Word Sense Disambiguation
Homework 2
Multilingual Natural Language Processing - 2023

Sameer Ahmed
Sapienza University of Rome
ahmed.2047250@studenti.uniroma1.it

1 Introduction

Determining the correct meaning or "sense" of a word in a specific context is a challenge in Natural Language Processing (NLP), and it is known as word sense disambiguation (WSD). Since many words in natural language have numerous meanings, it can be difficult to determine the intended meaning. WSD aims to resolve this issue. To better understand the challenge of WSD, let's consider the phrase "light bulb." Without any context, it could refer to an object that produces light when connected to electricity. However, with additional context, the meaning can change. In the sentence "He had a brilliant idea; a light bulb went off in his head," the term "light bulb" is used metaphorically to represent a sudden understanding or realization. The context surrounding the phrase disambiguates its sense. The approach that I followed to solve this issue, is supervised, and focused on two different models: a baseline model utilizing a normal Bi-LSTM architecture with pre-trained word embeddings (GloVe), and a more advanced model employing the RoBERTa transformer architecture. By comparing these two approaches, I am aimed to assess the effectiveness and performance of each model in the challenging task of word sense disambiguation.

2 Approach

To solve word sense disambiguation (WSD) we have 4 primary sorts of approaches. Supervised, Semi-Supervised, Un-Supervised, and Dictionary and Knowledge-Based approach. Also, to solve WSD we have a dataset with 2 different files, coarse-grained and fine-grained. Here I am using the coarse-grained file to solve WSD, and for that, I decide to use supervised learning (as shown in Fig 1) to solve the WSD task. I am taken 2 different supervised approaches, A baseline approach a normal Bi-LSTM architecture with pre-trained word embeddings (GloVe), and a more advanced model employing the RoBERTa transformer architecture, we will discuss more extensively in the model architecture section.

3 Preprocessing

In a coarse-grained file, we have different inputs (instance_ids, words, lemmas, pos_tags, and candidates) and the output is (senses) shown in Fig 2. To make a program as much as simple, I only take inputs as (words), and outputs as (Senses) shown in Fig 3. Most of the words are in different cases, So I convert all the words into the same case (lower). Then we need to make a list of both words and senses of training data. It will help us to convert into numbers and for out-of-vocabulary (OOV) problem. We have 2 models, So the next step for preprocessing is different for both models.

3.1 For Bi-LSTM Architecture with pre-trained word embeddings (GloVe)

For this model, the main and crucial part is to build ‘senses keys’. We have inputs (words) as well as outputs (senses), but the problem is that the length of the words is not equal to the length of the senses (as shown in Fig 4). In multi-class classification models, it is necessary for the input and output lengths to match to train the model effectively, because each input is associated with a single

output class label. The goal is to classify each input into one of several predefined classes. So, for that, I created a dummy sense key '__PADDING__', This sense key is assigned for those words, that are not homonyms (shown in Fig 4). Now, we have input (words) and output (senses), but most of the input sequences are not of the same length so for that I am applying the padding technique. For the padding technique, here I am using the same '__PADDING__' key for inputs/outputs. Why do I use the same '__PADDING__' key for both problems (Handling Words without Sense Key and Padding for Variable Length Inputs)? because in the loss function (categorical cross-entropy) we only ignore one key, not two, and we need to ignore both things. This thing helps the model to only learn sense keys not '__PADDING__' sense key. Now the last thing is to convert words and senses into numbers, using the list of senses and words (Build using training data), It also resolves the OOV issue. After this, both data will be passed through the Dataset class and then to divide into batches we use Dataloader class.

3.2 For RoBERTa Transformer

We have inputs (words) in lowercase, and now it's time to apply tokenization (shown in Fig 5). The tokenizers for Hugging Face models are pre-trained, meaning they are generated from the training set of the algorithm in advance. When using the RoBERTa model, it is crucial to tokenize the input text using the appropriate tokenizer. So here I am taking RoBERTa 'roberta-base'. In the case of 'roberta-base', it utilizes a subword tokenization process known as BPE (Byte-Pair Encoding) variant. This tokenizer divides words into subword units based on their frequency of occurrence in a corpus. This tokenization is only applied to the inputs by using the Datasets class. After tokenization, the output (senses) length will change, and to handle this, an align_label() function is used to assign keys only to homonyms, while non-homonyms are assigned a '__PADDING__' value of -100 (According to the previous model). This helps in maintaining the length of the list of sense keys (output). These -100 values will be ignored in training (By the categorical cross-entropy function). These inputs and outputs are divided into batches by using Dataloader class.

4 Model Architecture

4.1 Baseline Model

The baseline model architecture is based on Bidirectional LSTM (Schuster and K. Paliwal, 1997) model (shown in Fig 6). My approach makes use of the power of Bidirectional Long Short-Term Memory (Bi-LSTM) networks to gather contextual data from both the past and future words in a sentence. One LSTM layer of the Bi-LSTM component processes the input sequence forward, and the other layer processes it backward. Due to its bidirectional nature, the model may provide a complete representation of the input sequence that takes both the preceding and subsequent context into account. GloVe pre-trained embedding (Pennington et al., 2014) are used to represent the input for the Bi-LSTM. Based on the co-occurrence statistics of words in large corpora, GloVe embeddings offer dense vector representations of words. In this approach, we make use of 300-dimensional GloVe embeddings, with fixed features by assign (freeze = True), which have been shown to be successful at capturing word semantic relationships. To prevent overfitting, a dropout of 0.4 is added to the main BiLSTM's output. A fully connected layer with a softmax activation function is used to the BiLSTM output to classify the data.

4.2 Transformer Architecture (RoBERTa)

The second model, here I am using is RoBERTA Transformer (Stoyanov et al., 2019) (shown in Fig 7), Although Bidirectional LSTM (BiLSTM) models are effective at capturing contextual dependencies in a sequential manner by processing input in both forward and backward directions. The reason behind that is in WSD, BiLSTM with pre-trained word embedding (Glove) give 'same' embedding for same words. Although the contexts of words are different, Suppose I have two sentences: "He didn't receive fair treatment" and "Fun fair in New York City this summer. In both sentences the word 'fair' has two different meaning according to the context, but (Glove) embedding give both 'fair' words embedding same. The transformer-based model RoBERTa, on the other hand, is capable of effectively capturing contextual data. Transformers, like RoBERTa, use self-attention mechanisms to recognize relationships between words in a sentence, which improves their ability to effectively recognize contextual

information. RoBERTa can therefore learn contextualized word embeddings that change depending on the precise context in which the word appears, making it better suitable for tasks like word sense disambiguation. Here I am using pretrained RoBERTa model ("roberta-base") with fine-tuned, Then I use dropout layer of 0.2 to prevent overfitting then a fully connected layer with a softmax activation function is used to the RoBERTa output to classify the data.

5 Training

In training, I trained (Baseline model) with Adam optimizer, and the learning rate was 0.001. On the other hand, the RoBERTa model was trained from AdamW optimizer, with a learning rate of 5e-5. The whole training was done by google colab GPU. I tried categorical cross-entropy loss in both models with ignore index (for padding).

6 Results

Results are evaluated with respect to the Accuracy (Shown in Table 1). Accuracy is often used as a metric to assess the performance of WSD systems since it gives a clear indication of how effectively the model can identify the intended sense. The code to calculate the accuracy, I picked code from the `evaluate.py` file. Although, the accuracy is also calculated from seqeval library. On the baseline model, the accuracy of the validation data gradually increased up to 73% on the 33 epochs (Shown in Fig 8). After 33 epochs the model accuracy of validation data stuck to 73%. I also tried to evaluate the model on testing data, the accuracy of the model was 71.2 % on testing data. However, the RoBERTa transformer model gives the best results compared to the previous model. Only in 1 epoch, the model gives 82.8% of accuracy in validation data. The accuracy of the model increased to 90% on the 6th epoch (shown in Fig 9), The model accuracy will also be measured for testing data and this time also the model gives 89% accuracy in testing data as well. I tried BERT, BART, and ALBERT transformers as well. But the accuracy was not met given by RoBERTa transformer.

7 Conclusion

This paper presented a multiclass classification model for WSD based on BiLSTMs with pre-trained word embedding (Glove) and RoBERTa transformer model. We show that the RoBERTa transformer model works better compared to the Baseline model because it is capable of effectively learning contextualized word embeddings that change depending on the precise context in which the word appears. We also enhance our model accuracy by using models like (GlossBERT, EWISER, BEM, etc.)

References

Jeffrey Pennington, Richard Socher, Christopher Manning. 2014. GloVe: Global Vectors for Word Representation. *Association for Computational Linguistics*,1532–1543. https://aclanthology.org/D14-1162.pdf

Mike Schuster and Kuldip K. Paliwal. 1997. Bidirectional Recurrent Neural Networks. *International Conference on Neural Networks (ICNN)*. https://maxwell.ict.griffith.edu.au/spl/publications/papers/ieeesp97_schuster.pdf

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *Conference on Empirical Methods in Natural Language Processing (EMNLP)*. https://arxiv.org/abs/1907.11692v1

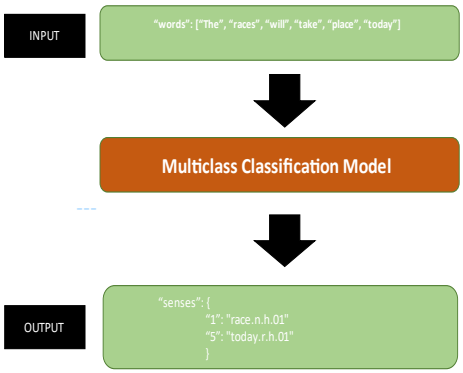

Figure 1: Flow diagram of supervised approach, discussed in section 1 (approach).

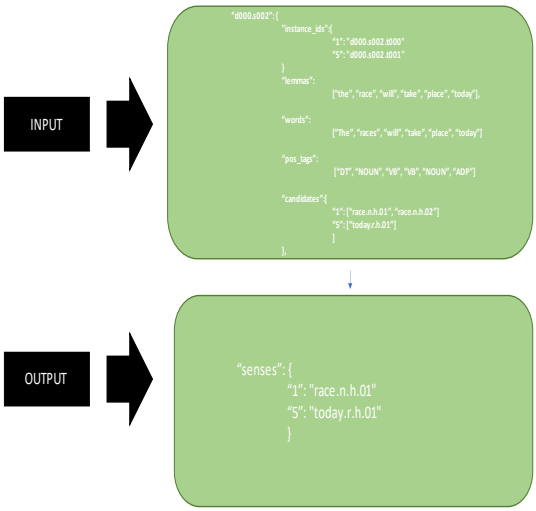

Figure 2: In a coarse-grained data file, we have different inputs (instance_ids, words, lemmas, pos_tags, and candidates) and the output is (senses).

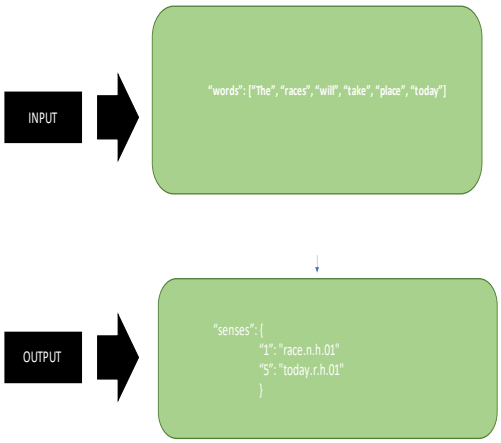

Figure 3: According to the coarse-grained data file, we have different inputs and outputs, here in this paper I am only taking words as (input) and senses as (output).

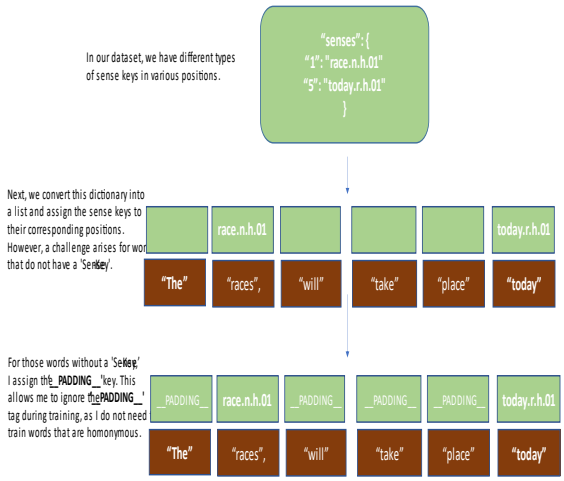

Figure 4: In multi-class classification models, it is necessary for the input and output lengths to match to train the model effectively, because each input is associated with a single output class label. The goal is to classify each input into one of several predefined classes. So, for that, I created a dummy sense key '__PADDING__', This sense key is assigned for those words, that are not homonyms. This will help us to make an equal length of output as an input.

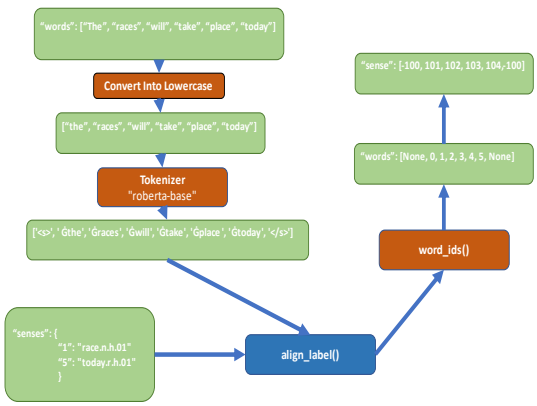

Figure 5: The flow diagram shows how we convert 'words' into subwords by using the 'roberta-base' tokenizer. Tokenizer changes the length of the 'words', and this thing also affects the output (senses), for that here align_label() function is used to make senses equal in length according to the words.

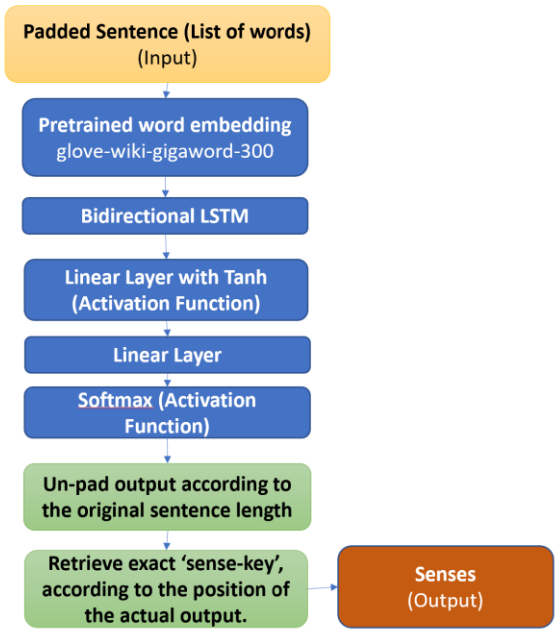

Figure 6: Architecture of the (Baseline model) – Bi-LSTM with pre-trained word embedding (Glove).

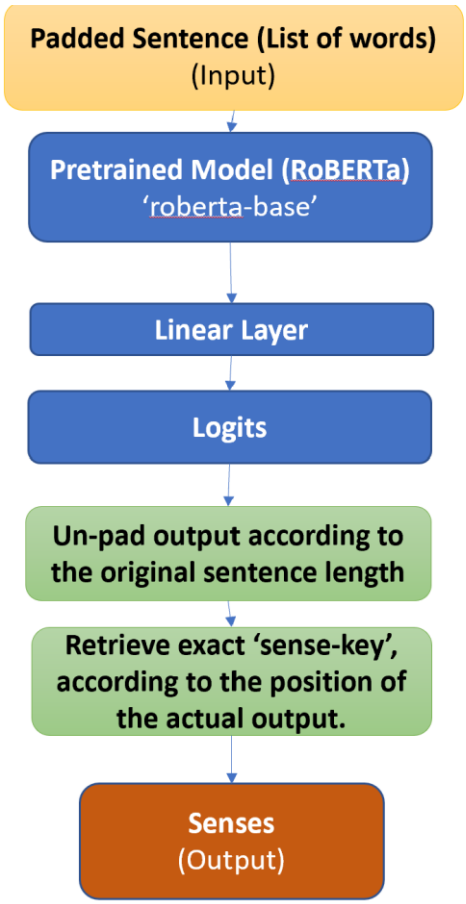

Figure 7: Architecture of the RoBERTa Transformer model.

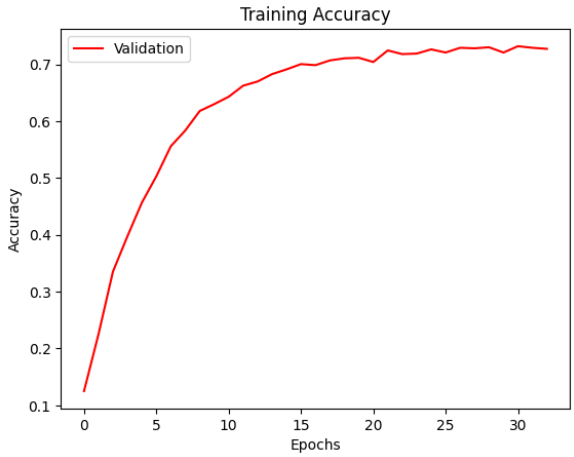

Figure 8: The baseline model work quite better it gives approximately 73% accuracy on the 33rd epoch. The figure shows the accuracy of the validation dataset of coarse-grained.

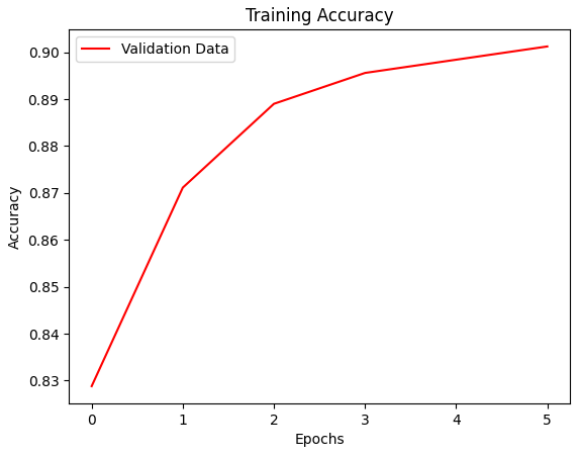

Figure 9: RoBERTa Transformer model gives high-performance accuracy of about 90% accuracy on the 6th epoch. The figure shows the accuracy of the validation dataset of coarse-grained.

Models	Accuracy	
	Validation Data	Test Data
Baseline Model (Bi-LSTM + Pretrained word embedding – Glove)	72.7 %	71.2 %
RoBERTa Transformer	**90.1 %**	**89.0%**

Table 1: Results of the different models tested on the development and testing dataset.