Word-in-Context Disambiguation

Anonymous ACL-IJCNLP submission

1 Introduction

Named Entity Recognition (NER), is the task of finding the entity of the word in a structured sentence. An entity is the word or series of the word that refers to the same category. We have 13 entity introduced in the dataset. And those are; B-PER, B-LOC, B-GRP, B-CORP, B-PROD, B-CW, I-PER, I-LOC, I-GRP, I-CORP, I-PROD, I-CW and O. The task is to tag each word in a given sentence with an appropriate tag such as Person, Location, etc. Identifying whether the entity belong location or person category, help us to exploit the context of the written text and give us a chance to associate part of the text with each other.

2 Preprocessing

Firstly, I iterated over the dataset to construct a vocabulary of the data with some utility functions with necessary indexing and the reverse indexing of the corpus which I than trim the embedding vectors to fit my vocabulary and than include my vocabulary with two extra tokens $< UNK >$ and $< PAD >$ which are going to be calculating with the mean of the vocabulary embedding and random embedding, respectively.

3 Model Architecture

When we consider word representation which we should provide to our model, there are several options: one of them is the one-hot encoding of the vocabulary space. Suppose we have n sized vocabulary $[w_0, w_1, ...w_n]$, to represent w_t. We should store the whole vocabulary for every word in the data. The issue with that approach is that vector space is too sparse for our model to learn the correlation between the words and also memory requirements are too high. Another option is to use state-of-the-art word embeddings, where we have a fixed number of dimensions which we have a continuous value for each of them to represent the word in the vocabulary. In my implementation, ”GloVe (Global Vectors for Word Representation)”, one of the most used word embeddings models in natural processing tasks, is used. More specifically, 50-dimensional GloVe embeddings of 400.000 words are downloaded as a word dictionary. Than after every sentence in a batch is padded to the appropriate length which is to the max sentence length. They fed to a bidirectional lstm layer which than directed to linear layer for calculating the emissions of the each label. I leveraged the word embeddings as the feature at the sentence level. Then catogorical cross entropy function figures out the loss. I used ADAM optimizer which is one of the fastest one to converge to the result.

4 Experiment

I used bidirectional long term short memory (biLSTM) with the padded input and batches. I first went with 50 dimensional word embedding with 50 epoch which I had 0.33 F1 score but model waas not converged than I increase the epoch lenght to 100 that makes model to converge but improvement was only up to 0.44 F1, than I tried with 100 dimensional embedding which increased my F1 up to 0.50. I tried with different models, such as; RNN and GRU. The model was overfitting to the data quickly and couldn't generalize the data even though I tried dropout which is known to be a top-notch technique to overcome overfitting. Then, I switch to bilstm which is proven to work good with NER tasks.

5 Evaluation

Below, the evaluation for the three model are shown. Training and validation accuracy and loss graphs are created using TensorBoard. The trainings are finished after the validation loss is converged.

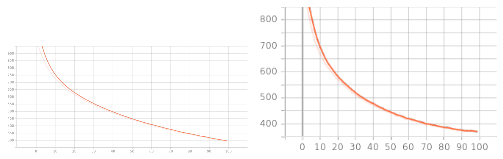

Figure 1: Train and Validation loss for Model 1 with 100 epoch and 50 embedding dimension

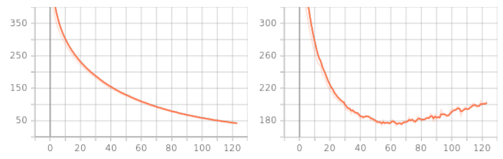

Figure 2: Train and Validation loss for Model 2 witn 100 epoch and 100 embedding dimension

6 Conclusion

As I examine the dataset, I noticed that 997 of the words doesn't appear in the training set among 12751 words which indicates that close to $\%8$ of the words are not on the training data. This might prevents the model from learning the patterns. To overcome that problem, I thought a complex model would work but surprisingly the best performer was the simplest one. Also increasing the dimension of the embeddings helped alot to get a better F1 score for my case. If I invested more time into tuning hyper parameter, I think I would get accuracy close to state of art one.