Named entity recognition

Bota Duisenbay
La Sapienza University of Rome
duisenbay.1849680@studenti.uniroma1.it

1 Introduction

Named Entity Recognition (NER) is a NLP classification task, where for every word there is a named entity label. For this task, labels are Person (PER), Location (LOC), Group (GRP), Corporation (CORP), Product (PROD), Creative Work (CW). Apart from that there is included prefixes to represent beginning of the chunk (B), inside the chunk (I) and out of the chunk (O), meaning that it doesn't belong to any entity label.

2 Dataset

The given dataset given as training and validation sets: containing 14535 and 765 sentences, with 240058 and 12751 tokens respectively. Tokens contains non English, non alphanumeric and punctuation symbols. The distribution of tags are represented in the pie chart ref to the picture 1

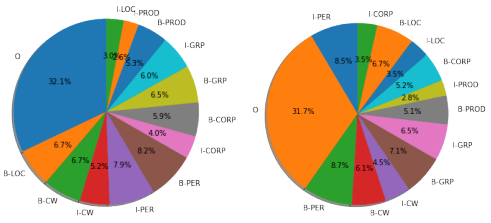

Figure 1: Tag distribution in train and validation sets

2.1 Pre-processing

The training and validation data are read from the table, where data is organized as word - label pair in each line and including lines for indication of beginning of the sentences. Words are converted into lowercase and cleaned by removing special characters, instead of leaving only English and numeric character, because words may contain letters from non-English alphabets. Words and tags are stored each sentence-wise and separately.

2.2 Vocabulary

To build vocabulary, instead of most frequent, all words were used from Glove6B together with $< PAD >$ and $< UNK >$ (400000 words). For $< PAD >$ all zeros vector, for $< UNK >$ average vector were used. For the vocabulary ids were given based on occurrence order, not frequency, and stored in two separate dictionaries: word - key and id - value, and the reverse, id - key and word - value, to accelerate word2id and id2word conversions. The same is done for tags, which were predefined. As long as vocabulary is ready, each sentence is converted into vectors with words/tags replaced by corresponding ids. 26075 out of 29279 (89%) words were found in Glove.

3 Model

I started with a simple model that consists of word level embedding layer, LSTM and FC layers followed by soft-max activation, and experimented by tuning parameters. Further replaced single LSTM layer with biLSTM and dropout layers.

For embedding layer, weight matrix was build from pre-trained Glove6B. 50 and 300 dimensional versions were tested as embedding sizes.

Embedding layer is followed by LSTM layer with h hidden size to learn the model. Further Fully Connected layer of size $< h, 14 >$ and soft-max layer were used to get classification. As a loss function, I have used negative log likelihood loss.

For biLSTM, that goes through a sentence in both forward and backward directions, hence, has 2 h hidden size. So FC layer's size is modified to $< 2h, 14 >$

In case of two stacked LSTM layers, two LSTM layers with identical shapes connected on after another. Same goes for biLSTM.

Dropout layer is used a separate layer that comes before FC, that makes node zero with a de-

	F1 score
LSTM + glove 50	0.3987
2 LSTM + glove 50	0.3982
2 LSTM + glove 50 + dropout 0.2	0.4071
biLSTM + glove 50 + dropout 0.2	0.4758
2 biLSTM + glove 50 + dropout 0.4	0.4851
2 biLSTM + glove 300 + dropout 0.4	0.5387

Table 1: Summary of model variations

fined probability. It is useful as regularization as model complexity increases to avoid over-fitting.

4 Experiments and results

The model was trained in batches and sentence size was fixed as max in each batch, but adding $< PAD >$ token to sentences with less tokens.

Hyper-parameters tuned for this task are batch size, hidden layers, number of layers, learning rate and optimizer. Parameter for regularization are dropout rate, weight decay and early stopping patience. Trained over 50 epochs and implemented early stopping on loss with patience 3.

As a starting baseline I have fixed word embedding of size 50 and trained single LSTM with varying hyper parameters. This revealed that trained on smaller batch size it results in better performance. The greatest improvements were coming with increase of hidden size, but it was also slowing down training time considerably.

Optimizer I have tested are Adam, SGD and RMSProp. SGD showed decrease in a loss function, but during the testing F1 was equal to 0, while RMSProp slightly outperformed Adam in certain settings with a cost of an increase training time. Learning rate was varied between 1e-2 and 1e-5, and it was found that most optimal value is 1e-4.

For single LSTM the best performance obtained is F1 = 0.3987 with the following parameters: embedding size 50, batch size 32, hidden size 128, learning rate 1e-4, weight decay 1e-6, Adam optimizer. With patience 3, it converged in 17 epochs.

Two stacked LSTM's have resulted in almost the same performance (F1 = 0.3982), but adding dropout of 0.2 to it led to a slight increase up to F1 = 0.4071.

As anticipated, replacing LSTM with one biLSTM has significantly improved the score (F1 = 0.4758) in 6 epochs, and hyper parameters changed are: batch size 16, hidden size 512, learning rate 1e-3.

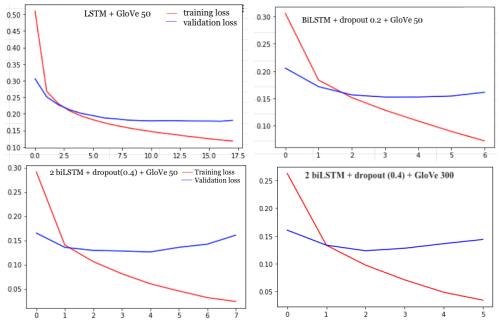

Figure 2: loss functions

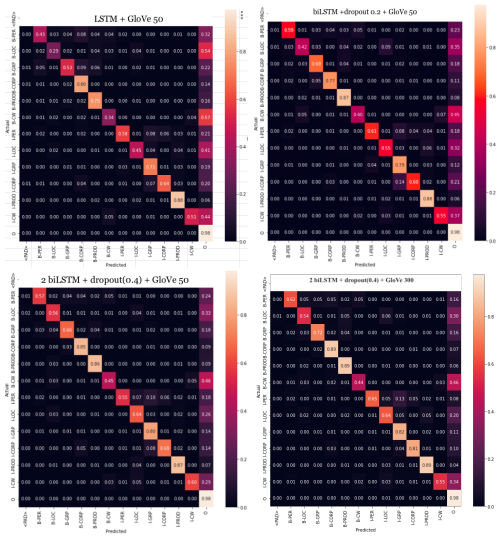

Figure 3: confusion matrices

Adding another biLSTM increased complxity of the model, hence dropout was increased to 0.4 and reduced hidden size (256) have resulted the F1 = 0.4851.

Further improvement has been achieved by increasing embedding size of Glove from 50d to 300d. For 2 biLSTM the score increased 0.5304 and 0.5387 with hidden size 256 and 512 respectively.

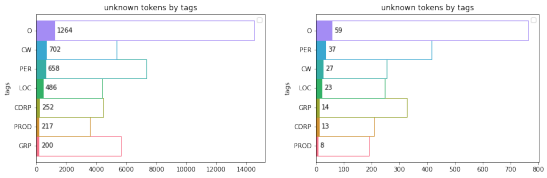

Figure 4: Unknown token distribution by tags for train and test sets. Not filled bars are total appearance of a tag and filled ones are with token unknown in vocabulary

5 Conclusion

Overall, the performance improvement has been shown over larger network and pre-trained embedding size. The highest F1 score is obtained by using BiLSTM and dropout and 300d embedding size. As it can be noticed from confusion matrices 3, considerably worse scored tags are creative work (CW), location (LOC) and persona(PER). One of the reason for that could be that most of the unknown tokens are coming from these tags as it can be illustrated in the charts for both train and dev sets 4. This could be addressed by using richer pre-trained word embedding. As it comes with computational cost, one way would be to use character-wise embedding as well. Other techniques that might lead to a better performance are using N-grams and Conditional Random Forest (CRF) to take into account neighboring tokens.