K + K = 120: Papers dedicated to L. Kálmán & A. Kornai on the occasion of their 60th birthdays, pp. 141–152

Language modeling with matrix embeddings

GÁBOR BORBÉLY
Budapest University of Technology and Economics
borbely@math.bme.hu

KEYWORDS

language modeling
neural language model
vector embedding
matrix embedding
matrix algebra

ABSTRACT

Vector representations of words, after decades of being at the periphery of computer linguistics, are today widely used and researched. According to our terminology, representing a word involves a function assigning a vector to each word from a finite set (vocabulary). In this paper we investigate certain properties and limitations of word vectors with the aim of improving them. We also present a novel method for learning not vector, but matrix representation of words. The matrices are the result of gradient descent learning where the objective function rewards the presence of a word in its neighboring context, similar to language modeling.

1. Preliminaries

Vector representations of words, after decades of being at the periphery of computer linguistics, are today widely used and researched.

According to our terminology, *representing a word* involves a function which assigns a vector (in $\mathbb{R}^d$) to every word from a finite set (vocabulary) V.

$$v : V \mapsto \mathbb{R}^d$$

Early experiments focused on language modeling with neural architectures (Xu & Rudnicky 2000 and Bengio et al. 2003) instead of n-gram models (Kneser & Ney 1995). Referred to as *distributional vector semantics*, *word embeddings*, or *word vectors*, they are now off-the-shelf tools in the field of natural language processing (NLP) as of Mikolov et al. (2013a) and Pennington et al. (2014). In case of these tools the function v is learned from a corpus of monolingual, unlabeled, tokenized text. Their learning objective is similar to that of language modeling in the sense that they maximize the likelihood of a word in its neighboring context.

Word vectors have proven useful in several applications: e.g., sentiment analysis Socher et al. (2013), diachronic semantics change Hamilton et al. (2016), zero-shot learning Dinu et al. (2015), neural dependency parsing Dozat et al. (2017), and have also been scientifically investigated, e.g., in Arora et al. (2016).

In this paper we investigate certain properties and limitations of word vectors with the aim of improving them. We also present a novel method for learning not vector, but matrix representations of words.

In sections 2 and 3 we provide some theoretical background. Section 4 presents the actual training objectives and models. Some numerical results are provided in section 5.

2. Vector space structure

As seen in Mikolov et al. (2013b) and in Mikolov et al. (2013c) the linear structure of trained vector models is undeniable, meaning that the semantic structure is well represented by vector operations (linear combination and dot product). From analogy questions (*king-man+woman=queen*) through word similarity (angle of word vectors) and translation ($\underline{v}_{\text{dog}} \cdot \underline{\underline{T}}_{\text{eng to ger}} = \underline{v}_{\text{Hund}}$) to even some phrases (*Chinese+river=Yangtze*), linear vector space structure seems to be empirically justified.

However, word vectors alone are not suitable for composing phrases or sequences of words. In the example above *Chinese+river* is not the same as ‘Chinese river’, at least not more so than ‘river Chinese’. Vector addition is *commutative*, i.e. results are independent of the order of the operands. This is why the vector addition itself is not suited for modeling composition. The sum of words may be used to represent a phrase, but tackling compositionality in general is a demanding task.

In Socher et al. (2013) parse trees were used to recursively process phrases to create sentence representations. In Hill et al. (2016) the LSTM architecture (Long Short-Term Memory, Hochreiter & Schmidhuber 1997) was used to represent phrases. Learning compositional mechanisms to embed entities of various length (words, phrases and sentences) are of central interest to modern neural language processing.

3. Algebras

The question arises naturally: which are the appropriate mathematical structures (and composition rules) for a word embedding. The performance of vector models suggests that vector space structure is a good starting point. We also mentioned that beside the useful + operation, words tend to require an additional operation, which composes them, and this composition is non-commutative. An *algebra* over the real field is a reasonable choice Rudolph & Giesbrecht (2010).

$$\begin{tikzcd}
 & A & \\
V, \cdot \arrow[ur] & & R, * \arrow[ul] \\
 & C, + \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

Figure 1: Algebras versus other structures

In Figure 1 the symbols C, V, R and A represent commutative groups, vector spaces, rings, and algebras, respectively. Commutative groups have a commutative addition operator (among others), vector spaces also have dot product (and also scalar multiplication). A ring has addition and (non-commutative) multiplication and an algebra is equipped with all of these operations.

Note that in a group, in theory, one can compute elements like *Chinese+river* or even subtract: *Volga − Russia*, but there is no way of comparing the result to existing vocabulary entries. In a vector space, the dot product can be used to measure similarities between elements, while scalar multiplication allows us to calculate *averages* over certain elements. In a ring, one can use the multiplication operator to model composition, but it still lacks some properties of the vector space. Algebras meet all of these requirements.

As a special case of algebras, *matrix algebras* consist of square matrices, which are central to our investigations. Hence the name *matrix embedding*: we want to train square matrices for each word in a vocabulary, given a corpus of sentences.

The algebra operations would look something like this:

$$\text{green} + \text{orange} \approx \text{yellow-ish color or a team with these colors}$$
$$\text{green} * \text{orange} \approx \text{“green orange” like an unriped fruit}$$

4. Learning matrices

Let V be our vocabulary: a finite set of symbols (words). Let $\mathcal{C} \subset V^*$ be a collection of sentences, i.e., a corpus. We seek a map which assigns a matrix to each word: $M : V \mapsto \mathbb{R}^{d\times d}$. The size of matrices ($d$) is a model parameter.

In order to train such a map we must impose an objective function that measures how good a sentence is.

$$f : V^* \mapsto \mathbb{R}$$

We wish to find an appropriate f and optimize it with respect to M given the corpus of sentences.

First we make some restrictions on the function f. Since we want to model composition via matrix multiplication, f will be evaluated solely on matrices, not on series of matrices. The score of a sentence should be the score of the product of its words.

$$f(\text{“the dog barks”}) = f(M_{\text{the}} \cdot M_{\text{dog}} \cdot M_{\text{barks}})$$

Note that compositionality takes place in the matrix product, the product of three matrices is also a matrix, which is in the same vector space as its components, although not necessarily in the vocabulary.

As in Pennington et al. (2014), we choose the scoring function to be linear in its components. In the example above, it is linear in all of its inputs: “the”, “dog” and “barks”.

$$f(M_{\text{the}} \cdot M_{\text{dog}} \cdot M_{\text{barks}}) \approx \log \mathbb{P}(\text{“the dog barks”})$$

In our work we choose f in a way similar to Rudolph & Giesbrecht (2010):

$$f(M) = \underline{v}^\top \cdot M \cdot \underline{w}$$
$$f(M_{\text{the}} \cdot M_{\text{dog}} \cdot M_{\text{barks}}) = \underline{v}^\top \cdot M_{\text{the}} \cdot M_{\text{dog}} \cdot M_{\text{barks}} \cdot \underline{w}$$

where $\underline{v}$ and $\underline{w}$ are column vectors, depending on the model which will be specified later.

In the following subsections we introduce various models which implement the above ideas. All of them are suitable for optimization and indeed train the embedding M but with different approaches. Numerical results are presented in section 5.

4.1. Neural network model

The following model one does not make predictions about the probabilities of full sentences, but only about probabilities of individual words appearing in a given context. f shall be such that

$$f(\text{context}_{\text{before}}, \text{word}, \text{context}_{\text{after}}) = \mathbb{P}(\text{word}|\text{context})$$
$$\sum_{w \in V} f(\text{context}_{\text{before}}, w, \text{context}_{\text{after}}) = 1.$$

Our architecture consists of an embedding layer M, a composition layer using matrix dot product, and a readout layer that is a softmax function over the vocabulary (given a fixed context).

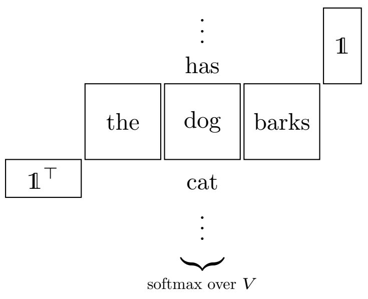

Figure 2: Neural architecture of the matrix embedding model

In formulas, the objective is to minimize the entropy in every context, like:

$$-\log \frac{\exp\left(\mathbb{1}^\top \cdot M_{\text{the}} \cdot M_{\text{dog}} \cdot M_{\text{barks}} \cdot \mathbb{1}\right)}{\sum_{v \in V} \exp\left(\mathbb{1}^\top \cdot M_{\text{the}} \cdot M_v \cdot M_{\text{barks}} \cdot \mathbb{1}\right)} \to \min$$

Or, more precisely, to maximize the following in M.

$$\sum_{(c_b,w,c_a)\in\mathcal{C}} \left[\left(\mathbb{1}^\top \cdot \left(\prod_{b\in c_b} M_b\right) \cdot M_w \cdot \left(\prod_{a\in c_a} M_a\right) \cdot \mathbb{1}\right) - \log \sum_{v\in V} \exp\left(\mathbb{1}^\top \cdot \left(\prod_{b\in c_b} M_b\right) \cdot M_v \cdot \left(\prod_{a\in c_a} M_a\right) \cdot \mathbb{1}\right)\right]$$

where c_b and c_a are the context before and after the word w and the products are ordered (non-commutative) matrix dot products. Contexts are not required to be symmetric or of constant width, an empty product yields the *identity matrix*, which can be used as a placeholder.

Note that this model does not assign probabilities to a whole sentence, only to certain choices of words. The probability of a sentence is hard to measure (see Kornai 2010), therefore we do not require the model to calculate them.

4.2. Direct probabilistic model

We modify the above model in a way that the probability of every sentence, phrase, and word is calculated directly. To obtain a full probabilistic model we eliminate the softmax in the neural model, this is achieved by constraining the matrices M_w to ensure they have a total probability of 1. Then the probability of the skip-gram “the »anything« barks” in the corpus can be defined as

$$\sum_{w\in V} \mathbb{P}(\text{the } w \text{ barks}) = \sum_{w\in V} \mathbb{1}^\top \cdot M_{\text{the}} \cdot M_w \cdot M_{\text{barks}} \cdot \mathbb{1} = \mathbb{P}(\text{the »anything« barks})$$

Since the above formula is linear in the middle matrix, we can calculate a placeholder

$$M_* := \sum_{w\in V} M_w$$

which does not change the probability of any sequence, no matter where it is inserted. We also require that $\mathbb{P}(\text{»anything«}) = 1$.

In this model matrices have an inevitable probabilistic interpretation. We postulate the following *constraints* over M:

- positivity of the elements: $(M_w)_{i,j} \geq 0$,
- right-stochastic sum:

$$\sum_{w \in V} M_w \text{ is a right-stochastic matrix}$$

  i.e., its rows sum up to 1.

Under these conditions we can state the following:

- $\frac{1}{d} \cdot \mathbb{1}^\top \cdot (M_*)^n \cdot \mathbb{1} = 1$ for $n = 0, 1, 2 \ldots$
- If $\underline{v} \in \mathbb{R}^{1 \times d}$ has non-negative entries and sums up to 1, then $\underline{v} \cdot M_*$ also has non-negative entries and sums up to 1 (i.e., keeps the probabilistic row vectors).
- $(M_*)^n$ is also a right-stochastic matrix, therefore

$$\sum_{w_1 \in V} \cdots \sum_{w_n \in V} \frac{1}{d} \mathbb{1}^\top M_{w_1} \cdots M_{w_n} \mathbb{1} = \frac{1}{d} \mathbb{1}^\top (M_*)^n \mathbb{1} = 1.$$

In this setup we can simply calculate the probability of any phrase or series of words as

$$\mathbb{P}(w_1 w_2 \ldots w_n) = \frac{1}{d} \mathbb{1}^\top M_{w_1} M_{w_2} \cdots M_{w_n} \mathbb{1}.$$

This model can be trained on a weighted corpus, where every sentence has an empirical probability p, in this case we must minimize the KL divergence.

$$\underset{\text{constraints on } M}{\arg\min} \sum_{\substack{c \in \mathcal{C} \\ \mathbb{P}(c) = p}} p \cdot \log \left(\frac{p}{\frac{1}{d} \mathbb{1}^\top \left(\prod_{w \in c} M_w \right) \mathbb{1}} \right)$$

If the corpus has no weights then we assume $p \equiv 1$.

The models so far were discriminative models.

4.3. Continuous WFSA

We can generalize weighted finite state automata by modifying the above model, and we can optimize a continuous finite state automaton to fit a weighted language. Similar connections between WFSAs and matrix representations of words can be found in Asaadi & Rudolph (2016). We also introduce a learning algorithm to obtain our matrix embeddings, which in turn can help us learn automata. As future work, these techniques may be relevant in MDL (Minimum Description Length) learning of automata, as in Kornai et al. (2013).

In the previous model the left-hand-side of the product can be considered as a context or state vector.

$$\underbrace{\underbrace{\underbrace{\frac{1}{d}\mathbb{1}^\top \cdot M_{\text{the}} \cdot M_{\text{dog}}}_{\text{previous state}} \cdot M_{\text{barks}}}_{\text{the state after ''barks''}} \cdot \mathbb{1}}_{\text{probability of the whole outcome}}$$

In a way, the initial row vector $\frac{1}{d}\mathbb{1}^\top$ is carried through the sentence and we can obtain the probability of the current state by applying the column vector $\mathbb{1}$.

Some modification is needed to justify this intuition and introduce WFSA. We change the constraints on the embedding M, since the state of an automaton should always sum up to 1. In the previous model the sum of the earlier mentioned row vector decreases as the sentence spans. Let M_w be a right-stochastic matrix for every word $w \in V$. Then the automaton starts from the uniform state $\frac{1}{d}\mathbb{1}^\top$ and every word acts as a transition on this state.

$$\underbrace{\underline{v}}_{\text{state}} \quad \overbrace{\rightarrow}^{\text{action of } w} \quad \underbrace{\underline{v} \cdot M_w}_{\text{new state}} \tag{1}$$

Some additional action is required, since the sum of every state is now 1 and we want to obtain meaningful probabilities. Let $R \in \mathbb{R}^{d\times|V|}$ be a matrix of non-negative entries which is responsible for emissions. In neural network terminology we would call this the *readout layer*.

At every state $\underline{v}$ the columns of the matrix R determine which word should follow.

$$\mathbb{P}(\text{next word is } w |\text{ state } \underline{v}) = \underline{v} \cdot \underbrace{R_{\bullet,w}}_{w^{\text{th}} \text{ column}} \tag{2}$$

Constraints on R and M are listed below.

- M_w is a right-stochastic matrix $\forall w \in V$.
- The rows of R sum up to 1 (and R has non-negative entries).

Under these constraints an automaton arises:

- The states are $1, 2 \dots d$.
- The initial state is uniform over the states: $\frac{1}{d}\mathbb{1}^\top$.
- A word w acts as a transition function on the states as in (1).
- The outcomes (or emissions) at a given state (probabilistic row vector) $\underline{v}$ follow as in (2).

Finally, the probability of an emission sequence is the following product:

$$\mathbb{P}(w_1 w_2 \dots w_n) = \underbrace{\mathbb{1}^\top R_{\bullet,w_1}}_{\mathbb{P}(w_1)} \cdot \underbrace{\mathbb{1}^\top M_{w_1} R_{\bullet,w_2}}_{\mathbb{P}(w_2|w_1)} \cdots \underbrace{\mathbb{1}^\top \prod_{i=1}^{n-1} M_{w_i} R_{\bullet,w_n}}_{\mathbb{P}(w_n|w_1 w_2 \dots w_{n-1})}$$

Given a corpus or a weighted language, we can use the same objective function as in the previous section and train M and R.

Note that, unlike in the previous two models, this model is not sensitive to future words. The next emission and state does not depend on following words. In contrast to the previous one, this is a generative model.

5. Results

Our experimental setup used the UMBC gigaword corpus (Han et al. 2013) which was tokenized and split at sentence boundaries (punctuation part-of-speech tag). The words were not converted into lowercase. It contains about 3.338G words, the average length of a sentence is about 24 with standard deviation 15. For computational reasons, we excluded long sentences, leaving 126.7M sentences to work with.

Words with frequency below 52 were replaced with a unique symbol `<UNK>`, leaving roughly 100k types in the vocabulary (precisely 100147).

The implementation is not detailed herein, but the code is available.[1]

We encountered some serious numerical obstacles in case of model 4.1. We are not certain whether these numerical issues are caused by the

[1] https://github.com/hlt-bme-hu/lm_me, see C++ code for neural model, python (theano) implementation for the other two models.

implementation or by the mathematical model, but the problem occurs if the stochastic gradient descent encounters the same token several times in the same sentence. Nevertheless, this problem did not occur in models 4.2 and 4.3. The first model differs from the others in several aspects: implementation language, mathematical model, and also in gradient descent strategy.

The performance of each model was measured on Google Analogy questions Mikolov et al. (2013a), see evaluation code below.[2] Cosine similarity was used on the flattened matrices. The third model did not achieve meaningful quality within reasonable computation time, here we only present results of the second model.

The table below shows the number of correctly answered questions of each trained model. *Commutative* means that the matrices were 100×100 diagonal matrices; they form a commutative algebra. This can be considered as a fallback to word vectors. The *dense* models consist of 10×10 dense matrices.

Algebra	Model	Nr. correct
commutative	4.2	555
dense	4.2	81

The model 4.3 could answer 1 or 2 questions after equal amount of training.

6. Outlook

We introduced several techniques to train matrix embeddings of words with various numerical efficiency and quality.

Training high quality embeddings and/or automata is our future interest. There are some obvious obstacles in computation time, since the training of a well tuned embedding usually takes days and matrix models are expected to require even more computation time.

A possible computational enhancement is the use of structured, sparse matrices, of which we train only certain elements, hence taking a sub-algebra of the full matrix algebra. This hastens some calculations but keeps the desired algebra properties intact. To this end, further studies of matrix algebras and their sub-algebras are considered.

[2] https://github.com/hlt-bme-hu/eval-embed

Currently these experiments are in a preliminary state, but many improvements and applications are possible. As my supervisor, András Kornai, has once described it, "Like socialism; appealing idea, but not working in practice".

References

Arora, S., Y. Li, Y. Liang, T. Ma and A. Risteski. 2016. RAND-WALK: A latent variable model approach to word embeddings. Transactions of the Association for Computational Linguistics 4. 385–399.

Asaadi, S. and S. Rudolph. 2016. On the correspondence between Compositional Matrix-Space Models of language and weighted automata. In Proceedings of the ACL Workshop on Statistical Natural Language Processing and Weighted Automata (StatFSM 2016).

Bengio, Y., R. Ducharme, P. Vincent and C. Janvin. 2003. A neural probabilistic language model. Journal of Machine Learning Research 3. 1137–1155.

Dinu, G., A. Lazaridou and M. Baroni. 2015. Improving zero-shot learning by mitigating the hubness problem. ICLR 2015, Workshop Track.

Dozat, T., P. Qi and C. D. Manning. 2017. Stanford's Graph-based Neural Dependency Parser at the CoNLL 2017 Shared Task. In Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies. Vancouver, Canada: Association for Computational Linguistics, 20–30.

Hamilton, W. L., J. Leskovec and D. Jurafsky. 2016. Diachronic word embeddings reveal statistical laws of semantic change. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics. 1489–1501.

Han, L., A. L. Kashyap, T. Finin, J. Mayfield and J. Weese. 2013. UMBC_EBIQUITY-CORE: Semantic textual similarity systems. In Second Joint Conference on Lexical and Computational Semantics (*SEM). 44–52.

Hill, F., K. Cho, A. Korhonen and Y. Bengio. 2016. Learning to understand phrases by embedding the dictionary. Transactions of the Association for Computational Linguistics 4. 17–30.

Hochreiter, S. and J. Schmidhuber. 1997. Long short-term memory. Neural Computation 9. 1735–1780.

Kneser, R. and H. Ney. 1995. Improved backing-off for m-gram language modeling. In International Conference on Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., vol. 1. IEEE, vol. 1, 181–184.

Kornai, A. 2010. Rekurzívak-e a természetes nyelvek? [Are natural languages recursive?] Magyar Tudomány 171. 994–1005.

Kornai, A., A. Zséder and G. Recski. 2013. Structure learning in weighted languages. In Proceedings of the 13th Meeting on the Mathematics of Language (MoL 13). Sofia, Bulgaria, 72–82.

Mikolov, T., K. Chen, G. Corrado and J. Dean. 2013a. Efficient estimation of word representations in vector space. In Y. Bengio and Y. LeCun (eds.) Workshop proceedings, International Conference on Learning Representations (ICLR 2013).

Mikolov, T., Q. V. Le and I. Sutskever. 2013b. Exploiting similarities among languages for machine translation. Manuscript. https://arxiv.org/abs/1309.4168.

Mikolov, T., I. Sutskever, K. Chen, G. S. Corrado and J. Dean. 2013c. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani and K. Q. Weinberger (eds.) Advances in neural information processing systems 26. Red Hook, NY: Curran Associates. 3111–3119.

Pennington, J., R. Socher and C. Manning. 2014. Glove: Global vectors for word representation. In Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2014).

Rudolph, S. and E. Giesbrecht. 2010. Compositional matrix-space models of language. In Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics. Uppsala, Sweden, 907–916.

Socher, R., J. Bauer, C. D. Manning and A. Y. Ng. 2013. Parsing with compositional vector grammars. In Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (ACL 2013). Sofia, Bulgaria: Association for Computational Linguistics, 455–465.

Xu, W. and A. I. Rudnicky. 2000. Can artificial neural networks learn language models? In International Conference on Statistical Language Processing. Beijing, 202–205.