CS 388 Natural Language Processing
Homework 2: Sequence Labeling

Victoria Anugrah Lestari (val565)

March 9, 2017

1 Introduction

The purpose of this assignment is to perform sequence labeling in the WSJ (Wall Street Journal) corpus using bidirectional LSTM with TensorFlow. We are instructed to augment the word embeddings with orthographic features and then compare the accuracy of the baseline method with the accuracy of the augmented method, both for overall accuracy and OOV (out-of-vocabulary) accuracy.

The dataset is obtained from the WSJ corpus, which contains more than 1 million words and 34 POS tags. Using the standard training split, the training set contains 851,676 words. The validation set contains 119,194 words, and the test set contains 134,596 words. Out of the words in the validation set, 4,337 words are OOV words. In the test set, there are 3,815 OOV words.

2 Methods

In essence, there are two tasks that we are supposed to do: (1) extracting orthographic features from the words and represent them as one-hot vectors, and (2) add those orthographic features to the word vectors and include them in the BiLSTM code.

There are two approaches to include the features in the BiLSTM code. The first is to concatenate them onto the input layer. The other is to concatenate them onto the output layer. We are instructed to compare the performance of both approaches.

2.1 Orthographic Features

For the orthographics features, I used 41 common English prefixes[1], 136 English suffixes[2], and binary flags for words that are capitalized, start with a digit, or contain hyphen.

2.1.1 Encoding

I encoded prefixes and suffixes the way the POS tags are encoded. Since there are 41 prefixes, I assigned the numbers 1 to 41 for each prefix and the number 0 if a word does not have a prefix. Similarly, I assigned the numbers 1 to 136 for each suffix and the number 0 if a word does not have a suffix. The lists of prefixes and suffixes can be seen in the appendix. For other features, I specified a column for each feature. I assigned 1 if the condition is true and 0 otherwise. If a word is capitalized, the column for capitalization will have the value 1. The same goes for digit-containing words and hyphen-containing words.

The orthographic features are appended to each word embedding. Previously, each word was represented by a 2-tuple. The first element is the word encoding in the dictionary, and the second element is the POS tag encoding. Now, each word is represented by a 7-tuple. The first two elements are the same. The third element is the prefix encoding, the fourth is the suffix encoding, the fifth is capitalization encoding, the sixth is the "start with a digit" encoding, and the seventh is "contains a hyphen" encoding. Note that the fifth, sixth, and seventh elements could only have values of either 0 or 1.

For example, the word "safety/NN" has the encoding (8, 5, 0, 128, 0, 0, 0). The fourth element is 128 because it has the suffix "-ty". Another example is the word "Transportation/NNP" that has the encoding

[1] https://www.learnenglish.de/grammar/prefixtext.html
[2] https://www.learnthat.org/pages/view/suffix.html

(1, 1, 0, 22, 1, 0, 0). The fourth element is 22 because it has the suffix "-tion", and the fifth element is 1 because its first letter is capitalized.

2.1.2 Code Implementation

I added three class variables: `prefix_words`, `suffix_words`, and `other_words`, which are TensorFlow placeholders. Similar to `input_words` and `output_words`, `prefix_words` and `suffix_words` have dimension BATCH_SIZE $\times$ sequence length. The encoding of prefixes and suffixes are then fed into these placeholders when we train the model. `prefix_words` and `suffix_words` are converted into one-hot vectors before they are concatenated with `lstm_input`.

The other placeholder `other_words` is slightly different from the previous two. Since I stored the 'other' features in a 3-dimensional matrix (with dimension number of words $\times$ length of sentence $\times$ 3), `other_words` has dimension BATCH_SIZE $\times$ sequence length $\times$ 3. Because the values in `other_words` are either or 1, I did not convert it into one-hot vectors. I just concatenated it with `lstm_input`.

2.2 Computing the OOV Accuracy

I created a method similar to `get_mask` (the method to find words that are not padding) to find the OOV words. However, instead of using `tf.not_equal`, I used `tf.equal` to know if a word is an OOV word. Since each OOV word is encoded with the size of input dimension -2, the method compares the word encoding with that number and returns 1 if the word encoding equals that number. To compute the OOV accuracy, I used `compute_accuracy` and replace the regular mask with OOV mask. The OOV accuracy is calculated by dividing the sum of OOV accuracy for each word by the total number of OOV words.

3 Experiments and Results

I conducted three experiments as instructed. The first experiment is the baseline. I only added the method for computing OOV accuracy in the original `pos_bilstm.py` code. The second experiment is concatenating orthographic features to the input layer. The third experiment is concatenating orthographic features to the classification layer. Each experiment is discussed in detail in the following subsections.

3.1 Baseline

The baseline accuracy and loss starts to converge when the batch size is above 300 and close to 400. Figure 1 shows the training accuracy and loss for the baseline experiment.

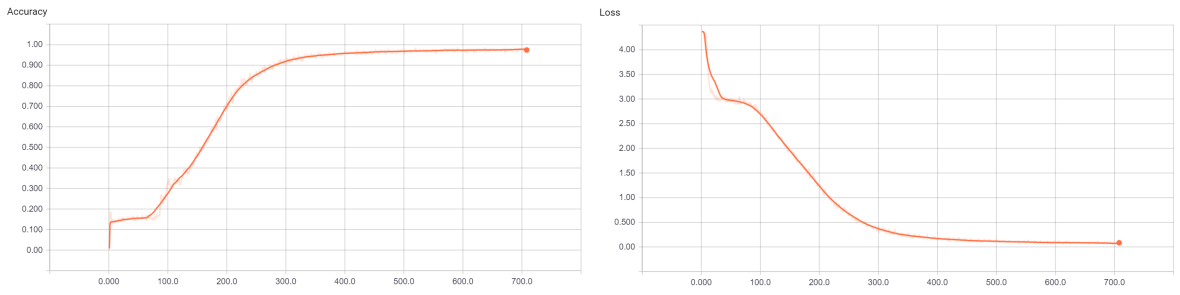
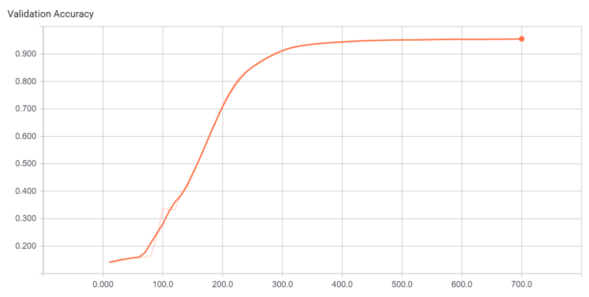


Figure 1: Accuracy (left) and loss (right) for baseline experiment

The validation accuracy and loss converge faster than the training accuracy and loss. They start to converge when the batch size is 300. For 700 batches, the validation accuracy is 95.5% and the validation loss is 15.8%. Figure 2 shows the graphs for the validation accuracy and loss for the baseline experiment.



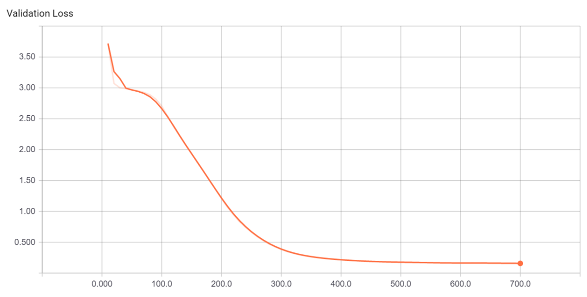


Figure 2: Validation accuracy (left) and loss (right) for baseline experiment

The OOV validation accuracy for the baseline experiment reached its peak at 59.8% for 430 batches, but it dropped again to 58.5% for 700 batches. Figure 6 shows the graph for the OOV validation accuracy.

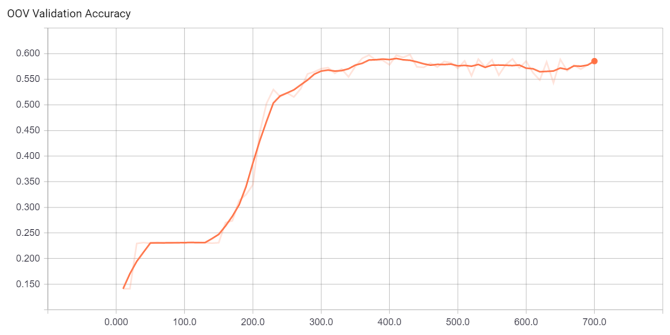
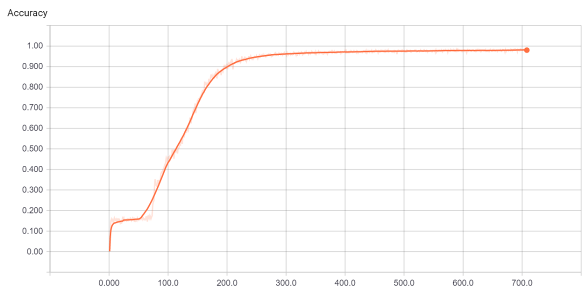


Figure 3: OOV validation accuracy for input-layer experiment

The running time for the baseline experiment is 31.48 in the CS server.

3.2 Concatenating Features with Input Layer

When the orthographic features are included in the input layer, it only takes 200 batches to start to converge. We also got higher accuracy and lower loss compared to the baseline. Figure 4 show the accuracy and loss graphs for the input-layer experiment.



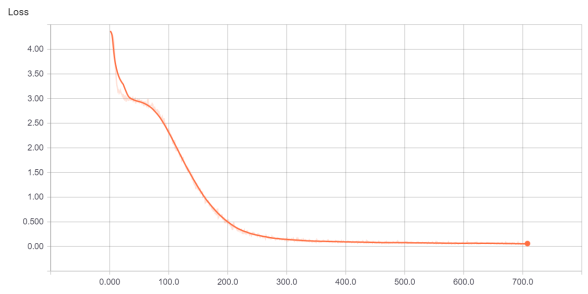
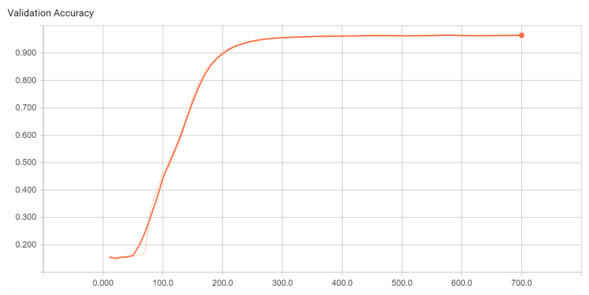


Figure 4: Accuracy (left) and loss (right) for input-layer experiment

The validation accuracy and validation loss of the input-layer experiment are significantly improving. The validation accuracy is 96.5% and the validation loss is 11.3%.



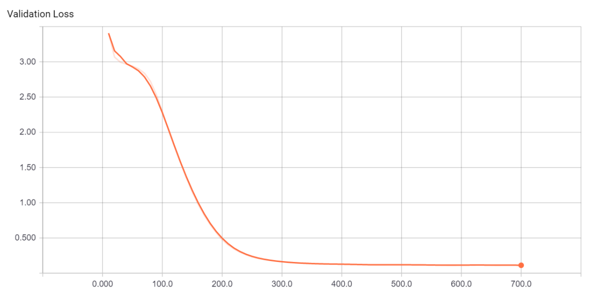


Figure 5: Validation accuracy (left) and loss (right) for input-layer experiment

The OOV validation accuracy of the input-layer experiment sharply increases from the baseline experiment. It reaches 80.3% for 430 batches but drops to 78.3% for 700 batches. It improves about 20% accuracy from the baseline experiment.

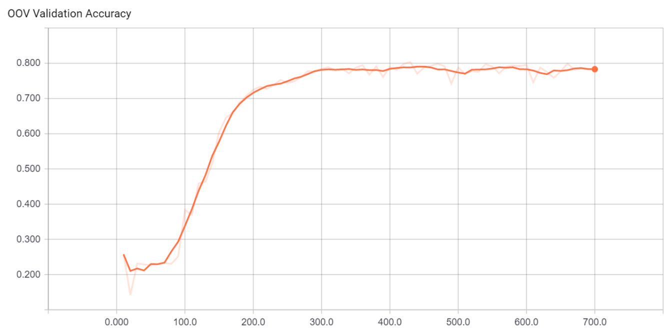
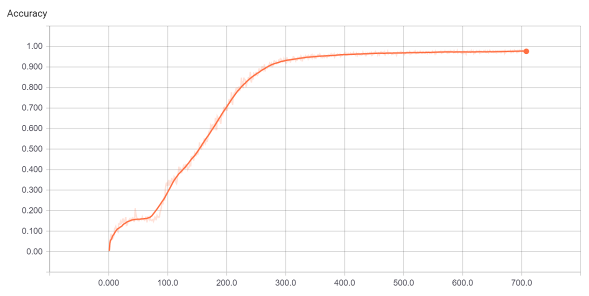


Figure 6: OOV validation accuracy for input-layer experiment

The running time for the input-layer experiment is significantly slower, 41.09 minutes in the CS server.

3.3 Concatenating Features with Classification Layer



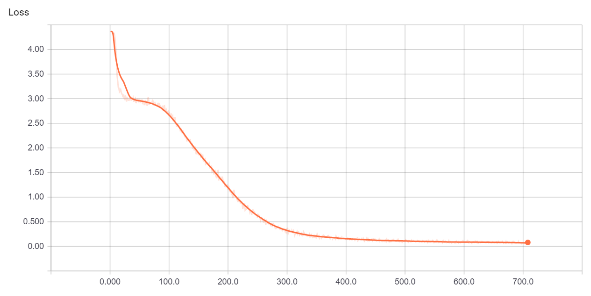
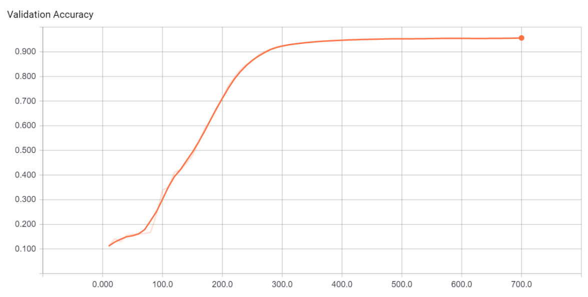


Figure 7: Accuracy (left) and loss (right) for classification-layer experiment

The classification-layer experiment produces results similar to the baseline experiment. The validation accuracy is 95.6%, which means it improves by 0.01%. The OOV accuracy is 59.8%, which means it improves by about 2%. The running time is 34.77 minutes in the CS server.



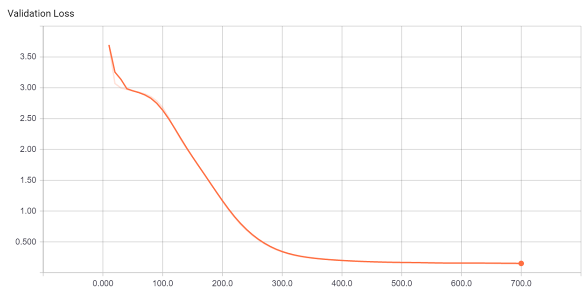


Figure 8: Validation accuracy (left) and loss (right) for classification-layer experiment

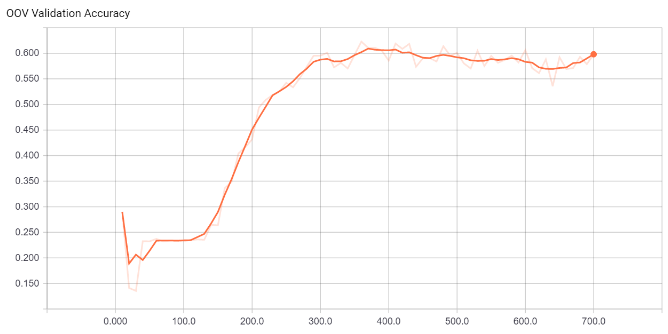


Figure 9: OOV validation accuracy for classification-layer experiment

4 Conclusion

Table 1: Comparison of accuracy between baseline, input-layer, and classification-layer experiments

	Overall Accuracy	OOV Accuracy	Test Overall Accuracy	Test OOV Accuracy	Running Time
Baseline	95.5%	58.5%	95.7%	56.7%	1888.811 s (31.48 min)
Input	96.5%	78.3%	96.5%	79.3%	2465.315 s (41.09 min)
Classification Layer	95.6%	59.8%	95.7%	58.6%	2086.150 s (34.77 min)

Adding orthographic features into the input layer slightly improves the overall accuracy and greatly improve the OOV accuracy. The OOV accuracy is increased because the features are crucial in determining the POS tag of a word, even if the word is unknown. For example, a word with '-tion' suffix is almost always a noun (NN), and a word with '-ize' suffix is almost always a verb (VB). A capitalized word has a high probability of being a proper noun (NNP), and a word with numbers probably has a tag CD (cardinal number).

However, the running time of the BiLSTM also increases. The reason is that the features are concatenated to the input vectors, so that the vectors have greater length. In consequence, the processing time also increases.

It is also interesting to note that adding orthographic features to the input layer improves the OOV accuracy by about 20%, while adding them to the classification layer hardly improves the OOV accuracy. The reason is that by adding the features to the input layer, the features are included while the LSTM is training the model. Thus, the features are significant in determining the word label.

On the other hand, adding the features to the classification layer does not include the features in training the model. They are only added after the model has been trained. Consequently, the accuracy improves slightly. However, the running time of the process is not as long as that of the input-layer experiment.

Appendix

Here is the list of prefixes:

```
['ante', 'anti', 'circum', 'co', 'de', 'dis', 'em', 'en', 'epi', 'ex', 'extra', 'fore',
'homo', 'hyper', 'il', 'im', 'in', 'ir', 'infra', 'inter', 'intra', 'macro', 'micro',
'mid', 'mis', 'mono', 'non', 'ob', 'omni', 'para', 'post', 'pre', 're', 'semi', 'sub',
'super', 'therm', 'trans', 'tri', 'un', 'uni']
```

Here is the list of suffixes:

```
['able', 'ac', 'acity', 'ocity', 'ade', 'age', 'aholic', 'oholic', 'al', 'algia',
 'an', 'ian', 'ance', 'ant', 'ar', 'ard', 'arian', 'arium', 'orium', 'ary',
 'ate', 'ation', 'ative', 'cide', 'cracy', 'crat', 'cule', 'cy', 'cycle', 'dom',
 'dox', 'ectomy', 'ed', 'ee', 'eer', 'emia', 'en', 'ence', 'ency', 'ent',
 'er', 'ern', 'escence', 'ese', 'esque', 'ess', 'est', 'etic', 'ette', 'ful',
 'fy', 'gam', 'gamy', 'gon', 'gonic', 'hood', 'ial', 'ian', 'iasis', 'iatric',
 'ible', 'ic', 'ical', 'ify', 'ile', 'ily', 'ine', 'ing', 'ion', 'ious', 'ish',
 'ism', 'ist', 'ite', 'itis', 'ity', 'ive', 'ization', 'ize', 'less', 'let',
 'like', 'ling', 'loger', 'logist', 'log','ly', 'man','ment', 'ness', 'oid', 'ology',
 'oma', 'onym', 'opia', 'opsy', 'or', 'ory', 'osis', 'ostomy', 'otomy', 'ous',
 'path', 'pathy', 'phile', 'phobia', 'phone', 'phyte', 'plegia', 'plegic', 'pnea', 'scopy',
 'scope', 'scribe', 'script', 'sect', 'ship', 'sion', 'some', 'sophy', 'sophic', 'th',
 'tion', 'tome', 'tomy', 'trophy', 'tude', 'ty', 'ular', 'uous', 'ure', 'ward',
 'ware', 'wise', 'woman','y']
```