WORD SENSE DISAMBIGUATION

Team No. - 10 (NLTK)

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

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, BiLSTM and BERT 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

Word sense disambiguation is the ability to detect which meaning of a word is used in the context of the sentence when a single word has multiple meanings. The task of Word Sense Disambiguation (WSD) consists of associating words in context with their most suitable entry in a pre-defined sense inventory.
Example -

I'm hearing a bass sound.

Grilled bass is one of his favorite foods.

The meaning of word “bass” is different in both the sentences above.

DATASET USED

- We used the Semcor 3.0 dataset here in this project.
- Semcor stands for semantically annotated English corpus.
- The Brown corpus contains 226,036 sense annotations from 352 texts in the SemCorpus corpus.
- 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.

PRE-PROCESSING OF CORPUS (contd.)

- 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.

SUPERVISED APPROACH

- We used K Nearest Neighbour as a baseline model to compare it with the neural network model.
- Algorithm
  - ➢ Read the data from csv file
  - ➢ Take context of each row of the dataset - preprocess that using lemmatization and create a window of size 5 of the neighbouring words of the target_word in the context.
  - ➢ Create BERT embedding of all the entries in the new_context and store the embedding in knn_bert.npy file.
  - ➢ Split the data in train, test data with 25% data of complete dataset in X_test and 75% in X_train.

SUPERVISED APPROACH (contd.)

- 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.
- Add the column of cosine similarity in the dataset and sort the cosine similarity in dataframe in descending order.
- 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
- Find accuracy with the correct matched gloss and the total number of columns in the X_test.

NEURAL NETWORK

- We used LSTM, BiDirectional LSTM with Glove word Embedding.
- Algorithm
  - ➢ Each row of dataset has following columns - context, target_word, gloss, is_proper_gloss. 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.
  - ➢ We appended the ‘context’ to ‘target_word’ and trained a Tokenizer from these appended sentences. These tokenized sequences are used for the encoder model.
  - ➢ We trained another tokenizer from glosses and These tokenized sequences are used in the decoder model.
  - ➢ The tokenized vectors are then passed through the Embedding layer which is initialized using Glove pre-trained embeddings.

NEURAL NETWORK (contd.)

- The Embedding for encoder input layer is then passed through LSTM layer which will return the states(cell states and hidden states) as well
- Tokenized gloss is passed through another Embedding layer which is also initialized using Glove pre-trained embeddings.
- The Embedding for decoder input layer is then passed through the LSTM layer which is initialized using the states of the encoder layer.
- 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.
- loss is calculated using binary cross entropy loss function

```
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
```

Bidirectional **LSTM**

EXPERIMENT

1. **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.

EXPERIMENT (contd.)

2. 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 less dataset**.**

3. 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 gloss and only one is correct, so for 9 example 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 we here we have used F1 scores.

EXPERIMENT (contd.)

4. 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 a imbalance in the training set.

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

RESULTS

- KNN

```
[ ]  accuracy = correct / len(X_test) * 100
     print(accuracy)

     81.39999999999999
```

RESULTS (contd.)

- 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
```

RESULTS (contd.)

- Bidirectional 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
```

ANALYSIS & 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

- 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