NLP 2021 Homework 3:
Word Sense Disambiguation of Word-in-Context Data

Leandro Maglianella - Matricola 1792507

1 Introduction

The Natural Language Processing task accomplished in this paper is *Word Sense Disambiguation* (WSD) which consists in the creation of a supervised classification model with the ability of determining, given a sentence's context, the most probable meaning of a polysemous target word in that sentence (Roberto N., 2009). In the next sections we will analyze the techniques used to carry out this work: after discussing *BERT* and the data pre-processing, my approach and my various experiments will be described in details. I will explain my design choices and explore my various experimental attempts and models. Finally, the results obtained will be shown and the best models will be applied on the *Word-in-Context* (WIC) task and compared with the ones from Homework 1.

2 BERT

The main component of my experimental models' architecture is the *Bidirectional Encoder Representations from Transformers* (BERT) model (Jacob D. et al., 2019), a very recent and powerful technique based on *Transformers* (Ashish V. et al., 2017), able to learn contextual relations between words in texts using the attention mechanism. A lot of pretrained models of the BERT series have been experimented but, in general, I only used its uncased English versions. As a first choice I picked *bert-base-uncased*, a 12 layers models of 768 hidden size dimension with 12 attention heads for a total of 110 million parameters. Being just an Encoder, BERT produces hidden vector representations for every word of the input sentence which can be used as contextualized word embedding. Because of this, to give a proper output I added a final classification head on top of the pretrained BERT: my choice was a simple couple of linear fully-connected layers of dimensions *hidden* × *hidden*//2 and *hidden*//2 × 1 (768 × 384 and 384 × 1). Predictions are generated by means of a *softmax* function. *Dropout* layers (Nitish S. et al., 2014) are inserted between all of the model's components. Finally, the model's loss function is computed using a *cross-entropy* loss function.

3 Data Pre-processing

In my project I used the *Unified Evaluation Framework* produced by Alessandro R. et al. (2017) for the training and validation phases. In particular, this framework is composed of the popular English training corpus *SemCor* and some validation datasets of the *Semantic Evaluation* series (*Senseval 2* and *3* and *SemEval 2007*, *2013* and *2015*). In particular, I used the concatenation of all these validation datasets for validation. As testing dataset, I properly used the WIC data provided to us for the Homework. Because of the limited resources offered in the *Google Colab* environment, I decided to decrease *SemCor*'s size: first I reduced it from 352 to 118 texts, then I created a totally new version *'SemCor_small'* containing only 24 texts. Every sentence featured in these datasets have been processed with the aim of forming suitable samples for BERT. In particular, as stated in the particularly famous and interesting paper of Luyao H. et al. (2019), I used the target word's definition along with the sentence as input for the model: thanks to the *WordNet Database* (George A. M. et al., 1993) I found the right definition for each word exploiting their lemmas, *POS* tags and sense keys. At this point, sentences and definitions were concatenated, tokenized and indexed according to the *WordPiece* embedding to form samples following the BERT's standard (*[ids, attention masks, ids categories, label]*) as indicated in Figure 1a. Because, again, of *Colab* resources, I decided to store externally the processed data as *Pickle* files (Python documentation, 2021).

4 Experimental Stages

The training phase was performed by means of a *AdamW* optimizer from the *Transformers* library. I have widely experimented with the training stage tuning the parameters and training multiple times: all of the results mentioned in the following subsections are visible in Table 1 and Table 2.

4.1 Experimental stage 1: Initial Results

My first model was trained on *SemCor* using the parameters settings proposed by Luyao H. et al. (2019) (batch size of 64, learning rate of 2e-5, epsilon of 1e-8 and dropout of 0.1). Because of the large size of *SemCor* the training failed, therefore I switched to *Semcor_small*, changed the batch size to 4 and trained only for 1 epoch: I was now able to run the entire epoch with no crashes. The result of this initial training was already quite good with an accuracy of **0.6526** on the test dataset.

4.2 Following Attempts

Continuing my experiments, I started using a little different pre-processing approach: I repeated the whole pre-processing phase but surrounding the target word in the sentence with a special token *[TRG]* (Boon Peng Y. et al., 2020) as indicated in Figure 1b. Using this method and the same parameters as before, I trained my BERT model achieving a test accuracy of **0.6543** (0.7043 on validation dataset). At this point I tried to train three other large models of the BERT series: *bert-large-uncased*, *bert-large-uncased-whole-word-masking* and *bert-large-uncased-whole-word-masking-finetuned-squad*. Despite my resources' precautions, unfortunately their training exhausted *Colab* very quickly, so their results are very low. The results until now are showed in Table 1.

4.3 Experimental stage 2: Batch Increment

My next attempts consisted in tuning the batch size, learning rate, dropout and the number of output layers in the model's classification head. The outcomes of this phase confirmed that the number of layers, the learning rate and the dropout were already in an optimal configuration, while instead a batch size of 4 was not; in fact, raising it to 8 I obtained a **0.6637** accuracy on the test set. The training plots of the aforementioned model are showed in Figure 2. I tried to experiment with higher batch sizes, but every size over 8 crashed *Colab*. The results until now are showed in Table 2 (which also contains the final model metrics).

5 Extra

After having fully tuned the parameters, as showed in Table 3, I added some more extra details to the training phase to improve the results even more. To try to obtain my model's highest achievable result, I tried to implement a *transfer learning* mechanism by re-training several times and with different parameters the previous best model from subsection 4.3. This brought me to my final best model for WSD, which I will briefly discuss in the following section. I am providing training plots for this model as well in Figure 3. A lot of best practices for NLP Deep Learning (Sebastian R., 2017) were considered and implemented. As mentioned, I employed *dropout* layers and some *transfer learning* techniques as using *pre-trained models* and *embeddings*. I used extra datasets and created my own training dataset *'SemCor_small'* to overcome *Colab*'s overflow. I documented my work with figures, plots, confusion matrices and tables. I exploited the *ModelCheckpoint* call-back to validate the model during the training epoch: in particular, I tested it ten times per epoch thus allowing me to plot the process. I considered the *EarlyStopping* call-back but obviously I did not use it because, being the training composed by only one epoch, it was unnecessary. I closely followed all the *PyTorch Lightning* standards while building my *data module*, *model* and *trainer* (PyTorch Lightning Documentation, 2021). I seeded my work to ensure its repeatability. Finally, I will conclude this report analyzing and comparing the results obtained and applying my WSD models to the WIC task.

6 Final Results

Table 4 shows the final parameters' configuration of my best model: it is able to reach **0.6698** accuracy on the test dataset (0.7125 on the validation one). Finally, I used the produced models onto the WIC task in a very easy way: given the two WSD outputs for a pair of sentences, if these outputs are the same sense key, then the ambiguous word is used with the same meaning in both sentences, else it is not. In Figure 4 I display some confusion matrices I computed for my major models. In Table 5 I point out and compare the test metrics achieved on the WIC task: as we can see, my best WSD model (BERT batch=8 retrained) is not also the best WIC model (BERT batch=8), that is so because the re-training procedure could have

overfitted the first one in favor of the WSD task. Moreover, it is evident how the results of this homework are extremely better than the first one (+11.33% improvement): this is a trivial demonstration of the effectiveness of *Transformers* and *BERT*. In conclusion, I think that the limitations of *Colab* resources and of the training dataset *SemCor_small* are the major causes of the non-optimality of the results.

7 Figures

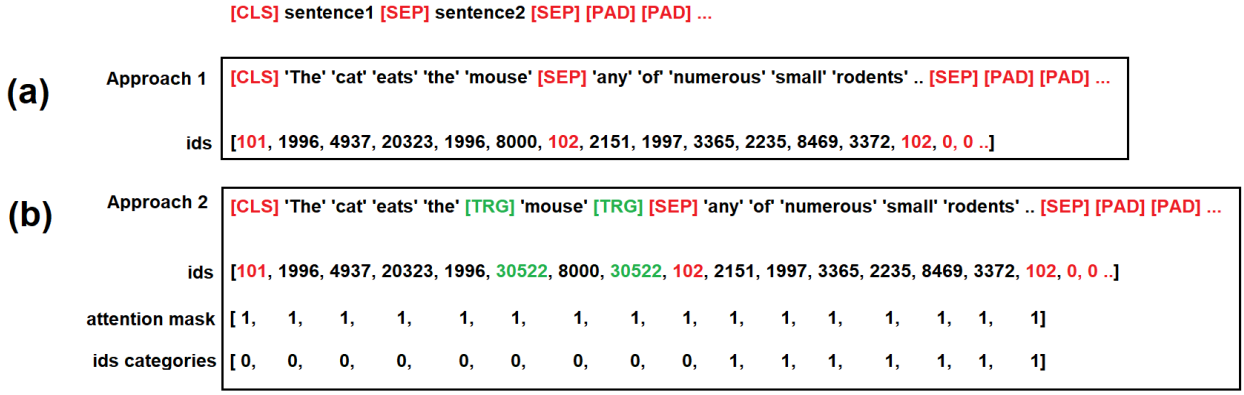

Figure 1: Sentence pre-processing first approach (without *[TRG]* token) (a) and second approach (with *[TRG]* token) (b)

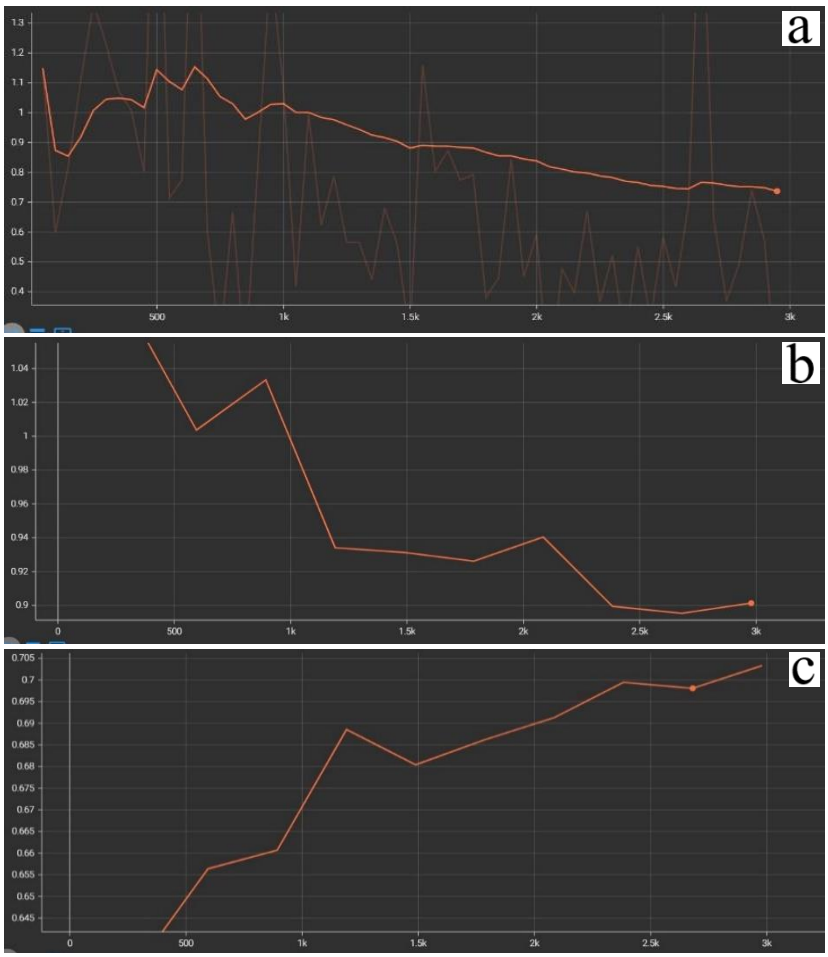

Figure 2: Train loss (a), validation loss (b) and validation accuracy (c) of *bert-base-uncased-batch=8*

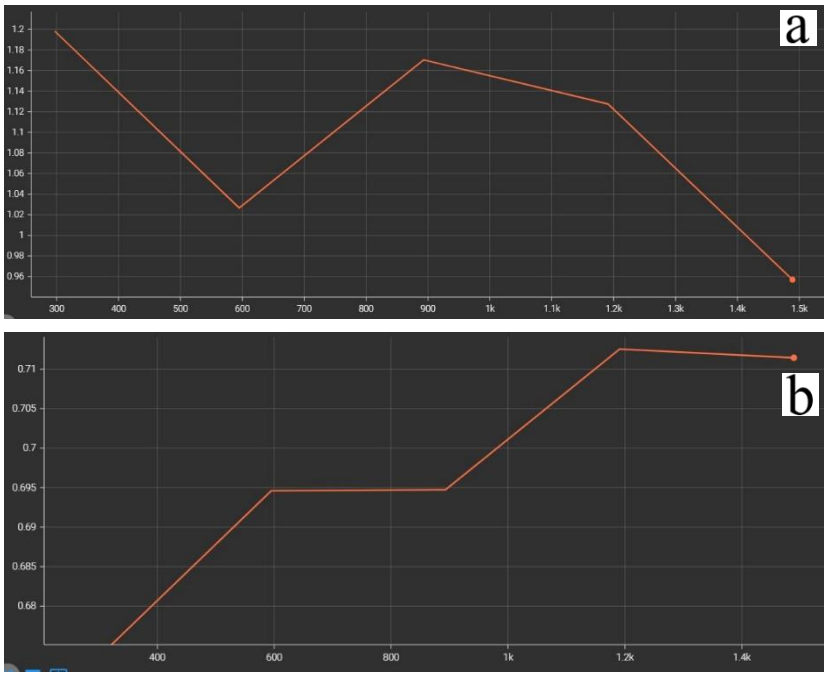

Figure 3: Validation loss (a) and validation accuracy (b) of *bert-base-uncased-batch=8 retrained*

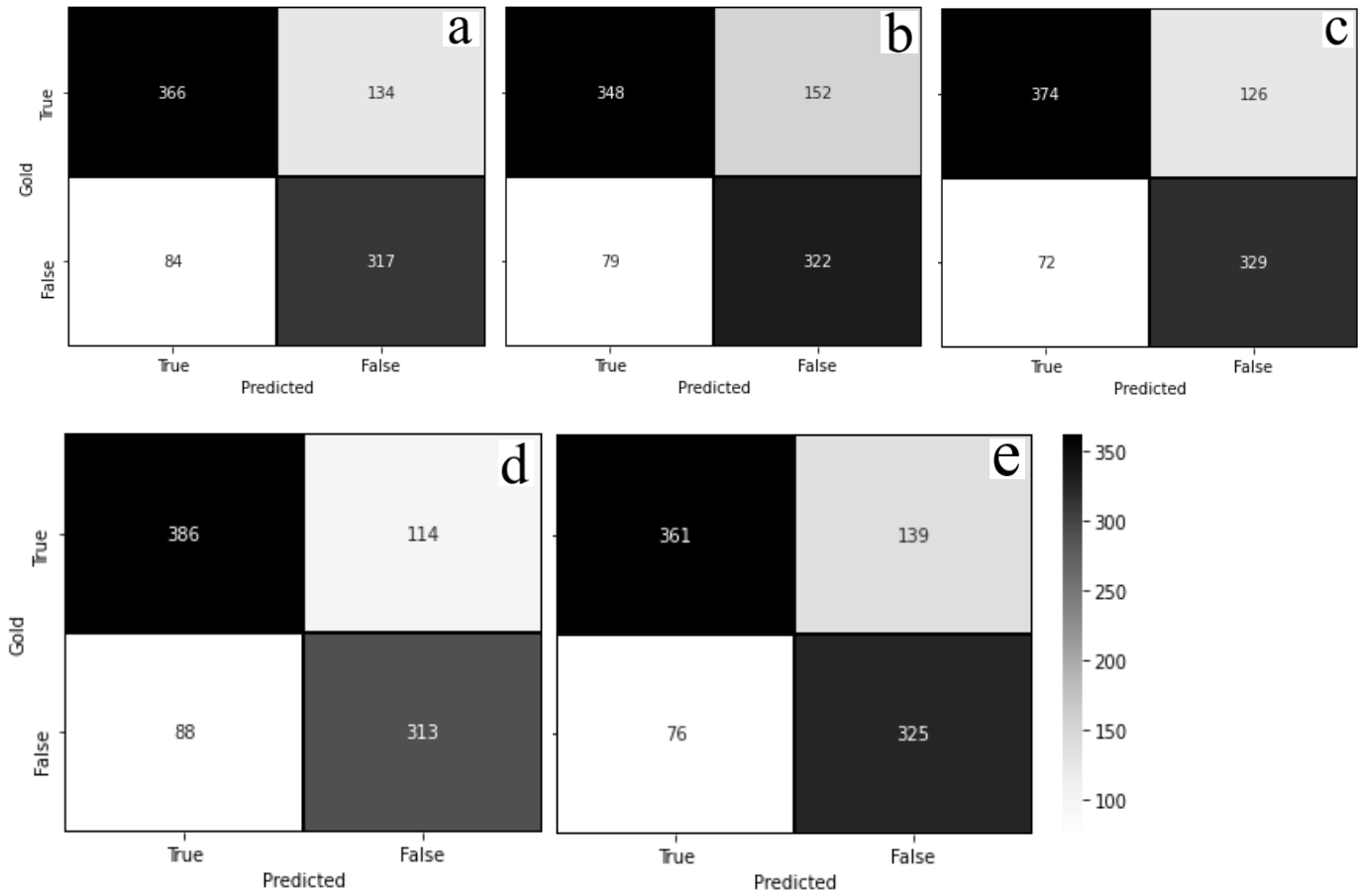

Figure 4: Confusion matrices of *bert-base-uncased batch=4* (a), *batch=8 dropout=0.2* (b), *batch=8 lr=3.5e-5*(c), *batch=8* (d) and *batch=8 retrained* (e) on the WIC task

8 Tables

Model	Loss	Accuracy
Random Baseline	-	0.1731
BERT (w/o TRG tokens)	0.9236	0.6526
BERT (with TRG tokens)	0.9022	**0.6543**
LARGE	1.2683	0.5377
LARGE WWM	1.5265	0.3158
LARGE WWM SQuAD	1.5397	0.3485

Table 1: Models' results on WSD task on the test dataset (Experimental Stage 1)

Model	Loss	Accuracy
BERT batch=8	0.8834	**0.6637**
BERT batch=8 dropout=0.2	0.9624	0.6304
BERT batch=8 layers=1	0.8994	0.6577
BERT batch=8 layers=4	0.9003	0.6538
BERT batch=8 lr=1e-4	1.1261	0.5480
BERT batch=8 lr=3.5e-5	0.8955	0.6559
BERT batch=8 retrained	1.0343	**0.6698**

Table 2: Models' results on WSD task on the test dataset (Experimental Stage 2)

Tested Parameter	Value (tested from X → to Y)
Batch size	2 → 64, crash above 8
Optimizer	AdamW
Learning Rate	2e-5 → 1e-4
Epsilon	1e-8 → 1e-6
Epochs	1 → 3
Validations per epoch	0 → 10
Dropout	0.0 → 0.3
Number of output layers	1 → 4
Layers dimensions	768 → 1024

Table 3: Parameters tuning and testing

Parameter	Value
Preprocess approach	With TRG tokens
Batch size	8
Optimizer	AdamW
Learning Rate	2e-5
Epsilon	1e-8
Epochs	1
Validations per epoch	10
Dropout	0.1
BERT	*bert-base-uncased*
Linear Layer1 in-out	768 → 384
Linear Layer2	384 → 1
Re-trained	Yes

Table 4: Best model's parameters

Model	Precision	Recall	Accuracy	F1 score	HW3 Improvement
Random Baseline	0.5	0.5	0.5	0.5	-
HW1 First Approach	0.6673	0.6670	**0.6669 (0.644 on secret test)**	0.6670	+ 0.00
HW1 Second Approach	-	-	0.6410	-	-
BERT batch=4	0.7581	0.7613	0.7580	0.7573	+ 0.0911
BERT batch=8	0.7808	0.7842	**0.7802**	0.7797	**+ 0.1133**
BERT batch=8 retrained	0.7633	0.7662	**0.7614**	0.7610	**+ 0.0945**

Table 5: HW1 and HW3 models' results on WIC task on the test dataset

9 References

Alessandro R., Jose C. C., and Roberto N. 2017. *Word Sense Disambiguation: A Unified Evaluation Framework and Empirical Comparison.* Università di Roma La Sapienza.

Ashish V., Noam S., Niki P., Jakob U., Llion J., Aidan N. G., Lukasz K., and Illia P. 2017. *Attention Is All You Need.* Google, and University of Toronto.

Boon Peng Y., Andrew K., and Eng Siong C. 2020. *Adapting BERT for Word Sense Disambiguation with Gloss Selection Objective and Example Sentences. + github.* Nanyang Technological University, Singapore.

George A. M., Richard B., Christiane F., Derek G., and Katherine M. 1993. *Introduction to WordNet: An On-line Lexical Database.*

Jacob D., Ming-Wei C., Kenton L., and Kristina T. 2019. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.* Google AI Language.

Luyao H., Chi S., Xipeng Q., and Xuanjing H. 2019. *GlossBERT: BERT for Word Sense Disambiguation with Gloss Knowledge.* Shanghai Key Laboratory of Intelligent Information Processing, Fudan University School of Computer Science, Fudan University 825 Zhangheng Road, Shanghai, China.

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.

Python Documentation. 2021. pickle - Python object serialization.

PyTorch Documentation. 2021. *PYTORCH DOCUMENTATION.*

PyTorch Lightning Documentation. 2021. *PYTORCH LIGHTNING DOCUMENTATION.*

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

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