4A2-3

Learning Syntactically Plausible Word Representations by Solving Word Ordering

Noriki Nishida Hideki Nakayama

Graduate School of Information Science and Technology
The University of Tokyo

Syntactic information is crucial to capture sentence structures such as word order. However, recent works on learning distributed word representations do not model word order explicitly. As the ability to represent syntactic information is missing in these embeddings models, the vectors learned with these methods are suboptimal for syntax-related tasks. In this paper, we propose a new approach to learning syntactically plausible word representations. The proposed method learns word embeddings by solving word ordering tasks using pointer networks. We evaluate our approach in comparison with prior works on part-of-speech tagging and word analogy task. The experimental results demonstrate that the proposed method produces vector spaces that are capable of capturing syntactic regularities better than existing methods.

1. Introduction

Various approaches for learning distributed word representations have been proposed in recent years. However, almost all existing methods such as Skip-Gram/CBOW [Mikolov 13a] and GloVe [Pennington 14] only consider co-occurrences between two words, and ignore word order which is an indispensable resource for learning syntactic representations of words. The vectors learned by these methods are known to be suboptimal for syntax-related tasks such as part-of-speech tagging and parsing [Andreas 14].

Word ordering tests, or linearization, are commonly used to evaluate students' ability of language. Suppose that we are given a set of out-of-order words {"yesterday", "ate", ".", "fish", "john"}, which are tokens in a original sentence. As we know the part-of-speech tag of each word and can easily predict the structure of the original sentence from the information, the candidate answers are narrowed down to two: "Fish ate John yesterday." or "John ate fish yesterday.". By considering the meaning of each word and checking the sentence meaning against our common-sense knowledge, we can finally recover the original sentence: "John ate fish yesterday.". We depict the above explanation in Figure 1. Of course, it might not be necessary for computers to mimic exactly the same reasoning process described above. However, the knowledge about words used in the above example is necessary to solve the problem.

Inspired by this observation, the fundamental question we want to explore in this paper is whether the word ordering task can be an objective for an acquisition of syntactic knowledge about words. Our underlying hypothesis is that, if we can obtain machine learning models such as neural networks which can solve word ordering, the models must have learned representations capturing syntactic information of words. To this end, we modify the recently proposed pointer network (Ptr-Net) [Vinyals 15] to solve word ordering task efficiently. The model consists of two components: the word embedding part (*encoder*) and the ordering part (*decoder*), which are jointly trained on the word ordering task.

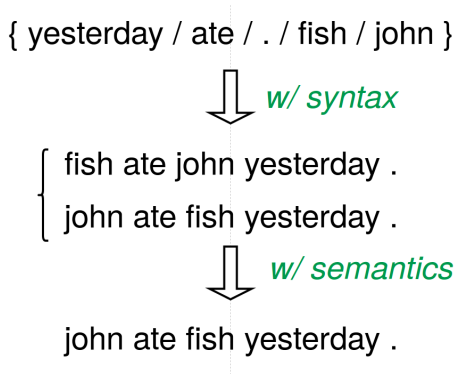

Figure 1: Illustration of word ordering task. The goal of this task is to predict an original order of a shuffled sentence. Our underlying hypothesis is that doing well on this task requires knowledge (especially syntactic knowledge) about words. In this paper, our goal is to build syntactic word representations through training machine learning models (e.g., neural networks) to solve the word ordering task.

To evaluate our approach, we quantitatively compare our method with previous word embedding models on part-of-speech tagging and word analogy task. The experimental results demonstrate that our approach is competitive with or outperforms the previous methods on these syntax-related tasks.

2. Approach

In this section, we will give a precise description of the pointer network which we extend to solve word ordering efficiently.

Contact: {nishida, nakayama}@nlab.ci.i.u-tokyo.ac.jp

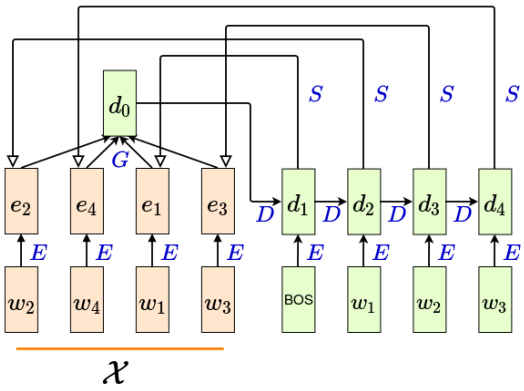

Figure 2: Overall view of the word ordering model. Given a set of input words $\mathcal{X}$, an encoder (orange) map the words to low-dimensional vectors independently with a function E. Then, we summarize the representations and compute the initial hidden state $\boldsymbol{d}_0$ with a function G. An decoder (green) generates an output sequence of selection decisions over input words with respect to the current state $\boldsymbol{d}_j$. D denotes a state update function, and S is a score function.

2.1 Encoder

We illustrate our modified Ptr-Net in Figure 2. Given a set of shuffled words $\mathcal{X} = \{w_1, \ldots, w_T\}$, the encoder network aims to produce a set of fixed-dimensional vectors $\{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_T\}$ where $\boldsymbol{e}_i \in \mathbb{R}^d$ corresponds to the word w_i. To this end, we first embed every single word w_i to m-dimensional vector independently, i.e.

$$\boldsymbol{v}_i = E_{\boldsymbol{\theta}_E}(w_i) \in \mathbb{R}^m, \tag{1}$$

where $E_{\boldsymbol{\theta}_E}$ is a word embedding function and $\boldsymbol{\theta}_E$ is the parameter of $E_{\boldsymbol{\theta}_E}$.

The original Ptr-Net of [Vinyals 15] uses an LSTM to encode each word with contextual information as follows:

$$\boldsymbol{e}_i = \mathrm{LSTM}_{\mathrm{enc}}(\boldsymbol{v}_i, \boldsymbol{e}_{i-1}) \in \mathbb{R}^d. \tag{2}$$

In the word ordering task, however, the order of the input sequence $\{w_1, \ldots, w_T\}$ to the $\mathrm{LSTM}_{\mathrm{enc}}$ is meaningless. Nonetheless, the representation $\boldsymbol{e}_i$ depends on the order of $\{w_1, \ldots, w_{i-1}\}$ according to Eq. (2). In addition, this order-dependence during encoding is problematic for learning generalized word representations due to data sparseness. In order to prevent such undesirable behavior, we eliminate the $\mathrm{LSTM}_{\mathrm{enc}}$ from Eq. (2) and simplify the encoder as follows:

$$\boldsymbol{e}_i = \boldsymbol{v}_i = E_{\boldsymbol{\theta}_E}(w_i). \tag{3}$$

In Eq. (3), the representation $\boldsymbol{e}_i$ does not depend on the input order anymore, which is suitable for the word ordering task.

2.2 Decoder

The decoder network aims to generate an output sequence of selection decisions $(y_1, \ldots, y_T)$ over the input elements, conditioned on the representations $\{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_T\}$ produced by the encoder network. Here $(y_1, \ldots, y_T)$ is a sequence of T indices, each between 1 and T. To reorder the set of words $\mathcal{X} = \{w_1, \ldots, w_T\}$, we need to feed the information about $\mathcal{X}$ to the decoder network. To this end, we initialize the hidden state of the decoder LSTM by using a function $G_{\boldsymbol{\theta}_G}$:

$$\boldsymbol{d}_0 = G_{\boldsymbol{\theta}_G}(\{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_T\}) \in \mathbb{R}^d, \tag{4}$$

where $\boldsymbol{\theta}_G$ denotes the parameter of $G_{\boldsymbol{\theta}_G}$. We formalize the function $G_{\boldsymbol{\theta}_G}$ as a simple vector summation followed by a non-linear transformation:

$$G_{\boldsymbol{\theta}_G}(\{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_T\}) = \tanh(\boldsymbol{W}_G \sum_{i=1}^{T} \boldsymbol{e}_i + \boldsymbol{b}_G) \tag{5}$$

where $\boldsymbol{W}_G \in \mathbb{R}^{d \times m}$ and $\boldsymbol{b}_G \in \mathbb{R}^d$ are a projection matrix and a bias vector.

The j-th hidden state is computed as follows:

$$\boldsymbol{d}_j = D_{\boldsymbol{\theta}_D}(\boldsymbol{e}_{y_{j-1}}, \boldsymbol{d}_{j-1}), \tag{6}$$

where $y_{j-1} \in \{1, \ldots, T\}$ denotes the index of the word selected at the previous step $(j-1)$. In this paper we use an LSTM as the function $D_{\boldsymbol{\theta}_D}$, and $\boldsymbol{\theta}_D$ denotes the parameters of the LSTM.

Then, the model predicts a selection distribution over the input words:

$$\Pr(i|\boldsymbol{d}_j) = \frac{\exp(S_{\boldsymbol{\theta}_S}(\boldsymbol{e}_i, \boldsymbol{d}_j))}{\sum_{k=1}^{T} \exp(S_{\boldsymbol{\theta}_S}(\boldsymbol{e}_k, \boldsymbol{d}_j))}. \tag{7}$$

where $S_{\boldsymbol{\theta}_S}$ is a score function that calculates a selection score of the candidate word w_i with respect to the current hidden state $\boldsymbol{d}_j$. Finally, at j-th decoding step, we select the index that maximizes the Eq. (7) as follows:

$$y_j = \underset{i \in \{1, \ldots, T\}}{\mathrm{argmax}} \Pr(i|\boldsymbol{d}_j). \tag{8}$$

Here y_j represents the index of the selected word $w_{y_j} \in \{w_1, \ldots, w_T\}$.

2.3 Loss Function

Our pointer network consists of four parameterized functions: $E_{\boldsymbol{\theta}_E}$, $G_{\boldsymbol{\theta}_G}$, $D_{\boldsymbol{\theta}_D}$ and $S_{\boldsymbol{\theta}_S}$. The parameters $\boldsymbol{\theta} = \{\boldsymbol{\theta}_E, \boldsymbol{\theta}_G, \boldsymbol{\theta}_D, \boldsymbol{\theta}_S\}$ are jointly optimized by minimizing the negative log-likelihood:

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^{N} \sum_{j=1}^{T} -\log \mathrm{P}(t_j^{(n)}|\boldsymbol{d}_j^{(n)}) \tag{9}$$

where N is a mini-batch size, and $t_j^{(n)} \in \{1, \ldots, T\}$ denotes an index of the word which should be placed at j-th position in the n-th sentence.

We also use the retrofitting technique [Faruqui 15] to enrich our syntax-focused word representations with distributional semantics. We initialize our word embedding parameters $\boldsymbol{\theta}_E^{\mathrm{init}}$ with the vectors learned by the SGNS model (See Section 4.1), and add the following penalty term to the loss function in Eq. (9):

$$\|\boldsymbol{\theta}_E - \boldsymbol{\theta}_E^{\mathrm{init}}\|_{\mathcal{F}}, \tag{10}$$

where $\boldsymbol{\theta}_E$ denotes current word embedding parameters and $\|\cdot\|_{\mathcal{F}}$ is the Frobenius norm. This retrofitting term tries to keep the word vectors close to the original vectors, while minimizing the loss function on word ordering (Eq. (9)).

3. Related Work

Learning distributed word representations has a long history in natural language processing. Latent semantic analysis (LSA) [Deerwester 90] performs a singular value decomposition (SVD) to co-occurrence matrices of "term-term" type, which produces a low-dimensional vector for each word. [Bengio 03] introduced a Neural Network Language Model (NNLM) where word vectors are simultaneously learned along with a language model. More recently, [Mikolov 13a, Mikolov 13b] proposed two log-bilinear models: the Skip-Gram model (SG) and the Continuous Bag-Of-Words model (CBOW). These models are formalized as a one-layer structure based on the inner product between two word vectors.

The main drawback of these methods except for the NNLM is that they do not model word order explicitly. Information about word order, however, is an indispensable resource for learning syntactic representations of words. The vectors learned by these models are thus suboptimal for syntax-related tasks such as part-of-speech tagging or dependency parsing [Andreas 14]. Although the NNLM utilizes word order, the NNLM is very slow to train unfortunately. This is a consequence of having to predict the probability distribution over entire vocabulary, which is avoided in our method.

Most recently, [Ling 15b] introduced two extensions of the SG and CBOW: the Structured Skip-Gram model (SSG) and the Continuous Window model (CWindow). The core idea of the recent work of [Trask 15] is also the same as the SSG and CWindow. Unlike the original SG and CBOW of [Mikolov 13b], these models are aware of relative positioning of contextual words and develop different context embeddings. [Ling 15a] also proposed a method which incorporates the CBOW with a soft attention mechanism over contextual words.

The major difference of our method from the LM-based approaches [Bengio 03, Mikolov 10] and the approaches considering relative positions of contextual words [Ling 15b, Trask 15, Ling 15a] is the model architecture and its objective where our word embeddings are optimized more explicitly for the word ordering task than those prior methods. We also modify the recently proposed pointer networks (Ptr-Nets) [Vinyals 15] for applying it to word ordering task efficiently.

4. Experiments

4.1 Setup

We used the English Wikipedia corpus for training the proposed method and baseline models. We tokenized and lowercased all tokens, then replaced all digits with 7. We also appended special "⟨EOS⟩" symbols to the last of each sentence. We built a vocabulary of the most frequent 300,000 words, and then replaced out-of-vocabulary tokens in the corpus with a special "⟨UNK⟩" symbol. The resulting corpus contains about 80 million sentences with 1.2 billion tokens. We randomly extracted $10,000$ sentences as the validation set.

We used 300-dimensional vectors to represent words throughout all experiments. The dimensionality of the hidden states in the LSTM was set to 512. We set the L2 regularization coefficient (called weight decay) to 4×10^{-6}. We used mini-batch stochastic gradient descent to train our model. The mini-batch size was set to 180. We computed the loss function on the validation set every $20,000$ iterations and utilized early stopping with patience 10 to avoid overfitting. We built word vectors using the Skip-Gram with Negative Sampling model (SGNS) of [Mikolov 13b] on the pre-processed corpus and then used the vectors for initializing the word embeddings $\boldsymbol{\theta}_E^{\text{init}}$ in Eq. (10).

Table 1: Test accuracy (%) of different word embeddings for part-of-speech tagging on the Penn Treebank (PTB) corpus.

Method	Accuracy (%)
SGNS	96.76
GloVe	96.31
SSG	96.94
CWindow	96.78
LSTM-LM	96.92
Our method (w/o ret.)	**97.04**
Our method	97.01

4.2 Part-of-Speech Tagging

Part-of-speech (POS) tagging is one of the major syntax-related tasks in NLP. In POS tagging, every word in a sentence is to be classified into its POS class. In this experiment, we evaluated learned word representations by using the vectors as features for supervised POS tagging. We used the Wall Street Journal portion of the Penn Treebank (PTB) corpus*1 [Marcus 93]. We followed the standard splits of sections 0-18 for training, 19-21 for development and 22-24 for test sets. The dataset contains 45 tags. The evaluation metric is the word-level accuracy.

In our experiments, three successive words are projected to feature vectors using learned word embeddings, which are concatenated and fed to a 2-layer perceptron that predicts the POS class of the center word. We set the number of hidden units to 300. We trained the classifier via stochastic gradient descent without updating the word embeddings.

Table 1 shows the results of the different word representations on the test set of the PTB corpus. The results show that our method yields better performance than the baselines. We also observe that the methods considering word order (SSG, LSTM-LM and our method) tend to outperform the methods without considering word order (SGNS and GloVe). The result supports our hypothesis that word order is crucial for learning syntactic representations. In

∗1 We use the LDC99T42 Treebank release 3 version.

Table 2: Accuracy (%) of different word embeddings for the word analogy task.

Method	Total Accuracy (%)
SGNS	69.9
GloVe	68.3
SSG	69.7
CWindow	58.2
LSTM-LM	27.0
Our method	**70.6**

addition, our method without retrofitting slightly outperforms that with retrofitting. This result is intuitive as the objective function of our method focuses on learning syntax rather than distributional semantics.

4.3 Word Analogy

The word analogy task has been used in many previous works to evaluate the ability of word embeddings to represent semantic and syntactic regularities. In this experiment we used the word analogy dataset produced by [Mikolov 13a]. The dataset consists of questions like "a is to b what c is to ?", denoted as "a : b :: c : ?".

In this experiment we only report the scores of our method with retrofitting because it consistently outperforms that without retrofitting. As can be seen in the Table 2, our method yields the best performance. Although we do not show here (due to the limited space), our method also achieves the best scores on 4 out of 9 syntactic question types, and 3 out of 5 semantic question types. Interestingly, our method can outperform the SGNS, which is used to initialize our word embeddings. This result implies that word order has a potential to improve not only syntactic but also semantic word representations.

5. Conclusion

Most of works on learning distributed word representations focus on distributional semantics and ignore word order, which is crucial for capturing syntactic information about words. In this work, in order to obtain syntactically plausible word representations, we presented an approach based on pointer networks trained to solve the word ordering task. The experimental results demonstrate that our method outperforms other baselines on POS tagging and word analogy tasks.

References

[Andreas 14] Andreas, J. and Klein, D.: How much do word embeddings encode about syntax?, in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics* (2014)

[Bengio 03] Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C.: A neural probabilistic language model, *Journal of Machine Learning Research*, Vol. 3, No. Feb, pp. 1137–1155 (2003)

[Deerwester 90] Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R.: Indexing by latent semantic analysis, *Journal of the American Society for Information Science*, Vol. 41, No. 6, p. 391 (1990)

[Faruqui 15] Faruqui, M., Dodge, J., Jauhar, S. K., Dyer, C., Hovy, E., and Smith, N. A.: Retrofitting word vectors to semantic lexicons, in *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (2015)

[Ling 15a] Ling, W., Chu-Cheng, L., Tsvetkov, Y., and Amir, S.: Not all contexts are created equal: Better word representations with variable attention, in *Proceedings of the 2015 Conference of Empirical Methods in Natural Language Processing* (2015)

[Ling 15b] Ling, W., Dyer, C., Black, A., and Trancoso, I.: Two/too simple adaptation of word2vec for syntax problems, in *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (2015)

[Marcus 93] Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B.: Building a large annotated corpus of English: The Penn Treebank, *Computational Linguistics*, Vol. 19, No. 2, pp. 313–330 (1993)

[Mikolov 10] Mikolov, T., Karafiát, M., Burget, L., Cernockỳ, J., and Khudanpur, S.: Recurrent neural network based language model, in *Proceedings of INTERSPEECH* (2010)

[Mikolov 13a] Mikolov, T., Chen, K., Corrado, G. S., and Dean, J.: Efficient estimation of word representations in vector space, *arXiv preprint arXiv:1301.3781* (2013)

[Mikolov 13b] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J.: Distributed representations of words and phrases and their compositionality, in *Advances in Neural Information Processing Systems* (2013)

[Pennington 14] Pennington, J., Socher, R., and Manning, C. D.: GloVe: Global vectors for word representations, in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing* (2014)

[Trask 15] Trask, A., Gilmore, D., and Russell, M.: Modeling Order in NeuralWord Embeddings at Scale, in *Proceedings of The 32nd International Conference on Machine Learning* (2015)

[Vinyals 15] Vinyals, O., Fortunato, M., and Jaitly, N.: Pointer networks, in *Advances in Neural Information Processing Systems* (2015)