Learning word embeddings from dictionary definitions only

Tom Bosc
MILA, Université de Montréal
Microsoft Maluuba
tom.bosc@umontreal.ca

Pascal Vincent
MILA, Université de Montréal, CIFAR,
Facebook AI Research
pascal.vincent@umontreal.ca

Abstract

How can we learn from dictionaries? This paper presents a simple model that learns to encode word embeddings by reading definitions and trying to reconstruct them. Crucially, the model also exploits the inherent recursivity of dictionaries by encouraging the embeddings produced by the encoder to be close to the embeddings that are used as inputs to the encoder when processing a definition. We evaluate the model on similarity and relatedness benchmarks and show preliminary results for the generalisation of the model in a one-shot learning setting.

Pretrained word representations are very useful features in NLP [Turian et al., 2010]. The usual methods for learning word embeddings (i.e. dense, low-dimensional, real-valued representations) are based on the distributional hypothesis, which states that words that occur in similar contexts tend to have similar meanings [Harris, 1954]. While very successful, such methods are data hungry and require the availability of large training corpora. Recent works that aim for better data efficiency include Herbelot and Baroni [2017] which shows that word2vec [Mikolov et al., 2013] is able to learn rare words using few examples if provided with suitable hyperparameter. [Wang et al., 2017] proposes a Bayesian model for few-shots learning.

Although we share a similar data-efficiency goal, our approach departs from distributional methods. To produce a word's embedding, we exclusively rely on a much more compact source of data: dictionary definitions. Our approach is based on a definition auto-encoder: the encoder processes the definition of a word to yield that word's embedding. The decoder then attempts to reconstruct the bag-of-word representation of the definition from that embedding.

Related works include Hill et al. [2016]which learn to produce a target embedding for a word by processing its definition and Noraset et al. [2017] which learn to produce a definition from a pretrained embedding. Another new method called dict2vec [Tissier et al., 2017] augments word2vec with the additional objective of reconstructing the words in the definition but without an encoder. Bahdanau et al. [2017] also uses definitions to produce embeddings tailored for downstream tasks. By contrast, our method combines both encoder and decoder and is entirely unsupervised and self-contained: it does not use pretrained embeddings, extra corpora, or other tasks.

When relying on definitions, one question remains largely unsolved: how to account for the recursivity in definitions? The definition of a word a can use a word b while the definition of b can include a. Tissier et al. [2017] deals with this by positing that the two words "form a strong pair" and the objective weighs strong pairs more than weak pairs. We propose a different solution: a soft representation-sharing scheme to ensure that embeddings produced as output by the encoder can also be used as input to the encoder when processing a definition.

All of this relates to meta-learning in the following way. We view embeddings as parameters which the encoder predicts. Without taking into account the recursivity, the model only learns to predict useful parameters for reconstructing the definition. But the fact that words are reused in other definitions make the parameters useful for predicting embeddings for other words. In a way, a model

that deals with recursion anticipates the future reuse of the representations, implementing a form of meta-learning. We evaluate the model in a simple one-shot learning setting on similarity and relatedness benchmarks and show promising results.

1 Model

We suppose that a monolingual dictionary is our only source of data. It maps words that we call keys to one or several definitions, which are sequences of words. We consider all definitions to be equally important and do not favor more frequent words. Our training criterion is built on the following principle: a representation is good if a model can to some extent recover the definition from which the representation was built. Intuitively, one knows a word if one can define it.

Let V_D the set of all words that are used in definitions and V_K the set of all words that are defined (the set of keys). We note w a key, defs(w) the set of its definitions. A definition $d \in \text{defs}(w)$ of a word $w \in V_K$ is a word sequence, represented as $d = (d_1,...,d_T)$ where d_t is the index of a word in vocabulary V_D. We encode such a definition d by processing it with a LSTM [Hochreiter and Schmidhuber, 1997]. Each definition is mapped to a representation so a word with several definitions will have several representations (multi-prototype word embeddings [Reisinger and Mooney, 2010]).

The LSTM is parametrized by θ and an input embedding E of size $|V_D| \times m$, whose i_{th} row E_i contains an m-dimensional input embedding for the i_{th} word of V_D. The last hidden state computed by this LSTM yields an m-dimensional definition embedding:

$$h = f_{E,\theta}(d) = \text{LSTM}_{E,\theta}(d)$$

The subsequent decoder can be seen as a conditional language model trained by maximum likelihood to regenerate definition d given definition embedding h(d). We use a simple conditional unigram model with a linear $|V_{out}| \times d$ parametrization matrix $\tilde{E}$ and bias vector b, and maximize the log-probability of definition d under that model[1]:

$$\sum \quad (\quad) \quad \sum \quad e^{\langle \tilde{E}_{d_t},h\rangle+b_{d_t}}$$

$$\log p_{\tilde{E},b}(d|h) = \sum_t \log p_{\tilde{E},b}(d_t|h) = \sum_t \log \operatorname{softmax}(\tilde{E}h + b)_{d_t} = \sum_t \log \sum_k e^{\langle \tilde{E}_k,h\rangle + b_k}$$

where ⟨, ⟩ denotes an ordinary dot product. We call $\tilde{E}$ the output embedding matrix. The basic auto-encoder training objective to minimize over the dictionary can then be formulated as:

$$J_r(E, \theta, \tilde{E}) = - \sum_{w \in V_K} \sum_{d \in defs(w)} \log p_{\tilde{E},b}(d|h = f_{E,\theta}(d))$$

We have 3 kinds of m-dimensional embeddings: definition embeddings h computed by the recurrent encoder, input embeddings E and output embeddings $\tilde{E}$. We add a penalty weighted by $\lambda > 0$ to the cost that brings definition embeddings closer to input embeddings:

$$J(E, \theta, \tilde{E}) = J_r(E, \theta, \tilde{E}) + \lambda \sum_{w \in V_D \cap V_K} \sum_{d \in defs(w)} ||E_w - f_{E,\theta}(d)||_2$$

where E_w denotes the input embedding (row of E) associated to word w. Note that $V_D = V_K$ as some words are defined but are not used in definitions and some words are used in definitions but inflected. We see that in the simple case where a word w has only one definition d(w) and using a large λ then $E_w \approx f_{E,\theta}(d(w))$ after optimisation. In this sense, the penalty is a proxy for recursively injecting embeddings using the same encoder.

[1]The order of the words is ignored by this simplistic model. One could use a recurrent decoder instead, but the number of definitions is so small that it may be hard to avoid overfitting.

2 Experiments

2.1 Data and optimisation

We use WordNet [Fellbaum, 1998] definitions as the only source of data. We do not include part of speech tags that go with definitions. We filter out function words.

Not only do we want to learn good embeddings: we also want to test the generalisation ability of our model on unseen definitions. For that, we split the keys and their definitions into a train, validation (for early stopping) and test splits. The algorithm for splitting the dictionary puts words in batches. It ensures two things: firstly, that words which share at least one definition are in the same batch; secondly, that each word in a batch is associated with all its definitions. We sort the batches by the number of distinct definitions they contain. We use the largest batch returned by the algorithm as the training set: it contains 36722 distinct definitions, mostly associated to frequent and polysemous words. The validation and test set consists 2109 and 77735 distinct definitions respectively.

All the models have embeddings of dimension 300. This is also the dimension of the hidden states of the LSTM. We train all the models with Adam [Kingma and Ba, 2014] with $\beta_1 = 0.9$, $\beta_2 = 0.999$ and a learning rate of $3 \cdot 10^{-4}$. We use a batch size of 32 definitions. Every 2000 batch, we compute the mean cost on the validation set and stop after 20000 batches without improvement.

2.2 Evaluation on similarity and relatedness benchmarks

Testing the learned representations is a complex issue [Faruqui et al., 2016]. Indeed, different evaluation methods give different rankings of embeddings, suggesting that there is no single embedding that outperforms others on any tasks [Schnabel et al., 2015] and thus no single best method. In addition, multi-prototype embeddings are not straightforward to use in downstream tasks because we somehow need to disambiguate and choose from the set of embeddings. We instead focus on intrinsic evaluation methods and in particular, similarity and relatedness benchmarks. They don't suffer from the problems of analogy prediction benchmarks. [Rogers et al., 2017] [Linzen, 2016]

The benchmarks consist of pairs of words scored according to some criteria. They vary in terms of annotation guidelines, number of annotators, selection of the words, etc. Typically, the predictions and the ground truth are ranked and the correlation between the ranks is measured by Spearman's ρ.

We adopt one of the methods advocated by Faruqui et al. [2016] and use separate datasets for model selection and tuning the hyperparameters (λ and the size of V_{out}), and we test on the rest. This development set comprises the development sets of MEN [Bruni et al., 2014] and SimVerb3500 [Gerz et al., 2016] (the only datasets that provide a dev/test split). At this step, we are using all the embeddings of all the definitions, seen at training time or not. Then, we test using the best models of each family on the other benchmarks.[2]

Because our model embeds definitions and not words directly, there are sometimes several embeddings per words to choose from. Reisinger and Mooney [2010] propose 2 models: the average of the pairwise cosine similarities and the maximum of the pairwise cosine similarities. We use the average (AvgCos) as it performs the best in our experiments. We evaluate the model with and without penalty.

We first compare the computed embeddings to pretrained GloVe [Pennington et al., 2014], word2vec and dependency-based [Levy and Goldberg, 2014] embeddings [3]. We also compare to a GloVe and word2vec model trained on the definitions by placing the keys in front of their definitions. These models are ill-suited for dictionaries because the words in the definition of a word a do not have the same distribution as the words in the context of a in regular corpora. Although, we have found that they are good baselines. We omit the results of GloVe on definitions which are consistently worse than that of word2vec on definitions. This first evaluation gives us information about what we can extract from the dictionary without considering one-shot learning abilities of the model.

[2]The other similarity benchmark are SimLex999 [Hill et al., 2016] and SimLex333, a challenging subset of SimLex999 which contains only highly related pairs. The other relatedness benchmarks are RG [Rubenstein and Goodenough, 1965], WS353 [Finkelstein et al., 2001], SCWS Huang et al. [2012] and MTurk Radinsky et al. [2011]Halawi et al. [2012].

[3]GloVe (840B) is available at https://nlp.stanford.edu/projects/glove/
word2vec at https://code.google.com/archive/p/word2vec/
dependency-based at https://levyomer.wordpress.com/2014/04/25/dependency-based-word-embeddings/

Page 4

	SL999	SL333	SV3500-t	RG	SCWS	MEN-t	MT	353
GloVe	39.9	26.0	22.6	69.2	56.6	75.6	71.6	70.8
w2v	44.2	29.7	35.8	76.1	66.0	75.6	67.1	70.0
dependency-based	44.6	34.2	30.5	59.3	66.9	59.3	62.0	60.6
w2v (defs)	34.5	16.0	36.4	65.7	54.5	59.9	56.1	61.9
Model wo/ penalty	27.5	24.4	26.3	48.2	47.6	38.5	36.0	34.1
Model	37.7	31.8	38.9	63.7	57.7	50.3	48.0	42.2

Table 1: Spearman's $\rho \times 100$ on various benchmarks (cf. 2.2). Standard corpora training (top), dictionary training (bottom). Penalty systematically helps the model. Model outperforms word2vec on similarity benchmarks (left part) while being worse on relatedness (right part).

	SL999	SV3500-t	WS353	MEN-t	SCWS	MTurk
All	37.7 (999)	38.9 (3000)	42.2 (353)	50.3 (1000)	57.7 (2003)	48.0 (771)
Train pairs	43.0 (702)	38.2 (2891)	45.7 (184)	56.8 (561)	62.4 (1488)	48.6 (546)
Test pairs	26.7 (297)	54.5 (109)	35.2 (41)	44.2 (439)	44.8 (515)	45.0 (225)

Table 2: Spearman's correlation coefficient ρ × 100 on subsets of benchmarks. Between brackets: sample size. Train pairs are pairs in the benchmarks for which both words are in the training or validation set. Test pairs contain at least one word that is in the test set. Correlation is lower for test pairs but remains quite strong.

The second part of the evaluation deals with one-shot learning. We compute correlation coefficients on subsets of the benchmarks where words have not all been seen during training and check how correlated the similarity and relatedness scores are with the ground truth.

3 Results

The results of the first evaluation are reported in table 1. Firstly, word2vec works relatively well on the definitions while using a fraction of the data (around 350k tokens against 300M tokens for word2vec, 6B for GloVe). It is roughly at the level of the worst distributional method, indicating that the dictionary contains useful information, even for a model that ignore its structure. The penalty improves the representations learned by our model by a quite large margin on every benchmark. The model with penalty outperforms word2vec on similarity benchmarks but captures less relatedness. This can be desirable for certain tasks such as machine translation Hill et al. [2016].

In table 1, we did not distinguish between definitions that were seen during training and unseen definitions. Table 2 presents the same correlation coefficient computed on two subsets of the pairs: the pairs for which all the definitions were seen during training (train pairs) and the pairs for which at least one word was defined in the test set (test pairs). We see that the correlation coefficients are lower for test pairs (except on SimVerb). Still, the scores predicted by the model on the test pairs are quite correlated with the ground truth: the correlation coefficient ρ is above 0.35 on every benchmark (except on SimLex). Our model seems to generalize well and to produce coherent embeddings in the one-shot learning setting.

4 Conclusion and future work

We have presented an alternative for embedding words that uses the definitions of dictionaries instead of relying on the distributional hypothesis. It is doubly data-efficient as it globally uses very few data and shows promising results in a one-shot learning setting as well. More work is needed to evaluate the embeddings on downstream tasks and to analyze the pros and cons of the distributional versus dictionary-based representations.

We are also interested in its potential lifelong learning abilities: given a small set of well-chosen words [Picard et al., 2009], we might be able to bootstrap on their representations, iteratively defining more and more words as our coverage on the definitions increase, until all words are defined.

References

D. Bahdanau, T. Bosc, S. Jastrzebski, E. Grefenstette, P. Vincent, and Y. Bengio. Learning to compute word embeddings on the fly. CoRR, abs/1706.00286, 2017. URL http://arxiv.org/abs/1706.00286.

E. Bruni, N.-K. Tran, and M. Baroni. Multimodal distributional semantics. J. Artif. Intell. Res.(JAIR), 49(2014):1–47, 2014.

M. Faruqui, Y. Tsvetkov, P. Rastogi, and C. Dyer. Problems with evaluation of word embeddings using word similarity tasks. arXiv preprint arXiv:1605.02276, 2016.

C. Fellbaum. WordNet. Wiley Online Library, 1998.

L. Finkelstein, E. Gabrilovich, Y. Matias, E. Rivlin, Z. Solan, G. Wolfman, and E. Ruppin. Placing search in context: The concept revisited. In Proceedings of the 10th international conference on World Wide Web, pages 406–414. ACM, 2001.

D. Gerz, I. Vuli, F. Hill, R. Reichart, and A. Korhonen. Simverb-3500: A large-scale evaluation set of verb similarity. arXiv preprint arXiv:1608.00869, 2016.

G. Halawi, G. Dror, E. Gabrilovich, and Y. Koren. Large-scale learning of word relatedness with constraints. In Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 1406–1414. ACM, 2012.

Z. S. Harris. Distributional structure. Word, 10(2-3):146–162, 1954.

A. Herbelot and M. Baroni. High-risk learning: acquiring new word vectors from tiny data. arXiv preprint arXiv:1707.06556, 2017.

F. Hill, R. Reichart, and A. Korhonen. Simlex-999: Evaluating semantic models with (genuine) similarity estimation. Computational Linguistics, 2016.

S. Hochreiter and J. Schmidhuber. Long short-term memory. Neural computation, 9(8):1735–1780, 1997.

E. H. Huang, R. Socher, C. D. Manning, and A. Y. Ng. Improving word representations via global context and multiple word prototypes. In Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1, pages 873–882. Association for Computational Linguistics, 2012.

D. Kingma and J. Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

O. Levy and Y. Goldberg. Dependency-Based Word Embeddings. In ACL (2), pages 302–308, 2014. URL http://www.aclweb.org/anthology/P/P14/P14-2050.pdf.

T. Linzen. Issues in evaluating semantic spaces using word analogies. arXiv preprint arXiv:1606.07736, 2016.

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, pages 3111–3119, 2013.

T. Noraset, C. Liang, L. Birnbaum, and D. Downey. Definition Modeling: Learning to Define Word Embeddings in Natural Language. In AAAI, pages 3259–3266, 2017.

J. Pennington, R. Socher, and C. Manning. Glove: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pages 1532–1543, 2014.

O. Picard, A. Blondin-Massé, S. Harnad, O. Marcotte, G. Chicoisne, and Y. Gargouri. Hierarchies in dictionary definition space. arXiv preprint arXiv:0911.5703, 2009.

K. Radinsky, E. Agichtein, E. Gabrilovich, and S. Markovitch. A word at a time: computing word relatedness using temporal semantic analysis. In Proceedings of the 20th international conference on World wide web, pages 337–346. ACM, 2011.

J. Reisinger and R. J. Mooney. Multi-prototype vector-space models of word meaning. In Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics, pages 109–117. Association for Computational Linguistics, 2010.

A. Rogers, A. Drozd, and B. Li. The (Too Many) Problems of Analogical Reasoning with Word Vectors. In Proceedings of the 6th Joint Conference on Lexical and Computational Semantics (* SEM 2017), pages 135–148, 2017.

H. Rubenstein and J. B. Goodenough. Contextual correlates of synonymy. Communications of the ACM, 8(10):627–633, 1965.

T. Schnabel, I. Labutov, D. M. Mimno, and T. Joachims. Evaluation methods for unsupervised word embeddings. 2015.

J. Tissier, C. Gravier, and A. Habrard. Dict2vec: Learning Word Embeddings using Lexical Dictionaries. In Conference on Empirical Methods in Natural Language Processing (EMNLP 2017), pages 254–263, 2017.

J. Turian, L. Ratinov, and Y. Bengio. Word representations: a simple and general method for semi-supervised learning. In Proceedings of the 48th annual meeting of the association for computational linguistics, pages 384–394. Association for Computational Linguistics, 2010.

S. Wang, S. Roller, and K. Erk. Distributional modeling on a diet: One-shot word learning from text only. In Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers), volume 1, pages 204–213, 2017.