Homework 1: Word-in-Context Disambiguation

Michela Proietti - 1739846
Sapienza University of Rome
proietti.1739846@studenti.uniroma1.it

1 Introduction

Most of the words we use in our everyday life are polysemous, which means they can have different meanings depending on the context in which they are used. Word-in-context disambiguation aims at identifying the exact meaning with which a word is used in a given context. In particular, our purpose is that of determining if two words in two different sentences are used with the same meaning, and therefore we can model our problem as a binary classification task. Two different approaches have been developed to address this problem: in the first one we just use a combination of the word embeddings of both sentences in order to train a relatively shallow neural network, while the second one is based on the use of an Embedding layer and a bidirectional LSTM.

2 Pretrained Word Embeddings

Word embeddings are numerical representations of our input sentences. In both approaches that have been tested, we have used pretrained word embeddings, that are learned on large corpora, and therefore they are more likely to give good results than the ones trained on smaller datasets. Moreover, performing an initial analysis of the given training and validation sets, we have discovered that the training set contains just 26395 distinct words, and that a quarter of the occurrences in the validation set are of words that appear in the training set less than 20 times (see figure 1 and table 1). For this reason, by using pretrained embeddings we are able to better generalize on these words and to consistently speed up the training time of our model. More specifically, we have used GloVe, trained on Wikipedia and English Gigaword (Fifth Edition), and Word2Vec, trained on Google News, and both of them are made up of 300-dimensional word embeddings.

3 Preprocessing

Before building the word vectors, all sentences have been preprocessed, and two types of preprocessing have been tested. In both cases, all sentences have been lowercased. Then, in the first method, some of the most common contracted forms, such as *I'm* and *can't*, have been replaced with the corresponding extended forms, and all the *'s* at the end of words have been removed. Additionally, all the characters different from alphabetic letters have been eliminated, included punctuation marks and numbers. Finally, we have removed all the repeated spaces, so that we are left with a sentence in which words are divided by single spaces. Then, before building the vector representation of the sentence pairs that are used during training, we also removed all the stopwords, such as *the*, *a* and *but*, so to be left with only few meaningful tokens, thus decreasing the training time and eventually improving the model's performance. In the second preprocessing method, instead, we have used lemmatization, which is a word normalization technique which lets us retrieve the lemma of each token. So, after lowercasing the sentence, we split it and we call the WordNetLemmatizer on each token which is not a stopword.

4 Models

As previously said in the introduction, we have implemented two different approaches in order to perform the given binary classification task, and in both of them we started by building three different dictionaries. First of all, we have built two initial dictionaries by using as keys all the words available respectively in GloVe and Word2Vec, and as values the corresponding word vectors. Then, we have built a dictionary containing the keys present in both previous dictionaries and having as values the concatenation of the word embeddings coming from GloVe and Word2Vec, thus getting for each word a 600-dimensional representation. Then, since the two initial dictionaries for GloVe and Word2Vec were too large, for efficiency reasons we have reduced the keys to those that were present in both GloVe and Word2Vec. In fact, we assumed that since these words are present in word embeddings coming from different corpora, they are probably very frequent words. So, all dictionaries contain the vector representation of 284859 words.

First Approach In the first approach, we build the word vectors by associating to each token of the preprocessed sentences the embedding we retrieve from one of the three previously mentioned dictionaries. If the word is not present in the vocabulary, we simply ignore it. Then, in the case of an individual use of the pretrained embeddings, we compute the mean of the word embeddings for each sentence and we concatenate them, thus getting a 600-dimensional vector. Instead, when we use the combination of GloVe and Word2Vec embeddings, we simply retrieve the embeddings of the tokens of both sentences, we concatenate them, and just then we compute the mean. In this way, even in this case, we will have a 600-dimensional vector, so that we will always have 600 input features. The model used in order to perform the classification is made up by 2 linear layers and we have also added a dropout layer in order to reduce overfitting, since the loss on the training set continued to grow while the one on the validation set started to increase again. The activation function that has been used is the PReLU function, which differs from ReLu because it has a non-zero slope for negative inputs, so that negative inputs are not simply zeroed out. Finally, since we are addressing a binary classification problem, the last layer has a sigmoid activation function, and as loss function we have used binary cross-entropy. Moreover, in a second moment, early stopping was added, because the loss on the validation set decreased just during the first 10-25 epochs, and then started to increase again, thus causing a worsening in the performance of the model.

Second Approach In the second approach, we perform word vectorization using an Embedding layer. In order to do this, we create a dictionary to index each word in the input sentences, and a list of the corresponding word embeddings, adding also two random vectors that correspond to indices 0 and 1, to handle respectively the pad token and the unknown token. Therefore, all words that are not present in the used vocabulary will be mapped to index 1 by default. The embedding layer is followed by a dropout layer and a bidirectional LSTM with just one layer. Then, in order to perform classification, we have added two linear layers. In this case, the activation function that has been used is the ReLU function, because it is the one that gave the best results, and as in the previous approach, we have used early stopping to limit overfitting.

5 Experiments

As regards the first approach, the two different types of preprocessing gave similar results, but the second one resulted in a slightly better accuracy in the case of GloVe and mixed pretrained embeddings. We can notice from table 4 that by increasing the batch size, the accuracy was lowered in all three cases, and that adding dropout benefited on the performance while using GloVe or mixed embeddings, but it strongly lowered the performance while using Word2Vec. Concerning early stopping, the patience has been set to 5, and this lets us stop the training as soon as the validation loss starts to increase, in order to prevent the performance from worsening. We can see that in doing this, the accuracy in the case of Word2Vec decreases, because despite the model starts overfitting, the validation accuracy continues to grow for a while, but this does not happen with the other pretrained embeddings. For the second approach, we have in general much lower results. Even in this case, several structural changes have been applied in order to try to improve the performance of the network, but none of them led to sufficiently good results. In fact, the model that gave the best results with this approach has an accuracy of 57%, which is much lower than the one obtained with the first method.

6 Conclusions

The first method, despite being very simple, allowed us to obtain quite good results, with also the advantage of a very fast training. On the contrary, the LSTM-based approach resulted in a much more unstable behaviour, due to its strong sensitivity to overfitting. This shows that simple approaches can sometimes allow us to obtain sufficiently good results making life much easier.

7 Figures and Tables

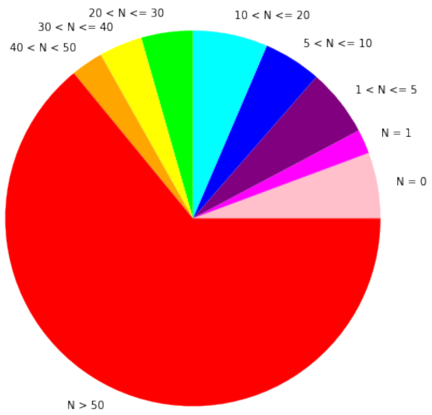

Figure 1: This pie chart shows how many times words that occur in the validation set are present in the training set. For example, we can see that a quarter of the occurrences are of words that appear less than 20 times in the training set (see table below for the exact values).

Seen in train set	Occurrences in dev set
$N = 0$	2580
$N = 1$	948
$1 < N \leq 5$	2651
$5 < N \leq 10$	2267
$10 < N \leq 20$	2934
$20 < N \leq 30$	2022
$30 < N \leq 40$	1698
$40 < N \leq 50$	1261
$N > 50$	29162

Table 1: This table shows how many times words that occur in the validation set are present in the training set. For example, 2580 occurrences in the validation set are of words that are not present in the training set, 984 words that occur in the validation set are present just once in the training set and so on.

Hyperparameter	Value
Batch size	16
Activation function	PReLU
Dropout	0.2
Number of layers	2
Hidden units	64
Optimizer	Adam
Learning rate	0.0001
Beta 1	0.85
Beta 2	0.999 (default)
Early stopping patience	5

Table 2: Values of the hyperparameters used in the final model in the case of the first approach. Notice that the value of beta 1 has been decreased from 0.9 to 0.85 to make the training of the model more stable by reducing the oscillations.

Metric	Value
Precision	0.6829
Recall	0.6790
F1 score	0.6773
Accuracy	0.6790

Table 3: Performance obtained on the test using the first approach.

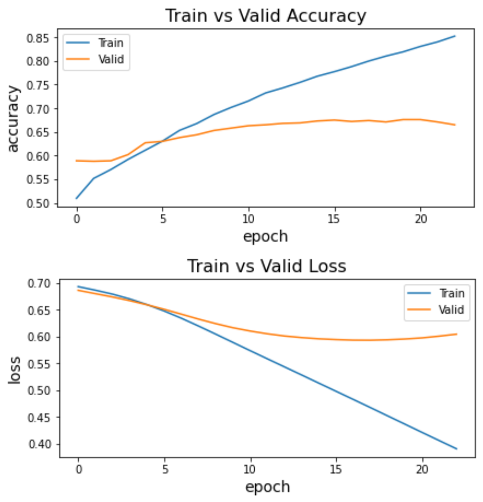

Figure 2: Training history obtained using the first approach with GloVe embeddings. As we can see, the validation accuracy does not grow much, but starts from a quite high value. The validation loss, instead, initially decreases together with the training loss, but then when the model starts to overfit, it becomes less steep. However, early stopping prevents it from increasing again.

Model variations	GloVe	Word2Vec	GloVe + Word2Vec
Final model	0.68	0.53	0.63
Final model (initial preprocessing)	0.67	0.58	0.62
Batch size 32	0.66	0.57	0.63
Batch size 64	0.66	0.57	0.63
Batch size 128	0.65	0.56	0.62
No dropout	0.65	0.58	0.62
ReLU	0.66	0.58	0.62
SGD (lr=0.1)	0.63	0.57	0.62
Adam default betas	0.65	0.56	0.63
3 layers NN	0.65	0.58	0.63
Adam lr=0.01	0.59	0.50	0.51
Early stopping 10	0.66	0.59	0.61
Early stopping 2	0.64	0.57	0.62

Table 4: This table shows the accuracy values obtained on the validation set using the first approach, changing the value of some hyperparameters or slightly changing the structure of the network. In particular, we start from the final structure, that has given the highest accuracy using GloVe embeddings, and we show the differences we get by increasing the batch size, removing dropout, using ReLu in place of PReLU, SGD in place of Adam, changing the parameters of Adam itself, decreasing the number of layers to two, changing the patience in early stopping, or using the initial type of preprocessing.

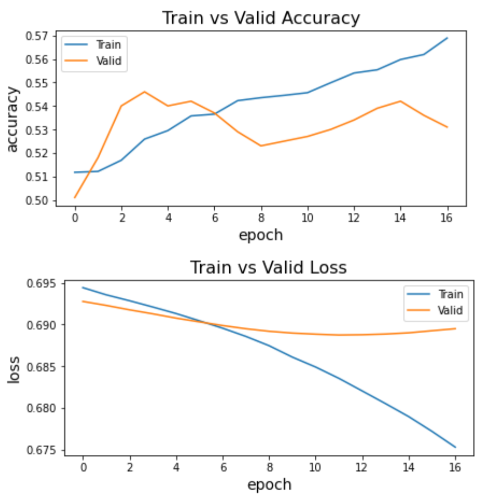

Figure 3: Training history obtained using the first approach with Word2Vec embeddings. We easily notice that the validation accuracy changes in a much more unstable way with respect to the one obtained with GloVe. Moreover, the loss stops decreasing sooner.

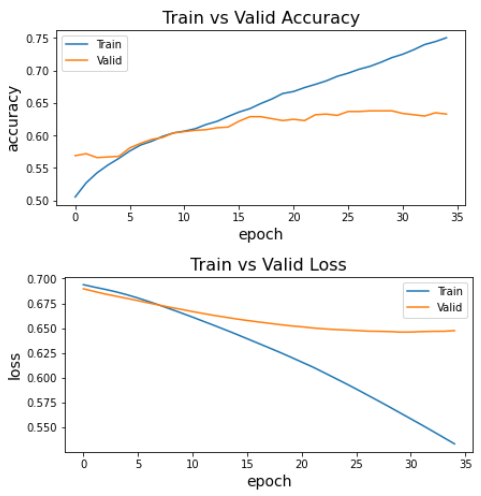

Figure 4: Training history obtained using the first approach with GloVe and Word2Vec embeddings. The validation loss is much similar to the one obtained with Word2Vec embeddings, but it starts from a lower value. The validation accuracy, instead is more stable, and its trend is more similar to the one obtained with GloVe. However, the accuracy reached is slightly lower.

Hyperparameter	Value
Batch size	32
Activation function	ReLU
Dropout	0.2
Number of linear layers	2
Hidden units	128
Optimizer	Adam
Learning rate	0.0001
Beta 1	0.85
Beta 2	0.999 (default)
Early stopping patience	5

Table 5: Values of the hyperparameters used in the best model in the case of the second approach.

Metric	Value
Precision	0.5707
Recall	0.5690
F1 score	0.5664
Accuracy	0.5690

Table 7: Performance obtained on the test using the second approach.

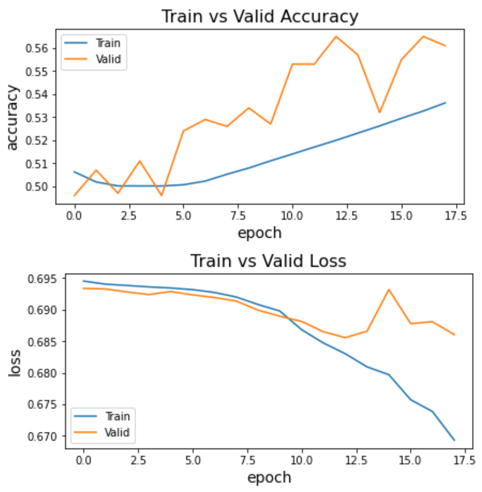

Figure 5: Training history obtained using the second approach with GloVe embeddings. As we can see, the validation accuracy has a much more unstable trend than with the first approach. The validation loss initially decreases together with the training loss, but then takes as well an oscillatory behaviour.

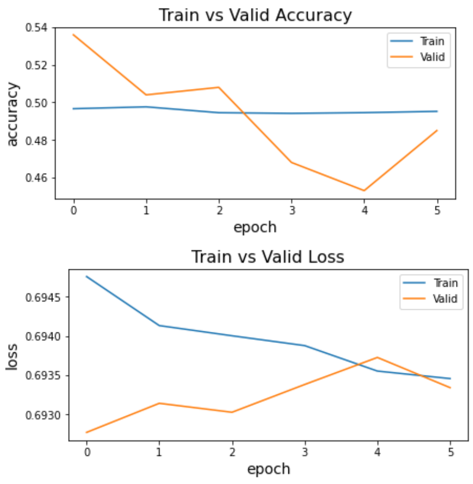

Figure 6: Training history obtained using the second approach with Word2Vec embeddings. We easily notice that the steps are too big, and this is why the results we obtain are very poor. However, even by decreasing the learning rate we were not able to stabilize the behaviour of the model

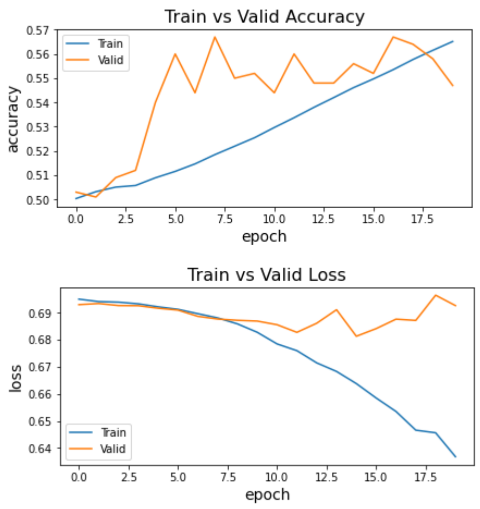

Figure 7: Training history obtained using the second approach with GloVe and Word2Vec embeddings. The trends of both loss and accuracy are very similar to those obtained using GloVe embeddings, and this justifies the very similar results that we have obtained.

Model variations	**GloVe**	**Word2Vec**	**GloVe + Word2Vec**
Final model	0.57	0.53	0.56
Final model (initial preprocessing)	0.56	0.51	0.55
No dropout	0.55	0.50	0.61
PReLU	0.54	0.52	0.53
Early stopping 10	0.54	0.49	0.53

Table 6: This table shows the accuracy values obtained on the validation set using the second approach, changing the value of some hyperparameters or changing the preprocessing method. Even in this case, we obtain the highest results using GloVe pretrained embeddings, but we see that the results obtained with the mixed embeddings are almost the same in this case.