Homework 1: Word-in-Context Disambiguation

Giammarco D'Alessandro
Sapienza University of Rome, matricola 1753102
dalessandro.1753102@studenti.uniroma1.it

2^{nd} May 2021

Abstract

This is the report for the Natural Language Processing course (a.y. 20/21) first homework on Word-in-Context disambiguation.

1 Introduction

In NLP Word-in-Context (WiC) disambiguation is a classification task where the goal is to disambiguate polysemous words, without relying on a fixed resource of word senses. In particular in this homework we were asked to identify whether, given a sentence pair and a polysemous target lemma, the target lemma is used in both sentences with the same meaning or not. To address this task I have pre-processed the training data, taking advantage of GloVe6B pre-trained embeddings (Pennington et al., 2014) for the latent representation of words, and I have developed two models following different architectures (*BaseMLPClassifier* and *RecurrentLSTMClassifier*), to implement the two approaches we were assigned: word-level and sequence encoding.

2 Data pre-processing

As common to many NLP tasks, I started with building a fixed size vocabulary over the dataset, in order to exploit the word embedding representation. This is also necessary because it is impossible to have a vocabulary covering all words, given that new words are often invented, and that dealing with a huge vocabulary would be computationally unpractical. Thus I have constructed the vocabulary over the 20000 most frequent words in the training data, between the 27206 present, including a token to represents all the so called Out-Of-Vocabulary (OOV) words, a token to separate the sentences, and a token to pad the sequences for the second approach (Tab.3).

Then I have processed the sentences in the data (assigning the correct vocabulary index to each word), trying to get information on the relative context, first applying lemmatization to the target word (i.e. substituting it with its lemma), and then removing the stopwords using nltk python package (Steven et al., 2009).

2.1 Pre-trained embeddings

Once built the vocabulary I have loaded the GloVe6B pre-trained embeddings (I chose dimension 50) to assign each word to a tensor representing its embedding, using a random tensor for the unknown token (and thus for all OOV words), and a random tensor for each word in the vocabulary which pre-trained embedding was missing in GloVe6B. The result of this process is a matrix of size $vocab_size$x$embedding_dim$ having in each row the embedding tensor of a vocabulary word, that can be used to create the embedding layer. This layer acts as a simple lookup-table that maps the the index of a word to the corresponding embedding.

3 Models

Then I developed the two different architectures, for the word-level and the sequence encoding problems. Both exploit the same embedding layer (as explained above), because this task is the same over the two approaches there is no need for different solutions.

3.1 First approach: word-level

The *BaseMLPClassifier* model implements the word-level solution in three steps.

i) First associates the pre-processed data, i.e. sequences of indexes, with the related embedding tensors, via the embedding layer. ii) Computes the average representation of each sentence of a pair (the mean tensor of its words embeddings), and concatenates them. iii) The resulting tensor (of size 2x$embedding_dim$) is then fed to a Multi Layer Perceptron network for the actual classification step.

The MLP is composed of 3 layers: two Fully-Connected layers followed by a reLu activation each (size 2x$embedding_dim$x100, and 100x100 respectively), and an output layer (size 100x1) followed by the sigmoid activation.

3.2 Second approach: sequence encoding

The *RecurrentLSTMClassifier* model implements the sequence encoding solution following the same semantic steps of the previous, but the aggregation step is computed via a bidirectional-LSTM (biLSTM) that catches a better representation of the sentence context, analyzing it in both direction (left-to-right and right-to-left).

The biLSTM takes as input the sequence of tensors resulting from the embedding step of each sentence in a pair. Each sequence is fed twice to the biLSTM, respectively for the left-to-right and right-to-left contexts, and then the two output sequences are merged, concatenating each tensor of a context to the corresponding tensor of the other direction, obtaining a unique output sequence. Then I compute the absolute value of the difference of the context representations of the two sentences, and this tensor represents the input of the final MLP classification step, again with three layers.

For this solution I have slightly increased the first layer size ($embedding\_dim$ x 128) to better deal with the higher amount of feature provided by the biLSTM w.r.t the simple aggregation of the *BaseMLPClassifier* model, and decreased the second (128x64) to reduce the overfitting. Both FC layers are followed by a reLu activation, and the output (64x1) is again followed by a sigmoid activation.

4 Training and validation

Training and evaluation have been carried out mainly considering the accuracy metrics, that better describes the performances of a classification task. The train loop was set to evaluate the model over the dev dataset at the end of each training epoch, and I implemented an early stopping technique with a patience level (5 for both models, decreased of 1 when both dev loss increased and dev accuracy increased), to reduce the overall overfitting.

For both model I achieved the best performances using a Steep Gradient Descent (SGD) optimizer, with a learning rate of 0.2, momentum 0.001 and a weight decay of $1e^{-3}$.

5 Experiments

The configuration of the models shown above are the parameters that allowed me to obtain the best results with both (Figs.1,2 and Tabs.1, 2). Indeed I tried various configuration at each step of the two models before achieving those results. The values are reported in the corresponding tables below ...

Pre-processing: I have tried to preprocess the dataset in different ways, to remove or keep the stopwords, to lemmatize or not the target word, and to move the target at the beginning or at the end of a sequence (when using the biLSTM for the sequence encoding).

Vocabulary: I have tested different sizes for the considered vocabulary (10000, 15000, 20000), and the higher gave me better results probably because increasing the vocabulary size reduces the number of words that need to be represented by the unknown token, a random vector, giving a more accurate representation of a sentence context (Tab.4).

Aggregation: For the word-level approach I tested the absolute value of difference and the difference of the square value of the sentences mean embeddings, but both gave me worse accuracy scores (Tab.5).

For the second approach I first tried to process the sentence pair as a single sequence, taking the overall average, and then to process the two sentences with two different biLSTM. In the former the accuracy was lower than the best, and in the latter the training took longer with no significant improvements.

Classification: For the SGD optimizer I tried different learning rates (0.1,0.2,0.3). With a too low learning rate I was always getting good performances but with significantly higher training time (80/100 epochs with lr=0.1 versus 10/15 with lr=0.2). I have encountered the same issue when testing the Adam optimizer, but obtaining lower accuracy performances with all the approaches.

6 Conclusions

It is clear from this analysis that the *RecurrentLSTMClassifier* model gives better results for the WiC disambiguation, and that given the similar configuration of the preprocessing, the bidirectional LSTM approach achieves a better accuracy.

7 Results

The following tables and plots reports the results of my experiments.

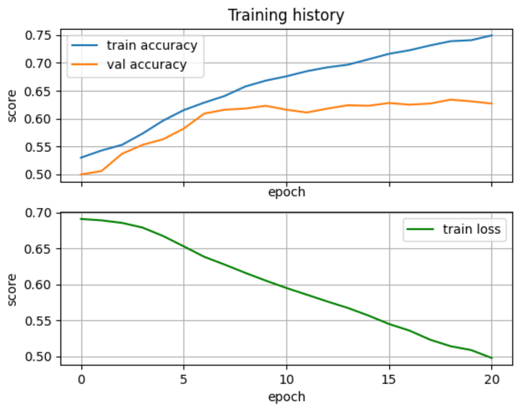

Figure 1: *The training history of the word-level best results*

Accuracy	avg	max
train	0.6599	0.7491
eval	0.5986	0.6340

conf.mat	True	False
True	2927	1073
False	934	3066

Table 1: *The accuracy scores of the word-level best results and the relative confuion matrix.*

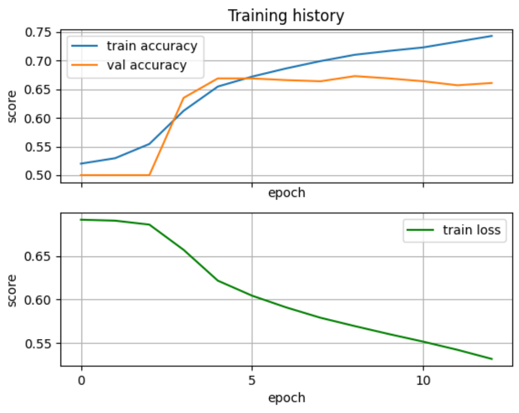

Figure 2: *The training history of the sequence encoding best results*

Accuracy	avg	max
train	0.6698	0.7565
eval	0.6321	0.6790

conf.mat	True	False
True	3052	948
False	1000	3000

Table 2: *The accuracy scores of the RecurrentLSTM best results and the relative confusion matrix.*

vocab size	10000	15000	20000
total vocab words	368424	382088	390966
missing embs	0.27%	1.03%	2.6%

Table 3: *The training dataset composition on 27206 distinct words present in 8000 sentence pairs.*

vocab size	10000	15000
word-level	0.6180	0.6410
sequence-enc	0.6580	0.6620

Table 4: *The best evaluation accuracy scores varying the vocabulary size for both models.*

	word-level	seq.enc.
concat	0.6340	0.6410
abs-diff	0.6170	0.6790
square-diff	0.6030	0.6520

Table 5: *The best evaluation accuracy scores varying the aggregation function for both models.*

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.

Bird Steven, Edward Loper, and Ewan Klein. 2009. [link].