Word Sense Disambiguation using RNNs for Context Embedding

CS396A - Project Report
Department of Computer Science and Engineering, IIT Kanpur

Rushab Munot (14405), rushab@iitk.ac.in

Supervisor - Prof. Harish Karnick, hk@cse.iitk.ac.in

Abstract

A given word can have multiple senses depending on the context that they are being used in. The task of Word Sense Disambiguation tries to distinguish between these senses based on the word's context.

We address this classical problem of disambiguating between multiple senses of a word using recurrent neural networks for context embedding and training a classifier on these embeddings. We propose a word centered approach, wherein we train an LSTM network for every word of interest. Specifically, we bidirectional LSTMs, i.e. one bidirectional LSTM per word. We try to improvise further adding POS tags. The model, when tested on the four words data-set from Senseval-2, reports an accuracy of more than 90% for the word hard (3 senses) and a F1 score of more than 80% for serve (4 senses) and interest (4 senses) using just LSTM model.

Contents

1 Introduction

In most languages a many words have multiple meanings, and what a word means is dependent on where it appears. The meaning of an unknown word becomes clear once its context is known to us, even without the need of an external dictionary. Word Sense Disambiguation (WSD) tries to select a sense for a given word from its known senses (supervised learning), or it tries to generate senses for a given word given examples of sentences where it occurs (unsupervised learning) and when given a test sentence, it assigns to it the sense that matches most with this sentence.

WSD could be of high use to many Natural Language processing tasks including Summarization, Co-reference resolution, Machine Translation, etc. We address the problem of WSD using bidirectional LSTMs, like [Kageback and Salomonsson, 2015], by training an end-to-end model for every word that has to be disambiguated, focusing on an individual word rather than a global model shared between all words.

1.1 Description of the WSD Task

The WSD task is one of oldest problems in Computational Linguistics, which aims to predict what meaning of a particular word is being used, depending on its context. More formally, let the given

text be represented as a sequence of words $\langle w_1, w_2, w_3, \ldots, w_n \rangle$. Given a word w_i from this text, the task is to predict, from a known repository of senses for that word, the correct sense amongst these. Thus, we need to find a mapping f, which maps a sequence of words $s_n = \langle w_1, w_2, w_3, \ldots, w_n \rangle$ to the set of senses of w_i from a known repository R. More generally, f is mapping from s_n to R_{w_i}, which is the set of senses of w_i, such that $f(\langle w_1, \ldots, w_n \rangle) \in R_{w_i}$ [R. Navigli, 2009]. Typically, we are interested in exactly one sense per word. The WSD task can thus be viewed as a classical classification task, where each word sense is a class (i.e. the dictionary R is the set of all classes) and the input is a sequence of the form of s_n.

The task described describes the *'lexical sample'* variant of the WSD task. Superwised approaches are observed to work better on such tasks. In contrast to this variant is the *'All Words WSD'* which aims to classify all words in the text with multiple textual meanings (in R) to a single meaning for each word [R. Navigli, 2009]. As an example, consider the following sentences -

1. **One** needs to follow the eight-fold path to lead a satisfactory life
2. There is only **one** candy left.
3. A binary sequence is a sequence of **one**s and zeros.

In the first sentence, *one* means the self, relating to an individual. In the second sentence *one* represents a quantity, that a single candy is left while in the third sentence *one* denotes the digit '1'. However, the distinction between senses is not so direct. Consider the following examples of the usage of the word *'interest'* -

1. The minister said that the decision was made in the **interest** of the nation.
2. She said that it was not in her **interest** to teach the boys

There is a subtle difference between the senses of *interest* in the above example, while the first sentence pertains to public interest or national interest, the second sentence implies the girl's self-interest. A WSD task may need to classify such instances depending on the application.

The word to be disambiguated shall be denoted as 'query word' henceforth.

2 Background and Related Work

2.1 Context Representation

To make any meaningful predictions about the senses of a word, the context that the word appears must be carefully taken into account. The context is generally given a numerical representation, but this is not the case with all algorithms. Some of the classical approaches towards context representation are as follows[R. Navigli, 2009]

- Use a local window around the word that is to be sense disambiguated. In Example 2.2, a window of size 4 (2 words on wither side) around the word *interest*, would be {*in, her, to, teach*}, a window of size 8 (4 words on either side) would be {*was, not, in ,her, to, teach, the, boys*}.

- A simple, yet not ineffective, approach is to represent the context as a bag of words of local window around the word of interest. These words are typically represented as word-vectors (word2vec[Mikolov et al., 2013], glove[Socher et al., 2014], or co-occurrence vectors).

- To improve upon the previous approach, sequential information can be added to the bag-of-words context, by assigning weights to each word based on its position. Thus, we use {*(in,f(-2)), (her,f(-1)), (to,f(+1)), (interest,f(+1))*}. f is a function that maps position to weight. The weights must become larger as we move closer to the query word and taper out as we move away from the query word.

- Information like Part-of-Speech (POS) tags can be added to the word in context.

- Using chunking - Divide the sentence into smaller parts which are syntactically correct and can stand by themselves, examples include dividing the sentence into phrases (noun, verb, adjective, adverb, ...).

- Parse the sentence to generate a parse tree, process this parse tree and use information from it as the context. The parse tree stores important syntactic information which may help in WSD.

2.2 Classical Takes on WSD[0]

2.2.1 Supervised Approaches

- Numerically represent the context and use a classifier (Naive Bayes, SVM, kNN, Decision Tree, Feed Forward NNs and Ensemble Methods)

- SVMs have better accuracy than others.

- Semi-supervised approaches include those using BootStrapping

2.2.2 Unsupervised Approaches

- Context-clustering: Represent the cluster as a context vector from the co-occurence matrix, reduce dimensionality, and clusterize these vectors. While testing assign the cluster with highest similarity.

- Another way is to use weighted co-occurence graphs

[0]Taken from Word Sense Disambiguation: A Survey [R. Navigli, 2009]

2.2.3 Knowledge-based methods

- These methods use a thesaurus/dictionary (e.g. WordNet) to add more information.
- **Lesk's Algorithm:** Lesk's algorithm is a classical standard algorithm. Lesk's algorithm measures similarity as the overlap between the *gloss* of the context and word. $gloss(w)$ represents (bag of) words in the definitions of w. $gloss(context(w))$ represents union of *glosses* of all words in context. Lesk's algorithm checks how much the textual definitions of words in context overlap with the definition of each sense for the word to be disambiguated. The sense with the highest gloss overlap with the context is predicted as the current sense.

$$Similarity_{Lesk} = |gloss(w) \cap gloss(context(w))|$$

- Many more approaches with several measures of similarity have been defined considering various factors but we will not discuss those here.

2.3 Different Approaches

2.3.1 Bilingual/Multilingual Corpora

It is highly improbable that multiple of two words each of from a different language can mean the same. This information can be exploited to disambiguate a word in the first language using the translation of its context into the second language. This approach has also been used to learn differents embeddings for different senses of a word [Simon Suster, Ivan Titov, Gertjan van Noord, 2016], [Yogarshi Vyas, Marine Carpuat, 2016].

2.3.2 Page Rank based Algorithms

Algorithms based one PageRank (Personalized PageRank) have also been applied to the WSD task [Eneko Agirre and Aitor Soroa, 2009].

2.4 LSTMs

An LSTM is a special type of a recurrent neural network that consists of three gates which regulate the change of information in the LSTM cell. These gates include the input, output, forget which respectively control what new information is added, what is outputted and what is forgotten from the LSTM cell. The cell state vector of the LSTM cell is what contains the information stored in the LSTM cell.

Let c_t denote the state vector of the LSTM cell, x_t denote the input vector and h_t denote the output vector at time t. Let W_i, W_o, W_f denote the weight matrices controlling the input output and forget gates respectively and $b_i, b_o and b_f$ be the biases. Let W_c and b_c be the biases for computing the change in cell state.

The operators $\odot$ and $\times$ denote element-wise multiplication and matrix multiplication (matrix with a vector in this case) respectively.

The output of the forget gate depends on the input and output of the previous state. A sigmoid layer is also added which decides how much of each element of the state vector is forgotten.

$$f_t = \sigma\Big(W_f \times \langle x_t, h_{t-1}\rangle + b_f\Big)$$

The output of the input gate, i_t, decides how much of the input gets stored in the cell state. Similarly, the output o_t of the output gate decides what part of the cell state is to go into the output vector.

$$i_t = \sigma\Big(W_i \times \langle x_t, h_{t-1}\rangle + b_i\Big)$$

$$o_t = \sigma\Big(W_o \times \langle x_t, h_{t-1}\rangle + b_o\Big)$$

Next the cell state c_t is updated

$$c_t = f_t \odot c_{t-1} + i_t \odot tanh\Big(W_c \times \langle x_t, h_{t-1}\rangle + b_c\Big)$$

f_t indicates what is to be forgotten (and how much) from the previous cell state c_{t-1}. The first term, thus denotes the cell state after forgetting what is indicated by f_t. i_t indicates the scale of change, while $tanh(W_c \times \langle x_t, h_{t-1}\rangle + b_c)$ denotes the change, thus the element-wise product indicates the effective change to the cell state. Also notice that the range of $tanh$ is $(-1, 1)$ while that of *sigmoid* is $(0, 1)$.

Finally to compute the output, we take the element-wise product of o_t with the cell state (operated upon by $tanh$).

$$h_t = o_t \odot tanh(c_t)$$

3 Using LSTMs

3.1 LSTMs and sequential data

LSTMs are widely used to model sequential data, because of their ability of efficiently manage and remember history. Many deep learning models use LSTMs to manage sequential data e.g. seq2seq [Sutskever et al., 2014], etc.

The working of a typical sequencer encoder is shown in Fig 1. Suppose that somehow the parameters of the model are trained, and given to us. To now embed a sequence into the LSTM, we pass the first element x_1 through the LSTM, then we use the hidden state of the LSTM and the the next element x_2 to update the hidden state. Then using this hidden set and x_3 we again update the hidden state. We do this till we reach the last element of the sequence the output of which is

the required embedding. The outputs of the LSTM through x_1 to x_{n-1} are usually discarded. Only the output of the last element is considered.

To embed a local window context in such a way, the context is passed as a sequence (query word may or may not be included). The context embedding is thus the output from this LSTM model

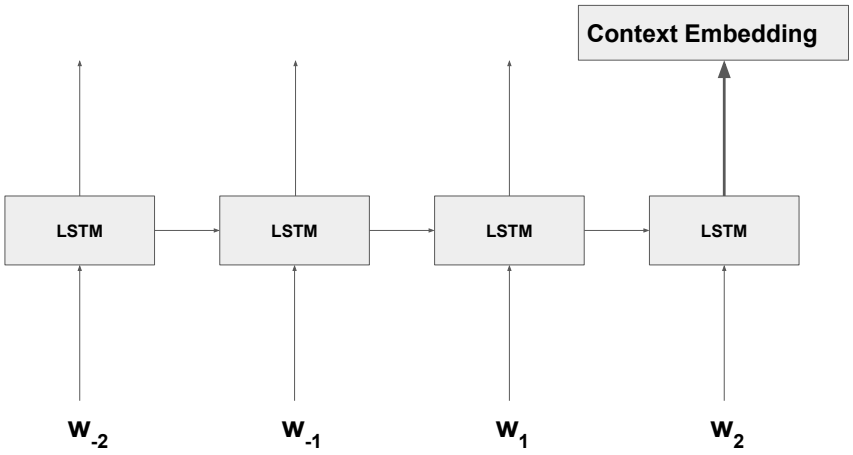

Figure 1: Encoding a sequence with an LSTM model
Based on *Sequence to Sequence Learning with Neural Networks*, [Sutskever et al., 2014]

3.2 Training such models

Models like the LSTM model described above, are generally trained using a supervised learning approach specific to task at hand. Thus, sequence to sequence learning [Sutskever et al., 2014] which follows an encoder-decoder approach, uses the output sequence to back-propagate the error to the decoder, and from the decoder to the encoder. The parameters of the LSTM model are fine-tuned to the specific task it is trained upon.

In the case of WSD as a classifier model, the classifier and the LSTM encoder are trained together. Thus, the parameters of the model are fine-tuned to be combined with the classifier. This is shown in Figure 2.

When an input context sequence is passed during training, it is forwarded through the model and a probability distribution on the senses is predicted. This is then compared with the expected label. The error is then back-propagated and the parameters are adjusted accordingly.

3.3 Bidirectional LSTMs

It is observed that instead of using LSTMs to scan the context only in the forward direction, scanning the context in the backward direction can also help. Some dependencies seem to get resolved

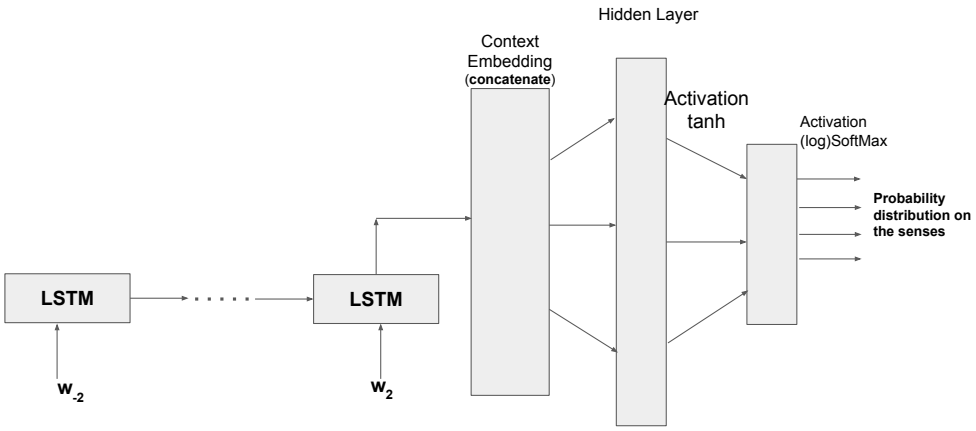

Figure 2: A unidirectional LSTM Model for classification - sequence(context) to label

by using Bidirectional LSTMs (BLSTMs) and generally they provide a better result compared to unidirectional LSTMs. The model is shown in Figure 3. We base our results on bidirectional LSTMs.

The model is implemented as follows:

- Keep two separate LSTMs - one for scanning the context in either direction.
- Concatenate the output from the two LSTMs and pass this to the classifier.
- Compare with the expected output and back-propagate the error through both the LSTMs

4 The Model

4.1 The 'One-LSTM-per-word' Approach

We propose a one-LSTM per word approach. For every word that has to be disambiguated, train an LSTM model for classification. Thus, if there are W words that can be possible query words, we train W classifier models. Thus, the same sentence (context) can have multiple possible embeddings when disambiguating different words. The model is depicted in Figure 4.

4.2 Modifications

- Add POS tags to the words. POS Tags can be passed in either of the following ways:

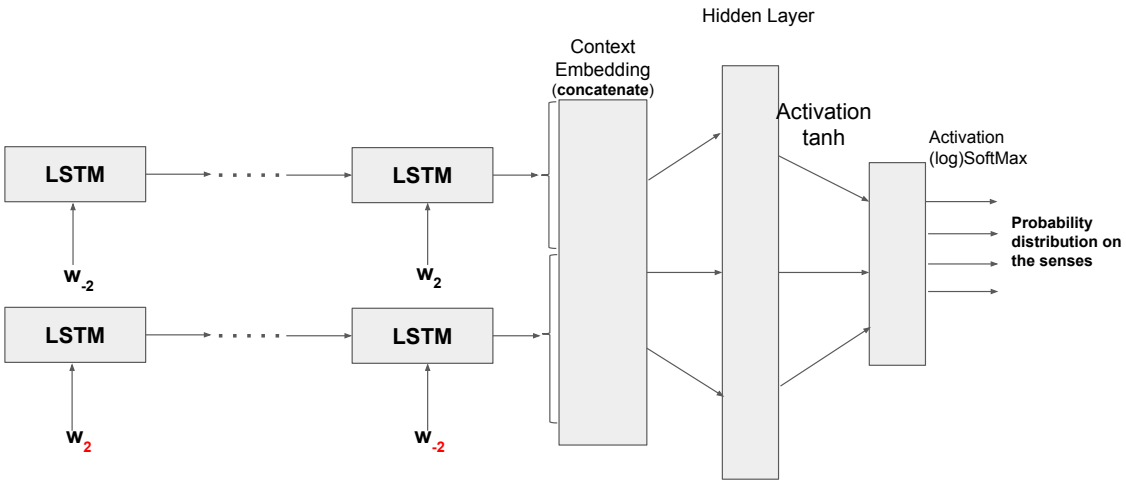

Figure 3: A Bidirectional LSTM Model for classification

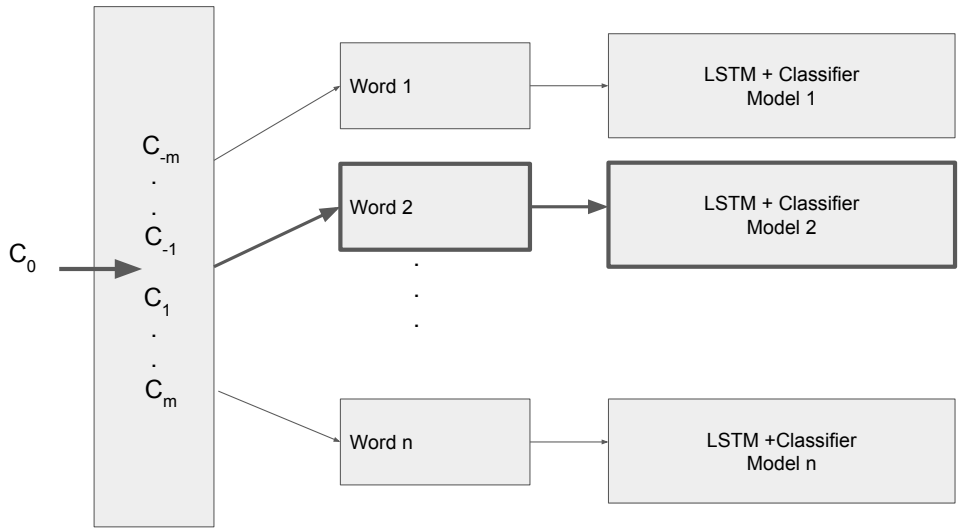

Figure 4: **The complete picture for one LSTM per word**
The path to be followed if $C_0 = word_2$: As C_0 matches with $word_2$ it uses the model for $word_2$ and predicts the relevant sense.

1. Convert words to <word, pos> tuples, and train as per the model above.
2. Have a separate LSTM per model that sequentially scans POS tags (uni-directional) and outputs a small sized (kept 5) hidden vector. This is concatenated with the context embedding and this is followed by classification as usual.

Both methods perform equally well in terms of maximum accuracy as is experimentally verified, however the maxima for the second method is wider (persists through a larger number of training epochs). We use the first method to report accuracies (as the accuracy is same for both, and the first method is easier to train and preprocess)

- Change the classifier (try to use SVM based classifier or one-vs-rest Logistic Regression)
- In the current model senses are assumed to be independent of each other, but this is not the case. In the example of public-interest vs self-interest, there is a certain overlap between the two senses. These senses are more related to each other than, say, the sense of interest meaning *interest rates*. We discuss this point later.

5 Dataset

- We use the Senseval-2 Lexical Sample Task dataset [Senseval 2, 2001] which consists of 4 words - hard, serve, line and interest having 3,4,6,6 senses respectively. The data distribution across senses is shown in Table 1

Word	#Senses	Total # of examples	Distribution across senses
hard	3	4333	(3455, 502, 376)
serve	4	4378	(1814, 1272, 853, 439)
interest	6	2368	(1252, 500, 361, 178, 66, 11)
line	6	4146	(2217, 429, 404, 374, 373, 349)

Table 1: **Senseval-2 Four-Words Dataset [Senseval 2, 2001]**

- Certain words from the One-million word corpus - interest, position, serve. The data distribution across senses is shown in Table 2 This corpus is a semi-automatically tagged corpus with an accuracy of 83.7% [Taghipour and Ng, 2015](as reported by the authors on 1000 randomly selected examples.)

Word	#Senses	Total # of examples	Distribution across senses
interest	4	2111	≈500 each
position	5	2571	≈500 each
serve	8	3499	≈500 for 6 senses, ≈300, ≈150

Table 2: **One million word corpus** [Taghipour and Ng, 2015]

6 Experiments

6.1 Preprocessing

- We train an end-to-end model, where one-hot representations of the words in context are passed sequentially. Word embeddings are trained along with the model.
- The context window size is set to 5 words on either side of the query word. Experiments showed that a window size of 5 gives optimal results for the above datasets.
- Punctuation marks and stop-words are not removed. Experiments indicate that their removal does not increase the accuracy, rather in some cases slightly decreases it. Removing punctuation marks/stopwords and then reducing window size to say 4 or 3 but that does not help.
- Unknown words in the test sentences are removed from the context, however the context size is not re-adjusted after that. Depending on the dataset the number of unknown words per context varies (≈5-15%). Removing unknown words experimentally show an improvement in accuracy.
- The senses with 11 and 66 instances for the word *interest* in the Senseval-2 dataset are ignored, as predictions on these cannot be conclusive.

6.2 Training Details

We use a batch-size of 10 or 20 for most words. The context-embedding has a size of 300 per LSTM (i.e. 600 for a bidirectional LSTM). We use a learning rate of 0.01 with a decay factor of 0.0001.

We perform a 10-fold cross validation (5% training data used for validation) and use the first k rounds to determine the average. (k=5 for line and interest, k=10 for hard and serve; for the million word corpus, k=7 for interest, 5 for position and 3 for serve). This is a bad strategy, however due to lack of time, this approach had to be undertaken.

7 Results

The results are shown in Tables 3 and 4

Word	**#Senses**	**BLSTM Embedding**		**BLSTM + POS embedding**	
		Accuracy	**F1**	**Accuracy**	**F1**
hard	3	90.90	79.28	91.04	79.36
interest	4	87.67	80.42	87.24	80.65
serve	4	84.79	81.76	84.14	81.09
line	6	79.05	70.65	78.64	68.53

Table 3: Accuracy and F1(macro) score on the Senseval-2 dataset

Word	**#Senses**	**BLSTM Embedding**		**BLSTM + POS embedding**	
		Accuracy	**F1**	**Accuracy**	**F1**
interest	4	76.61	76.96	76.40	72.20
position	5	61.39	60.50	62.86	61.76
serve 0	8	53.29	54.32	54.64	50.00

Table 4: Accuracy and F1(macro) score on the million-word dataset

7.1 Some observations

- The first dataset is highly skewed, but has more examples per word. On the other hand the second dataset has unbiased data but lesser number of examples.
- Accuracy is not a good evaluation metric for the Senseval-2 dataset. F1 score is a much better evaluation metric.
- Thus, hard which has an accuracy of more than 90% has an F1 score of less than 80% compared with serve which has an accuracy of about 84% but a higher F1 score of about 81%.

- Further, POS tags do not play a significant role in the Senseval-2 dataset.
- For the second dataset, however F1 score and accuracies are almost the same. This is justified by the fact that this dataset has unbiased data.

8 POS Tags

There is a need to report the effect of POS tags separately as the comparisons with and without POS tags must be performed on the same data i.e. same training and test data. This is because the data is highly skewed. Note that the cross-validation accuracies reported in the above table were calculated independently. The effect of POS tags is more stark in the million-word corpus due to insufficient data. Adding more data in the form of POS tags may prove beneficial. In most (not all) of the cases, adding POS tags increases the accuracy by 1-2%. Following are plots where this improvement is considerable(Figure 5 and Figure 6) .

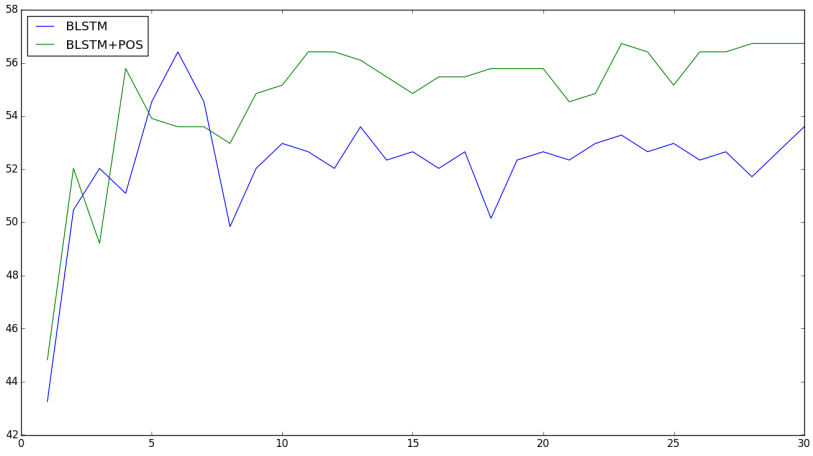

Figure 5: BLSTM vs BLSTM+POS for *serve*: Accuracy vs #Epochs

9 Problems

- Using a SVM based classifier or a one-vs-rest Logistic Regression gives slightly better results (1-2%), but the C parameter (in sklearn) needs to be fine-tuned. This value turns out to be data dependent, and changes with different words as also (in some cases) with different runs for the same word.

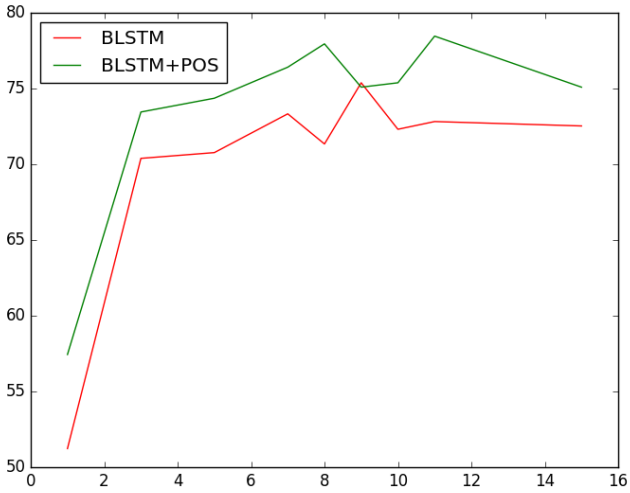

Figure 6: BLSTM vs BLSTM+POS for *interest*: Accuracy vs #Epochs

- Insufficient data - difference between accuracy with 60% training data and 90% training data is significant
- Skewed data - The actual instances that go into the training set and test set largely affect the accuracies (as much as 8-9% in certain cases)
- There is no guarantee that if a model works on a certain dataset, it will work in general.
- Accuracy is high on some words low on others. This has also been observed in the literature.
- For words with equal number of examples (one million words corpus) in each class, F1 $\approx$ Accuracy. However, more data is needed to correctly classify the senses as 500 examples of each sense are not enough (especially with the number as high as 7-8).

10 Future Work

- Try to construct a dataset, with sufficient examples, which can guarantee universally valid results. That is, a model when tested on this dataset can be said to work on most other datasets. Sadly, such a dataset is not available.
- Instead of a classifier use regression models more extensively. First replace all instances of the query word in the training examples by their corresponding sense. Train word vectors using this new data, and extract the senses of the word. Till now it was assumed that the

senses of a word are independent of each other, and a one-hot vector was used to predict the output. However, if vectors are assigned to the senses, we have a notion of similarity between the senses. Using the original training data (that is not sense tagged), try to predict the sense vector instead of the sense label.

- To extend the the notion of similar senses, build a structural hierarchy of senses, say a tree where the topmost node represents the given word and as we go down every node represents a cluster of senses. Two senses in the same cluster are more similar that those in different clusters. The classification goes in a top-down manner like a decision tree.

Acknowledgments

I would like to thank Prof. Harish Karnick for his experienced guidance in this project. I would also like to thank Lalchand Pandia, for providing insights into certain problems.

References

[Senseval 2, 2001] Philip Edmonds and Scott Cotton. 2001. *SENSEVAL-2: overview.* In The Proceedings of the Second International Workshop on Evaluating Word Sense Disambiguation Systems (SENSEVAL '01), Judita Preiss and David Yarowsky (Eds.). Association for Computational Linguistics, Stroudsburg, PA, USA, 1-5.

[Taghipour and Ng, 2015] Kaveh Taghipour and Hwee Tou Ng. 2015. *One Million Sense-Tagged Instances for Word Sense Disambiguation and Induction.* Proceedings of the 19th Conference on Computational Language Learning, pages 338–344, Beijing, China, July 30-31, 2015.

[Juergen Schmidhuber, 1997] Juergen Schmidhuber. 1997. *Long short-term memory.* Neural computation, Vol. 9, pages 1735-1780.

[R. Navigli, 2009] Roberto Navigli. 2009. *Word Sense Disambiguation: A Survey.* ACM Computing Surveys, Vol. 41, No. 2, Article 10.

[Sutskever et al., 2014] Ilya Sutskever, Oriol Vinyals, Quoc V. Le. 2014. *Sequence to Sequence Learning with Neural Networks.*

[Mikolov et al., 2013] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, Jeffrey Dean. 2013. *Distributed Representations of Words and Phrases and their Compositionality.*

[Socher et al., 2014] Jeffrey Pennington, Richard Socher, Christopher D. Manning. 2014. *GloVe: Global Vectors for Word Representation.*

[Kageback and Salomonsson, 2015] Mikael Kageback, Hans Salomonsson. 2016. *Word Sense Disambiguation using a Bidirectional LSTM.*

[Cuong Anh Le and Akira Shimazu, 2004] Cuong Anh Le and Akira Shimazu. 2004 *High WSD accuracy using Naive Bayesian classifier with rich features* ACL 2004

[Simon Suster, Ivan Titov, Gertjan van Noord, 2016] Simon Suster, Ivan Titov, Gertjan van Noord 2016. *Bilingual Learning of Multisense Embeddings with discrete autoencoders.* NAACL-16.

[Yogarshi Vyas, Marine Carpuat, 2016] Yogarshi Vyas, Marine Carpuat. *Sparse Bilingual Word Representations for Cross-lingual Lexical Entailment. 2016. NAACL 2016.*

[Eneko Agirre and Aitor Soroa, 2009] Eneko Agirre and Aitor Soroa. 2009. Personalizing PageRank for Word Sense Disambiguation.