Instructions for ACL 2023 Proceedings

Anonymous ACL submission

Abstract

This report shows the proposed architecture for the Word Sense Disambiguation task. The architecture is a combination of the usage of Transformer and POS Tags.

1 Introduction

Some words have different meanings, for example 'bank' could refer to 'the land alongside a river or lake' or 'a financial institution' therefore the word is ambiguous.

Word Sense Disambiguation (WSD) is a fundamental challenge in Natural Language Processing, it refers to the task of determining the correct sense of an ambiguous word given a context. In this report I present a possible solution, and the step to reach it, to a WSD task combining Transformer and POS Tags.

2 Preprocessing

2.1 The Data

Two datasets were given, one with coarse-grained senses and the other with fine-grained senses.

The main difference is that the fine-grained senses are more specific than the coarse ones.

2.2 Sentence preprocessing

Since a large amount of samples had more than a word to disambiguate, I extended the number of sentences in order to have one target word per sentence. Therefore, if the sentence 's' had 3 target words, the Dataset will have three equal sentences 's', each with one target word associated.

2.3 Word preprocessing

In this step I preprocess the words in a very simple way:

- I lowercased the text in order to have data uniformity and better generalization.
- I removed all the spaces in every word in order to have a single entity for all the words. For example, 'New York' becomes 'NewYork' and here I have a single entity instead of two.
- I added two special tokens, [//START//] and [//END//] at the beginning and end of the target word. Since the tokenization process could produce meaningful subwords, the target word could have multiple positions. Therefore with the special tokens is easier to locate all the subwords.
- I replaced '[ ]' with '( )' in order for the Tokenizer to not mistake simple square brackets with the special tokens.
- I cleaned all the words by removing the non-ASCII characters in order to have certain compatibility with Tokenizers.

3 Transformers

Transformers play an important role in the architecture of my model. They are a type of neural network architecture first introduced in the paper 'Attention Is All You Need'. They are known for the use of multi-head attention and self-attention mechanism, which are very useful to make them efficiently handle sequence of variable length and capture complex relationships between words in a text.

A Transformer is typically composed of two blocks: encoder and decoder. The first is responsible for receiving an input and building representation of it, the second use the encoder's representation along with other inputs to generate a target sequence.

The chosen transformers for this task are BERT and RoBERTa, which uses both Encoder-only models (that are suited for tasks that require understanding of the input).

Bidirectional Encoder Representations from Transformers (BERT) is pre-trained on a large corpus of text data using a masked language modelling (MLM) objective and learns to predict relationships between pairs of sentences (Next Sentence Prediction NSP).
A Robustly Optimized BERT Pretraining Approach (RoBERTa) is based on the architecture and principles of BERT. The main differences are that RoBERTa simplifies the pretraining objective by removing the NSP task and was trained on a much larger dataset.
The chosen models were 'bert-cased' for BERT and 'roberta-base' for RoBERTa, which are both case-sensitive and therefore give me better performance encoding and locating the special tokens since the rest of words are lower-cased.

3.1 Tokenizer

Tokenizers are fundamental because they are responsible for translating text into data that can be processed by the Transformer model. A tokenizer splits the inputs into tokens, maps each token to an integer and adds additional inputs that may be useful to the Transformer model.
The chosen tokenizers were the ones related to BERT and RoBERTa models.

4 Model

4.1 First Attempt

For this WSD task I started with the architecture in Figure 1, in which the tokenized input is passed to a transformer model (in this case BERT, 'bert-cased') and the output is fed to a classifier after a dropout is applied.
This approach was completely wrong because I tried to classify each word, given the label 0 if it was not to disambiguate otherwise the integer that corresponded to the right sense. Obviously, the results were very poor, giving less than 1% of accuracy.

4.2 Second Attempt

The second approach is shown in Figure 2 and the main differences with the First Attempt are that this time I took the embeddings only of the target word and applied a mask to the output of the classifier. Basically, the mask is a binary list of all possible candidates. Therefore, when the mask is applied to the logits, only the candidates related to the target word are taken into account (this is done by giving highly negative values to the others).

4.3 Final Attempt

The third and final architecture is shown in Figure 4. This is a combination of the Second Attempt with the use of POS tags, which are linguistic labels used to indicate the grammatical category of each word in a text.
Taking inspiration from what I did in Homework 1, from the POS tag of the target word I created an embedding of the same size of the word embeddings of Second Attempt.
Each POS tag corresponds to a specific value.
Then the embeddings from the Transformer model and the pos embeddings are combined by taking the mean and the rest is equal to Second Attempt.

5 Training Setup

The model is trained using the cross-entropy loss and the optimizer is AdamW, which is Adam with weight decay. I tried different values of learning rate (0.001, 0.0001, e-5, 0.01) and weight decay coefficient (0.001, 0.01, 0.1) but it turned out that the default ones were better.

6 GPU Limitations

I trained my model on the base version of Google Colab that offers a not so powerful GPU, and it often crashed after just one epoch because the model and dataset were too heavy.
In order to overcome this issue, I used various strategies.
The most important one was the Freeze, it consists on preventing the layers of the Transformer model from being updated during the training, essentially the weights and gradients remain unchanged. But this procedure comes at a cost, in fact whenever I froze all the layers of the Transformer model, the performances drastically went down, therefore I

decided to unfreeze three layers and I got better results than before.
Another strategy that I used was to take the embeddings of the target word only from the last layer of the Transformer model in order to lighten the computational complexity.
Also, I trained the model for few epochs, maximum 10 but usually 5 because the performances did not improve after the fifth.
First, I set up a relatively small limit (150) for the truncation in tokenization and for the maximum length of a sentence to have in order to be added to the Dataset. Despite having a faster training, with this method I did not take into account a lot of candidates therefore I decided to raise the value to 300.

7 Overfitting

In order to avoid overfitting, I used two techniques:

- Dropout: I put a dropout (with value=0.2) layer after each embedding layer, which allows to randomly ignore network units with a 20% probability.
- Early stopping: the training stops after 2 epochs of patience whenever the validation loss starts to increase.

8 Experiments

All the experiments are evaluated through accuracy and can be seen in Table 1, all the Tests (except the fourth) are done by using the BERT model and the first 5 Tests uses coarse-grained candidates.

8.1 Test 1: First Attempt

As it was expected the first version of the model produces very poor results. This is because the approach was completely wrong, in fact the non-target words had a wrong label (all of them the same) and even if the target word was correctly classified, all the rest are not.

8.2 Test 2: Second Attempt

In this case it is possible to see how much the performances can increase if a better approach (classify only the target word) is chosen, in fact despite the GPU limitations and consequent adaptations, the model still reached a good baseline.

8.3 Test 3: Final Attempt

By adding the use of POS tags as POS embeddings the model reaches its maximum accuracy, meaning that the combined embeddings better represent the target word.
The trend of the loss is shown in Figure 3.
After Test 3 all the Tests are done with the final version of the model.

8.4 Test 4: RoBERTa

RoBERTa performs slightly worse than BERT, it could mean that the latter is better suited for this WSD task since it uses the NSP task.

8.5 Test 5: Lemmas

In this experiment I decided to use lemmas (preprocessed as words), instead of words and got slightly worse results. This is means that despite the words are more specific, lemmas are still a valid option for a WSD task.

8.6 Test 5: Fine

This experiment shows that the model performs way better on coarse candidates meaning that lower specificity is better suited for these Transformers.
This great difference of performances can be explained on the fact that fine-grained senses are much more than the coarse ones, and some of them are very similar, therefore it is easier to misclassify.

9 Conclusions and Future Improvements

With the proposed model I tried to show how a good accuracy can be achieved despite the limits of a GPU if the right approach is taken.
Further improvements can be made by using advanced GPU.
Another possible improvement could be combining also pre trained word embeddings (e.g., from Glove) and combining different architecture with the Transformer model, such as LSTMs or CNNs.

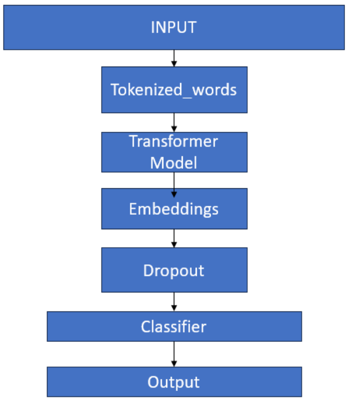

Figure 1: First Attempt

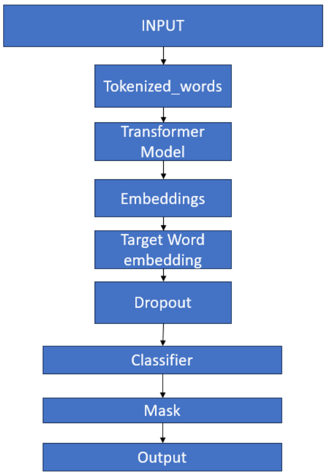

Figure 2: Second Attempt

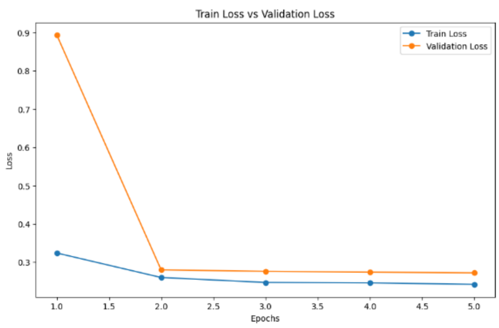

Figure 3: Loss

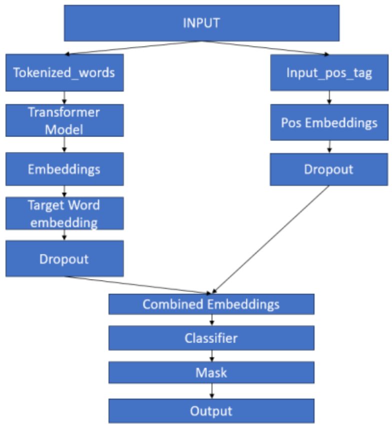

Figure 4: Last Attempt

Model		**Accuracy (%)**
First Attempt		0.7
Second Attempt		85
Last Attempt (BERT)		89.79
L.A. + RoBERTa		89.15
L.A. + lemmas		88.78
L.A. + fine senses		75.7

Table 1: Performances