Prepositional error correction using Tree Convolutional Networks

Anant Gupta

Code available online at https://github.com/anant0224/tree-convolution.

Problem

I've implemented a system to correct grammatical errors in English, particularly prepositional errors. Grammatical error correction (GEC) is an ongoing area of research in NLP. The current state-of-the-art systems use a machine translation approach: either phrase based statistical models or recurrent neural networks. These approaches use a general framework for correcting a broad class of errors. Therefore, they don't perform very well on any single error type. On the other end of the spectrum, there is the error classification approach. This is used for error types which involve confusion between usage of different words, eg. determiners (a, an, the), prepositions (at, of, by, etc), noun number (book vs books), etc. The ideology here is that each classifier handles a particular error type well instead of trying to correct multiple error types. However, recent classification approaches for articles and prepositions using DNNs have not performed well, and are still worse than the baseline Naive bayes classifiers. I suspect that this is due to the use of suboptimal neural network architectures, which are not designed to handle structured inputs like sentences.

Idea

I claim that sentences should be represented as dependency parse trees before feeding them to neural classifiers. The dependency parse tree provides a natural way to represent the structure of the sentence, highlighting the relations between different (possibly non-adjacent) words in the sentence. For prepositions, it turns out that just looking at the parent and child node is usually enough to predict the correct preposition. This is what I use as my baseline. However, some cases require you to look at the rest of the dependency tree as well. For handling those, I propose performing convolutions on the dependency tree over the immediate neighbourhood of each node. Doing this multiple times will let each node learn a high-level "phrase embedding" over its subtree. These phrase embeddings, which have context of the whole sentence, and contain high-level features can then be used to do the classification. Just like CNNs, my network benefits from a reduced parameter space through weight sharing.

Motivating examples

Here I try to illustrate through examples why the dependency parse tree is needed for preposition classification. The correct preposition to be used depends on the context of the whole sentence (words) and also the structure (dependency graph). Eg:
He gave the book **to** me.
He took the book **from** me.
In these sentences, the preposition to be used depends on the verb (**gave** vs **took**). This shows that the whole context is important.

What is the meaning **of** life?
I have been meaning **to** talk to you.
In these sentences, the preposition depends on the structure of the sentence (is the previous word a noun or a verb?).

Metrics

The metrics reported in literature for grammar error correction systems are Precision, Recall and F0.5 score. However, these numbers are only available for all error types combined. Eg. the winning team for the CoNLL-14 shared task had Precision, Recall and F0.5 of 52.44, 29.89 and 45.57 respectively for the combined errors. It is possible to define these metrics only for preposition errors, but there wouldn't be a standard to compare to. Therefore, I defined my own baseline to compare against. Also, I considered classification **accuracy** as the metric, since it's easier to calculate and is more intuitive to compare. The goal would be to beat the baseline in terms of accuracy.
I've limited the focus of the system to the following 13 preposition classes:
of, for, by, at, on, in, about, to, from, with, before, after, during

Data Preparation

Training and evaluation data:
I've used the BAWE corpus for extracting training and evaluation data. It contains a collection of assignment texts from universities, grouped by discipline and department. I chose **English** discipline assignments, since they will probably be the most grammatically correct. I read a few texts and they seem to be perfect with regard to grammar. I extracted ~22000 training samples and ~2500 evaluation samples.
First, the corpus had to be cleaned to remove some annotation tags. I did some regex pattern matching in Python to do that. Next, the sentences had to be tokenized, to break a single document into separate sentences, and to break each sentence into separate words. I used the NLTK library in Python to do this sentence and word tokenization.
Each training sample consists of a sentence (list of words), an index corresponding to the preposition occurrence, POS tags, dependency parse trees, parse labels and the true label. The

true label was obtained simply from the actual preposition used in the sentence (assumed to be correct). The POS tags, parse trees and labels were obtained using the SyntaxNet parser. I had to change the output code to print POS tags and parse labels for each word of the sentence in order. The dependency tree was printed in the form of an adjacency matrix, where each row corresponds to one word, and the entries are 0, 1 or -1 depending on whether the other word is unrelated, a child or a parent. Finally, I wrote a script to select sentences containing instances of one of the 13 prepositions. Sentences with more than 1 valid preposition were included more than once.

Testing data:
I've used the CoNLL-14 GEC (grammatical error correction) shared task dataset for testing my model. The test set has sentences containing grammatical errors, which have been marked and corrected by annotators. The errors are of various types, so I only considered the preposition replacement annotations and ignored the rest. For each sentence in the test set, I searched for occurrences of any of the 13 prepositions. If there existed a preposition replacement annotation for it, I considered it to be the ground truth. Otherwise, the original preposition was assumed to be correct and taken as the ground truth. Also, some sentences which contained another grammatical error immediately following a preposition error were ignored, because the errors might be related. The final testing dataset contained 2625 examples.

Software Implementation

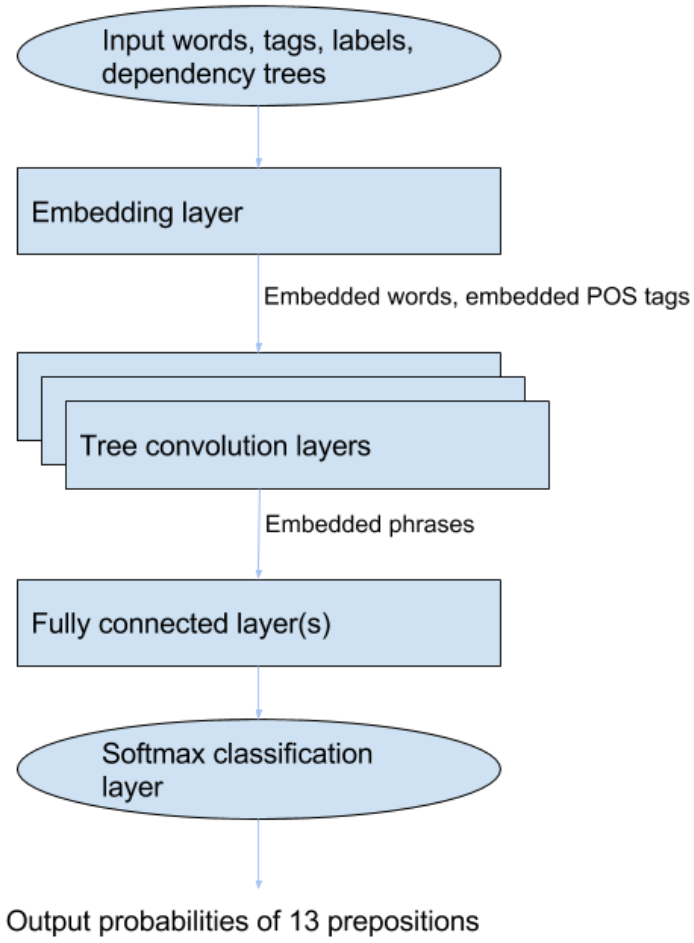
To generate POS tags, dependency parse trees and parse labels, I first tried installing SyntaxNet on my computer. SyntaxNet is a state-of-the-art parser developed by Google. I was able to install it, however, it did not seem to work well. For some sentences, the output was null. I suspected that there were some installation issues. Therefore, I set up a Google Cloud Platform account, and tried installing it there. That worked pretty well, so I used it to generate the features for all my training, evaluation and test example sentences.
Next, I installed Tensorflow to implement the actual neural network. I found a github project cnn-text-classification-tf which I forked so that I could reuse some parts of it for data input, vocabulary generation and loading of word embeddings. The actual network was written from scratch. My code is available at https://github.com/anant0224/tree-convolution.

Network architecture

The diagram shows the architecture for my preposition classifier system.
Embedding layer:
The network consists of an embedding layer, which converts sparse one-hot encodings of words into dense embeddings. There are several pre-trained weights available online for word embeddings, and I used the **GloVe** embeddings of dimension 50. The POS tags and labels are also embedded, and the weights for these are learned. The embedding dimensions for these are 4.

Tree convolution layer:
This is followed by the tree convolution layers. Each such layer outputs new word and POS tag embeddings, which we call **tree-weighted word and POS tag embeddings** respectively. Each dimension of the tree-weighted embedding at a node is a weighted combination of the same dimension of the input embedding at the parent, children and current node. The weights are formed by multiplying another weight matrix with the label embeddings of the parent, children and current node. The intuition is that for different roles (denoted by label embeddings) played by the parent and children nodes, their contribution to the current node's new embedding should be different. Eg. a noun node with a determiner child should put more weight on the current node than the child node, whereas a preposition node with a verb parent should put more weight on the parent. All tree convolution layers share these label-weights, so that the same operation of updating a node's embedding with its neighbours' embeddings is repeated, and each node ends up with an embedding of the entire phrase under it.
To allow for some flexibility in each layer's role, we define separate **parent** and **child** weights for each layer, which are multiplied with the parent's and children's label-weighted contributions before being added to the current node's label-weighted embedding. These weights can be seen as a way to control the *relative flow* of information from parents and children in each layer, allowing each node to end up with a representation of the entire-subtree (phrase) under it.

Fully connected and classification layers:
From the phrase embeddings output by the tree convolution layers, we select the ones corresponding to the preposition's parent and child, concatenate them and pass them to the fully connected layer. Finally, a softmax layer is used to output probabilities of the 13 prepositions.



Experiments and Results

Training on my computer usually converged within 5000 steps, and took about 30 minutes. Therefore, I was able to experiment with a lot of network configurations and hyperparameter settings. Here are the best configurations I obtained.

Baseline:

The baseline model consists of only an embedding layer, fully connected layers and softmax classification layer. The best evaluation accuracy was obtained using 3 fully connected layers, each containing 26 (twice the number of preposition classes) neurons.

My model:

The best results were obtained using 6 tree convolution layers, and a single fully connected layer containing 13 nodes.

Weight normalization:

Since my phrase embeddings were formed using weighted combinations of word embeddings, I thought that it would make sense to normalize the weights so that they sum to one. Instead of

constraining the weight matrices, I added normalization layers after the tree convolutions, which divided each phrase embedding with a normalization factor. However, the network failed to train, as the gradients started blowing up to infinity. I tried changing the learning rates, as well as weight initializations, but nothing helped. I ultimately gave up on weight normalization, but it wasn't all in vain as I discovered that I could increase the learning rate quite a bit without sacrificing training accuracy.

Activation function:
I experimented with various activation functions to add non-linearity between the tree convolutions. However, it didn't make any significant difference. ReLU, hyperbolic tangent, sigmoid, all seemed to give the same accuracy as linear activation.

Embedding dimension:
For the first set of experiments, I used the Google 300 dimension word2vec embedding. Note that all weight matrices used in the tree convolution and the first fully connected layer are proportional in size to the word embedding dimension. I was able to get an evaluation accuracy of ~70% with my model. However, even the baseline performed similar to this. On inspecting the training plots, I found that both were overfitting due to the large number of parameters. To make a fair comparison, I would have needed a lot more training examples and time (which I didn't have). Therefore, I switched to the 50 dimension GloVe embedding, hoping that the number of weights would now be small enough to meaningfully compare the tree convolution architecture with the baseline. I obtained 66% evaluation accuracies for my model, compared to 65% for the baseline. This is a significant enough improvement due to my proposed architecture. Even though I lost ~4% accuracy using the low dimension embedding, it was more insightful as it allowed me to compare the 2 models.

Conclusions and Future work

In conclusion, I have shown that it is useful to look at the context of the whole sentence to predict preposition. It remains to be seen what is the limit of this usefulness. Is it possible to improve the tree convolution process enough to see substantial gains in accuracy with limited data and network size, or do we have no other option than to train a large enough basic network to "memorize" the different combinations of occurrence?
The current way of performing convolutions is agnostic of the node level in the tree. This might interfere with the convergence of the phrase embeddings to stable values. Ideally, we would want to perform convolutions sequentially starting from the leaf nodes, then moving to the next level, and so on. This will be somewhat tricky to implement, but should lead to better results. There can also be other ways to learn phrase embeddings from word embeddings (Yu et al., 2015), using a combination of hand-designed and learned features.

References

http://tcci.ccf.org.cn/conference/2015/papers/173.pdf

https://arxiv.org/abs/1609.02907
https://arxiv.org/pdf/1606.00189.pdf
http://www.aclweb.org/anthology/N16-1042
https://dl.acm.org/citation.cfm?id=2002589
http://www.comp.nus.edu.sg/~nlp/conll14st/CoNLLST01.pdf
https://aclanthology.info/pdf/Q/Q15/Q15-1017.pdf