

Classifying human written text from GPT-2 generated text with LR, CNN and BERT

Millie Lou

Komal Mistry

Zoe Xu

Abstract

In the field of Natural Language Processing (NLP), there has been a dramatic shift towards utilizing pre-trained deep language models. To perform text classification, this study uses state-of-the-art neural network models, Bidirectional Encoder Representations of Transformers (BERT) and convolutional neural networks (CNN), as well as a non-neural classifier, Logistic Regression (LR), as baseline. Specifically, the task is to distinguish human-generated text from fake text generated by the GPT-2 language model.

Results show BERT beat the baseline, achieving 90.34% F-score compared to LR's F-score of 88.23%. BERT and LR both outperformed CNN, which attained an F-score of 80.41%. In the error analysis, this study further confirms previous research's finding that sequence length affects neural network models performance. For example, any truncation in the input documents has a detrimental influence on the effectiveness of BERT. The primary contribution of this study is to introduce a simple but effective model in the field of fake text detection.

Introduction

OpenAI 's GPT-2 text generating model was released in 2019 and it immediately dominated the headlines. The language model has the capacity to produce an entire short story when fed with a seed like "Once upon a time". The generated story is an amazing improvement over GPT-2's contemporaries, other text generation algorithms. OpenAI initially created four language models with varying hyperparameters (i.e., 117M, 345M, 762M and 1542M).

Due to concerns of misuse like spreading disinformation, these models were released in stages from smallest to largest over the span of 10 months.

The ability to analyze and detect fake news will help combat the threat of weaponizing language models. This paper will contribute to the field of fake text detection by building binary classification models that differentiates between human generated text and GPT-2 generated text. Human generated text consists of paragraphs from the WebText dataset, which is a collection of text from blogs, papers and code curated by OpenAI. GPT-2 generated text are the outputs from the GPT-2 model, trained on WebText. All of the research from this study was performed on GPUs provided by Google Colab.

Related Works

The field of fake text detection is growing. The following are some recent research papers related to this field, broken down by theme

GPT-2

- [“Language Models are Unsupervised Multitask Learners”](#) (2018)
This paper explores the unsupervised learning approach in training a multitask language model. The 4 models produced from this research included GPT-2, which gave state-of-the-art level performance in 7 out of 8 language related tasks. This paper showcases the capabilities of GPT-2 models.
- [“Release Strategies and the Social Impacts of Language Models”](#) (2019)
OpenAI discusses the risks of releasing the four variants of the GPT-2 models and outlines the concerns of misuse. The largest GPT-2 model is so efficient that even human readers correctly classify fake text only 66% of the time. The two key insights from the paper are that shorter text was much harder to classify and that RoBERTa outperformed BERT in text classification tasks since it is a discriminative model, and not a generative one.

- ["Deep Faking" Political Twitter using Transfer learning and GPT-2](#) (2019)

This paper describes how using GPT-2 and transfer learning can mimic the writing style of individuals like Donald Trump. It shows GPT-2 can generate human-like text and, in turn, highlights the importance of fake text detection.

CNN

- [Understanding Convolutional Neural Networks for Text Classification](#) (2018)

This paper describes how the CNN, a model traditionally used in a continuous domain like image classification, can be adapted for use in the discrete domain of text classification. The authors emphasize the significance of filters and ngrams in extracting text features. The methods in this paper directly apply to this study.

BERT

- [Factuality Classification Using the Pre-trained Language Representation Model BERT](#) (2019)

This paper showcases how natural language understanding can be performed using pre-trained embeddings for English and Spanish text. The task was factuality classification, more specifically, assigning the correct factual tag to the verbal events present in the dataset. This paper uses BERT for text classification, which is the same task as this study.

- [Is BERT Really Robust? A Strong Baseline for Natural Language Attack on Text Classification and Entailment](#) (2020)

This paper shows that the state-of-the-art performance of BERT can be severely hindered by adversarial examples that have some minor modifications to the original text. Furthermore, the TextFooler, which has been used to create adversarial data, can reduce BERT's accuracy to below 15%. This paper reveals BERT's weaknesses which can help this study understand BERT's results.

- [DocBERT: BERT for Document Classification](#) (2019)

For tasks like document classification which do not require an

intensive semantic analysis of the text, this paper shows a fine-tuned BERT can outperform discriminative models like Logistic Regression and Support Vector Machines. Even though BERT is not discriminative, it can give a high accuracy in text classification tasks.

- [Human and Automatic Detection of Generated Text](#) (2019)
This research paper compares BERT to human raters in text classification by domain. While a fine-tuned BERT can easily outperform human raters, the performance drops when replicated in a cross-domain setting. A major factor in increasing the efficiency of BERT discriminators is adapting the sampling techniques to produce a more cross-domain training set.

Other Approaches to Fake Text Detection

- [GLTR: Statistical Detection and Visualization of Generated Text](#)(2019)
This paper introduces the Giant Language Model Test Room (GLTR), which is a statistically based model that helps identify fake text from human generated text. It uses BERT and 117M GPT-2 models. The authors conclude that synthetic text tends to have fewer synonyms and text-generating models tend to over utilize words and phrases that have a higher probability attached to them. This paper takes a different approach to fake text detection than this study, and it also provides fake text features that classifiers can exploit.

Datasets

Data description and collection

The datasets for this study are from [OpenAI](#). Due to engineering limitations, this study uses a subset of the original:

- Webtext: first 50k out of 250k documents in json format. The genre is a collection from blogs, papers and code curated by OpenAI.
- GPT-2: first 50k out of 250k documents of the train-split file in json format. GPT-2 was trained on WebText so the style of its generated text will mirror those found in Webtext.

The subset is not randomly selected from the original because the original datasets are already unordered and random. Hence, extracting the top-k documents is acceptable practice. Previous attempts to process a total of 200k documents failed because of an out-of-RAM problem on Colab.

Dataset processing

Since the original Webtext and GPT-2 data are provided separately and without labels, Webtext documents are assigned "1" and GPT-2 documents are assigned "0". Then the two files are concatenated into one Panda DataFrame. The combined 100k dataset is randomly shuffled and split into train, validation, and test datasets (60k:20k:20k), and stored as ".csv" format.

The same procedure described above is followed to process a 1k dataset (500 from Webtext and 500 from GPT-2). The purpose of this subset is to make sure codes run error-free. The final results will be based on the 20k test set.

Exploratory Dataset Analysis

This is a breakdown of each split dataset by label:

	GPT-2	WebText
Train	30118	29882
Validation	9876	10124
Test	10006	9994

The datasets are well balanced so there are no class-imbalance issues. The datasets are not perfectly balanced because the 50k Webtext and 50k GPT-2 data are combined into a Dataframe first, and then split.

Summary Statistics

	100k dataset	50k GPT-2	50k Webtext
average words/document	560	522	597
maximum words	4301	2211	4301
minimum words	2	10	2
average sentences/document	22	21	23
maximum sentences	257	162	257
minimum sentences	1	1	1
vocabulary size	296,067	NA	NA

Method

1) Logistic Regression

This study follows OpenAI's footsteps in using LR as the baseline. The code for LR is adapted directly from [OpenAI](#). The particular hyperparameters of LR are:

- No dedicated tokenizer was used.
- Documents were vectorized using tf-idf vectorizer.
- The C hyperparameter was fine-tuned to 100 for optimal results.

2) Convolutional Neural Networks

CNNs are very fast to train among neural networks and the code for this study's text classification CNN is adapted directly from this [CNN tutorial](#). The hyperparameters of this study's CNN are:

- Tokenizer: spacy for english
- vocabulary_size = 145,549 (min_freq=2)
- embedding_dim = 300
- kernel_sizes = [2,3,4]
- kernels_num = 32
- dropout = 0.5
- output_size = 2
- learning_rate = 0.0001
- epoch = 20
- Initial weights sampled from a normal distribution

This is the CNN architecture:

```
CNN_Text(  
  (embeddings): Embedding(146142, 300)  
  (convolution_layers): ModuleList(  
    (0): Conv2d(1, 32, kernel_size=(2, 300), stride=(1, 1))  
    (1): Conv2d(1, 32, kernel_size=(3, 300), stride=(1, 1))  
    (2): Conv2d(1, 32, kernel_size=(4, 300), stride=(1, 1))  
  )  
  (dropout): Dropout(p=0.5, inplace=False)  
  (fc): Linear(in_features=96, out_features=2, bias=True)  
)
```

Optimizer : Adam, Loss Function : CrossEntropyLoss

3) BERT

Unlike some of the dominant language models in the NLP, BERT shows a dramatic improvement on multiple NLP tasks (Devlin et al., 2019). Inspired by previous work on text classification using a fine-tuned BERT model (Adhikari et al., 2019; Radford et al., 2018), this study further trained a BERT-base model using Webtext and GPT-2 dataset to see if BERT can outperform CNN and LR models.

To alleviate the computational burden, this study used the BERT_{base} uncased model, which is a variant of BERT with 12-layer, 768-hidden, 12-heads and 110M-parameters (see Appendix A for the architecture). It is pre-trained on lower-cased English text only. The model was downloaded from the transformers package from [Hugging Face](#). For the binary classification task, this study adopted Huggingface's BertForSequenceClassification, which has an additional single linear layer on top for classification.

BERT requires raw text to be tokenized by the tokenizer included with the model because it has particular ways of handling fixed sequence length and out-of-vocabulary (OOV) words. It also requires specific formatting for the input data, which is carried out by the tokenizer.encode_plus function:

- 1) Add a special token [CLS] to the start of each sentence. The [CLS] token is designed for classification tasks. BERT consists of 12 Transformer encoder layers, and each transformer takes in a list of token embeddings and produces the same number of embeddings on the output with the feature values changed. The BertForSequenceClassification model only uses the final hidden state corresponding to the first embedding, which is corresponding to the [CLS] token, as the aggregate sequence representation for doing classification tasks (Devlin et al., 2019).
- 2) Append a [SEP] token at the end of each input sentence.
- 3) Pad or truncate all the input sentences to a single constant length. This step must be done because BERT can only deal with fixed-length input, and input sentences are restricted to a

maximum of 512 tokens. Padding is accomplished using [PAD] tokens, whose index number is 0 in BERT vocabulary. Also, the maximum length can be set by a hyperparameter "MAX_LEN".

- 4) Explicitly differentiate real tokens from padding tokens with the attention mask. It is an array consisting only 1s and 0s, where 1s indicating the tokens are not [PAD]. This step is essential because it ensures BERT's self-attention mechanism knows not to incorporate PAD tokens into its interpretation of input sentences.

The BERT tokenizer returns a dictionary of the tokenized sentences, the respective token IDs, and the attention masks.

Based on the experiments of BERT performance on different maximum sequence length (MSL) that are discussed in the next section, this study fine-tuned BERT on input sentences with MSL of 256 and batch size of 32. The optimizer is AdamW, which is a class from the Huggingface library where "W" stands for fixed weight decay. The learning rate was $2e-5$ and the epsilon parameter was set to be $1e-8$. Epsilon is a very small number to prevent any division by zero (Kingma and Ba, 2017). Also, the paper recommends 2-4 training epochs for BERT, so this study's BERT is trained for 4 epochs.

Experiment

1) Logistic Regression (Baseline)

The baseline LR is a discriminative model that only takes seconds to fit to the data. It already achieves high classification accuracy with an overall F-score of 88% on the test set.

2) Convolutional Neural Networks

In an attempt to beat the baseline, the following experiments were carried out to find the best CNN model:

- `TEXT.build_vocab(min_freq=5)` - vocabulary size shrunk from 145,529 (originally `min_freq=2`) to 78,406. This is an attempt at seeing if the model would perform better with more targeted, higher frequency vocabulary.
- `TEXT.build_vocab(min_freq=2, vectors = "glove.6B.300d")` - pre-trained word embeddings were tried and the performance was not bad, however it was a few points behind the CNN that sampled from a normal distribution for its initial embedding weights.
- `kernel_num = 10` - no improvement in performance so set back to 32.
- `batch_size = 32` - after reducing batch size, the model ran slower and performed worse.
- `drop_out = 0.7` - no improvement in performance, so set back to 0.5.
- `learning rate=0.01` - model was not learning and gave the same validation F-score for all epochs.
- `SGD optimizer(lr=0.0001, momentum=0.9, weight_decay=0.001)` - model stopped learning and gave the same validation F-score after one epoch.
- `SGD optimizer(lr=0.0001, momentum=0.5, weight_decay=0.01)` - model improved with each epoch but performance was poor.

After deciding on the hyperparameters from the fine-tuning attempts, the model from the last epoch performed the best (80.65% on the validation set) so that was the model used for the test set. See Appendix B for the results from each epoch.

3) BERT

Adhikari et. al. (2019) mentioned that varying maximum input length of different datasets yield different results in loss of F-score. They found a minor decrease in the F-score on the Reuters dataset whereas on the IMDB dataset, lowering the maximum sequence length (MSL) corresponds to a drastic fall in accuracy, suggesting that the entire document is necessary for this dataset (Adhikari et. al., 2019). Like previous research, this study hypothesizes that fine-tuning BERT on training data with longer sequence lengths might yield higher macro F-scores. Since the average document length of this research's input dataset is 560 at the token-level, fine-tuning the model on shorter sequences, like only 32 or 64 tokens, might cause loss of information and hinder the model's performance. To further examine whether MSL can affect the accuracy of this study's BERT, training data with different sequence lengths were fed to BERT and the results from the validation set were recorded:

BERT performance by sequence length:

Max length	Precision		Recall		F-score	
	Webtext	GPT-2	Webtext	GPT-2	Webtext	GPT-2
32	0.85	0.97	0.97	0.83	0.91	0.89
64	0.89	0.98	0.98	0.88	0.93	0.92
128	0.85	0.99	0.99	0.82	0.91	0.90
256	0.85	0.99	0.99	0.82	0.92	0.90

(Note: test max_len = 512 was not tested because of out-of-memory error)

The table above shows that BERT performed consistently well across all sequence lengths. It may be because the train, validation and test set were from the same two sources, Webtext and GPT-2, hence the model learned to detect even the shorter sequences. However, training BERT on longer sentences made the model more prone to overfitting. The model trained on sequences of length 128 and 256 overfitted on the training data after the first epoch whereas the model trained on sequences of length 32 did not overfit until epoch 3 (see Appendix C for training results on different MSL).

Although tuning the hyperparameter `max_length` did not show an obvious improvement on the number of false positives, the number of false negatives (incorrectly predicted Webtext) reduced dramatically, from over 200 to 65. The drop in the number of false negatives might be because human-generated texts are more consistent in structures, grammar, and word choices; hence, it was easier for BERT to differentiate Webtext from GPT-2 generated texts. It proved that by inputting longer sequences, the model was able to generate better understanding of the differences between human-generated and GPT-2 generated texts.

Here are the results of training loss, validation loss, validation accuracy, and time elapsed:

	Training Loss	Valid. Loss	Valid. Accur.	Training Time	Validation Time
epoch					
1	0.247503	0.258737	0.91490	0:24:30	0:02:39
2	0.078372	0.343340	0.91555	0:23:44	0:02:30
3	0.027861	0.277208	0.94710	0:23:48	0:02:30
4	0.009999	0.514460	0.92440	0:23:45	0:02:30

From the training statistics, although the validation accuracy increased a little bit in the later epochs, the model seriously over-fitted. Therefore, this study uses the model from the first epoch, which showed minimal overfitting.

Results and Analysis

Here is an overview of the final results on the test set:

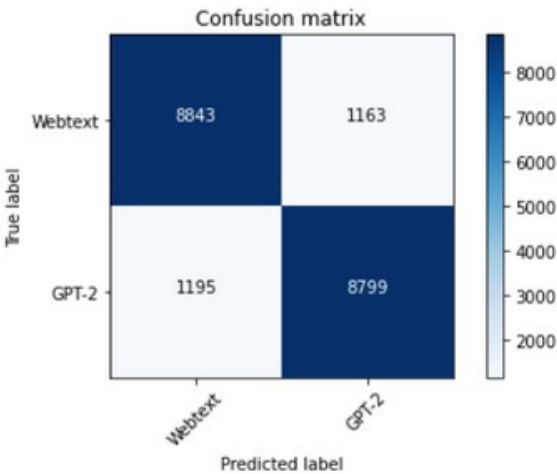
Model	Epochs	Training Time	F-score on Test Set
LR	1	< 1 minute	88.23%
CNN	20	11 minutes	80.41%
BERT	4	1.5 hour	90.34%

The following sections will provide an in depth analysis of each model’s predictions, as well as an error analysis.

1) Logistic Regression

The baseline did very well with an F-score of 88.23%. This is the classification report, followed by the confusion matrix:

	precision	recall	f1-score	support
Webtext	0.88	0.88	0.88	10006
GPT-2	0.88	0.88	0.88	9994
accuracy			0.88	20000
macro avg	0.88	0.88	0.88	20000
weighted avg	0.88	0.88	0.88	20000



From the above confusion matrix, it is apparent that LR did a good job in the binary classification task. The True positives and True Negatives are both high and the numbers are pretty close to each other (8843 and 8799 respectively). The False Positive and False Negative numbers are also about the same, showing that LR had equal difficulty in correctly assigning the two classes.

Analysis:

To investigate the shortcomings of the classifier, two strings, one each from GPT-2 and WebText, were extracted and passed to the model to evaluate.

- GPT-2 string : “James Harden is shooting 29 percent down the floor when guarded by Kawhi Leonard in this series, including just 10 percent from three. When not guarded by Leonard, Harden is shooting nearly 50 percent.”

The LR model incorrectly predicts the string is WebText, with the following class probabilities:

```
array([[0.11316767, 0.88683233]])
```

The model was 88.68% confident that the string is human-generated. This shows that the classifier is unable to pick up on the irregularities in the sentence. For example, phrases like “down the floor” are out of context and are indicators of the string being synthetic to human raters.

- WebText string : "COPYRIGHT NOTICE
This project is licensed under the Creative Commons Attribution-ShareAlike 3.0 Unported License.

Feel free to use, copy, modify, publish and distribute, including commercial products or services."

The logistic regression model had the following class probabilities for the above string:

```
array([[0.97010162, 0.02989838]])
```

Here, the model is very confident (97%) in its wrong prediction that this string belongs to class GPT-2. Even though the document is coherent, it is not a typical declarative statement like the GPT-2 string in the first example. Rather, this is an official notice. Perhaps due to the type of document, the classifier is unable to distinguish it clearly from the GPT-2 generated text.

2) Convolutional Neural Networks

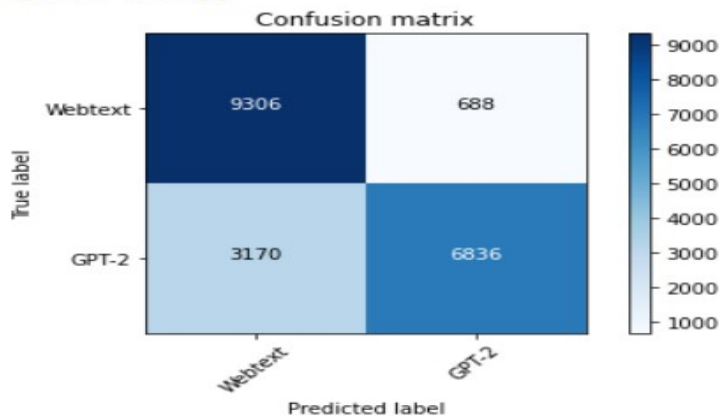
The best CNN model predicted about 80% of the data correctly (16,142/20,000).

```
Overall f-score: 0.8041
Classification report:
              precision    recall  f1-score   support

   Webtext      0.75      0.93      0.83     9994
   GPT-2       0.91      0.68      0.78    10006

 accuracy      0.81     20000
 macro avg      0.83     20000
 weighted avg    0.83     20000

Confusion matrix, without normalization
[[9306  688]
 [3170 6836]]
```



In terms of error, the model had difficulty correctly detecting GPT-2 generated text. (3,170/3,858). As the confusion matrix shows, 82% of the wrong predictions are categorizing GPT-2 text as Webtext.

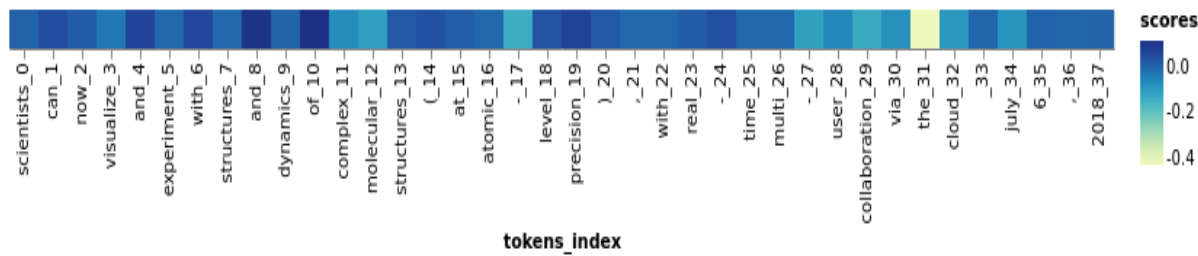
Analysis:

As previous research pointed out, shorter fake text is harder to detect. For the analysis, two of the shortest GPT-2 documents and two of the longest GPT-2 documents from the validation dataset were extracted for the model to make predictions. See Appendix D for the GPT-2 strings. Unsurprisingly, the model correctly predicted the longer documents but not the shorter documents.

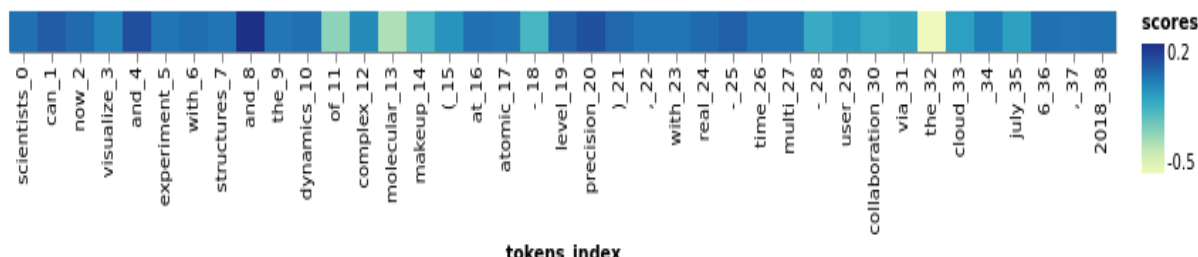
Strings	Sequence length	Labels	Probabilities
GPT-2_string_1	38	1	68.72%
GPT-2_string_2	42	1	81.43%
GPT-2_string_3	179	0	83.38%
GPT-2_string_4	186	0	96.23%

- String_1 analysis: interestingly, the model was not as confident with its prediction for this string compared to the other short string. By trial and error in modifying the string, the model's prediction probability increased to 74% when one "the" was added and a repeated word was replaced (i.e., the second "structure" with "makeup"). See the changes in String_5 in Appendix D. Experiments with adding or removing punctuations and dates were conducted, but these features did not make a difference. The biggest gain in probability comes from replacing the repeated word, hinting that repetition is a feature for synthetic text detection. In string_1 and string_5 (the modified string_1) visualizations, surprisingly stop words like "and" and "of" were considered important features.

String_1:

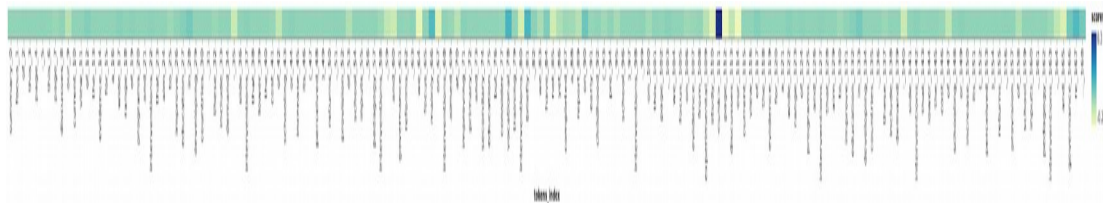


String_5:



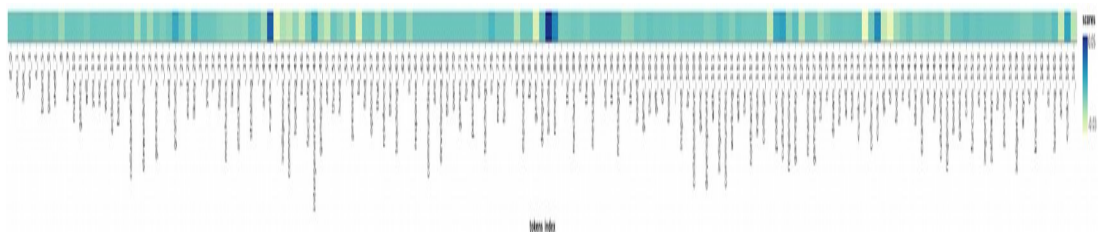
- String_3 analysis: having established that repeated words appear to be an indicator for synthetic text detection for a CNN model, this string actually has a few repeated sentences. Hence, the model confidently predicted it is GPT-2 generated. When repetition was removed in string_6 (see Appendix D), which also shortened the sentence, the model then made the wrong prediction, thinking the string_6 is human-generated. In the string_3 visualization, the more important token was an <unk> token.

String_3:



- String_4 analysis: Perhaps the model correctly predicted this document because the writing is unnatural. Even though the writing is grammatical, phrases like “the mechanic is in the message” does not make sense. Experiments of sequence length were conducted by deleting one sentence at a time to see how the CNN responded. Surprisingly, the document needed to be truncated to one sentence (i.e., 28 words long) in order to fool the model (see string_7 in the Appendix D). There are definitely certain features in this particular document that makes the model very confident it is fake text. In the visualization, the name “Brenda” and an <unk> token were the most important features to the model.

String_4:



- Most important and least important words to CNN were extracted. Unfortunately, it does not appear to reveal anything interesting.

Most Important Words			Least Important Words		
	word	score		word	score
1	<u>pendleton</u>	0.74		<u>cooke</u>	-12.57
2	sandwiched	0.72		unsigned	-4.37
3	<u>c'mon</u>	0.69		prev	-1.52
4	<u>leland</u>	0.67		mick	-1.1
5	<u>yeager</u>	0.64		laird	-1.01
6	intermediate	0.62		<u>mel</u>	-0.89
7	weaves	0.61		<u>capito</u>	-0.82
8	aseptic	0.61		536	-0.79
9	steered	0.61		<u>bsi</u>	-0.71
10	straightening	0.6		<u>pepe</u>	-0.7

3) BERT

The classification report and confusion matrix of the best BERT model is shown below.

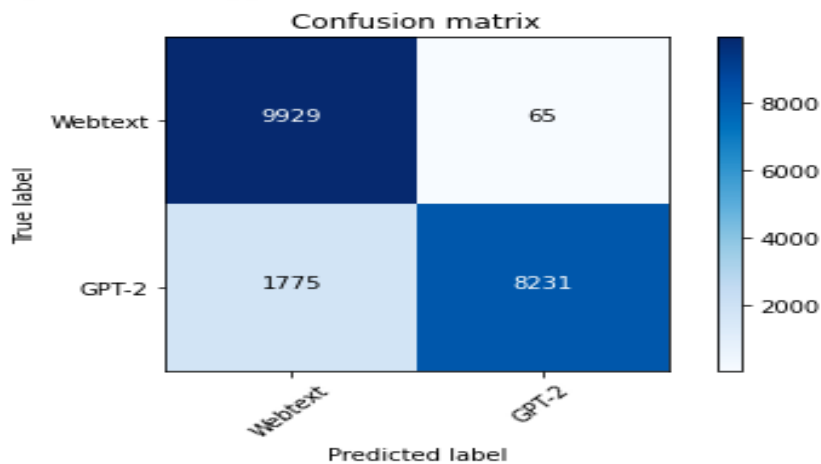
Macro F1 score: 0.9073320484787553

Classification report:

	precision	recall	f1-score	support
GTP-2	0.99	0.82	0.90	10006
Webtext	0.85	0.99	0.92	9994
accuracy			0.91	20000
macro avg	0.92	0.91	0.91	20000
weighted avg	0.92	0.91	0.91	20000

Confusion matrix, without normalization

```
[[9929  65]
 [1775 8231]]
```

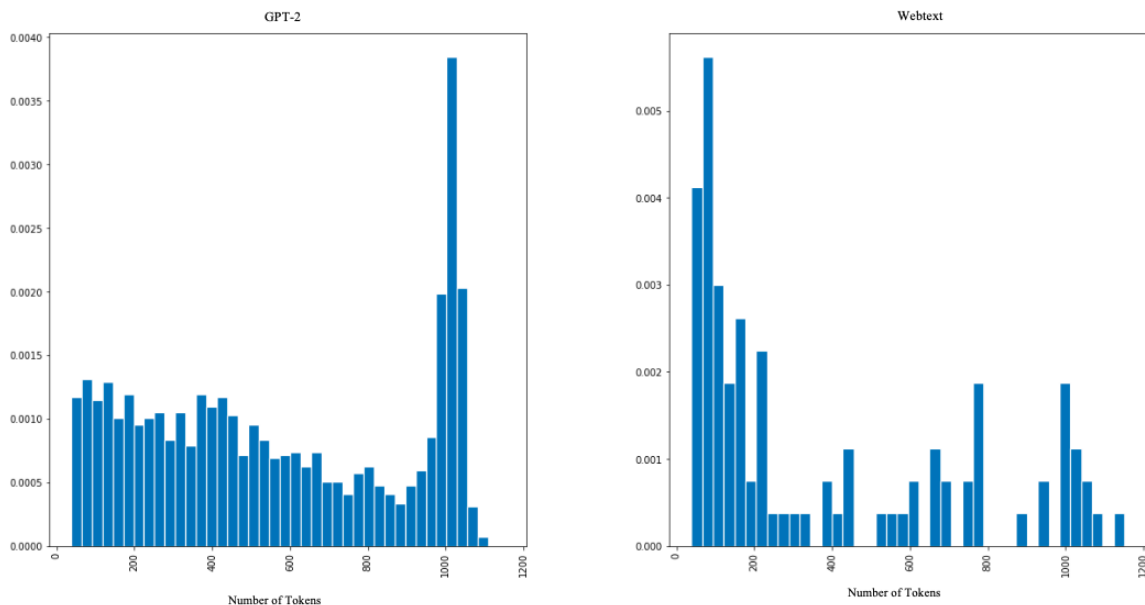


BERT outperforms LR and CNN on the test set. It predicted 90.73% of the data correctly (18,160/20,000), and the error rate was 9.2% (1840/20000). In terms of error, the BERT model showed the same problem that CNN had: it had trouble correctly labelling GPT-2. It incorrectly categorized 1775 GPT-2 text as Webtext, which is 96.47% of the errors (1775/1840). The cause of the model mislabelling GPT-2 text may be because:

- 1) GPT-2 did a good job in generating texts that are truly human-like.
- 2) Since we truncated the documents to a maximum of 256 tokens, the model was unable to learn from a full set of data, affecting its performance.
- 3) As previous research mentioned, shorter sentences are harder to classify.

Analysis

Since previous research stated sequence length influences the fake text detection, we created two histograms of incorrectly predicted documents to see how sequence length affects predictions. In the following histograms, “GPT-2” means the documents are GPT-2 generated but our model incorrectly predicted them as human-generated, and “Webtext” means the documents are from Webtext but our model incorrectly predicted them as GPT-2 generated. The x-axis is the actual count of tokens in the document, not just the 256 tokens that BERT was trained on.



Two major observations from the histograms:

1. The shorter Webtext documents were more prone to wrong predictions than GPT-2 data. The number of wrong predictions for GPT-2 data dropped from the minimum tokens (less than 100) to about 900 tokens, but the drop is gradual, unlike Webtext’s histogram.
2. At around 1000 tokens, there is a spike in wrong predictions for GPT-2 generated texts. Perhaps truncating documents to 256 tokens seriously and adversely affected BERT’s

performance when the actual document was 1000 tokens long. However, the BERT model was able to predict long Webtexts correctly, indicating that most human-generated texts contain enough information in the first 256 tokens to differentiate themselves from synthetically generated texts. Sequence length might not have played a significant role in fooling our BERT model, rather it was the inability to feed the full length document to BERT that hindered its performance.

These observations further consolidate the conclusion from Adhikari et. al. (2019) that any amount of truncation is detrimental in BERT's performance of text classification.

Since BERT has the same problem as CNN in correctly labelling GPT-2 data, this study uses the same test strings in the CNN analysis for BERT. The results are as follows.

Strings	Sequence length	Labels	Probabilities
GPT-2_string_1	38	0	99.439%
GPT-2_string_2	42	1	99.6138%
GPT-2_string_3	179	1	99.9541%
GPT-2_string_4	186	0	99.9988%
GPT-2_string_5 (alternation of string 1)	39	0	94.7512%
GPT-2_string_6 (alternation of string 3)	116	0	99.5405%

Results from these test strings show that BERT outperforms CNN in correctly classifying GPT-2 generated text. Unlike CNN, which is definitely influenced by sequence length, BERT correctly classifies shorter sequences (string_1 and 2, see Appendix E for multi-head attention weight on the last hidden layer of the first [CLS] token). Furthermore, by looking at the plots of the multi-head attention weights on [CLS] token, BERT tends to assign the highest attention to conjunctions, especially for long sentences. The following are an analysis by test string:

- String_1 and string_5 analysis: by altering string_1 to make it more fluent and resemble human-generated text better, the predictions and the probabilities did not change a lot. Both string_1 and 5 were predicted as GPT-2 generated text with a very minor drop in the probabilities, from 99.439% to 94.751%. By comparing the multi-head attention weights on the [CLS] token of the last hidden layer of these two strings, for the first string the first two words “scientists” and “can” had the high attention weights. The word “with” also has a high attention weight across multi heads compared to other tokens. After alternating string_1 and it more human-like, the tokens that have the highest attention weights were the same but the attention weights of the token “can” showed a minor drop and shifted to the token “with”.
- String_3 and string_6 analysis: removing the repetitive parts in string_3 to the form string_6 enabled the BERT model to make a correct prediction. The attention weights of the previously heavily focused tokens “and” and “we” dropped while the token “provision” got the highest attention. However, this improvement was not a direct consequence from removing the repeated part, rather it was potentially because the word representations of some tokens in the repetitive part misled BERT and resulted in the wrong prediction. Therefore, by removing the repetitive parts, the BERT model shifted its attention toward other tokens, and assigned this string to the correct class by identifying the representation vectors of GPT-2 generated tokens.
- String_4 and string_7 analysis: in the CNN analysis section above, string_4 was truncated to 28 tokens long in order to fool the model to make a wrong prediction. In order to make BERT predict string_4 as a human-generated sentence, the document had to be truncated to only 16 tokens long. This suggests that BERT could use the first 17 word-representations in this sentence to make a correct prediction, and with only information of the first 16 tokens, it failed to collect enough information for an accurate prediction. From the plot of the multi-head attention weights on the first [CLS] token of string_4 in the appendix, we could see that the model assigned higher attention weights to the two

“and”s in string_4, but truncating string_4 to 16 tokens long removed these two words; thus, the model was forced to pay more attention on the first 16 tokens. Compared to the number of tokens truncated to fool the CNN model, we could still address that the fine-tuned BERT outperformed the CNN model.

Conclusion

Out of three classifiers from this study, BERT is certainly superior in fake text detection, followed by LR and finally the CNN. However, BERT also took the longest to train on truncated text, which was an hour and a half compared to mere seconds for LR. From the analysis, BERT is also better than CNN because it appears to be less affected by sequence length, as long as the entire sequence can be fed to BERT.

For future directions, fastText, RoBERTa, GLTR and even GPT-2 itself are all possible models to try on this study's dataset. [OpenAI](#) reported that RoBERTa achieved about 95% accuracy and performed better than generative models. With only 100k documents truncated to 128 tokens per document, would RoBERTa beat this study's BERT? With more time and resources, it would be interesting to see if any of these models would perform better than BERT.

References

- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. **BERT: Pre-training of deep bidirectional transformers for language understanding**. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pages 4171-4186, Minneapolis, Minnesota. Association for Computational Linguistics.
- Ashutosh Adhikari, Achyudh Ram, Raphael Tang, and Jimmy Lin., 2019. **Docbert: BERT for document classification**.
- Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever, 2018. **Improving language understanding with unsupervised learning**. Technical report, OpenAI.
- Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amode, Ilya Sutskever, 2018: **Language Models are Unsupervised Multitask Learners**
- Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever, 2019. **Release Strategies and the Social Impacts of Language Models**. Technical report, OpenAI.
- Ryan Ressmeyer, Sam Masling. Stanford University, 2018: **"Deep Faking" Political Twitter using Transfer learning and GPT-2**
- Alon Jacovi, Oren Sar Shalom, Yoav Goldberg, 2018 : **Understanding Convolutional Neural Networks for Text Classification** .Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP
- Jihang Mao¹, Wanli Liu (Montgomery Blair Highschool), 2018:**Factuality Classification Using the Pre-trained Language Representation Model BERT**
- Di Jin, Zhijing Jin, Joey Tianyi Zhou, Peter Szolovits., 2019 : **Is BERT Really Robust? A Strong Baseline for Natural Language Attack on Text Classification and Entailment**.
- Daphne Ippolito, Daniel Duckworth, Chris Callison-Burch, Douglas Eck., (University of Pennsylvania) , 2019 : **Human and Automatic Detection of Generated Text**
- Sebastian Gehrmann, Hendrik Strobelt, Alexander M. Rush, 2019: **GLTR: Statistical Detection and Visualization of Generated Text**

Appendix A - BERT Architecture

```
BertForSequenceClassification(  
  (bert): BertModel(  
    (embeddings): BertEmbeddings(  
      (word_embeddings): Embedding(30522, 768, padding_idx=0)  
      (position_embeddings): Embedding(512, 768)  
      (token_type_embeddings): Embedding(2, 768)  
      (LayerNorm): LayerNorm((768,), eps=1e-12,  
elementwise_affine=True)  
      (dropout): Dropout(p=0.1, inplace=False)  
    )  
    (encoder): BertEncoder(  
      (layer): ModuleList(  
        (0): BertLayer(  
          (attention): BertAttention(  
            (self): BertSelfAttention(  
              (query): Linear(in_features=768, out_features=768,  
bias=True)  
              (key): Linear(in_features=768, out_features=768,  
bias=True)  
              (value): Linear(in_features=768, out_features=768,  
bias=True)  
              (dropout): Dropout(p=0.1, inplace=False)  
            )  
            (output): BertSelfOutput(  
              (dense): Linear(in_features=768, out_features=768,  
bias=True)  
              (LayerNorm): LayerNorm((768,), eps=1e-12,  
elementwise_affine=True)  
              (dropout): Dropout(p=0.1, inplace=False)  
            )  
          )  
          (intermediate): BertIntermediate(  
            (dense): Linear(in_features=768, out_features=3072,  
bias=True)  
          )  
          (output): BertOutput(  
            (dense): Linear(in_features=3072, out_features=768,  
bias=True)  
            (LayerNorm): LayerNorm((768,), eps=1e-12,  
elementwise_affine=True)  
            (dropout): Dropout(p=0.1, inplace=False)  
          )  
        )  
      )  
    )  
  )  
)
```

```

(1): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768,
bias=True)
      (key): Linear(in_features=768, out_features=768,
bias=True)
      (value): Linear(in_features=768, out_features=768,
bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768,
bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072,
bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768,
bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(2): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768,
bias=True)
      (key): Linear(in_features=768, out_features=768,
bias=True)
      (value): Linear(in_features=768, out_features=768,
bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768,
bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
)

```

```

        )
    )
    (intermediate): BertIntermediate(
        (dense): Linear(in_features=768, out_features=3072,
bias=True)
    )
    (output): BertOutput(
        (dense): Linear(in_features=3072, out_features=768,
bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
    )
)
(3): BertLayer(
    (attention): BertAttention(
        (self): BertSelfAttention(
            (query): Linear(in_features=768, out_features=768,
bias=True)
            (key): Linear(in_features=768, out_features=768,
bias=True)
            (value): Linear(in_features=768, out_features=768,
bias=True)
            (dropout): Dropout(p=0.1, inplace=False)
        )
        (output): BertSelfOutput(
            (dense): Linear(in_features=768, out_features=768,
bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
        )
    )
    (intermediate): BertIntermediate(
        (dense): Linear(in_features=768, out_features=3072,
bias=True)
    )
    (output): BertOutput(
        (dense): Linear(in_features=3072, out_features=768,
bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
    )
)
(4): BertLayer(
    (attention): BertAttention(
        (self): BertSelfAttention(

```

```

        (query): Linear(in_features=768, out_features=768,
bias=True)
        (key): Linear(in_features=768, out_features=768,
bias=True)
        (value): Linear(in_features=768, out_features=768,
bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
        (dense): Linear(in_features=768, out_features=768,
bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
    )
    )
    (intermediate): BertIntermediate(
        (dense): Linear(in_features=768, out_features=3072,
bias=True)
    )
    (output): BertOutput(
        (dense): Linear(in_features=3072, out_features=768,
bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
    )
    )
    (5): BertLayer(
        (attention): BertAttention(
            (self): BertSelfAttention(
                (query): Linear(in_features=768, out_features=768,
bias=True)
                (key): Linear(in_features=768, out_features=768,
bias=True)
                (value): Linear(in_features=768, out_features=768,
bias=True)
                (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
                (dense): Linear(in_features=768, out_features=768,
bias=True)
                (LayerNorm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
                (dropout): Dropout(p=0.1, inplace=False)
            )
        )
        (intermediate): BertIntermediate(

```

```

        (dense): Linear(in_features=768, out_features=3072,
bias=True)
    )
    (output): BertOutput(
        (dense): Linear(in_features=3072, out_features=768,
bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
    )
)
(6): BertLayer(
    (attention): BertAttention(
        (self): BertSelfAttention(
            (query): Linear(in_features=768, out_features=768,
bias=True)
            (key): Linear(in_features=768, out_features=768,
bias=True)
            (value): Linear(in_features=768, out_features=768,
bias=True)
            (dropout): Dropout(p=0.1, inplace=False)
        )
        (output): BertSelfOutput(
            (dense): Linear(in_features=768, out_features=768,
bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
        )
    )
    (intermediate): BertIntermediate(
        (dense): Linear(in_features=768, out_features=3072,
bias=True)
    )
    (output): BertOutput(
        (dense): Linear(in_features=3072, out_features=768,
bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
    )
)
(7): BertLayer(
    (attention): BertAttention(
        (self): BertSelfAttention(
            (query): Linear(in_features=768, out_features=768,
bias=True)
            (key): Linear(in_features=768, out_features=768,

```

```

bias=True)
        (value): Linear(in_features=768, out_features=768,
bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
        (dense): Linear(in_features=768, out_features=768,
bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
    )
    )
    (intermediate): BertIntermediate(
        (dense): Linear(in_features=768, out_features=3072,
bias=True)
    )
    (output): BertOutput(
        (dense): Linear(in_features=3072, out_features=768,
bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
    )
    )
    (8): BertLayer(
        (attention): BertAttention(
            (self): BertSelfAttention(
                (query): Linear(in_features=768, out_features=768,
bias=True)
                (key): Linear(in_features=768, out_features=768,
bias=True)
                (value): Linear(in_features=768, out_features=768,
bias=True)
                (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
                (dense): Linear(in_features=768, out_features=768,
bias=True)
                (LayerNorm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
                (dropout): Dropout(p=0.1, inplace=False)
            )
        )
        (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072,
bias=True)
        )
    )

```

```

        (output): BertOutput(
          (dense): Linear(in_features=3072, out_features=768,
bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
    (9): BertLayer(
      (attention): BertAttention(
        (self): BertSelfAttention(
          (query): Linear(in_features=768, out_features=768,
bias=True)
          (key): Linear(in_features=768, out_features=768,
bias=True)
          (value): Linear(in_features=768, out_features=768,
bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (output): BertSelfOutput(
          (dense): Linear(in_features=768, out_features=768,
bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (intermediate): BertIntermediate(
        (dense): Linear(in_features=768, out_features=3072,
bias=True)
      )
      (output): BertOutput(
        (dense): Linear(in_features=3072, out_features=768,
bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (10): BertLayer(
      (attention): BertAttention(
        (self): BertSelfAttention(
          (query): Linear(in_features=768, out_features=768,
bias=True)
          (key): Linear(in_features=768, out_features=768,
bias=True)
          (value): Linear(in_features=768, out_features=768,
bias=True)

```



```

        (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
        (dense): Linear(in_features=768, out_features=768,
bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
    )
    )
    (intermediate): BertIntermediate(
        (dense): Linear(in_features=768, out_features=3072,
bias=True)
    )
    (output): BertOutput(
        (dense): Linear(in_features=3072, out_features=768,
bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
    )
    )
    (11): BertLayer(
        (attention): BertAttention(
            (self): BertSelfAttention(
                (query): Linear(in_features=768, out_features=768,
bias=True)
                (key): Linear(in_features=768, out_features=768,
bias=True)
                (value): Linear(in_features=768, out_features=768,
bias=True)
                (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
                (dense): Linear(in_features=768, out_features=768,
bias=True)
                (LayerNorm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
                (dropout): Dropout(p=0.1, inplace=False)
            )
        )
        (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072,
bias=True)
        )
        (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768,
bias=True)

```

```
        (LayerNorm): LayerNorm((768,), eps=1e-12,  
elementwise_affine=True)  
        (dropout): Dropout(p=0.1, inplace=False)  
    )  
    )  
    )  
    )  
    (pooler): BertPooler(  
        (dense): Linear(in_features=768, out_features=768, bias=True)  
        (activation): Tanh()  
    )  
    )  
    (dropout): Dropout(p=0.1, inplace=False)  
    (classifier): Linear(in_features=768, out_features=2, bias=True)  
    )
```

Appendix B - CNN Training and Validation Results

Epoch: 0%| | 0/20 [00:00<?, ?it/s]
Epoch: 5%| | 1/20 [00:50<16:08, 50.95s/it]

Epoch: 01 | Time: 0m 33s

Epoch [1/20], Train Loss: 0.6163, Validation Loss: 0.6705, Validation Accuracy: 0.5815, Validation F1: 0.4993

Epoch: 10%| | 2/20 [01:38<14:59, 49.97s/it]

Epoch: 02 | Time: 0m 32s

Epoch [2/20], Train Loss: 0.5944, Validation Loss: 0.6342, Validation Accuracy: 0.6254, Validation F1: 0.5769

Epoch: 15%| | 3/20 [02:26<13:58, 49.35s/it]

Epoch: 03 | Time: 0m 32s

Epoch [3/20], Train Loss: 0.5799, Validation Loss: 0.6162, Validation Accuracy: 0.6469, Validation F1: 0.6103

Epoch: 20%| | 4/20 [03:01<12:00, 45.01s/it]


Epoch: 04 | Time: 0m 33s

Epoch [4/20], Train Loss: 0.5663, Validation Loss: 0.5998, Validation Accuracy: 0.6696, Validation F1: 0.6418

Epoch: 25%| | 5/20 [03:37<10:34, 42.30s/it]


Epoch: 05 | Time: 0m 33s

Epoch [5/20], Train Loss: 0.5524, Validation Loss: 0.5829, Validation Accuracy: 0.6925, Validation F1: 0.6715

Epoch: 30% | 6/20 [04:12<09:20, 40.04s/it]


Epoch: 06 | Time: 0m 32s

Epoch [6/20], Train Loss: 0.5373, Validation Loss: 0.5693, Validation Accuracy: 0.7116,
Validation F1: 0.6951

Epoch: 35% | 7/20 [04:46<08:19, 38.42s/it]


Epoch: 07 | Time: 0m 32s

Epoch [7/20], Train Loss: 0.5225, Validation Loss: 0.5645, Validation Accuracy: 0.7179,
Validation F1: 0.7013

Epoch: 40% | 8/20 [05:22<07:31, 37.61s/it]


Epoch: 08 | Time: 0m 33s

Epoch [8/20], Train Loss: 0.5087, Validation Loss: 0.5510, Validation Accuracy: 0.7359,
Validation F1: 0.7229

Epoch: 45% | 9/20 [05:57<06:44, 36.75s/it]


Epoch: 09 | Time: 0m 32s

Epoch [9/20], Train Loss: 0.4916, Validation Loss: 0.5406, Validation Accuracy: 0.7491,
Validation F1: 0.7379

Epoch: 50% | 10/20 [06:31<06:01, 36.15s/it]

Epoch: 10 | Time: 0m 32s


Epoch [10/20], Train Loss: 0.4752, Validation Loss: 0.5365, Validation Accuracy: 0.7535,
Validation F1: 0.7423

Epoch: 55% | 11/20 [07:06<05:21, 35.76s/it]

Epoch: 11 | Time: 0m 32s


Epoch [11/20], Train Loss: 0.4583, Validation Loss: 0.5265, Validation Accuracy: 0.7668,

Validation F1: 0.7577

Epoch: 60% | 12/20 [07:42<04:46, 35.81s/it]


Epoch: 12 | Time: 0m 33s

Epoch [12/20], Train Loss: 0.4421, Validation Loss: 0.5209, Validation Accuracy: 0.7725,
Validation F1: 0.7640

Epoch: 65% | 13/20 [08:17<04:08, 35.46s/it]


Epoch: 13 | Time: 0m 32s

Epoch [13/20], Train Loss: 0.4256, Validation Loss: 0.5094, Validation Accuracy: 0.7883,
Validation F1: 0.7820

Epoch: 70% | 14/20 [08:52<03:31, 35.22s/it]


Epoch: 14 | Time: 0m 32s

Epoch [14/20], Train Loss: 0.4107, Validation Loss: 0.5098, Validation Accuracy: 0.7857,
Validation F1: 0.7788

Epoch: 75% | 15/20 [09:26<02:55, 35.04s/it]


Epoch: 15 | Time: 0m 32s

Epoch [15/20], Train Loss: 0.3977, Validation Loss: 0.5059, Validation Accuracy: 0.7905,
Validation F1: 0.7844

Epoch: 80% | 16/20 [10:02<02:21, 35.41s/it]


Epoch: 16 | Time: 0m 34s

Epoch [16/20], Train Loss: 0.3861, Validation Loss: 0.4993, Validation Accuracy: 0.7977,
Validation F1: 0.7927

Epoch: 85% | 17/20 [10:37<01:45, 35.26s/it]


Epoch: 17 | Time: 0m 32s

Epoch [17/20], Train Loss: 0.3767, Validation Loss: 0.4947, Validation Accuracy: 0.8040,
Validation F1: 0.7996

Epoch: 90% | 18/20 [11:12<01:10, 35.09s/it]


Epoch: 18 | Time: 0m 32s

Epoch [18/20], Train Loss: 0.3686, Validation Loss: 0.4898, Validation Accuracy: 0.8079,
Validation F1: 0.8043

Epoch: 95% | 19/20 [11:48<00:35, 35.27s/it]

Epoch: 19 | Time: 0m 33s

Epoch [19/20], Train Loss: 0.3605, Validation Loss: 0.4912, Validation Accuracy: 0.8063,
Validation F1: 0.8024

Epoch: 100% | 20/20 [12:23<00:00, 37.15s/it]

Epoch: 20 | Time: 0m 32s

Epoch [20/20], Train Loss: 0.3543, Validation Loss: 0.4874, Validation Accuracy: 0.8098,
Validation F1: 0.8065

Appendix C - Training Results on different MSL

1. Training results for fine-tuning BERT model with maximum length = 32.

	Training Loss	Valid. Loss	Valid. Accur.	Training Time	Validation Time
epoch					
1	0.248571	0.262627	0.89880	0:08:34	0:00:43
2	0.133522	0.359143	0.88800	0:08:35	0:00:43
3	0.073083	0.372432	0.90635	0:08:34	0:00:43
4	0.046404	0.441291	0.90610	0:08:32	0:00:43

2. Classification report and confusion matrix for BERT model with max_len = 32

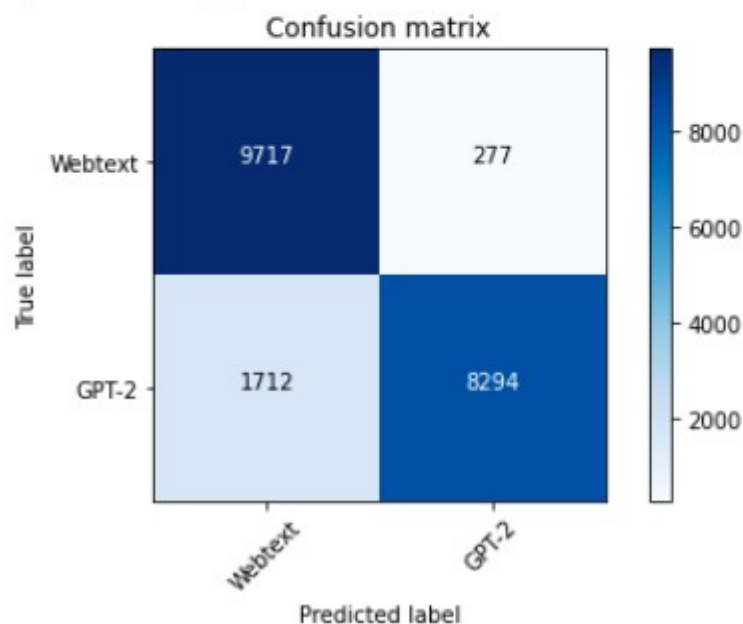
Macro F1 score: 0.9000439904438136

Classification report:

	precision	recall	f1-score	support
GPT-2	0.97	0.83	0.89	10006
Webtext	0.85	0.97	0.91	9994
accuracy			0.90	20000
macro avg	0.91	0.90	0.90	20000
weighted avg	0.91	0.90	0.90	20000

Confusion matrix, without normalization

```
[[9717  277]
 [1712 8294]]
```



3. Training results for fine-tuning BERT model with maximum length = 64

	Training Loss	Valid. Loss	Valid. Accur.	Training Time	Validation Time
epoch					
1	0.305080	0.244534	0.90945	0:13:46	0:01:20
2	0.130088	0.388695	0.88620	0:13:53	0:01:20
3	0.065183	0.343333	0.92395	0:13:53	0:01:20
4	0.031288	0.489703	0.91265	0:13:50	0:01:20

4. Classification report and confusion matrix for BERT model with max_len = 64

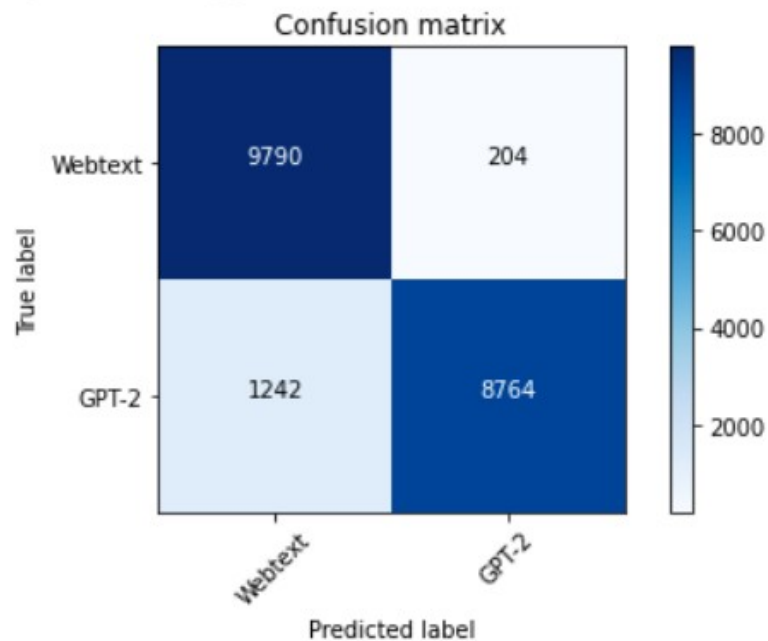
Macro F1 score: 0.9275092267569641

Classification report:

	precision	recall	f1-score	support
GPT-2	0.98	0.88	0.92	10006
Webtext	0.89	0.98	0.93	9994
accuracy			0.93	20000
macro avg	0.93	0.93	0.93	20000
weighted avg	0.93	0.93	0.93	20000

Confusion matrix, without normalization

```
[[9790  204]
 [1242 8764]]
```



5. Training results for fine-tuning BERT model with maximum length = 128

	Training Loss	Valid. Loss	Valid. Accur.	Training Time	Validation Time
epoch					
1	0.261006	0.303544	0.90060	0:24:36	0:02:35
2	0.100613	0.378018	0.90885	0:24:45	0:02:35
3	0.046581	0.355956	0.92775	0:24:43	0:02:35
4	0.018528	0.534139	0.91460	0:24:38	0:02:36

6. Classification report and confusion matrix for BERT model with max_len = 128

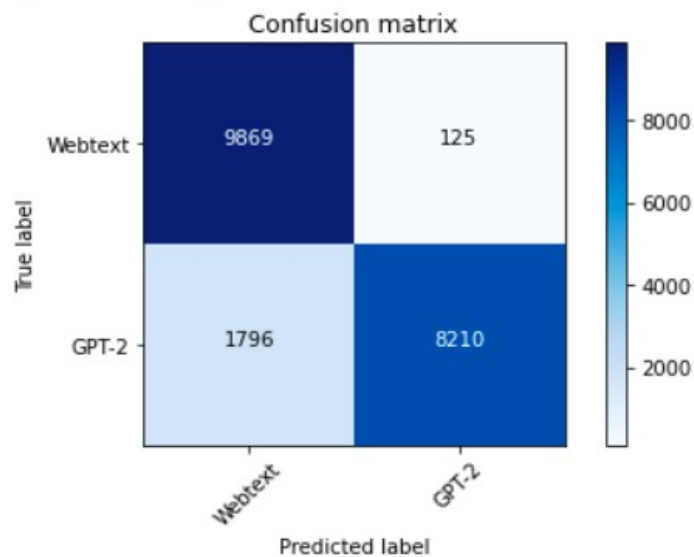
Macro F1 score: 0.9032845296211757

Classification report:

	precision	recall	f1-score	support
GPT-2	0.99	0.82	0.90	10006
Webtext	0.85	0.99	0.91	9994
accuracy			0.90	20000
macro avg	0.92	0.90	0.90	20000
weighted avg	0.92	0.90	0.90	20000

Confusion matrix, without normalization

```
[[9869 125]
 [1796 8210]]
```



5. Training results for fine-tuning BERT model with maximum length = 256

	Training Loss	Valid. Loss	Valid. Accur.	Training Time	Validation Time
epoch					
1	0.249846	0.226071	0.90760	0:24:25	0:02:39
2	0.076881	0.355969	0.90820	0:23:41	0:02:30
3	0.028838	0.366751	0.93460	0:23:48	0:02:30
4	0.010618	0.543406	0.92125	0:23:46	0:02:30

6. Classification report and confusion matrix for BERT model with max_len = 256

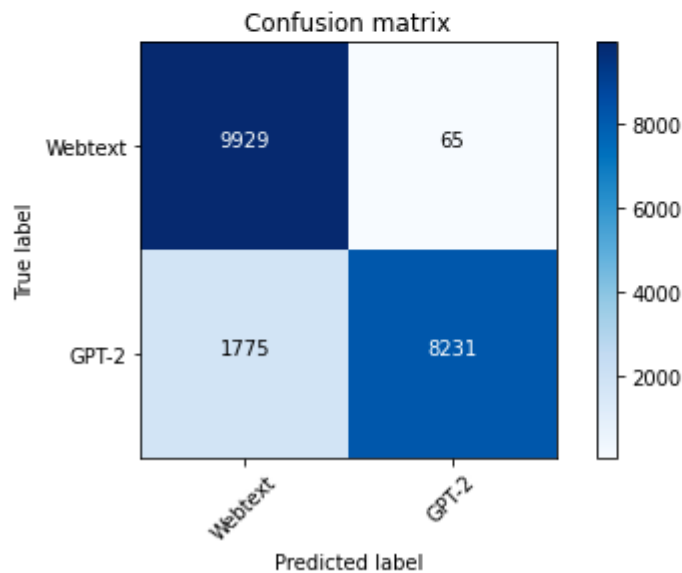
Macro F1 score: 0.9073320484787553

Classification report:

	precision	recall	f1-score	support
GTP-2	0.99	0.82	0.90	10006
Webtext	0.85	0.99	0.92	9994
accuracy			0.91	20000
macro avg	0.92	0.91	0.91	20000
weighted avg	0.92	0.91	0.91	20000

Confusion matrix, without normalization

```
[[9929  65]
 [1775 8231]]
```



Appendix D - GPT-2 Test Strings for CNN and BERT

- GPT-2_string_1 is 38 words long. The text: "Scientists can now visualize and experiment with structures and dynamics of complex molecular structures (at atomic-level precision), with real-time multi-user collaboration via the cloud

July 6, 2018"

The prediction for GPT-2_string_1 is 1 and the model was 0.6872% confident.

- GPT-2_string_2 is 42 words long. The text: "Pro Tour Amonkhet is now down to just eight players. Take a look at what they brought for this weekend's Standard rounds, and what they will be battling with on Sunday for the title of Pro Tour Champion."

The prediction for GPT-2_string_2 is 1 and the model was 0.8143% confident.

- GPT-2_string_3 is 169 words long. The text: "Summary Focus of this page: This page discusses our current view of the evidence for a wide range of programs and interventions that aim to improve education in developing countries. These include demand-side interventions that lower the cost of schooling or increase its (perceived) returns, provision of school inputs, pedagogy interventions, and governance reforms. We focus mainly on interventions aimed at improving primary and secondary education but consider vocational training interventions briefly. We have not yet completed a report on early childhood (pre-school) interventions. On this page, we focus on evidence from experimental study designs.\n\nThis page discusses our current view of the evidence for a wide range of programs and interventions that aim to improve education in developing countries. These include demand-side interventions that lower the cost of schooling or increase its (perceived) returns, provision of school inputs, pedagogy interventions, and governance re..."

The prediction for GPT-2_string_3 is 0 and the model was 0.8338% confident.

- GPT-2_string_4 is 169 words long. The text: "If you look on a board game shelf, how many games will you see with actions based on collaboration, stewardship, generosity, and gratitude? Most likely, you'll find mechanics like attacking, stealing, and backstabbing. Indigenous communities looking to facilitate intergenerational gameplay are thus hard-pressed to find options that reinforce their teachings. In response, communities are developing their own games for passing on teachings in many forms. As espoused by game designer Brenda Romero, the mechanic is the message. And the messages in the board game The Gift of Food—inspired by collaborative game development with Indigenous communities working with the Northwest Indian College—produce culturally responsive gameplay, meaning gameplay that is drawn from and that uplifts the cultures involved.\n\nThe Gift of

Food is an ideal example of how culturally responsive board games can function as important pathways for passing on Indigenous ways of knowing, as learning and reinforcing..."

The prediction for GPT-2_string_4 is 0 and the model was 0.9623% confident.

- GPT-2_string_5 is 39 words long. The text: "Scientists can now visualize and experiment with structures and the dynamics of complex molecular makeup (at atomic-level precision), with real-time multi-user collaboration via the cloud

July 6, 2018"

The prediction for GPT-2_string_5 is 1 and the model was 0.7422% confident.

- GPT-2_string_6 is 113 words long. The text: "Summary Focus of this page: This page discusses our current view of the evidence for a wide range of programs and interventions that aim to improve education in developing countries. These include demand-side interventions that lower the cost of schooling or increase its (perceived) returns, provision of school inputs, pedagogy interventions, and governance reforms. We focus mainly on interventions aimed at improving primary and secondary education but consider vocational training interventions briefly. We have not yet completed a report on early childhood (pre-school) interventions. On this page, we focus on evidence from experimental study designs."

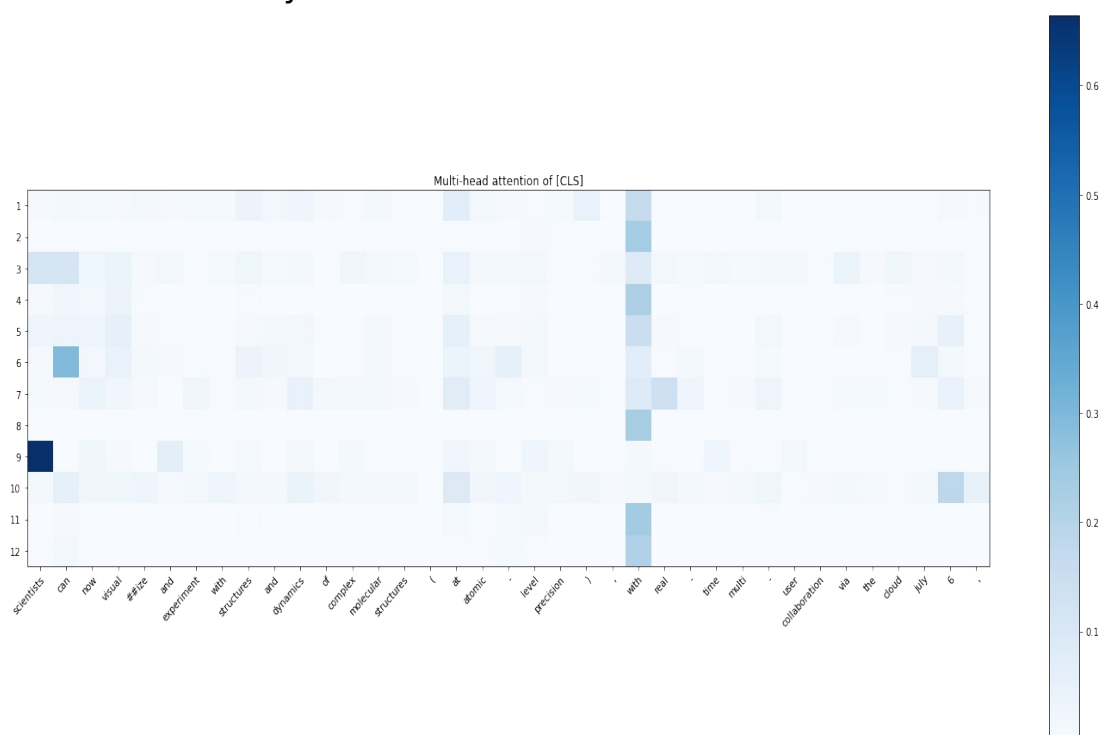
The prediction for GPT-2_string_6 is 1 and the model was 0.6829% confident.

- GPT-2_string_7 is 28 words long. The text: "If you look on a board game shelf, how many games will you see with actions based on collaboration, stewardship, generosity, and gratitude?"

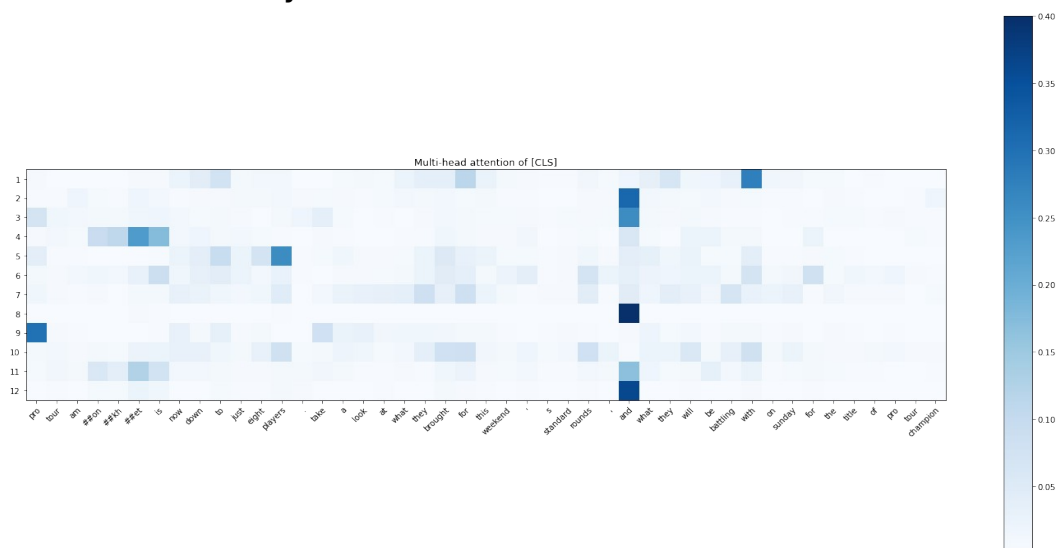
The prediction for GPT-2_string_7 is 1 and the model was 0.6283% confident.

Appendix E - Multi-head Attention Weights on the First [CLS] Token

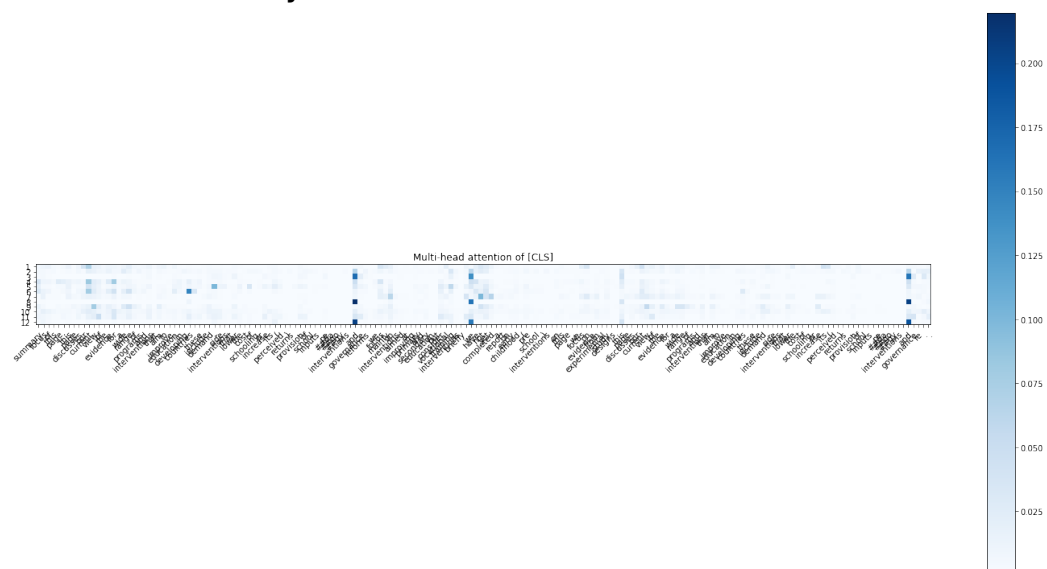
1. String_1 Multi-head attention weights on the first [CLS] token of the last hidden layer



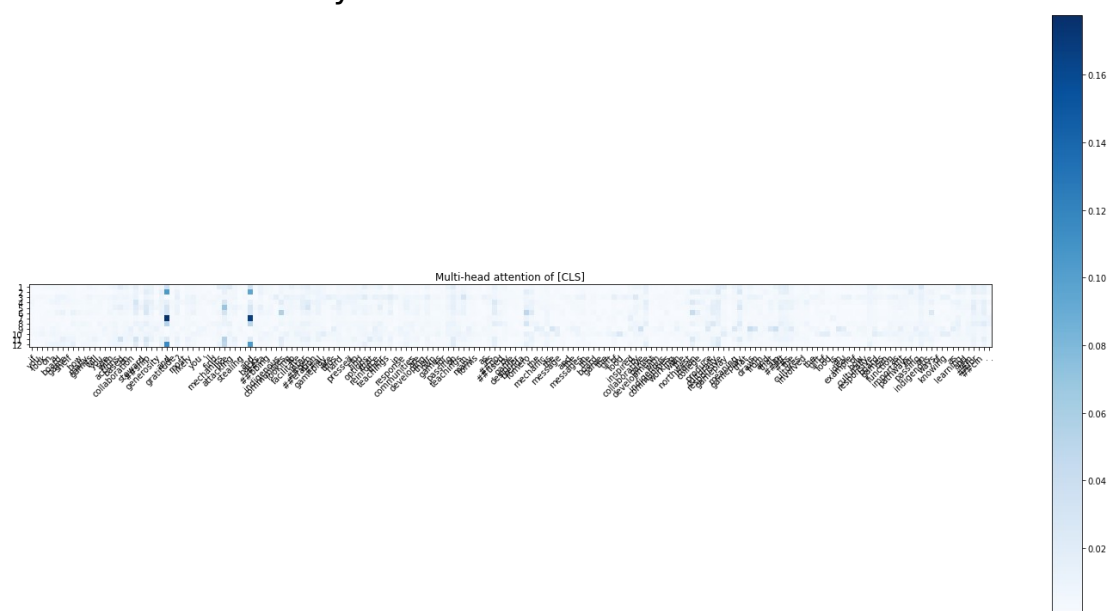
2. String_2 Multi-head attention weights on the first [CLS] token of the last hidden layer



3. String_3 Multi-head attention weights on the first [CLS] token of the last hidden layer



4. String_4 Multi-head attention weights on the first [CLS] token of the last hidden layer



6. String_7 Multi-head attention weights on the first [CLS] token of the last hidden layer. This test string is resulted from truncating string_4 to only 16 tokens long.

