Part-of-Speech Tagging with LSTMs

Zeyuan Hu
Computer Science Department
University of Texas at Austin
Austin, Texas
iamzeyuanhu@utexas.edu

Abstract

We build a Bidirectional Long Short Term Memory networks (BiLSTMs) and apply it to Part-Of-Speech (POS) tagging using data from the Penn Treebank (Marcus et al., 1994). We incoporate orthographic features (e.g., capitalization, suffix, prefix) into the model and improve the baseline model overall accuracy by 1% and OOV accuracy by 23.1% on the validation set, and 0.8% overall accuracy and 24.4% OOV accuracy on the test set.

1 Introduction

Parts-of-Speech (POS) tagging is a task that assigns a tag to each word in a given sentence. Tag may include: noun, verb, pronoun, preposition, adverb, and so on. POS tagging is an important task in NLP because knowing the tags of words can give us the information about likely neighbouring words and the syntactic structure of the sentence, which will be useful for Syntactic Parsing, Named Entity Recognition, and other information extraction tasks (Jurafsky and H.Martin, 2017, Chapter 10). However, since the amount of tags and sentences can be large, manually tagging each words in a sentence can be extremely time-consuming. Thus, we want to invoke some statistical model to automatically generate the tags for us. In this report, we employ the Bidirectional Long Short Term Memory networks (BiLSTMs) model.

Bidirectional Long Short Term Memory networks (BiLSTMs) model is one kind of Long Short Term Memory network (LSTM) models with architecture shown in Figure 1 (Socher, 2016). x represents a token (word) as a vector; y represents the output POS tag; and h represents the memory, which is computed from the past memory and current word (Olah, 2015). The motivation for us to use BiLSTMs model is that POS tagging is a classification problem (i.e., we classify each word into one of the tag category) and for classification on a given word, we want to incorporate information from both the preceding words (i.e., forward direction in BiLSTMs) and the words following it (i.e., backward direction in BiLSTMs).

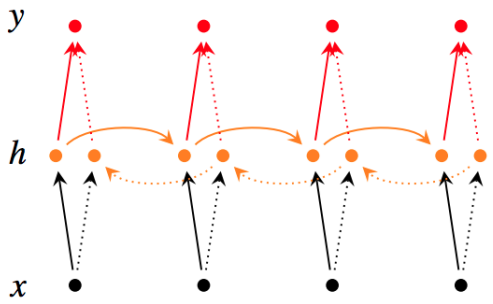

Figure 1: BiLSTMs Architecture

2 Orthographic Features

Baseline uses the BiLSTMs model with the `<word_ID, pos_tag>` pair. In other words, words are represented by its `word_ID` in the vocabulary and fed into the model. In this report, besides word ID, we use the following orthographic features from each word to improve the model performance:

- Capitalization
- Contains common English suffix (Honig et al.,

2000) [1]

- Contains common English prefix (Honig et al., 2000)
- Contains hyphen
- Starts with a number

The intuition for using these orthographic features is that orthographic features may be good indicators for building the connection between words and POS tags. For example, capitalization may be a strong indicator for `PRP` (e.g., *I*) or `NNP` (e.g., *China*). Similarly, prefix (e.g., *anti-*) and suffix (e.g., *-ly*) are good indicators for `NN` (e.g., *ant-government*) and `RB` (e.g., *lovely*) respectively.

3 Implementation

Orthographic features are represented by its ID in the corresponding "feature vocabulary" like we do with word ID feature. For example, capitalization is a binary feature: whether a word contains capitalization: 1 means "yes" and 0 means "no". Thus, for capitalization feature, the "feature vocabulary" has size of 2. We construct such "feature vocabulary" for each feature during the preprocessing step.

We use two ways to incoporate the orthographic features into BiLSTMs model: at the input and at the output. Figure 2 shows an implementation view of BiLSTMs when we incoporate the orthographic features at the input. At the input, orthographic features are concatenated with the original word ID feature. Each column (one is colored in green) in Figure 2 is a vector representation of a word ("was" in this case). The vector is constructed by concatenating the vector representation of each feature. The dimension for word ID feature is `embedding dimension` $\times$ 1. The rest features (i.e., all orthographic features) will have their dimensions determined the way we construct them. We use three ways to construct the orthographic feature vectors: *embedding*, *one_hot*, and *int*. *embedding* constructs the orthographic features in exactly the same way as the word ID feature: using embedding. The only difference is that we embedd the orthographic features into dimension `orthographic_embedding_dim` $\times$ 1 with

[1] We use 31 prefixes and 23 suffixes.

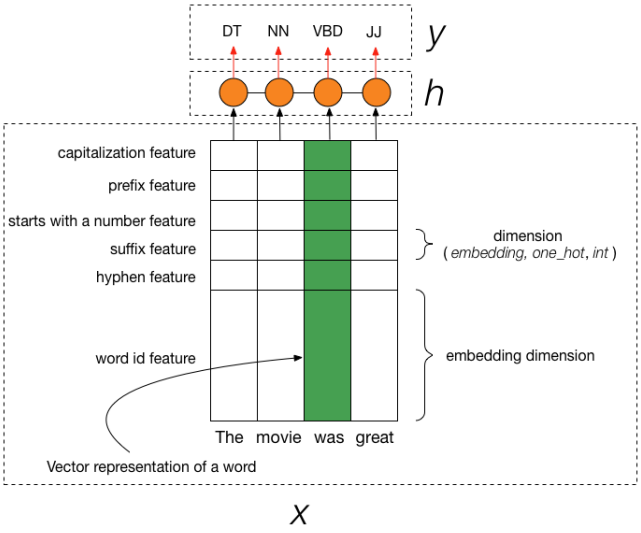

Figure 2: BiLSTMs Implementation when Adding Orthographic Features at the Input

`orthographic_embedding_dim` set to 10 by default. *one_hot* option is similar to *embedding* but we use the size of each "feature vocabulary" to determine the one-hot vector dimension. The advantage of doing so over the *embedding* is that each orthographic feature's one-hot vector may have different length. Since the size of "feature vocabulary" is very small (i.e., the largest size is 31), we do not need to worry about the computation inefficency brought by one-hot vector. The last way is *int*, which directly feed the feature ID as part of word vector representation into BiLSTMs.

When we concatenate the orthographic features into the BiLSTMs at the output, we use the same ways to construct our orthographic feature vector representations. The only difference is that we concatenate our orthographic features with the outputs from BiLSTMs. The outputs of BiLSTMs is connected with a fully connected layer so that we can know that which POS tag has the highest probability for the corresponding word. The fully connected layer is needed because we need to map the output dimension from BiLSTMs into the dimension that is well-aligned with the number of POS tags. By concatenating orthographic features at the output, we increase the dimension of the output vector that is to be fed into the fully connected layer.

Description	Values
batch size	128
number of epochs	6
embedding size for word ID	300
embedding size for orthographic features	10
optimizer learning rate	0.0005
optimizer	Adam

Table 1: BiLSTMs Configuration

Validation	BASELINE		
	Accuracy	OOV Accuracy	Training Time
embedding	95.6%	57.1 %	860s
one_hot	95.6%	57.1 %	860s
int	95.6%	57.1 %	860s
	INPUT		
	Accuracy	OOV Accuracy	Training Time
embedding	96.5%	78.3%	927s
one_hot	**96.6%**	**80.2%**	911s
int	96.1%	66.7%	**859s**
	OUTPUT		
	Accuracy	OOV Accuracy	Training Time
embedding	96.1%	70.5%	878s
one_hot	95.6%	57.2%	885s
int	95.5%	56.6%	878s

Table 2: Model Performance on Validation Set

Test	BASELINE	
	Accuracy	OOV Accuracy
embedding	95.8%	55.4%
one_hot	95.8%	55.4%
int	95.8%	55.4%
	INPUT	
	Accuracy	OOV Accuracy
embedding	**96.6%**	**78.8%**
one_hot	**96.6%**	77.7%
int	96.1%	66.3%
	OUTPUT	
	Accuracy	OOV Accuracy
embedding	96.2%	70.4%
one_hot	95.6%	57.2%
int	95.7%	54.2%

Table 3: Model Performance on Test Set

4 Experiments and Analysis

We perform six experiments: two place to concate orthographic features in combination with three ways to construct feature vector representations. The hyperparameters for the experiments are shown in Table 1. All the experiements are performed on a machine that has 4 Intel(R) Core(TM) i5-6200U CPU @ 2.30GHz processors and 4GB of memory. Experiment results are shown in Table 2 and Table 3.

From the results, we can see that adding the orthographic features increases the BiLSTMs performance on both the overall accuracy and the OOV accuracy on both validation set and test set. For the validation set, adding the orthographic features as one-hot vectors at the input improves the baseline performance by 1%. At the same time, there is a large jump on OOV accuracy, which raises from 57.1% to 80.2% (i.e., 23.1% increase). The trade-off for more accurate model is the longer training time. To train orthographic-enhanced BiLSTMs with *one_hot* takes 51 more seconds. Surprisingly, if we directly add feature ID as the input, the running time is actually similar to baseline model. The reason behind this is that adding feature ID as the input does not increase computational complexity (i.e., does not involve significant matrix multiplication and transformation).

On the test set, the story is similar with *embedding* and *one_hot* at the input leading the chart. From these two tables, we can see that adding the orthographic features at the input outperforms adding the orthographic features at output (i.e., classification layer). One possible reason is that adding the orthographic features at the input provides BiLSTMs a chance to learn the weights of features according

to the ground truth. However, if we directly add the orthographic features at the classification layer, there is no chance for the model to fine tune the weights. In addition, adding the orthographic features at the input enriches the information encoded in the vector representation of a word. By incorporating more characteristic information about a word in the vector representation, we can better learn the connection between word and POS tag. Adding orthographic features at the output, however, is not all bad. From the experiment results we can see that adding orthographic feature at the classification layer generally requires less training time compared with at the input. This is because computation at the memory units of BiLSTMs is much more complex than the computation at the fully connected output layer. Thus, sending larger vector to the memory units (i.e., adding orthographic features at the input) takes more time to compute than sneding larger vector to the classification layer (i.e., adding orthographic features at the output).

5 Conclusion and Future Work

In this report, we use orthographic features to improve the BiLSTMs model overall accuracy by 1% and OOV accuracy by 23.1% on the validation set, and 0.8% overall accuracy and 24.4% OOV accuracy on the test set. As the following, we show that orthographic features can indeed improve the model performance. In the future, we may want to experiment with different languages to see if our conclusion still holds.

References

Bill Honig, Linda Diamond, Linda Gutlohn, and Jacalyn Mahler. 2000. *Teaching Reading Sourcebook: For Kindergarten Through Eighth Grade*, volume 1. Academic Therapy Publications, 20 Leveroni Court, Novato, CA 94949-5746.

Dan Jurafsky and James H.Martin. 2017. Speech and language processing. preprint on webpage at `https://web.stanford.edu/~jurafsky/slp3/`.

Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. 1994. The penn treebank: Annotating predicate argument structure. In *Proceedings of the Workshop on Human Language Technology*, HLT '94, pages 114–119, Stroudsburg, PA, USA. Association for Computational Linguistics.

Christopher Olah. 2015. Understanding LSTM Networks. Accessed: 2018-03-18.

Richard Socher. 2016. CS224d Deep NLP Lecture 8: Recurrent Neural Networks.