Word-in-Context Disambiguation
Natural Language Processing course 2021 – Homework 1

Andrea Gasparini
Sapienza University of Rome
gasparini.1813486@studenti.uniroma1.it

1 Introduction

Word-in-Context Disambiguation is the task of addressing the disambiguation of polysemous words, without relying on a fixed inventory of word senses. In our case, given two sentences, the task was to determine whether the indicated target words had the same meaning or not. In this report I am going to describe the approaches I used to provide a solution based on Neural Networks.

2 Preprocessing and Word embeddings

In order to train a Neural Network with textual data it is necessary to create an "embedding layer" that associates each word to a vector of numeric values. This layer has been implemented with a pre-trained set of word vectors like GloVe[1] (Pennington et al., 2014), so that each word already has a rich representation obtained by training on a huge corpus.

After some analysis on the provided data, the changes that led to the minimum amount of unkown tokens (i.e. words for which we do not have an embedding) have been the replacement of separation characters like hyphens and underscores with an actual space (to handle some compound words) and the removal of special characters and punctuation.

The handling of the remaining OOV (Out Of Vocabulary) words, that are less than 5000, has been left to a randomly initialized vector with the same size of the other embeddings.

3 Models

In this section I am going to give an overview on the two different architecture approaches I used to deal with this task, also showing the experiments and variants I made, highlighting the best ones.

[1]GloVe embeddings have been downloaded from here

3.1 Word-level model

The first implemented model is based on a word-level approach in which the classification is made by taking in consideration which words are contained in the sentences without keeping track of the overall context. The model architecture is based on a 2-layer MLP (Multilayer Perceptron), where the first fully connected layer takes the econded sample and outputs a new tensor, on which is applied a ReLU activation function. The second, and last, fully connected layer takes the output of the previous one and gives a single classification value through a sigmoid activation function.

3.1.1 Variants and experiments

Embeddings average The first way of econding the samples was to concatenate the two sentence embeddings, separated by a special word vector to indicate where the embeddings of the first sentence end and the ones of the second start. In order to reduce the dimensionality of the network input, the concatenation is replaced by a weighted average of the embeddings, where the only embeddings with a weight bigger than 1 are the ones representing the target words.

Individual average concatenation At the cost of a larger input vector, I obtained a better representation of the samples averaging individually the two sentences and then concatenating them in a single vector of two times the embedding size dimension, so that the model had more information about the sentences separately. In fact this change reflected well on the overall performance of the model.

Stop words removal A small set of stop words has been introduced in order to ignore words of common use that does not add much meaning to the sentences, and as we could expect this led to an improvement in the performance on both the previous variants as highlighted in Table 1.

3.2 Sequence encoding model

The second architecture approach makes use of RNNs (Recurrent Neural Networks) in order to exploit sequence level semantics that could not be catch with the first approach. The implementation is based on a Long Short-Term Memory layer (Hochreiter and Schmidhuber, 1997), which starts from the beginning of an input sequence (a sentence in our case) and proceeds forward obtaining at each timestep new information about the context.

The output of the LSTM layer is a vector of hidden states representing the information obtained at each timestep. One of these states can be used to summarize the sequence given to the LSTM and then as input of the following network's part.

The last block of the architecture is composed by 2 fully connected layers, based on the word-level model (subsection 3.1), designed to map the LSTM output to a classification value.

3.2.1 Variants and experiments

Baseline LSTM As baseline implementation of this architecture I used a single LSTM layer that takes the two sentences as different inputs at different times, so that it was possible to have the hidden states directly separated in different outputs. In a previous test I used a single concatenated tensor as LSTM's input but this clearly led to worse performances since the hidden states of the second sentence were affected by the first ones.

Since the hidden states of the last tokens in the two sentences should summarize their previous tokens, a "summary vector" composed by the concatenation of these two has been used as input for the following fully connected layers of the network.

Dropout regularization The baseline LSTM implementation turned out to be highly affected by overfitting, with a training loss always under 0.5 and a validation loss often over 1.0 (Figure 3).

That's why I introduced dropout layers, that have proven to be an effective way to prevent this behaviour, dropping random units and their connections from the network during training (Srivastava et al., 2014). After some tests, the addition of a dropout layer after the embedding one, after the first fully connected and also for the LSTM layer, with a dropout rate of 0.5, led to a significant improvement in both accuracy and F1 (Table 2).

Bidirectional LSTM In order to improve the summary vector obtained from the hidden states of the LSTM I also implemented a variant based on a Bidirectional one. A Bi-LSTM is composed of two LSTMs, one that proceeds from left-to-right (like the normal behaviour) and the second one that goes from right-to-left, so that the output represents better the whole context. The aim was to make the summary vector correspond to the target word's hidden states instead of the ones of the last word. With a single LSTM this would mean to ignore words that come after the target word.

4 Training

In order to tune the hyperparameters and evaluate the results, a training phase for each implemented model variant has been carried out multiple times. The monitored validation metric is the loss (Binary Cross Entropy function) and it has also been used to prevent overfitting, thanks to early stopping. A patience of 3 epochs always turned out to be the best choice; examples of loss histories are in Figure 1 and 2.

Other hyperparameters like, dropout rate, number of layers, type of optimizer and learning rate have been tuned and evaluated with the set of values described in Table 3 and Table 4. Adam optimizer (Kingma and Ba, 2017) turned out to always make training significantly faster than SGD and the improvements in using the latter were marginal if not null; also further effort in tuning momentum or weight decay showed the same behaviour.

5 Results and conclusions

The results, computed on `dev.jsonl` for each model variant, are reported in Table 1 and 2, along with the confusion matrices reported in Figure 5 and 7 that highlight the harder prediction of False samples in the second approach, affecting the overall accuracy of the model.

Even thought the word-level approach was not highly sophisticated it easily got better performances than the sequence encoding one, reaching an accuracy and F1-score of **67%**. The main reason behind this is probably in the difficulty of exploiting sequence-level semantics, and LSTMs based models were not enough to overcome the simpler approach. Contextualized word embeddings and transformer-based models, that were not allowed for the purposes of this homework, would certainly be an improvement to solve this issue, and they have proven to be really effective for this kind of tasks (Hettiarachchi and Ranasinghe, 2021).

References

Hansi Hettiarachchi and Tharindu Ranasinghe. 2021. Transwic at semeval-2021 task 2: Transformer-based multilingual and cross-lingual word-in-context disambiguation. *CoRR*, abs/2104.04632.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9:1735–80.

Diederik P. Kingma and Jimmy Ba. 2017. Adam: A method for stochastic optimization.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958.

A Tables, plots and images

Model	**Accuracy**	**F1-score**
Embeddings average	0.6200	0.6175
+ stop words removal	0.6450	0.6447
Individual avg. concat.	0.6580	0.6537
+ stop words removal	**0.6770**	**0.6770**

Table 1: Metrics for the word-level model and the different variants

Model	**Accuracy**	**F1-score**
Baseline LSTM	0.5810	0.5801
+ dropout	0.6280	0.6276
++ stop words removal	**0.6440**	**0.6391**
Bidirectional LSTM	0.6040	0.5984
+ dropout	0.6140	0.6084

Table 2: Metrics for the sequence encoding model and the different variants

Hyperparameter	**Tested values**
Adam learning rate	{0.01, 0.005, **0.001**, 0.0001}
SGD learning rate	[0.01 – 0.5]
SGD momentum	[0.0 – 0.5]
Hidden size	{100, 200, **n_features // 2**}

Table 3: Some of the hyperparameters tested during the training phases of the word-level models (the best ones are highlighted in bold)

Hyperparameter	**Tested values**
Adam learning rate	{0.01, 0.005, 0.001, **0.0001**}
SGD learning rate	[0.01 – 0.5]
SGD momentum	[0.0 – 0.5]
Embedding dropout	[0.0 – **0.5**]
Fully conn. dropout	[0.0 – **0.5**]
LSTM dropout	[0.0 – **0.5**]
LSTM hidden size	{**100**, 200, 250, 300}

Table 4: Some of the hyperparameters tested during the training phases of the sequence encoding models (the best ones are highlighted in bold)

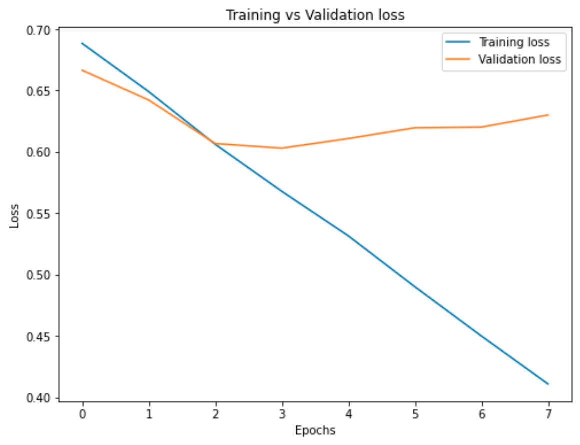

Figure 1: Loss history of the best model trained with the word-level approach (highlighted in Table 1)

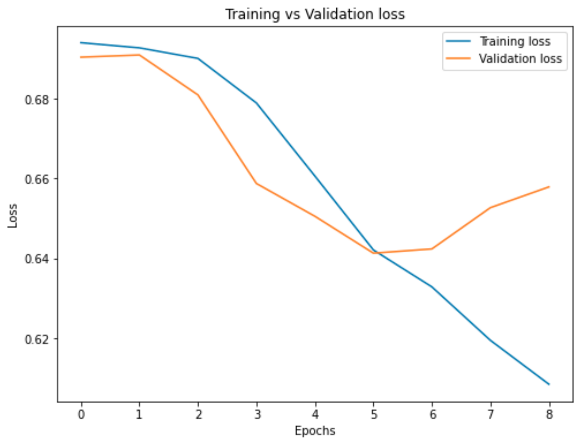

Figure 2: Loss history of the best model trained with the sequence encoding approach (highlighted in Table 2)

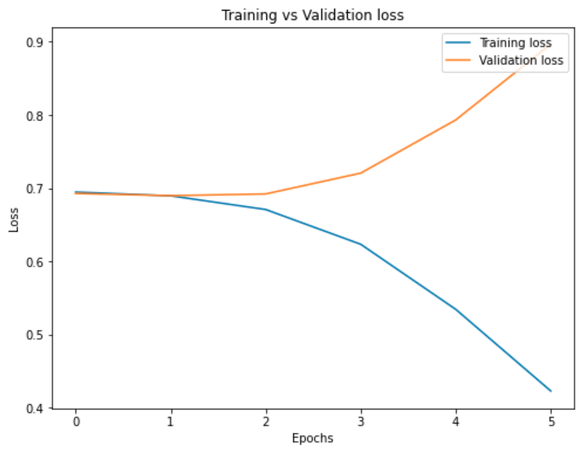

Figure 3: Loss history of the baseline LSTM model, highly affected by overfitting

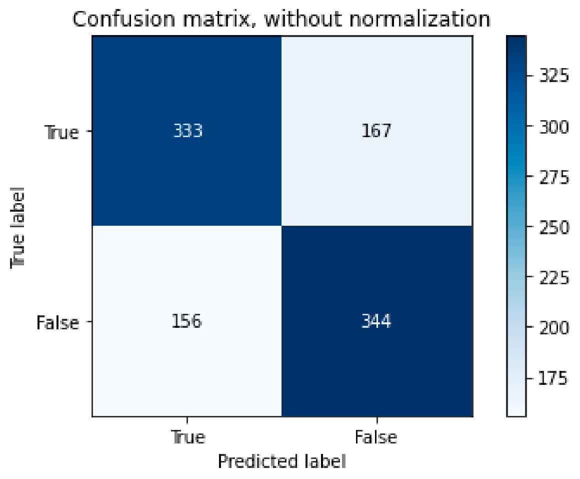

Figure 4: Confusion matrix of the best model trained with the word-level approach (highlighted in Table 1)

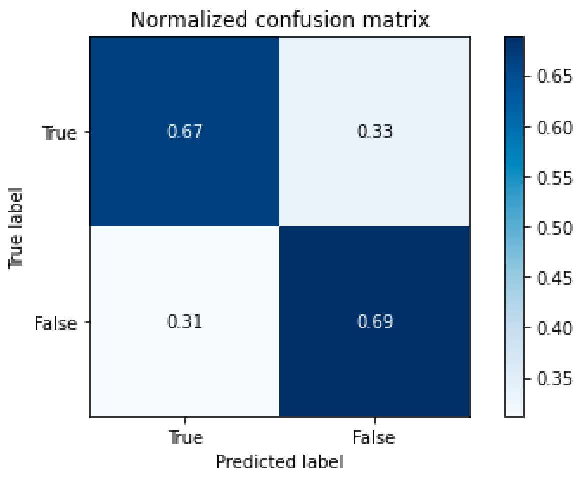

Figure 5: Normalized confusion matrix of the best model trained with the word-level approach (highlighted in Table 1)

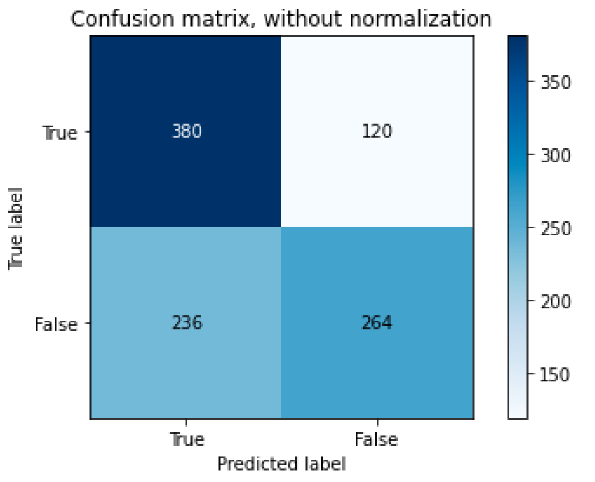

Figure 6: Confusion matrix of the best model trained with the sequence encoding approach (highlighted in Table 2)

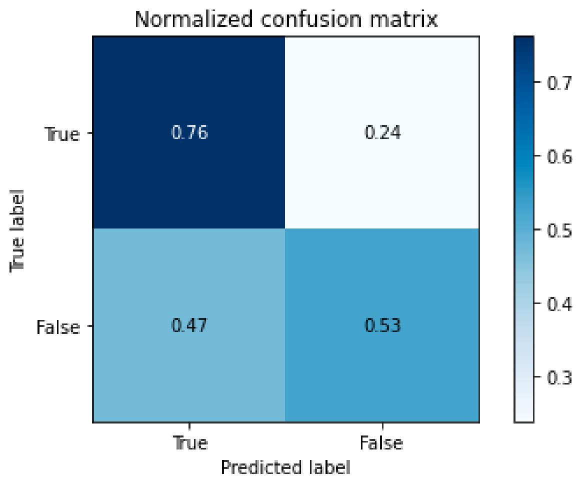

Figure 7: Normalized confusion matrix of the best model trained with the sequence encoding approach (highlighted in Table 2)

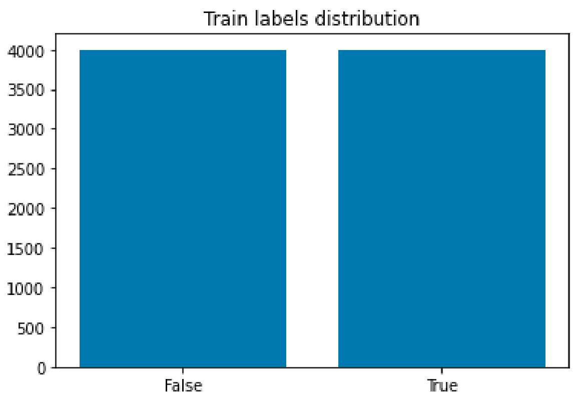

Figure 8: Distribution of the training dataset

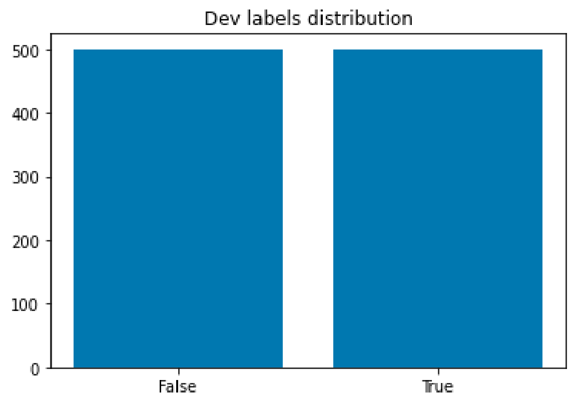

Figure 9: Distribution of the development dataset