Knowledge-based Word Sense Disambiguation using a Bidirectional LSTM

Kun Li

Department of Computer Science, the University of Texas at Dallas

Kxl162530@utdallas.edu

Abstract

In this paper I present a clean, yet effective, model for *word sense disambiguation*. My approach leverages a *bidirectional long short-term memory* network which is shared between all words. Central to this technology is a learned word vector embedding that has separate word vectors for each sense of a word. The model is trained end-to-end, directly from the raw text to sense labels, and makes effective use of word order. I evaluate my approach on two standard datasets, using identical hyperparameter settings, which are in turn tuned on a third set of held out data. I employ no external resources (e.g. knowledge graphs, part-of-speech tagging, etc.), language specific features, or hand-crafted rules, but still achieve statistically equivalent results to the best state-of-the-art systems.

1 Introduction

It's common to see word vector embeddings as ways to represent words that are fed into machine learning models. These word vectors are often trained on large text corpora and convey some sort of relationship relative to surrounding context words. However, many common word vector embeddings would include the same vector representation of a word even though it might be used in different ways between contexts. Consider the following two examples for instance:

1. He sat down beside the Seine river **bank**.
2. He deposited the money at the Chase **bank**.

Although 'bank' is clearly being used in two different senses in these two examples, they would be represented by the same vector. Providing the sense in which a word is used could give machine learning systems valuable information that can contribute to how well they can perform a task. Word Sense Disambiguation (WSD) is considered an AI-Complete problem (Navigli 2009), meaning it is one of the centrally difficult problem in AI, so potential solutions and effective approaches towards this problem can have far-reaching implication for other key AI tasks. Consider the task of named entity recognition (NER), for instance. In NER, I attempt to classify different entities as organization, people, places, etc. In the first case, I should identify "bank" as a location instead of an organization, as is the case in the second sentence. By being able to predict word senses, I can help clarify these ambiguous situations. Furthermore, we can develop separate word vectors for each sense, which can ideally allow us to better represent words within their own context.

In this paper, I describe one WSD algorithm. It is based on a Long Short Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997). Since this model can consider word order when classifying, it perform significantly better than a continuous bag of words model (Word2vec) (Mikolov et al., 2013; Iacobacci et al., 2016).

Organization: I review related work in Section 2. I introduce dataset in section 3, and the supervised WSD algorithm in Section 4. Experimental results are discussed in Section 5. I provide further discussion in Section 6 and future work in Section 7.

2 Background/Related Work

In this section I introduce the most important underlying techniques for my proposed model.

2.1 Bidirectional LSTM

Long short-term memory (LSTM) is a gated type of recurrent neural network (RNN). LSTMs were introduced by Hochreiter and Schmidhuber (1997) to enable RNNs to better capture long term dependencies when used to model sequences. This is achieved by letting the model copy the state between timesteps without forcing the state through a non-linearity. The flow of information is instead regulated using multiplicative gates which preserves the gradient better than e.g. the logistic function. The bidirectional variant of LSTM (BLSTM) (Graves and Schmidhuber, 2005) is an adaptation of the LSTM where the state at each time step consist of the state of the two LSTMs, one going left and one going right. For WSD this means that the state has information about both preceding words and succeeding words, which in many cases are necessary to correctly classify the sense.

2.2 Word embeddings by GloVe

Word embeddings is a way to represent words as real valued vectors in a semantically meaningful space. *Global Vectors for Word Representation* (GloVe), introduced by Pennington et al. (2014) is a hybrid approach to embedding words that combine a log-linear model, made popular by Mikolov et al. (2013), with counting based co-occurrence statistics to more efficiently capture global statistics. Word embeddings are trained in an unsupervised fashion, typically on large amounts of data, and can capture fine grained semantic and syntactic information about words. These vectors can subsequently be used to initialize the input layer of a neural network or some other NLP model.

2.3 Related Work

Google has released a WSD dataset, with which the paper “Semi-supervised Word Sense Disambiguation with Neural Models” by Yuan et al. is associated. Yuan et al. explore an LSTM based supervised WSD model. Their LSTM model is trained on sense supervised unlabeled data and generates context vectors based on previous history. To predict senses, they hold out the sense evaluated word and run the entire context through their LSTM. The LSTM predicts the top k predictions for the held-out word. In addition to the LSTM, they have sense-labeled sentences for each polysemous word in their vocabulary. After finding the top k word predictions, they run the sense-labeled sentences through the LSTM, holding out the same word and again generating the top k word predictions. They then use a nearest-neighbor classifier to establish which sense best represents the original word based on those k predictions.

Kageback and Salomonssn detail their usage of bidirectional LSTMs for WSD in “Word Sense Disambiguation using a Bidirectional LSTM.” Their LSTMs use Glove embeddings to distinguish senses by computing “a probability distribution over the possible senses corresponding to that word,” similar to how I will approach sense classification with my LSTM.

3 Dataset

For this project, I use the publicly available Google Word Sense Disambiguation Corpora to train my sense-tagged word vectors as well as train and evaluate my LSTM models. The Google WSD Corpora, released on January 17, 2017, is one of the largest labeled WSD corpora, and consists of the popular SemCor and MASC datasets manually labeled with NOAD and WordNet senses.

Google commissioned the labeling of these datasets by having expert linguists label a small seed set used as a gold standard, and then having many other workers label the remainder of the datasets. In developing the corpus, Google prioritized having a high inter-rater reliability score to ensure high quality of tagged tokens. They achieved a Krippendorff’s Alpha score of 0.869, implying the labeling are highly reproducible (usually a score above 0.67 is considered acceptable). However, as a result, despite having 1.1 million tokens, only 248k are polysemous tokens labeled with word senses, with many polysemous tokens instead being tagged with an ambiguous sense (since the reliability score for the tags on these tokens is lower than their standard).

For my experiments with LSTM models, I train the LSTM with at least the entire MASC dataset, with some experiments having the LSTM be trained with as much as ¾ of the SemCor dataset.

4 Supervised WSD

4.1 Creation of Sense Embeddings

My model for representing a dictionary’s sense vector relies on the pre-trained GloVe word embeddings, and on the sense’s definition in the dictionary. In practice, my method is very similar to the one presented by Ferrero et al. (2017), who create sense embeddings in knowledge-based word sense disambiguation. My sense vector is computed as the normalized sum of all the terms’ vectors present in the sense’s definition, weighed in function of their part of speech (noun, verb, adjective or adverb), and weighted by their inverse document frequency (IDF), i.e. the inverse of the number of occurrence in the entire dictionary. More formally, I denote:

• $\mathrm{D(s)} = \{ \omega_0, \omega_1, \omega_2, \ldots, \omega_\mathrm{n}\}$ the definition of sense S in the dictionary

• $\mathrm{pos}(\omega_\mathrm{n}) = \{n, v, a, r\}$ the part of speech of the term ω_n (noun, verb, adjective or verb)

• $\mathrm{weight(pos)}$ the weight associated with a specific part of speech

• $\mathrm{idf(f_n)}$ the IDF value of ω_n, computed as $\log \frac{N_{tot}}{N(\omega_n)}$, with N_{tot} being the total number of definitions in the dictionary (206, 941 in WordNet 3.0), and $N(\omega_n)$ the number of definitions containing at least one occurrence of the word ω_n.

The definition of the vector of the sense S, denoted $\phi(S)$ is hence the following:

$$\phi(\mathrm{S}) = \sum_{i=0}^{n} (\phi(\omega_n) \times weight(pos(\omega_n)) \times idf(\omega_n))$$

$\phi(S)$ is then normalized, to be the same length as the vectors contained in the word embeddings model (generally the length is 1). The chosen POS weights are the same that Ferrero et al. (2017) used for representing English and Spanish sentences, but they can be trained as parameters of the model.

4.2 The Model

Given a document and the position of the target word, i.e. the word to disambiguate, the model computes a probability distribution over the possible senses corresponding to that word. The architecture of the model, depicted in Figure 1, consist of a Dense layer, a hidden layer, and a BLSTM. See Section 2.1 for more details regarding the BLSTM. The BLSTM and the hidden layer and the dense all share parameters over all word types and senses. This structure enables the model to share statistical strength across different word types while remaining computationally efficient even for a large total number of senses and realistic vocabulary sizes.

The input to the BLSTM at position n in document D is computed as

$$\mathrm{x_n} = W^x v(\omega_n), n \in \{1, \dots, |D|\}$$

Here, $v(\omega_n)$ is the one-hot representation of the word type corresponding to $\omega_n \in \mathrm{D}$. A one-hot representation is a vector with dimension V consisting of |V|-1 zeros and a single one which index indicate the word type. This will have the effect of picking the column from W^x corresponding to that word type. The resulting vector is referred to as a word embedding. Further, W^x can be initialized using pre-trained word embeddings, to leverage large unannotated datasets. In this work GloVe vectors are used for this purpose, see Section 2.2 for details.

The model output,

$$\mathrm{y(n)} = \mathrm{Dense}(\mathrm{a}, W_{\omega_n}^{ay}, b_{\omega}^{ay})$$

is the predicted sense vector for the word at position n, where $W_{\omega_n}^{ay}$ is the weights for the Dense layer corresponding to the word type at position n. Further, the hidden layer a is computed as

$$\mathrm{a} = W^{ha}[h_{n-1}^{L}; h_{n+1}^{R}] + b^{ha}$$

where $[h_{n-1}^{L}; h_{n+1}^{R}]$ is the concatenated outputs of the right and left traversing LSTMs of the BLSTM at word n. W^{ha} and b^{ha} are the weights and biases for the hidden layer.

Loss function The parameters of the model, $\Omega = \left\{W^x, \Theta_{BLSTM}, W^{ha}, b^{ha}, \{W_{\omega}^{ay}, b_{\omega}^{ay}\}_{\forall \omega \in V}\right\}$, and fitted by minimizing the MSE error.

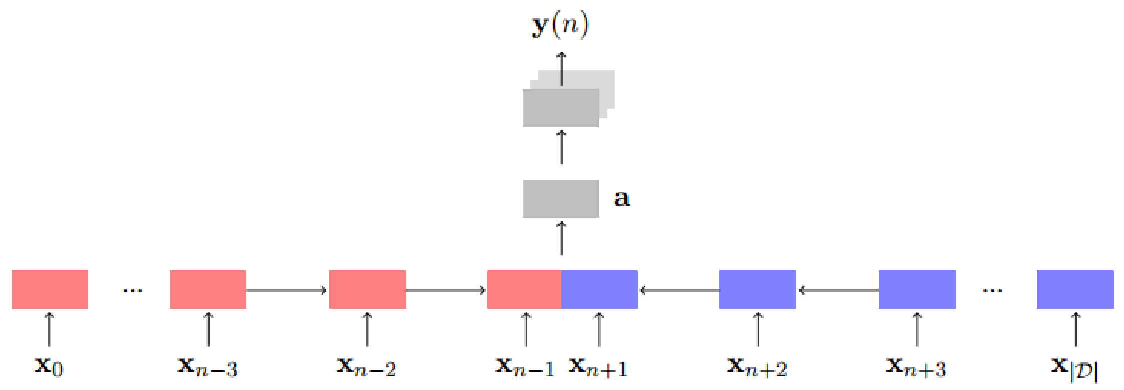

Figure 1: A BLSTM centered around a word at position n. Its output is fed to a neural network sense classifier consisting of one hidden layer with linear units and a softmax. The softmax selects the corresponding weight matrix and bias vector for the word at position n.

5 Experiment

5.1 Evaluation Metric

The testing set for evaluation my approaches consists of 30k tokens from the SemCor dataset. I use cosine similarity to evaluate the models. The fewer cosine similarity between the predicted sense vector and the actual sense vector, the more close these two vector at the vector space.

5.2 Experimental settings

The hyperparameter settings used during the experiments, presented in Table 1, were tuned on a separate validation set with data. The source code, implemented using Keras with TensorFlow (Abadi et al., 2015) backend, has been released as open source.

Embeddings The embeddings are initialized using a set of freely available GloVe vectors trained on Wikipedia and Gigaword. Words not included in this set are initialized from N(0,0.1). To keep the input noise proportional to the embeddings it is scaled by σ_i which is the standard deviation in embedding dimension I for all words in the embeddings matrix, $W^x\sigma_i$ is updated after each weight update.

Data preprocessing The only preprocessing of the data that is conducted is replacing numbers with a <number> tag. Words not present in the training set are considered unknown during test. Further, I limit the size of the context to max 140 centered around the target word to facilitate faster training.

Hyperparameter	Range searched	Value used
Embedding size	{100, 200}	100
BLSTM hidden layer size	[50, 100]	$2 * 74$
Dropout on word embeddings $\mathbf{x}_n$	[0, 50%]	50%
Dropout on the LSTM output $[\mathbf{h}^L_{n-1}; \mathbf{h}^R_{n+1}]$	[0, 70%]	50%
Dropout on the hidden layer $\mathbf{a}$	[0, 70%]	50%
Dropword	[0, 20%]	10%
Gaussian noise added to input	[0, 0.4]	$\sim \mathcal{N}(0, 0.2\sigma_i)$
Optimization algorithm	-	Stochastic gradient descent
Momentum	-	0.1
Initial learning rate	-	2.0
Learning rate decay	-	0.96
Embedding initialization	-	GloVe
Remaining parameters initialized	-	$\in \mathcal{U}(-0.1, 0.1)$

Table 1: Hyperparameter settings used for both experiments and the ranges that were searched during tuning. "-" indicates that no tuning were performed on that parameter.

5.3 Results

My proposed model achieves the top scores. The model built with TensorFlow trained on all of the MASC dataset and roughly 3/4 of the SemCor dataset. It trains its own word embeddings during the training process. These word embeddings are still sense-tagged like our pre-trained vectors. The results were better than expected, getting a weighted accuracy of 79.95% on our evaluation set.

Model	Accuracy
Most Likely Sense (Baseline)	34.77%
Keras LSTM: 10% of MASC, pre-trained embeddings	24.16%
Keras LSTM: all of MASC, pre-trained embeddings	46.37%
Tensorflow LSTM: most of all data	79.95%

Table 2: Results for LSTM models

6 Discussion

6.1 Analysis and Insights

When developing my TensorFlow LSTM, we noted a high accuracy. This does make sense, given that in this model, the word vectors were generated on the fly as part of the LSTM's optimization and hence tailed to the specific task at hand, whereas the pre-trained vectors used in the Keras LSTM were made to optimize a different objective. However, I feel there may be potential overfitting of the model to the data, given that I trained this model on the full vocabulary. Nevertheless, I feel these results show potential for the use of LSTMs in tasks like WSD.

6.2 Key Challenges

In developing this project, I encountered various key challenges with the amount of data and the specific demands of the task.

One of the big challenges I encountered with my WSD models was the lack of enough labeled data. While the Google WSD corpus is one of the largest of its type, it is still quite small for my purpose. For example, when training word vectors, I attempted to train 64-dimensional vectors for the top 20,000 words (this is both a modest number of dimensions and size of vocabulary; many practical applications use larger values). This works has roughly 1.28 million parameters to train, by only 1.1 million tokens (at max), meaning the vectors were highly susceptible to being influenced by faulty or outlying data. Also, the amount of time I can see a word in a sense is necessarily more infrequent than with sense-unlabeled word vector training. Furthermore, I wouldn't be able to necessarily see the different senses of each word represented equally in all their respective contexts, meaning rarer senses have less information. I anticipate that with more labeled training data, I would have much higher quality word vectors and LSTM models that would lead to higher accuracy scores.

Part of my task presented an interesting problem: training on sense-labeled data while attempting to predict on sense-unlabeled data for the LSTM approach. To overcame this, I attempt to use sense embedding based on WordNet to provide an initial history with sense-labeled data for the model to build off, essentially eliminating the problem.

7 Conlusion

Overall, I find that an LSTM and word vector focused approach show great promise in tackling the AI-complete problem of WSD. While there are some difficulties in modeling the problem in a format conducive to LSTM application (especially given the lack of large corpora of publicly-available, labeled data), it is possible to achieve promising results through clever workarounds, manipulations, and hyperparameter tuning.

7.1 Future Directions

In the future, I would be interested in exploring the application of my sense-tagged word vectors to traditional NLP tasks such as machine translation and NER. This would involve the task of both labeling the senses and assigning them high quality word vectors. By assessing performance on these tasks, I can gain a better idea of how much word sense information improves performance on these tasks. It would also be interesting to see if this task can be integrated into an end-to-end model and see what kind of performance this model achieves.

In order to improve my existing model, I can try semi-supervised techniques to bootstrap a large training dataset. This would involve using a small labeled seed dataset to train a language model that could then be used to label unlabeled corpora, providing billion-token-scale datasets that can lead to much greater accuracy (such techniques were explored in Yuan et al. 2016). Also, I could also explore different metrics besides cosine similarity that may provide us with a more accurate representation of my approach's effectiveness and generalizability.

8 References

Evans, Colin, and Dayu Yuan. ”A Large Corpus for Supervised Word-Sense Disambiguation.” Google Research Blog. January 18, 2017. Accessed March 22, 2017.

Hochreiter, Sepp and Jurgen Schmidhuber. Long short-term memory. Neural Computation, vol. 9, no. 8, pp. 17351780, Nov. 1997.

Kageback, Mikael, and Hans Salomonsson. ”Word Sense Disambiguation using a Bidirectional LSTM.” 2016. ArXiv e-prints.

Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffery Dean. ”Distributed Representations of Words and Phrases and their Compositionality.” Advances in Neural Information Processing Systems 26 (2013): 3111-119.

Navigli, Roberto. ”Word sense disambiguation.” ACM Computing Surveys 41, no. 2 (2009): 1-69. doi:10.1145/1459352.1459355.

Yuan, Dayu, Julian Richardson, Ryan Doherty, Colin Evans, and Eric Altendorf. ”Semi-supervised Word Sense Disambiguation with Neural Models.” 2016. ArXiv e-prints.

Loic Vial, Benjamin Lecouteux, and Didier Schwab. “Sense Embeddings in Knowledge-Based Word Sense Disambiguation.” 2017. ArXiv e-prints.