Homework 1 NLP, May 2023

Teglia Simone, 1836794

1 Data Analysis

Before I started writing the code of the model I decided to take a look at the training data in order to check for some biases or some patterns. As expected I found out that the tag 'O' is the most frequent one, labelling 90% of the words in the training sentences (Figure 1). Then I tried to find some hidden pattern in the training data: I built a co-occurrence matrix (Figure 2) to check if some tag appears more frequently with some other tag in the same sentence but apparently there is no correlation. Manually analyzing the training set I have noticed that some words can be labelled in two different ways in different sentences but I think this is normal in most datasets.

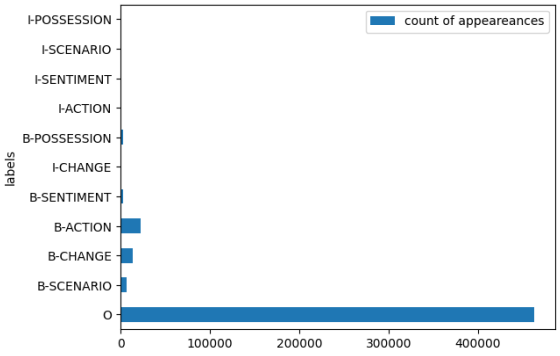

Figure 1: Count of tags appearances in the training set

2 Setup

Since we're working with a relatively small dataset I decided to use a pre-trained word embedding in order to start from a better point and help my architecture learn the task. I've used the famous Word2Vec *GoogleNews-vectors-negative300*. The class 'MyDataset' used during the training phase contains all the boilerplate code that load the jsonl files and create the vocabulary necessary to encode and decode the sentences.

In order to fulfill the homework task I've decided

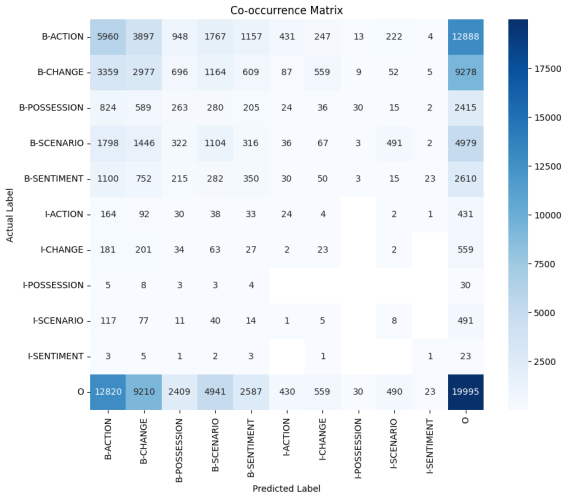

Figure 2: Co occurence matrix of the tags present in each sentence. No exploitable pattern seems to emerge but it was worth trying

to use a Bidirectional-LSTM architecture due to the fact that it can obtain informations about the input sentence forward and backwards. I've chosen to use two layers in order to improve the capabilities of the network. The last classification layer is a Linear Layer. My architecture takes in input one sentence at a time because I obtained much better results in this way rather than when using a batch. This approach have a major downside though: easier overfitting. I've taken this into account and adjusted my hyperparameters to minimize the overfitting as much as possible, as explanined in the experiments section.

The loss function that I've chosen is the CrossEntropyLoss, because it turned out to be better with respect to the NLLLoss, and the optimizer that I've chosen is Adam because of its adaptive learning rate and because it's like a better version of SGD.

3 Experiments

I've tried to keep the architecture as simple as possible so that I could have more time to play with the

hyperparameters. As already mentioned overfitting is very easy, because the model is seeing one sentence at a time but also because the training set is quite small. During a lot of trainings I saw that the f1 score or the accuracy did some big leaps in the first epochs: this means that the model can move towards the correct hyperspace even if the training has just started.

In order to prevent this I've found that the best hidden dimension of my LSTM was 350. Fifty more than the embedding layer seemed to leave the correct amount of *freedom* to the architecture. Another straightforward way to decrease the overfitting is adding dropout layers. I've added a dropout layer after the embedding layer, after the LSTM layer and after the linear layer so that the model had more difficulty to learn the training set. Each dropout ended up having 0.2 as final value. I tried many different values for the dropouts layers and I think that maybe it would be possible to achieve a bit better results by pushing the dropouts values a little higher but more epochs would be needed so I decided to stick with 0.2 for efficiency reasons.

Apparently neither initializing a random hidden state helped the lstm learning: i tried to play with some parameters but none had particular benefit. I also tried to give the CrossEntropyLoss a weight list in order to try to give less weight to the 'O' tag but also this solution didn't came with good results.

Lastly I ran some tests also with more LSTM layers but eventually 2 layers were the correct amount because even though the loss kept decreasing the f1 score was not increasing and "clipped" around 0.66, so I deduced that the architecture was becoming too large for the traininig dataset. This intuition was confirmed by many training loops: to obtain some good results in fact I needed to keep the neural network dimension as low as possible because as soon I gave the neural network too much power it overfitted the training data.

In the end, in order to select the best model during the training phase I performed a test on the validation set after each epoch and selected the model that had an higher macro f1 score on the validation set. The number of epochs necessary to obtain the best model with my setup was 12.

4 Results

In the end my complete architecture consists of a two layer Bi-LSTM connected to an Embedding layer and a Linear layer. The embedding dimension is 300 and the hidden dimension is 350. Of course the output dimension is 11 as the number of tags.

With this setup I was able to achieve an f1-score of 0.72 on the test set and a micro precision of 0.949. Considering that the baseline of the micro precision it's 0.9, because that's what we can obtain by tagging each word with the 'O' tag, it doesn't seem a great result but we need to take into account that the remaining 10% of the tag distribution is not splitted evenly and that some tags appear less than 20 times in the training set, making the correct tagging of them really difficult for an architecture like this.

As we can see in Figure 4 I did not obtain an f1-score of 0.72 on the validation, this means that the model for some reasons performed better on the test set than on the validation set. Using the tester inside the homework folder and connecting to your server I obtained an f1 score of 0.7038 and accuracy of 0.9491. Analyzing the confusion matrix of the predicted tags (Figure 3) it's possible to see that the diagonal has the darkest squares, meaning that the model most of the time correctly tag the words but there is a relevant number of tags that are incorrectly predicted as 'O', being it the most frequent class. Most probably this last set of wrong predictions is what bring the f1-score down to 0.7

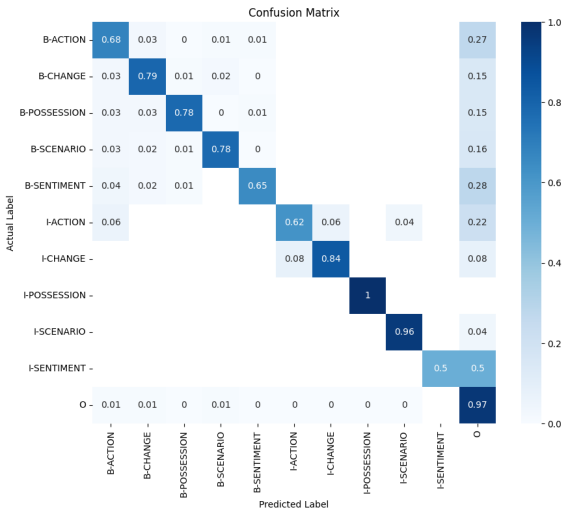

Figure 3: Confusion Matrix of predicted tags during the test phase

5 Conclusion and future work

I think this is a good result considering that the training is quite small and the model too. I believe that the most straightforward way to improve the

performances of the model would be to add CRF layer: specifying the correct and incorrect transitions between tags would really help the model, for example, to not predict a 'I-' tag of a class after a 'B-' tag of another class. Another thing that I would have loved to try, and I think that could improve a bit the performances of the model, is to perform a final fine tuning after the training phase on some meaningful data. Since the dataset is so unbalanced I would have selected the sentences containing more tags, and so the most informative sentences, and performed a final fine tuning loop, just to reinforce the model's knowledge of the tags that appeared less during the training.

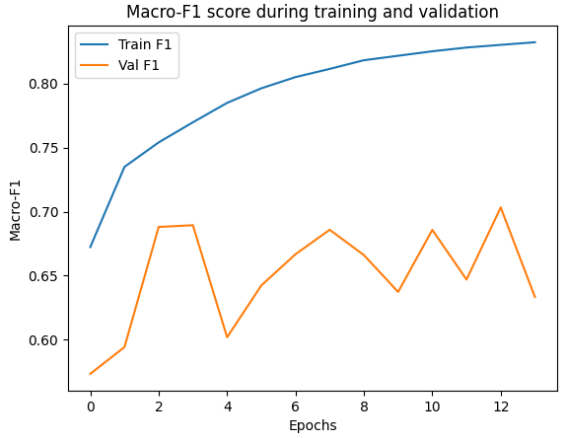

Figure 4: Macro f1 score during training and validation loops

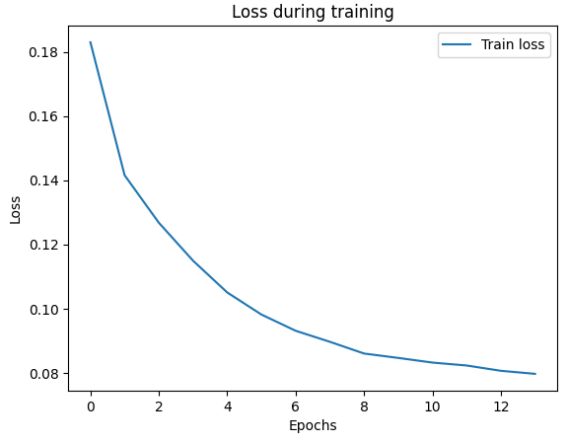

Figure 5: The loss was correctly decreasing during the training