HW3 Report – NLP

Yonathan Mizrahi – 209948728 / Raphael Bellahsen – 931188684

Table of Contents

Training

Dependency tree parsing is a method of analyzing the grammatical structure of a sentence and representing it in the form of a tree. Each word in the sentence is a node, and the relationships between the words are represented as edges. Dependency tree parsing is used in natural language processing to understand the structure and meaning of a sentence, and it has many applications, including machine translation and text summarization.

For this homework, we decided to use the same model for the test and the competition part. We used the paper written by Kiperwasser and Goldberg to implement our model.

First, for the data preprocessing, we used the provided code in the tutorial (collab part) with few modifications1. Then, we implemented our LSTM model following the architecture described in the paper (bi-LSTM + MLP) and we used the same hyper-parameters. We decided to use a to compute the sentences one by one (eq. batch size equal to 1). The forward method processes the input word and POS tag indices, concatenates the resulting embeddings, passes them through the LSTM to get hidden representations, constructs a tensor of features by concatenating pairs of hidden representations, and passes this tensor through the MLP to get a score tensor for each possible edge in the parse tree. For the loss function, we implemented the NNLoss discussed in the tutorial to return the loss for each word in the sentence. Finally, the UAS accuracy was calculated for each word in the sentence.

Then some changes have been done to the model to improve our UAS accuracy but mainly to reduce the overfitting of our model. We will discuss those changes in this report's "competition" part.

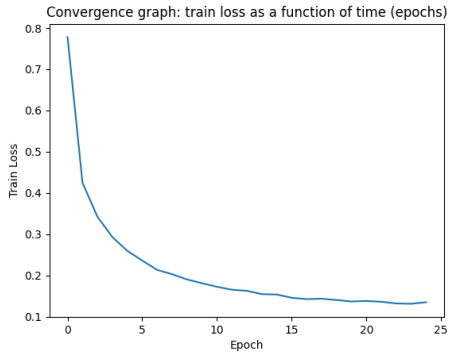
The final[2] hyper-parameters are:

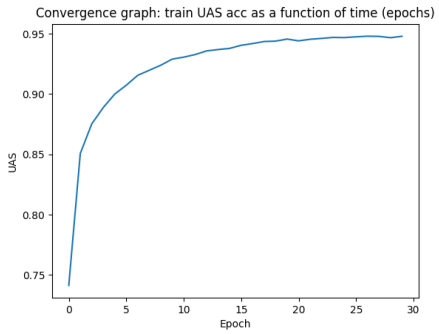
[1] For example, since the module torchtext.Vocab isn't relevant yet, we removed this part from the tutorial and we assigned manually to each word/pos an key index.

[2] After the improvements explained in the comp part

- Words Embedding dimension: 300
- Pos Embedding dimension: 25
- Dropout: 0.5
- Learning rate: 0.001
- LSTM hidden dimension: 125
- MLP hidden dimension: 4*125 ->100 -> 1

Then after doing the relevant changes for the competition part, we ran the model for 25 epochs (approximately 1.5 hours). The obtained results were the following:





During the training, the loss is decreasing and the UAS score is increasing, as expected for both. Last UAS score on the training dataset was 94.79 %. The last loss score was 12.23%.

```
Train_Loss : 0.12236692022448314 Train_UAC : 0.9479448795318604
```

Inference

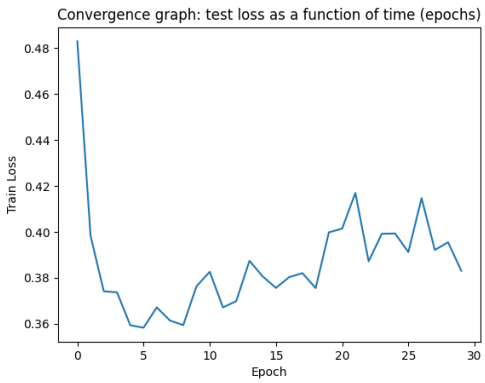
Since the inference is done by the Chu-Liu-Edmonds algorithm, we didn't change the provided code. The inference part takes less than 1 minute in the evaluating part (for the comp file).

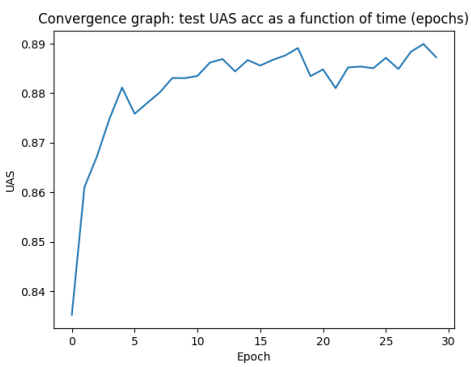
Test

Accuracy on all test file after a train of 30 epochs: 89.93%

```
Training terminated .......
Evaluating Started (over all the test file)
Number of Test Tagged Sentences  1000
Evaluating Ended
Accuracy over all the test set: 0.8993216753005981
```

Test plots:





On one hand, we can see that the loss decreases, then increases starting at epoch 5. However, the UAS continues to increase even after epoch 5. This is likely due to overfitting. Nevertheless, our tentative (explained in the comp part) to reduce as maximum as possible the overfitting seems to work correctly. Indeed, before our changes, the loss was increasing faster and was achieving higher values. Also, the test file's accuracy dropped after a few epochs. Now, we're thrilled that even if our loss started to increase again, the slope is very low and the max delta is 0.04.

Competition

As previously said before, we made some changes to get the best model possible for the comp part. Those changes were composed of some modifications on the hyper-parameters like the word dropout from 0.25 to 0.5 or the learning rate from 0.1 to 0.001 (those changes were tested manually). We also decided to use a famous pre-trained glove model *glove.6B.300d.txt*[3](and for that reason the word emb dim changed from 100 to 300). Finally, our last major change was to add an additional layer of bi-LSTM to the model. In the paper "Deep Biaffine Attention for Neural Dependency Parsing" (https://arxiv.org/abs/1611.01734), the authors use a double layer of bi-LSTM as part of their model for dependency parsing. The paper states: "We find that using three or four layers gets significantly better performance than two layers, and increasing the LSTM sizes from 200 to 300 or 400 dimensions likewise significantly improves performance". However, since the accuracy wasn't the issue for us but the overfitting, we decided to add only one layer since we are assuming that it will be more than enough.

It appears that these modifications had a significant effect on our model. However, we may still see different results on the competition files. This could be due to a number of factors, such as the competition files being very different from the train/test files and containing many unknown words and contexts. Another issue is that our models tend to overfit the data, and despite the fact we saw on the test file that the problem seems to be resolved, it is still very difficult to predict how well we will perform on the competition files.

We're estimating that our UAS score on the comp file will be around 85%.

Work repartition

In this homework, Raphael did the data pre-processing and the glove embedding. Yonathan did the LSTM model. The rest of the code and the hyper-parameters were done by both of us.

[3] Since the model is too heavy to be uploaded to Moodle, Tomer accepted in the forum to provide a direct link to download the model: https://nlp.stanford.edu/projects/glove. The link is also available in the submission under the file "link_to_glove.txt"