Experiment 3

1. Dataset

The following links and points explain the various text files used as a part of our third experiment.

1. **"1_combined.txt":** This text corpus contains the combined Formal Idioms from the EPIE dataset where the first half are informal sentences with idioms and the second half has their literal translations.

   EPIE Corpus Link here

2. Simple End to End model built on top of bert to classify sentences

The following steps explain how we proceeded in building a very basic classification model using the BERT embeddings.

1. Data Preparation: - Load text data from 'combined.txt' and '1_combined.txt'. - Create labels, with half labeled as '1' and half as '0'. - Combine data into tuples of text and labels. - Shuffle data for randomization.
2. Data Splitting: - Split data into training 60%, validation (20%), and test (20%) sets.
3. TensorFlow Datasets: - Create TensorFlow datasets for training, validation, and testing. - Batch data for efficiency.
4. BERT Model Loading: - Load BERT model from TensorFlow Hub ('small_bert/bert_-en_uncased_L4_H-512_A-8/1').
5. BERT Preprocessing: - Load BERT preprocessing module. - Preprocess a sample sentence and display the processed inputs.
6. BERT Model Outputs: - Pass a sample sentence through BERT and display pooled and sequence outputs.
7. Building the Classifier Model: - Define a custom classifier model using Keras. - Preprocess text input, pass it through BERT, add dropout, and a dense layer for binary classification.
8. Model Prediction: - Predict sentiment score for a test sentence using the classifier model.
9. Model Architecture Visualization: - Visualize the architecture of the classifier model.
10. Model Compilation: - Compile the classifier model with loss, metrics, and optimizer.
11. Model Training: - Train the classifier model for 10 epochs with training and validation data. - Monitor and print training progress.
12. Model Evaluation: - Evaluate the model on the test dataset and display loss and accuracy.
13. Example Sentences Prediction: - Predict sentiment scores for three example sentences using the trained classifier model. - Display input sentences and their sentiment scores.

With this very basic model the classifier reached 50% accuracy confirming our hypothesis that a more complex architecture was required.

3. Building a Custom RNN Classifier for Recognizing Sentences with Idioms

We confirmed that a more complex architecture was required for our classification task to actually understand the meaning of the idioms better.

1. Loaded text data from the file 'combined.txt' and created corresponding labels. The labels were evenly distributed, with half designated as '1' and half as '0'. Data tuples were formed, and randomization was applied by shuffling the data.

2. Partitioned the combined dataset into three segments: training (60%), validation (20%), and testing (20%). TensorFlow datasets were established, and data was batched for efficient processing.

3. Defined class names for the labels, namely 'Not' and 'Idiom'. Demonstrated sample text examples along with their corresponding labels from the training dataset.

4. Conducted text preprocessing, including text encoding. Developed an encoder for text vectorization.

5. Constructed a model architecture that included an embedding layer, two Bidirectional LSTM layers, and additional dense layers. The model design allowed for handling variable sequence lengths with masking.

6. Compiled the model, specifying binary cross-entropy loss and employing the Adam optimizer with a learning rate of 1e-4. Model performance metrics were also defined, with accuracy as one of the key metrics.

7. Trained the model over ten epochs, monitoring its performance on the validation dataset. The training process involved the evaluation of both training and validation accuracy.

8. Evaluated the model on the test dataset, calculating the test loss and accuracy. The evaluation process assessed the model's generalization to unseen data.

9. Provided sample text input and observed the model's predictions for the given text. Furthermore, explored an alternative model architecture featuring two Bidirectional LSTM layers and compared its performance to the initial model. The accuracy in fact decreased when we used another LSTM layer but not very drastically.

10. Compiled and trained the alternative model, monitoring its performance. Finally, evaluated the alternative model's accuracy on the test dataset.

Conducted a comparative analysis of model performance by investigating the impact of employing 1 and 2 Bidirectional LSTM Layers between the embedding and dense layers. Achieved an accuracy of approximately 67%. Developed models comprising an embedding layer with two dense layers, each with a 50% dropout rate. With further tuning we will explore whether accuracy shows significant improvements.