Coarse-Grained WSD – Homework 2

Multilingual Natural Language Processing

SAPIENZA
UNIVERSITÀ DI ROMA

Presented To : Prof. Roberto Navigli
Presented By : Sameer Ahmed

Word Sense Disambiguation (WSD)

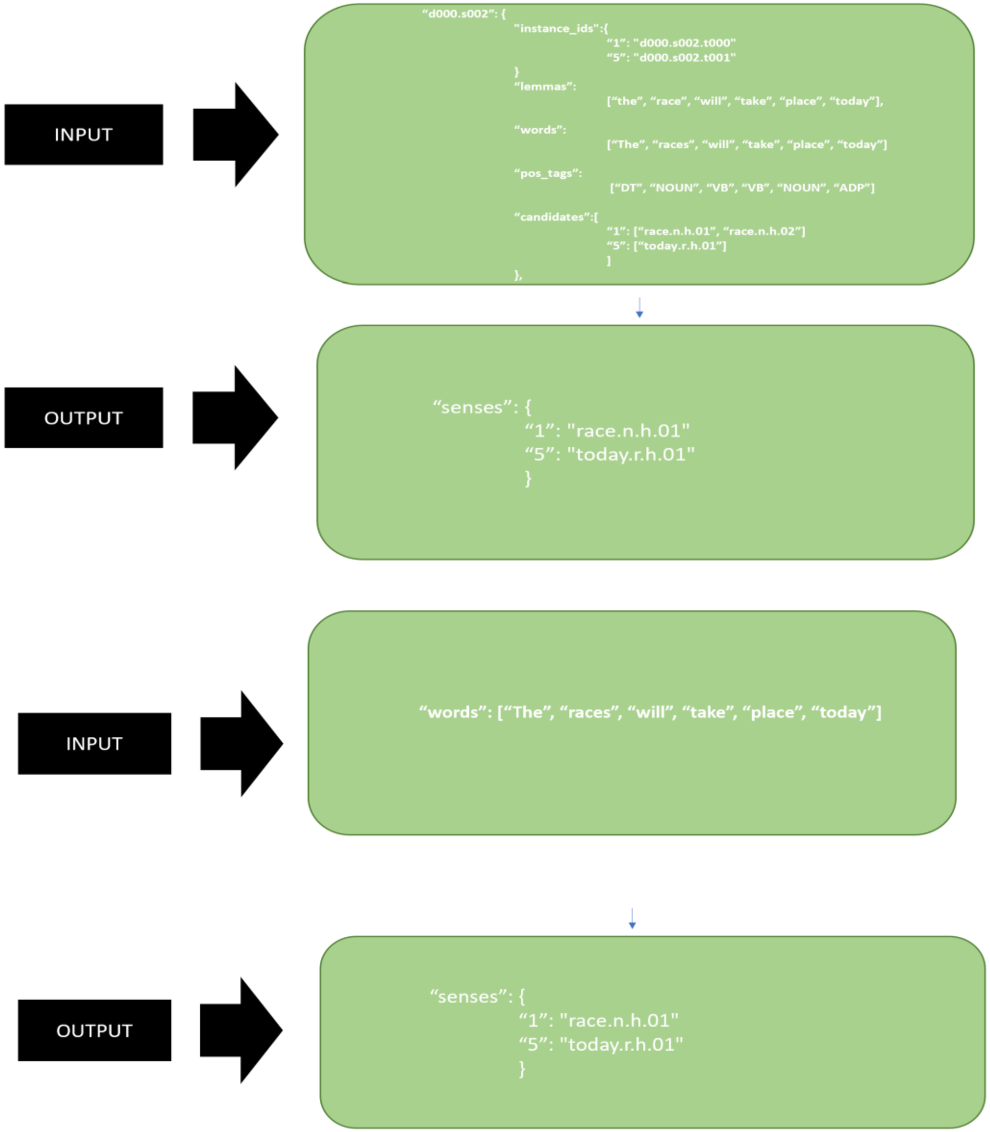
Determining the correct meaning or "sense" of a word in a specific context is a challenge in Natural Language Processing (NLP), and it is known as word sense disambiguation (WSD). Since many words in natural language have numerous meanings, it can be difficult to determine the intended meaning. WSD aims to resolve this issue.

For Example:

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.

Data Preparation

Here I am using the coarse-grained file to solve WSD. In a coarse-grained file, we have different inputs (instance_ids, words, lemmas, pos_tags, and candidates) and the output is (senses). To make a program as much as simple, I only take inputs as (words), and outputs as (Senses)



INPUT

```
"d000.s002": {
        "instance_ids":{
                "1": "d000.s002.t000"
                "5": "d000.s002.t001"
        }
        "lemmas":
                ["the", "race", "will", "take", "place", "today"],

        "words":
                ["The", "races", "will", "take", "place", "today"]

        "pos_tags":
                 ["DT", "NOUN", "VB", "VB", "NOUN", "ADP"]

        "candidates":[
                "1": ["race.n.h.01", "race.n.h.02"]
                "5": ["today.r.h.01"]
                ]
        },
```

OUTPUT

```
"senses": {
        "1": "race.n.h.01"
        "5": "today.r.h.01"
        }
```

INPUT

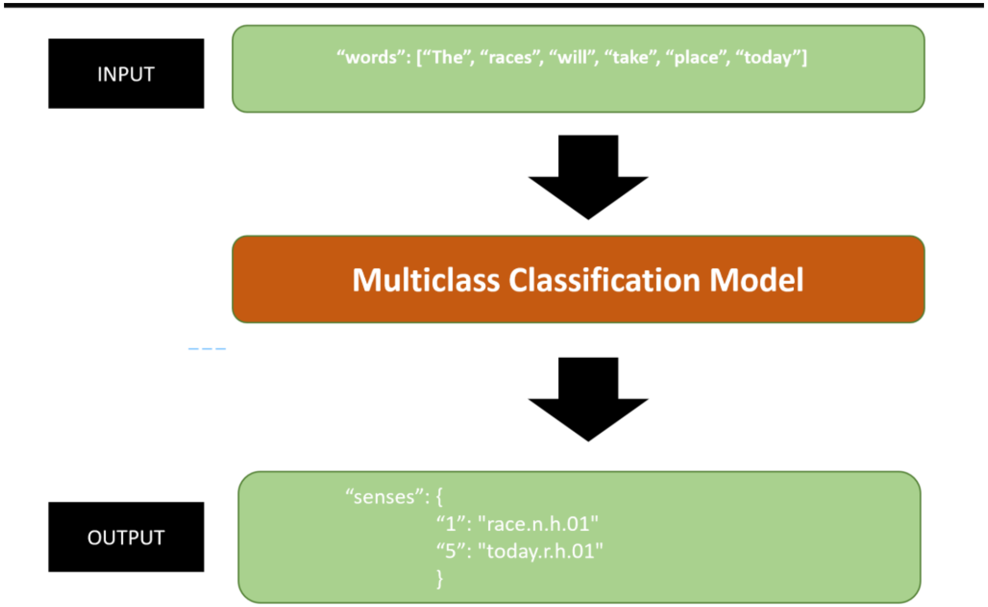
```
"words": ["The", "races", "will", "take", "place", "today"]
```

OUTPUT

```
"senses": {
        "1": "race.n.h.01"
        "5": "today.r.h.01"
        }
```

Flow Diagram

This flow diagram illustrates the complete process of the WSD task.



INPUT

“words”: [“The”, “races”, “will”, “take”, “place”, “today”]

Multiclass Classification Model

OUTPUT

```
“senses”: {
            “1”: "race.n.h.01"
            “5”: "today.r.h.01"
            }
```

Model Approach

In this task, I am considering two different approaches:

- Baseline Model (Bidirectional LSTM + GloVe pre-trained embedding)
- Transformer Architecture (RoBERTa)

Baseline Model

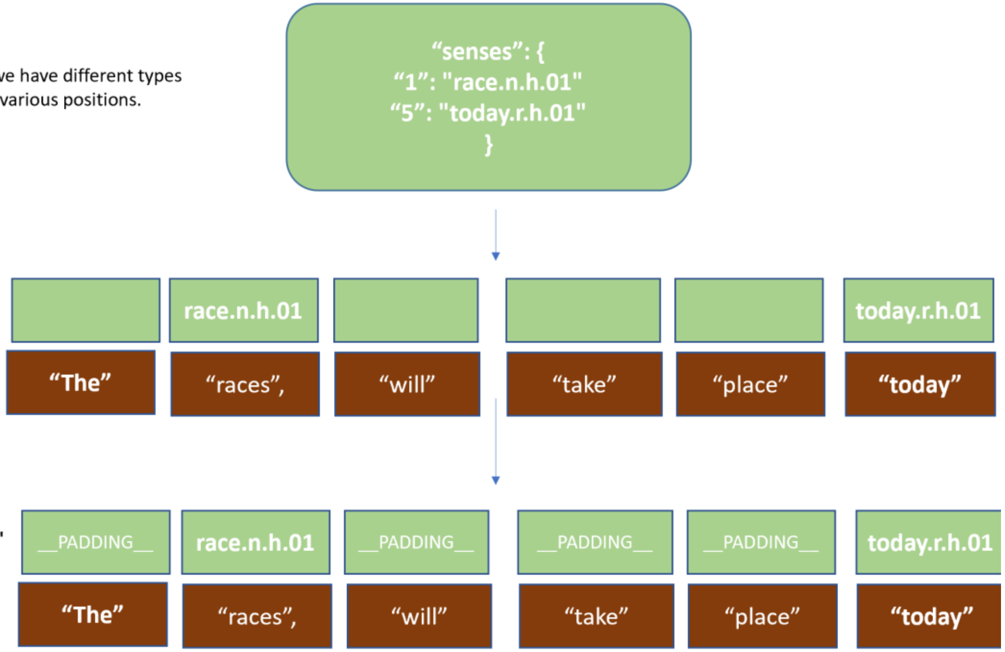
The approach described utilizes a Bidirectional Long Short-Term Memory (Bi-LSTM) network architecture to capture contextual information from both preceding and subsequent words in a sentence. The model consists of two LSTM layers, with one processing the input sequence forward and the other processing it backward. Pre-trained GloVe embeddings, which provide dense vector representations of words based on co-occurrence statistics, are used as input for the Bi-LSTM. The GloVe embeddings have 300 dimensions and are kept fixed during training. To prevent overfitting, a dropout of 0.4 is applied to the Bi-LSTM's output. Finally, a fully connected layer with softmax activation is used to classify the data.

Preprocessing Data

1. Most of the words are in different cases, So I convert all the words into the same case (lower).
2. 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.
3. The model aims to classify words into senses, but the lengths of the words and senses don't match to address this, a dummy sense key 'PADDING' is created for words that are not homonyms. Padding technique is applied to handle variable length inputs, using the same 'PADDING' key for both inputs and outputs.
4. The use of the same key helps in the loss function (categorical cross-entropy) by ignoring only one key instead of two. This approach allows the model to focus on learning sense keys and not the 'PADDING' sense key.
5. Words and senses are converted into numbers using pre-built lists generated from the training data, resolving out-of-vocabulary (OOV) issues.

Preprocessing Data (Cont.)

In our dataset, we have different types of sense keys in various positions.



“senses”: {
“1”: "race.n.h.01"
“5”: "today.r.h.01"
}

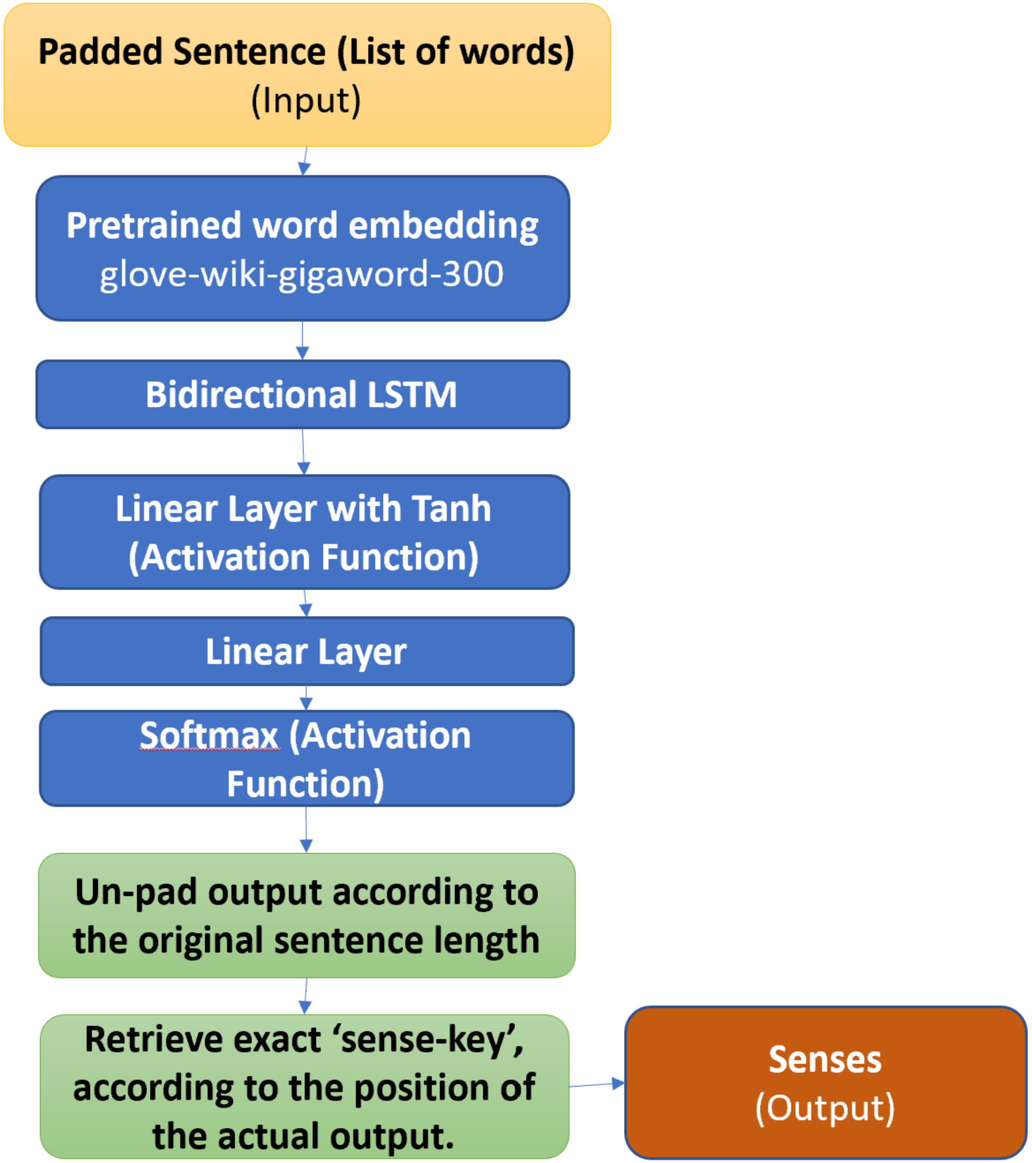
Next, we convert this dictionary into a list and assign the sense keys to their corresponding positions. However, a challenge arises for words that do not have a 'Sense-Key'.

	race.n.h.01				today.r.h.01
“The”	“races”,	“will”	“take”	“place”	“today”

For those words without a 'Sense-Key,’ I assign the **‘__PADDING__'** key. This allows me to ignore the **‘__PADDING__'** tag during training, as I do not need to train words that are homonymous.

__PADDING__	race.n.h.01	__PADDING__	__PADDING__	__PADDING__	today.r.h.01
“The”	“races”,	“will”	“take”	“place”	“today”

Flow Diagram of the Model


Padded Sentence (List of words)
(Input)
Pretrained word embedding
glove-wiki-gigaword-300
Bidirectional LSTM
Linear Layer with Tanh
(Activation Function)
Linear Layer
Softmax (Activation
Function)
Un-pad output according to
the original sentence length
Retrieve exact 'sense-key',
according to the position of
the actual output.
Senses
(Output)

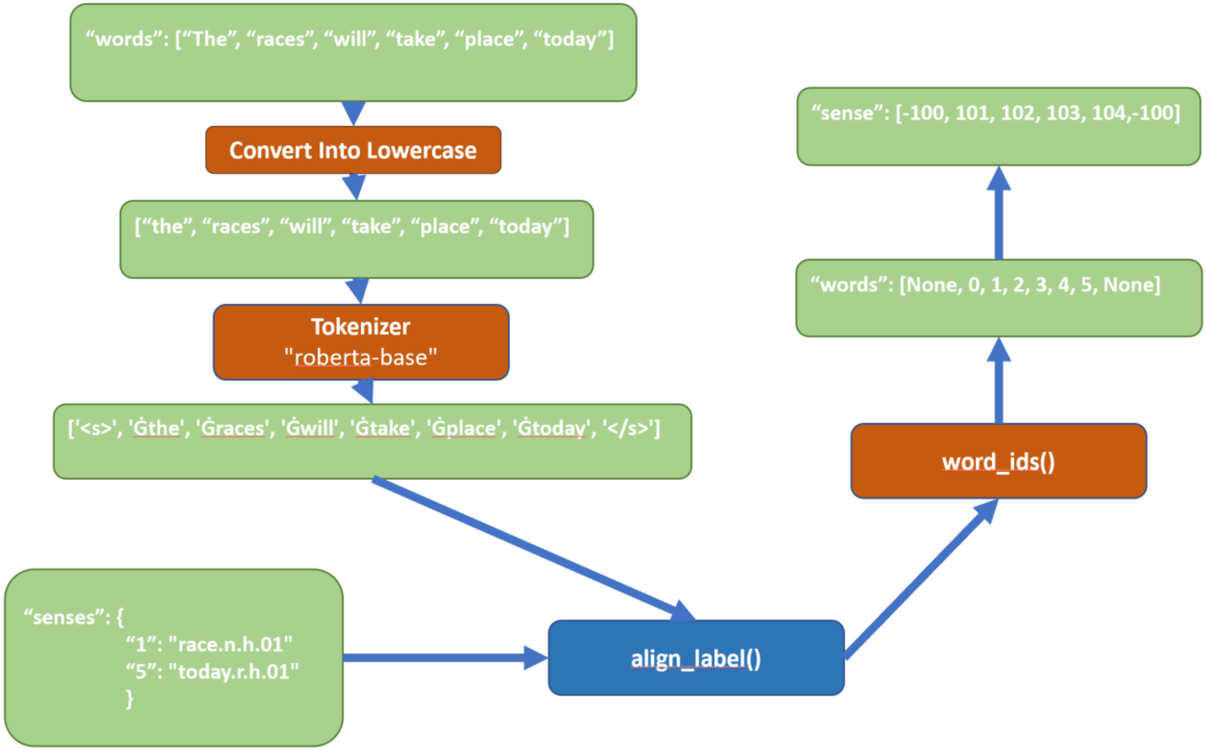
Transformer Architecture (RoBERTa) Model

- The second model, here I am using is RoBERTA Transformer, 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.

Preprocessing Data

1. Most of the words are in different cases, So I convert all the words into the same case (lower).
2. 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.
3. Inputs (words) are in lowercase and need to be tokenized. Tokenization is crucial for RoBERTa models, and the 'roberta-base' tokenizer is used. 'roberta-base' utilizes subword tokenization (BPE variant) that divides words into subword units based on frequency. Tokenization is applied to inputs using the Datasets class.
4. Tokenization changes the length of the output (senses). An align_label() function is used to assign keys only to homonyms, while non-homonyms are assigned 'PADDING' value (-100). This maintains the length of the sense key list for outputs.
5. The -100 values are ignored during training with categorical cross-entropy loss.

Preprocessing Data (Cont.)



“words”: [“The”, “races”, “will”, “take”, “place”, “today”]

Convert Into Lowercase

[“the”, “races”, “will”, “take”, “place”, “today”]

Tokenizer
"roberta-base"

['<s>', 'Ġthe', 'Ġraces', 'Ġwill', 'Ġtake', 'Ġplace', 'Ġtoday', '</s>']

```
“senses”: {
        “1”: "race.n.h.01"
        “5”: "today.r.h.01"
        }
```

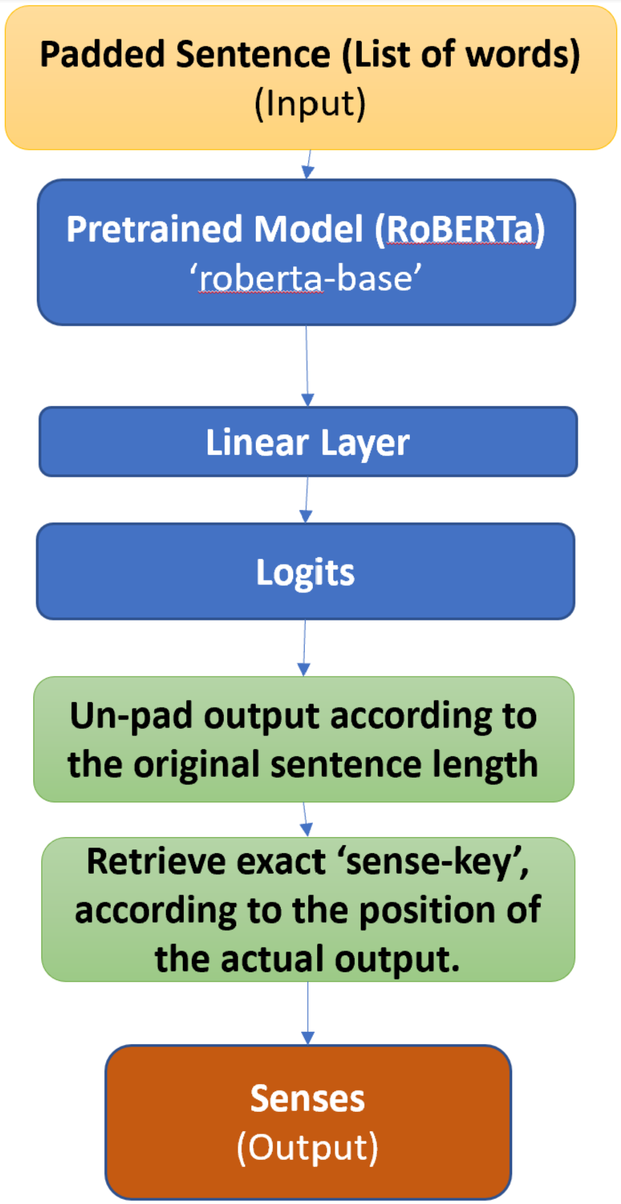
align_label()

word_ids()

“words”: [None, 0, 1, 2, 3, 4, 5, None]

“sense”: [-100, 101, 102, 103, 104,-100]

Flow Diagram of the Model



Padded Sentence (List of words)
(Input)

Pretrained Model (RoBERTa)
'roberta-base'

Linear Layer

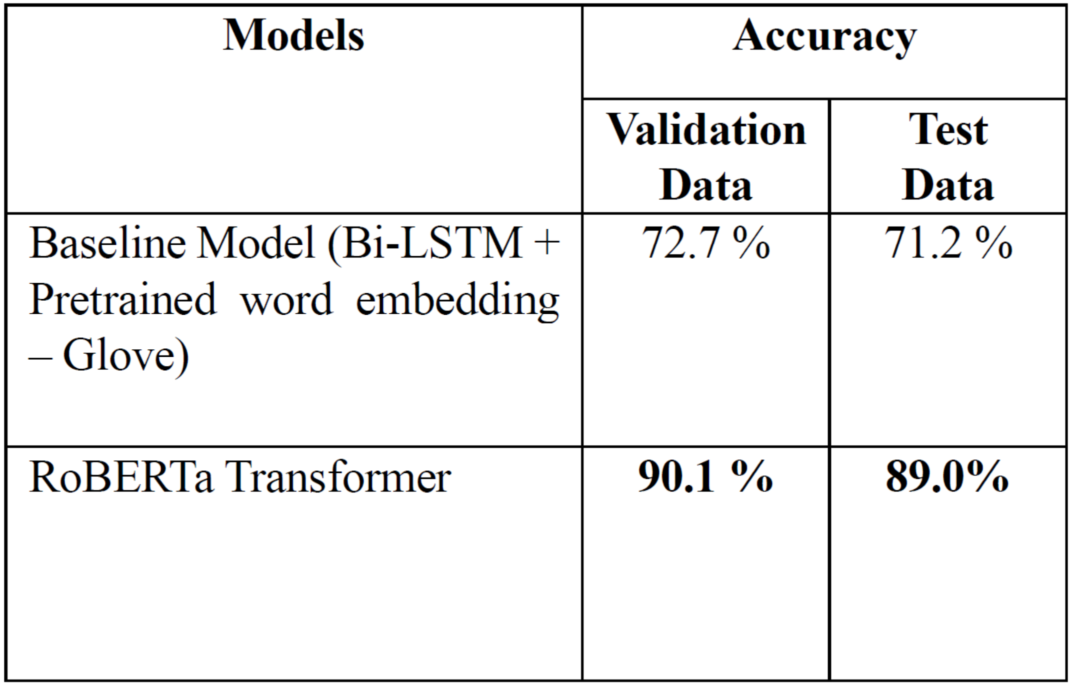
Logits

Un-pad output according to the original sentence length

Retrieve exact 'sense-key', according to the position of the actual output.

Senses
(Output)

Results



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

THANK YOU FOR YOUR KIND ATTENTION