Language Modeling with Matrix Embeddings

Gábor Borbély
Department of Algebra,
Budapest University of Technology and Economics,
Egry József u. 1.
1111 Budapest, Hungary
borbely@math.bme.hu

1 Preliminaries

Vector representations of words were at the periphery of computer linguistics for decades however, recently they became widely used and researched.

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

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

The first results concentrated on language modeling with neural architectures ((Xu and Rudnicky, 2000) and (Bengio et al., 2003)) instead of n-gram models ((Kneser and Ney, 1995)).

Referred as *distributional vector semantics* or *word embeddings* or *word vectors*, they are now off-the-shelf tools in NLP as of (Mikolov et al., 2013a) and (Pennington et al., 2014). In these tools, the function v is learned from a corpus of unilingual, unlabeled, tokenized text. Their learning objective is similar to language modeling in the sense that they maximize the presence of a word in its neighboring context.

Word vectors are proven to be useful in applications: e.g. sentiment analysis (Socher et al., 2013), diachronic semantics change (W. Hamilton, 2016) or zero-shot learning (Dinu et al., 2015) and neural dependency parsing (Dozat et al., 2017) and scientifically investigated like (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 representation of words.

In section 2 and 3 we lay out some theoretical backgrounds. Section 4 presents the actual training objectives and models. There are a few numerical results in section 5

2 Vector space structure

As seen in (Mikolov et al., 2013c) and in (Mikolov et al., 2013b) the linear structure of a trained vector model 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*), the 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 than *"river Chinese"*. The vector addition is *commutative*, namely, one cannot distinguish the interchange of the components. This is why the vector addition itself is not suited for modeling composition. The sum of its words may represent a phrase but tackling the compositionality, in general, is a demanding task.

In (Socher et al., 2013) a parse tree was used to recursively process the phrases and make them into one sentence representation. In (Hill et al., 2016) an LSTM architecture (Long Short-Term Memory (Hochreiter and Schmidhuber, 1997)) was used to represent a phrase.

Learning compositional mechanisms and embeddings of various length entities (words, phrases and sentences) are in the central interest of novel neural language processing.

3 Algebras

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

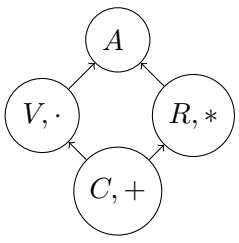

Figure 1: Algebras versus other structures

In Figure 1 the symbols C, V, R and A represent commutative groups, vector spaces, rings and algebras respectively. A commutative group has a commutative

addition operator (among others); in addition to that, a vector space has 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 measure similarities between elements and the scalar multiplication gives us the ability to calculate *averages* over certain elements. In a ring, one can use the multiplication operator to model composition, but still lacks some aspects of the vector space. In an algebra, all of these requirements are met.

As a special case of algebras, *matrix algebras* consist square matrices and they are in the target of our studies. 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. corpus. We seek a map which gives a matrix for each word: $M : V \mapsto \mathbb{R}^{d\times d}$. The size of the matrices (d) is a model parameter.

In order to train such a map, one has to impose an objective function which 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 of all, we make some restrictions of the function f. Since we want to model the 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 the 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 score function to be linear in the 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 similarly to (Rudolph and 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 vector, 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. We detail the numerical results in section 5.

4.1 Neural network model

In the following model one does not assert a probability to a full sentence, but only a probability that a word fits in a 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 an embedding layer M, a composition layer, which is matrix dot product, and a readout layer, which is a softmax over the vocabulary (given a fixed context).

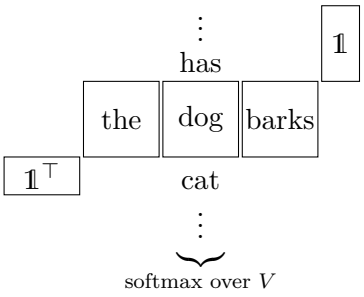

Figure 2: Neural architecture of matrix embedding model

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 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. The context are not bound to be symmetric or 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

One can rewrite the above model in a way that every sentence, phrase and word gets a directly calculated probability. As a motivation, we will try to eliminate the softmax in the neural model, which will lead us to the full probabilistic model.

If the softmax is to be eliminated, then the matrices M_w should be restricted to ensure the total probability of 1. A constraint will be added to M such that

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

giving the probability of the skip-gram "the »anything« barks" in the corpus. Since the above formula is linear in the middle matrix, one 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. And we also want that $\mathbb{P}(\text{»anything«}) = 1$.

In this model the 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 on can state the followings.

- $\frac{1}{d} \cdot \mathbb{1}^\top \cdot (M_*)^n \cdot \mathbb{1} = 1$ for $n = 0, 1, 2 \dots$
- 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

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

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

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

This model can be trained on a weighted corpus, where every sentence has an empirical probability p. In such a case, one has to 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

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

One can see in the previous model that 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 one can ask the probability of the current state by applying the column vector $\mathbb{1}$.

Some modification is needed to justify this intuition and introduce a 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 $w \in V$. In this way the automaton starts from the uniform state $\frac{1}{d}\mathbb{1}^\top$ and every word acts as a transition on this state.

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

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 the emissions. In neural network terminology we would call that *read-out layer*.

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

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

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 on the states: $\frac{1}{d}\mathbb{1}^\top$
- A word w acts as a transition function on the states as (1).
- The outcomes (or emissions) at a given state (probabilistic row vector) $\underline{v}$ follows as (2).

And 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, one 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 the future words to come. The next emission and state does not depend on the 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 less than 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. It is not certain that the numerical issues are the result of implementation or come from the mathematical model, but the problem occurs if the stochastic gradient descent encounters the same token repeated several times in the same sentence. Although, this problem did not occur neither in model 4.2 nor 4.3. The first model differs from the others in many aspects: implementation language, mathematical model, and also in gradient descent strategy.

Only the results of the second model are presented. The third model did not reach a reasonable quality within a reasonable computational time . The quality of the models were measured on Google Analogy questions (Mikolov et al., 2013a), see evaluation code below[2]. Cosine similarity was used on the flattened matrices.

In table 1 one can read 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**	**# correct**
commutative	4.2	555
dense	4.2	81

Table 1

The model 4.3 could answer 1 or 2 questions after the same 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 a day and matrix models are expected to require more computations.

A possible computational enhancement is the use of structured, sparse matrices, where we train only certain elements in the matrices, 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 subalgebras are considered.

Currently these experiments are in a preliminary state but many improvements and applications are possible. As my supervisor once has described it:

> *"Like socialism; appealing idea, but not working in practice."*
> — András Kornai

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

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

References

[Arora et al.2016] Sanjeev Arora, Yuanzhi Li, Yingyu Liang, Tengyu Ma, and Andrej Risteski. 2016. Rand-walk: A latent variable model approach to word embeddings. *Transactions of the Association for Computational Linguistics (TACL)*, 4:385–399.

[Asaadi and Rudolph2016] Shima Asaadi and Sebastian 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)*, Berlin, Germany.

[Bengio et al.2003] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155.

[Dinu et al.2015] Georgiana Dinu, Angeliki Lazaridou, and Marco Baroni. 2015. Improving zero-shot learning by mitigating the hubness problem. In *ICLR 2015, Workshop Track*.

[Dozat et al.2017] Timothy Dozat, Peng Qi, and Christopher 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*, pages 20–30, Vancouver, Canada, August. Association for Computational Linguistics.

[Han et al.2013] Lushan Han, Abhay L. Kashyap, Tim Finin, James Mayfield, and Jonathan Weese. 2013. UMBC_EBIQUITY-CORE: Semantic textual similarity systems. In *Second Joint Conference on Lexical and Computational Semantics (*SEM)*, pages 44–52, Atlanta, Georgia, USA. Association for Computational Linguistics.

[Hill et al.2016] Felix Hill, KyungHyun Cho, Anna Korhonen, and Yoshua Bengio. 2016. Learning to understand phrases by embedding the dictionary. *Transactions of the Association for Computational Linguistics*, 4:17–30.

[Hochreiter and Schmidhuber1997] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780, November.

[Kneser and Ney1995] Reinhard Kneser and Hermann Ney. 1995. Improved backing-off for m-gram language modeling. In *International Conference on Acoustics, Speech, and Signal Processing, 1995. ICASSP-95.*, volume 1, pages 181–184. IEEE.

[Kornai et al.2013] András Kornai, Attila Zséder, and Gábor Recski. 2013. Structure learning in weighted languages. In *Proceedings of the 13th Meeting on the Mathematics of Language (MoL 13)*, pages 72–82, Sofia, Bulgaria, August. Association for Computational Linguistics.

[Kornai2010] András Kornai. 2010. Rekurzívak-e a természetes nyelvek? *Magyar Tudomány*, 171(8):994–1005.

[Mikolov et al.2013a] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. In Y. Bengio and Y. LeCun, editors, *Proceedings of the ICLR 2013*.

[Mikolov et al.2013b] Tomas Mikolov, Quoc V Le, and Ilya Sutskever. 2013b. Exploiting similarities among languages for machine translation. Xiv preprint arXiv:1309.4168.

[Mikolov et al.2013c] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff 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, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc.

[Pennington et al.2014] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*.

[Rudolph and Giesbrecht2010] Sebastian Rudolph and Eugenie Giesbrecht. 2010. Compositional matrix-space models of language. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 907–916, Uppsala, Sweden.

[Socher et al.2013] Richard Socher, John Bauer, Christopher D. Manning, and Ng Andrew Y. 2013. Parsing with compositional vector grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (ACL 2013)*, pages 455–465, Sofia, Bulgaria. Association for Computational Linguistics.

[W. Hamilton2016] D. Jurafsky W. Hamilton, J. Leskovec. 2016. Diachronic word embeddings reveal statistical laws of semantic change. *ACL*.

[Xu and Rudnicky2000] Wei Xu and Alexander I Rudnicky. 2000. Can artificial neural networks learn language models? In *International Conference on Statistical Language Processing*, pages 202–205, Beijing, China.