
# Compressed Sensing Project

*A compressed sensing view of unsupervised text embeddings*

Salomé Do, Lucas Zanini

March, 22nd 2019

# Introduction

Text documents have been, until recently, an underexploited source of information due to their complex, non-numerical structure. Low level natural language processing (NLP) tasks as text classification, sequence tagging (POS-tagging, named entity recognition, ...) and parsing have long been tackled through bag-of-words methods and hidden markov models. The application of deep learning to NLP tasks (started around 2010s) has opened new, complementary and effective ways to solve these problems, eventually leading to core innovations in high-level tasks such as question answering or language generation. Using feed-forward networks to generate low-dimensional representation of sparse, high dimensional word vectors; recurrent neural networks as LSTMs to fit the sequential nature of sentences; and later on attention mechanims, deep learning techniques have set new baselines on traditional benchmarks. However, the efficiency of these results is totally empirical and is not backed by theoretical results, in addition to having high computationnal costs. The article studied in this project [1] aims at giving some results on unsupervised text embeddings.

To do so, it focusses on a compressed sensing view of the text embedding problem, which aims at provinding low-dimensional representation for texts. As words can be viewed as very sparse vectors in the vocabulary space, providing a low-dimensional representation of a text can be seen as measuring words signals in a compressed way. The article explores this idea, and links compressed sensing to some LSTMs-learned text representations. In order to evaluate such representations (through their performances on text classification), authors adapts a result in [2] on the quality of low-dimensional compressed sensing representations as inputs for a classification problem.

In this work, we will first recall what are words and text embeddings, in order to recall the basic NLP background for a non-specialized reader, and we will present author's text embeddings. Then, we will focuss on the core part of this project : the compressed sensing part, and its application in text embeddings. Finally, we will present a reproduction of author's results, and discuss their statings and their implications.

# Contents

# 1 Text embeddings

A text embedding is a vector representing a text. Usually, we want this representation to be low-dimensional, and to keep some information on the meaning of the text. By necessity, these representations are generally unsupervised : it would be hard to define a "standard" representation to be learned in a supervised way. In this section, we recall usual ways to learn to generate word embeddings - upon which lie LSTMs text embeddings -, text embeddings, and we explain in which ways they can be used for tasks as text classification.

## 1.1 Word embeddings

The very beginning of most popular text embeddings is word embeddings. In this part we will briefly explain how such embeddings are learned, and we will set the definitions and the general context for the rest of the work.

Supposing that we have a vocabulary $\mathcal{V} = \{w_1, ..., w_V\}$, of size $|\mathcal{V}| = V$, a natural vector representation of any word $w_i, i = 1, ..., V$ is the following :

$$w_i = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \leftarrow i, \quad w_i \in \mathbb{R}^V$$

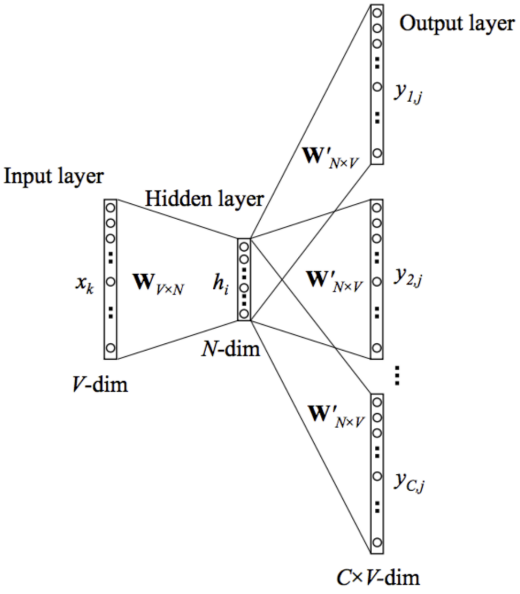
The problem with this simple representation is the dimension ($V$) and its sparsity ($||w_i||_0 = 1, \forall i = 1, ..., V$) of the vectors. To adress this issue, many techniques have been proposed, especially in recent deep learning developments. Non-deep learning techniques are based on co-occurence matrices : given a set $\mathcal{D}$ of documents, we first build a matrix containing the number of co-occurrences of every word in all documents in a context window of size $c$, for instance :

$$M = \begin{pmatrix} C(w_1, w_1) & \dots & C(w_1, w_V) \\ \vdots & \ddots & \vdots \\ C(w_V, w_1) & \dots & C(w_V, w_V) \end{pmatrix} \in \mathbb{R}^{V \times V}$$

Where $C(w_i, w_j)$ is the number of times that the word $w_j$ appears in the context of $w_i$, the context being defined as the words far from at most $c$ words from $w_i$ in any of the documents in $\mathcal{D}$. This matrix is usually regularized using positive pointwise mutual information (PPMI). A word $w_i$ is then represented as the $i$-th line of $M$, or its PPMI counterpart. However, this representation does not get rid of dimension neither sparsity problems. Assuming the PPMI matrix is very sparse, we use, as it is frequently done in high-dimensional statistics, a singular value decomposition $M^{\text{PPMI}} = W \times D \times C$, keeping only the $k$ most important singular values and vectors, leading to a new truncated version of $W$, of size $V \times k$, which becomes the embedding matrix, each line being a word's embedding. This decomposition has good denoising and generalization properties.

In parallel, the most popular deep learning embedding is the Word2Vec embedding, proposed in [3]. This embedding is learned by trying to predict a word from its context (CBOW), or vice versa to predict a context from a given word (skip-gram); using a 1-hidden layer feed-forward neural network (Figure 1).

Figure 1: Skip-gram architecture for Word2Vec




Using some regularization and computationnal tricks during the training, as replacing the softmax output function computation by a hierarchical softmax computation, or as using negative sampling loss as an objective function, or doing frequency subsampling; which are detailed in [4]; Word2Vec embeddings achieved better generalization and denoising properties than SVD. The main idea behind both Word2Vec and SVD embeddings, which makes them coherent as a good semantic representation, is called the distributionnal hypothesis. This linguistic hypothesis is built upon Firth's statement that *"a word is characterized by the company that it keeps"*. Thus, words employed in similar contexts are more likely to have similar meanings. However, latest deep learning developments seem to have abandoned this approach, replaced by language modelling as in [5, 6, 7], which are outside the scope of this project.

The evaluation of such embeddings is difficult, as they are learned in an unsupervised way. However, two techniques are generally used to assess embeddings representation quality : the evaluation of the embedding through a downstream task, i.e. for instance the evaluation of classic text classifiers using the evaluated embedding as an input; and an evaluation through word similarities, with the objective that near vectors (regarding cosine distance) represent words with similar meanings. This evaluation techniques are used in the same manner for text embeddings.

## 1.2 From word embeddings to text embeddings

In natural language processing, and more specifically in high-level tasks, there is no interest on studying single words. Thus, a natural question when working on texts, is : how to transform word embeddings to a single text embedding? In this section we will describe 'natural' text embeddings, resembling the co-occurence matrix described above, used in the article; and LSTMs embeddings, which are now more commonly used.

### 1.2.1 Bag of n-Grams/Cooccurrences

In this part, we will suppose that we have a vocabulary $\mathcal{V} = \{v_1, ..., v_V\}$ and a document $d = w_1, ..., w_T$, where $w_i \in \mathcal{V}$ for any $i = 1, ..., T$, and a fixed integer $n \in \mathbb{N}$. We recall that a $n$-gram is a sequence of $n$ words, for instance the set of unigrams is $\{v_1, ..., v_T\}$, the set of bigrams is $\{(v_1, v_2), (v_2, v_3), (v_1, v_3)..., (v_{V-1}, v_V)\}$, etc. We let $V_k$ be the number of possible $k$-grams over $\mathcal{V}$, independently of order ($V_1 = V$), and $V_n^{\text{sum}} = \sum_{k=1}^{n} V_k$.

For any $k \in \mathbb{N}$, let $B_k$ be a vector indicating, for any possible $k$-gram over $\mathcal{V}$, the number of occurences of this $k$-gram in the document. A Bag-of-$n$-Grams, denoted $x^{\text{BonG}}$ is the concatenation of all the $B_k$ vectors for $k = 1, ..., n$ :

$$x^{\text{BonG}} = [B_1, ..., B_N]$$

In order to simplify and get rid of order in the n-grams, author merge the $k$-grams in the vocabulary that are the same words in a different order. Then, we can write : $B_k^{\text{Co-oc}} = \sum_{t=1}^{T-k+1} e_{\{w_t, ..., w_{t+k-1}\}}$. A Bag-of-$n$-Co-occurrences (BonC) is then :

$$x^{\text{BonC}} = [B_1^{\text{Co-oc}}, ..., B_N^{\text{Co-oc}}]$$

These embeddings are sparse, $V_n^{\text{sum}}$-dimensional vectors.

### 1.2.2 DisC embeddings

This document embeddings relies on word embeddings. Supposing that we have learned word embeddings and that we denote $x_{v_i} \in \mathbb{R}^d, d << V$ the embedding of the word $v_i \in \mathcal{V}$, for any $i = 1, ..., n$; and that we still have our document $d = w_1, ..., w_T$; then the unigram embedding of the document is :

$$z^u = \sum_{t=1}^{T} x_{w_t}$$

$z^u$ can be re-written as the product of a compression matrix $A$ in which columns are word embeddings $x_w$; and the "Bag-of-1-grams" document embedding :

$$z^u = Ax^{\text{Bo1G}} = \sum_{t=1}^{T} Ae_{w_t} = \sum_{t=1}^{T} x_{w_t}$$

Authors extend this unigram definition to $n$-co-occurrences by using element-wise multiplication of word embeddings, defining:

$$\tilde{x}_{\{w_1, ..., w_n\}} = d^{(n-1)/2} \odot_{t=1}^{n} x_{w_t} \quad \in \mathbb{R}^d$$

Then, DisC (distributed co-occurence) embeddings are defined as the concatenation of:

$$z^{(n)} = \left[ C_1 \sum_{t=1}^{T} \tilde{x}_{w_t}, ..., C_n \sum_{t=1}^{T-n+1} \tilde{x}_{\{w_t, ..., w_t+n-1\}} \right] \in \mathbb{R}^{nd}$$

With $C_1, ..., C_n$ being scaling factors, detailed later. As in the unigram case, DisC embeddings are directly related to compressed sensing as we can find $A^{(n)} \in R^{dn \times V_n^{\text{sum}}}$ such that:

$$z^{(n)} = A^{(n)} x^{BonC}$$

### 1.2.3 LSTMs

LSTMs, which stands for Long-Short-Term Memory are a kind of recurrent neural networks (RNN), introduced in [8]. As with classic RNNs, the point in LSTMs is to be able to handle sequential data, say for instance a document made of already embedded words :

$$x = \{x_1, ..., x_T\}$$

A given LSTM cell $A$ 'evolves' with the time, and is caracterized in time by a *hidden state* $h_t$ and a *memory* $c_t$. We denote $A^{(t)}$ the state of cell $A$ at time $t$. $A^{(t)}$ has three inputs :

- $h_{t-1}$, the hidden state transmitted by the previous cell state $A^{(t-1)}$.
- $c_{t-1}$, the memory transmitted by the previous cell state $A^{(t-1)}$.
- $x_t$, the 'real' sample input

And two outputs :

- $h_t$, the updated hidden state transmitted to $A^{(t+1)}$.
- $c_t$, the updated memory transmitted to $A^{(t+1)}$.

These inputs and outputs are schematized in Figure 2 [1]. The core question of LSTMs is to define update rules for $h_t$ and $c_t$. First, we want to know how to update memory, or in other words, which informations to forget and which informations to remember. To do so, we define :

- $i_t = \sigma\left(x_t U^i + h_{t-1} X^i\right) \in [0, 1]$, where $U^i, W^i$ are weights to learn. This function is called the *input gate*. The input gate is a weighted sum of the sample data at time $t$ and the previous hidden state of the cell, regularized by the sigmoid function. The inpute gate decides, what new information we are going to store in the updated memory, and in which proportion, depending on the precedent data.
- $\tilde{c}_t = \tanh\left(x_t U^c + h_{t-1} X^c\right) \in [-1, 1]$, where $U^c, W^c$ are weights to learn. $\tilde{c}_t$ is a candidate for creating 'new $c_t$ values'.
- $f_t = \sigma\left(x_t U^f + h_{t-1} X^f\right) \in [0, 1]$, where $U^f, W^f$ are weights to learn. This is called the *forget gate*. The forget gate decides the proportion at which each element of the memory is going to be forgotten.

The memory can then be updated, with respect to the following rule:

$$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t$$

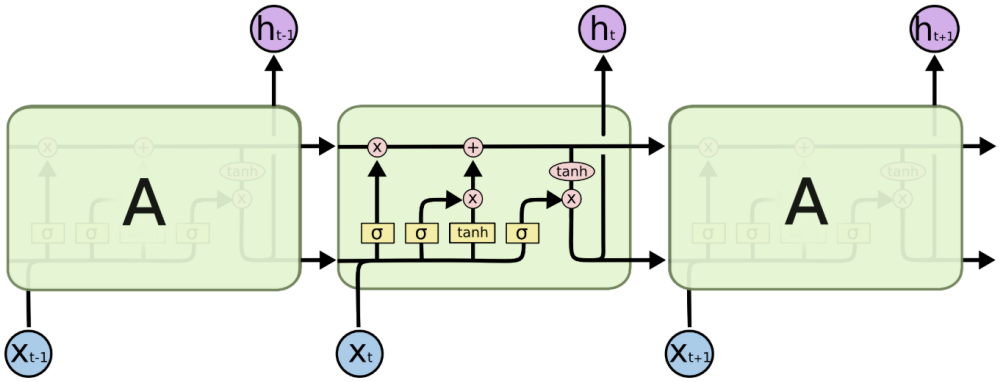
Finally, to update the hidden state, we define the *output gate* in the same fashion, and we update $h_t$ :

$$\begin{aligned} o_t &= \sigma\left(x_t U^o + h_{t-1} X^o\right) && \in [0, 1] \\ h_t &= \tanh(c_t) * o_t && \in [-1, 1] \end{aligned}$$

Some variants of LSTM cells may define input, output and forget gates differently. For instance, the peephole version of LSTM implements the natural idea that the previous memory state $c_{t-1}$ has to be considered when deciding the input, output and forget rates. GRU, Gated Recurrent Units are other variants of LSTMs.

[1] The figure is extracted from Christopher Olah's excellent blog : http://colah.github.io/

Figure 2: A representation of a LSTM cell in time




One question remains : what exactly is the hidden state $h$? $(h_t)_{t=1...T}$ can be viewed as the 'output' of the LSTM. In our case, the final hidden state $h_T$ is directly used as the document embedding. This representation of the document is unsupervised, and the LSTM is usually trained on another task, regarding which $(h_t)_{t=1...T}$ are features (see for instance [9] for an example of LSTMs trained for named entity recognition).

Appart from having a very flexible and modular structure, LSTMs cells do not suffer of the same vanishing gradient problem as RNNs in practice, and are thus more effective in modelling sequential data. They are particularly adapted to NLP problems because of their sequential nature.

## 1.3 Why text embeddings?

# 2 Text embedding as a Compressed Sensing problem

## 2.1 Results on compressed sensing for classification

## 2.2 Links with LSTMs

## 2.3 Sparse recovery with pretrained embeddings

# 3 Experiments and discussion

## 3.1 Results reproduction

## 3.2 Testing DisC embeddings on real life data

## 3.3 Discussion

# References

[1] S. Arora, M. Khodak, N. Saunshi, and K. Vodrahalli, "A compressed sensing view of unsupervised text embeddings, bag-of-n-grams, and lstms," in Proceedings of the 6th International Conference on Learning Representations (ICLR), 2018.

[2] R. Calderbank, "Compressed learning : Universal sparse dimensionality reduction and learning in the measurement domain," 2009.

[3] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed Representations of Words and Phrases and their Compositionality," in Advances in Neural Information Processing Systems 26 (C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, eds.), pp. 3111–3119, Curran Associates, Inc., 2013.

[4] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," pp. 1–12, 2013.

[5] J. Howard and S. Ruder, "Universal language model fine-tuning for text classification," 2018.

[6] M. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), 2018.

[7] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2018.

[8] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural Comput., vol. 9, pp. 1735–1780, Nov. 1997.

[9] G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer, "Neural architectures for named entity recognition," Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, 2016.