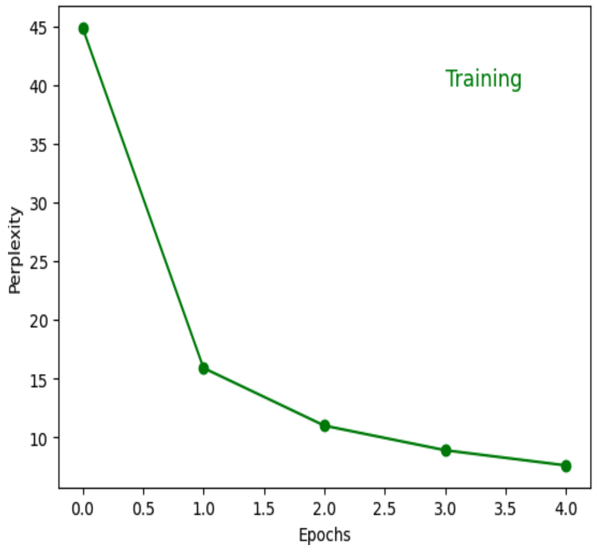
FFNN

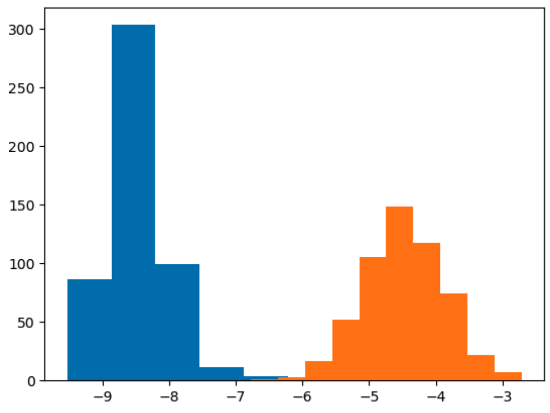
For a Bengio like model task, We created a model (context_size = 6, embedding_dimensions=200, hidden_layer (with skip connections) = (100+context_size*embedding_dimensions, loss=Crossentropy,optimizer=Adam, learning_rate = 0.001, num_epoch=5) and we took the tokenized data from "mix.*.tok" and trained our model with the given training file.

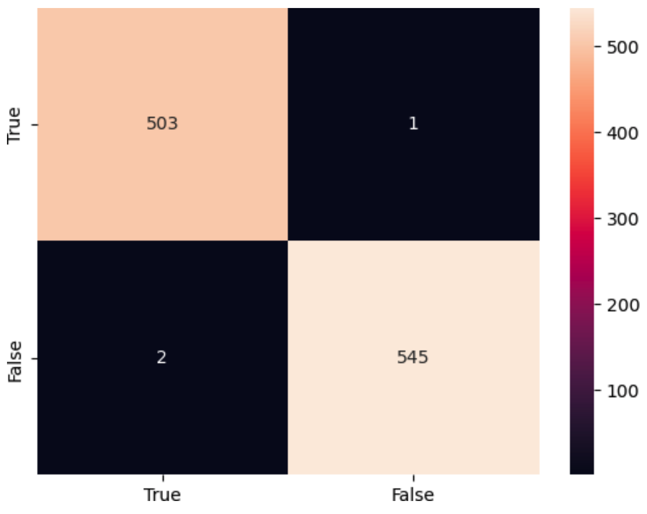
The model consists of 6 * 200 nodes in the input layer, which are 6 words with embedding dimensions as 200. The output of this is passed through a tanh nonlinearity and added with the input (6*200 as skip connection) and the output is distribution of negative log probabilities over the vocabulary of words (>30k). We used pytorch NN module to construct layers.

Once we trained a model with output as log probability over the vocab. We are taking the test bio and breaking it into 6 word chunks for prediction of the 7th word. Then we average the log probabilities of the entire bio and plot then using histograms for running a grid search over threshold of log probabilities to get classification results

Training a language model took us ~2 hours on a colab GPU. We could have used more data preprocessing like Named Entity Recognition, marking numbers, years with <NUM> token to handle out-of-vocabulary tokens. This would have somewhat reduced the vocabulary size, and hence both model size and training time could have been reduced.



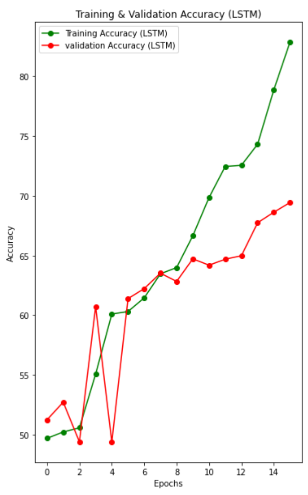


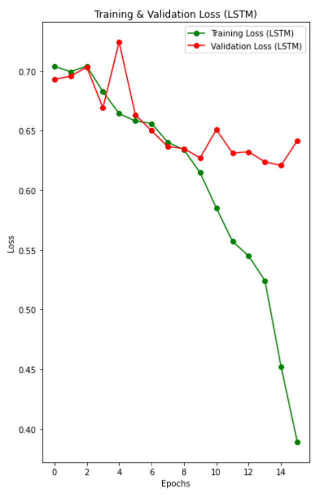


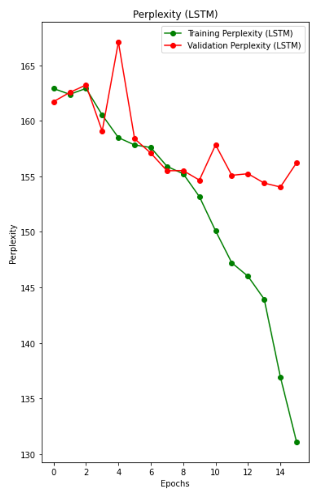
LSTM

For this task, we trained our LSTM model over the combined mix, real, and fake datasets. We used the ‘*.*txt*’ files and cleaned the data by converting the text into lowercase, removing extra spaces, punctuations, and stop words. We used SnowballStemmer and word_tokenize from NLTK for stemming and tokenization, respectively. We used Stanford’s “glove-wiki-gigaword-300” vectors for word embeddings, that are pre-trained over 2B tweets, 27B tokens, and 1.2M uncased vocabulary.

We used PyTorch’s LSTM module as a classification model to perform a binary classification over the given datasets. Our model consists of a two-layer LSTM followed by a linear layer that maps the hidden space with the binary output space. Here’s a list of the hyper-parameters of our LSTM model: (embedding_dim=300, hidden_dim=512, dropout=0.3, n_layers=2, lr=5e-3, num_epochs=15, batch_size=128, savename=”lstm_classification_model.pt”, optimizer=Adam(with betas = [0.9, 0.999], loss_fn=BCEWithLogitLoss)

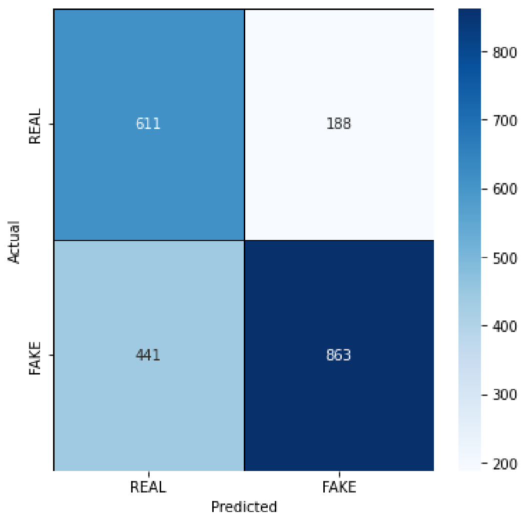






Since we used LSTM as a classifier, we are presenting the training/validation accuracy and loss plots. Additionally, we plotted the perplexity of the model as the exponential of the cross-entropy.

Report Fakes Detection results in a confusion matrix with labeled axes.



```
                  precision    recall  f1-score   support

  Predicted Fake       0.76      0.58      0.66      1052
  Predicted True       0.66      0.82      0.73      1051

        accuracy                           0.70      2103
       macro avg       0.71      0.70      0.70      2103
    weighted avg       0.71      0.70      0.70      2103
```

- We used GloVe for embedding, which is predominantly trained on English language corpus. Additionally, the data cleaning and preprocessing that we did was with an assumption of English texts as input. We've noticed multilingual texts in the given dataset. For a better outcome, we should have considered this factor. Here are a few resources and examples of multilingual text processing:
    1. https://medium.com/starschema-blog/text-preprocessing-in-different-languages-for-natural-la n guage-processing-in-python-fb106f70b554
    2. https://github.com/aboSamoor/polyglot
- From the above learning curves, it appears that the gap between training and validation metrics has significantly widened after the 12th epoch. The direction of curves suggests that our model started to overfit pretty early in the training stage. We should try out some better regularisation techniques to prevent that.

Transformer (HuggingFace)

Data preprocessing

We've followed similar data preprocessing like we did for FFNN and LSTM. We've combine all of the *.train.txt files to form one train file. Similarly combining *.val.txt and *.test.txt we are getting one file each for val and test set.
The data distribution after combining and removing duplicates is as follows:

```
        DatasetDict(
  { train: Dataset({
      features: ['bio', 'label'],
      num_rows: 19824
  })
  val: Dataset({
      features: ['bio', 'label'],
      num_rows: 2452
  })
  test: Dataset({
      features: ['bio', 'label'],
      num_rows: 2650
  })
})
```

Model

We chose the BERT model to train it for binary classification tasks (real or fake). More details about the model are available here. Model hyperparameters are as follows:

```
Model config BertConfig
        "architectures": [
        "BertForSequenceClassificat
        ion"
        ],
        "attention_probs_dropout_pr
        ob": 0.1,
        "classifier_dropout": null,
 "gradient_checkpointing": false,
 "hidden_act": "gelu",
 "hidden_dropout_prob": 0.1,
 "hidden_size": 768,
 "initializer_range": 0.02,
 "intermediate_size": 3072,
 "layer_norm_eps": 1e-12,
 "max_position_embeddings":
 512, "model_type": "bert",
 "num_attention_heads": 12,
 "num_hidden_layers": 12,
 "pad_token_id": 0,
 "position_embedding_type": "absolute",
 "problem_type": "single_label_classification",
 "torch_dtype": "float32",
 "transformers_version": "4.26.1",
 "type_vocab_size": 2,
 "use_cache": true,
 "vocab_size": 28996
}
```

Other hyperparameters are
Max_sequence_length = 512, num_train_epochs = 3, batch size = 16, loss function = categorical cross entropy

Plots and results

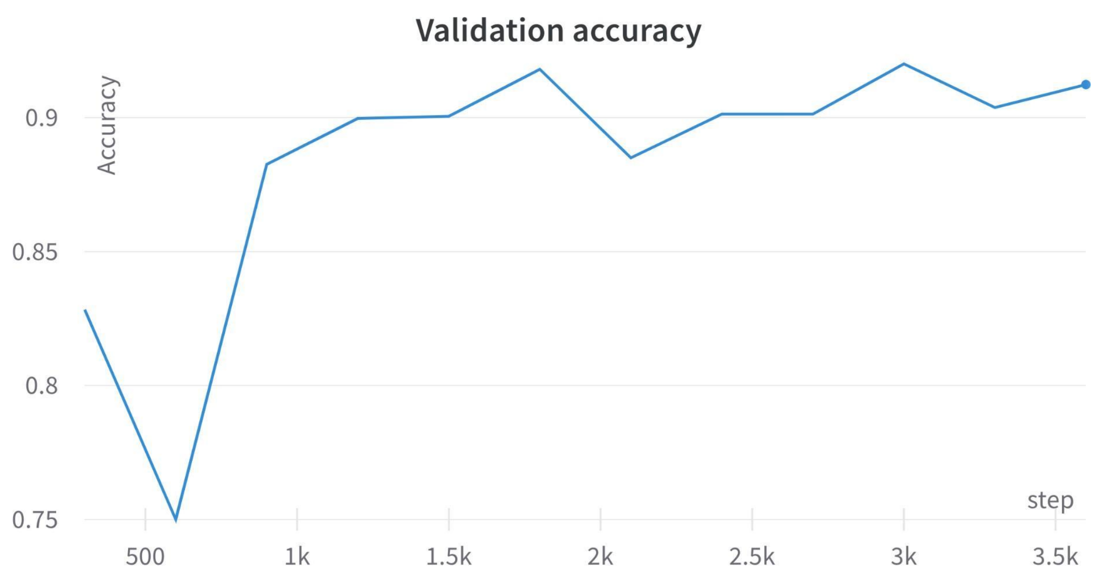
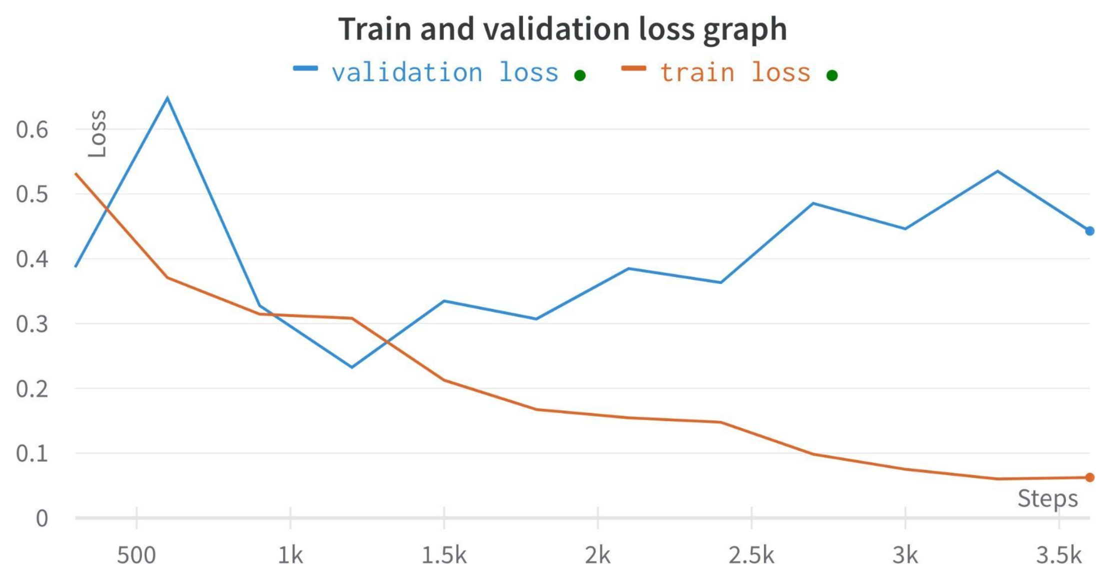
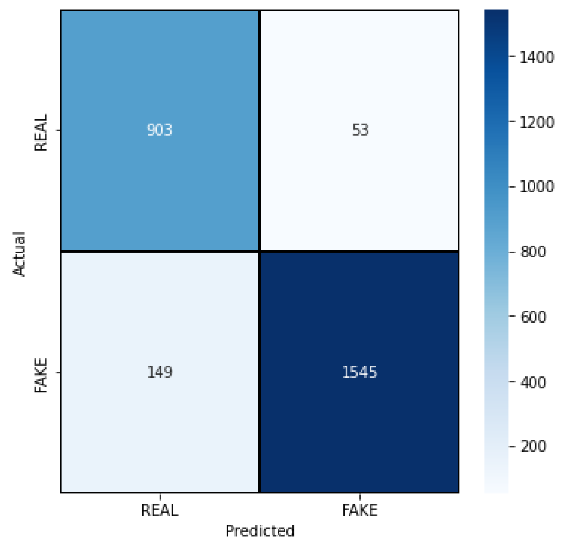


Fig. Validation accuracy as training progresses



Confusion matrix and classification report over test set



```
                 precision    recall  f1-score   support

Predicted Fake        0.94      0.86      0.90      1052
Predicted True        0.91      0.97      0.94      1598

      accuracy                            0.92      2650
     macro avg        0.93      0.91      0.92      2650
  weighted avg        0.92      0.92      0.92      2650
```

Results

The csv file with labels is attached in the zip file named as `blind_test_results_transformers.csv`.
Note: The huggingface model checkpoint size was around 1 gb so we could not attach it with the submission. However we've uploaded the best performing model here.

We've selected BERT as the best model because we can see from the loss curves that it's less prone to overfitting compared to LSTM and FFNN. This is confirmed by the best results on the test set as shown above in the classification report.
Some of the limitations of the BERT model are:

a) Large training time

One possible solution of this is we can freeze the weights of BERT. We can just train the final classification layer weight. Using this the model won't have to run back propagation over the full model and hence trains faster. However the model size is still an issue.

b) Huge model size

BERT model is very huge (around 500 MBs). To fix this we could've used DistilBERT which gives similar performance but is smaller than BERT.

Model	**Num. Parameters (millions)**	**Inference Time (ms)**
BERT	110	668
DistilBERT	66	410