NLP 2021 Homework 1: Word-in-Context Disambiguation

Leandro Maglianella - Matricola 1792507

1 Introduction

The Natural Language Processing task accomplished in this paper consists in the creation of a supervised binary classification model with the ability to establish whether in two sentences a polysemous word is used with the same meaning or not (Roberto N. 2009). In the next sections we will analyze the techniques used to carry out this work. Then, the two macro approaches (working on word-level and using a sequence encoder) and various experiments will be described in details: we will explore my experimental models in order to explain the design choices made and we will look at their layer's components, in particular the bidirectional LSTM architecture will be discussed. Finally, the results obtained will be shown.

2 Random Baseline Model

A random baseline model was tested to understand and verify the working environment's functioning. The provided datasets to train and validate the model contain respectively 8000 and 1000 samples and they are balanced: for this reason, this *approach zero* has an average accuracy of 0.5 with respect to this binary problem.

3 Pre-Trained Word Embeddings

In the subsequent approaches I employed a pre-trained word embedding: in particular, I have used a *Word2Vec Continuous Skipgram* (LTG. 2017) trained with the *English CoNLL17 corpus*. It is a vocabulary which maps 40271169 words to 100-dimensional vectors. Each of these vectors tries to represent its associated word identifying the word's latent features. Word embeddings are in fact exploited to speed up and improve the model's learning process by providing hidden linguistic properties on which a neural network can train. Of the over four million words, I decided to select only the first 100 thousand in order not to make the work too resource demanding: furthermore, this word embedding is organized in order of use frequency of the words, so the selected words are the most relevant ones. Finally, I added to this limited vocabulary three random vectors which I have used as tokens to represent paddings, unknown words and sentence separators. In addition, in my approaches I have also experimented with *GloVe 6B 300d* (Stanford NLP. 2016) word embedding, obtaining lower results; because of this, I continued my work only using *Word2Vec*.

4 Data Pre-processing

In the first approach, each pair of sentences was processed by concatenating them (separating them with the separator token), transforming their words into their hidden features' vectors and, finally, the average of each vector's component was calculated among the considered words. Furthermore, data have been pre-processed so that every punctuation mark was eliminated and I chose to ignore words whose length was less than four characters. Finally, each *(x, y)* sample that enters the training phase is composed of a 100-dimensional vector *x* and, being this a supervised problem, its label *y*.

In the second approach, data were processed in a very similar way: the main difference is that the embedding phase is transferred to an embedding layer in the model and, for each pair of sentences, the words are transformed into the indexes representing the position of such words in the embedding dictionary. In conclusion, each *(x, y)* sample that enters the training phase is composed of its label *y* and, using a collate function, of a vector *x* of variable size among the various batches: in a batch each vector is padded using the padding token to make it as large as the largest vector in the batch it is contained in.

5 First Approach's Model

Many simple architectures have been experimented; the neural network that has proven to achieve better results on validation data is made of five linear layers, each followed by a *ReLu* activation function and a *dropout* layer to reduce overfitting. A *binary cross-entropy* loss function and a *sigmoid* output function were used. Training and validation loss and accuracy are computed and observable at training time, however I decided not to use an early stopping mechanism to allow a complete training which I then refined myself by modifying the hyperparameters' values (the final ones are reported in Table 1). Furthermore, the training was programmed to automatically save the model with the best validation accuracy. Training plots are shown in Figure 1.

6 Second approach's Models

As for the first approach, many architectures have been experimented. The simplest starting model has an embedding layer that uses the embedding dictionary. This layer is followed by a sequence encoder whose output is manipulated, flattened and given as input to a linear classification head composed of two linear layers. As before, also here a *ReLu* activation function, a *binary cross-entropy* loss function and a *sigmoid* output function were used. The used parameters are reported in Table 2.

6.1 Sequence encoder choice

I have empirically tested three sequence encoders: recurrent neural networks (*RNN*), gated recurrent unit RNN (*GRU*) and long short-term memory RNN (*LSTM*). The *LSTM* based model (PyTorch. 2019) achieved the best validation results among them therefore I chose it as a starting point for the continuation of my experiments. As a very first addition, I inserted a *dropout* layer between the embedding layer and the *LSTM*, lightly improving the results. Training plots are shown in Figure 2.

6.2 Bidirectional LSTM

The next step in my experiments was to replace the just mentioned unidirectional *LSTM* with a *bidirectional LSTM*. The operation of these architectures is basically the same: the difference is that while the unidirectional *LSTM* analyze the input vector only from left to right (in a positive time direction), the bidirectional one uses a second layer to compute the output also onto the reversed input (from right to left, in a negative time direction). In *NLP*, this is done so that for each sentence it is possible to obtain a greater context: in fact, in this way it is as if the model "read" the sentences *left-to-right* and *right-to-left*, accumulating more information. The two output sequences resulting from this approach are then merged together, usually concatenating them: in this case I decided to do differently to make sure that the output dimension was identical to the input one; therefore, I correspondingly summed the output vectors of the two different sequences. Training plots are shown in Figure 3.

6.3 Variable learning rates

At this point I added some more extra details (Sebastian R. 2017) to the training phase to improve the result even more. I created a scheduler for my optimizer so that the learning rate could vary during the training itself: in particular, I made sure that when the validation accuracy during training exceeds a limit set by me, then the learning rate decreases. This same dynamic continues to repeat every time validation accuracy reaches a new maximum. I added this feature to try to obtain my model's highest achievable result. Finally, I tried to implement a transfer learning mechanism by re-training several times and with different parameters the previous best models that I had managed to obtain: this last technique did not bring improvements and I therefore discarded it from my model. Training plots are shown in Figure 4.

7 Results

The final results that I was able to achieve are shown in Table 3 and Table 5. As can be seen, despite my additions, improvements and many hyperparameters' tunings made during the second approach (which I do not report in completeness but show partially in Table 4) I was unable to overcome the validation accuracy obtained by the first approach. In fact, due to a strong overfitting to the training data, training hangs around a validation accuracy of 0.63-0.64, if not even below these values. The reasons for this can be traced back to various things. Among these, basic weaknesses of the RNN architecture, the relative non-complexity of the created models and an imperfect pre-processing of the data: in fact, the info of the lemma to be classified, its positions in the sentences or its POS tag were not used.

8 Figures

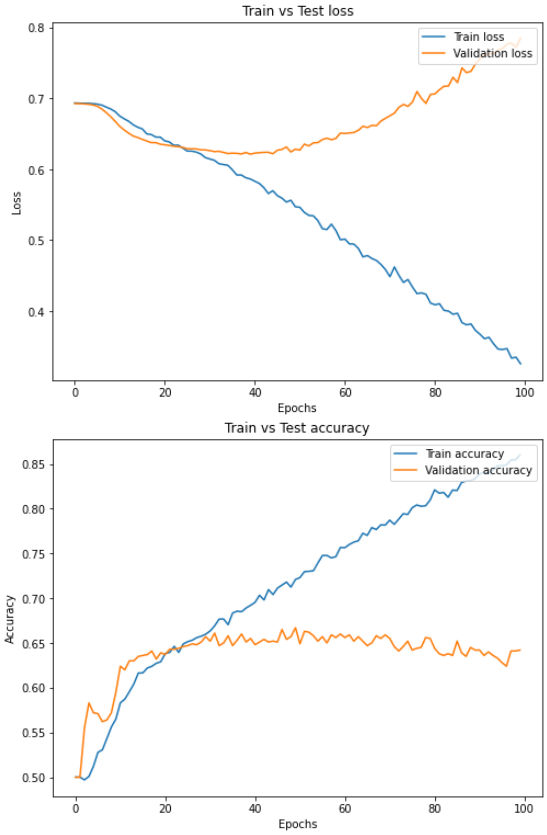

Figure 1: First approach's plots (max epoch 50)

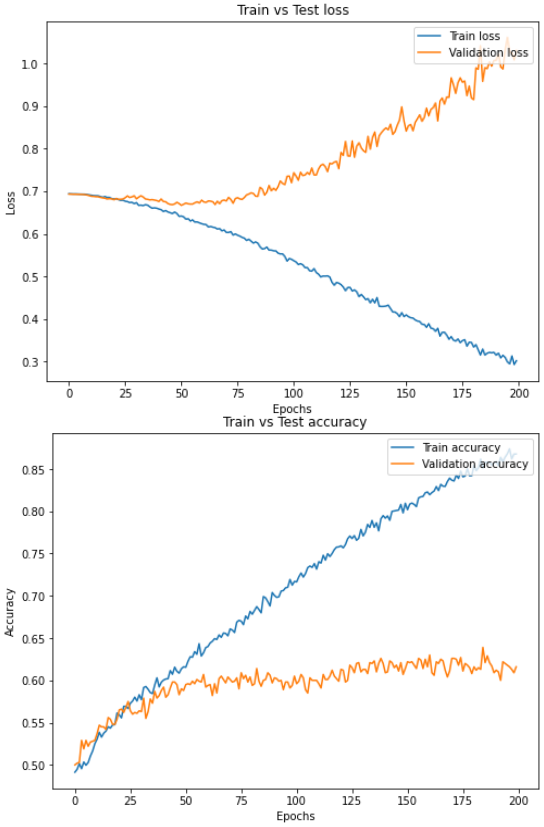

Figure 3: Second approach's plots 2 (max epoch 185)

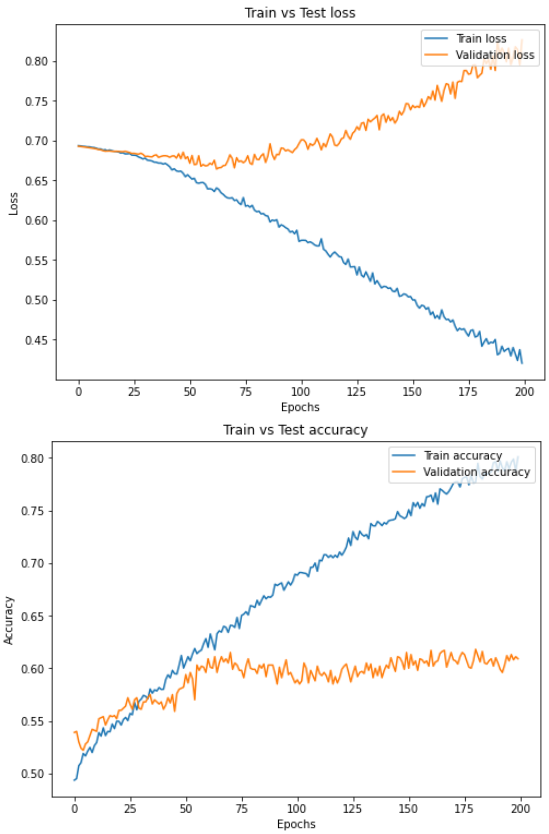

Figure 2: Second approach's plots 1 (max epoch 181)

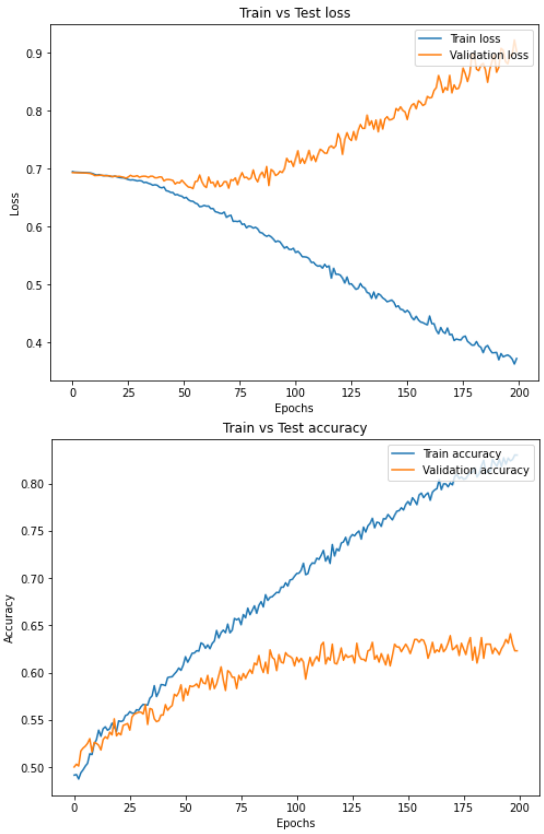

Figure 4: Second approach's plots 3 (max epoch 197)

9 Tables

Parameter	Value
Batch size	32
Optimizer	Adam
Learning Rate	0.00005
Epochs	100
Dropout	0.4
Linear Layer1 in-out	100 → 1024
Linear Layer2	1024 → 512
Linear Layer3	512 → 256
Linear Layer4	256 → 128
Linear Layer5	128 → 1

Table 1: First approach's parameters

Parameter	Value
Batch size	32
Optimizer	Adam
Learning Rate	0.0001
Epochs	200
Embedding Layer	$X \rightarrow 100$
LSTM	100 → 100
Linear Layer1 in-out	100 → 64
Linear Layer2 in-out	64 → 1
Dropout	0.5
Bi-LSTM	100 → 200 → 100 (flat)
Learning Rate gamma	0.9 (LR decayed 6 times in my training)

Table 2: Second approach's parameters

Model	Validation Accuracy
Random Baseline	0.5
First Approach	0.667
Second Approach	
RNN	0.585
GRU	0.591
LSTM	0.615
LSTM + Dropout	0.618
Bi-LSTM + Dropout	0.639
Bi-LSTM + Dropout + variable Learning Rate	0.641

Table 3: Models' results (accuracy performance)

Tested Parameter	Value (tested from X → to Y)
Minimum word length	1 → 5
Batch size	32 → 128
Optimizer	Adam, SGD
Learning Rate	0.1 → 0.000001
Epochs	10 → 1000
Dropout	0.1 → 0.6
Layers dimensions	16 → 2048

Table 4: Parameters tuning and testing

Precision	0.6673
Recall	0.6670
F1-score	0.6669
Accuracy	0.6670

Table 5: Best model's validation metrics

10 References

LTG. 2017. *NLPL word embeddings repository.* University of Oslo.

Nitish S., Geoffrey H., Alex K., Ilya S., and Ruslan S. 2014. *Dropout: A simple way to prevent neural networks from overfitting.* Journal of Machine Learning Re-search.

Diederik P. Kingma and Jimmy Ba. 2014. *Adam: A method for stochastic optimization.*

Stanford NLP. 2016. *GloVe model for distributed word representation.*

Sebastian R. 2017. *Deep Learning for NLP Best Practices.*

Roberto N. 2009. *Word Sense Disambiguation: A Survey.* Università di Roma La Sapienza.

PyTorch. 2019. *PyTorch LSTM's documentation.*