Homework 3: Sequence Tagging of Tweets

Seokchan Ahn
Department of Computer Science
University of California, Irvine
seokchaa@uci.edu

Abstract

Sequence tagging, often dubbed as sequence labeling is a task that assigning a tag to each word in a sentence. The objective of this homework is to investigate context dependencies and label dependencies on POS tagging and NER tagging tasks. For the model to take account of the context dependencies and label dependencies, I added bi-LSTM encoders and used CRFs, respectively. Both methods improved the POS tagging accuracy because a POS tag relies on the tags around it. However, they did not improve NER tagging accuracy because it is difficult to distinguish a named entity and a general noun based on the context. Using pre-trained embeddings also helped for both tasks, but overall NER accuracy went down because of the increased false positives.

1 Dataset

Twitter POS dataset and Twitter NER dataset are used for the experiments. Fig 1 shows the frequency of each tag for the POS and NER tasks. NER dataset is severely imbalanced, 94.7% of the total words are labeled as 'O'.

2 Implementing Accuracy Per Label Metric

Accuracy is the most frequently used metric for evaluation, but accuracy over all possible labels is not that informative if a model perform good on some labels and poor on other labels. For example, a NER tagger model that simply tags 'O' for every word can achieve the accuracy of 94.7% for the Twitter NER dataset while doing nothing. Thus, we also have to take account of the per-tag accuracy. I implement the *AccuracyPerLabel* class that counts per label and tested using *metric_test.py*.

3 Improving Simple Tagger

I perform experiments to improve the simple tagger by using GloVe (Pennington et al., 2014), a pre-trained embeddings with the size of 50 and introducing bidirectional LSTM (Hochreiter and Schmidhuber, 1997) encoder layers between the embedding layer and the linear projection layer.

For encoder, the best hyper-parameter is found using grid search over hidden size $h \in [20, 25, 30]$ and number of layers $l \in [1, 2, 3]$. The best models found were $(h = 30, l = 3), (h = 25, l = 1)$ for POS with and without pre-trained embeddings, $(h = 25, l = 1), (h = 30, l = 1)$ for NER. Results are on the Table 1.

Models	w/o GloVe		w/ GloVe	
	dev	test	dev	test
POS Simple	0.64	0.63	**0.78**	**0.80**
+ Encoder	0.79	0.79	**0.85**	**0.86**
NER Simple	0.94	0.90	0.93	0.87
+ Encoder	0.95	0.90	0.95	0.90

Table 1: Simple tagger accuracy for different settings

3.1 Using Pre-trained Embedding

The effect of using a pre-trained model was significant for the POS tagging task, but it did not help or make the accuracy even worse for NER. In GloVe word embeddings, similar words are represented as similar vectors because they are trained using word-word co-occurrence statistics in the train corpus. Therefore, it would help classifying part-of-speech because similar words are likely to have similar part-of-speeches. NER accuracy was dropped after using GloVe, but tag-wise, accuracy for 'O' was slightly dropped while accuracy for other tags was improved a lot. Considering the weights for the cells in the confusion matrix, we may not conclude the word embeddings are not helpful for the NER tagging task.

3.2 Adding Encoder

Like adding word embeddings, adding an encoder improved the accuracy a lot for the POS tagging

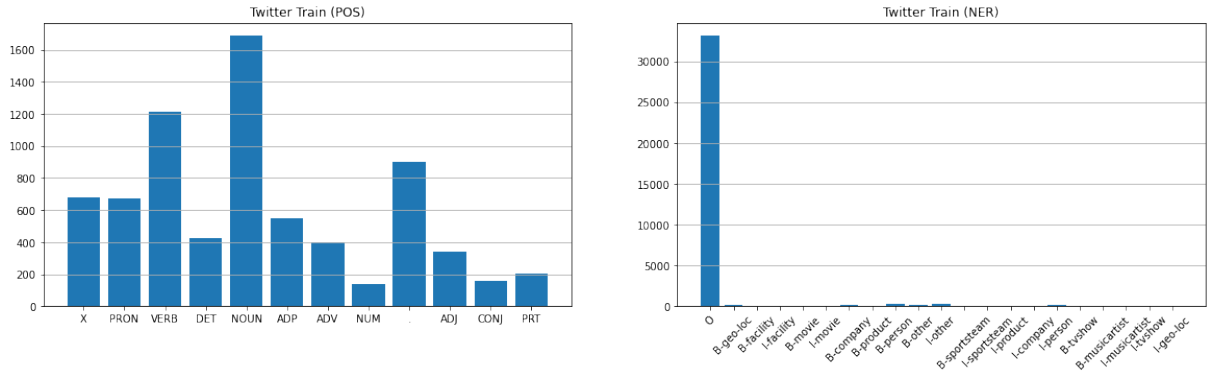

Figure 1: Training sets for POS and NER tasks. 94.7% of words are tagged as 'O' in the NER training set.

task, but the effect was not significant for the NER task. Bidirectional LSTM encodings contain contextual information because hidden layers are connected both forward and backwards. Therefore, giving such information would help classifying the part-of-speech of a word because it is decided by the words around it. However, for the NER task, adding encoder made the model more likely to classify the word as 'O', slightly improving the overall accuracy but harming the accuracy for the tags other than 'O'. This might because the encoder learned to assign 'O' to all the tags to maximize the overall accuracy.

3.3 Changing both Embedder and Encoder

For POS tagging task, using both pre-trained word embeddings and encoder layer achieved the best performance because the reasons discussed above. For the NER tagging task, adding encoder layer over pre-trained word embedding compensated the performance degradation, while not classifying most of the words as 'O'. This might because the model can distinguish the words from the word embedding information.

3.4 Accuracy Improvement per Tag

For POS tagging task, 'X' tag improved the most, by 0.50 from 0.31 to 0.81. The tag 'X' is used for words that cannot be assigned a part-of-speech category, '@paulwalk', a twitter mention for example. Such tags are very common in Twitter corpus and easily clustered by word embeddings and recognizable by contextual information.

For POS tagging task, introducing embedding layers degraded accuracy for all the tags other than 'O' to almost zero. The overall accuracy stayed the same because the frequency of 'O' dominates others, taking 94.7% of the words. When added GloVe only, accuracy for 'B-person' improved the most, by 0.39 from 0.30 to 0.70. This might because famous entities such as politicians' names are easily recognizable in GloVe representations.

4 Viterbi Decoding Implementation

Viterbi algorithm is implemented using dynamic programming. Dynamic programming is an optimization that avoiding identical computations by storing the results of the subproblems in the memory that can be reused for the future computation. I implemented the Viterbi algorithm in three steps. First, I initialized the $T[0, y]$ for all the label ys by adding start scores and emission scores for the first token. Then, I filled up the table $T[i, y]$ by using dynamic programming and stored the $bp[i, y]$ that stores pointer to the previous cell in the table. Finally, I returned the best score and the path for the score by choosing the maximum value on the table and following the $bp[i, y]$ table back to the first label.

5 Improving CRFs and Compare CRFs to Simple Tagger

Models	w/o GloVe		w/ GloVe	
	dev	test	dev	test
`POS CRF`	0.75	0.74	**0.80**	**0.81**
`+ Encoder`	0.78	0.78	**0.85**	**0.85**
`NER CRF`	0.95	0.90	0.94	0.88
`+ Encoder`	0.95	0.90	0.95	0.90

Table 2: CRF tagger accuracy for different settings

Unlike the simple models, conditional random fields (CRFs) (Lafferty et al., 2001) utilize not only the arbitrary features, but also the sequential information of the labels.

The same experiments are performed on CRFs and the results are on Table 2. The best models found were $(h = 30, l = 2), (h = 25, l = 1)$ for POS with and without pre-trained embeddings, $(h = 25, l = 1), (h = 20, l = 1)$ for NER.

I leave out the detailed explanations about improving CRF tagging models because the effects of adding pre-trained embeddings and bi-LSTM encoder layers were almost the same as the simple models for the same reasons.

5.1 Default CRF vs. Default Simple Tagger

For the POS tagging task, The CRF model with the default settings performed much better than the default simple model, surpassing by 0.11 both for dev and test set. This is because CRFs utilizes sequential information of the labels and a part-of-speech tags can be inferred by the neighboring tags. For example, since English is an SVO language, a subject is always followed by a verb and we can expect a word after a subject tag would have a verb tag.

In contrast, NER accuracy stayed the same after changing the model. This is because a named entity cannot be inferred by the tags around it, except for the cases such as a tag between B and I or O and I, which is very rare in the entire dataset.

5.2 Default CRF vs. Best Simple Tagger

For the POS tagging task, the default CRF accuracy was lower by almost 0.1 compared to the best simple tagger. This is because the best simple tagger uses the pre-trained word embeddings and the bi-LSTM encoder also can give the contextual information, similar to the ψ_t, ψ_s, ψ_e terms in the CRFs do.

For the NER tagging task, there was no significant difference between the default CRF and the best simple tagger models for the same reason as above.

5.3 Best CRF vs. Default Simple Tagger

For the POS tagging task, the best CRF model outperformed the default simple tagger by more than 0.1. This is because the default simple tagger is not using the contextual information at all. The NER model performed the almost the same for the same reason.

5.4 Best CRF vs. Best Simple Tagger

The best CRF models and the best simple models showed almost the same results. This shows that the gain from introducing CRFs can also achieved by the bi-LSTM encoders because both provides the contextual information which is helpful for the POS tagging task.

5.5 Accuracy Improvement per Tag

For POS tagging task, 'NUM' tag improved the most, by 0.44 from 0.32 to 0.76. This is because numbers are often followed by nouns like 'years' or 'pm', which are easy to expect by the tags around them.

For POS tagging task, accuracy for 'B-person' improved the most, by 0.44 from 0.31 to 0.75, when GloVe is added to the default model. This also account for the corpus GloVe is trained on.

6 Statement of Collaboration

I did this homework by myself without collaboration.

References

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 282–289, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.