IS DEEP LEARNING REALLY NECESSARY FOR WORD EMBEDDINGS?

Rémi Lebret[a] Joël Legrand Ronan Collobert

Idiap-RR-44-2013

DECEMBER 2013

[a]Idiap

Is deep learning really necessary for word embeddings?

Rémi Lebret, Joël Legrand and Ronan Collobert
Idiap Research Institute
Rue Marconi 19, CP 592
1920 Martigny, Switzerland
{rlebret,jlegrand,collober}@idiap.ch

Abstract

Word embeddings resulting from neural language models have been shown to be successful for a large variety of NLP tasks. However, such architecture might be difficult to train and time-consuming. Instead, we propose to drastically simplify the word embeddings computation through a Hellinger PCA of the word co-occurence matrix. We compare those new word embeddings with some well-known embeddings on NER and movie review tasks and show that we can reach similar or even better performance. Although deep learning is not really necessary for generating good word embeddings, we show that it can provide an easy way to adapt embeddings to specific tasks.

1 Introduction

Building word embeddings have always generated much interest for linguists. Popular approaches such as Brown clustering algorithm [1] have been used with success in a wide variety of NLP tasks [2, 3, 4]. Those word embeddings are often seen as a low dimensional-vector space where the dimensions can be seen as features potentially describing syntactic or semantic properties. Recently, distributed approaches based on neural network language models (NNLM) have revived the field of learning word embeddings [5, 6, 7, 8, 9]. However, a neural network architecture can be hard to train. Finding the right parameters to tune the model is often a challenging task and the training phase is in general computationally expensive.

This paper aims to show that such good word embeddings can be obtained using simple linear operations. We show that similar word embeddings can be computed using the word co-occurrence statistics and a well-known dimensionality reduction operation such as Principal Component Analysis (PCA). We then compare our embeddings with the CW [5], Turian [7], HLBL [10] embeddings which come from deep architectures and the LR-MVL [11] embeddings which also come from a spectral method on several NLP tasks.

We claim that a simple spectral method as PCA can generate word embeddings as good as with deep-learning architectures. On the other hand, deep-learning architectures have shown their potential in several supervised NLP tasks by using these word embeddings. As they are usually generated over large corpora of unlabeled data, words are represented in a generic manner. Having generic embeddings, good performance can be achieved on NLP tasks where the syntactic aspect is dominant such as Part-Of-Speech, chunking and NER [7, 8, 11]. For supervised tasks relying more on the semantic aspect as sentiment classification, it is usually helpful to adapt the existing embeddings to improve performance [12]. We show in this paper that such embedding specification can be easily done via deep-learning architectures and that helps to increase general performance.

2 Related Work

As 80% of the meaning of English text comes from word choice and the remaining 20% comes from word order [13], it seems quite important to preserve word order. Connectionist approaches have therefore been proposed to develop distributed representations which encode the structural relationships between words [14, 15, 16]. Most recently, a neural network language model was proposed in Bengio et al. [17] where word vector representations are simultaneously learned along with a statistical language model. This architecture inspired other authors: Collobert and Weston [5] designed a neural language model which eliminates the linear dependency on vocabulary size, Mnih and Hinton [10] proposed a hierarchical linear neural model, Mikolov et al. [18] investigated a recurrent neural network architecture for language modeling. Such architectures being trained over large corpora of unlabeled text with the aim to predict correct scores end up learning the co-occurence statistics.

Linguists assumed long ago that words occurring in similar contexts tend to have similar meanings [19]. Using the word co-occurrence statistics is thus a natural choice to embed similar words into a common vector space [20]. Common approaches calculate the frequencies, apply some transformations (tf-idf, PPMI), reduce the dimensionality and calculate the similarities [21]. Considering a fixed-sized word vocabulary $\mathcal{D}$, the co-occurence matrix is then vocabulary size dependent. To reduce the dimensionality of the co-occurence matrix F of size $W \times |\mathcal{D}|$ by mapping F into a matrix f of size $W \times d$, where $d \ll |\mathcal{D}|$, techniques such as Singular Valued Decomposition (SVD) are widely used (e.g. LSA [22], ICA [23]). However, word co-occurence statistics are discrete distributions. We thus believe that information theory distance measure such as Hellinger distance should be more efficient than Euclidean distance to smooth the matrix F. In this paper we will compare our method with the Low Rank Multi-View Learning (LR-MVL) method which is another spectral method based on Canonical Correlation Analysis (CCA) to learn word embeddings [11].

It has been proved that using word embeddings as features helps to improve general performance on many NLP tasks [7]. However these embeddings can be too generic to perform well on other tasks such as sentiment classification. For such task, word embeddings must capture the sentiment information. Maas et al. [24] proposed a model for jointly capturing semantic and sentiment components of words into vector spaces. More recently, Labutov and Lipson [12] presented a method which takes existing embeddings and by using labeled data re-embed them in the same space to be better predictor in a supervised task. Inspired by the work of Collobert et al. [8], we rather apply a deep learning architecture to our NLP systems and we propose to fine-tune our existing embedding for each task by backpropagation.

3 Spectral Method for Word Embeddings

A NNLM learns which words among the vocabulary appear more likely after a given context sequence of words. More formally, it learns the next word probability distribution. Instead, simply counting words on a large corpus of unlabeled text can be performed to retrieve those word distributions and to represent words [20].

3.1 Word co-occurence statistics

"You shall know a word by the company it keeps" [25]. It is a natural choice to use the word co-occurence statistics to acquire representations of word meanings. Raw word co-occurence frequencies are computed by counting the number of times each word $w \in \mathcal{D}$ occurs after a context sequence of words T:

$$p(w|T) = \frac{p(w,T)}{p(T)} = \frac{n(w,T)}{\sum_w n(w,T)} \tag{1}$$

where $n(w,T)$ is the number of times each context word w occurs after the context T. The next word probability distribution p for each word or sequence of words is thus obtained. It is a multinomial distribution of $|\mathcal{D}|$ classes (words). A co-occurence matrix of size $N \times |\mathcal{D}|$ is thus obtained by computing those frequencies over all the N possible sequences of words.

3.2 Hellinger distance

Similarities between words can be derived by computing a distance between their corresponding word distributions. Several distances (or metric) over discrete distributions exist, such as the Bhattacharyya distance, the Hellinger distance or Kullback-Leibler divergence. We chose here the Hellinger distance for its simplicity and symmetry property (as it is a true distance). Considering two discrete probability distributions $P = (p_1, \dots, p_k)$ and $Q = (q_1, \dots, q_k)$, the Hellinger distance is formally defined as:

$$H(P,Q) = -\frac{1}{\sqrt{2}}\sqrt{\sum_{i=1}^{k}(\sqrt{p_i} - \sqrt{q_i})^2} \tag{2}$$

which is directly related to the Euclidean norm of the difference of the square root vectors:

$$H(P,Q) = \frac{1}{\sqrt{2}}\|\sqrt{P} - \sqrt{Q}\|_2 \tag{3}$$

Note that it makes more sense to take the Hellinger distance rather than the Euclidean distance for comparing discrete distributions, as P and Q are unit vectors according to the Hellinger distance ($\sqrt{P}$ and $\sqrt{Q}$ are units vector according to the ℓ_2 norm).

3.3 Dimensionality Reduction

As discrete distributions are vocabulary size dependent, using directly the distribution as a word embedding is not really tractable for large vocabulary. We propose to perform a principal component analysis (PCA) of the word co-occurence probabilities square root matrix to represent words in a lower dimensional space while minimizing the reconstruction error according to the Hellinger distance.

4 Deep Learning Architectures for NLP tasks

Traditional NLP approaches extract from documents a rich set of hand-designed features which are then fed to a standard classification algorithm. The choice of features is a task-specific empirical process. In contrast, we want to pre-process our features as little as possible. In that respect, a multilayer neural network architecture seems appropriate as it can be trained in an end-to-end fashion on the task of interest.

4.1 Sentence-level Approach

The sentence-level approach aims at tagging with a label each word in a given sentence. Embeddings of each word in a sentence are fed to a convolutional layer followed by some linear and non-linear layers, ending by a sentence tags inference layer.

Convolutional layer Our convolutional network successively takes the complete sentences, produces local features around each word of the sentence thanks to convolutional layers, combines these feature into a global feature vector which can then be fed to standard affine layers.

Sentence tags inference There exists strong dependencies between tags in a sentence: some tags cannot follow other tags. To take this specificity into account, we infer tag paths from the previous scores using a lattice. A transition score A_{tu} for jumping from tag $t \in \mathcal{T}$ to $u \in \mathcal{T}$ is introduced, as well as an initial score A_{t0} for starting from the t^{th} tag. Each node G_{t_n} is assigned a score $s(x_n)_t$ from the previous layer of the architecture. Given a pair of nodes G_{t_n} and $G_{u_{n+1}}$, an edge with transition score A_{tu} is added on the lattice.

Finally, a score for a path $[t]_1^N$ in the lattice G is obtained, as the sum of scores along $[t]_1^N$ in G:

$$S([w]_1^N, [t]_1^N, \theta) = \sum_{n=1}^{N}(A_{t_{n-1}t_n} + s(x_n)_{t_n})$$

where θ represents all the trainable parameters of the complete architecture. The sentence tags $[t^*]_1^N$ are then inferred by finding the path which leads to the maximal score:

$$[t^*]_1^N = \underset{[t]_1^N \in \mathcal{T}^N}{\operatorname{argmax}} S([w]_1^N, [t]_1^N, \theta)$$

The Viterbi algorithm is the natural choice for this inference. All parameters θ are trained in a end-to-end manner.

Training The neural network is trained by maximizing a likelihood over the training data, using stochastic gradient ascent. The score for a path can be interpreted as a conditional probability over a path by taking it to the exponential (making it positive) and normalizing with respect to all possible paths (summing to 1 over all paths). We define $\mathcal{P}$ the set of possible tag path for a sentence. Taking the $log(.)$ leads to the following conditional log-probability:

$$\log([t]_1^N, [w]_1^N, \theta) = S([w]_1^N, [t]_1^N, \theta) - \underset{\forall [u]_1^N \in \mathcal{T}}{\operatorname{logadd}} S([w]_1^N, [u]_1^N, \theta)) \tag{4}$$

where we adopt the notation $\operatorname{logadd}_{z_n} = \log\left(\sum_i e^{z_i}\right)$

Computing the log-likelihood efficiently is not straightforward, as the number of terms in the logadd grows exponentially with the length of the sentence. Fortunately, it can be computed in linear time with the Forward algorithm, which derives a recursion similar to the Viterbi algorithm (see Rabiner [26]).

4.2 Document-level Approach

The document-level approach is a document classifier. Embeddings of each word in a sentence are fed to a convolutional layer followed a max layer and ending by a class prediction layer. Some linear and non-linear layers can be added between the first and the last layer.

Max Layer The size of the output will depend on the number of words in the document fed to the network. Local feature vectors extracted by the convolutional layer have to be combined to obtain a global feature vector, with a fixed size independent of the document length, in order to apply subsequent standard affine layers. We used a max approach, which forces the network to capture the most useful local features produced by the convolutional layer, for the task at hand. Given a matrix f_θ^{l-1} output by a convolutional layer $l-1$, the Max layer l outputs a vector f_θ^l:

$$[f_\theta^l]_i = max[f_\theta^{l-1}]_{i,t} \quad 1 \leq i \leq n_{hu}^{l-1} \tag{5}$$

The hyper-parameter n_{hu}^{l-1} is the *number of hidden units* of the convolution layer. This fixed sized global feature vector can be then fed to standard affine network layers.

Class prediction As any classical neural network, the architecture performs several matrix-vector operations on its inputs, interleaved with some non-linear transfer function. The output size of the last layer is equal to the number of possible classes for the task of interest. Each output can be then interpreted as a score of the corresponding class (given the input of the network).

Training The network is trained by maximizing a likelihood over the training data, using stochastic gradient ascent. We note $[f_\theta]_y$ the y^{th} output of the network and θ all the trainable parameters. Using a training set T, we want to maximize the following log-likelihood with respect to θ:

$$\theta \leftarrow \sum_{(x,y)\in T} \log p(y|x,\theta)$$

where x corresponds to a training document and y represents the corresponding class. The probability $p(y|x,\theta)$ is computed from the outputs of the neural network by adding a softmax operation over all the classes:

$$p(i|x,\theta) = \frac{e^{[f_\theta]_i}}{\sum_j e^{[f_\theta]_j}}$$

We can express the log-likelihood for one training example (x, y) as follows:

$$\log p(y|x,\theta) = [f_\theta]_y - \log(\sum_j e^{[f_\theta]_j})$$

4.3 Embedding Fine-Tuning

As seen in section 3, the process to compute generic word embedding is quite straightforward. These embeddings can then be used as features for supervised NLP systems and help to improve the general performance [7, 8, 9]. However, most of these systems cannot tune these embeddings as they are not structurally able to. By leveraging the deep architecture of our system, we can define a lookup-table layer initialized with existing embeddings as the first layer of the network.

Lookup-Table Layer We consider a fixed-sized word dictionary $\mathcal{D}$. Given a sequence of N words $w_1, w_2, \dots, w_N$, each word $w_n \in W$ is first embedded into a d_{wrd}-dimensional vector space, by applying a lookup-table operation:

$$LT_W(w_n) = W \left(\begin{array}{ccc} 0,\dots,0, & \underset{\text{at index } w_n}{1} & ,0,\dots,0 \end{array} \right) = \langle W \rangle_{w_n} \tag{6}$$

where the matrix $W \in \mathbb{R}^{d_{wrd} \times |\mathcal{D}|}$ represents the embeddings to be tuned in this lookup layer. $\langle W \rangle_{w_n} \in \mathbb{R}^{d_{wrd}}$ is the w^{th} column of W and d_{wrd} is the word vector size. Given any sequence of N words $[w]_1^N$ in $\mathcal{D}$, the lookup table layer applies the same operation for each word in the sequence, producing the following output matrix:

$$LT_W([w]_1^N) = \begin{pmatrix} \langle W \rangle^1_{[w]_1} & \langle W \rangle^1_{[w]_2} & \dots & \langle W \rangle^1_{[w]_N} \end{pmatrix} \tag{7}$$

Training Given a task of interest, a relevant representation of each word is then given by the corresponding lookup table feature vector, which is trained by backpropagation, starting from existing embeddings.

5 Experimental Setup

We evaluate the quality of our embeddings obtained on a large corpora of unlabeled text by comparing their performance against the CW [5], Turian [7], HLBL [10], and LR-MVL [11] embeddings on NER and movie review tasks. We also show that the general performance can be improved for these tasks by fine-tuning the word embeddings.

5.1 Building Word Representation over Large Corpora

Our English corpus is composed of the entire English Wikipedia[1] (where all MediaWiki markups have been removed), the Reuters corpus and the Wall Street Journal (WSJ) corpus. We chose to consider lower case words to limit the number of words in the vocabulary. Additionally, all occurrences of sequences of numbers within a word are replaced with the string "NUMBER". The resulting text was tokenized using the Stanford tokenizer[2]. The data set contains about 1,652 million words. As vocabulary we considered all the words within our corpus which appear at least one hundred times. This results in a 178,080 words vocabulary. To build the co-occurence matrix we used only the 10,000 most frequent words within our vocabulary as context words. Each word is represented in a 50-dimensional vector as the other embeddings in the literature. The resulting

[1]Available at http://download.wikimedia.org. We took the May 2012 version.

[2]Available at http://nlp.stanford.edu/software/tokenizer.shtml

embeddings will be referred as H-PCA in the following sections. To highlight the importance of the Hellinger distance, we also computed the co-occurence probabilities matrix without the square root. The resulting embeddings after PCA are called E-PCA.

5.2 Existing Available Word Embeddings

We choose to compare our H-PCA's embeddings with the following publicly available embeddings:

- **LR-MVL**[3]: it covers 300,000 words with 50 dimensions for each word. They were trained on the RCV1 corpus using the Low Rank Multi-View Learning method. We only used their context oblivious embeddings coming from the eigenfeature dictionary.
- **CW**[4]: it covers 130,000 words with 50 dimensions for each word. They were trained for about two months, over Wikipedia, using a neural network language model approach.
- **Turian**[5]: it covers 268,810 words with 25, 50, 100 or 200 dimensions for each word. They were trained on the RCV1 corpus using the same system as the CW embeddings but with different parameters. We used only the 50 dimensions.
- **HLBL**[5] : it covers 246,122 words with 50 or 100 dimensions for each word. They were trained on the RCV1 corpus using a Hierarchical Log-Bilinear Model. We used only the 50 dimensions.

5.3 Supervised Evaluation Tasks

Using word embeddings as feature proved that it can improve the generalization performance on several NLP tasks [7, 8, 9]. Using our word embeddings, we thus trained the sentence-level architecture described in section 4.1 on a NER task.

Named Entity Recognition (NER) It labels atomic elements in the sentence into categories such as "PERSON" or "LOCATION". The CoNLL 2003 setup[6] is a NER benchmark data set based on Reuters data. The contest provides training, validation and testing sets. The networks are fed with two raw features: word embeddings and a capital letter feature. The "caps" feature tells if each word was in lowercase, was all uppercase, had first letter capital, or had at least one non-initial capital letter. No other feature has been used to tune the models. This is a main difference with other systems which usually use more features as POS tags, prefixes and suffixes or gazetteers. The optimal hyper-parameters were a window of 5 words for the convolutional layer and 300 hidden units. As benchmark system, we report the system of Ando et al. [27] which reached 89.31% F1 with a semi-supervised approach and less specialized features than CoNLL 2003 challengers.

The NER evaluation task is mainly syntactic. We wish to evaluate whether our word embeddings can also capture semantic. We thus trained the document-level architecture described in section 4.2 with movie reviews to predict whether they are positives or negatives.

IMDB Review Dataset We used a collection of 50,000 reviews from IMDB[7]. It allows no more than 30 reviews per movie. It contains an even number of positive and negative reviews, so randomly guessing yields 50% accuracy. Only highly polarized reviews have been considered. A negative review has a score ≤ 4 out of 10, and a positive review has a score ≥ 7 out of 10. It has been evenly divided into training and test sets (25,000 reviews each). For this task, we only used the word embeddings as features. The optimal hyper-parameters for the network were a window of 5 words for the convolutional layer and 1,000 hidden units. As benchmark system, we report the system of Maas et al. [24] which reached 88.90% accuracy with a mix of unsupervised and supervised techniques to learn word vectors capturing semantic term-document information as well as rich sentiment content.

[3]Available at http://www.cis.upenn.edu/ ungar/eigenwords/

[4]From SENNA: http://ml.nec-labs.com/senna/

[5]Available at http://metaoptimize.com/projects/wordreprs/

[6]http://www.cnts.ua.ac.be/conll2003/ner/

[7]Available at http://www.andrew-maas.net/data/sentiment

5.4 Results

Approach	Fixed Embedding	Tuned Embedding
Benchmark System	89.31	
	Non-Linear Approach	
H-PCA	87.91	89.37
E-PCA	79.91	86.17
LR-MVL	76.80	82.65
CW	88.09	88.93
Turian	85.80	87.28
HLBL	83.42	85.76
	Linear Approach	
H-PCA	84.39	88.04
E-PCA	67.95	77.48
LR-MVL	65.52	80.02
CW	84.52	86.90
Turian	83.30	86.60
HLBL	80.19	84.90

Table 1: Comparison in performance on NER task with different embeddings. Results are reported in F1 score.

Results summarized in Table 1 reveal that performance on NER task can be as good with word embeddings from a word co-occurence matrix decomposition as with a neural network language model trained for weeks. The best F1 score (89.37), which outperforms the benchmark system, is indeed obtained using the H-PCA tuned embeddings. Results for the movie review task in Table 2 show that H-PCA's embeddings also perform as well as all the other embeddings on the movie review task. This task confirms the importance of embedding fine-tuning for NLP tasks with a high semantic component. We note that our tuned embeddings leads to a performance gain of about 1% or 2% for NER, while the gain is between about 4% and 8% for the movie review. We thus show in Table 3 that the embeddings after fine-tuning give a higher rank to words that are related to the task of interest which is movie-sentiment-based relations in this case. It is worth mentioning that on both tasks H-PCA's embeddings outperform the LR-MVL's and E-PCA's embeddings which are also obtained from a spectral method, demonstrating the value of the Hellinger distance. On the other hand, they give similar or slightly better results than the embeddings obtained via deep-learning architectures. We also report results with a linear version of our neural networks (instead of having linear layers interleaved with a non-linearity). We note that this approach performs as well as the non-linear approach for the movie review task. However, having non-linearity helps for NER.

Approach	Fixed Embedding	Tuned Embedding
Benchmark System	88.90	
	Non-Linear Approach	
H-PCA	83.88	89.86
E-PCA	68.08	89.58
LR-MVL	81.28	89.19
CW	85.01	89.58
Turian	84.88	89.99
HLBL	84.93	89.24
	Linear Approach	
H-PCA	84.01	89.78
E-PCA	67.65	89.55
LR-MVL	81.28	89.03
CW	88.06	89.85
Turian	83.90	89.41
HLBL	84.71	89.39

Table 2: Comparison in performance on movie review task with different embeddings. Results are reported in classification accuracy.

BORING		**BAD**		**AWESOME**	
before	*after*	*before*	*after*	*before*	*after*
SAD	CRAP	HORRIBLE	TERRIBLE	SPOOKY	TERRIFIC
SILLY	LAME	TERRIBLE	STUPID	AWFUL	TIMELESS
SUBLIME	MESS	DREADFUL	BORING	SILLY	FANTASTICE
FANCY	STUPID	UNFORTUNATE	DULL	SUMMERTIME	LOVELY
SOBER	DULL	AMAZING	CRAP	NASTY	FLAWLESS
TRASH	HORRIBLE	AWFUL	WRONG	MACABRE	MARVELOUS
LOUD	RUBBISH	MARVELOUS	TRASH	CRAZY	EERIE
RIDICULOUS	SHAME	WONDERFUL	SHAME	ROTTEN	LIVELY
RUDE	AWFUL	GOOD	KINDA	OUTRAGEOUS	FANTASY
MAGIC	ANNOYING	FANTASTIC	JOKE	SCARY	SURREAL

Table 3: Set of words with their 10 nearest neighbors before and after fine-tuning for the movie review task (using the Euclidean metric). H-PCA embeddings are used here.

6 Conclusion

We have demonstrated that appealing word embeddings can be obtained by computing a *Hellinger PCA* of the word co-occurence matrix. While a neural network language model can be painful and long to train, we can get a word co-occurence matrix by simply counting words over a large corpus. The resulting embeddings give similar results on NLP tasks even from a $N \times 10,000$ word co-occurence matrix. It reveals that having a significant but not too large set of common words seems sufficient for capturing most of the syntactic and semantic characteristics of words. As PCA of a $N \times 10,000$ matrix is really fast and not memory consuming, our method gives an interesting and practical alternative to neural language models for generating word embeddings. However, we showed that deep-learning architectures is a good choice for supervised NLP tasks as we outperform benchmark results on NER and movie review tasks. By leveraging the deep architecture of our systems, we also showed that existing embeddings can be fine-tuned to a specific task which leads to improve general performance for this task. Finally, deep learning is not that necessary for generating word embeddings but it can be extremely convenient for adapting these embeddings. We then encourage people wishing to play with NLP tasks to use our word embeddings[8] in a deep-learning framework.

Acknowledgments

This work was partly supported by the HASLER foundation through the grant "Information and Communication Technology for a Better World 2020" (SmartWorld) and by NEC Laboratories America, Princeton, USA.

References

[1] P. F. Brown, P. V. deSouza, R. L. Mercer, V. J. D. Pietra, and J C. Lai. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479, 1992.

[2] H. Schütze. Distributional part-of-speech tagging. In *Proceedings of the Association for Computational Linguistics (ACL)*, pages 141–148. Morgan Kaufmann Publishers Inc., 1995.

[3] T. Koo, X. Carreras, and M. Collins. Simple semi-supervised dependency parsing. In *Proceedings of the Association for Computational Linguistics (ACL)*, pages 595–603, 2008.

[4] L. Ratinov and D. Roth. Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL)*, pages 147–155. Association for Computational Linguistics, 2009.

[5] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *International Conference on Machine Learning, ICML*, 2008.

[8]Available soon online

[6] F. Huang and A. Yates. Distributional representations for handling sparsity in supervised sequence-labeling. In *Proceedings of the Association for Computational Linguistics (ACL)*, pages 495–503. Association for Computational Linguistics, 2009.

[7] J. Turian, L. Ratinov, and Y. Bengio. Word representations: A simple and general method for semi-supervised learning. In *ACL*, 2010.

[8] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537, 2011.

[9] Y. Chen, B. Perozzi, R. Al-Rfou', and S. Skiena. The expressive power of word embeddings. *CoRR*, abs/1301.3226, 2013.

[10] A. Mnih and G. Hinton. A Scalable Hierarchical Distributed Language Model. In *Advances in Neural Information Processing Systems*, volume 21, 2008.

[11] P. S. Dhillon, D. Foster, and L. Ungar. Multi-view learning of word embeddings via cca. In *Advances in Neural Information Processing Systems (NIPS)*, volume 24, 2011.

[12] I. Labutov and H. Lipson. Re-embedding words. In *ACL*, 2013.

[13] T. K. Landauer. On the computational basis of learning and cognition: Arguments from lsa. In N. Ross, editor, *The psychology of learning and motivation*, volume 41, pages 43–84. Academic Press, San Francisco, CA, 2002.

[14] G. E. Hinton. Learning distributed representations of concepts. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pages 1–12. Hillsdale, NJ: Erlbaum, 1986.

[15] J. B. Pollack. Recursive distributed representations. *Artificial Intelligence*, 46:77–105, 1990.

[16] J. L. Elman. Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, 7:195–225, 1991.

[17] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, March 2003. ISSN 1532-4435.

[18] T. Mikolov, M. Karafit, L. Burget, J. ernock, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Proceedings of the 11th Annual Conference of the International Speech Communication Association (INTERSPEECH 2010)*, volume 2010, pages 1045–1048. International Speech Communication Association, 2010. ISBN 978-1-61782-123-3.

[19] L. Wittgenstein. *Philosophical Investigations*. Blackwell, 1953.

[20] P. D. Turney and P. Pantel. From frequency to meaning: Vector space models of semantics. *CoRR*, abs/1003.1141, 2010.

[21] W. Lowe. *Towards a theory of semantic space*, pages 576–581. 2001.

[22] T. K. Landauer and S. T. Dumais. A solution to Plato's problem: The Latent Semantic Analysis theory of the acquisition, induction, and representation of knowledge. *Psychological Review*, 104:211–240, 1997.

[23] J. J. Väyrynen and T. Honkela. Word category maps based on emergent features created by ICA. In Heikki Hyötyniemi, Pekka Ala-Siuru, and Jouko Seppänen, editors, *Proceedings of the STeP'2004 Cognition + Cybernetics Symposium*, number 19 in Publications of the Finnish Artificial Intelligence Society, pages 173–185. Finnish Artificial Intelligence Society, 2004.

[24] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. Learning word vectors for sentiment analysis. In *ACL*, pages 142–150, 2011.

[25] J. R. Firth. A synopsis of linguistic theory 1930-55. 1952-59:1–32, 1957.

[26] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, pages 257–286, 1989.

[27] R. K. Ando, T. Zhang, and P. Bartlett. A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research*, 6:1817–1853, 2005.