Word-in-Context Disambiguation (WiC) - A comparison of three methods

Lorenzo Proietti 1754547
Sapienza University of Rome
proietti.1754547@studenti.uniroma1.it

1 Introduction

In natural language processing, Word-in-Context Disambiguation (WiC) is a task in which the goal is to detect if the meaning of a target word used in a pair of sentences is the same and, in order to accomplish this task, there is no reliance on a fixed inventory of word senses.

2 Methods

In this scenario, I considered three methods. In the first and third methods I simply combine the word embeddings of the sentences in a certain way, the result of this combinition is then given as input to a multilayer perceptron to perform the binary classification task. In the second method, instead, the combination of the two sentences is obtained through sequence encoding of the sentences using a BiLSTM. In all of the methods, early stopping was applied with patience equal to 20 and a learning rate equal to $2 \cdot 10^{-5}$.

2.1 Preprocessing

The preprocessing of a given sentence is the same in all three methods since changing it (for example, applying also lemmatization to words), I got worse results in all of them. In particular, I remove all characters that represent punctuation or that are digits and transform them into their lowercased form.

2.2 Pretrained word embeddings

As pretrained word embeddings in all three methods *GloVe* (Pennington et al., 2014) *840B 300d* vectors were used. With these embeddings I have coverage for 89.53% of the distinct word present in training sentences and for 98.04% of all spawned text (so, also considering word repetitions) after preprocessing. Hence, especially considering the second statistic, there weren't many OOV words problems during the training. To manage the OOV words I tried three approaches:

- Take the average of all available word embeddings.
- Consider a random vector.
- Ignore the OOV words.

Of these three approaches, the last one is the one that gave me the best results in all three methods and therefore is the one I adopted.

2.3 IDF-average word embedding

This is the first method of those mentioned above. Here, given a pair of preprocessed sentences (s_1, s_2), I compute the sentence embedding of them as follows:

$$s_e = \frac{1}{|s|} \sum_{w \in s} IDF_w \cdot w_e \tag{1}$$

Where s_e denotes the sentence embedding of the sentence s, $|s|$ represents the number of words in the preprocessed sentence for which I have in the dictionary the corresponding word embedding (indicated by w_e) and IDF_w is the inverse document (in this case sentence) frequency of the word w in the training sentences:

$$IDF_w := \log \frac{N+1}{N_w+1} \tag{2}$$

Where N is the overall number of sentences in the training set (16000) and N_w is the number of preprocessed sentences in which word w appears. The addition to 1 to both numerator and denominator is considered in order to take into account words for which $N_w = 0$. As follows from the definition, the rarer a word (and therefore important and representative for a given sentence) the higher IDF_w

is. With this way of producing sentence embedding, I can therefore give more importance to the direction spawned by the word embedding vectors corresponding to the most significant words for a given sentence. At this point, given the sentence embeddings of the two input sentences for a single sample s_{e_1} and s_{e_2}, I create the input features for the cascaded MLP with the following concatenation: $(s_{e_1}, s_{e_2}, |s_{e_1} - s_{e_2}|, s_{e_1} * s_{e_2})$. By combining the features of the two sentences in this way, the classifier can have a better representation of the relationship between them. The MLP is formed by a fully connected layer of size 600, where ReLu activation function is used and finally an output layer with sigmoid activation and then binary cross entropy loss function. For this method, I used a batch size of 2048, with which I got best results. A representation of the model architecture is represented in figure 1. As can be seen from the comparison table 1, among all methods that don't use the BiLSTM, this one allowed me to get the best results. Trend of accuracy and loss during training and confusion matrix can be viewed respectively in figures 2, 3 and 4. Two advantages of this method, in addition to a good accuracy compared to the other methods, are the training speed, as it took a few minutes to reach those results and also the fact that its predictions are balanced.

2.4 Sequence encoding

This is the second method of those mentioned above and makes use of a BiLSTM (Hochreiter and Schmidhuber, 1997). In particular, given two preprocessed sentences of an input sample, the word embeddings related to these sentences are obtained through the embedding layer and will form the input sequences for the BiLSTM. The use of the BiLSTM here is due to two facts: unlike the classic simple form of the RNN, it is not affected by the vanishing gradients problem and, as opposed to the classic LSTM, we can capture dependencies in both directions. At this point, all the hidden states in output will be combined through average (I tried also max pooling, but with worse results) and, the vector (1) for that particular sentence, will be concatenated to the previously defined average hidden states vector. Instead of taking the average of the last hidden states, I also tried to take only the hidden state of the target word, but I got worse results. This allows us to have as features for sentence embedding both those derivating from the first method and those obtained thanks to BiLSTM. Once this is done for both input sentences (the weights of the BiLSTM are the same for both of them), their embeddings will be concatenated as in the previous method: $(s_{e_1}, s_{e_2}, |s_{e_1} - s_{e_2}|, s_{e_1} * s_{e_2})$ and this will be the input for the MLP, that is composed by a fully connected layer with 250 units and an output layer with sigmoid activation and binary cross entropy loss function. For this method I used a batch size of 1024, with which I got best results. From the comparison table 1, we can see that this is the method with which I achieved the best performance. Trend of accuracy and loss during training and confusion matrix can be viewed respectively in figures 9, 10 and 11. A drawback of this method is certainly the slowness of the training (about three hours).

3 Extras

3.1 SIF: Smooth Inverse Frequency

This is the third method previously named. It refers to the paper (Arora et al., 2017) and here the average word embedding of a sentence is produced as follows:

$$\tilde{c_s} = \frac{1}{|s|} \sum_{w \in s} \frac{a}{a + p(w)} \cdot w_e \tag{3}$$

Where a is a constant (as recommended in the paper, I used 10^{-3} as value) and $p(w)$ is the prior probability of the word w estimated from the training sentences:

$$p(w) = \frac{c_w + 1}{(\sum_w c_w) + 1} \tag{4}$$

Where c_w denotes the number of occurences of the word w in the training sentences (after preprocessing). The final sentence embedding is obtained by subtracting the projection of $\tilde{c_s}$'s to their first principal component. As can be seen from the comparison table 1, applying this method instead of IDF-average, I got worse results. Metrics of this method can be seen in figures 12, 13 and 14. The architecture of this method is similar to that of the first one (figure 1).

3.2 fastText

The first and third methods were also tested with the *fastText* (Bojanowski et al., 2016; Joulin et al., 2016) pretrained word embeddings (figures 5, 6, 7,15, 16 and 17) but, as can be seen from the comparison table 1, I got worse results.

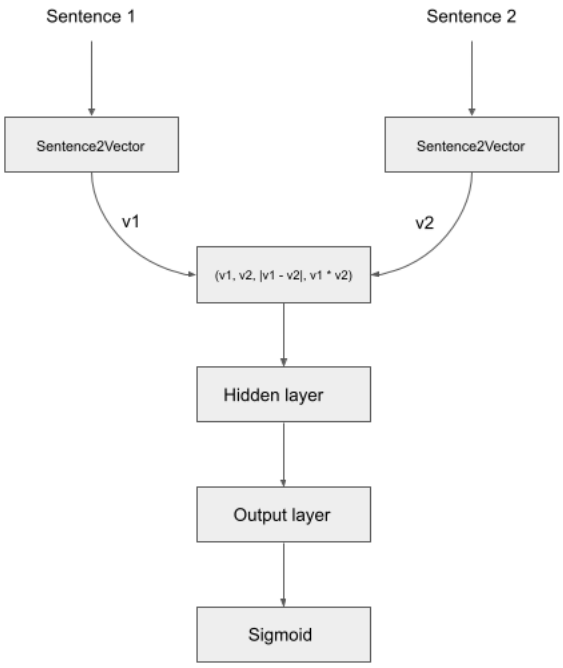

Figure 1: IDF-average and SIF model architecture

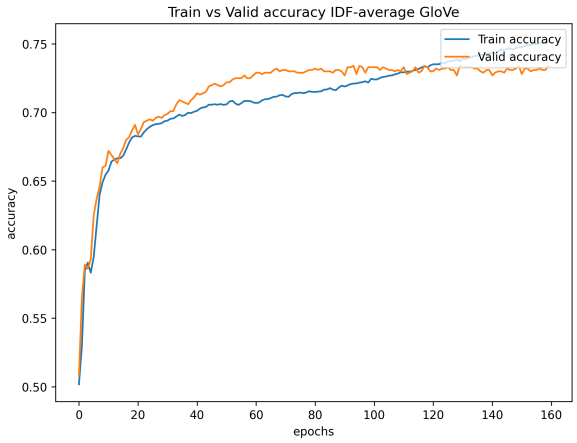

Figure 2: Train and Dev accuracy of the IDF-average model with GloVe embeddings

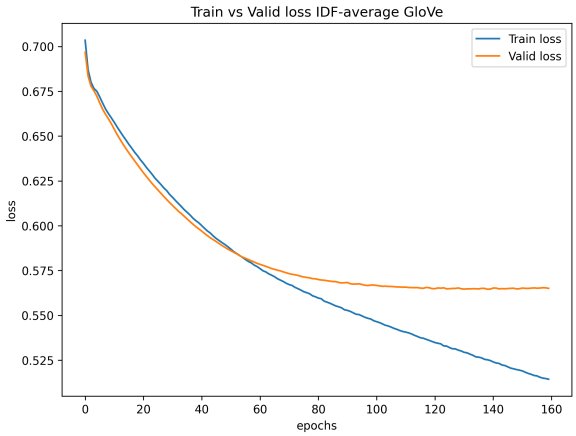

Figure 3: Loss IDF-average model with GloVe embeddings

Method	Precision	Recall	F1-Score	Accuracy
IDF average GloVe	0.73	0.73	0.73	0.73
IDF average fastText	0.66	0.68	0.67	0.66
SIF GloVe	0.70	0.66	0.68	0.69
SIF fastText	0.63	0.63	0.63	0.63
Sequence Encoding GloVe	0.73	0.75	0.74	0.73

Table 1: Comparison.

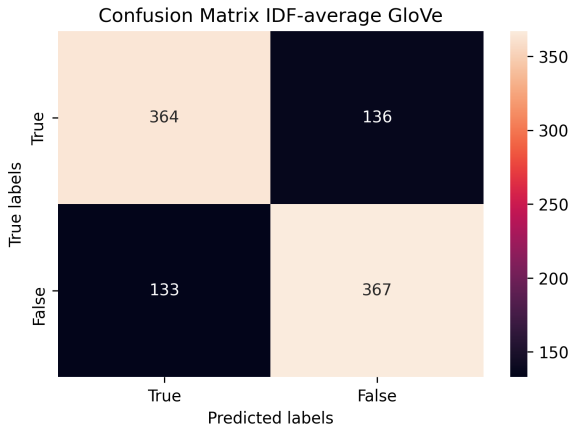

Figure 4: Confusion matrix IDF-average model with GloVe embeddings

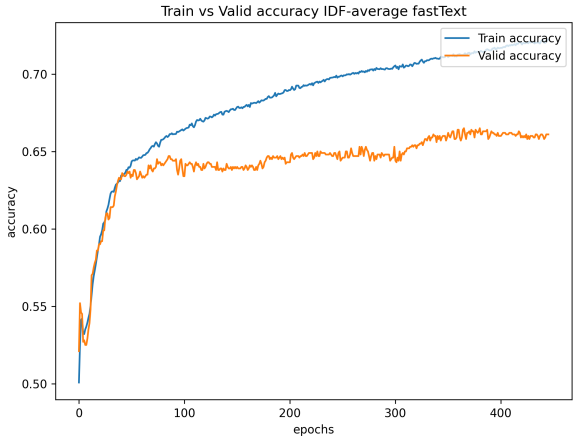

Figure 5: Train and Dev accuracy of the IDF-average model with fastText embeddings

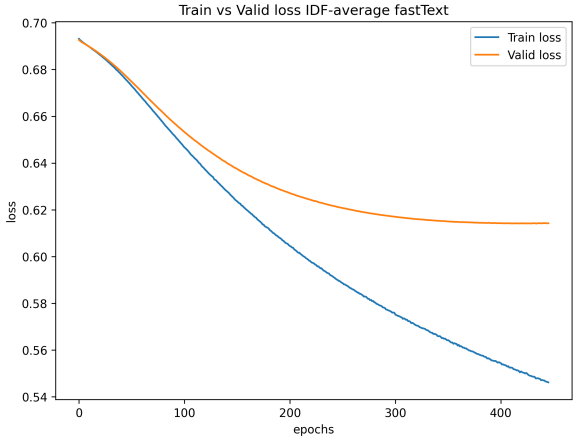

Figure 6: Loss IDF-average model with fastText embeddings

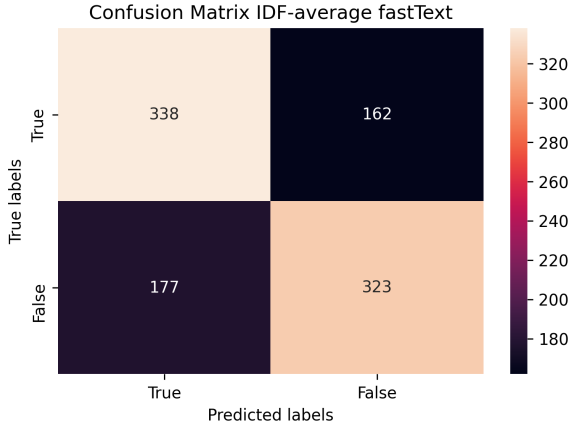

Figure 7: Confusion matrix IDF-average model with fastText embeddings

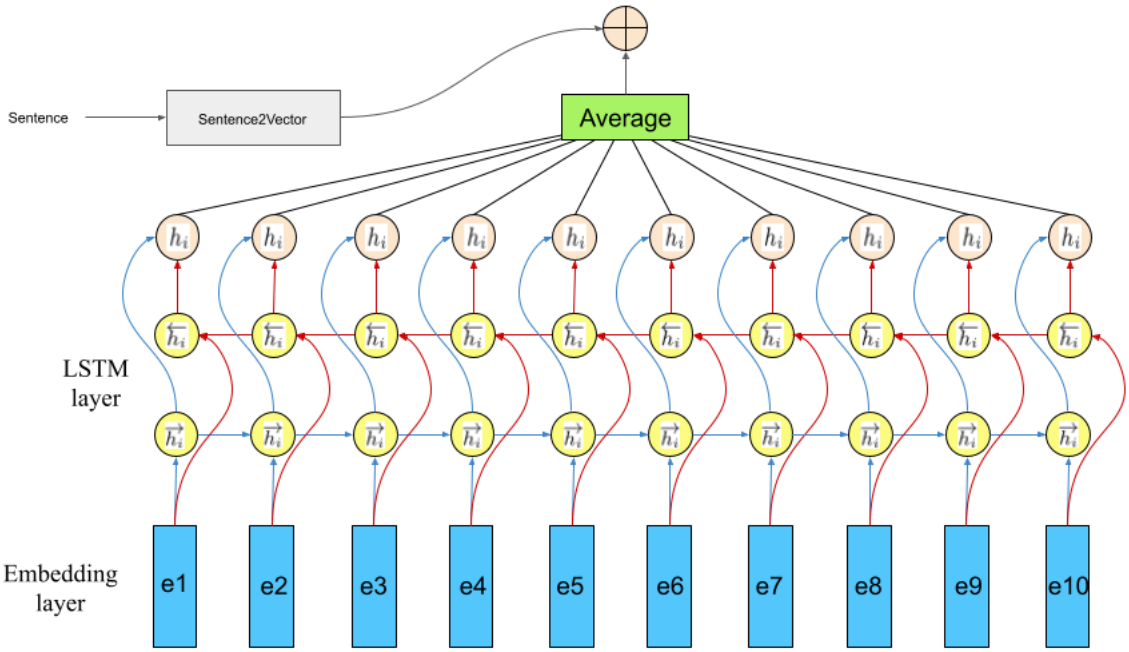

Figure 8: Sequence encoding model architecture

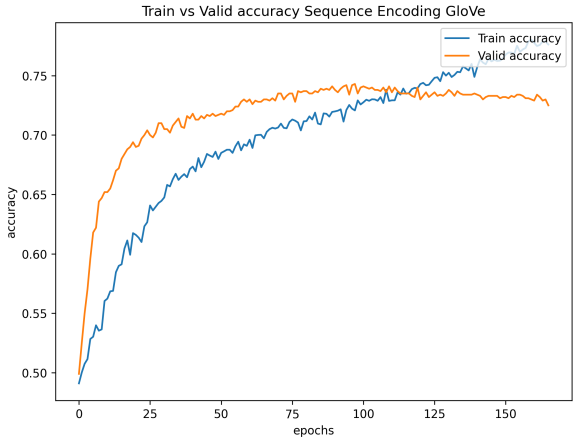

Figure 9: Train and Dev accuracy of the Sequence Encoding model with GloVe embeddings

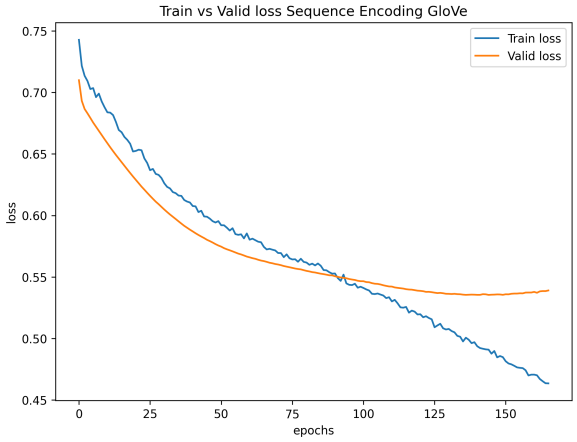

Figure 10: Loss Sequence Encoding model with GloVe embeddings

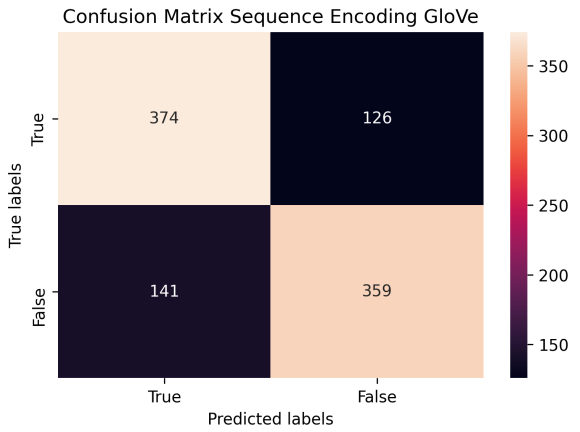

Figure 11: Confusion matrix Sequence Encoding model with GloVe embeddings

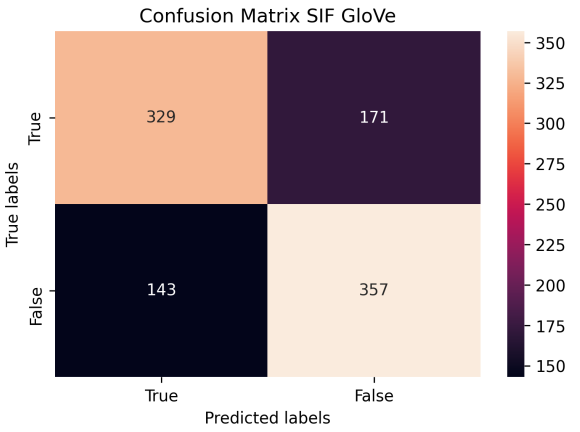

Figure 14: Confusion matrix SIF model with GloVe embeddings

Figure 12: Train and Dev accuracy of the SIF model with GloVe embeddings

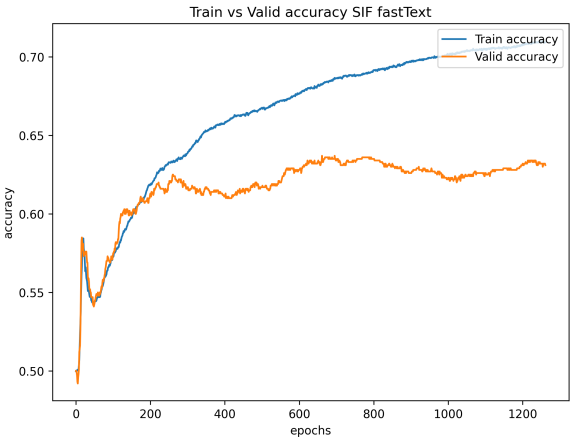

Figure 15: Accuracy SIF model with fastText embeddings

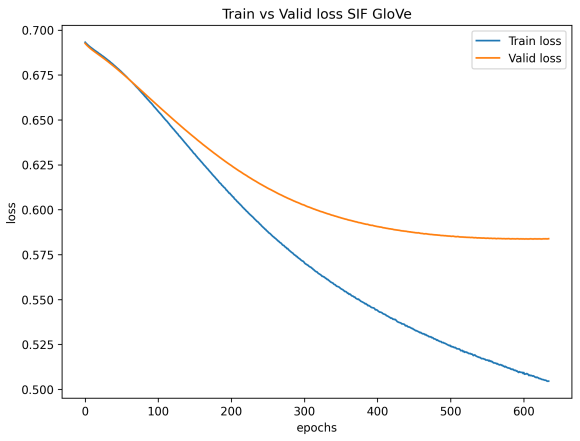

Figure 13: Loss SIF model with GloVe embeddings

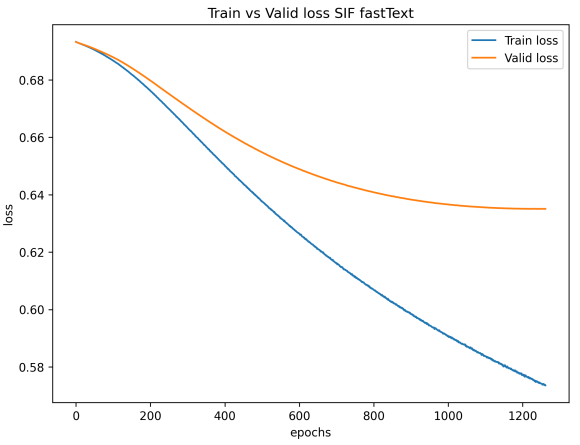

Figure 16: Loss SIF model with fastText embeddings

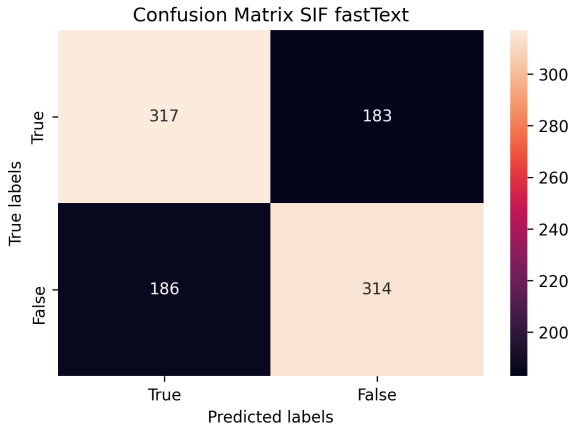

Figure 17: Confusion matrix SIF model with fastText embeddings

References

Sanjeev Arora, Yingyu Liang, and Tengyu Ma. 2017. A simple but tough-to-beat baseline for sentence embeddings. In *International Conference on Learning Representations*.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomás Mikolov. 2016. Enriching word vectors with subword information. volume abs/1607.04606.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. volume 9, page 1735–1780, Cambridge, MA, USA. MIT Press.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomás Mikolov. 2016. Bag of tricks for efficient text classification. volume abs/1607.01759.

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.