NATURAL LANGUAGE PROCESSING HOMEWORK 3

WORD-SENSE DISAMBIGUATION

RICCARDO GOZZOVELLI

1849977

ACCADEMIC YEAR 2018-2019

INTRODUCTION

In this report we investigate a very well known problem in NLP called word-sense disambiguation (WSD). WSD is the task in which we try to associate every target word to its most suitable sense. This can be done only in presence of sense annotated datasets. Unfortunately there are not many of them available because they are very expensive to produce but also they are dependent, to a certain extent, on the language used. The richer is the vocabulary of a language, the more difficult will be the task of annotating the correct sense for words. Polysemy[1] and homonymy[2] are two of the biggest problems in WSD. Therefore only a small subset of annotated words is present in the available datasets. Nonetheless even if the premises are very discouraging, many have tried to confront WSD with different techniques that produced always different models. The purpose of this report is then to dive into the WSD task by presenting various developed systems that were inspired from different other models.

SENSE INVENTORY AND DATASET

In order to associate a word to their senses it is necessary to have a sense inventory. The more informations are contained in the inventory, the more fine-grained can be the search for the correct meaning of a word. The main advantage of big sense inventories is that any system exploiting them is able to deal with most, if not all, the senses of a word. But at the same time if we don't limit the sense inventory informations, the system might suffer from over-specification. The sense inventory that we used is WordNet, one of the biggest sense inventory today available. In Wordnet words that share the same meaning are grouped into sets called "synsets", leading to a total of 117'000 synsets. Synsets are also linked together by different "conceptual relations". Of particular interest is the super-subordinate relation also called "hypernymy", used to link general synsets to more specific ones. We will talk more in depth about hypernymies later on. With a sense inventory at hand, the second thing required is a sense annotated dataset. Among all the available ones we opted for the SemCor dataset, one of the largest annotated lexical resource with about 200K words. Actually we also tried to use part of the combined version of SemCor with the OMSTI dataset, reaching more than 100K sentences, but we were not able to overcome memory problems given by the large size of the data to manage.

PRE-PROCESSING

The pre-processing phase of the dataset required first the removal of all punctuation symbols from the sentences but also a cleaning of any special character that was not correctly formatted. The dataset is then obtained by tokenizing the sentences and extracting the lemma of every token. We then removed all the repeated sentences that did not contain any useful information, i.e. sentences composed by words that didn't have a single annotated word. Finally we ordered and grouped the sentences by ascending length. This last step is particularly helpful for the next one, which involves the representation of sentences with embeddings.

ELMo EMBEDDINGS

Representing sentences with correct embeddings is crucial for the development of a well-perfoming model. The embeddings should represent not only the syntactic and semantic characteristics of a word but also the context in which a word is used. ELMo embeddings are capable to exactly represent both of these informations. Elmo is infact a *deep contextualized word representation based on a deep bidirectional language model trained on very large text corpus*[3]. ELMo embeddings are generally more accurate than other embeddings and can improves significantly the state-of-the-art for many NLP tasks. The generation of the embeddings can be memory and time consuming, in particular if the data is directly feed into ELMo without taking any precaution. We decided to implement this "embedding layer" outside the main neural network because we wanted to store, and have available at any time, the embeddings for every sentence. By feeding ELMo with batches of sorted sentences with the same length we:

- prevented ELMo from taking care of padding, so as to have the most accurate embeddings;
- we decreased the total time required for the generation of all the embeddings; this was also proven to be effective in Le et al.[3].

In the end the embeddings were stored into a .pickle file, a particular format used to serialize/de-serialize python object structure.

---

1 One word can have different meanings.
2 Different words with the same meaning.
3 Definition given in the paper: arXiv:1802.05365

OUTPUT VOCABULARY

The output vocabulary is the set containing all the labels from which the model has to extract answers. Without any further modification, the total number of labels contained is 56'865 and only 1/3 are sensekeys. We can easily guess how difficult and time consuming can be for a system to deal with such an high number of labels but still we have been able to implement systems that can handle this situation. Nonetheless we tried different techniques in order to lessen this heavy load. The method that we followed is based on label compression. The idea of label compression, which proven to be effective in Vial et al.[2], exploits the structure on which Wordnet is built on. Through synsets and relations it is possible to group polysemous words and obtain a more compact representation. A synset is a group of different words that share the same meaning. So, instead of predicting for a word the sensekey, we can generalize the procedure and let the model predict the synset instead. This decreases the number of total entries of the output vocabulary[4] and it affects also the time requested for training. But synsets are still to specific in many situations. Let's consider the example shown in Figure 1. The image shows a tree structure that represents the hypernimy/hyponymy relations between two different senses of the word "mouse". An hypernymy is a generalization of a term, an hyponymy is the opposite. At some point both senses will "converge" to a same parent node. By choosing the node immediately below the common node we are always sure to correctly distinguish between the two senses desptie the high degree of generalization. The idea is therefore to compare all the hypernymy relations of all the sensekeys contained in the output vocabulary and extract from them the lowest hypernymy that allow to correctly disambiguate the senses. Not all the sensekeys are associated to hypernymy relationships, in particular verbs and adjectives, so the final output vocabulary will be composed by both sensekeys and hypernyms. As a consequence the number of output labels decreases to 41045.

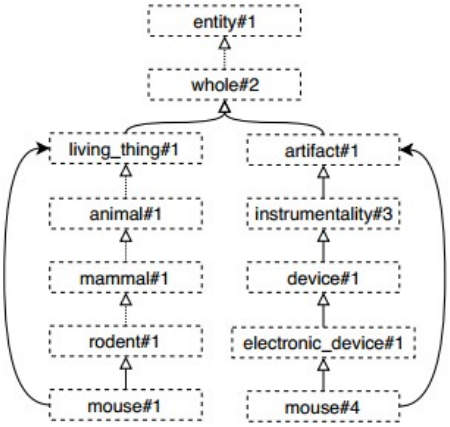

Figure 1 – Hypernymy relation for the word "mouse"

MODELS

The models we have developed take strong inspiration from the ones implemented in Raganato et al.[1]. In particular we were interested in understanding how and if it could be possible to improve some of the architectures there proposed. We focused on two architectures in particular, the "Bidirectional LSTM tagger" and the "Attentive Bidirectional LSTM tagger with multi-task learning". The first architecture is very symple. Two bidirectional LSTM layers, each composed by a forward and a backward layer, are stacked one on top of the other. The output generated is then passed to a dense layer that with a softmax activation function produce the probabilty for each of the classes contained in the output vocabulary. Concerning this architecture we have had since the beginning problem of overfitting. To deal with overfitting, we used the concept of variational dropout proposed by Gal et Ghahramani [4]. For a RNN it is possible to specify three different types of dropout, one for the input, one for the output and one for the state. As we can see from [Figure 2] by using variational dropout we let the RNN drop all the possible connections, with a certain probability value, between inputs, states and outputs. In particular with variational dropout we allow the repetion of the same dropout mask at each time step. In other words the same network units will be dropped at each time step.

4 Reaching a total of 49'451.

The first architecture is also used as a starting point for the second architecture. Here an attention layer computes a vector that is then concatenated with the output of the last layer of the second BLSTM. The idea of an attention layer is to get an additional look to all the information produced by the hidden states of the LSTM layer, in order to have a more robust result. This concept is implemented by computing a weighted sum of the hidden states. The formal procedure followed is the same implemented in Raganato et al.[1]. Concerning the "multi-task learning" part we simply had to implement multiple dense layers. The loss is then computed as the weighted sum of all the losses. The losses taken in account are the ones related to both fine-grained and coarse-grained tasks. The term fine-grained is used to refer to the task of predicting the correct sensekey for a target word. This is the main task that we wanted to achieve, but at the same time it suffers from over-specification and sparsity because of the great number of classes to predict. The second term refers to the fact that words can be associated to semantic domains (sport, biology, politic, etc) and ontological classes (verb.competition, noun.animal, etc). Being this latter tasks more general that the first one, we believe that combining fine-grained WSD task with coarse-grained WSD tasks, might increase the perfomances of the entire model.

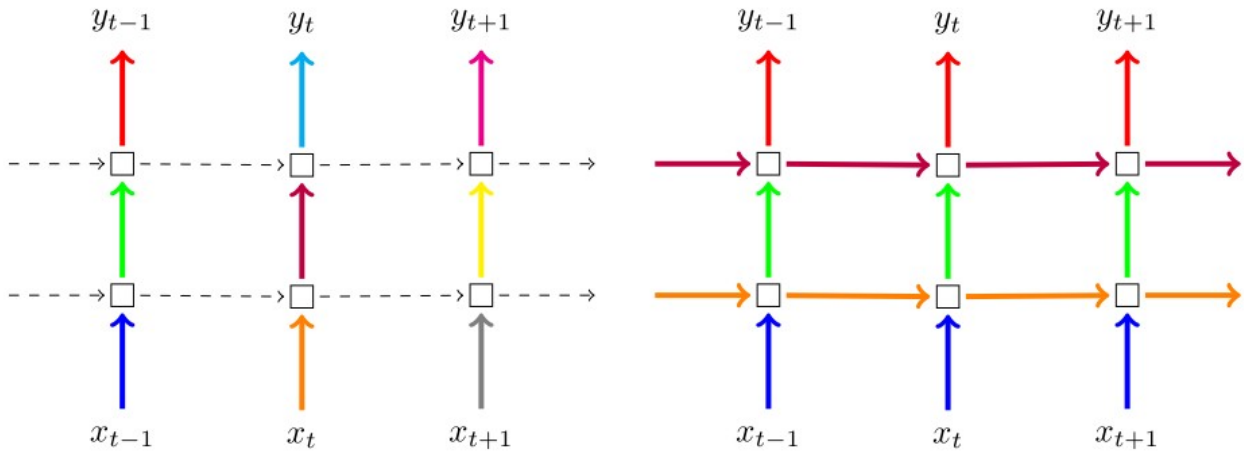

Figure 2 – Dropout in RNN, dashed line = no dropout, colored line = dropout

TRAINING

As a development set we used the Semeval2007 dataset, the smallest one among all the available ones. Both models have been trained for 25 epochs using Adadelta with learning rate set to 1 and batches of 32 elements. The number of hidden elements for one LSTM layer is 1024, the size of the embeddings used is 400 and all the sentences, and the labels aswell, have been padded to a total length of 60 tokens. In the first architecture we played with different combination for the dropout values till overfitting reduced. In the second architecture we weighted the coarse-grained task losses (domain and lexname predictions) with a factor λ between 0 and 1, in order to understand if one of the two sub-tasks could be able to affect more the total perfomances.

RESULTS

Before analyzing the results obtained, we need to introduce the particular evaluation method used to compare the perfomances of the models. We are interested in developing models that are able to disambiguate words, but during training all the models predicted labels both for words to disambiguate and for words that didn't need to be disambiguated. For a correct evaluation in WSD we have to consider for a word only the labels that are associated to it. In case a word was never met during training, which means that we do not know to which label it must be associated to, we use a backup strategy known as Most Frequent Sense (MFS). In this way we always provide an answer. With this in mind, the architecture that performed better is the second one with values of the hyperparameters shown in Table 1:

Dataset	SemCor 3.0
Embedding size	400
Hidden size	1024
Batch size	32
Adadelta	1
Lambda 1	1
Lambda 2	1
Padding	60
Compression	No

Table 1: Multi-task Attentive BLSTM, best model

By setting to 1 the weights assigned to both of the two coarse-grained tasks, the best result is obtained. Actually we noticed during the various tests that between the two auxiliary tasks the second one (prediction of ontological classes) is more impactful than the other. Concerning the embedding size, the ELMo embeddings have an original dimension of 1024. However reducing the total size didn't affect at all the perfomances but only the time required for training. The same is true for padding, increasing the total length led to overfitting, while decreasing it resulted in worst perfomances. Concerning compression, only synset compression proved to achieve better results. On the other hand hypernym compression turned out to be harmful for the WSD task. We believe that this result might be due to the fact that hypernyms are to general for our situation.

In Table 2 we show the results obtained by all the models that we have developed (underlined result correspond to the best result among all the models developed) and we compare them with the results obtained from Raganto et al.[1].

	Dev	Test				
	SE07	SE2	SE3	SE13	SE15	ALL
BLSTM	55,8	61,5	60,1	56,8	55,4	58,8
BLSTM (wordnet compression)	56,5	63,8	61,7	61,9	56,4	61,3
BLSTM (hypernym compression)	54,7	61,7	57,2	61,2	53,3	58,8
BLSTM + att + Lex + Domain	*60,9*	*64,6*	*64,5*	60,5	*60,2*	*62,8*
BLSTM + att + Lex + Domain (hypernym compression)	58,5	62,8	61,9	*63,5*	59,3	61,9

BLSTM	61,8	71,4	68,8	65,6	69,2	68,9
BLSTM + att + Lex + Pos	**64,8**	**72,0**	**69,1**	**66,9**	**71,5**	**69,9**

Table 2: F-Scores (%) for English all-words fine-grained WSD. The result in italic are associated to the best model we developed. The second table contain the results obtained by Raganato et al.[1]

The table shows that our best model is still far from the state-of-art. In particular it performs worst than the simple BLSTM architecture. We believe that the reason why we were not able to achieve same, or better result, is due to the presence of padding. In Raganato et al.[1] is applied sentence splitting but no padding, therefore we might be loosing some information. Unexpectedly we can observe that for the Semeval2013 dataset, using hypernym compression returned the highest result. In Table 3,4 and 5 we present instead the results obtained from the three different coarse-grained tasks involving babelnet synset, wordnet domain and lexical name predictions. The attentive BLSTM with multi-task learning and without compression performs again better than any other model.

	Dev	Test				
	SE07	SE2	SE3	SE13	SE15	ALL
BLSTM	56,3	64,0	61,2	60,8	57,0	61,1
BLSTM (wordnet compression)	56,5	63,8	61,7	61,9	56,4	61,4
BLSTM (hypernym compression)	20,0	33,0	23,0	13,9	27,9	24,6
BLSTM + att + Lex + Domain	**61,3**	**67,0**	**65,6**	**64,5**	**61,8**	**65,0**
BLSTM + att + Lex + Domain (hypernym compression)	21,8	34,1	25,5	14,4	28,7	25,9

Table 3: F1-Score (%) for English all-words coarse-grained WSD on BabelNet synsets.

	Dev	Test				
	SE07	SE2	SE3	SE13	SE15	ALL
BLSTM	87,3	89,4	85,1	75,9	80,4	83,9
BLSTM (wordnet compression)	86,6	90,1	85,4	76,6	80,0	84,2
BLSTM (hypernym compression)	76,5	81,2	71,1	60,1	73,5	72,5
BLSTM + att + Lex + Domain	**89,5**	**91,3**	**87,1**	**78,2**	**83,3**	**86,0**
BLSTM + att + Lex + Domain (hypernym compression)	79,3	81,9	72,1	60,3	76,0	73,5

Table 4: F1-Score (%) for English all-words coarse-grained WSD on WordNet Domains.

	Dev	Test				
	SE07	SE2	SE3	SE13	SE15	ALL
BLSTM	71,9	80,0	75,0	71,8	73,8	75,5
BLSTM (wordnet compression)	70,8	79,6	75,4	72,6	73,4	75,5
BLSTM (hypernym compression)	63,1	75,1	67,2	67,2	66,4	69,3
BLSTM + att + Lex + Domain	**73,8**	**82,7**	**79,2**	**75,4**	**76,9**	**78,8**
BLSTM + att + Lex + Domain (hypernym compression)	65,7	77,0	71,8	68,9	70,6	72,2

Table 5: F1-Score (%) for English all-words coarse-grained WSD on Lexical Names.

For the first coarse-grained task (babelnet synset prediction), we can see that the hypernym compression method is totally incorrect. This is due again to the fact that it is too general with respect to the task. On the other hand, using synset compression or no compression at all, produce alternate best result depending on the dataset. For the second coarse-grained task (wordnet domain predictions), the scores achieved are particularly high due to the nature of the task itself. Many words are infact associated to the generic class “factotum”, therefore the task is particularly easy. Also in this case depending on the dataset, synset compression and no compression will show alternate best results. The same is true also for the last coarse-grained task (lexical name predictions).

As a final note, the results of the BLSTM with attention layer and multi-task learning using wordnet synset compression are not present because we didn’t have enough time to finish the training of the model. Nevertheless, we will add them and show them in the presentation if the results obtained will be worth noting.

REFERENCES

[1] A. Raganato, C. D. Bovi, and R. Navigli, “Neural sequence learning models for word sense disambiguation”
[2] L. Vial, B. Lecouteux and D. Schwab, “Sense Vocabulary Compression through the Semantic Knowledge of WordNet for Neural Word Sense Disambiguation”
[3] M. Le, M Postma, J. Urbani and P. Vossen, “A Deep Dive into Word Sense Disambiguation with LSTM”
[4] Y. Gal and Z. Ghahramani, “A Theoretically Grounded Application of Dropout in Recurrent Neural Networks”