Word Sense Disambiguation Using Skip-Gram and LSTM Models

Shrey Gupta
Department of Computer Science
Stanford University
shreyg19@stanford.edu

Armin Namavari
Department of Computer Science
Stanford University
arminn@stanford.edu

Tyler Otha Smith
Department of Computer Science
Stanford University
tosmith@stanford.edu

Abstract

In this paper, we explore supervised word sense disambiguation (WSD) through two main techniques. Central to both techniques is a learned word vector embedding that has separate word vectors for each sense of a word. When predicting on an unlabeled word, we can consider a window of word that contains it, and iterate over all possible combinations of word senses within this window. Out of all these possibilities, we choose the sense labeled window with the highest probability of occurring. Another way to compute senses would be to do sense-label initial window (either through exhaustive search or some other greedy method like assigning most likely sense) and compute the following senses with an RNN. The RNN will output a probability distribution over the next possible words. We choose the sense of the next word with the maximum probability when predicting with the RNN. The results of our models are promising and should improve as more hand-labeled sense-tagged data becomes available.

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 particular task. Word Sense Disambiguation (WSD) is considered an AI-Complete problem[1], meaning it is one of the centrally difficult problems in AI, so potential solutions and effective approaches towards this problem can have far-reaching implications for other key AI tasks. Consider the task of named

[1] Navigli 2009

entity recognition (NER), for instance. In NER, we attempt to classify different entities as organizations, people, places, etc. In the first case, we 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, we 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.

Regarding prediction of word sense, we can find combinations of word vectors that maximize the probability of their mutual occurrence (further detailed in section 3.1). We can also use a language model, implemented as an LSTM, to predict word senses (further detailed in section 3.2).

2 Background/Related Work

Mikolov et al. describes a word vector embedding that is quite commonly used in NLP research. The paper presents the Skip-gram word2vec model whose objective is " to learn word vector representations that are good at predicting the nearby words"[2]. In doing so, the model trains vectors that seek to approximate a function that computes the probability of seeing an outside context word vector given a center word vector:

$$p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w=1}^{W} \exp(u_w^T v_c)}$$

where W is the number of words in our vocab, u denotes outer words, and v denotes center words. Overall, $p(o|c)$ gives us a metric that can help us determine the probability of a window of words occurring together given their skip-gram word vector representations.

Google has released a WSD dataset[3] , 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 Salomonsson 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,"[4] similar to how we will approach sense classification with our LSTM. Their end-to-end model achieves "statistically equivalent results to the best state-of-the-art systems"[5].

3 Dataset

For this project, we use the publicly available Google Word Sense Disambiguation Corpora[6] to train our sense-tagged word vectors as well as train and evaluate our 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

[2]Mikolov et al. 2013

[3]Evans and Yuan, Google Research Blog

[4]Kageback and Salomonsson 2016

[5]Kageback and Salomonsson 2016

[6]Corpora can be found at `github.com/dmorr-google/word_sense_disambigation_corpora`

labelings are highly reproducible (usually a score above 0.67 is considered acceptable)[7]. 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 our Window Probability Maximization model, our sense-tagged word vectors were trained with all of the MASC dataset and $\frac{3}{4}$ of the SemCor dataset. The remaining section of the SemCor dataset was used for evaluation. For our experiments with LSTM models, we train the LSTM with at least the entire MASC dataset, with some experiments having the LSTM be trained with as much as $\frac{3}{4}$ of the SemCor dataset.

4 Approach

4.1 Word2Vec with Skip-gram

Traditional pre-trained word embeddings such as GloVe only have one vector per word. This is a problem for polysemous words; the multiple meanings are averaged together into a single embedding. The true meaning of a word is not properly conveyed with these types of embeddings, for the word may take on a different definition depending on the context. To combat this, we train sense-tagged word vectors in our approach. That is, there are multiple word vectors per word - one for each sense. This allows us to accurately convey the various meanings of a word depending on the context in which the word appears.

Our word vectors were trained using the Skip-gram model outlined in Mikolov et al. The training objective of this model is to train a network that predicts the likelihood of context words occurring given a center word. We feed in a one-hot vector that corresponds to a particular word in our vocabulary. This vector is passed through a hidden network layer and activation softmax layer to create a probability distribution of words appearing in the original word's context. We wish to minimize the following loss equation:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t)$$

The rows of the learned weight matrix in the hidden layer correspond to the word embeddings for each word in the vocabulary.

Our Skip-gram model also uses negative sampling to only modify a small amount of parameters during backpropagation.

A diagram illustrating the Skip-gram model can be seen in Figure 1.

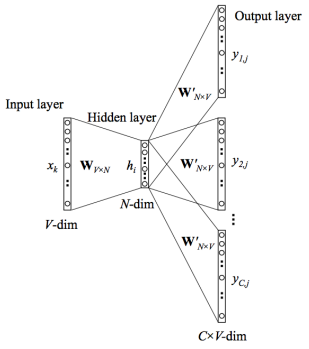

Figure 1: Diagram of Skip-gram technique for sense-tagged word vector training. (Image credit: User *Moucrowap* on Wikipedia)

[7]Evans and Yuan, Google Research Blog

4.2 Window Probability Maximization (WPM)

With the WPM method, we attempt to predict the sense of a center word by iterating through all possible senses of words within a particular window. More formally, consider our window of sense unlabeled words as a tuple: $(w_{c-n}, \ldots, w_c, \ldots, w_{c+n})$. Suppose we have a function S that maps unlabeled words to a set of their possible senses. We iterate over the Cartesian product

$$P = S(w_{c-n}) \times S(w_{c-n+1}) \times \ldots \times S(w_c) \times \ldots \times S(w_{c+n})$$

to generate a set of every possible sense labeled window as tuples. To evaluate a possible sense labeled window $(w'_{c-n}, \ldots, w'_c, \ldots, w'_{c+n})$, we compute the sum of the log probabilities of the outside words occurring with the center word:

$$E(w'_{c-n}, \ldots, w'_c, \ldots, w'_{c+n}) = \sum_{j \in c-n \ldots c+n, j \neq c} \log(p(w_j|w_c))$$

Our $p(w_j|w_c)$ is given by $p(o|c)$ as described in section 2. Once we have evaluated the probability of each possible sense labeled window occurring, we choose the one that has the greatest probability and use the label it generates for the center word. That is, for a window $(w_{c-n}, \ldots, w_c, \ldots, w_{c+n})$, we predict the sense of the center word as w^*_c where

$$(w^*_{c-n}, \ldots, w^*_c, \ldots, w^*_{c+n}) = \underset{(w_{c-n}, \ldots, w_c, \ldots, w_{c+n})}{\arg\max} E(w_{c-n}, \ldots, w_c, \ldots, w_{c+n})$$

We choose to only predict the probability of the center word so as to allow appropriate context for each word when its sense is being predicted.

The diagram below in Figure 2 shows a concrete example of how this algorithm may work. The example has three polysemous, ambiguous words. By considering the probability of each possible interpretation of the words in the context window (as determined using the approach described above), we can choose the combination yielding the maximal probability.

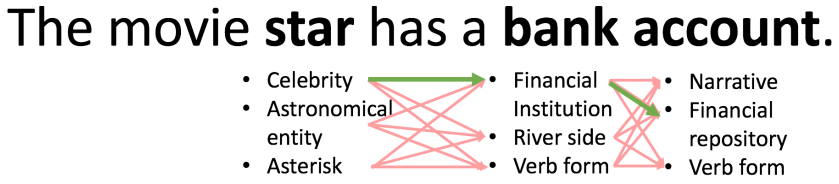

Figure 2: Example of WPM algorithm on a short sentence. If we consider a context window centered around 'bank', we see after searching all possible combinations (in light red), the algorithm ultimately chooses the interpretation marked with green arrows.

4.3 LSTM Model

With our LSTM model, we approximate a language model, i.e. a probability distribution that can predict the probability of a particular word occurring next given a history: $P(x_{t+1}|x_t, x_{t-1}, \ldots, x_{t-n})$. Suppose we are given a sense labeled history of words $(x'_t, x'_{t-1}, \ldots, x'_{t-n})$ and we wish to predict the sense of x_{t+1}. By feeding $(x'_t, x'_{t-1}, \ldots, x'_{t-n})$ into our RNN, we get a probability distribution over the next possible words. From this probability distribution we select all of the senses of x_{t+1}: $x'_{t+1_1}, x'_{t+1_2}, \ldots x'_{t+1_k}$ and we choose the one with maximum probability as our predicted sense x^*_{t+1}. That is, we select

$$x^*_{t+1} = \underset{x'_{t+1_j}}{\arg\max} P(x'_{t+1_j}|x'_t, \ldots, x'_{t-n})$$

The LSTM we use consists of an embedding layer, an LSTM unit, and a final dense layer with a softmax activation. The LSTM update equations[8] are as follows:

$$
\begin{aligned}
i_t &= \sigma(W^{(t)}x_t + U^{(i)}h_{t-1}) && \text{(Input Gate)} \\
f_t &= \sigma(W^{(f)}x_t + U^{(f)}h_{t-1}) && \text{(Forget Gate)} \\
o_t &= \sigma(W^{(o)}x_t + U^{(o)}h_{t-1}) && \text{(Output/Exposure Gate)} \\
\tilde{c}_t &= \tanh W^{(c)}x_t + U^{(c)}h_{t-1} && \text{(New memory cell)} \\
c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t && \text{(Final memory cell)} \\
h_t &= o_t \circ \tanh(c_t)
\end{aligned}
$$

For the architecture of the neural network, we input a vector containing a list of history words (represented as unique numerical IDs corresponding to their index in an embeddings matrix). We use the history word indices to index into an Embedding Layer, and push the embedding vectors through an LSTM Cell Layer with a hidden dimension of 256. We then map the outputs through a Dense Layer to regain a dimensionality compatible with the size of the vocabulary, and use a Softmax Activation Layer to output a probability distribution over the words in the vocabulary. Our final hyperparameters were:

- Sequence length (history): 25
- Num of training epochs: 50
- Learning Rate: 0.002
- Decay Rate: 0.97
- Gradient Clip: 5.0
- Hidden dimensionality: 256

The architecture is pictured below in Figure 3.

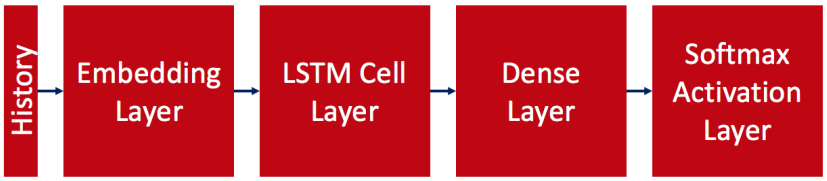

Figure 3: Architecture of the LSTM model.

5 Experiments

5.1 Evaluation Metric

The testing set for evaluation our approaches consists of 30k tokens from the SemCor dataset. We use "weighted accuracy" to evaluate our models. At the beginning of our evaluation, we initialize the total running score and our model's score to 0. For a given polysemous word, we add the total number of senses for that word to our running score. We add the same amount to our model's score if and only if our predicted sense is equivalent to the actual sense. The scores are only impacted by polysemous words; the senses of monosemous words are not considered. This metric ultimately weights words with more senses higher, since there are more possibilities and the correct label is presumably harder to predict as a result. In this section, we use "accuracy" to refer to refer to this notion of a weighted accuracy.

[8] Mohammadi et al.

5.2 Sense-tagged Word Vector Results

Through our Skip-gram approach discussed above, we trained sense-tagged word vectors. We initially tried to train with 128 dimensional vectors for 40k vocabulary words, but realized that given our dataset size, this would yield highly inaccurate vectors. By scaling down to 64 dimensions for 20k words, we were able to still sufficiently capture key information and have the vectors learn information that is consistent with basic human linguistic intuition, as shown in Figure 4.

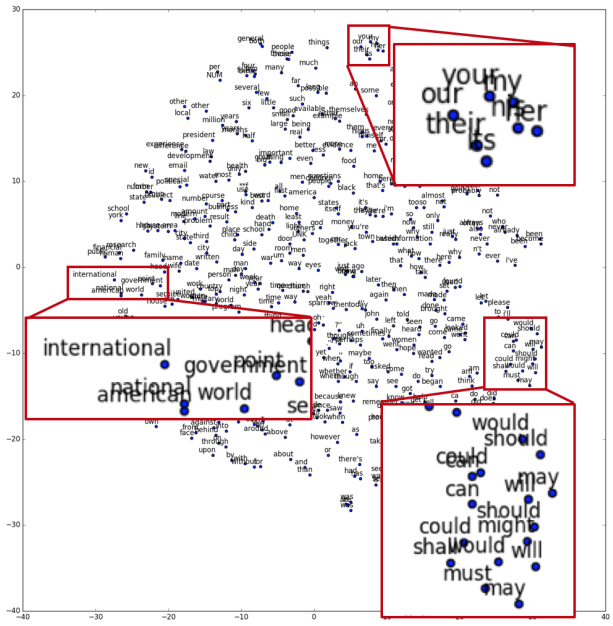

Figure 4: T-distributed Stochastic Neighbor Embedding (TSNE) diagram of sense-tagged word vectors. As shown, modal verbs, possessive pronouns, and words about sociopolitical entities are clustered together, suggesting the word vectors are learning the appropriate meanings. (The vectors shown are for the 64 dimension, 20k vocabulary variant.)

5.3 WPM Experiments

Our window-based evaluation model saw modest results. We decided on a 5-word window: the center word plus two words on each side. We thought this would give the center word enough context to accurately predict the sense while also maintaining a manageable testing time. It took about 40 minutes per file in SemCor to evaluate the 5-word window model. The long testing time is due to the calculation of all the Cartesian products.

The baseline for our models is Most Likely Sense (MLS), a common baseline in WSD. MLS naively tags a word with the most frequently seen sense during training. As seen in the table below, our 5-word window model outperformed MLS by roughly 8%. We attribute this to the implicit advantage with the window model. MLS simply tags the most frequently seen sense, while some rarer word senses rely on context to derive the meaning - one of the core components of WPM.

5.4 LSTM Experiments

We trained and evaluated three different LSTM models. The first two were built with Keras using our pre-trained sense tagged word vectors and were trained on the MASC dataset. Our first model was only trained on 10% of MASC for two epochs. We created this model to observe the effectiveness of LSTMs on small datasets and ensure that our model is trainable prior to training a fully-fledged model. The second model had the same architecture but was trained on all of the MASC data for ten epochs. Consequently, we see an increase in accuracy rate. From this, we see

Model	Accuracy
Most Likely Sense (Baseline)	34.77%
WPM: 5-word window with pre-trained embeddings	42.12%

Table 1: Results for WPM

that our model operates better when fed more data. Still, the model did not learn as much as we had hoped and faced various learning problems, potentially due to vanishing/exploding gradients or other issues. Therefore, we decided to create a final model using Tensorflow.

The model built with Tensorflow trained on all of the MASC dataset and roughly $\frac{3}{4}$ 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

Comparing the WPM and LSTM approaches, we see that the LSTM approach is generally more accurate and faster than the WPM approach. The increased accuracy is likely because LSTM can take into account a much larger history, and the increased speed is because the LSTM does not need to search an exponentially growing space, unlike the WPM approach, which must explore an entire Cartesian product. Regardless, both do end up scoring appreciably higher than the baseline, MLS, indicating that there is likely merit to both approaches in solving WSD tasks.

Another interesting point is how well LSTM does with limited training data. We observe that even with 10% of the training data, LSTM is able to achieve results that are almost comparable with the baseline, showcasing the power of LSTM.

When developing our Tensorflow LSTM, we noted a significantly higher accuracy than in the other experiments. 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 tailored to the specific task at hand, whereas the pre-trained vectors used in the Keras LSTM were made to optimize a different objective. However, we feel there may be potential overfitting of the model to the data, given that we trained this model on the full vocabulary. We also feel that our specific "weighted score" evaluation metric may have potentially inflated the reported accuracy of the model, since many words with fewer senses usually have a distinct dominant sense, meaning an algorithm with a larger vocabulary would be able to guess these senses easier and enjoy a higher score. Nevertheless, we feel these results show potential for the use of LSTMs in tasks like WSD.

6.2 Key Challenges

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

One of the big challenges we encountered with our 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 our purposes. For example, when training word vectors, we attempted to train 64-dimensional vectors for the top 20000 words (this is both a modest amount of dimensions and size of vocabulary; many practical applications use larger values). This works out to roughly 1.28 million parameters to train, but 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 times we can see a particular word in a particular sense is necessarily more infrequent than with sense-unlabeled word vector training,. Furthermore, we wouldn't be able to necessarily see the different senses of each word represented equally in all of their respective contexts, meaning rarer senses have less information. We anticipate that with more labeled training data, we would have much higher quality word vectors and LSTM models that would lead to higher accuracy scores.

Part of our task presented an interesting problem: training on sense-labeled data while attempting to predict on sense-unlabeled data for the LSTM approach. To overcome this, we attempted to provide an initial history with sense-labeled data for our models to build off of, essentially eliminating the problem. We did this by predicting the most likely sense or by exhaustively searching through sense combinations (similar to the WPM model, but only for an initial window). The former was not as accurate, and the latter was rather inefficient however, but both provided decent workarounds to the issue.

7 Conclusion

Overall we find that an LSTM and word vector focused approach shows 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, we would be interested in exploring the application of our 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, we 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 our existing model, we 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 explore in Yuan et al. 2016). Also, our given LSTM model only looks at prior history (i.e. words appearing before a given word in the sentence), but we could make predictions more robust by including a full context window (i.e. also words appearing after a given word). This could be done by implementing approaches like a bidirectional LSTM, as seen in (Kageback & Salomonsson 2016). Finally, we could also explore different metrics besides weighted score that may provide us with a more accurate representation of our approach's effectiveness and generalizability.

8 Acknowledgements

We would like to thank our project mentor, Danqi Chen, for her continued advice throughout our experimentation process. We also thank Dayu Yuan from Google for very recently releasing an appreciable corpus for this task and for answer our questions regarding his paper. Thank you Microsoft

for providing Azure GPU credits to help us accelerate our research. We would also like to extend our thanks to the github user hunkim for their word-level rnn model code that we adapted for our task. Finally, we thank the CS224N professors and staff for teaching us about the concepts and intuitions used in this project.

9 Contributions

While all members of the team contributed to the various tasks at hand, below is a rough guideline of what each person focused on roughly. Shrey focused on word vector generation, LSTM evaluation code, and creating the project poster. Armin focused on identifying papers of interest, working on the Keras and Tensorflow LSTM training implementations, and LSTM evaluation code. Tyler focused on word vector generation, LSTM evaluation code, and brainstorming experiments and hyperparamater tuning.

10 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.

Mohammadi, Milad et al. "CS224n: Natural Language Processing with Deep Learning, Lecture Notes: Part V." CS224N: Natural Language Processing with Deep Learning. January 2017. Accessed March 22, 2017.

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

Trask, Andrew, Phil Michalak, and John Liu. "Sense2vec - a fast and accurate method for word sense disambiguation in neural word embeddings." 2015. ArXiv e-prints.

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