NLP HW1: Word-in-Context Disambiguation (WiC)

Luigi Sigillo
sigillo.1761017@studenti.uniroma1.it

1 Introduction

Word in context disambiguation (WiC) concerns the address of disambiguation of polysemous words, without relying on a fixed inventory of word senses. So our task, given two sentences, is to determine whether the target words, present in both sentences, have the same meaning.
For example given the sentences:

- "The cat eats the **mouse**"
- "Use the **mouse** to click on the button"

the target word **mouse** is used with a different meaning, so our model have to return the False label.
I developed a RNN model, using Bi-LSTM, to face this problem, the results are not comparable with the state of the art that are higher, like this work that uses BERT (Huang et al., 2019), however we are not allowed to use Transformer models, so I expected these differences in the results.

2 Preprocessing

We have a dataset whose main component are: two sentences, the target word and a label that indicates if the words have the same meaning.
The datasets are balanced in the sense that we have the same number of positive and negative samples. This for both training set and dev set, the latter is used for evaluation since we do not have the test set. I have adopted some of NLP best practices: a lemmatisation of the target words, and removal of stop-words and punctuation. Furthermore I decided to substitute every number with the word "number".
So for example, the sentence:

- "The tragedy of 11 September 2001, which we commemorated just three days ago, has so numbed our minds."

will be preprocessed in this way:

- "tragedy *number* September *number* **commemorate** three days ago numbed minds"

I decide to concatenate the two sentences in one and separate them with the token "SEP".

2.1 Pretrained embeddings

In order to vectorize the words of a sentence I have used the pretrained GloVe (Pennington et al., 2014) word embeddings. Those pretrained embeddings are useful because they have been trained on a larger corpus and are way more precise with respect to an embedding layer that we could have trained from scratch. GloVE embeddings are provided with different sizes, I have decided to use the smaller one with a length per vector of 50, because I have not noticed improvements of the model with respect to the one with a length of 300 that is the maximum size.
I introduced the `UNK` token to map OOV (Out of Vocabulary) words, so that are not present in GloVE and added also the previously mentioned `SEP` token. I assigned random vectors to them, later we will see how this can affects our model.

3 Model architecture

I have tried two different approaches, the first one is a word-level, that will use a simple multi layer perceptron to classify our sentences, the second one is more powerful since it use a recurrent neural network model, for instance a Bi-LSTM.

3.1 Base Model

I have decided to use the pretrained embeddings also in this first approach. I tried two different methods to put together the words of a sentence. I first performed a sum of the resultant matrix of embeddings of a sentence, but than the the best

method so far seems to be the mean instead of the sum, so I used it. The network has two linear layers fully connected and uses the binary cross entropy as loss function. The activation function used is a ReLU, the output activation function is a sigmoid. Since the performance are worst with respect to the RNN (figure 3) I decided to focus and to improve more the second model instead of this, but it was a useful point of start, since the preprocessing ideas are the same.

3.2 Bidirectional LSTM

For this approach I used Bi-LSTMs (Graves and Schmidhuber, 2005), that are two LSTMs stacked together where one takes the input in a forward direction, while the other in a backwards direction, this is useful to have more information on the context of the sentences. Since a RNN outputs one vector for each word, instead of using the last hidden representation, I decided to extract the two hidden states (h_{t1} and h_{t2}) of the target words of the sentences, performing a multiplication of the two vectors and taking the result as input for my multi layer perceptron classifier. This approach is called "many-to-one" and is used in the classification of text sequences using RNNs. I have tried also performing a difference instead of the multiplication, but the results are worst, probably because the numbers are close to 0 and 1 so multiplying we are, in some sense, rounding them. To reach consistency, I have padded the sentences with zeros at the end, in this way all the input vectors will be of the same length, because there will be sentences with different lengths in a batch. Indeed the network uses the embedding layer provided by torch with the GloVe embeddings and two recurrent layers for the Bi-LSTM and two fully connected layer to classify using the Leaky ReLU as activation function and the BCE as loss function. The architecture is showed in figure 1.

4 Experiments

An important part in the success of a neural network is the tuning of hyperparameters and trying to find the ones that best suits for our network. In this case I have noticed that the learning rate should be 0.0001, with values just a little bit greater or lower than this the network not train at all.
The optimizer that I used is Adam, I have tried others like SGD, that performs well in the first approach, but they are not powerful as Adam is with this RNN approach.
Another hyperparameter that I have tuned is the number of hidden layers, I used a small number of hidden layers, I noticed that a size of 82 performs well and increasing the size does not bring benefits. I also tuned the dropout probability of the LSTM, I have chosen a value of 0.15, even if I saw that in this paper (Kågebäck and Salomonsson, 2016), where they use a Bi-LSTM for a similar task, with a value of 0.5 the network reached better results.
I then used gradient clipping in order to avoid exploding gradients in LSTMs and noticed that a value of 4 results in a improvement of 1-2% of the network. In general the network overfits approximately after ten epochs, this will be clearly visible in the figure 2. This is a behaviour that I tried to control using early stopping and also some kind of decay of the learning rate after a certain epoch, but this neither prevent this overfitting.

5 Results

To measure the performances of this WiC system I used accuracy and also F1-score that is a weighted average of the precision and recall.
The results in term of accuracy are in the range of $65 - 68\%$ depending on the run. In fact there are known non-determinism[1] of torch RNNs with some CUDA versions. Furthermore, as I mentioned earlier, the results may vary based on the values of the random vectors associated to the UNK and SEP tokens, so I try to adjust this with a fixed random state value of torch. I decided to compare different runs to justify the choice of the hyperparameters and the best result of the second approach with respect to first, these are shown in table 2. I also wanted to show how different preprocessing can affect the results of the model as shown in table 1 and the worst performance using only LSTM with respect to the bidirectional one, this is clearly visible in the figure 4.

6 Conclusion

As anticipated in the introduction, this network does not achieve SOTA performances, and even after many tuning of hyperparameters the results are stuck under the soil of 70% accuracy. Surely the usage of pre-trained embeddings and an adequate preprocessing improved the network but, as showed in the comparative analyses below, the improvement is only slight.

[1] PyTorch documentation

References

Alex Graves and Jürgen Schmidhuber. 2005. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5):602–610. IJCNN 2005.

Luyao Huang, Chi Sun, Xipeng Qiu, and Xuanjing Huang. 2019. Glossbert: BERT for word sense disambiguation with gloss knowledge. *CoRR*, abs/1908.07245.

Mikael Kågebäck and Hans Salomonsson. 2016. Word sense disambiguation using a bidirectional LSTM. In *Proceedings of the 5th Workshop on Cognitive Aspects of the Lexicon (CogALex - V)*, pages 51–56, Osaka, Japan. The COLING 2016 Organizing Committee.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Tables

Preprocessing	Accuracy	Loss	F1
Best configuration	68.1	0.603	67.6
Without lemmatization	65.9	0.614	66.5
Lower case	65.8	0.613	65.9
Without "SEP"	61.1	0.661	61.1
With stopwords	57.3	0.631	64.4

Table 1: Metrics to justify the preprocessing choices, calculated on the dev dataset using the best Bi-LSTM model. Best configuration is the one discussed on section 2, so with lemmatization, stopwords removal and with the separator.

Hyperparameters	Accuracy	Loss	F1
Best configuration	68.1	0.603	67.6
Higher hidden size	66.4	0.599	62.4
No dropout	66.2	0.607	66.1
1 LSTM layer	65.6	0.627	63.8
No clipping gradients	66.3	0.612	67.7
Small batch size	65.7	0.621	65.3

Table 2: Metrics to see the hyperparameters tuning. Best configuration is the one discussed on section 4, so with the Bi-LSTM, learning rate = 0.0001, batch size = 40, max norm of gradient clipping = 4, 2 LSTM layers and 98 hidden layers.

Figures

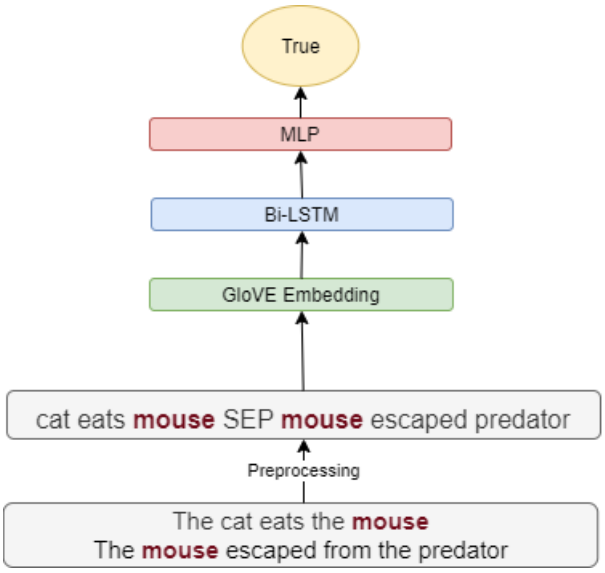

Figure 1: Overview of the network architecture

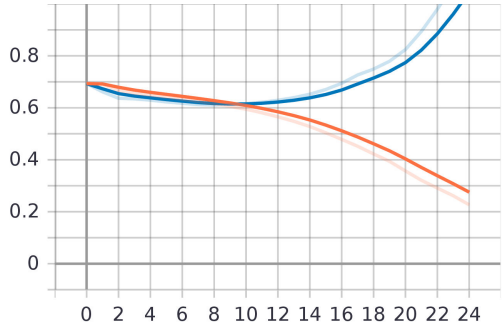

Figure 2: Comparison of losses between training in orange and the evaluation in blue. We see that after ten epochs the RNN overfits.

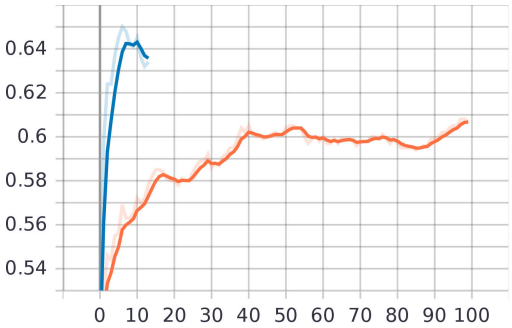

Figure 3: Evaluation accuracy, in orange the first approach, in blue the RNN one. Even after 100 epochs the first approach is not reaching the accuracy of the RNN one, hence my decision to try to improve the RNN.

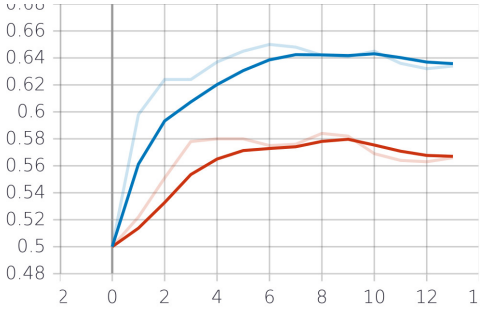

Figure 4: Evaluation accuracy, in red the normal LSTM, in blue the Bi-LSTM one. We see that with the Bi-LSTM we gain about 6% in accuracy, and this probably because of the more contextuality that this network provides