Homework 1: Word-in-Context Disambiguation

Michele Conti
conti.1599133@studenti.uniroma1.it

1 Introduction

WiC disambiguation is a binary classification task. Each instance in the dataset has a certain target word w and two contexts (i.e., sentences) c_1 and c_2 in which w is contained. The task is to identify whether the occurrences of the target word w in the two sentences carry the same meaning or not.

In this paper I present two architectures to address this task, both based on the use of pretrained embeddings and bilinear layers.

2 Preprocessing

Various options have been considered in order to preprocess the data, but, ultimately, I decided to follow a very basic pipeline in this phase. Specifically, all the pairs of sentences have been lowercased, all the punctuation has been stripped, and finally I removed all the stopwords and digits from the sentences. As for stemming and lemmatization, instead, I specifically decided to avoid these operations for two reasons: the first one being that I already had in mind to exploit pretrained word embeddings, which are often computed for non lemmatized/stemmed words, and also because I felt like the specific form of each word could help disambiguate one sentence from the other.

To create the vocabulary, it was my first decision to retain all the words appearing in the training dataset, no matter their frequency, since the training set is not very large (i.e., 26119 unique words after the preprocessing part).

In the next sections we are going to see that even this step can be improved, creating the vocabulary from an external source. Specifically, I decided to inherit the vocabulary of Glove, an unsupervised learning algorithm for obtaining vector representations for words (Pennington et al., 2014).

3 Models

3.1 Models architecture

Two general structures have been used as baselines to assess the complexity of the given task, and to identify possible improvements in the models. Namely, the first structure I implemented is only based on linear layers, while the second one is composed by an LSTM followed by linear layers.

3.2 Embeddings

The very first module each architecture has is an embedding layer, where each token of each sentence gets projected into a vector space, so that the data can subsequently be fed to the next layers. This layer works as a lookup table between an index, indicating the position of the word in the vocabulary, and its vector representation, called embedding. The vector embeddings are randomly initialized, and they are then tuned during the training phase to minimize the loss function.

3.3 Embeddings aggregation

Once the embeddings for the words of both sentences are computed, we need a way to aggregate this information, to extract the meaning of the sentences as a whole. There are two general approaches we can use when trying to accomplish this task. The first one, the simplest, is to compute some summary statistic (e.g., sum, average, weighted average, etc.) of the set of embeddings. The second approach, on the other hand, involves the use of a recurrent linear layer to encode the sequence of embedding into a single vector.

It's not always the case that harder is better: in fact, I noticed that simple summary statistics works much better than the sequence encoding. Specifically, taking the average of the embeddings seems to be the best option between the considered options (i.e., sum, average, weighted average centered

around the target word).

3.4 Classification head

The embeddings aggregation is then linked to a classification head, which is composed of one linear layer and a ReLU activation function, followed by a final fully connected layer and a sigmoid function.

4 Experiments

In this section, we are going to explore possible modifications to these two baseline models we just described, comparing at each step the performance of the two structures.

Since a wide range of features have been tested, at each step we're only going to keep the ones that proved to be effective, actually increasing the performance of the two models.

4.1 Pretrained word embeddings

The first and obvious feature I added to my models is to substitute randomly initialized word embeddings with pretrained ones. Pretrained word embeddings are in fact much more reliable than the ones we could train on our data, since they are trained on very large corpora and therefore allow very high generalization.

Moreover, I noticed that training my own word embedding, my models couldn't even surpass a random classifier, thus the mandatory need of a different strategy.

For this purpose, as I already anticipated in the preprocessing section, I used Glove, adopting its vocabulary as my own, and considering as OOV words the ones that didn't appear in it. For these words, I decided to use a randomly initialized vector as embedding.

5 Figures and tables

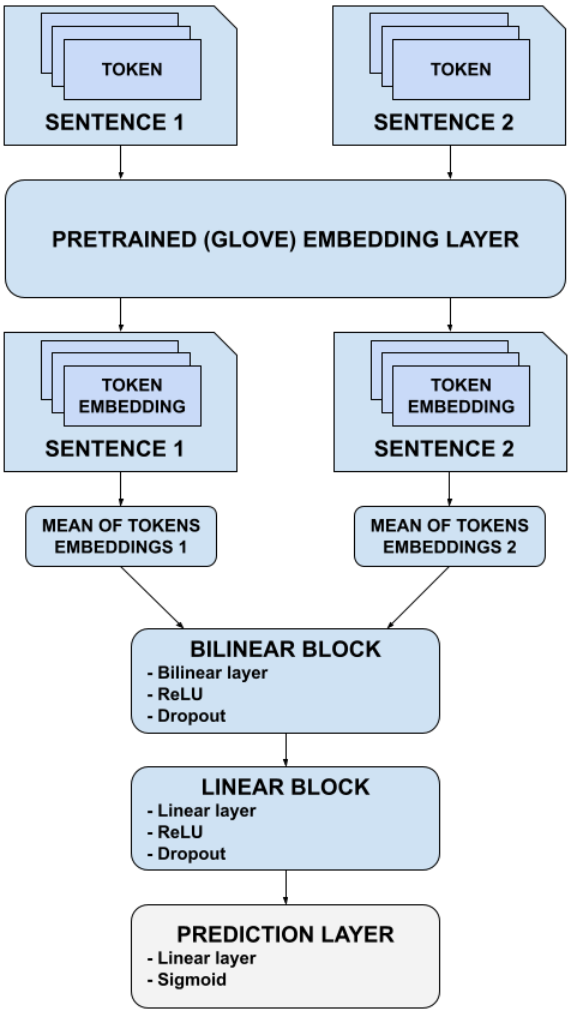

Figure 1: Final model architecture.

Model	Dev. acc.
Glove + Average + Conc. + Linear	0.677
Glove + LSTM + Conc. + Linear	0.651
Glove + Average + Bilinear + Linear	0.707
Glove + LSTM + Bilinear	0.671

Table 1: Best architectures performances.

References

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.