CS 388 HW2: Part-of-Speech Tagging with LSTMs

Mit Shah (UT EID: mks3226)
University of Texas at Austin
Austin, TX 78712
mkshah@utexas.edu

Abstract

This assignment is aimed at exploring the usage of Bidirectional Long Short Term Memory networks (BiLSTMs) in Part-Of-Speech (POS) tagging. Initially Recurrent Neural Networks (RNN) were being used for such NLP tasks. But sentences does contain long term dependencies, such as subject-verb agreement which due to vanishing gradients problem, RNN can not handle. To overcome this shortcoming, Long Short Term Memory (LSTM) networks were introduced. Here, an extended version of them which processes sentences in both the directions - BiLSTMs are used for the purpose of POS Tagging. Also, orthographic features of words are combined with BiLSTMs in two different manners in order to improve performance for the same and analyze it.

1. Introduction

We will be using a combination of BiLSTMs and orthographic features for the purpose of Part of Speech Tagging. So, let's first get an overview about LSTMs in general, orthographic features and Part of Speech Tagging.

1.1. Long Short Term Memory (LSTM) Networks

A LSTM unit is a recurrent neural network unit that is capable of remembering values for either long durations of time. The key to this ability is that it uses no activation function within its recurrent components. Thus, the stored value is not iteratively reduced over time, and the gradient term does not suffer from vanishing problems when we train it through back-propagation for a longer period pf time.

Mainly, architecture of LSTM can be described as follows. It adds additional gating units in each memory cell. Generally three gates: Forget gate, Input gate, and Output gate are added. They allow network to retain information over a long period pf time. Each unit maintains a Cell State vector. Forget gate helps removing existing information of a prior subject when new one is encountered, while Input gates add new information. Both of them do so by updating the Cell State vector. Output gate uses Sigmoid function in order to determine which elements of the cell state to output. This was an abstract level overview for LSTM unit.

An LSTM network is an ANN that has LSTM units in addition to other network units. It can be single or multilayer. They are mainly used in Language Modeling, Sequence Labeling and Sequence Classification.

1.1.1 BiLSTMs

They are a small extension of regular LSTMs, in which two separate LSTMs process sequence forward and backward. At each time stamp, hidden layers are concatenated to from the output.

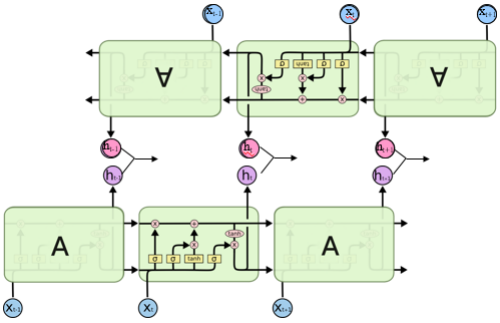

Figure 1. Bidirectional LSTM

1.2. Orthographic features

Orthographic features, in some sense describe some of the aspects of the words. Mainly, they can be various prefixes, suffixes or something like whether the word contains a number, hyphen, its first letter is capitalized or not etc. They help cluster the words according to these aspects. As we will see later, they are helpful in labeling out of vocabulary (OOV) - unknown words during testing.

1.3. Part of Speech (PoS) Tagging

It is the process of marking up a word in text to a particular part of speech, based on both its definition and context

of the sentence it is occurring in. In simple terms, it is the process of marking words as Nouns, Verbs, Adverbs, Adjectives, etc.

2. Method

Three approaches to perform POS tagging is presented in this paper using BiLSTMs. We will discuss each of the approaches in the following subsections. Their performances are measured by Accuracy, Loss and accuracy on Out Of Vocabulary (OOV) words.

2.1. Without using any Orthographic Features (Model 1)

Here, only words themselves are used as input to the LSTMs. After collecting all the words from the training data, they are embedded into a low dimensional space of 300. This 300 dimensional feature vector is then fed to the LSTMs. The output generated by LSTM is then passed to final POS classification layer on top of that to produce POS tags as output.

2.2. Concatenating Orthographic Features to LSTM input (Model 2)

In this model, we concatenate orthographic features in one hot vector form in the input layer of LSTMs with the words.

2.3. Concatenating Orthographic Features to LSTM output (Model 3)

Here, we concatenate the orthographic features in one hot vector form to the output of LSTMs and then they are passed to the final POS classification layer.

3. Implementation

First of all let's discuss, how one of the evaluation metric - Out Of Vocabulary (OOV) words was calculated. Then we will discuss about selecting orthographic features and adding them at the desires places.

3.1. Calculating OOV Accuracy

While preprocessing the files in order to generate a unique word id for each word, a dictionary structure is used. In training mode, each new word is assigned a id that is current length of the dictionary. While if a new word is found during Validation mode, it is treated as an unknown word and all such words are assigned the same id, that is current length of the dictionary. Similar dictionary structure is maintained for getting ids for POS tags.

For calculating the accuracy of only OOV words, an oov mask is created that works in similar manner to pad mask. It multiplies all the indexes with word id that is not equal to the current length of the dictionary with zero. So, only OOV words are left and then we can calculate their accuracy.

3.2. Extracting Orthographic Features

In this section, we will discuss exact orthographic features used and how they were parsed. Mainly words were checked for different prefixes and suffixes and at the end, they were converted into one hot vector.

3.2.1 Prefix features

Initially, a dictionary structure containing 23 different prefixes and their ids was defined. Not all the prefixes were given different ids. Prefixes which semantically meant same things were assigned the same ids, in order to convey this information to the model. For example, prefixes "en-" and "em-" were given the same id. Here is the complete list of prefixes used and their ids:

$\{"anti" : 1, "de" : 2, "dis" : 3, "en" : 4, "em" : 4, "fore" : 5, "in" : 6, "im" : 6, "il" : 6, "ir" : 6, "inter" : 7, "mid" : 8, "mis" : 9, "non" : 10, "over" : 11, "pre" : 12, "re" : 13, "semi" : 14, "sub" : 15, "super" : 16, "trans" : 17, "un" : 18, "under" : 19\}$

3.2.2 Suffix features

In same manner, a dictionary structure was created for 30 suffixes. Here is the complete list of suffixes used and their ids:

$\{"able" : 1, "ible" : 1, "al" : 2, "ial" : 2, "ed" : 3, "en" : 4, "er" : 5, "est" : 6, "ful" : 7, "ic" : 8, "ing" : 9, "ion" : 10, "tion" : 10, "ation" : 10, "ition" : 10, "ity" : 11, "ty" : 11, "ive" : 12, "ative" : 12, "itive" : 12, "less" : 13, "ly" : 14, "ment" : 15, "mess" : 16, "ous" : 17, "eous" : 17, "ious" : 17, "s" : 18, "es" : 18, "y" : 19\}$

3.2.3 Parsing words

For parsing words, they were written in the form of regular expressions containing prefix, root word and suffix. Prefix and suffix should be one of those present in the dictionaries mentioned above. Following regular expression was used:

$word = (ZeroOrMore(prefix)("prefixes") + SkipTo(suffix|endOfString)("root") + Optional(suffix)("suffix"))$

3.2.4 Constructing One-hot vector

Once a word broken down into (prefix, root, suffix); if present, prefix and suffix are looked up in their dictionaries to get the ids. If they are not present in the word, they are assigned id 0. So, along with word id and pos tag id vector in the original code, now we have a prefix vector and

suffix vector also. Both of them are then converted to on-hot vectors. Then these one-hot vectors can be concatenated to either at LSTMs input or at output, as per the experiment.

3.3. Using Orthographic features

Once calculated, they were passed in the code at all places wherever word ids and corresponding pos ids were passed. For doing so appropriate changes were made in code including initialization, creating placeholders, etc.

4. Experiments and Results

As discussed in the second section, experiments were performed for three different approaches. Penn TreeBank Dataset used for this purpose.

4.1. Qualitative Analysis

Let's look at training accuracies for all of them. On 2, it is shown for 700 batches. We can see that adding orthographic features at the output does not cause much improvement over the one without them. For both of them, values are almost equal everywhere. On the other hand, we can see that while adding at input, accuracy picks up very early, grows at a fast speed and reaches saturation much early than other 2 models. So, adding orthographic features at LSTM inputs, clearly have huge improvement.

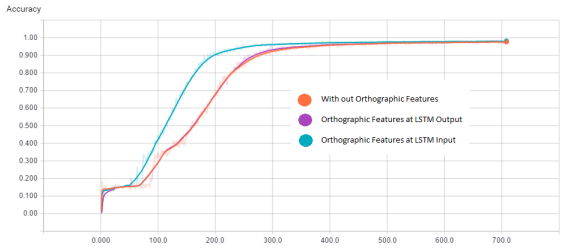

Figure 2. Training Accuracy

As expected, same behavior can be noticed in the training loss 3. Loss for the model 2 drops significantly early and quickly than other two models.

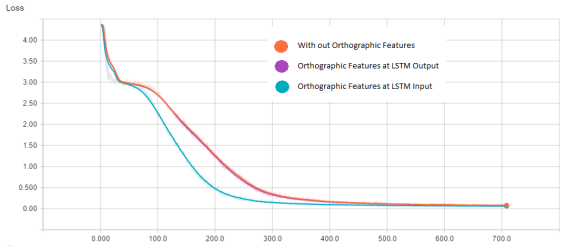

Figure 3. Training Loss

In validation accuracy also, similar trends are observed 4. Model 2 performs much better than model 1 or 3. But we

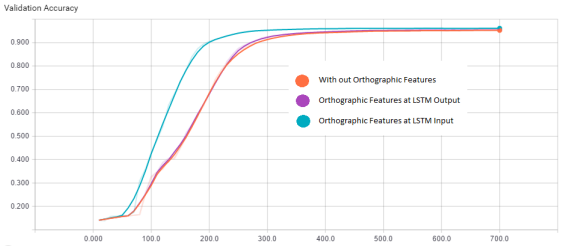

Figure 4. Validation Accuracy

can see that model 3 slightly performs better at some of the places than model 1.

Similarly, for validation loss also model 2 performs significantly better than others, while model 3 performs slightly better than model 1 5.

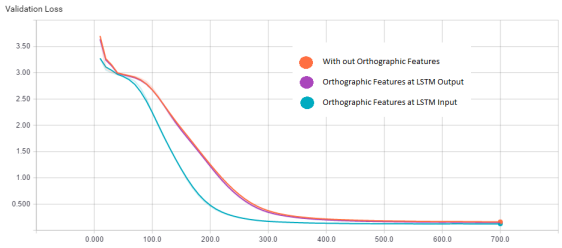

Figure 5. Validation Loss

For OOV accuracy on validation data, differences are much more visible 6. Again, model 2 performs far more better than others. But also main difference from previous plots is that, model 3 performs much better than model 1.

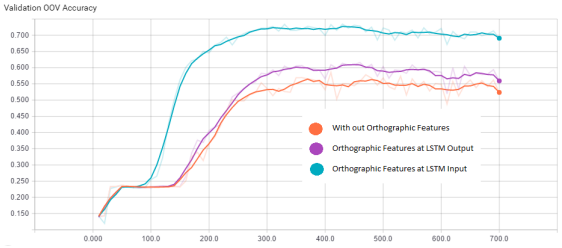

Figure 6. Validation OOV Accuracy

4.1.1 Reasoning

So, from the discussion above we can conclude the following in terms of accuracy/loss:

1) Training Accuracy: Adding orthographic features at LSTM input increases training accuracy much faster, while adding them at the output does not cause any improvement.

2) Validation Accuracy: Same as for training accuracy, except adding orthographic features at LSTM output causes slight improvement.

3) OOV Accuracy: Last plot 6 suggests that adding orthographic features indeed helped in improving POS tag-

ging for unknown words. Also, they caused much more improvement when added at LSTM inputs compared to adding them at LSTM outputs.

Now, let us try to understand why this happens by understanding the behavior of these models.

As discussed in section 1, orthographic features capture some aspects of the word. So, while training those 'aspects' are also mapped to a particular POS tag in addition to words. So, when you encounter a new word during validation / testing, you still can extract these orthographic features and try to assign most probable POS tag based on that. This is why we see huge increase in OOV accuracy. This is why Models 2 and 3 perform better than Model 1.

Now, these 'aspects' can also have long term dependencies, based on which POS tag to them can be assigned. So, when we add them to the LSTM input, LSTM also processes them and captures such dependencies which leads to significant improvement in the performance. But if we add them to the output of LSTM, only one final classification layer makes use of them and LSTM do not process them. So, such dependencies are not captured and it does not cause any significant improvement. This is why Model 2 performs better than Model 3.

4.2. Quantitative Analysis

Now, let's turn to quantitative results and check the best accuracies achieved by this models. All the three models were run for 6 epochs. As we can see from 4 and 6, validation accuracy keeps increasing smoothly, while oov accuracy keeps fluctuating after getting the highest value. In table 1, highest oov accuracy achieved by each model is displayed and corresponding validation accuracy and loss are displayed. We can see that three models achieved highest OOV accuracies of 58.5%, 73.5% and 61.6% respectively.

In order to maximize OOV Accuracy for test, checkpoint files nearest to this time were used for testing, i.e, checkpoint files after 400, 450, and 450 batches respectively for the 3 models.

Accuracy, loss and OOV accuracy for testing are shown in table 2. We can see that OOV accuracies achieved by these 3 models are 58.2%, 72.1% and 60.8% respectively.

4.3. Runtime Analysis

These experiments were done on the CS machines of the department. Coming to runtime; Model1, Model2 and Model3 took 1897.29s, 2218.14s and 1866.57s respectively. We can see that Model1 and Model3 almost took same time. While Model2 took around 400s more time than them. This increase is likely due to passing additional orthographic one-hot vectors in LSTMs.

Table 1. Highest OOV accuracy achieved by all the three models, and corresponding Validation accuracy and loss.

Model	Batches	Validation Accuracy	OOV Accuracy
No O.F	410	94.4%	58.5%
O.F. at Input	430	96%	73.5%
O.F. at Output	460	95.1%	61.6%

Table 2. Test Accuracy, Loss and Test OOV Accuracy

Model	Test Accuracy	Test OOV Accuracy
No O.F.	94.3%	58.2%
O.F. at Input	95.2%	72.1%
O.F. at Output	94.5%	60.8%

5. Conclusion

In this assignment, we used Bidirectional LSTMs to perform Part Of Speech (POS) tagging task. In addition we also made use of orthographic features of a word, which mainly included various prefixes and suffixes in one hot vector form. It was concluded that adding orthographic features to the input of BiLSTMs improved the Out Of Vocabulary (OOV) accuracy significantly and adding them at the output also showed some improvement. Reasoning for the same was discussed.