.

4. Word2Gaussian directly represents notions of uncertainty and enables a richer geometry in the embedded space as it considers densities over a latent space.

References

- Andrew Trask, David Gilmore, Matthew Russell. Modeling Order in Neural Word Embeddings at Scale, 2015.

- Luke Vilnis, Andrew McCallum. Word Representations via Gaussian Embedding, 2015.

- Tomas Mikolov, Kai Chen, Greg Corrado, and Jerey Dean. Ecient estimation of word representations in vector space. Proceedings of the International Conference on Learning Representations, 2013a.

- Le, Quoc V. and Mikolov, Tomas. Distributed representations of sentences and documents, 2014.