Regularizing Hypernym Prediction with Recurrent Networks

Dobrovolskii V. A. (v.dobrovolskii@hotmail.com)

Russian State University for the Humanities, Moscow, Russia

This paper presents a new approach to regularizing hypernym prediction by treating the neural network used to predict hypernym vectors as a recurrent unit to output higher-level hypernyms based on the lower-level predictions and additionally penalizing higher-level error. It is shown that the model with such regularization outperforms the model proposed by Nayak [10] with the result improving even further in case sense embeddings are used for training. It is also demonstrated that it is possible to successfully combine this approach with other regularization techniques.

Key words: hypernym prediction, sense embeddings, word embeddings, recurrent neural networks

1 Introduction

Since the introduction of WordNet [3], ontologies and lexical databases have proved useful in many natural language processing tasks, such as coreference resolution, natural language inference and question answering. However, building such resources is hard and time-consuming; naturally, various research has been conducted to automate or at least semi-automate the process. One of the most important relations that needs to be established between words to build a semantic hierarchy is hypernymy or *is-a* relation. In this paper I am going to briefly outline the existing approaches to hypernymy extraction (section 2) and present a novel approach to further improve the state-of-the-art method (section 3). The code for reproducing the results will be made available at github.com/vdobrovolskii/hypernyms.

2 Related Work

Some of the approaches (for instance, [5]) would make use of lexico-syntactic patterns in text: for example, in *NP_0 such as {NP_1, NP_2 ... (and / or) NP_n}*, one can extract *NP_0* as the hypernym of all the other *NP*. It it evident, though, that such patterns can only produce a limited number of results. Also, such patterns are language-dependant, thus, it might not be easily possible to apply the same algorithm to another language. A similar approach would be to extract the information from dictionaries with the help of morphological and syntactic parsing [13]. While it does achieve high accuracy, it is still limited to what is already available in manually built dictionaries.
More promising solutions have been made possible by the rise of distributional semantics and the introduction of such models as Word2vec [9]. Several works (for example, [15, 12]) have been devoted to building binary classifiers to determine whether two candidate word vectors exhibit a hyponym-hypernym relation. However, as it is shown in [7], such classifiers tend to simply learn whether the second word vector in the candidate pair is a prototypical hypernym, while ignoring the relation between the two vectors.
Another approach, initially suggested in [4], involves training a regression model that takes as input a word vector and outputs the vector of its hypernym. In [10] it was demonstrated that deep feed-forward neural networks perform better than simple linear regression models. Finally, Ustalov et al. [14] showed that hypernym prediction could be improved by applying

certain linguistic constraints via regularization. More precisely, the loss function was modified in such a way that it would be penalized for outputting vectors too similar to the vectors of the synonyms of the desired hypernym.
The approaches described in [4] and [14] used k-means clustering to learn the *is-a* projection separately for each cluster of the input space. The number of clusters k was tuned on a development dataset. Yet, a recent work by Chen et al. [2] demonstrated that the clustering of input space hampers the prediction of unseen hypernyms.

3 Predicting Hypernyms with Recurrent Regularization

3.1 Baseline Approach

I am going to use a modified version of the approach suggested by Nayak [10] as the baseline model, the main difference being using cosine distance as the loss function instead of mean Euclidian distance used in the original paper. The main reason for this change is that the model produces a vector $\hat{Y}$ that does not correspond to any vector in the database, so the final prediction is the nearest neighbor of $\hat{Y}$ (nearest neighbors are computed by using cosine distance).
More concretely, the baseline is a deep feed-forward network. The number of hidden layers and their sizes have been tuned on a development dataset: the model has three hidden layers, the number of neurones is scaled relative to the input vector dimensions, where the scaling factors are 4, 2 and 2.

3.2 Recurrent Regularization

Intuitively we would expect that if there is a function $y = f(x)$ such as y is the hypernym of x, applying this function n times should yield an n^{th} level hypernym. However, none of the existing models possesses such a property. In this paper, I suggest using the baseline model as a recurrent unit, essentially turning it into a recurrent neural network. The model is fed an input vector X together with a single integer D representing the level of the hypernym the model is to predict. It is then expected to apply the recurrent unit D times to obtain the D^{th} level hypernym Y. That ensures encouraging the model to make lower level predictions closer to the expected hypernym vector (or at least to its co-hyponyms). Because the number of available training examples decreases dramatically with each level, to ensure correct predictions on higher levels, the loss function is weighted at each level:

$$F_{loss} = (1 - cos(\hat{Y}, Y)) \cdot \frac{M_1}{M_D} \tag{1}$$

where M_n is the number of training examples at level n.

3.3 Using Sense Embeddings

To my knowledge, no previous work has used sense embeddings for the task of hypernym prediction. While it is intuitive that sense embeddings should produce better performance than word embeddings, there is no consistent mapping between the senses output by any unsupervised method and the senses distinguished by people and available in human-made dictionaries. I address this problem in the following section by filtering out the senses that seem irrelevant to the embedding model or that are not available in the chosen hypernymy database.
For the purposes of this paper a pre-trained word embedding model from the RusVectōrēs

project [6] was chosen. The model was trained on a large corpus of 788 million words from the Russian National Corpus and Russian Wikipedia using the Continuous Skipgram algorithm [8] with the context window size of 2 words, producing 248,978 300-dimensional vectors.
To obtain the sense embeddings, the method proposed by Pelevina et al. was used [11]: with the default parameters it output 348,346 senses (approx. 1.4 senses per word). More concretely, this approach performs clustering of 200 nearest neighbors of each word vector by using Chinese Whispers algorithm [1], which is able to detect the number of clusters automatically. The sense vectors for each of the word-cluster combinations are then calculated.

4 Experiment 1: Sense Embedding Models

In this experiment two word embedding models (with and without recurrent regularization) are compared with two sense embedding models (similarly, with and without recurrent regularization).

4.1 Setup

Following [17] and [14], the data was obtained from Wiktionary[1]. For the purpose of getting higher level hypernyms more easily, a Word-Net like [3] database was built on top of the Russian section of Wiktionary. More concretely, the following information was extracted from each entry of the Wiktionary snapshot[2]:

1. lemma;
2. its part of speech;
3. its hyponyms and hypernyms of the same part of speech.

The words were not grouped into synsets. Also, the words without a corresponding pre-trained embedding were removed from the database. Then the database was "vectorized", i.e. words were replaced by their embeddings. To avoid expecting different outputs for the same input, only one hypernym was chosen for each hyponym, the main criterion being the cosine similarity between the two (thus choosing the most relevant hyponym-hypernym mappings to the given distributional model).
A similar procedure was used to build the dataset to predict sense embeddings: for all the senses available for each word, the sense of the hypernym maximizing the cosine similarity was chosen among all the senses of all the possible hypernyms. A certain threshold t was necessary to ensure that the chosen hyponym-hypernym mapping is relevant to the model so that the mappings with cosine similarity lower than the threshold would not added to the dataset. After manually studying the mappings, a threshold of $t = 0.6$ was chosen.
The training data for the baseline approaches was then built by taking the hyponyms as the input data and their nearest hypernyms as the output data. For the recurrent regularization approaches, hypernyms up to level 9 were used, the hypernymy level being one of the input features as well.
The data was then randomly split into training, validation (5%) and test data (2.5%). The hyponyms from the test data were chosen from the first level and then completely removed from the training and validation sets on all levels. Higher level hypernyms were not included in the test data to allow direct comparison of the recurrent models with the baseline.
The models were trained on batches of 1024 samples until the validation loss stopped

[1]wiktionary.org
[2]dated 01.04.2019

Table 1: The total number of hyponyms by level

Embedding type	1	2	3	4	5	6	7	8	9
Word	27,225	20,210	13,351	8,224	5,858	3,964	2,699	2,035	1,686
Sense $t = 0.6$	20,399	9,956	4,210	1,979	893	343	119	41	19

decreasing: that resulted in about 100 epochs for both baseline models and 500 epochs for regularized models. Despite the exponentially growing penalties, the models with recurrent regularization achieved smaller training and validation loss and greater accuracy.

4.2 Evaluation

The predicted outputs on the test datasets were examined manually. For more consistent evaluation, only one predicted hypernym was chosen for all the models, including those with recurrent regularization. For them, it was chosen automatically based on the confidence metric, which is defined as the cosine similarity between the predicted vector $\hat{Y}$ and the closest vector Y available in the embedding model:

$$\text{confidence} = \cos(\hat{Y}, Y) \tag{2}$$

Then the lowest hypernym with the confidence above a certain threshold was chosen to be the final prediction. The thresholds were set based on the validation data to be 0.6 for the word embedding recurrent model and 0.97 for the model trained on sense embeddings. The difference between the two optimal thresholds can be accounted for by homonymy, polysemy and the sense filtering. While the first two negatively affect the performance of any distributional model based on word embeddings, the filtering that was performed during the data preparation minimized the number of outliers in the training data for the sense embedding model, thus making it more apt to learn the correct hypernym projections.
The results are presented below:

Table 2: Performance comparison

Model	Correct (percent)
Word embeddings, baseline	49.19%
Word embeddings, recurrent regularization	53.60%
Sense embeddings, baseline	39.33%
Sense embeddings, recurrent regularization	**62.43%**

The recurrent regularization does seem to improve the performance of models based both on word and sense embeddings. While the increase in performance is not dramatic in case of word embeddings, the regularized sense embedding model outperforms its own baseline as well as both word embedding models despite the significantly higher density of the vector space (348,346 and 248,978 vectors, respectively). In the next section I will lay focus only on sense embedding models to see whether the result can be further improved by combining recurrent regularization with other regularization techniques.

5 Experiment 2: Combining Regularization Techniques

In this experiment, the effects of combining recurrent regularization with other regularization techniques are studied. More concretely, I will be using a variant of neighbor regularization proposed by Yamane et al. [16], where the loss function is penalized if the output vector $\hat{Y}$ is closer to n nearest neighbors of the expected hypernym vector Y.

5.1 Setup

In the previous section it was shown that the sense embedding models outperform the models based on word embeddings. Consequently and due to time constraints, in this section I will only compare the performance of the sense embedding models.
The same dataset as before was used for this experiment, though the sense filtering parameter was set to $t = 0.7$ to improve the quality of the training data.

Table 3: The total number of hyponyms by level

Embedding type	1	2	3	4	5	6	7
Sense t = *0.7*	15,455	6,350	2,190	806	234	52	12

The loss function for the models in this experiment is defined as follows:

$$F_{loss} = (1 - cos(\hat{Y}, Y) + NeighborTerm) \cdot RecurrentTerm \tag{3}$$

$NeighborTerm$ is defined below. Following [14], one of the neighbors was chosen to be the input vector itself. The number of neighbors was set to $n = 20$.

$$NeighborTerm = cos(z, \hat{y}) - cos(y, \hat{y}) \tag{4}$$

where z is such a vector from $NearestNeighbors(y, n) \bigcup \{x\}$ that has the greatest cosine similarity $cos(y, z)$.
As it was presented in the previous section, $RecurrentTerm = \frac{M_1}{M_D}$, where M_n is the number of training examples at level n. For models that do not use neighbor or recurrent regularization, the terms were set equal to 0 and 1, respectively.
Four models were built in total: 1) ModelRN, combining both recurrent and neighbor regularization; 2) ModelR, using only recurrent regularization; 3) ModelN, using only neighbor regularization; 4) ModelBaseline without any regularization. The models were trained on batches of 1024 samples until the validation loss stopped decreasing.

5.2 Evaluation

The results are presented below.

Table 4: Performance comparison

Model	Correct, %
modelBaseline	42,1%
modelN	49,4%
modelR	54,9%
modelRN	**67,3%**

It is evident from the table that it is possible to combine recurrent regularization with other regularization techniques to yield even better results.

6 Discussion

In the previous sections, I demonstrated how recurrent regularization can be used to improve hypernym prediction. While it is arguable that such accuracy can make the process of taxonomy building significantly easier, by using confidence metric (equation 2) it is possible

Table 5: The correlation between confidence and accuracy, modelRN

Confidence	Accuracy	Test data fraction
0,99	88,9%	4,5%
0,98	79,2%	27,9%
0,97	77,6%	61,7%
0,96	74,3%	72,2%
0,95	73,3%	80,1%
0,75	67,3%	100%

to sort the predicted hypernyms by the probability that they are correct. This allows the expert to first concentrate on stronger predictions to check them more efficiently.
Further research may be devoted to obtaining more training data, to obtaining training data of higher quality and to determining how the choice of algorithm to build sense embeddings can affect the performance of the model. Also, the possibility of extracting other relations (such as *is-part-of* or *is-property-of*) from vector models should also be studied.

References

[1] Christian Biemann. Chinese whispers - an efficient graph clustering algorithm and its application to natural language processing problems. 2006.

[2] Hong-You Chen, Cheng-Syuan Lee, Keng-Te Liao, and Shou de Lin. Word relation autoencoder for unseen hypernym extraction using word embeddings. In *EMNLP*, 2018.

[3] Christiane Fellbaum. *WordNet: An Electronic Lexical Database*. Bradford Books, 1998.

[4] Ruiji Fu, Jiang Guo, Bing Qin, Wanxiang Che, Haifeng Wang, and Ting Liu. Learning semantic hierarchies via word embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1199–1209, 2014.

[5] Marti A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *COLING*, 1992.

[6] Andrey Kutuzov and Elizaveta Kuzmenko. *WebVectors: A Toolkit for Building Web Interfaces for Vector Semantic Models*, pages 155–161. Springer International Publishing, Cham, 2017.

[7] Omer Levy, Steffen Remus, Christian Biemann, and Ido Dagan. Do supervised distributional methods really learn lexical inference relations? In *HLT-NAACL*, 2015.

[8] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013.

[9] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, pages 3111–3119, USA, 2013. Curran Associates Inc.

[10] Neha Nayak. Learning hypernymy over word embeddings. 2015.

[11] Maria Pelevina, Nikolay Arefiev, Chris Biemann, and Alexander Panchenko. Making sense of word embeddings. In *Proceedings of the 1st Workshop on Representation Learning for NLP*, pages 174–183, Berlin, Germany, August 2016. Association for Computational Linguistics.

[12] Stephen Roller, Katrin Erk, and Gemma Boleda. Inclusive yet selective: Supervised distributional hypernymy detection. In *COLING*, 2014.

[13] V.Sh. Rubashkin, V.V. Bocharov, L.M. Pivovarova, and B.Yu. Chuprin. The approach to ontology learning from machine-readable dictionaries. In *Computational Linguistics and Intellectual Technologies*. Dialogue, International Conference on Computational Linguistics, 2010.

[14] Dmitry Ustalov, Nikolay Arefyev, Chris Biemann, and Alexander Panchenko. Negative Sampling Improves Hypernymy Extraction Based on Projection Learning. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 543–550, Valencia, Spain, April 2017. Association for Computational Linguistics.

[15] Julie Weeds, Daoud Clarke, Jeremy Reffin, David J. Weir, and Bill Keller. Learning to distinguish hypernyms and co-hyponyms. In *COLING*, 2014.

[16] Josuke Yamane, Makoto Miwa, and Yutaka Sasaki. Distributional hypernym generation by jointly learning clusters and projections. 12 2016.

[17] Torsten Zesch, Christof Müller, and Iryna Gurevych. Extracting lexical semantic knowledge from wikipedia and wiktionary. In *LREC*, 2008.