



‘Tackling AI Text Generators’

Alex Ivanov

Contents

Contents.....	2
1 - Planning the project.....	3
1.1 Brief.....	3
1.2 Project Aim.....	4
1.3 Solutions.....	5
1.4 Developing a plan.....	9
2 - Throughout the project.....	11
2.1 Design of solution.....	11
2.2 Implementation.....	18
3 - Finalising the project.....	30
3.1 Evaluation.....	30
3.2 Reflection.....	36
3.3 Conclusion.....	37
References.....	38

1 - Planning the project

1.1 Brief

Throughout 2023, everyone has experienced an exponential increase in the use of services such as ChatGPT - first developed by OpenAI in 2018, where they use the Generative Pre-trained Transformers (GPT) framework to create novel human-like text. As more GPT foundation models are released, using more training data than the last model and parameter count increasing to hundreds of billions, the problem of determining human text between generated text becomes more and more prominent.

We can already see how such technologies affect us - good or bad. For example, news articles such as 'BuzzFeed' have begun to rely on AI to help publish travel guides (Peters 2023). Although only experimental, it sheds light on generative AI's potential in improving writing-dominated sectors. On top of this, GPT-4 is also being used to revolutionise how legal services are used and how accessible they are, demonstrated by services such as DoNotPay (Kelley 2018), which generates lawsuits against robocalls in seconds. Therefore, AI can improve sectors, including industries such as education, by helping educators quickly create personalised content that is socio-culturally relevant for their students and help motivate and assess learners in new ways.

However, issues such as cheating and plagiarism have also become prevalent. Students can now easily complete coursework and homework assignments with AI text generators - discrediting most students' hard work. Just over half of students believe using AI tools to complete tasks and exams counts as cheating (Johnson 2023), exemplifying how this poses a significant threat to the integrity of academia. Moreover, there continue to be more methods of fooling AI detection tools released daily, including GPT-Minus1 (Tahir 2023), and students even training AI models to mimic their writing style, making detection even more challenging.

So, how can we combat this and successfully devise a method to identify 'real' human text from AI-generated content?

1.2 Project Aim

This project aims to design an accessible, suitable system for detecting generated text, successfully determining whether it was human-written or artificially generated. A final product could be applied in educational settings and essay-based competitions, which can successfully distinguish genuine academic work - deterring the use of AI for such applications.

Goals

- ❖ I have researched and assessed existing solutions for AI text classifiers.
- ❖ I have successfully created an efficient AI content detector.
- ❖ I have evaluated my solution to analyse its success.
- ❖ I have suggested other potential methods of tackling AI content detection, which may be more successful.

Objectives

- ❖ Research and choose a variety of existing AI content detectors to assess.
- ❖ Design a suitable way to assess the existing solutions.
- ❖ Research methods for creating an AI content detector and identifying suitable approaches.
- ❖ Formulate a plan to build a successful AI classifier based on previous research.
- ❖ Design a user interface, followed by an interview to ensure its success.
- ❖ Produce a fully functional AI text classifier, which receives text input and outputs the likeliness of AI assistance.
- ❖ Form any conclusions and evaluations after implementation

1.3 Solutions

Under this section, I will research and assess existing solutions. After this, I will propose what approaches I can take towards building my system. To clarify, assessing existing solutions allows me to build on the range of approaches I can take towards building my classifier and consider the front-end development of my project.

Assess criteria

To assess each promising solution, I will have to devise a method to measure the success of each system. Using a LinkedIn article (Nelson 2022) as an initial framework would be a good start. The article details the reliability of AI detection tools and 'Fooling AI Detection Tools', which may attack the methods used to build such AI tools. The reliability of each system would be worth exploring as this could be a criterion to analyse my solution at the end of the project. However, including a criterion such as 'employing strategies to fool AI detection tool' may be too tedious and difficult to test since there would be many methods to trick an AI detector. In addition, with reliability, accuracy would also be a good measure. Besides the analysis of how successful a detector is, it's worth evaluating the usability/accessibility of each existing solution, as anyone should ideally be able to interact and use my AI classifier easily.

OpenAI classifier

Considering I first mentioned ChatGPT, it would be worth evaluating OpenAI's current attempt to create a text classifier. According to OpenAI in their blog (OpenAI 2023) titled, "New AI classifier for indicating AI-written text", their text classifier is a fine-tuned GPT model that predicts how likely an AI-generated piece of text, including ChatGPT and AI plagiarism. However, it's unreliable and inaccurate, as shown by the tests they conducted - "In our evaluations on a 'challenge set' of English texts, our classifier correctly identifies 26% of AI-written text (true positives) as 'likely AI-written,' while incorrectly labelling human-written text as AI-written 9% of the time (false positives)".

Unfortunately, their classifier is unavailable; however, thanks to the YouTube channel 'GPT-3 Demo' (GPT-3 Demo 2022), we can see a live demonstration of using it.

In Aldous Huxley's novel "Brave New World," Mustapha Mond is portrayed as a powerful and mysterious figure. The novel depicts a dystopian society in which the government, led by Mond, maintains strict control over its citizens through the use of advanced technology and manipulation of emotions. Despite this, I argue that Mond should be viewed positively for three key reasons: his efforts to maintain stability in society, his recognition of the limitations of happiness, and his belief in individual freedom.

Firstly, Mond's role as World Controller is to maintain stability in society. He recognizes that in order for society to function, there must be a balance between individual desires and the needs of the community. He also understands that in order to maintain this balance, it is necessary to control certain aspects of society, such as the use of technology and the manipulation of emotions. This is evident in his decision to ban literature, which he believes will cause dissent and disrupt the stability of society. In this way, Mond can be seen as a pragmatic leader who is willing to make difficult decisions for the greater good.

By submitting content, you agree to our [Terms of Use](#) and [Privacy Policy](#). Be sure you have appropriate rights to the content before using the AI Text Classifier.

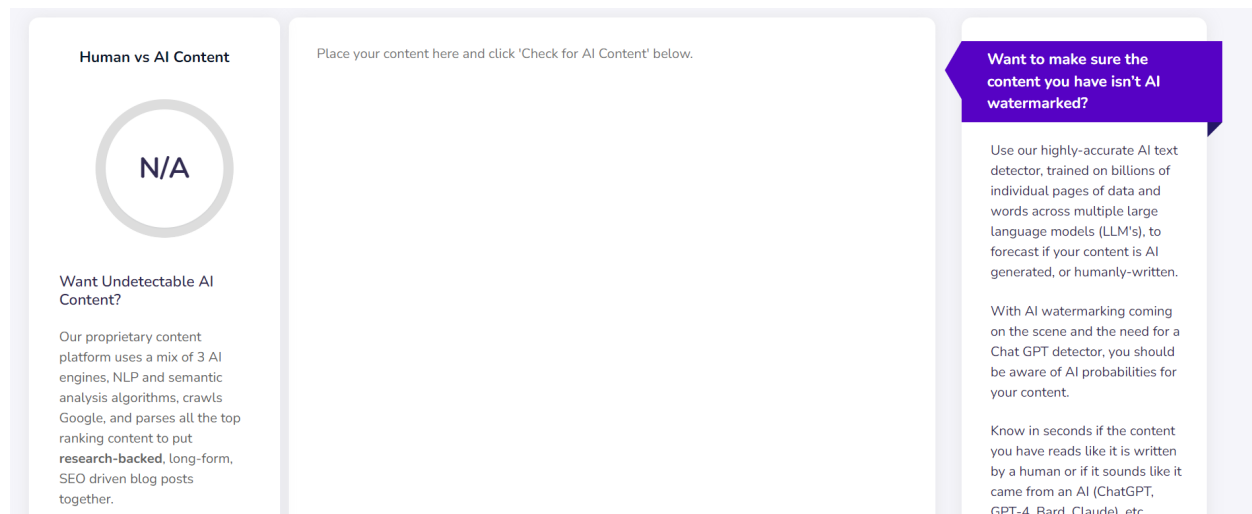
The classifier considers the text to be **possibly** AI-generated.

(Fig 1.3.1 - screenshot of YouTube video demonstrating OpenAI's classifier)

This clear and practical design uses an expandable text box, followed by interactable buttons to clear the text and submit, which then outputs to the user whether it's 'possibly AI-generated' or not.

Content at Scale classifier

Another example I'd like to analyse is 'Content at Scale' (Content at Scale 2021) (CAS). They claim their AI content detector to be 98% accurate - a significant improvement compared to the previous example. However, according to this blog, it was pretty unreliable when analysing human-written text, "In total, the tool was successful 12/20 times, which is only 60%", which is still better than OpenAI's version. Although this is a higher percentage, we should question the authenticity behind their tests since only 20 trials were conducted. The unsatisfactory number of trials may misrepresent the classifier's true reliability.



(Fig 1.3.2 - CAS AI content detector)

CAS similarly uses a textbox for which users are meant to input text (up to 25,000 characters) and outputs a score on the scale (from 0% to 100%) shown on the left-hand side. This method of displaying the likeliness of the text being AI-generated compared to an output of a sentence or two is more precise. A negative could be that users may be confused; for example, “10% of AI content detected”, and the user may think that to be negative as they expect it to be zero. Including an additional output, as shown in the openAI classifier, would reassure the user.

Also, CAS include a summary on the right-hand side of what processes were done to make their AI text detector “trained on billions of individual pages of data” and using “multiple large language models”. This does sound very effective, and initially, I was planning to use an approach as such; however, gathering billions of individual pages of data and words from the internet may seem unfeasible since collecting such data may be too expensive, or I have insufficient technical requirements to process such data. In addition, collecting so much data also requires the correct procedures to ensure the safety of all data - unless it's all public information. It would also require me to comply with the Data Protection Act 2018 (DPA) that “controls how your personal information is used by organisations, businesses or the government”, which also includes following the ‘data protection principles’.

Methods

This section will explore the more technical approaches towards the system's back end. I will use originality.ai's (Gillham, n.d.) blog to start. Originality.ai is also famous for its accurate (94% accuracy) detection tool, and it acts as a plagiarism tool aimed towards writers/creators who want to ensure their work is unique.

Important details noted from their article:

- The tool's core is based on a modified version of the BERT model. BERT, also known as Bidirectional Encoder Representations from Transformers. This was because a discriminate model was more powerful in detection than a generative model.
- The larger the model used, the more accurate and robust the predictions will be.
- Then, fine-tune the model on a training dataset. In originality's scenario, they used a dataset of up to millions of samples.
- Since the quality of the model depends a lot on the input data, it's essential to consider the methods for collecting and processing large amounts of actual text from many sources.

Other suitable approaches to developing my AI content detector are mentioned in a levelup article (Biswas 2023):

- Using natural language processing (NLP) techniques to analyse the text's structure and syntax to determine its origin. Like Grammarly, NLP can measure a text's coherence, complexity, diversity, sentiment, or emotion.
- Using statistical methods to compare the text with a large corpus of human-written texts to find anomalies or deviations. This would include using mathematical models and formulas to analyse data and find patterns or trends. This could lead to using perplexity to measure how well a model can predict the next word in a text.

A similar article by Quetext (guide 2023) also discusses using perplexity to measure AI interference. On top of calculating perplexity, we can also measure burstiness, a sudden increase in the quality of a piece of writing and coherence. The positives of this strategy are the simplicity behind implementing these formulas into a program and that it is arguably more reliable than using a neural network. This is because using a model heavily relies on the data used to train it. The model will likely provide a worse prediction in a scenario with an input different to the training dataset. On the other hand, using machine learning for natural language processing has promising potential for more accurate predictions, especially as most successful current AI content detectors somewhat incorporate it. The only negative to this is collecting samples and gathering a large enough dataset to be accurate, as well as pre-training and fine-tuning my model since I will likely need to hire a cloud TPU for efficient training.

Besides developing the AI, users must select the text they want to check. One method is using a text box in which a user can either type or paste in the text for submitting the text to be scanned. This would allow users to easily paste in large chunks of text and directly type into the text box. However, it does mean that the user must use another tab for my application, which can create a clunky workspace. To minimise a clunky workspace, which creates a better user experience, we could explore creating a Chrome extension in which users highlight text on the page they want to check. This means that my system would be very quick to access and, in turn, be more effective. However, this may deter people from checking large pieces of text as it's easier to highlight a small amount of text than a larger amount, which affects the quality of the prediction and can become unreliable and inaccurate.

1.4 Developing a plan

After carefully considering my aims and objectives, as well as researching other existing solutions, to achieve our aim of “tackling AI text generators”, I will take the following approach:

- Conduct further research on algorithms and strategies for creating the model
- Create a user-friendly GUI for entering text, which will be passed to the model
- Gather and process data to be used for training the learning model
- Create a model that successfully predicts if the content is AI-generated or likely human-written
- Conclude the project, reflect on the project and how I did by using evaluation metrics

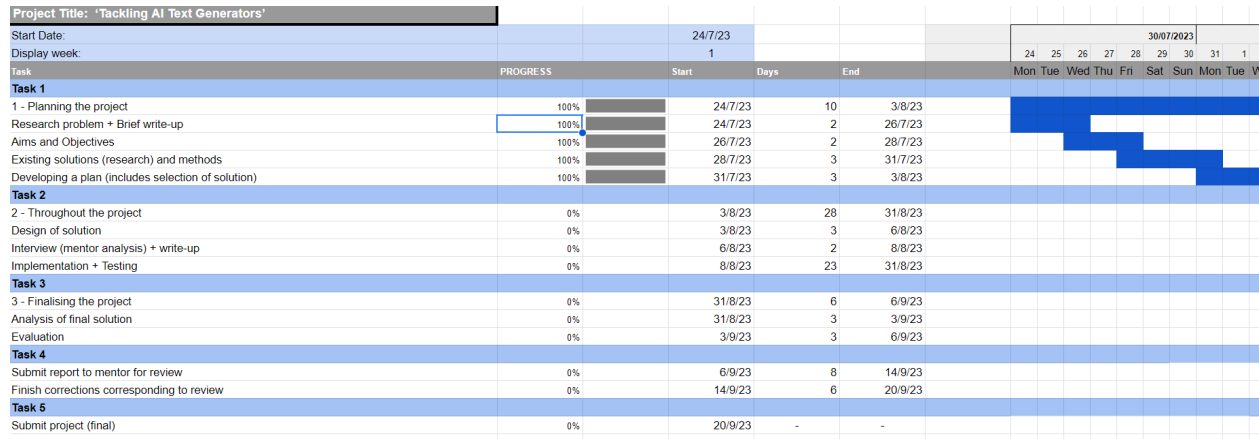
Most information for this project can be gathered from online - more specifically, using articles and YouTube. For tasks that may benefit more input from an outsider's perspective, for example, creating the user-friendly GUI, I will be interviewing my mentor (Joe Nicholson) since they are a valuable stakeholder. My project mentor is a teacher, so this is an excellent opportunity to see how it can be used in education and evaluate my project's success - increasing the impact of my work.

I also wanted to make the most of this project by better educating myself on machine learning as much as possible and potentially impacting education and competition by providing a free, accessible tool. I will use accessible tools by developing my model using ‘Google Colab’, a ‘hosted Jupyter Notebook service that requires no setup and provides free access to computing resources, including GPUs and TPUs.’ For developing a model built using a neural network, training it using a GPU would significantly improve training speeds since they can process multiple parallel tasks, thanks to their multiple cores - maximising the project's overall efficiency.

At the end of this project, I want to have created a functional AI content detector to fulfil the project aim and have the potential to be used by teachers and organisations to maintain academic integrity.

Organisation

I will use a Gantt chart to plan and organise my time effectively during this project - pivotal since I am conducting this project primarily by myself alongside occasional support from my mentor.



(Fig 1.4.1 - evidence of Gantt chart used)

I used the Gantt chart to give dates (start/end) for each work section; see Fig 1.4.1.

I structured my project logically into three areas (planning, throughout, and finalising the project), which made the most sense and included additional milestones for reviewing/refining the project. Also, under each section, the main task (for each large area) had been decomposed into smaller objectives to visualise better what needs to be done during each section.

For dependencies, such as 'submit report to mentor for review', I had given a kind chunk of time for my mentor to ensure they have enough time, on top of their work, for review. This serves as a contingency in case they can't review it immediately.

Many research/planning tasks (e.g., project planning) have been allocated only a few days since I have researched and collected resources. So, I only need to write it up - hence, it doesn't need to take long. At the same time, most tasks overlap, which acts as a contingency in case I don't finish a task by the due date in case I encounter any issues or am busy.

Fig 1.4.1 also shows that a task such as 'implementing + testing' has significantly more days devoted to it because this will be the central core of the project. Therefore, I will likely encounter most problems and need time to refine and implement the system. Giving myself this extended timeframe for this task means I can safely finish it by the deadline, 'submit project (final)'.

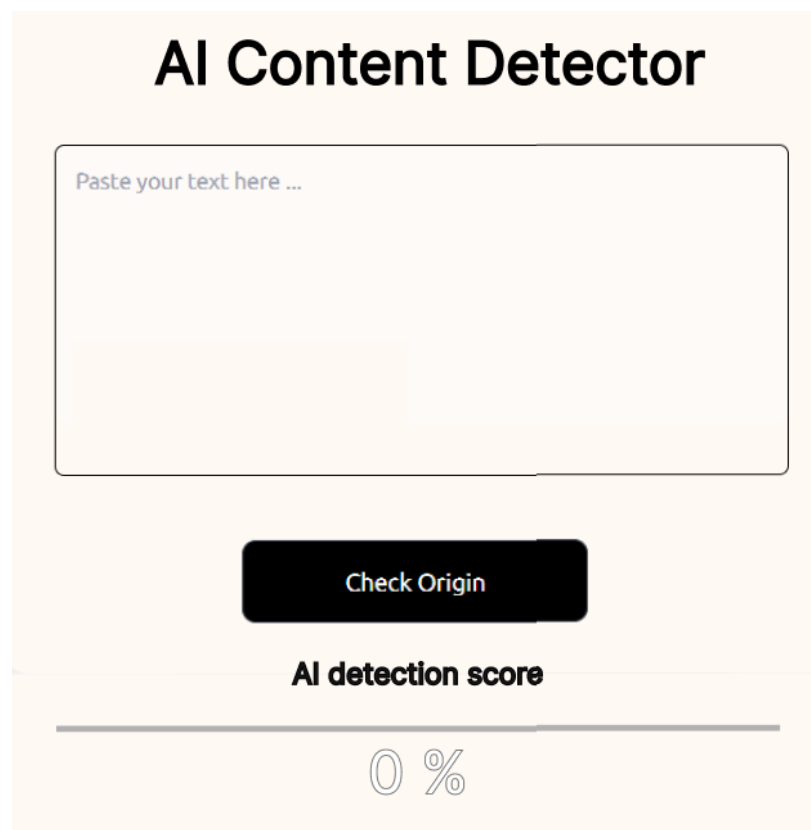
2 - Throughout the project

This section focuses more technically on designing and implementing the algorithms required, as well as designing and implementing other relevant parts of the solution.

2.1 Design of solution

User interface

Using Figma, I first created a first draft of a user interface that can be used for pasting text, then submitted to see the prediction score for that text being AI-generated.



(Fig 2.1.1 - first draft of UI)

To better ensure my project's overall success and not miss any important features, I interviewed my mentor (a teacher) and used their feedback to refine my user interface.

Interview questions

- 1) What measures have you taken to tackle AI content generation throughout this year?
- 2) What features would you change/add to the AI content detector?
- 3) How user-friendly would you say the UI is on a scale of 1-10? If under 7, what would you suggest to make it more user-friendly?

Interview answers

- 1) I have had to deal with a fair amount of AI generation code/work. I tend to cut and paste the content in ChatGPT and ask if it wrote it!
- 2) Features: if a text is AI generated, can you give an indication of where it was generated?
- 3) UI rating, given that it is so simple it is hard to rate. You may want to add buttons to clear the text box so that information is not pasted twice.

Analysis of interview

It's evident that ChatGPT poses an issue, as teachers have had to start combatting cheating from students using the model to complete assignments. Interestingly, ChatGPT could be used to detect AI-generated content. However, it may not be best suited for this task as the model has been trained to predict the next word in a sequence, also known as sequence-to-sequence learning, hence more suitable for generating content. Also, more specifically, ChatGPT is a decoder-only transformer model; decoders are typically used to convert a previously generated vector into a recognisable type.

I also understood that the user interface is too simple and planned to add the following features:

- Clearing text in the textbox
- Provide examples of AI-generated and human-written text
- Highlighting parts of the text which indicate AI interference
 - Benefit is it makes it easier for the user to also make a judgement by looking at that specific part

Reviewed user interface

In Fig 2.1.2, the layout demonstrated two buttons which can be pressed to fill the textbox with an example then. Fig 2.1.3 shows what would appear after submitting the text, including a prompt below the score to clarify what a high score means - in our scenario, this indicates the text is likely AI-generated. See the figures below.

The image shows the initial state of the 'AI CONTENT DETECTOR' application. At the top, the title 'AI CONTENT DETECTOR' is centered. Below it, there are two tabs: 'AI-generated' and 'Human-written'. A large, empty text input box is positioned below the tabs. At the bottom of the input box, there are two buttons: 'Clear text' and 'Submit'. Below the input box, a progress bar is shown with the text '0 %' in the center.

(Fig 2.1.2 - UI when application first opened)

The image shows the 'AI CONTENT DETECTOR' application after text has been entered. The text input box now contains a paragraph of text: 'I enjoyed a leisurely afternoon by heading to the park, where the gentle breeze rustled through the trees, creating a soothing backdrop to the cheerful sounds of children playing and birds singing. The vibrant atmosphere and open spaces provided a refreshing escape from the daily routine.' Below the input box, the 'Clear text' and 'Submit' buttons are still present. Below the buttons, a progress bar is shown with the text '98 %' in the center. Below the progress bar, a message box states: 'The classifier considers the text to be likely AI-generated.'

(Fig 2.1.3 - UI after entering text)

Functionality

After interviewing, researching and developing a plan, we can design a table demonstrating the planned functionality the solution should have. We can also refer back to this after implementation to reflect on our success.

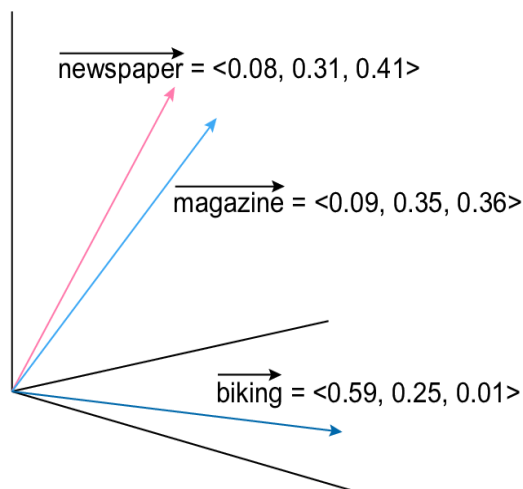
Front-end	Back-end
Must submit text (fed into the AI)	Must return a probability of how likely the overall text is to be AI-generated
Must include an output for the score	Should return where the text is likely to be generated - using indications such as highlighting.
Could include a visual scale for a score between 0 - 100	Could store examples of AI-written and human-written text, which will be loaded into the application
Should include labels for all buttons and titles where appropriate	
Could include clearing text pasted in the text box	

Exploring neural networks

I will use neural networks (NN) compared to fundamental machine learning (ML) algorithms such as SVM or logistic regression to create an accurate prediction model.

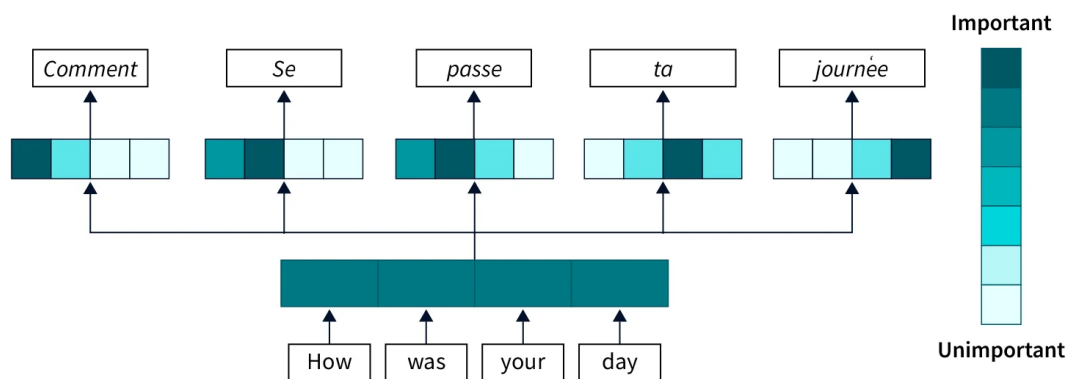
First, I would like to highlight why a neural network better suits this application than other ML algorithms. NNs typically have higher accuracy, especially deep architectures; however, for simple problems, a simple model would also work, but a deep model is required in our scenario due to the complexity behind our problem. A very valuable component of using NNs is feature learning. This is an NN's ability to automatically learn relevant features from raw data, reducing the need for manual feature engineering - which is required in algorithms in SVM and logistic regression. But, this would mean lots of data would be required - typically $>100,000$. This shows how computationally intensive a neural network is to train. Another benefit is after learning all relevant features, NNs can be used to model highly complex relationships between features and target outputs. In contrast, logistic regression and SVM are limited in their representation of non-linear patterns.

As mentioned, NNs require a large amount of data and computational resources for training. Using pre-trained models found online freely, I can avoid the resources required and time to train a model to possess semantic representation instantly, captured via word embeddings. A large-scale pre-trained language model would also provide significant performance boosts compared to models directly trained on that downstream task (Krishna et al. 2023). Word embeddings are a numerical representation of words in a continuous vector space - see Fig 2.1.4 - and are important in NLP for text analysis. However, unlike BERT, word embeddings can differ between models, such as Word2Vec, which don't account for context in the text. Then, a certain model can be fine-tuned for a specific application, such as the classification of ChatGPT-generated content. Therefore, I will use the pre-trained model, RoBERTa (Robustly Optimised BERT Approach). To better explain why I took this approach, it would be worth explaining how RoBERTa works.



(Fig 2.1.4 - word embeddings in 3D (Kenter, n.d.))

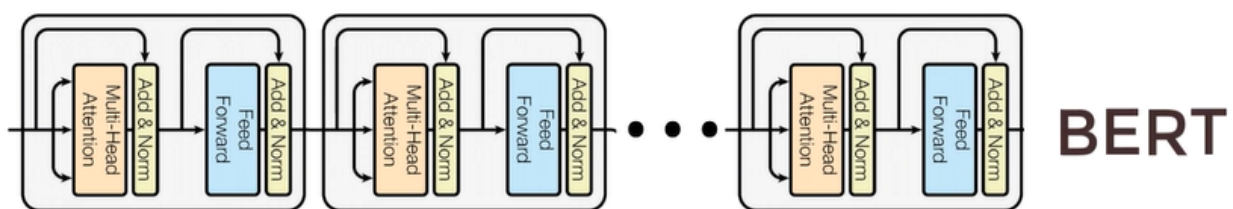
RoBERTa (Liu et al. 2019) is a variant of BERT (Bidirectional Encoder Representations from Transformers), a powerful deep-learning model developed by Google based on transformer architecture. For context, the 'transformer' architecture was first proposed in 2017 in a paper (Vaswani et al. 2017) titled, "Attention is all you need", a new simple neural network architecture based solely on attention mechanisms. Attention mechanisms essentially help preserve the context of every word in a sentence by assigning an attention weight relative to all other words. This way, even if the sentence is large, the model can preserve the contextual importance of each word. Therefore, this allows the model to extract increasingly complex features from the data as it passes through the network, whilst the model can still identify the most relevant information while data passes through layers. Overall, this leads to improved prediction accuracy, and the model becomes more efficient by only processing the most important data.



(Fig 2.1.5 - visualisation of given 'attention' to each word (Scaler Topics, n.d.))

To summarise why using a model based on the transformer architecture is better: a) faster to train, b) context is better learned.

Although BERT and RoBERTa are based on the transformer architecture, these pre-trained language models differ from the exact architecture proposed in the research paper. They are comprised of encoders. We can imagine BERT as a stack of encoders.



(Fig 2.1.6 - a visual representation of encoders stacked to create BERT (Code Emporium 2020))

The transformer encoder comprises multiple self-attention and feed-forward layers, allowing the model to effectively process and understand the input sequence (Scaler Topics 2023). Hence, it is better suited to understand the language, grammar and context than a decoder network.

Pre-training

The goal of pre-training is to teach BERT what is language and context. This is done by simultaneously training BERT on two unsupervised tasks: Masked Language Modelling (MLM) and Next Sentence Prediction (NSP). Overall, these help BERT grasp a deep understanding of language.

MLM → BERT takes in a sentence with random words filled with masks. For example, “The [MASK1] brown fox [MASK2] over the lazy dog.”

The goal is to output appropriate mask tokens such as the following:

[MASK1] = quick

[MASK2] = jumped

This helps BERT understand the bidirectional context in a sentence.

NSP→ BERT takes in two sentences, determining if the second sentence follows the first sentence, which resembles a simple binary classification problem. This helps BERT understand the context between different sentences.

RoBERTa vs BERT

As mentioned before, I will be using RoBERTa instead of BERT, and after briefly explaining how BERT works, we can now investigate the central differences between both these models.

BERT	RoBERTa
Static masking/substitution	Dynamic masking/substitution
Inputs are two concatenated document segments	Inputs are sentence sequences that may span document boundaries
Next Sentence Prediction (NSP)	No NSP
Training batches of 256 examples	Training batches of 2,000 examples
Word-piece tokenization	Character-level byte-pair encoding
Pretraining on BooksCorpus and English Wikipedia	Pretraining on BooksCorpus, CC-News, OpenWebText, and Stories
Train for 1M steps	Train for up to 500K steps
Train on short sequences first	Train only on full-length sequences

(Fig 2.1.7 - central differences between BERT and RoBERTa (Stanford Online 2022))

Dynamic masking

To avoid using the single static mask, training data is duplicated and masked 10 times, each time with a different mask strategy over 40 epochs, thus having 4 epochs with the same mask (Jain 2023).

Next Sentence Prediction

Removing NSP loss matches or slightly improves downstream task (Baeldung 2023) performance, and a downstream task depends on the output of a previous task or process. It involves applying the pre-trained model's knowledge to a new problem. The output of the previous task serves as input to the downstream task, and the model can only perform the downstream task once completing the previous one.

Training batches

Originally, BERT was trained for 1M steps with a batch size of 256 sequences, whereas RoBERTa was trained on more batches of larger sequence lengths. Although the number of steps is shorter (500K), overall, the training is substantially larger due to the large batch sizes. This has two advantages: the large batches improve perplexity on masked language modelling objectives and end-task accuracy. Large batches are also easier to parallelize via distributed parallel training.

Character-level byte-pair encoding

Character-level byte-pair encoding leads to many more word pieces and initialises the vocabulary with individual characters, and gradually merges frequently occurring byte pairs into new tokens. This leads to: a) Effective Subword Tokenization ⇒ enabling models to handle rare or unknown words more effectively, b) Data Compression ⇒ reduces the memory footprint and speeds up computation during training and inference, c) Improved Generalization.

Datasets

RoBERTa uses 160GB of text compared to BERT, which uses only 16GB of text data. The datasets more specifically:

- BOOK CORPUS and English Wikipedia dataset - 16GB
- CC-NEWS - 76GB of text after filtering
- OPENWEBTEXT - 38GB of text
- STORIES - 31GB of text

From these differences and evidence of benchmarks in the official RoBERTa paper, RoBERTa's performance is greater than BERT.

2.2 Implementation

Fine-tuning

To use RoBERTa, I will attempt to fine-tune it. This method is re-training a pre-trained language model to solve a specific task, taking advantage of the model's already established language understanding. Hence, by performing supervised training using a labelled dataset, the output parameters are learnt from scratch and within the model, the parameters are only slightly tuned. Hence, training time for a model to solve a task is improved.

I decided to use fine-tuning as detailed above (and use an additional hidden layer and classification layer); this process is called transfer learning. This is defined as learning a new task that relies on previously learned tasks. In our case, this would RoBERTa's initial understanding of the English language (and context).

Dataset

Data is the first significant part of the ML pipeline; the model's performance heavily relies on the quality of the data. I will use a public HuggingFace dataset found freely on HuggingFace, GPT-wiki-intro (Bhat 2023). Overview: *"Dataset for training models to classify human written vs GPT/ChatGPT generated text. This dataset contains Wikipedia introductions and GPT (Curie) generated introductions for 150k topics"*. The dataset contains a generated response and a short Wikipedia intro for 150k topics, meaning we have 300,000 examples.

Considering half the dataset comprises text from the official Wikipedia page, I must ensure everything is legal. After researching, I found that to reuse text under the Creative Commons Attribution-ShareAlike, credits need to be provided for the authors, which hyperlinks can do. Within the dataset used, for each row, each corresponding row has a link pointing to the source, which includes the author. Therefore, Wikipedia's text can be used under the terms of the Creative Commons Attribution Share-Alike license (CC-BY-SA).

It is also important to note how we will split our dataset: training and testing. A training dataset would contain the data for the NN to explore and learn from, consequently updating its parameters to decrease its overall training loss. Whereas a testing dataset would contain examples (labelled) to evaluate its performance after training has finished. Furthermore, the model is never trained on the same data. Otherwise, the evaluation would not be meaningful for how the model reacts to unseen data. The project utilises an 80/20 split to ensure there is significant data for training.

Because the model was developing on Google Colab, I mounted my (Google) drive and then, using the library pandas, created a DataFrame. A DataFrame is a data structure that organizes data into a 2-dimensional table of rows and columns, much like a spreadsheet.

A dataset class's goal is to return the appropriate tensors for each data item essentially. However, a problem I encountered was getting the error: ' **Type error: new(): invalid data type 'str'** '. I found that `tokenizer.encode_plus()` received 'item' as a parameter for the text, whereas it was meant to be 'text' since the item was the entire item dictionary in the dataframe which included text and labels. Also, `label = torch.FloatTensor(item['label'])` was changed to `label = torch.FloatTensor([item['label']])` for the label to be passed as a single-element tensor instead of returning just a string.

Pre-processing

Pre-processing is refining the raw data to prepare it for input. In our pre-processing, we carry out the following procedures:

- Cleaning the dataset into examples and labels
- Tokenization

I created a dataset class to initialise the training and testing datasets. By separating the generated examples (and labelling each example with a 0) and the Wikipedia intro (and labelling each example with a 1), we can concatenate only these relevant examples into a single dataframe, our dataset.

```
if stage == 'train':
    #dataframe for generated data, additional column label added
    generated = pd.DataFrame({'text': data.iloc[0:int(len(data)*0.8)]['generated_intro'], 'label': 0})
    #dataframe for human-written data, additional column label added
    wiki = pd.DataFrame({'text': data.iloc[0:int(len(data)*0.8)]['wiki_intro'], 'label': 1})
    #concatenate both dataframes
    self.data = pd.concat([generated, wiki])
```

(Fig 2.2.1 - example of implementing test dataset)

When retrieving an item from the dataset, we need the value in a helpful format/type for the model. Hence, I first isolated the message, found by an index, and then converted the corresponding label into a tensor. The message is also tokenized and converted into a pyTorch tensor. Since the tokenizer we use is the pre-trained tokenizer from RoBERTa, we can also use the method `encode_plus` to tokenize the text and return the attention mask. The dataset returns a dictionary consisting of input ids, attention masks and labels.

- Input ids: the numerical representations of tokens building the sequences that will be used as input by the model
- Attention masks: indicate to the model which tokens should be attended to and which should not before passing into the model.
- Label: 0 for humans and 1 for AI.

We also use padding and truncation to ensure all messages are the same size before training the model on the data. Hence, since those tokens shouldn't be attended to, padding tokens are represented as '1's in the input ids and '0's in the attention mask.

Data module

The data cleaning/preparation is usually scattered across many files. Using a data module (creating a data module class), we can instantiate the datasets within the class and create corresponding data loaders, specifically a train data loader and test data loader. A DataLoader object's role is to fetch data from a dataset and serve the data up in batches. For example, the 'train_dataloader' fetches the data from the test dataset and divides all the samples into batches we set. Within the DataLoader class, we can then set the num_workers, which denotes the number of processes that generate batches in parallel. Each worker process will load a batch of data and send it to the main process, then train the model on the data. Typically, we assign the num_workers aligning with the number of CPU cores on that machine. This was an important choice as setting the value of num_workers significantly improves the training performance since loading data in parallel can reduce the time to load all the data, speeding up the training process.

A common error diagnostic received in the terminal when running the cell was:

"UserWarning: This DataLoader will create 4 worker processes in total. Our suggested max number of worker in current system is 2, which is smaller than what this DataLoader is going to create. Please be aware that excessive worker creation might get DataLoader running slow or even freeze, lower the worker number to avoid potential slowness/freeze if necessary.
warnings.warn(_create_warning_msg("

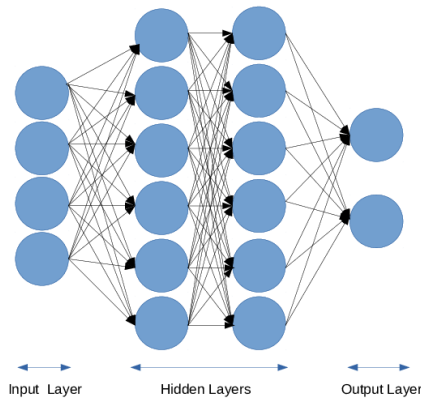
Now, this isn't a typical 'error' which would usually cause the program to crash. Instead, this was a warning indicating the number of workers in the dataloader was too large. Num_workers refers to the number of processes that generate batches in parallel. Initially, I set the number of workers to 4; however, at the time, I wasn't using a GPU, which explains why the program sends a warning because the num_workers is directly proportional to the number of cores your system has. I found that the best thing to do is to use trial and error (regardless of the context). For example, too many num_workers may be unnecessary and cause too much computation to be used where the same level of performance may be reached with fewer num_workers. Therefore, you can find a suitable value by slowly increasing the number of num_workers.

Building the classifier

The classifier is the model that the data will be passed into and learns to distinguish between negative and positive classes. I'll also use Pytorch's lightning module to reduce boilerplate code, making the code more readable. The model's class is structured into 4 distinct sections: the constructor, which initialises the base model. The forward method passes data through the model and training/testing/prediction steps for receiving output from the model and configuring the optimisers. We will first explore the constructor.

The model is structured as the pre-trained model (RoBERTa) plus other layers we choose to initialise. In our scenario, on top of RoBERTa's layers, we have a hidden layer and then a final

classification layer. The hidden layers act as an intermediate layer between the final classification layer and the input and introduce non-linearity (such as activation functions) into the model, allowing the neural network to learn the suitable 'patterns' for detecting AI-generated content. This is good because it allows the model to be more flexible and fit more diverse and complex data. The final layer (classification layer) contains two nodes and is the output layer. Since we are testing for positive and negative labels, two nodes will output the final probabilities for the text to be AI-generated or human-generated.



(Fig 2.2.2 - example of a Neural Network with two hidden layers and 2 output nodes (Chatterjee 2019))

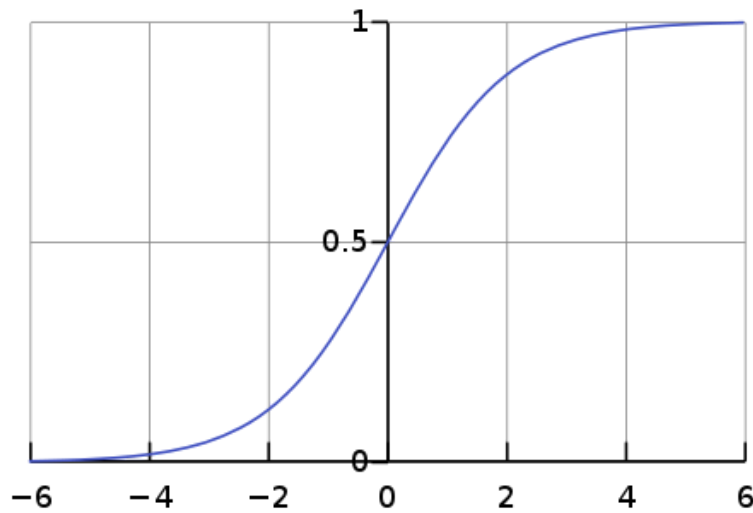
Within the model, 'BCEWithLogitsLoss' is used as the loss function. The PyTorch documentation (PyTorch, n.d.) defines this as combining a sigmoid layer and the BCELoss in one class. This is beneficial for us as this version is more numerically stable than using a plain sigmoid followed by a BCELoss since by combining the operations into one layer, we take advantage of the log-sum-exp trick for numerical stability. To clarify, BCELoss creates a criterion that measures the Binary Cross Entropy (BCE) between the targets and the input probabilities.

I will sequentially explain the sigmoid function, BCE and the combined effect/loss equation to understand better how these functions are used and why they are helpful.

The sigmoid function is particularly useful since it exists between 0 and 1, which is very helpful for models that have to predict the probability of something as an output.

$$\text{Sigmoid}(x) = \sigma(x) = \frac{1}{1 + \exp(-x)}$$

(Fig 2.2.3 - sigmoid function, exp represents e (base of the natural log) (PyTorch, n.d.))



(Fig 2.2.4 - the logistic function, an example of a sigmoid function (Wikipedia 2023))

The figures above demonstrate an example of a sigmoid function called the logistic function, which is useful for predicting probabilities as its codomain is (0, 1). However, other examples include the hyperbolic tangent function, which maps values to a range (-1, 1). In our context, this may not be helpful as a prediction can't be negative, but using a sigmoid function such as tanh is proven to be more useful in problems involving two-player board games. For example, -1 represents the enemy, and 1 represents you as the player.

$$f(x) = \tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

(Fig 2.2.5 - tanh function (Hyperbolic tangent) (Wikipedia 2023))

BCE is “the negative average of the log of corrected predicted probabilities” (Saxena 2021). Essentially, BCE compares the predicted probabilities to actual class output, which can be 0 or 1. It then calculates the score that penalises the probabilities based on the distance from the expected value. The definition defined at the beginning can be explained using a formula for the log loss:

$$\text{Log loss} = \frac{1}{N} \sum_{i=1}^N - (y_i * \log(p_i) + (1-y_i) * \log(1-p_i))$$

(Fig 2.2.6 - log loss function for BCE (Saxena 2021))

- y_i = actual class of data sample
- p_i = probability of positive class

Since we are only using two labels (1 and 0) for the positive and negative classes we can manipulate the loss function to produce a negative average of the log of *corrected predicted probabilities*. I mean that if the samples class were 0 ($y_i = 0$), for example, this would cause the ' $y_i \cdot \log(\pi_i)$ ' part of the formula to become 0 since anything multiplied by 0 is 0. This means we now only measure the log of the corrected predicted probability since if we imagine the probability is a measure for class 1, then to find the corrected predicted probability, we would have to find one minus that probability hence $\log(1 - \pi_i)$. Also, a negative is used in front of the calculation to cancel the negative output as a log of any number below one is negative, and we want to have an output of a loss score between 1 and 0. To summarise, BCE is a model metric that tracks incorrect labelling of the data class by a model, penalizing the model if probability errors occur in classifying the labels.

So, 'BCEWithLogitsLoss', as defined earlier, combines these methods into a single loss function.

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^\top, \quad l_n = -w_n [y_n \cdot \log \sigma(x_n) + (1 - y_n) \cdot \log(1 - \sigma(x_n))]$$

(Fig 2.2.7 - BCEWithLogitsLoss formula (PyTorch, n.d.))

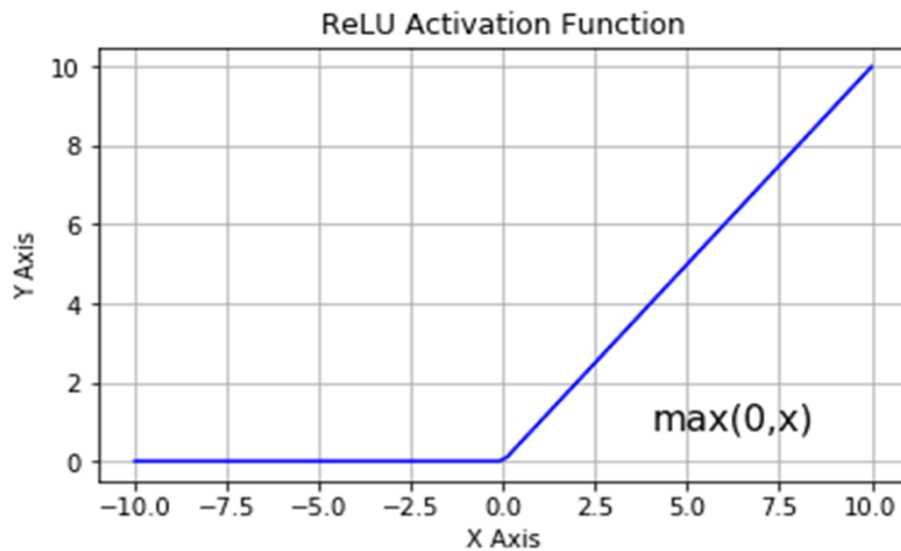
The loss function is very similar to BCE, and the only difference is passing x_n (our probability - π_i) through the sigmoid function before carrying out the log operation, shown by ' $\sigma(x)$ '.

Finally, this will produce a score at every iteration where a small 'loss' value/score signifies the model fits the data accurately (good), and a large loss value indicates that the model is having trouble distinguishing between the different data samples.

Also, within the constructor is the initialisation of the dropout layer. During training, the dropout layer randomly zeroes some of the elements of the input tensor with probability p using samples from a Bernoulli distribution. Each channel will be zeroed out independently on every forward call. The Bernoulli distribution is a discrete probability distribution where the Bernoulli random variable can have only 0 or 1 as the outcome, effectively deciding which units in the layers to deactivate. This has proven to be an effective technique for regularization and preventing the co-adaptation of neurons, as described in the paper '*Improving neural networks by preventing co-adaptation of feature detectors*' (Hinton et al. 2012). Regularization refers to minimizing the adjusted loss function whilst preventing overfitting or underfitting, allowing for better generalization and improved model performance when analysing unseen data.

Now, looking at the forward function, the forward pass of the neural network involves processing the input text, making predictions based on the text and calculating the loss during training. The forward function first takes in three parameters: `input_ids`, `attention_mask` and `labels` (if the data has labels), which are all input text components. The input is then processed through the RoBERTa model, and then using a pooling layer, we create a fixed-size representation of the entire output sentence. Essentially, a mean pooling operation is applied along the first

dimension, which computes the mean value for each feature across all tokens. The output from this process is then stored in a variable called 'pooled_output'. The output from this process is then passed through the hidden layer, which performs linear transformations to a dropout layer



to a Rectified Linear Unit (ReLU) activation function.

(Fig 2.2.8 - ReLU diagram (Nomidl 2022))

The activation function essentially causes any input below 0 to be discarded and returns 0. Whereas if the input is above 0, then the value is unaffected. In this process, ReLU introduces non-linearity into the model, which allows the model to model complex relationships in the data.

After this, the output is passed into the classifier layer, which returns us logits, referring to the model's raw predictions before applying a final activation function such as the sigmoid function.

If data passed into the neural network contains labels, we can calculate loss during the training/testing/validation phase; hence by using a conditional check, we check if the input text has labels. If they are present, we calculate the loss using BCEWithLogitsLoss (as defined earlier within the model's class definition) and finally return the loss and logits.

Within the class, we also define the functions for 'training_step', 'testing_step' and 'predict_step', which all take in the batch and are passed in by the data loader. Each function is necessary for later procedures, such as using the PyTorch lightning's trainer to begin training the model. For example, when training the model, it will look for and run the function called 'testing_step'. Also, for a function such as 'predict_step', we only need to return logits compared to the other two, which return loss and logits, as we only would want the predictions from an unseen sample (and there would be no labels attached alongside the test to evaluate the prediction).

Finally, we also configure the optimisers. This surrounds creating the particular optimiser, gathering the total steps required, the scheduler, and using other tools such as warm-up steps.

Hyper-parameters

Hyper-parameters involve the conditions we select (as the programmer) and are not learned throughout the training phase. The hyper-parameters which were involved and tuned when training the model were:

- Batch sizes
- Learning rate
- Epochs
- Weight decay

The learning rate is a hyperparameter that controls how much to change the model in response to the estimated error each time the model weights are updated. Choosing the learning rate can lead to a very long training process if set too small, whereas a value too large will likely lead to a sub-optimal set of weights due to the fast/unstable training process.

Weight decay is an example of a regularization technique by adding a small penalty to the loss function, which in turn reduces the complexity of the model and can also prevent overfitting.

Epochs refer to passing through an entire dataset, forwards and backwards, through the neural network only once. However, as an epoch of the entire dataset would be too large to feed into the computer at once, we divide it into several smaller batches (hence configuring the batch sizes). To clarify what a batch size means, it is the total number of training examples present in a single batch, so the division of the dataset into different samples.

Selecting more than one epoch prevents problems such as underfitting from arising; however, as the number of epochs increases, the weights continue to increase and can lead to overfitting. You can see how, like with many of these hyper-parameters, changing them from too low to too high can lead to their problems. The challenge is to find the most 'good' enough combination which leads to the most efficient and accurate model.

Training the model

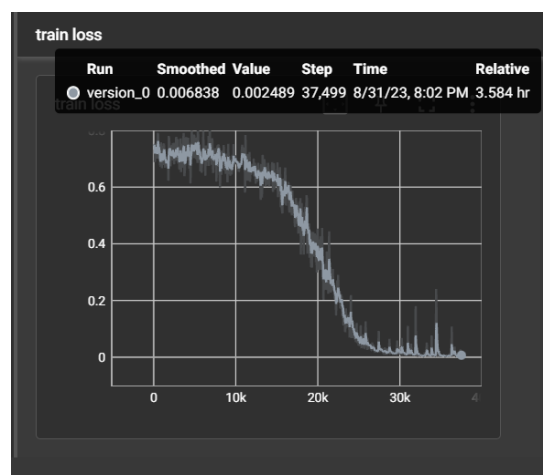
Using the pytorch lightning trainer, we initialise the trainer by calling `pl.Trainer` and passing in the number of epochs to use. We can then proceed with 'fitting' the model to the data after

passing in the model and the training data module.

```
INFO:pytorch_lightning.utilities.rank_zero:GPU available: True (cuda), used: True
INFO:pytorch_lightning.utilities.rank_zero:TPU available: False, using: 0 TPU cores
INFO:pytorch_lightning.utilities.rank_zero:TPU available: False, using: 0 IPUs
INFO:pytorch_lightning.utilities.rank_zero:HPU available: False, using: 0 HPUs
INFO:pytorch_lightning.utilities.rank_zero:You are using a CUDA device ('NVIDIA A100-SXM4-40GB') that has Tensor Cores. To properly utilize the
WARNING:pytorch_lightning.loggers.tensorboard:Missing logger folder: /content/lightning_logs
INFO:pytorch_lightning.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
INFO:pytorch_lightning.callbacks.model_summary:
| Name | Type | Params |
|-----|-----|-----|
0 | pretrained_model | RobertaModel | 82.1 M
1 | hidden | Linear | 590 K
2 | classifier | Linear | 769
3 | loss_func | BCEWithLogitsLoss | 0
4 | dropout | Dropout | 0
|-----|-----|-----|
82.7 M | Trainable params
0 | Non-trainable params
82.7 M | Total params
330.839 | Total estimated model params size (MB)
Epoch 4: 100%  7500/7500
INFO:pytorch_lightning.utilities.rank_zero:Trainer.fit` stopped: `max_epochs=5` reached.
```

(Fig 2.2.9 - output from running trainer (Authors own))

We receive this output from running the required cell to running the trainer. This concerns all the layers used and the total number of parameters trained. From Fig 2.2.9, the significant number of parameters within RoBERTa compared to the additional layers, since RoBERTa is a large language pre-trained model, is apparent from the total of 82.1M parameters. With all this combined, the model is 330 MB large.



(Fig 2.2.10 - graph displayed using the tensor board showing training loss(Authors own))

Testing the model

Before using evaluation metrics, we can quickly test the model by using the Trainer initialised before to receive a testing loss.



(Fig 2.2.11 - tensor board displaying the loss value (Authors own))

The loss value received is 0.02421 - extremely positive. This indicates that the model has little to no issue distinguishing between AI-generated and human-written text. It's not helpful to only evaluate the model on its training loss therefore, I will also need to use other evaluation metrics, such as confusion matrices and an AUC ROC curve, to indicate better the model's accuracy, precision and the quality of the model's predictions.

One of the largest issues I encountered was during testing, which turned out to be an issue using the `lightning.trainer.test` method after training the model. I originally defined the testing function within my class definition as 'testing_step'; however, upon researching and countless error messages, I found that `trainer.test` looks for a method named 'test_step', which will execute the according function.

```
predictions = classify_raw_comments(loader_model, testing_chatgpt_data_loader)
```

Test metric	DataLoader 0
test loss	0.7055180103403508

```
TypeError                                 Traceback (most recent call last)
<ipython-input-77-d76f2f2dd18a> in <cell line: 1>()
----> 1 predictions = classify_raw_comments(loader_model, testing_chatgpt_data_loader)

1 frames
<ipython-input-12-73e6e35a4a20> in <listcomp>(.0)
     7 #It's then applying the sigmoid function using torch.sigmoid() to the tensor.
     8 #All these modified tensors are then stacked together using np.stack() to create a new numpy array called flattened_predictions
----> 9 flattened_predictions = np.stack([torch.sigmoid(torch.Tensor(p)) for batch in predictions for p in batch])
    10 return flattened_predictions

TypeError: new(): invalid data type 'str'
```

(Fig 2.2.12 - error message after running `classify_raw_comments` function (Authors own))

Initially, I created a 'classify_raw_comments' function which would call the trainer.test function and store all the following results in a variable named 'predictions'. As all the predictions would be in tensors, separated for each batch, I would have to 'flatten' all the predictions to be accessible from one array. However, this is where I started getting the error 'TypeError: new(): invalid data type 'str''. As a starting point, I used the model to predict one comment by creating a separate function called classify_single_comment.

```
[42] def classify_single_comment(model, data_item):
    input_ids = data_item['input_ids']
    am = data_item['attention_mask']
    label = data_item['labels']
    with torch.no_grad():
        _, logits = model(input_ids.unsqueeze(dim=0), am.unsqueeze(dim=0), label.unsqueeze(dim=0)) # Assuming your model expects a dictionary of inputs.

    # Apply the sigmoid function to the logits to get probabilities.
    probabilities = torch.sigmoid(logits)

    # Convert probabilities to a numpy array.
    flattened_predictions = probabilities.numpy()

    return flattened_predictions

[40] data_item = testing_ChateGPT_ds._getitem__(0)

[43] prediction = classify_single_comment(loaded_model, data_item)

print(prediction)

[[0.00365336]]
```

(Fig 2.2.12 - error message after running classify_raw_comments function (Authors own))

In the specified function, I extracted a data item directly from the dataset previously defined and under the torch.no_grad() context-manager, which disables all gradient calculation - since we don't want to change the weights of the classifier - use the model to return the logits. After passing the probability and 'flattening' the single prediction, I returned the flattened prediction. Therefore, this function returned a suitable prediction of approximately 0.0037 (2 s.f.), which was a positive sign since the concerning data example was labelled 0. Therefore, I knew that the issue stemmed from the lightning.trainer.test method and not the actual model or the flattened_predictions stage in 'classify_raw_comments'. After researching lightning.trainer on Stackoverflow, I found that the method looks for a function called 'test_step'. Alternatively, in my model's class definition, I labelled this function as 'testing_step'.

Instead of the long and likely more straightforward way of renaming the function and re-training the model again, I decided to, immediately after training, save the model to my local desktop using a library 'joblib', which meant I could change the model later.

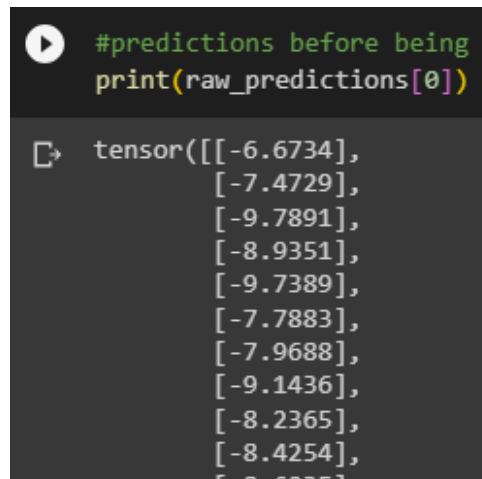
```
import types

def test_step(self, batch, batch_index):
    loss, logits = self(**batch)
    self.log("test loss", loss, prog_bar = True, logger = True)
    return loss, logits, batch['labels']

loaded_model.test_step = types.MethodType(test_step, loaded_model)
```

(Fig 2.2.13 - writing the function 'test_step' into the pre-existing model (Author's own))

I could use types after defining the new (correct) function, `test_step.MethodType` to write the new function into the pre-existing model without having to retrain the model.

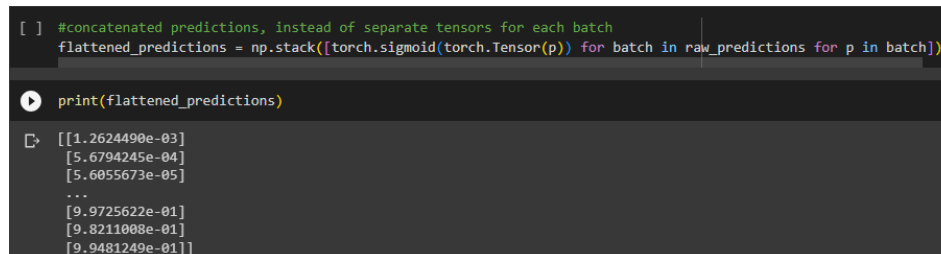


```
#predictions before being
print(raw_predictions[0])

tensor([[ -6.6734],
        [ -7.4729],
        [ -9.7891],
        [ -8.9351],
        [ -9.7389],
        [ -7.7883],
        [ -7.9688],
        [ -9.1436],
        [ -8.2365],
        [ -8.4254],
        [ -8.6035]])
```

(Fig 2.2.14 - checking raw_prediction (Authors Own))

Also, just for extra precaution, I removed the function to ensure `flattened_predictions` wouldn't cause the program to crash to ensure the `raw_predictions` are saved. This is because if `raw_predictions` were correctly loaded into the `predictions` variable, for example, but in the same function, the next instruction would not work, and all the values stored in `predictions` would be lost, so I would have to re-test the model. After obtaining all predictions for each batch, I can create a new variable called `flattened_predictions` to store all the values for each prediction in one array, making it easier to access later when comparing predictions with actual labels when using the evaluation metrics.



```
[ ] #concatenated predictions, instead of separate tensors for each batch
flattened_predictions = np.stack([torch.sigmoid(torch.Tensor(p)) for batch in raw_predictions for p in batch])

print(flattened_predictions)

[[1.2624490e-03]
 [5.6794245e-04]
 [5.6055673e-05]
 ...
 [9.9725622e-01]
 [9.8211008e-01]
 [9.9481249e-01]]
```

(Fig 2.2.15 - flattened_predictions debugged (Authors Own))

For better modularisation, this could have been written in a single function. However, for the purpose of debugging, it was easier to check each stage of the method to see if everything was working correctly.

GUI

As planned earlier, this project should also allow any user to input text into a textbox where output is printed, indicating the probability of the text being AI-written. Since this application shouldn't be front-end heavy, I decided to use Tkinter, a built-in library in Python.

I encountered a problem aligning all the elements, such as clear and submit buttons. I overcame this by using the method `grid`, which separates the screen into a grid with rows and columns - like a matrix.

```
clearBtn.grid(row = 2, column = 0, padx = 100, sticky = E)
submitBtn.grid(row = 2, column = 0, padx = 10, sticky = E)
```

By setting the parameter, 'sticky' to E (symbolising east), the certain elements would automatically be shifted to the edge (east) of the screen. Then, using padding, I could manipulate the offset between the buttons and the edge of the window.

3 - Finalising the project

3.1 Evaluation

This section details evaluating our model created and the possible impact this can make on the wider world based on the results that arise.

Confusion matrix

A confusion matrix represents the prediction summary in matrix form, where we can also gather metrics such as accuracy and precision. In Fig 3.1.1, TP represents true positive, FP represents false positive, FN represents false negative, and TN represents true negative.

		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

(Fig 3.1.1 - diagram of the confusion matrix (Ragan 2018))

From the values obtained in the matrix, we can then use the following equations:

$$\frac{TP + TN}{TP + TN + FP + FN}$$

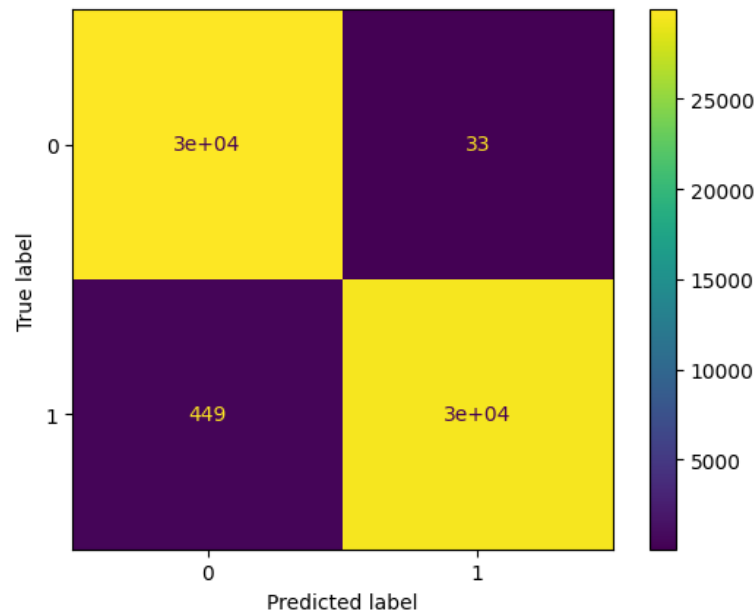
(Fig 3.1.2 - accuracy equation (Authors own))

$$\frac{TP}{TP + FP}$$

(Fig 3.1.3 - precision equation (Authors own))

Accuracy allows us to determine the proportion of **correct** predictions the model made, just like covered when researching previous solutions in section 1.3. Precision allows us to specifically look at the proportion of correctly labelled positive examples, which distinguishes the human-written text from the AI-generated text. It is helpful in our context as we want to evaluate

the model's ability to not mistake a human-written text as AI-written, more importantly than flagging AI-generated text as human-written.



(Fig 3.1.4 - confusion matrix from evaluating my model (Authors own))

We can now calculate the accuracy and precision from the results presented in the confusion matrix.

Accuracy ≈ 0.9920

Precision ≈ 0.9989

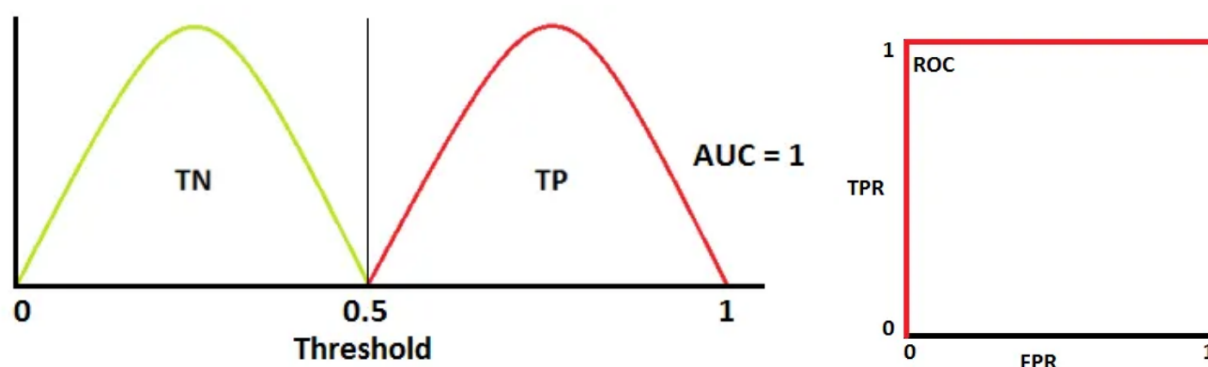
We observe near-perfect results from testing the model on the testing data. However, the example above used a threshold of 0.5 to classify probabilities into binary labels. Using an AUC ROC curve, we can further evaluate the quality of our predictions and select the most optimal threshold.

AUC ROC Curve

The AUC ROC curve represents the area under the curve (AUC) receiver operating characteristics (ROC).

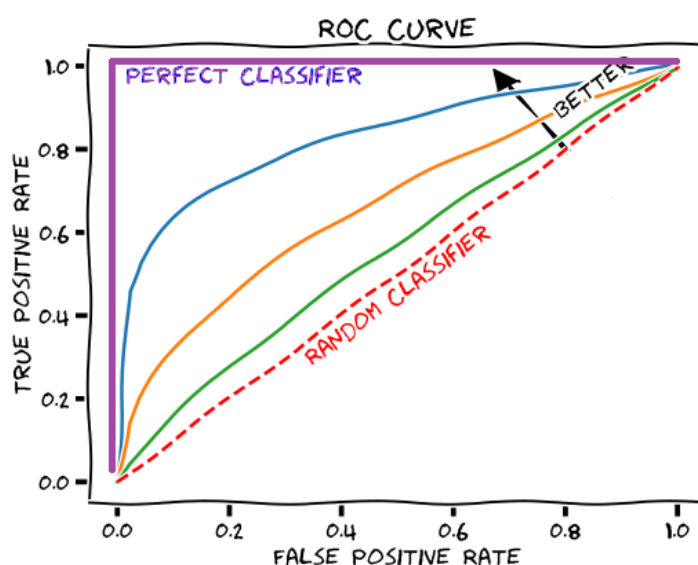
The diagram can be thought to be split into two distinct parts: AUC and ROC. ROC is the probability curve showing the performance of a classification model at all classification thresholds, and AUC represents the degree of separability. Hence, a higher AUC means the model is better at predicting 0 classes as 0 and 1 classes as 1. This can be achieved by representing the y-axis as the true positive rate and the x-axis as the false positive rate.

To better understand how ROC is the curve of probability, we can imagine the distributions of a distribution curve (in red) of the positive class and another distribution curve (in green) of the negative class.



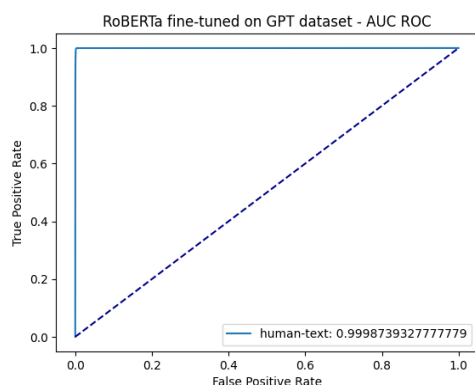
(Fig 3.1.5 - probability distribution leading to perfect ROC curve (Narkhede 2018))

Since the two curves don't overlap, the model has an ideal separability measure. It can perfectly distinguish between positive and negative classes, leading to the following shape for the ROC curve. The following is what multiple classifiers graphed together may look like:



(Fig 3.1.6 - AUC ROC Curve example (Draelos 2019))

Now, for my AUC ROC curve:



(Fig 3.1.7 - AUC ROC Curve for my model (Author's own))

The figure demonstrates that the classifier is near perfect, with an AUC score of 0.99987. This looks great, as we know the classifier works? Due to a general understanding of the problem, this score and shape may

be too good to be true due to the nature of detecting AI-generated text, as in reality, it's quite difficult to predict the text's origin with certainty. However, regarding my procedure in carrying out this project, the above evaluation metrics solidify that this was a success in fine-tuning RoBERTa.

From the prior figure, we can now determine the optimal threshold when classifying text using an algorithm that extracts information from the AUC ROC curve.

```
#find optimal threshold
optimal_idx = np.argmax(tpr - fpr)
optimal_threshold = thresholds[optimal_idx]
print(f"Threshold value is: {optimal_threshold}")
```

```
Threshold value is: 0.021813813596963882
```

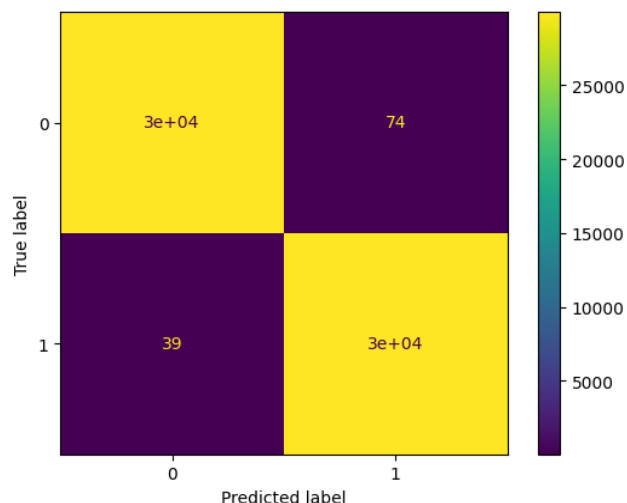
Essentially, this tells us if the probability of a prediction for the label positive is above 0.02 then we can confidently classify the text as human-written. The Youden's J statistic achieves this.

The statistic is calculated as follows:

- $J = \text{Sensitivity} + \text{Specificity} - 1$
- $J = \text{Sensitivity} + (1 - \text{FalsePositiveRate}) - 1$
- $J = \text{TPR} - \text{FPR}$

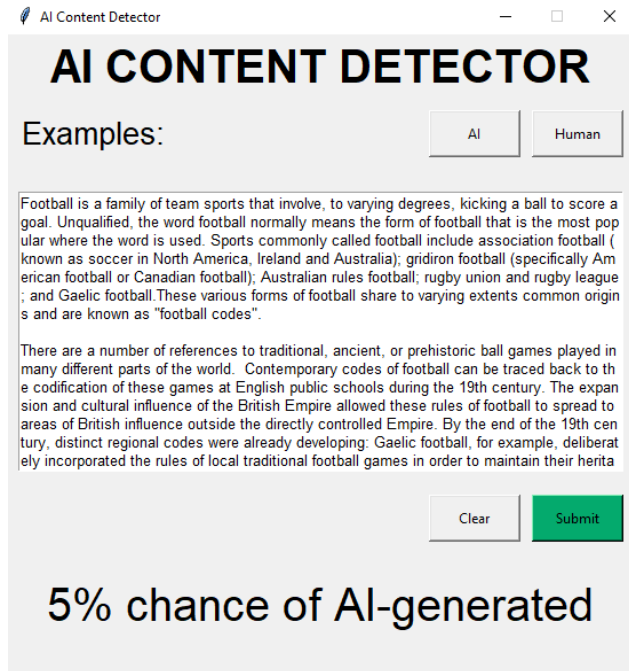
(Sensitivity = TPR, Specificity = $1 - \text{FPR}$
 Sensitivity = $\text{TruePositive} / (\text{TruePositive} + \text{FalseNegative})$
 Specificity = $\text{TrueNegative} / (\text{FalsePositive} + \text{TrueNegative})$)

Even after calculating an optimal threshold, our decision may be affected by the cost of one type of misclassification being more important than another type of misclassification. Also, since the threshold value is so low, the classifier is likelier to classify AI-generated text as human-written text when predicting unseen data. However, utilising the new optimal threshold, we obtain a hopefully more accurate classifier shown by the following updated confusion matrix:

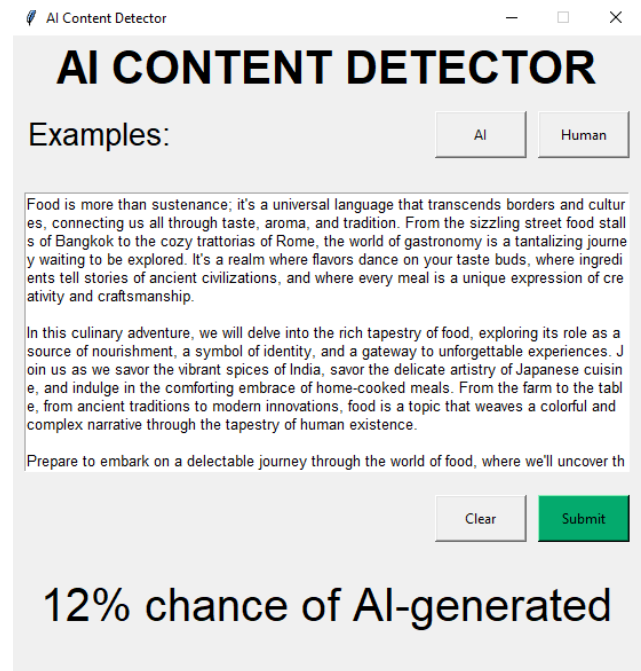


(Fig 3.1.8 - updated confusion matrix
(Authors own))

Post-development problems



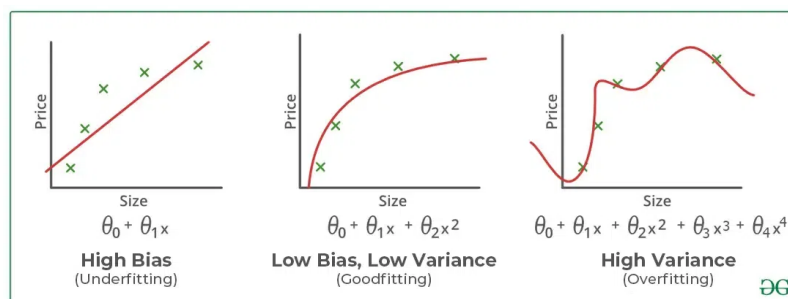
(Fig 3.1.9 - human example (Authors Own))



(Fig 3.1.10 - AI example (Authors Own))

Using the examples from the program, which were obtained via Wikipedia and ChatGPT, I passed them into the model, and the classifier correctly predicted the human text. However, the ChatGPT-generated message only flags a 12% warning. This may stem from a lack of generalisation, resulting from strong overfitting and how the classifier was trained.

Overfitting is an undesirable machine learning behaviour that occurs when the machine learning model gives accurate predictions for training data but not new data. After training the model on a known data set, the model tries to predict outcomes for new data sets. An overfitted model can give inaccurate predictions and perform poorly for all new data types. Although the model was tested on unseen data, it originated from the same training dataset. This means that the model has likely overfitted to data similar to the samples used in the dataset.



(Fig 3.1.11 - examples of increasingly complex models ((Jain 2023))

Overfitting may be caused by three reasons:

- High variance and low bias
- The model is too complex
- The size of the training data

Improvements

Because we used a large training data, we can rule this out already and focus on the two other reasons. In terms of our model being too complex, this may be because of a lack of regularisation techniques which would've enhanced the generalisation capabilities of the model. Referring to variance and bias, in Machine Learning, there's a concept called the bias-variance tradeoff (Karagiannakos 2021), which is also linked to the model's ability to generalise. The bias error is an error from wrong assumptions in the learning algorithm. High bias can cause an algorithm to miss the relevant relations between features and target outputs. For example, if the model has a bias too large, this would cause most data samples to be predicted only one class regardless of other features. The variance is an error from sensitivity to small fluctuations in the training set. A high variance may result in modelling the random noise in the training data. The bias-variance tradeoff describes how we can reduce the variance by increasing the bias. Good regularization techniques strive to minimize the two sources of error simultaneously. Hence achieving better generalization. Therefore, using regularisation techniques, we should've been able to increase the model's overall performance to predict new data.

Regularisation is a set of strategies used in Machine Learning to reduce the generalisation error. I used weight decay, classified as an L2 regularisation technique, in my project. This means that a constant factor on each training step will reduce each weight of the weight vector. The L2 regularizer will have a relatively small effect on the directions contributing to the loss function while eliminating those that stray from the loss function. As a result, we reduce our model's variance, making it easier to generalize on unseen data. Although I used a regularisation technique, this doesn't mean it will automatically 'work' or be the best option when training my model. Other techniques, such as L1 regularisation, aim to entirely discard features from the training process. This achieves the same as L2 regularisation - reducing the average magnitude of all weights - but by introducing sparsity in the weights by forcing more weights to be zero. Hence, there is a magnitude of techniques I could've used, and whilst I used weight decay, the value is decided by me. Therefore, by undergoing hyper-parameter tuning using cross-validation and analysing the performance of different models, which would've been trained on different values of weight decay such as 0.0001, 0.001, 0.01... I would've been able to select the most appropriate value from the results of a validation test.

So, if I could re-do this project or improve it in the future, I would have used strategies such as cross-validation to determine the most suitable values for hyper-parameters and experiment using different regularisation techniques. However, the issue stems from using such a large dataset and training complex models, which require so much computational power and time to train. Therefore, this would still be hard to implement with insufficient resources and time.

3.2 Reflection

I feel that this project has greatly benefited my computer science and technological skillset, from understanding complex concepts like attention and word embeddings to improving my proficiency in using many Python libraries typically used in mainstream data science roles such as pandas. Alongside this, I have improved my time management skills and ability to work independently by researching using academic papers and popular newspapers such as 'medium'. I now understand the basic concepts in building a machine learning model and delving into complex concepts such as RoBERTa, and transformers. This led me to understand encoder networks and attention used to improve the performance of large deep-learning models. This project (besides learning deep-learning concepts) has allowed me to investigate the procedures an individual would take before building any model, regardless of the domain, which will likely encourage me to continue creating many more projects.

Looking at my performance, I can say that I am satisfied with the skills I learnt and practised. I used Tkinter to create GUIs and learned about aligning techniques and widgets. This was the hardest part of creating the user interface because I had to use padding and grid-mapping techniques. I encountered many problems in creating the model, but I was happy with how I executed each section (data preprocessing → data modules → creating the model → evaluation). All my issues were resolved as documented throughout implementation (2.2), but I also had to climb a learning curve of using many libraries to manage each section. Although using Google Colab automatically divides sections into cells, I would have improved my modularisation throughout the project to make it more readable. For example, whilst debugging, I had to deconstruct functions into individual statements, which meant I had to run each line individually where I could have created functions to execute each line under a single function with a meaningful function name.

As stated before, the model's performance could have been improved by using validation datasets to manage the model's performance created for each hyper-parameter measurably. In that way, I could find the most suitable hyper-parameters for better yield - although the accuracy and precision calculated were more than sufficient, it was great to see such high accuracy with scores such as AUC 0.9987. To improve the performance, I would also consider creating a variety of completely separate models developed using different pre-trained language models like BERT or by handcrafting a neural network from scratch using architectures such as LSTM (long-short-term memory). However, due to time constraints and lack of resources, this would cost too much time and money to train each individual model.

Apart from technicalities, I would also try to keep to the schedule better by setting more reasonable goals for each stage, although I managed to finish approximately by the time I scheduled. I could have also taken account of my holiday (going to a different country) in the summer, which would have affected planning and availability to work on this project during the day. Also, looking back at my functionality table, I was able to achieve all required functionalities and missing two other side features which is a relative success.

3.3 Conclusion

Overall, my aim of upholding academic integrity by designing an AI content detector has been fulfilled by creating a model by fine-tuning RoBERTa on Wikipedia datasets surrounding various topics from different backgrounds. My project has also achieved other goals initially set out at the beginning by researching ideas behind deep learning and other methods which could have been used to tackle this complex problem whilst learning how to evaluate and compare the success of models.

Hopefully, this has shown the potential for deep learning in building a solution whilst maximising efficiency using techniques such as transfer learning. However, I have also highlighted that when training large deep-learning models, other considerations must be made concerning time and resources and their negative environmental consequences. For example, a study titled 'Energy and Policy Considerations for Deep Learning in NLP' (Strubell, Ganesh, and McCallum 2019) highlighted why models are costly to train and develop financially and environmentally. This is due to the hardware, electricity, or cloud computing time cost and the carbon footprint required to fuel modern tensor processing hardware. In fact, from this study, they estimated that

Consumption	CO₂e (lbs)
Air travel, 1 passenger, NY↔SF	1984
Human life, avg, 1 year	11,023
American life, avg, 1 year	36,156
Car, avg incl. fuel, 1 lifetime	126,000
Training one model (GPU)	
NLP pipeline (parsing, SRL)	39
w/ tuning & experimentation	78,468
Transformer (big)	192
w/ neural architecture search	626,155

(Fig 3.3.1 - table with estimated CO₂ emissions from 'Energy and Policy Considerations for Deep Learning in NLP' paper)

training a large deep-learning model produces 626,000 pounds of planet-warming carbon dioxide, equivalent to the lifetime emissions of five cars!

Overall, hopefully, others will see the usefulness of employing techniques such as transfer learning for building accurate and powerful models quicker and cheaper, as my project demonstrated this can be done from the comfort of one's home and a laptop. What's more, as hardware resources continue to be developed and become cheaper, people will find it easier to develop their own models for any specific task needed.

References

Baeldung. 2023. "What Are Downstream Tasks?" Baeldung.

<https://www.baeldung.com/cs/downstream-tasks>.

Bhat, Aaditya. 2023. "aadityaubhat/GPT-wiki-intro · Datasets at Hugging Face." Hugging Face.

<https://huggingface.co/datasets/aadityaubhat/GPT-wiki-intro>.

Biswas, Abhishek. 2023. "How to Build an AI Content Detector from Scratch with Python." Level Up Coding.

<https://levelup.gitconnected.com/how-to-build-an-ai-content-detector-from-scratch-with-python-c3cc432a2ee5>.

Chatterjee, Chayan. 2019. "Figure 2. An Artificial Neural Network (ANN) with two hidden layers and..." ResearchGate.

https://www.researchgate.net/figure/An-Artificial-Neural-Network-ANN-with-two-hidden-layers-and-six-nodes-in-each-hidden_fig1_335855384.

Code Emporium. 2020. "BERT Neural Network - EXPLAINED!" YouTube.

https://www.youtube.com/watch?v=xI0HHN5XKDo&ab_channel=CodeEmporium.

Content at Scale. 2021. "AI Content Detector Checks GPT-4, ChatGPT, Bard, & More." Content @ Scale. <https://contentatscale.ai/ai-content-detector/>.

Draelos, Rachel. 2019. "Measuring Performance: AUC (AUROC) – Glass Box." Glass Box.

<https://glassboxmedicine.com/2019/02/23/measuring-performance-auc-auroc/>.

Gillham, Jonathan. n.d. "How Does AI Content Detection Work? – Originality.AI." Originality.AI.

<https://originality.ai/blog/how-does-ai-content-detection-work>.

GPT-3 Demo. 2022. "." - YouTube.

https://www.youtube.com/watch?v=lozIPw-hu9U&t=3s&ab_channel=GPT-3Demo.

- guide, step. 2023. "How Reliable are AI Content Detectors? Which is Most Reliable?" Quetext.
<https://www.quetext.com/blog/how-reliable-are-ai-content-detectors-which-is-most-reliable>.
- Hinton, Geoffrey, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2012. "[1207.0580] Improving neural networks by preventing co-adaptation of feature detectors." arXiv. <https://arxiv.org/abs/1207.0580>.
- Jain, Sandeep. 2023. "Overview of ROBERTa model." GeeksforGeeks.
<https://www.geeksforgeeks.org/overview-of-roberta-model/>.
- Jain, Sandeep. 2023. "ML | Underfitting and Overfitting." GeeksforGeeks.
<https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/>.
- Johnson, Reece. 2023. "Half of College Students Say Using AI Is Cheating." BestColleges.com.
<https://www.bestcolleges.com/research/college-students-ai-tools-survey/>.
- Karagiannakos, Sergios. 2021. "Regularization techniques for training deep neural networks." AI Summer. <https://theaisummer.com/regularization/>.
- Kelley, Jaclyn. 2018. "ROBOT LAWYER: App allows you to sue anyone with press of a button." *FOX 5 DC*, October 18, 2018.
<https://www.fox5dc.com/news/robot-lawyer-app-allows-you-to-sue-anyone-with-press-of-a-button>.
- Kenter, Tom. n.d. "1: Visualization of 3-dimensional word embeddings. | Download Scientific Diagram." ResearchGate.
https://www.researchgate.net/figure/Visualization-of-3-dimensional-word-embeddings_fig1_325451970.
- Krishna, Kundan, Saurabh Garg, Jeffrey P. Bigham, and Zachary C. Lipton. 2023. "Downstream Datasets Make Surprisingly Good Pretraining Corpora." ACL Anthology.
<https://aclanthology.org/2023.acl-long.682.pdf>.

Liu, Yinhan, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike

Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. “[1907.11692] RoBERTa: A Robustly Optimized BERT Pretraining Approach.” arXiv.

<https://arxiv.org/abs/1907.11692>.

Narkhede, Sarang. 2018. “Understanding AUC - ROC Curve | by Sarang Narkhede.” Towards Data Science.

<https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5>.

Nelson, James. 2022. “. ” - YouTube.

<https://www.linkedin.com/pulse/reliability-ai-detection-tools-impact-academic-integrity-james-nelson/>.

Nomidl. 2022. “What is ReLU and Sigmoid activation function?” Nomidl.

<https://www.nomidl.com/deep-learning/what-is-relu-and-sigmoid-activation-function/>.

OpenAI. 2023. “New AI classifier for indicating AI-written text.” OpenAI.

<https://openai.com/blog/new-ai-classifier-for-indicating-ai-written-text/>.

Peters, Jay. 2023. “BuzzFeed is using AI to write SEO-bait travel guides.” The Verge.

<https://www.theverge.com/2023/3/30/23663206/buzzfeed-ai-travel-guides-buzzy>.

PyTorch. n.d. “BCEWithLogitsLoss — PyTorch 2.0 documentation.” PyTorch.

<https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html>.

PyTorch. n.d. “BCEWithLogitsLoss — PyTorch 2.0 documentation.” PyTorch.

<https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html>.

PyTorch. n.d. “Sigmoid — PyTorch 2.0 documentation.” PyTorch.

<https://pytorch.org/docs/stable/generated/torch.nn.Sigmoid.html>.

Ragan, Allison. 2018. “Taking the Confusion Out of Confusion Matrices | by Allison Ragan.”

Towards Data Science.

<https://towardsdatascience.com/taking-the-confusion-out-of-confusion-matrices-c1ce054b3d3e>.

Saxena, Shipra. 2021. "Binary Cross Entropy/Log Loss for Binary Classification." Analytics Vidhya.

<https://www.analyticsvidhya.com/blog/2021/03/binary-cross-entropy-log-loss-for-binary-classification/#h-what-is-binary-classification>.

Scaler Topics. 2023. "What is Encoder in Transformers." Scaler.

<https://www.scaler.com/topics/nlp/transformer-encoder-decoder/>.

Scaler Topics. n.d. "Attention Mechanism in Deep Learning." Scaler.

<https://www.scaler.com/topics/deep-learning/attention-mechanism-deep-learning/>.

Stanford Online. 2022. "." . - YouTube.

https://www.youtube.com/watch?v=EZMOBbu_5b8&ab_channel=StanfordOnline.

Strubell, Emma, Ananya Ganesh, and Andrew McCallum. 2019. "[1906.02243] Energy and Policy Considerations for Deep Learning in NLP." arXiv. <https://arxiv.org/abs/1906.02243>.

Tahir, Bilal. 2023. "Fool GPT by randomly replacing words with synonyms in your text."

GPT-Minus1. <https://www.gptminus1.com/>.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan Gomez,

Lukasz Kaiser, and Illia Polosukhin. 2017. "[1706.03762] Attention Is All You Need."

arXiv. <https://arxiv.org/abs/1706.03762>.

Wikipedia. 2022. "Wikipedia:Reusing Wikipedia content." Wikipedia.

https://en.wikipedia.org/wiki/Wikipedia:Reusing_Wikipedia_content.

Wikipedia. 2023. "Sigmoid function." Wikipedia. https://en.wikipedia.org/wiki/Sigmoid_function.