Word Embeddings through Hellinger PCA

Rémi Lebret
Idiap Research Institute
Rue Marconi 19, CP 592
1920 Martigny, Switzerland
remi@lebret.ch

Ronan Collobert
Idiap Research Institute
Rue Marconi 19, CP 592
1920 Martigny, Switzerland
ronan@collobert.com

Abstract

Word embeddings resulting from neural language models have been shown to be a great asset for a large variety of NLP tasks. However, such architecture might be difficult and time-consuming to train. 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 named entity recognition 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 has always generated much interest for linguists. Popular approaches such as Brown clustering algorithm (Brown et al., 1992) have been used with success in a wide variety of NLP tasks (Schütze, 1995; Koo et al., 2008; Ratinov and Roth, 2009). Those word embeddings are often seen as a low dimensional-vector space where the dimensions are 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 (Collobert and Weston, 2008; Huang and Yates, 2009; Turian et al., 2010; Collobert et al., 2011). 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 (mostly 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 (Collobert and Weston, 2008), Turian (Turian et al., 2010), HLBL (Mnih and Hinton, 2008) embeddings, which come from deep architectures and the LR-MVL (Dhillon et al., 2011) embeddings, which also come from a spectral method on several NLP tasks.

We claim that, assuming an appropriate metric, 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 (Turian et al., 2010; Collobert et al., 2011; Dhillon et al., 2011). For supervised tasks relying more on the semantic aspect as sentiment classification, it is usually helpful to adapt the existing embeddings to improve performance (Labutov and Lipson, 2013). We show in this paper that such embedding specialization can be easily done via neural network 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 (Landauer, 2002), it seems quite important to leverage word order to capture all the semantic information. Connectionist approaches have therefore been proposed to develop distributed representations which encode the struc-

Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics, pages 482–490,
Gothenburg, Sweden, April 26-30 2014.

tural relationships between words (Hinton, 1986; Pollack, 1990; Elman, 1991). More recently, a neural network language model was proposed in Bengio et al. (2003) where word vector representations are simultaneously learned along with a statistical language model. This architecture inspired other authors: Collobert and Weston (2008) designed a neural language model which eliminates the linear dependency on vocabulary size, Mnih and Hinton (2008) proposed a hierarchical linear neural model, Mikolov et al. (2010) 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 (Wittgenstein, 1953). Using the word co-occurrence statistics is thus a natural choice to embed similar words into a common vector space (Turney and Pantel, 2010). Common approaches calculate the frequencies, apply some transformations (tf-idf, PPMI), reduce the dimensionality and calculate the similarities (Lowe, 2001). Considering a fixed-sized word vocabulary $\mathcal{D}$ and a set of words $\mathcal{W}$ to embed, the co-occurence matrix C is of size $|\mathcal{W}|\times|\mathcal{D}|$. C is then vocabulary size-dependent. One can apply a dimensionality reduction operation to C leading to $\bar{C} \in \mathbb{R}^{|\mathcal{W}|\times d}$, where $d \ll |\mathcal{D}|$. Dimensionality reduction techniques such as Singular Valued Decomposition (SVD) are widely used (e.g. LSA (Landauer and Dumais, 1997), ICA (Väyrynen and Honkela, 2004)). However, word co-occurence statistics are discrete distributions. An information theory measure such as the Hellinger distance seems to be more appropriate than the Euclidean distance over a discrete distribution space. In this paper we will compare the Hellinger PCA against the classical Euclidean PCA and the Low Rank Multi-View Learning (LR-MVL) method, which is another spectral method based on Canonical Correlation Analysis (CCA) to learn word embeddings (Dhillon et al., 2011).

It has been shown that using word embeddings as features helps to improve general performance on many NLP tasks (Turian et al., 2010). 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. (2011) proposed a model for jointly capturing semantic and sentiment components of words into vector spaces. More recently, Labutov and Lipson (2013) presented a method which takes existing embeddings and, by using some labeled data, re-embed them in the same space. They showed that these new embeddings can be better predictors in a supervised task. In this paper, we consider word embedding-based linear and non-linear models for two NLP supervised tasks: Named Entity Recognition and IMDB movie review. We analyze the effect of fine-tuning existing embeddings over each task of interest.

3 Spectral Method for Word Embeddings

A NNLM learns which words among the vocabulary are likely to appear after a given 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 (Turney and Pantel, 2010).

3.1 Word co-occurence statistics

"You shall know a word by the company it keeps" (Firth, 1957). 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 context word $w \in \mathcal{D}$ occurs after a sequence of words T:

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

where $n(w,T)$ is the number of times each context word w occurs after the sequence T. The size of T can go from 1 to t words. 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 finally built 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 metrics) 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 $P = (p_1, \ldots, p_k)$ and $Q = (q_1, \ldots, q_k)$, the Hellinger distance is formally defined as:

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

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

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

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 probability matrix to represent words in a lower dimensional space while minimizing the reconstruction error according to the Hellinger distance.

4 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 linear and non-linear classification models followed by a CRF-type sentence tag inference. We chose here neural networks as classifiers.

Sliding window Context is crucial to characterize word meanings. We thus consider n context words around each word x_t to be tagged, leading to a window of $N = (2n + 1)$ words $[x]^t = (x_{t-n}, \ldots, x_t, \ldots, x_{t+n})$. As each word is embedded into a d_{wrd}-dimensional vector, it results a $d_{wrd} \times N$ vector representing a window of N words, which aims at characterizing the middle word x_t in this window. Given a complete sentence of T words, we can obtain for each word a context-dependent representation by sliding over all the possible windows in the sentence. A same linear transformation is then applied on each window for each word to tag:

$$g([x]^t) = W[x]^t + b\,, \qquad (4)$$

where $W \in \mathbb{R}^{M \times d_{wrd}N}$ and $b \in \mathbb{R}^M$ are the parameters, with M the number of classes. Alternatively, a one hidden layer non-linear network can be considered:

$$g([x]^t) = Wh(U[x]^t) + b\,, \qquad (5)$$

where $U \in \mathbb{R}^{n_{hu} \times d_{wrd}N}$, with n_{hu} the number of hidden units and $h(.)$ a transfer function.

CRF-type inference There exists strong dependencies between tags in a sentence: some tags cannot follow other tags. To take the sentence structure into account, we want to encourage valid paths of tags during training, while discouraging all other paths. Considering the matrix of scores outputs by the network, we train a simple conditional random field (CRF). At inference time, given a sentence to tag, the best path which minimizes the sentence score is inferred with the Viterbi algorithm. More formally, we denote θ all the trainable parameters of the network and $f_\theta([x]_1^T)$ the matrix of scores. The element $[f_\theta]_{i,t}$ of the matrix is the score output by the network for the sentence $[x]_1^T$ and the i^{th} tag, at the t^{th} word. We introduce a transition score $[A]_{i,j}$ for jumping from i to j tags in successive words, and an initial score $[A]_{i,0}$ for starting from the i^{th} tag. As the transition scores are going to be trained, we define $\tilde{\theta} = \theta \cup \{[A]_{i,j} \forall i, j\}$. The score of a sentence $[x]_1^T$ along a path of tags $[i]_1^T$ is then given by the sum of transition scores and networks scores:

$$s([x]_1^T, [i]_1^T, \tilde{\theta}) = \sum_{t=1}^{T} (A_{[i]_{t-1}, [i]_t} + [f_\theta]_{[i]_t, t})\,. \qquad (6)$$

We normalize this score over all possible tag paths $[j]_1^T$ using a softmax, and we interpret the resulting ratio as a conditional tag path probability. Taking the log, the conditional probability of the true path $[y]_1^T$ is therefore given by:

$$\log p([y]_1^T, [x]_1^T, \tilde{\theta}) = s([x]_1^T, [y]_1^T, \tilde{\theta}) - \underset{\forall [j]_1^T}{\text{logadd}}\, s([x]_1^T, [j]_1^T, \tilde{\theta}) \,, \tag{7}$$

where we adopt the notation

$$\underset{i}{\text{logadd}}\, z_i = \log\,(\sum_i e^{z_i})\,. \tag{8}$$

Computing the log-likelihood efficiently is not straightforward, as the number of terms in the logadd grows exponentially with the length of the sentence. It can be computed in linear time with the Forward algorithm, which derives a recursion similar to the Viterbi algorithm (see Rabiner (1989)). We can thus maximize the log-likelihood over all the training pairs $([x]_1^T, [y]_1^T)$ to find, given a sentence $[x]_1^T$, the best tag path which minimizes the sentence score (6):

$$\underset{[j]_1^T}{\text{argmax}}\, s([x]_1^T, [j]_1^T, \tilde{\theta})\,. \tag{9}$$

In contrast to classical CRF, all parameters θ are trained in a end-to-end manner, by backpropagation through the Forward recursion, following Collobert et al. (2011).

4.2 Document-level Approach

The document-level approach is a document binary classifier, with classes $y \in \{-1, 1\}$. For each document, a set of (trained) filters is applied to the sliding window described in section 4.1. The maximum value obtained by the i^{th} filter over the whole document is:

$$\max_t \Big[w_i [x]^t + b_i\Big]_{i,t} \quad 1 \leq i \leq n_{filter}\,. \tag{10}$$

It can be seen as a way to measure if the information represented by the filter has been captured in the document or not. We feed all these intermediate scores to a linear classifier, leading to the following simple model:

$$f_\theta(x) = \boldsymbol{\alpha} \max_t \Big[W[x]^t + b\Big]\,. \tag{11}$$

In the case of movie reviews, the i^{th} filter might capture positive or negative sentiment depending on the sign of α_i. As in section 4.1, we will also consider a non-linear classifier in the experiments.

Training The neural network is trained using stochastic gradient ascent. We denote θ all the trainable parameters of the network. Using a training set $\mathcal{T}$, we minimize the following soft margin loss function with respect to θ:

$$\theta \leftarrow \sum_{(x,y)\in\mathcal{T}} \log\Big(1 + e^{-y f_\theta(x)}\Big)\,. \tag{12}$$

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 (Turian et al., 2010; Collobert et al., 2011; Chen et al., 2013). 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 $w_1, w_2, \ldots, w_N$, each word $w_n \in W$ is first embedded into a d_{wrd}-dimensional vector space, by applying a lookup-table operation:

$$\begin{aligned} LT_W(w_n) &= W \Big(0, \ldots, \underset{\text{at index } w_n}{1}, \ldots, 0 \Big) \\ &= \langle W \rangle_{w_n}\,, \end{aligned} \tag{13}$$

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:

$$LT_W([w]_1^N) = \Big(\langle W \rangle^1_{[w]_1} \;\; \ldots \;\; \langle W \rangle^1_{[w]_N} \Big)\,. \tag{14}$$

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. Word representations are initialized with 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 (Collobert and Weston, 2008), Turian (Turian et al., 2010), HLBL (Mnih and Hinton, 2008), and LR-MVL (Dhillon et al., 2011) 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 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. To get embeddings for words, we needed to only consider sequences T of $t = 1$ word. After PCA, each word can be represented in any n-dimensional vector (with $n \in \{1, \ldots, 10000\}$). We chose to embed words in a 50-dimensional vector, which is the common dimension among the other embeddings in the literature. The resulting embeddings will be referred as H-PCA in the following sections. To highlight the importance of the Hellinger distance, we also computed the PCA of the co-occurence probability matrix with respect to the Euclidean metric. The resulting embeddings are denoted E-PCA.

Computational cost The Hellinger PCA is very fast to compute. We report in Table 1 the time needed to compute the embeddings described above. For this benchmark we used Intel i7 3770K 3.5GHz CPUs. As the computation of the covariance matrix is highly parallelizable, we report results with 1, 100 and 500 CPUs. The Eigendecomposition of the C matrix has been computed with the SSYEVR LAPACK subroutine on one CPU. We compare completion times for 1,000 and 10,000 eigenvectors. Finally, we report completion times to generate the emdeddings by linear projection using 50, 100 and 200 eigenvectors. Although the linear projection is already quite fast on only one CPU, this operation can also be computed in parallel. Those results show that the Hellinger PCA can generate about 200,000 embeddings in about three minutes with a cluster of 100 CPUs.

	time (s)		
# of CPUs	1	100	500
Covariance matrix	9930	99	20
1,000 Eigenvectors	72	-	-
10,000 Eigenvectors	110	-	-
50D Embeddings	20	0.2	0.04
100D Embeddings	29	0.29	0.058
200D Embeddings	67	0.67	0.134
Total for 50D	10,022	171.2	92.04

Table 1: Benchmark of the experiment. Times are reported in seconds.

5.2 Existing Available Word Embeddings

We 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.

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

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

[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/

- **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.

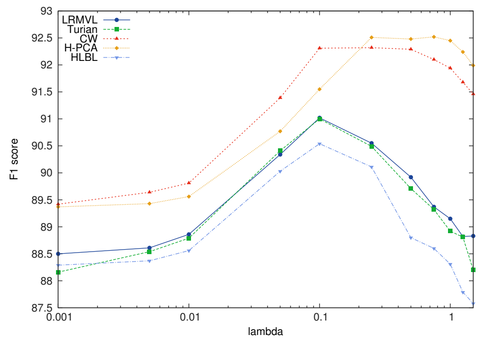
5.3 Supervised Evaluation Tasks

Using word embeddings as feature proved that it can improve the generalization performance on several NLP tasks (Turian et al., 2010; Collobert et al., 2011; Chen et al., 2013). Using our word embeddings, we thus trained the sentence-level architecture described in section 4.1 on a NER task.

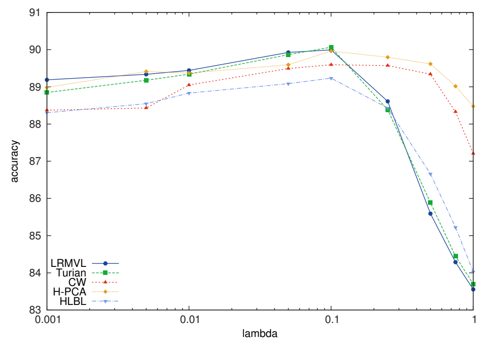
Named Entity Recognition (NER) It labels atomic elements in the sentence into categories such as “PERSON” or “LOCATION”. The CoNLL 2003 setup[6] is a NER benchmark data set based on Reuters data. The contest provides training, validation and testing sets. The networks are fed with two raw features: word 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. Hyper-parameters were tuned on the validation set. We selected $n = 2$ context words leading to a window of 5 words. We used a special “PADDING” word for context at the beginning and the end of each sentence. For the non-linear model, the number of hidden units was 300. As benchmark system, we report the system of Ando et al. (2005), 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. As we wish to evaluate whether our word embeddings can also capture semantic, we trained the document-level architecture described in section 4.2 over a movie review task.

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. We perform a simple cross-validation on the training set to choose the optimal hyper-parameters. The network had a window of 5 words and $n_{filter} = 1000$ filters. As benchmark system, we report the system of Maas et al. (2011), 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.



(a) NER validation set.



(b) IMDB review dataset.

Figure 1: Effect of varying the normalization factor λ with a non-linear approach and fine-tuning.

5.4 Embeddings Normalization

Word embeddings are continuous vector spaces that are not necessarily in a bounded range. To avoid saturation issues in the network architectures, embeddings need to be properly normalized. Considering the matrix of word embeddings E, the normalized embeddings are:

$$\tilde{E} = \frac{\lambda(E - \bar{E})}{\sigma(E)} \tag{15}$$

[6] http://www.cnts.ua.ac.be/conll2003/ner/
[7] Available at http://www.andrew-maas.net/data/sentiment

where $\bar{E}$ is the mean of the embeddings, $\sigma(E)$ is the standard deviation of the embeddings and λ is a normalization factor. Figure 1 shows the effect of λ on both supervised tasks. The embeddings normalization depends on the type of the network architecture. In the document-level approach, best results are obtained with $\lambda = 0.1$ for all embeddings, while a normalization factor set to 1 is better for H-PCA's embeddings in the sentence-level approach. These results show the importance of applying the right normalization for word embeddings.

5.5 Results

H-PCA's embeddings Results summarized in Table 2 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 scores are indeed obtained using the H-PCA tuned embeddings. Results for the movie review task in Table 3 show that H-PCA's embeddings also perform as well as all the other embeddings on the movie review task. It is worth mentioning that on both tasks, H-PCA's embeddings outperform the E-PCA's embeddings, demonstrating the value of the Hellinger distance. When the embeddings are not tuned, the CW's embeddings slightly outperform the H-PCA's embeddings on NER task. The performance difference between both fixed embeddings on the movie review task is about 3%. Embeddings from the CW neural language model seems to capture more semantic information but we showed that this lack of semantic information can be offset by fine-tuning.

Embeddings fine-tuning We note that tuning the embeddings by backpropagation increases the general performance on both NER and movie review tasks. The increase is, in general, higher for the movie review task, which reveals the importance of embedding fine-tuning for NLP tasks with a high semantic component. We show in Table 4 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.

Linear vs nonlinear model We also report results with a linear version of our neural networks. Having non-linearity helps for NER. It seems important to extract non-linear features for such a task. However, we note that the linear approach

Approach	**Fixed**	**Tuned**
Benchmark	89.31	
	Non-Linear Approach	
H-PCA	87.91 ± 0.17	89.16 ± 0.09
E-PCA	84.28 ± 0.15	87.09 ± 0.12
LR-MVL	86.83 ± 0.20	87.38 ± 0.07
CW	88.14 ± 0.21	88.69 ± 0.16
Turian	86.26 ± 0.13	87.35 ± 0.12
HLBL	83.87 ± 0.25	85.91 ± 0.17
	Linear Approach	
H-PCA	84.64 ± 0.11	87.97 ± 0.09
E-PCA	78.15 ± 0.15	85.99 ± 0.09
LR-MVL	82.27 ± 0.14	86.83 ± 0.17
CW	84.50 ± 0.19	86.84 ± 0.08
Turian	83.33 ± 0.07	86.79 ± 0.11
HLBL	80.31 ± 0.11	85.06 ± 0.13

Table 2: Performance comparison on NER task with different embeddings. The first column is results with the original embeddings. The second column is results with embeddings after fine-tuning for this task. Results are reported in F1 score (mean $\pm$ standard deviation of ten training runs with different initialization).

Approach	**Fixed**	**Tuned**
Benchmark	88.90	
	Non-Linear Approach	
H-PCA	84.20 ± 0.16	89.89 ± 0.09
E-PCA	74.85 ± 0.12	89.70 ± 0.06
LR-MVL	85.33 ± 0.14	90.06 ± 0.09
CW	87.54 ± 0.27	89.77 ± 0.05
Turian	85.33 ± 0.10	89.99 ± 0.05
HLBL	85.51 ± 0.14	89.58 ± 0.06
	Linear Approach	
H-PCA	84.11 ± 0.05	89.90 ± 0.10
E-PCA	73.27 ± 0.16	89.62 ± 0.05
LR-MVL	84.37 ± 0.16	89.77 ± 0.09
CW	87.62 ± 0.24	89.92 ± 0.07
Turian	84.44 ± 0.13	89.66 ± 0.10
HLBL	85.34 ± 0.10	89.64 ± 0.05

Table 3: Performance comparison on movie review task with different embeddings. The first column is results with the original embeddings. The second column is results with embeddings after fine-tuning for this task. Results are reported in classification accuracy (mean $\pm$ standard deviation of ten training runs with different initialization).

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	FANTASTIC
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 4: Set of words with their 10 nearest neighbors before and after fine-tuning for the movie review task (using the Euclidean metric in the embedding space). H-PCA's embeddings are used here.

performs as well as the non-linear approach for the movie review task. Our linear approach captures all the necessary sentiment features to predict whether a review is positive or negative. It is thus not surprising that a bag-of-words based method can perform well on this task (Wang and Manning, 2012). However, as our method takes the whole review as input, we can extract *windows* of words having the most discriminative power: it is a major advantage of our method compared to conventional bag-of-words based methods. We report in Table 5 some examples of windows of words extracted from the most discriminative filters α_i (positive and negative). Note that there is about the same number of positive and negative filters after learning.

α_i	$[x]^t$
-	the worst film this year very worst film i 've very worst movie i 've
-	watch this unfunny stinker . , extremely unfunny drivel come , this ludicrous script gets
-	it was pointless and boring it is unfunny . unfunny film are awful and embarrassing
+	both really just wonderful . . a truly excellent film . a really great film
+	excellent film with great performances excellent film with a great excellent movie with a stellar
+	incredible . just incredible . performances and just amazing . one was really great .

Table 5: The top 3 positive and negative filters $\alpha_i w_i$ and their respective top 3 windows of words $[x]^t$ within the whole IMDB review dataset.

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 computed with only one word of context. 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 is an interesting framework to fine-tune embeddings over specific NLP tasks. Our H-PCA's embeddings are available online, here: `http://www.lebret.ch/words/`.

Acknowledgments

This work was supported by the HASLER foundation through the grant "Information and Communication Technology for a Better World 2020" (SmartWorld).

References

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

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

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

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

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

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

P. S. Dhillon, D. Foster, and L. Ungar. 2011. Multiview learning of word embeddings via CCA. In *Advances in Neural Information Processing Systems (NIPS)*, volume 24.

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

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

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

F. Huang and A. Yates. 2009. 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.

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

I. Labutov and H. Lipson. 2013. Re-embedding words. In *ACL*.

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

T. K. Landauer. 2002. 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.

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

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

T. Mikolov, M. Karafiat, L. Burget, J. Cernocky, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model.

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

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

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

L. Ratinov and D. Roth. 2009. 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.

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

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

P. D. Turney and P. Pantel. 2010. From frequency to meaning: Vector space models of semantics. *J. Artif. Int. Res.*, 37(1):141–188, January.

J. J. Väyrynen and T. Honkela. 2004. Word category maps based on emergent features created by ICA. In *Proceedings of the STeP'2004 Cognition + Cybernetics Symposium*.

S Wang and C. D. Manning. 2012. Baselines and bigrams: Simple, good sentiment and topic classification. ACL '12.

L. Wittgenstein. 1953. *Philosophical Investigations*. Blackwell.