Natural Language Processing

Word Sense Disambiguation

Team No. - 10

(NLTK)

Rajat Pal (2019900005)
Purnima Grover (2021201014)
Debashish Roy (2021201034)

Index

Abstract	3
Introduction	3-4
Dataset Used	4-5
Pre-processing of Corpus	5
Approaches Used	5-8
Experiment	9
Results	10-11
Discussion	11
Conclusion	12
References	12

Abstract

Word Sense Disambiguation is the technique to find meaning of the word based on the context of the sentence where the target_word is used. The data is trained on Supervised (KNN model) which is a baseline and neural network (LSTM, Bidirectional LSTM model). The hyperparameters are evaluated on validation data and the results are checked on the test data and the accuracies of all the models are compared.

Introduction

We recognise that words have multiple meanings depending on the context in which they are used in a phrase. When it comes to human languages, they are also ambiguous because many words can be understood in a variety of ways depending on the context in which they appear.

In natural language processing (NLP), word sense disambiguation is the capacity to detect which meaning of a word is activated by its use in a specific context. One of the first problems that any NLP system faces is lexical ambiguity, whether it is syntactic or semantic. Word's syntactic ambiguity can be solved with high-accuracy part-of-speech (POS) taggers. The difficulty of resolving semantic ambiguity, on the other hand, is known as WSD (word sense disambiguation). It's more difficult to resolve semantic ambiguity than it is to resolve syntactic ambiguity.

Consider the two distinct meanings of the term "bass" for example.

I'm hearing a bass sound.

Grilled bass is one of his favorite foods.

The presence of the word bass indicates that it has a specific connotation. It signifies frequency in the first statement and fish in the second. As a result, if WSD can disambiguate the aforementioned sentences, the right meaning can be assigned as follows:

I can hear sound in the bass/frequency range. He likes grilled bass and seafood.

Application of Word Sense Disambiguation -

WSD (word sense disambiguation) is used in practically every language technology application.

- Automated Translation - The most obvious application of WSD is machine translation, or MT. Lexical choice for words with multiple translations for different senses is handled by WSD in MT. In the target language, the sensations in MT are represented as words. The majority of machine translation systems do not include a WSD module.
- Information retrieval - Information retrieval (IR) is a software programme that manages the organization, storage, retrieval, and evaluation of data from document repositories, particularly textual data. The technology essentially aids users in locating the information they require, but it does not directly return the questions' answers. WSD is used to address the ambiguities in the IR system's inquiries. Current IR systems, like MT, do not use the WSD module directly, instead relying on the assumption that the user will put enough context in the query to only get relevant pages.
- Information Extraction and Text Mining (IE) - WSD is required in most applications to provide accurate text analysis. WSD, for example, aids intelligence gathering systems in correctly flagging terms. For example, instead of "medical medications," a medical intelligent system could need to detect "illegal drugs."
- Lexicography - Because current lexicography is corpus-based, WSD and lexicography can work in tandem. WSD provides rough empirical sense groups as well as statistically significant contextual indications of sense through lexicography.

Dataset Used

We used the Semcor 3.0 dataset here in this project. Semcor stands for semantically annotated English corpus. The SemCor corpus is a set of English texts that have been semantically annotated. The semantic analysis was carried out manually using WordNet 1.6 senses (SemCor version 1.6), which were then automatically transferred to WordNet 3.0. (SemCor version 3.0). The Brown corpus contains 226,036 sense   annotations from 352 texts in the SemCorpus corpus.

This corpus has been sense-tagged. By mapping WordNet 1.6 to WordNet 3.0 senses, SemCor 3.0 was automatically constructed from SemCor 1.6. Princeton University developed SemCor 1.6

and owns it. Rada Mihalcea (rada@cs.unt.edu) was in charge of the automatic mapping. Multi-word expressions (MWE) indicated with an underscore (_) are also present in the corpus, such as manor house. Siva Reddy has annotated these multi-word units.

Dataset link - https://github.com/rubenIzquierdo/wsd_corpora/tree/master/semcor3.0

Pre-processing of Corpus

- The data is present in multiple files in the brown folder in .naf format.
- Read the data of all files and add them in the corpus variable one by one and add wn_index = lemma + “%” + lexical_key new column in the corpus. Drop lexical_key column from dataset.
- Gloss is calculated only for those rows where wn_sense_num != 0.
- Gloss, other_gloss, idx is added in the dataset as new columns.
- The final dataset has file, context, target_word, gloss, is_proper_gloss as columns.
    - ➢ File stores the path of the file from where the data is taken.
    - ➢ Context contains the sentence in which we are finding the correct meaning.
    - ➢ Target_word is that word which has multiple meanings.
    - ➢ Gloss contains the meaning of the target word with respect to the context word.
    - ➢ is_proper_gloss tells whether that gloss has correct meaning with the context.

Approaches Used

The approaches which we used here in our project are as follows:-

- **Supervised Approach** - Machine learning approaches use sense-annotated corpora to train for disambiguation. These strategies believe that the context can supply sufficient evidence to resolve the ambiguity on its own. The terms "knowledge" and "reasoning" are not used in these procedures. The context is represented by a set of "word characteristics." It also contains information about the words that surround it. We used KNN as a supervised approach and treat this model as a baseline model.

  Algorithm

  1. Read the data from csv file

2. Extract each line from the dataset and from the context part create a window of size 5, 2 words before the target_word and 3 words after the target_word if they exist in the context. For creating the window, the steps are as follows
   a. Pre-process the line using tokenization and lemmatization.
   b. Remove stop words in the complete line.
   c. Create a window of size 5 with that sentence.
   d. Add all the context in the dataset with a new column as entry named "new_context"

3. Create BERT embedding of all the entries in the new_context and store the embedding in knn_bert.npy file.

4. Split the data in train, test data with 25% data of complete dataset in X_test and 75% in X_train.

5. For each row in X_test, find cosine similarity with all X_train data if the target_word of X_test is same as target_word of X_train otherwise add similarity to -1.

6. Add the column of cosine similarity in the dataset and sort the cosine similarity in dataframe in descending order.

7. Check if gloss of the tested sentence matches with gloss of the first 5 gloss of X_train whether they are same or not, if they are same then it's a correct output

8. Find accuracy with the correct matched gloss and the total number of columns in the X_test.

- **Neural Network Approach** - A neural network is a collection of artificial neurons or nodes that form a network or circuit of neurons. Artificial Neural Networks (ANNs) are layers of compute units that analyze data independently to replicate the functioning of the human brain. ANNs, like human brains, learn via experience and exhibit gains in tasks as the amount of data available increases. Shallow neural networks have one or two hidden layers, while deep neural networks have many hidden layers.

  Network Description (LSTM)

  1. Each row of dataset has following columns:
  - Context: Input sentence

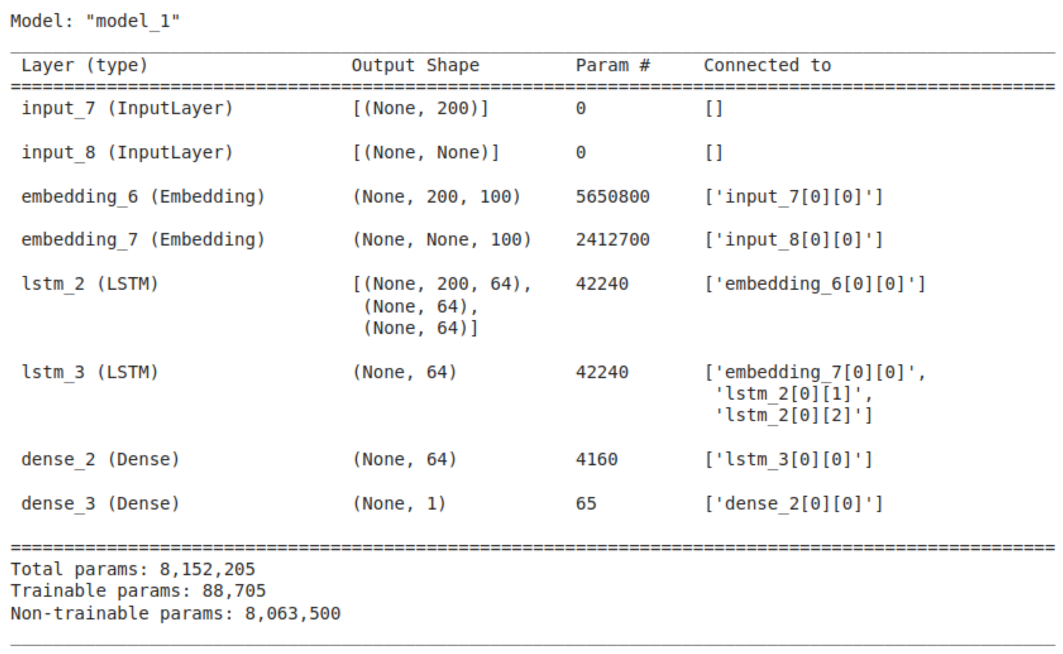
- Target_Word : Target word for which sense has to be predicted.
- Gloss: definition of target word
- is_proper_gloss : Flag which indicates if the gloss is the actual meaning of the target word in the sentence.

So for each context-target_word pair, there will be multiple rows in the dataset with different glosses but only one row will have ‘is_proper_gloss’ set to true which will indicate that the gloss represents the true meaning of target word in the sentence.

2. We appended the ‘context’ to ‘target_word’ and trained a Tokenizer from these appended sentences. These tokenized sequences are used for the encoder model.
3. We trained another tokenizer from glosses and These tokenized sequences are used in the decoder model.
4. The tokenized vectors are then passed through the Embedding layer which is initialized using Glove pre-trained embeddings.
5. The Embedding for encoder input layer is then passed through LSTM layer which will return the states(cell states and hidden states) as well
6. Tokenized gloss is passed through another Embedding layer which is also initialized using Glove pre-trained embeddings.
7. The Embedding for decoder input layer is then passed through the LSTM layer which is initialized using the states of the encoder layer.
8. The output of the decoder layer is passed through 2 dense layers. The last dense layer has a Sigmoid activation function which converts the output to a scaler.
9. loss is calculated using binary cross entropy loss function

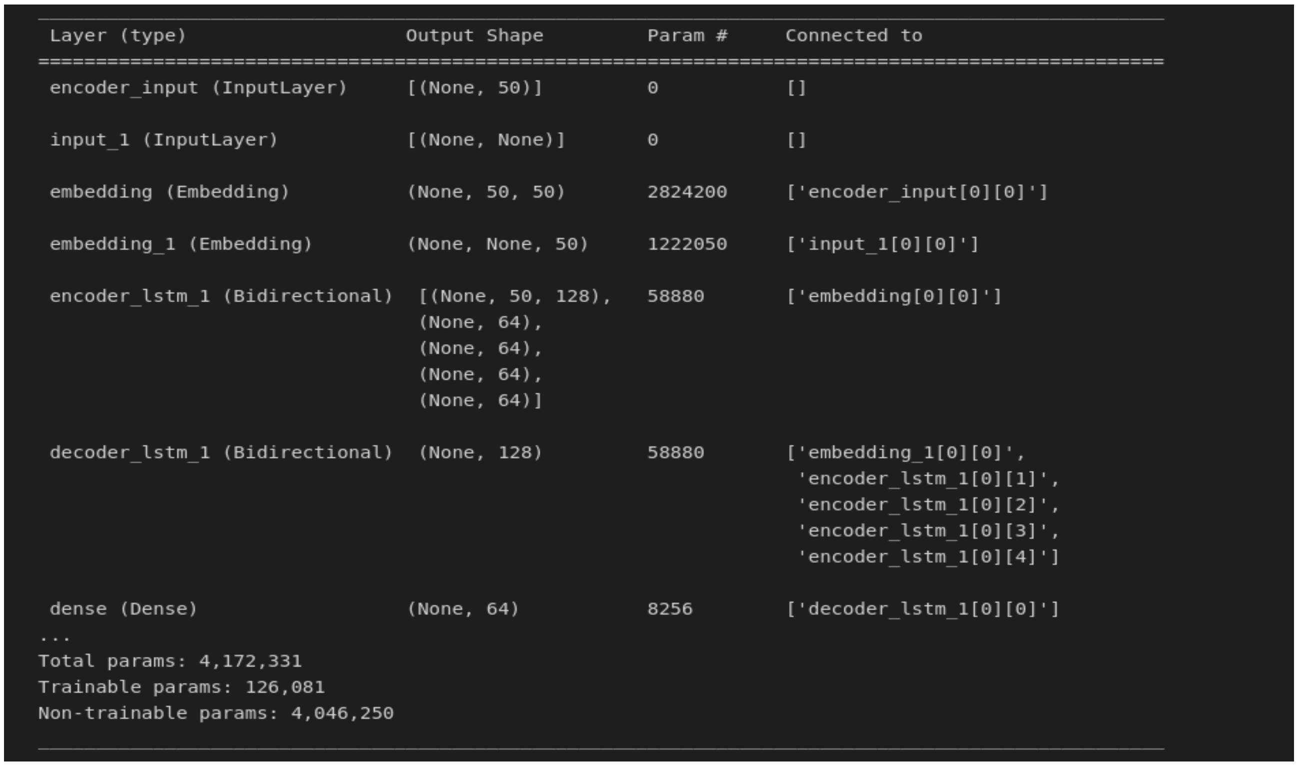
Network Description (BiLSTM)

Sometimes to understand the context of the word we have to read a few more, upcoming words. This context is not catched in the LSTM, but BiLSTM considers left to right context as well as right to left.



```
Model: "model_1"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_7 (InputLayer)           [(None, 200)]        0           []

 input_8 (InputLayer)           [(None, None)]       0           []

 embedding_6 (Embedding)        (None, 200, 100)     5650800     ['input_7[0][0]']

 embedding_7 (Embedding)        (None, None, 100)    2412700     ['input_8[0][0]']

 lstm_2 (LSTM)                  [(None, 200, 64),    42240       ['embedding_6[0][0]']
                                 (None, 64),
                                 (None, 64)]

 lstm_3 (LSTM)                  (None, 64)           42240       ['embedding_7[0][0]',
                                                                  'lstm_2[0][1]',
                                                                  'lstm_2[0][2]']

 dense_2 (Dense)                (None, 64)           4160        ['lstm_3[0][0]']

 dense_3 (Dense)                (None, 1)            65          ['dense_2[0][0]']

==================================================================================================
Total params: 8,152,205
Trainable params: 88,705
Non-trainable params: 8,063,500
__________________________________________________________________________________________________
```

LSTM



```
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 encoder_input (InputLayer)     [(None, 50)]         0           []

 input_1 (InputLayer)           [(None, None)]       0           []

 embedding (Embedding)          (None, 50, 50)       2824200     ['encoder_input[0][0]']

 embedding_1 (Embedding)        (None, None, 50)     1222050     ['input_1[0][0]']

 encoder_lstm_1 (Bidirectional)  [(None, 50, 128),   58880       ['embedding[0][0]']
                                 (None, 64),
                                 (None, 64),
                                 (None, 64),
                                 (None, 64)]

 decoder_lstm_1 (Bidirectional)  (None, 128)         58880       ['embedding_1[0][0]',
                                                                  'encoder_lstm_1[0][1]',
                                                                  'encoder_lstm_1[0][2]',
                                                                  'encoder_lstm_1[0][3]',
                                                                  'encoder_lstm_1[0][4]']

 dense (Dense)                  (None, 64)           8256        ['decoder_lstm_1[0][0]']
...
Total params: 4,172,331
Trainable params: 126,081
Non-trainable params: 4,046,250
__________________________________________________________________________________________________
```

BiLSTM

Experiment

Embedding:

We have used two embedding.

- **Glove:** GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.
- **BERT:** Bidirectional Encoder Representations from Transformers. BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers.

Models:

We have experimented with the models ranging from RNN to biLSTM and then moving towards BERT. We also tried with **K Fold validation** on a smaller dataset**.**

Imbalanced Classification:

There was huge variation in the positive and negative samples. For example a target word can have 10 meanings, for a given context and target word, only one can be correct and rest 9 will be false. So there is an imbalance in the training set.

So, we have two options: undersampling and oversampling. Oversampling will increase in the data, and we are already facing issues with training the data so we opted for undersampling and sacrificing some efficiency.

Evaluations Matric:

Simply reporting accuracy is not a good example and also has been discussed in the class. But here we have seen a real life example. In our case if a word has 10 different glosses and only one is correct, so for 9 examples it will be wrong. So even if we predict that any gloss is not correct, our accuracy will be 90%. But in reality the model will make no sense. So here we have used F1 scores.

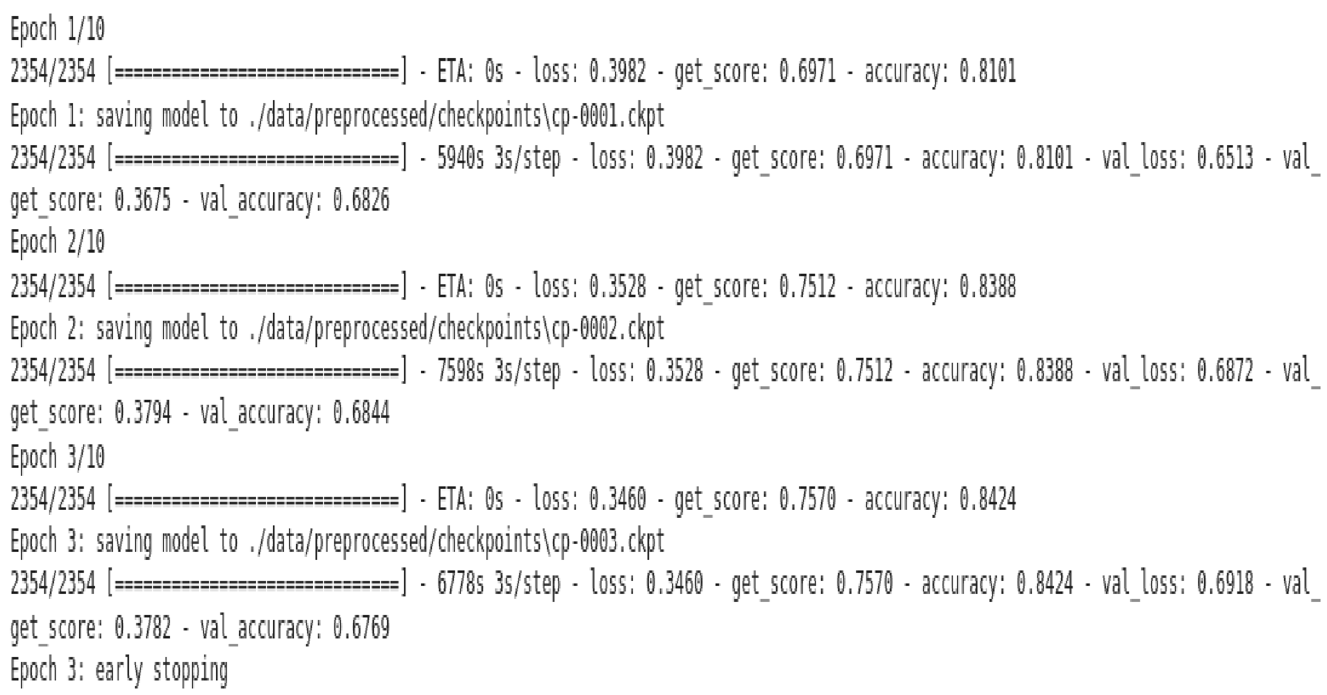
Results

KNN

Accuracy is 81.33% for the KNN.

Neural Network:

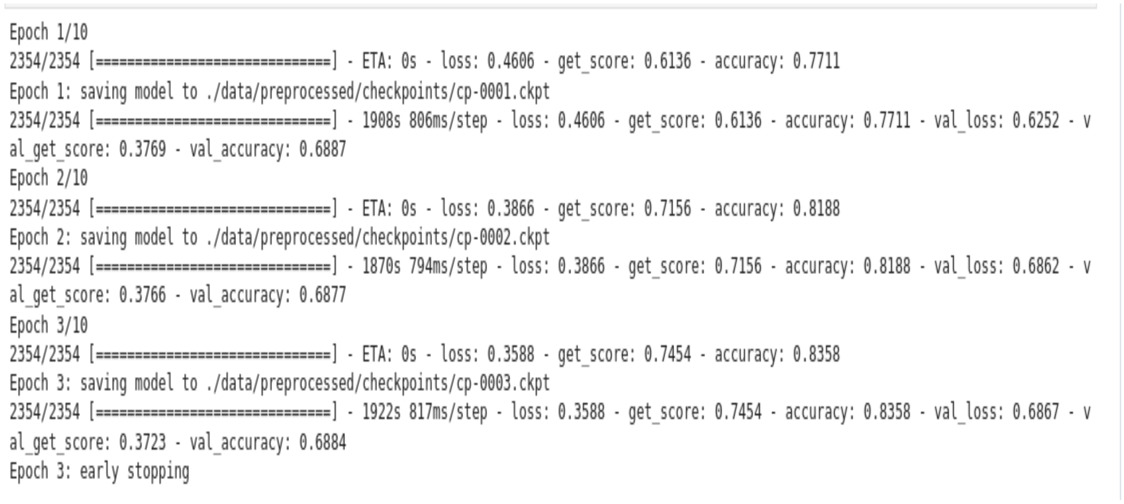
a. **BiLSTM**



```
Epoch 1/10
2354/2354 [==============================] - ETA: 0s - loss: 0.3982 - get_score: 0.6971 - accuracy: 0.8101
Epoch 1: saving model to ./data/preprocessed/checkpoints\cp-0001.ckpt
2354/2354 [==============================] - 5940s 3s/step - loss: 0.3982 - get_score: 0.6971 - accuracy: 0.8101 - val_loss: 0.6513 - val_
get_score: 0.3675 - val_accuracy: 0.6826
Epoch 2/10
2354/2354 [==============================] - ETA: 0s - loss: 0.3528 - get_score: 0.7512 - accuracy: 0.8388
Epoch 2: saving model to ./data/preprocessed/checkpoints\cp-0002.ckpt
2354/2354 [==============================] - 7598s 3s/step - loss: 0.3528 - get_score: 0.7512 - accuracy: 0.8388 - val_loss: 0.6872 - val_
get_score: 0.3794 - val_accuracy: 0.6844
Epoch 3/10
2354/2354 [==============================] - ETA: 0s - loss: 0.3460 - get_score: 0.7570 - accuracy: 0.8424
Epoch 3: saving model to ./data/preprocessed/checkpoints\cp-0003.ckpt
2354/2354 [==============================] - 6778s 3s/step - loss: 0.3460 - get_score: 0.7570 - accuracy: 0.8424 - val_loss: 0.6918 - val_
get_score: 0.3782 - val_accuracy: 0.6769
Epoch 3: early stopping
```

With BiLSTM we are getting f1 score of 0.7570 on training and 0.3782 at validation dataset. For accuracy we are getting 84.2% for training and 67.69% for validation data. At the model is stopping at 3rd epochs due to early stopping set by us. The accuracy are near about the same for the LSTM.

b. LSTM


```
Epoch 1/10
2354/2354 [==============================] - ETA: 0s - loss: 0.4606 - get_score: 0.6136 - accuracy: 0.7711
Epoch 1: saving model to ./data/preprocessed/checkpoints/cp-0001.ckpt
2354/2354 [==============================] - 1908s 806ms/step - loss: 0.4606 - get_score: 0.6136 - accuracy: 0.7711 - val_loss: 0.6252 - v
al_get_score: 0.3769 - val_accuracy: 0.6887
Epoch 2/10
2354/2354 [==============================] - ETA: 0s - loss: 0.3866 - get_score: 0.7156 - accuracy: 0.8188
Epoch 2: saving model to ./data/preprocessed/checkpoints/cp-0002.ckpt
2354/2354 [==============================] - 1870s 794ms/step - loss: 0.3866 - get_score: 0.7156 - accuracy: 0.8188 - val_loss: 0.6862 - v
al_get_score: 0.3766 - val_accuracy: 0.6877
Epoch 3/10
2354/2354 [==============================] - ETA: 0s - loss: 0.3588 - get_score: 0.7454 - accuracy: 0.8358
Epoch 3: saving model to ./data/preprocessed/checkpoints/cp-0003.ckpt
2354/2354 [==============================] - 1922s 817ms/step - loss: 0.3588 - get_score: 0.7454 - accuracy: 0.8358 - val_loss: 0.6867 - v
al_get_score: 0.3723 - val_accuracy: 0.6884
Epoch 3: early stopping
```

With BiLSTM we are getting f1 score of 0.7454 on training and 0.3723 at validation dataset. For accuracy we are getting 83% for training and 68% for validation data. At the model is stopping at 3rd epochs due to early stopping set by us.

Discussion

Analysis and Insights:

We have seen good accuracy with LSTM. We try to experiment with the BERT, but most of the time it is crashing the RAM due to limit exceeding. We are expecting better results through that. We are looking forward to extending our work with the BERT model. The BERT Model is promising to be better than the LSTM, BiLSTM.

Key Challenges:

While creating the project, few key challenges which we faced are as follows:-

- The dataset which we used is so large and the BERT model which we use in our analysis that the memory gets crashed while running the program.
- Creating BERT embedding in the KNN takes a lot of time as there are a lot of sentences in the dataset.
- There are some rare words in which the gloss is not present.
- It is expected to have significantly higher quality word vectors and LSTM models with more labeled training data, resulting in greater accuracy scores.

Conclusion

The baseline model which we used has an accuracy of 81.33%.

LSTM model and word embedding is giving validation accuracy of 68.77%.

Bidirectional LSTM and word embedding (glove) is giving validation accuracy of 67.69%.

References

- https://nlp.stanford.edu/projects/glove/
- https://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html
- https://arxiv.org/abs/1810.04805
- https://machinelearningmastery.com/random-oversampling-and-undersampling-for-imbalanced-classification/
- https://www.researchgate.net/publication/324014399_Neural_Network_Models_for_Word_Sense_Disambiguation_An_Overview