KING'S College LONDON

University of London

6CCS3PRJ Final Year
Automating OpenAI's GPT-2 Text Generating Language Model

Final Project Report

Author: Danyaal Khan

Supervisor: Prof. Mischa Dohler

Student ID: 1709353

April 21, 2020

Abstract

Language models are getting better and better at generating paragraphs of coherent text through the use of groundbreaking work in machine learning. The ability of these models to produce the next word of a given input can be used to generate samples of any size, allowing them to perform common natural language processing tasks including but not limited to answering questions, completing text, reading comprehension, and summarising text. They can also be used to assist in speech recognition, auto-correction, optical character recognition, and keyboard suggestions.

OpenAI's GPT-2 is a large language model trained on 8 million web pages giving it the potential to generate high quality samples. Generated samples can be made more suitable for a given context by finetuning the model using a relevant dataset. However making use of GPT-2 and other high performing language models can be difficult without the technical knowledge required to interact with the underlying models.

The aim of this project is to design, and implement software solutions that utilise GPT-2 to allow users to generate text, and allow users to generate fictional news articles. This project also explores aspects of interacting with GPT-2, and the effects of an easy to use tool that facilitates usage of language models that produce high quality text.

Originality Avowal

Danyaal Khan

April 21, 2020

Acknowledgements

I would like to thank my supervisor Professor Mischa Dohler for his support throughout the duration of this project. His proposition of the topic of this dissertation has allowed me to fulfil my interest in it by exploring it.

Contents

Chapter 1

Introduction

A language model is a probability distribution that gives the probability of a given sequence of words. This can be used to determine the probability of a certain words appearing after a given sequence of words, from which the most likely word to appear after a provided sequence of words can be determined. Alternatively, the word appearing after a given sequence can be sampled from the distribution, which allows the whole probability distribution to be taken into account as words with a non zero likelihood can be selected.

Predictive text is an accessible example of this that is commonly used in on-screen keyboards in phones, providing suggestions for likely next words. However due to constraints requiring the predictions to be very quick while using minimal computing resources these predictions tend not to look far back in the text, which can lead to repeated predictions of the same sequence of words. However taking these long term dependencies into account in order to consider enough relevant context is important in generating high quality text.

Neural networks can be effective in being used as a language model as they can represent the information associated with words in an abstract way that scales well as they can avoid the curse of dimensionality [1]. The architecture of a neural network can affect the quality of the generated samples. Architectures used as language models include but are not limited to recurrent neural networks (RNNs) which use an internal state (memory) to store information about previous parts of the word sequence, Long short-term memory which is a type of RNN that have sub-components that control what it forgets as the sequence progresses, and Transformers which encode sequences of inputs and take contextual information into account via an attention mechanism before decoding and producing output probabilities.

GPT-2 is large Transformer based model developed by OpenAI that has 1.5 billion parameters (weights and biases) and has been trained on 8 million web pages. Because of the size and diversity of this dataset it outperforms language models trained on specific domains without needed domain-specific training datasets, producing human quality samples [2]. Four versions of GPT-2 are currently publicly available, the largest of which is the original model with 1558 million parameters whilst the other versions have 774 million, 355 million, and 124 million parameters [3]. These are often referred to as 1558M, 774M, 355M, and 124M models respectively.

These models can be fine-tuned on provided datasets in the form of written text. While this is theoretically possible with all four models, efforts to fine-tune the 1558M and 774M models have been met with difficulties due to the memory constraints of modern hardware [4]. While there are efforts to increase the viability of fine-tuning the larger models using software optimisations, finetuning models in this project will focus on the 124M model as it is computationally cheaper to finetune and generate samples from.

1.1 Project Aims

This project's core objective is to design and implement a program that allows for the generation of news articles by leveraging GPT-2 models. This will have functionality that allows the user to interact with it either via a built-in GUI, or through the command line. These options provide the program with both the ease of use of a GUI, and the higher capacity of integrating it into an existing or new software solution that a command line application has.

This tool will be able to take input from the user, allowing for the title to be specified and some optional initial content. From this information, a news article will be generated and outputted either through the GUI, the command line, or written to a file.

A secondary piece of software will also be designed and written to allow for the use of GPT-2 models to generate samples via the Discord messaging app. This will be via commands issued via a Discord message, generating a continuation of the provided prompt and sending it as a Discord message. Discord was selected as it is a platform that has grown in popularity, with "56,000,000 monthly [users]" and "963,000,000 messages sent a day" [5].

This project will also explore some of the factors that can influence the samples (text) generated by GPT-2, such as prior fine-tuning and model size. These factors can contribute to improving sample quality so that it appears to be more human written, and generating novel samples as opposed to repeating information in the dataset used to train/finetune it. Fine-tuning on datasets from a particular domain can allow the models to produce better results, but also risk the introduction of any bias in the dataset potentially appearing in samples produced. Bias towards extreme views could potentially be done purposefully by bad actors leading to "concerns about the potential for misuse" [3], suggesting "GPT-2 relatively quickly integrates the nuances of the ideology it is trained on when responding to a specific prompt". The benefit of fine-tuning for specific domains and the risk of bias will be explored and evaluated manually.

1.2 Own Research Contributions Summary

This section touches upon some of the research contributions this project can provide. These points are explored in detail in section 6.3.

- Language models can be effectively finetuned to generate samples in various formats.
- Despite being made for natural language processing, language models can be used to produce samples that aren't purely natural language.
- The GPT-2 language model can learn clear formatting rules in datasets quickly, even if the datasets used for finetuning are small.
- The GPT-2 language model is able to learn the use of emojis from finetuning on a dataset that includes emojis, despite no emojis appearing in the original dataset it was trained on.
- Language models are capable of learning the writing styles of multiple authors, resulting in a model that can produce samples that include different styles depending on the context. For example a different writing style based on the specified name of the author.
- If an insufficiently sized dataset is used when finetuning GPT-2, the model will eventually overfit the dataset if finetuned for enough steps. This can limit the quality of generated samples. However using a smaller version of GPT-2 can aid in reducing overfitting.

Chapter 2

Background

2.1 Language Models

Unigrams are a simple type of language model that predict the probability of a sequence of words, by independently considering the probability of each word appearing in any sequence of words. They make the assumption that multiplying the probability of each word considered individually approximates the probability of the whole sequence.

$$P\left(w_1, ..., w_n\right) \approx \prod_{i=1}^{n} P\left(w_i\right) \tag{2.1}$$

N-grams are language models that use the fact that the probability of a sequence is equal to multiplying the probability for each word of that word appearing given the sequence of every word before, and the assumption that this can be approximated by considering up to $n-1$ words preceding it. This is an nth order Markov property.

$$P\left(w_1, ..., w_n\right) = \prod_{i=1}^{n} P\left(w_i \mid w_1, ..., w_{i-1}\right) \approx \prod_{i=1}^{n} P\left(w_i \mid w_{i-(n-1)}, ..., w_{i-1}\right) \tag{2.2}$$

Neural network based models (neural language models or continuous space language models) predict the probability of a given sequence by using continuous representations [6]. Types of neural networks used as language models include recurrent neural networks (RNNs), long short term memory (LSTMs), and Transformers.

2.2 Sequence-to-sequence Models

Information from this section is sourced from “Visualizing A Neural Machine Translation Model” by Jay Alammar [7].

Sequence-to-sequence models are a type of deep learning model used in natural language processing tasks [8]. They can be used to take a sequence of words and output another sequence of words, allowing for natural language processing tasks such as summarising text, and translating text [9]. This model is composed of an encoder and decoder.

2.2.1 Context

An encoder processes each word in the input and produces a vector of numbers, known as the context, that encapsulates the information of the original sequence. This context is then passed from the encoder to the decoder, which uses it to generate the output sequence.

To convert a sequence of words into the context an RNN is used. RNNs maintain a hidden state in the form of a vector, and use this alongside an input vector to update the hidden state and generate an output. This allows information about element of an input sequence to be used when generating outputs for following elements of the sequence.

The encoder is an RNN updating its hidden state as it processes each word in the sequence, this hidden state is the context. After the whole sequence has been processed the final hidden state is passed to the decoder. The decoder then uses the context to generate the sequence of words, maintaining its own hidden state. For the decoder each time it runs a computation it generates and outputs a single word, its hidden state is only used internally.

2.2.2 Attention

Attention mechanisms allow the decoder to focus on parts of the sequence by amplifying the signal from relevant parts of the input sequence [10][11]. Attention allows models to produce better results than models that don’t use attention.

For models that use attention instead of just passing the final hidden state to the decoder, the encoder passes the hidden state after each update. Since the hidden state is updated after each word is processed, the decoder has knowledge of what the hidden state looked like through the processing of the sequence.

Every time the decoder generates a word, it weights each of the hidden states it has by a softmaxed score. Meaning states with low scores have less of an impact on the current word and states with a high score have a greater impact. This score is calculated for each hidden state each time the decoder generates a word.

2.3 Transformer Models

Information from this section is sourced from “The Illustrated Transformer” by Jay Alammar [12].

The transformer model is a type of model that is “based solely on attention mechanisms, dispensing with recurrence and convolutions entirely” [13]. These techniques have been shown to produce models “superior in quality while being more parallelizable and requiring significantly less time to train” [13].

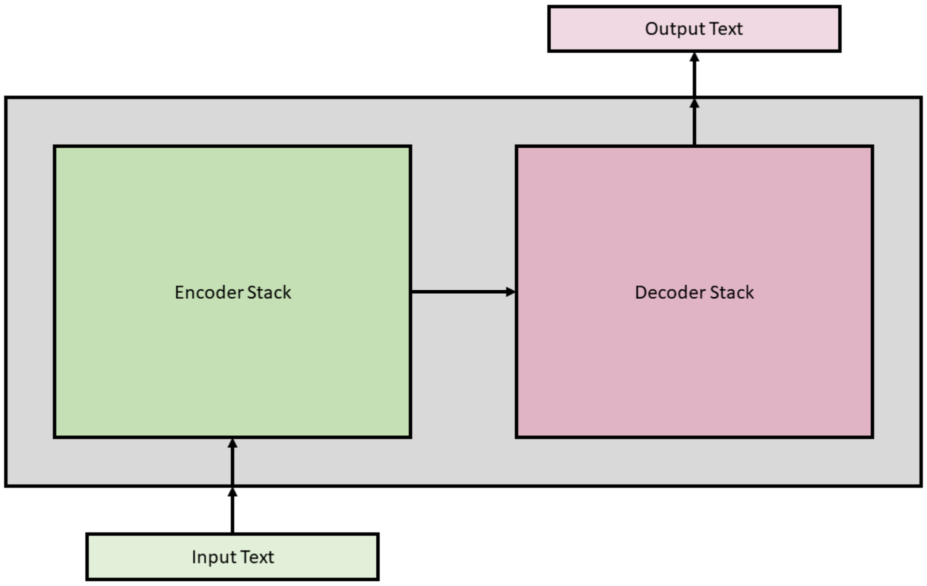


Figure 2.1: A high level visualisation of the architecture of a transformer model

Transformer models are made up of an encoder stack and decoder stack made up of encoders chained together and decoders chained together, where the number of encoders is equal to the number of decoders. The input is passed to the first encoder after being encoded and the output of each encoder is passed to the next encoder until the last encoder is reached. Before the sequence is passed to the encoder it is converted into a sequence of vectors, where each vector represents a word in the sentence.

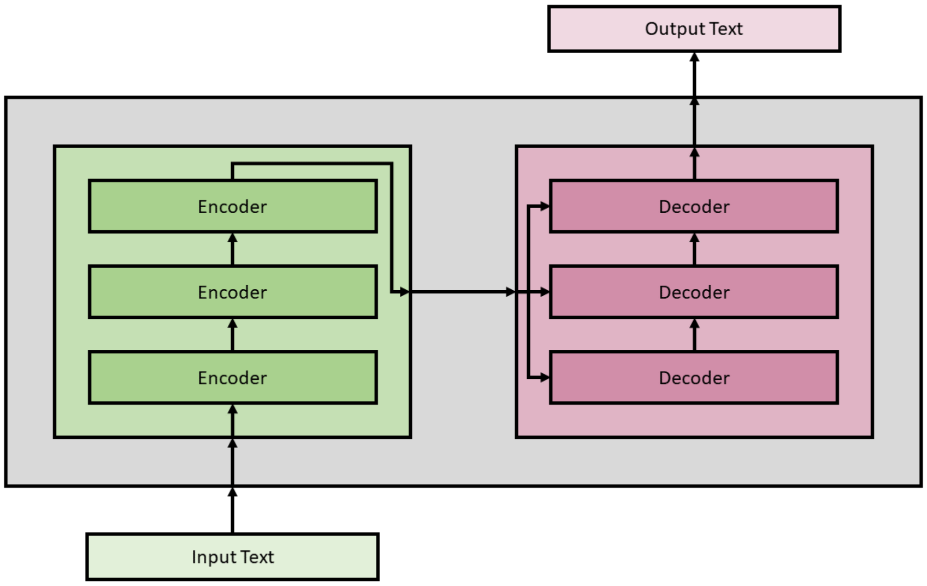


Figure 2.2: A visualisation of the architecture of a transformer model and its components

2.3.1 Encoders

Each encoder contains a self-attention layer and a feed forward neural network. When the encoder is passed an input it is first processed by the self-attention layer in the encoder before being passed to its feed forward neural network, the output of which is the output of the encoder. Each encoder receives a list of vectors as input, each vector being an representation of a word in the sequence. This means the words in the sequence don't have to be processed one after the other or in any specific order, but this doesn't mean the order of words is ignored.

Self-attention

The self-attention layer in each encoder takes this list of vectors as a whole, creating a new vector for each word in the sequence. This vector has a length equal to the number of words in the sequence. For a given word in the sequence, this new vector assigns a score to every word in the sequence (including itself) based on how relevant that word is when considering the current word. For example in the sentence "Alice won the tournament, so she is proud." it could be expected that the vector for "she" would give a high score to the word "Alice" because Alice is the subject being referred to by the word "she".

The self-attention layer first uses each of the input vector representing the embeddings to independently calculate a Query vector, a Key vector, and a Value vector. Each of these are calculated by multiplying the embedding vector by a Query matrix, Key matrix, and Value matrix. These matrices remain constant throughout this process after being generated during the training process.

For a given word to calculate the score of other words, the dot product between the Query vector of the current word and the Key vector of the other word is calculated. This is then divided by the square root of the dimension of the Key vectors, which is constant between words. Once this is done for every word (including the current word), these values are softmaxed. This softmaxed score represents how relevant that word is for the current word. Each word's Value vector is then multiplied by the corresponding score, resulting in weighted Value vectors wherein the impact of less relevant words is reduced in later calculations. Finally the weighted Value vectors are summed, producing a matrix that is the self-attention layer's output for the current word.

The self-attention layer applies these calculations using matrices for every word in the sequence, generating an output for each of them. These outputs are then passed to the feed forward neural network.

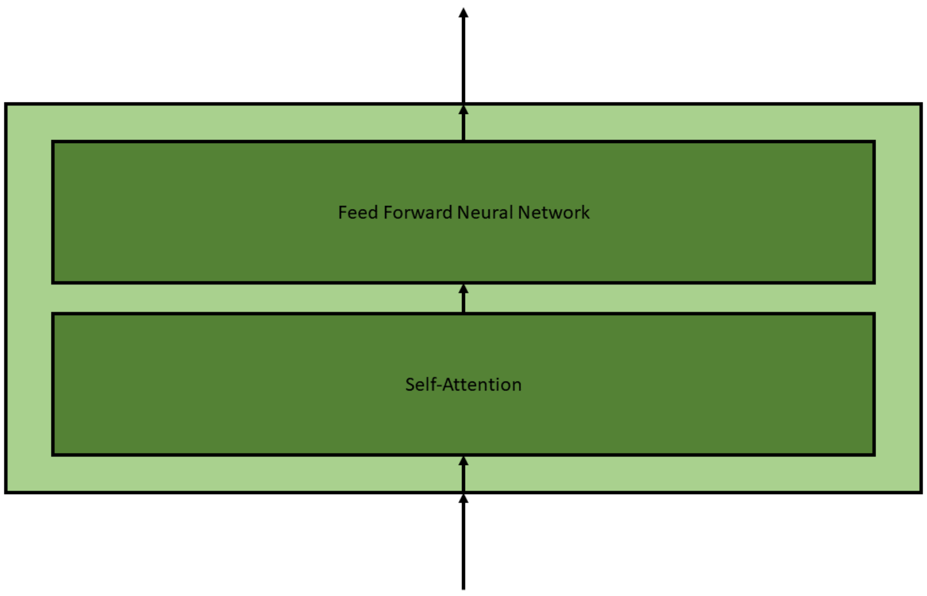


Figure 2.3: A visualisation of an encoder and its components

Multi-headed Self-attention

The self-attention layer can be defined with the addition of a multi-headed attention mechanism which allows for improved focus on multiple words in the sequence via multiple representation subspaces. These subspaces are represented by using multiple Query, Key, and Value matrices, each of which is part of its own attention head. These matrices still remain constant throughout this process after being generated during the training process.

Each attention head performs the same self-attention calculation explained above. Due to the Query, Key, and Vector matrices being different in each head, the matrices calculated by head attention head will be different. However before these matrices can't be passed to the feed forward neural network without first being combined.

The results of the attention heads are concatenated to produce a larger matrix, and then multiplies by a weight matrix with dimensions such such that the result will have the correct dimensions to be passed to the feed forward neural network. This weight matrix remains constant throughout this process after being generated during the training process.

2.3.2 Accounting for Sequence Ordering

The process described so far does not take the order of the input sequence into account. To do this, when the words are encoded a vector is added to each word that represents that words position. This vector is generated by a predefined function and is based on the number of words and the size of the embedding. The patterns of the vectors generated by this function are learned by the model during the training process. Adding these vectors to each word's vectors allows the model to better determine each word's position, and the distance between words in the sequence.

2.3.3 Decoders

Each decoder contains a self-attention layer, an encoder-decoder attention layer, and a feed forward neural network. In each decoder this layer is fed the Key attention vector and the Value attention vector from the last encoder. These vectors better allow each decoder to focus on the relevant words in the input sentence.

Once the decoding phase has completed the encoding phase begins. Each step of this phase takes the output of the previous step as input and produces a new word as part of the output sequence, building up the output sequence. However a key difference is that attention can only be on previous words of the sequence when generating a new word, which is implemented by setting future positions to negative infinity before the softmax step of the self-attention calculation.

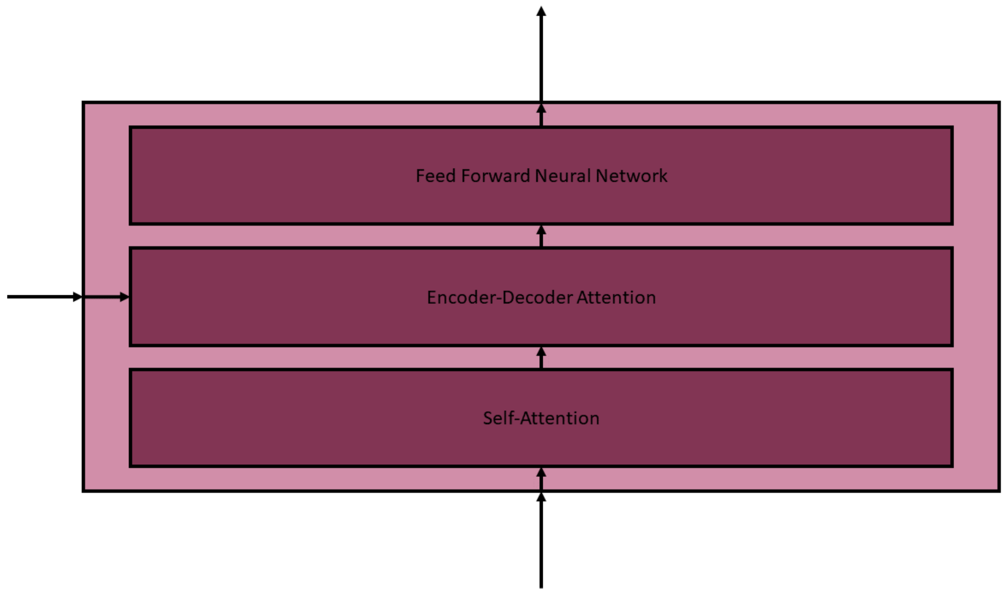


Figure 2.4: A visualisation of an decoder and its components

2.3.4 Generating Words

The decoder stack generates an output in the form of a vector of floats. This is converted into a string by the final Linear layer and Softmax layer. First the linear layer uses the vector from the decoders to generate a vector with a size equal to the number of possible words. Each value in this vector corresponds to a specific word where higher values mean the word is more likely.

The softmax layer then converts this into a vector in which the value for each word is the probability of that word, meaning the values in the vector must add up to 1. Therefore cell with the highest value in the vector has the highest probability of being the word to generate, and can therefore be selected as the word to be generated.

2.4 gpt-2-simple

This project uses version 0.7.1 of gpt-2-simple which is an open-source “Python package that wraps existing model fine-tuning and generation scripts for OpenAI’s GPT-2 text generation model” [14]. It also has specific functionality for Google Colab to make it easier to utilise the provided cloud storage.

2.4.1 Downloading GPT-2 Models

This library includes a function to facilitate downloading the GPT-2 models. Downloading the GPT-2 models allows for the interaction with them. All for model sizes are downloadable, which are: 124M, 355M, 774M, and 1558M. These vary in size being approximately 480MB, 1.3GB, 2.9GB, and 5.8GB from smallest to largest. Due to the larger models having more parameters they perform better when generating text, but they also take longer to finetune and generate text.

The function in the library that downloads the models is `download_gpt2`. It takes up to 2 arguments, called `model_dir` and `model_name`. `model_dir` specifies the folder in which the model will be located, in which a subfolder is created for each downloaded model. The default value for this is `'models'`. `model_name` specifies which model to download, which is `'124M'` by default. The values this can take are `'124M'`, `'355M'`, `'774M'`, and `'1558M'`.

2.4.2 Generating Text With GPT-2 Models

The `generate` function facilitates generating text using GPT-2 models. Both the default downloaded models and finetuned models can be used to generate text. The function takes the following arguments:

- `sess`: The TensorFlow session object that represents the current TensorFlow session. This can be generated using the `start_tf_sess` function.
- `run_name`: A string that is the name of the model which is used when loading a model that has already been finetuned, with a default value of `'run1'`.
- `checkpoint_dir`: A string that is the directory containing models that have already been finetuned, with a default value of `'checkpoint'`.
- `model_name`: A string that is the name of the model size the sample is being generated using, values this can take are `'124M'`, `'355M'`, `'774M'`, and `'1558M'`, with a default value of `None`.
- `model_dir`: A string that is the directory containing already finetuned models that may be used, with a default value of `'models'`.
- `sample_dir`: This is currently not used when generating samples, and has a default value of `'samples'`.

- `return_as_list`: A boolean that represents whether or not the generated sample(s) will be returned as a list where each value in the list is a sample (when it is `True`) or will be printed to console (when it is `False`), where the default value is `False`.
- `truncate`: A string for which the sample generated will be cut at, making the sample anything before the truncated value, this has a default value of `None` which disables this functionality.
- `destination_path`: A string that is the filename to write the generated samples to, with a default value of `None` which disables this functionality.
- `sample_delim`: A string used to separate samples, with a default value of `'='  20 + '\n'` which represents 20 equals characters followed by a newline.
- `prefix`: A string to start each sample with, with a default value of `None` which means samples are generated with no specified prompt.
- `seed`: An integer that sets TensorFlow's seed for generating pseudo-random values [15].
- `nsamples`: An integer that is the number of samples to be generated in total with a default value of `1`.
- `batch_size`: An integer that is the number of samples to generate simultaneously with a default value of `1`. This value must divide the value of `nsamples`, and only has a performance impact.
- `length`: An integer that is the number of tokens (words) each sample has with a default value of `1023`. This is in addition to the prefix (if there is one), and truncation is performed after the sample of this length is generated (if at all).
- `temperature`: A "Float value controlling randomness in boltzmann distribution. Lower temperature results in less random completions. As the temperature approaches zero, the model will become deterministic and repetitive. Higher temperature results in more random completions" [16].
- `top_k`: An "Integer value controlling diversity. 1 means only 1 word is considered for each step (token), resulting in deterministic completions, while 40 means 40 words are considered at each step. 0 (default) is a special setting meaning no restrictions" [16].

- `top_p`: A float value between `0.0` and `1.0` that dictates what proportion of the top of the distribution to sample words from with a default value of `0.0`. This is only used if the value is nonzero and then overrides the use of `top_k` [16].
- `include_prefix`: A boolean that represents whether or not the prefix specified by `prefix` (if there is one) will be included in the generated sample with a default value of True, meaning the prefix will be included in the sample.

2.4.3 Finetuning GPT-2 Models

The `finetune` function allows for the finetuning of GPT-2 models, which is a type of transfer learning. "Transfer learning refers to a learning scheme where weights of a model that are already optimized for a certain task are used for learning a slightly different task" [17]. Both the default downloaded models and the finetuned models can be further finetuned by this, allowing for finetuning of the same model at different points in time, and even using different finetuning arguments. The function takes the following arguments:

- `sess`: The TensorFlow session object that represents the current TensorFlow session. This can be generated using the `start_tf_sess` function.
- `dataset`: A string that is the relative path to the file containing the text to finetune the model on.
- `steps`: An integer that is the number of epochs to train the model for until stopping.
- `model_name`: A string that is the name of the model size being finetuned, values this can take are `'124M'`, `'355M'`, `'774M'`, and `'1558M'`, with a default value of `None`.
- `model_dir`: A string that is the directly containing already finetuned models that may be further finetuned, with a default value of `'models'`.
- `batch_size`: An integer that is the number of samples to generate simultaneously when generating samples, with a default value of `1`. This value must divide the value of `sample_num`, and only has a performance impact.
- `learning_rate`: A float that is the learning rate passed to the TensorFlow optimizer being used [18][19].

- `accumulate_gradients`: An integer that represents the number of training steps to perform "without updating the model variables while accumulating the gradients of those steps and then using the accumulated gradients to compute the variable updates" [20].
- `restore_from`: A string that represents whether at the start of finetuning a new model that has not been finetuned is loaded (with the value being `'fresh'`), or the latest version of the model with the same `model_name` is loaded (with the value being `'latest'`). The default value for this is `'latest'`.
- `run_name`: A string that is the name of the model which is used when saving the model and when loading a model that has already been finetuned, with a default value of `'run1'`.
- `checkpoint_dir`: A string that is the directory containing models that have already been finetuned, with a default value of `'checkpoint'`.
- `sample_every`: An integer that is the number of steps before generating and printing another sample, with a default value of `100`. This generating and printing is repeated every time this number of steps passes until the finetuning stops.
- `sample_length`: An integer that is the number of tokens (words) each sample has when generated, with a default value of `1023`.
- `sample_num`: An integer that is the number of samples to be generated every time samples are generated, with a default value of `1`.
- `multi_gpu`: A boolean that represents whether or not to utilise all GPUs the system has, where `True` means to use all GPUs and `False` means to use at most 1 GPU. This has a default value of `False`
- `save_every`: An integer that is the number of steps before saving the model to disk, with a default value of `1000`. This saving is repeated every time this number of steps passes until the finetuning stops.
- `print_every`: An integer that is the number of steps before saving the model to disk, with a default value of `1000`. This saving is repeated every time this number of steps passes until the finetuning stops.
- `max_checkpoints`: An integer that is the number of versions of the model to save to disk before deleting the oldest versions, with a default value of `1`.

- `use_memory_saving_gradients`: A boolean that if `True` finetunes using “memory efficient gradient implementation inspired by ’Training Deep Nets with Sublinear Memory Cost”’ [21][22]. The default value is `True` but if a model larger than `’124M’` is being used, this will be set to `True` regardless of the value passed.

- `only_train_transformer_layers`: A boolean value that if `True` only changes the transformer layers when fintuning the model. The default value is `True` but if a model larger than `’124M’` is being used, this will be set to `True` regardless of the value passed.

- `optimizer`: A string that is the name of the optimiser used to train the TensorFlow model in the finetuning process. This can either be `’adam’` or `’sgd’`, and is `’adam’` by default.

- `overwrite`: A boolean value that represents whether or not to overwrite an existing model with the same `run_name` when saving the model to disk. The default value of this is `False`.

2.5 Discord.py

This project uses version 1.2.5 of discord.py which is an open source “async ready API wrapper for Discord written in Python” [23]. It has functionality that allow it to trigger functions from command words by “attaching [the command annotation] to a regular Python function” [24].

2.6 Datasets

All the News is a dataset consisting of over 200,000 news articles from a variety of publications where “the data primarily falls between the years of 2016 and July 2017, although there is a not-insignificant number of articles from 2015, and a possibly insignificant number from before then” [25].

The basis of this will form the dataset that will be used in the finetuning process of the GPT-2 model behind the article generating program.

A dataset of a WhatsApp chat log was finetuned on for informal testing. This was comprised of approximately 30,000 messages sent from up to 9 members and was used in the raw format WhatsApp exported it as. A dataset made up of approximately 1,000 cocktail recipies was also used to finetune a model for informal testing.

2.7 Existing Tools

Due to the recent release date of GPT-2 in February 2019 [26], there are not many publicly available online services to allow users to interact with GPT-2.

Talk to Transformer is a website that uses GPT-2 to show "how a modern neural network completes your text" [27]. The website "runs the full-sized GPT-2 model, called 1558M" [27] with no finetuning beyond the fresh model, with all processing being performed server side. This model is used to generate text by taking a prompt from the user and using that as the prefix when generating so that a continuation of it is produced.

Write With Transformer is a "web app, built by the Hugging Face team" [28] that uses "modern neural network[s] to auto-complete" [28] text written by the user. This tool allows for using any of the four model sizes. It also allows for adjusting the `top_p` parameter to a value between 0 and 1 (inclusive), and the `temperature` parameter to a value between 0 and 3 (inclusive), both to 2 decimal places.

However these websites have limitations in their usage:

- There is no publicly available API available to allow for using these tools to generate text as part of a software package.
- There is no way for the user to finetune or use an already finetuned model to generate text via these platforms.
- These tools cannot be used offline or utilise the user's own hardware to generate text

These are limitations that will be addressed by the software developed in this project.

Chapter 3

Requirements & Specification

Both of these pieces of software should be designed so that a user with no technical knowledge can use them to interact with the GPT-2 models and generate text. The text generated will be specialised from a pre-finetuned GPT-2 model in the case of the Article Generator, and will be more generalised from a fresh GPT-2 model in the case of the Discord bot.

They should also allow for a user/administrator with the sufficient technical knowledge to use a GPT-2 model that they have finetuned themselves. Enabling both users with specialised knowledge to make use of their own GPT-2 models in these programs.

3.1 Requirements

Both the Discord Bot and the Article Generator will be written in Python and therefore require the suitable version of Python for the operating system being run on.

TensorFlow is required by gpt-2-simple, either a version with GPU support or without GPU support. As recommended by gpt-2-simple "for finetuning, [...] use a GPU, although you can generate using a CPU (albeit much more slowly)" [14]. TensorFlow gpu support also has a hardware requirement of an "NVIDIA® GPU card with CUDA® Compute Capability of 3.5 or higher" [29]. The required version of TensorFlow without GPU support can be installed with the command `pip install tensorflow==1.15.2`. If this version of TensorFlow is to be used, the hardware GPU requirement and the last 4 requirements in each table below are not necessary.

3.1.1 Article Generator Requirements

The Article Generator will make use of the dependencies below and will therefore require them in order to function.

Name	Version	Source
GPT-2 Simple	0.7.1	pip install gpt-2-simple==0.7.1
TensorFlow	1.15.2	pip install tensorflow-gpu==1.15.2
NVIDIA GPU drivers	418+	https://www.nvidia.com/drivers/
CUDA Toolkit	10.1+	https://developer.nvidia.com/cuda-toolkit-archive/
CUDA Profiling Tools	10.1+	[Included with CUDA Toolkit]
cuDNN SDK	7.6+	https://developer.nvidia.com/cudnn

3.1.2 Discord Bot Requirements

The Discord Bot will make use of the dependencies below and will therefore require them in order to function.

Name	Version	Source
GPT-2 Simple	0.7.1	pip install gpt-2-simple==0.7.1
Discord.py	1.2.5	pip install discord.py==1.2.5
TensorFlow	1.15.2	pip install tensorflow-gpu==1.15.2
NVIDIA GPU drivers	418+	https://www.nvidia.com/drivers/
CUDA Toolkit	10.1+	https://developer.nvidia.com/cuda-toolkit-archive/
CUDA Profiling Tools	10.1+	[Included with CUDA Toolkit]
cuDNN SDK	7.6+	https://developer.nvidia.com/cudnn

3.2 Specification

3.2.1 Article Generator Specification

- The application should be capable of generating articles using a GPT-2 model on disk.
- The application should not rely on an internet connection, using only local resources.
- The application should have a GUI that provides access to its functionality.
- The user should be able to provide a title for the fictional article being generated.
- The user should optionally be able to provide some initial content for the article.

- The user should be able to provide this above information by either manually entering it, or via a text file on disk.
- The user should be able to generate multiple articles at once, each with a varied completion of the input provided.
- The user should be able to specify the maximum length of articles generated.
- The user should be able to do all of the above from the command line.
- The user should be able to use the command line to either print the generated articles, or write them to files on disk, or both at once.

3.2.2 Discord Bot Specification

- There must be a way for an administrator to provide a Discord API key without editing the script.
- The generation of text must utilise the administrator's hardware.
- Users should be able to execute commands via Discord messages sent in the same Discord server as the bot.
- Users should be able to download fresh GPT-2 models for the bot to use.
- Users should be able to generate text with a downloaded model with a specified prompt.

3.2.3 Limitations

Finetuning the models in preparation for specialising them for a specific type of content can very be computationally expensive, especially with the larger models. "The 774M 'large' model may support finetuning because it will cause modern GPUs to go out-of-memory" [14]. While generating text using the models is also more difficult with the larger models "you can still generate from the default pretrained model using [the 774M model]" [14].

The programs will use the smallest model, which is the 124M model, in order to minimise the difficulty of computation. This can allow text to be generated faster and also reduce the performance required of the machines running them. This also reduces the likelihood of the models overfitting datasets that may not be large enough.

Chapter 4

Design

This chapter will detail how the article generator program and the discord bot are structured from a high level view. This will provide the foundation for which the implementation of each will be based on.

4.1 Article Generator Architecture

The architecture of the article generator program consists of multiple subsystems that will interact with each other as part of a three-tiered architecture. The three-tiered architecture was selected as it allows components to better adhere to the single-responsibility principle. It can also allow for the modules in each tier to be maintained separately from those in other tiers. This reduction is coupling allows for easier maintenance of the program as there is more focus on relying on the promised interface as opposed to the specific implementation.

The architecture diagram below in Figure 4.1 shows which tier each component is in. Arrows represent a dependency, where the component at the base of the arrow depends on the component at the head of the arrow.

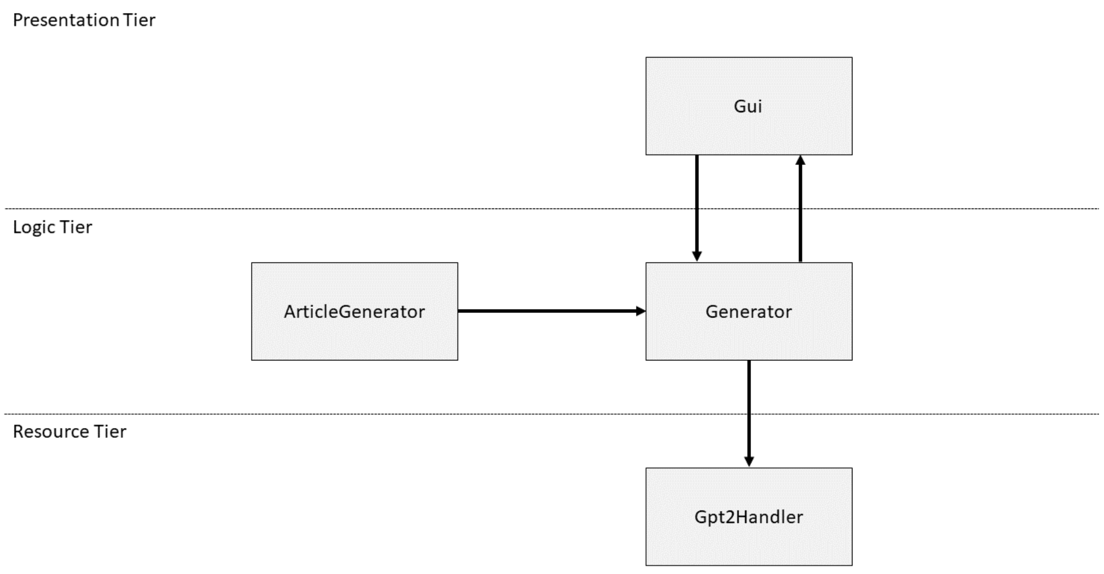


Figure 4.1: The Architecture Diagram for the Article Generator

4.1.1 Article Generator Architecture Tiers

Each of the components in this program will be separated into one of the three tiers which are the presentation tier, the logic tier, and the resource tier.

Presentation Tier

The presentation tier is the highest level of the program. It contains a single Gui component, whose responsibilities relate to displaying the necessary information to the user. It is also responsible to passing all communication and interaction from the user to other tiers. This can range from passing button clicks, to text input from the user to the other tiers. This tier also performs basic data validation such as ensuring a field contains an input or ensuring the input in a field is of the right type. The Gui should also respect the singleton design pattern as only a single instance of it should exist.

Logic Tier

The logic tier controls the functionality of the program. It contains the ArticleGenerator component, and the Generator component. The ArticleGenerator is responsible for parsing command line arguments and ensuring the combination of arguments is valid. Then it invokes the relevant methods in the Generator component, passing the necessary arguments if required.

The Generator component is responsible for facilitating interaction between the resource tier and the presentation tier. The methods in this component pass arguments to Gpt2Handler after performing small pre-processing steps on the user input from the presentation and post-processing on the output from the resource tier. It should also respect the singleton design pattern as only a single instance of it should exist.

Resource Tier

The resource tier contains a single Gpt2Handler component, whose responsibilities involve interacting with the already finetuned GPT-2 model. It converts the input into a format so that the completion the model generates can be properly split by this component. This completion can then be returned as an output with the title and sample separated. It should also respect the singleton design pattern as only a single instance of it should exist.

4.2 Discord Bot Architecture

The architecture of the discord bot consists of a simple bot component and multiple cog components. The bot component loads the cog components and starts the bot via functions in the `discord.py` library/ Cog components “organize a collection of commands [...] and some state into one class” [30].

The architecture diagram below in Figure 4.2 shows the interactions between components, including cogs. Arrows represent a dependency, where the component at the base of the arrow depends on the component at the head of the arrow.

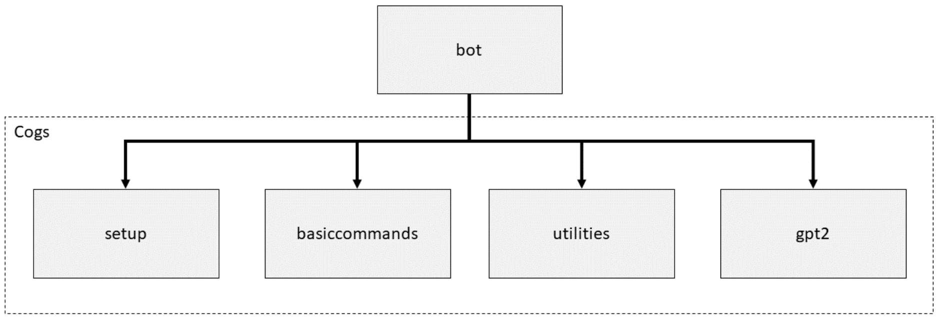


Figure 4.2: The Architecture Diagram for the Discord Bot

4.2.1 Discord Bot Cogs

Cogs are a feature of the `Discord.py` library that facilitate separating functionality into their own classes. Cogs can be loaded into a bot, allowing the bot to use all the functionality provided in the cog. Use of cogs can allow for better adherence to the single-responsibility principle, as the functionality each cog provides can be written to be related, thus increasing cohesion. Cogs also do not need to interact with each other, allowing for easier separate maintainability. These allow for reduced coupling, making maintenance of the bot easier. Due to the cogs acting as separate modules, they can even be taken from one bot and placed in another without the need of much refactoring.

setup Cog

The setup cog will include anything required in setting up the bot that isn't covered by the `discord.py` library or other cogs. This can even be as simple as sending a message to notify users that the bot is ready, or print such a message to console so that only the admin will be notified.

basiccommands Cog

The basiccommands cog will include basic commands that aren't crucial to the functionality to the bot but can provide information about the bot. Because these commands will not be necessary to making the bot function, they will be placed in their own cog. These may include commands used for convenience or debugging, such as checking the latency of the bot.

utilities Cog

The utilities cog will allow the user to perform commands that directly affect the functionality of the bot. This is used as a convenience feature that means the administrator does not have to manually restart the bot or or use the command line to perform the provided functionality.

For example when adding a new cog or editing an existing cog, the administrator would usually have to restart the bot so that all cogs are loaded again. The utilities load (if loaded) will allow for loading cogs by sending a command via a discord message.

gpt2 Cog

The gpt2 cog will provide the main functionality of the bot, being interacting with GPT-2 models. It will allow users to generate samples with a custom prompt, and change various parameters that affect the configuration of the bot (e.g. the name of the model to use). It will also provide additional functionality that is either useful or required to generate samples, such as downloading one of the four default models.

4.3 Article Generator GUI

The GUI (Graphical User Interface) of the program will be a core aspect of how the user interacts with it. It must be simple enough that a user can use it intuitively, without compromising on the functionality of the program.

4.3.1 Home Screen

The home screen is the first window that appears when the user launches the program. It contains fields for user to input text and adjust parameters used in the generation process.

Title

The first line contains a label, a drop down menu, and a text field. The label contains the text `Title:` to indicate to the user that the text field in this line represents the title. The user can write any text in the text field except for newline characters, ensuring user input is a single line.

The drop down menu allows the user to select between `Text` and `File`, with the former being the default value. If the user selects `File` in the drop down box, a file dialog will appear allowing the user to select a text (`.txt`) file by default, with the option to switch the type to any file. The contents of this file will overwrite any text in the text field and disable entry in the field. But if the text field contains content, a dialog box will appear before the file dialog asking the user if they wish to continue.

If the user selects `Text` in the drop down menu, the contents of it will remain the same and user input will be enabled (if it was disabled).

Initial Content

The second line is very similar to the previous, but the label contains the text `Initial Content:`. An additional difference is that the text field spans a height of 10 lines, indicating that the user input can do so to. For this reason unlike the title text field, this text field can accept newline characters.

Number of Samples

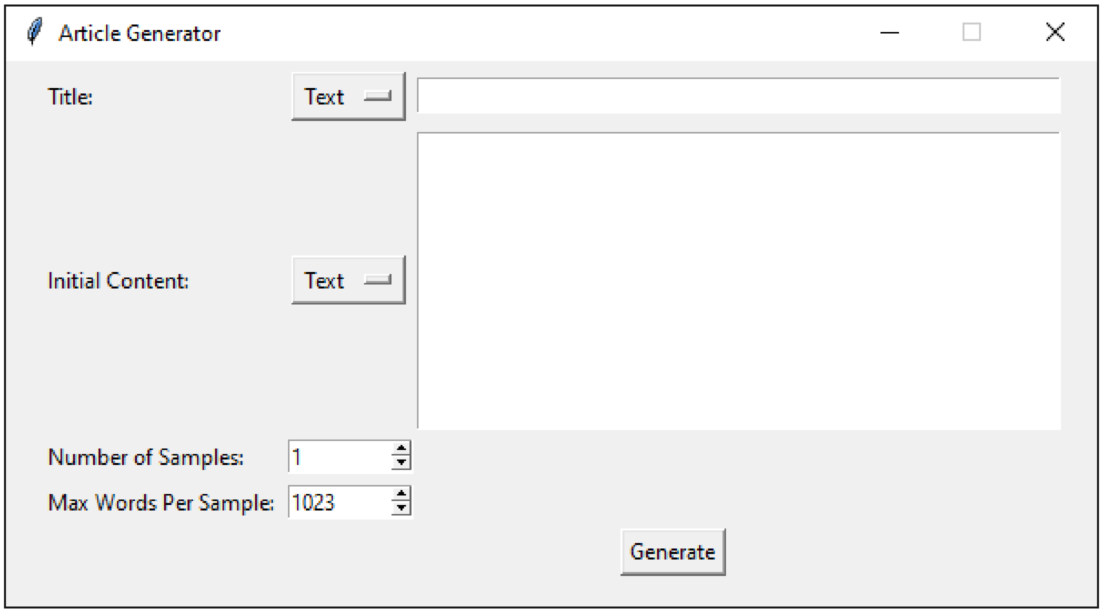
The third line contains a label with the text `Number of Samples:` and a spinbox with a default value of 1. This spinbox can take values that are integers ranging from 1 to 99 (inclusive). The user can also input a value by typing it into the spinbox, but it will be rejected when text generation is attempted if it's not a valid value.

Max Words Per Sample

The fourth line is much like the previous line, but the label contains the text `Max Words Per Sample:`. An additional difference is that the spinbox for this takes values that are integers ranging from 1 to 1023 (inclusive), with a default value of 1023.

Generate

The fifth and final line contains a button that is centered in the column of the above text fields. This button contains the text `Generate` and once pressed begins the process of generating the article(s) based on the user input. However before generating the article(s), some basic checks on the inputs will take place. This includes ensuring the text field for the title is not empty, and ensuring the values in the spinbox are both valid for their respective spinbox.



Number of Samples: 1
Max Words Per Sample: 1023

Figure 4.3: The Article Generator's Home Screen as it First Appears

4.3.2 Article Viewer

The article viewer is a window that appears once the samples have been generated. It's purpose is to display all of the samples such that the user can switch between them. When this appears it will be on top of the home screen which can only be interacted with after the sample viewer is closed.

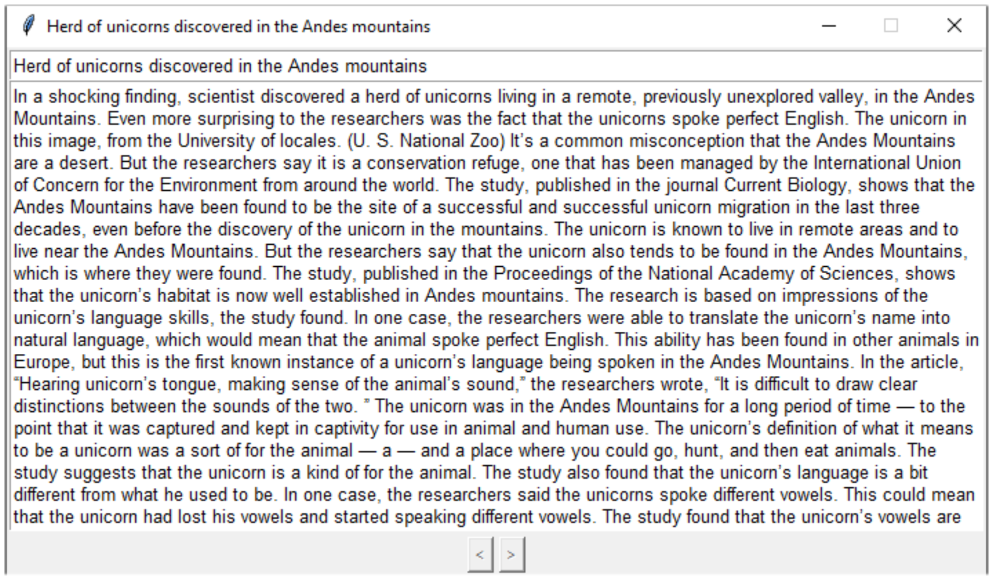
Displaying the Articles

The first line contains a text field that spans the width of the whole line. This text field contains the title of the article the user inputted, and cannot be edited by the user.

The second line contains a text field that spans the width of the whole line and is 20 lines tall. This allows the text field to accommodate a larger amount of text, as expected in the body of a news article. The text field contains the body of the article, starting with the initial context the user inputted, and cannot be edited by the user.

4.3.3 Switching Between the Articles

The third line contains 2 buttons around the center line. The first contains a <, and the second contains a >. Since all of the articles must share the same title, only the body of the articles need to change. The first button changes the body of the article to that of the previous one, and the second changes the body of the news article to that of the next one. If either the first or last articles are reached, the first or second button (respectively) will be disabled.


Herd of unicorns discovered in the Andes mountains

Herd of unicorns discovered in the Andes mountains

In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English. The unicorn in this image, from the University of locales. (U. S. National Zoo) It's a common misconception that the Andes Mountains are a desert. But the researchers say it is a conservation refuge, one that has been managed by the International Union of Concern for the Environment from around the world. The study, published in the journal Current Biology, shows that the Andes Mountains have been found to be the site of a successful and successful unicorn migration in the last three decades, even before the discovery of the unicorn in the mountains. The unicorn is known to live in remote areas and to live near the Andes Mountains. But the researchers say that the unicorn also tends to be found in the Andes Mountains, which is where they were found. The study, published in the Proceedings of the National Academy of Sciences, shows that the unicorn's habitat is now well established in Andes mountains. The research is based on impressions of the unicorn's language skills, the study found. In one case, the researchers were able to translate the unicorn's name into natural language, which would mean that the animal spoke perfect English. This ability has been found in other animals in Europe, but this is the first known instance of a unicorn's language being spoken in the Andes Mountains. In the article, "Hearing unicorn's tongue, making sense of the animal's sound," the researchers wrote, "It is difficult to draw clear distinctions between the sounds of the two. " The unicorn was in the Andes Mountains for a long period of time — to the point that it was captured and kept in captivity for use in animal and human use. The unicorn's definition of what it means to be a unicorn was a sort of for the animal — a — and a place where you could go, hunt, and then eat animals. The study suggests that the unicorn is a kind of for the animal. The study also found that the unicorn's language is a bit different from what he used to be. In one case, the researchers said the unicorns spoke different vowels. This could mean that the unicorn had lost his vowels and started speaking different vowels. The study found that the unicorn's vowels are

Figure 4.4: The Article Generator's Article Viewer as it First Appears

4.4 Article Generator CLI

The CLI (Command Line Interface) will be crucial for allowing other programs to interact with this program. It must be clear and consistent in how it takes and handles arguments, while providing providing the necessary functionality for such a use case. The functionality it provides must be as much as or more than the functionality provided by the GUI.

Passing no arguments to the program results in the GUI being launched. If arguments are passed but they are an invalid combination or have invalid values, an exception will be thrown to notify the user of this. Bellow are the arguments that can be passed and what they do:

- `-h`, `--help`: Print a help message for using each of the command line arguments, then quit the program.
- `-f FILENAME`, `--filename FILENAME`: Use the title and initial content in a file with the filename specified by `FILENAME`. The file should contain the title in the first line, and initial content (if any) in the second line..
- `-o OUTPUT_FILENAME`, `--output_filename OUTPUT_FILENAME`: Write the generated article to a new file with the filename specified by `OUTPUT_FILENAME`. The file must not already exist. The sample number is appended to the filename before the extension if multiple samples are to be generated.
- `-p`, `--print`: Print the generated sample(s) to console.
- `-n NUM_SAMPLES`, `--num_samples NUM_SAMPLES`: Generate a number of samples equal to `NUM_SAMPLES`. It must be a positive integer. Default: `1`.
- `-w NUM_WORDS`, `--num_words NUM_WORDS`: Generate samples with a number of words that is not greater than `NUM_WORDS`. It must be a positive integer that does not exceed `1023`. Default: `1023`.
- `-t TITLE_FILENAME`, `--title_filename TITLE_FILENAME`: Use the text in the first line of the file specified by `TITLE_FILENAME` as the title. This will be ignored if a filename for `--filename` is specified.
- `-c CONTENT_FILENAME`, `--content_filename CONTENT_FILENAME`: Use the text in the first line of the file specified by `CONTENT_FILENAME` as the initial content. This will be ignored if no filename for `--title-filename` is specified.

- `-T TITLE`, `--title TITLE`: Use the text specified by `TITLE` as the title. This will be ignored if a filename for `--filename` or `--title_filename` is specified.
- `-C CONTENT`, `--content CONTENT`: Use the text specified by `CONTENT` as the initial content. This will be ignored if no title for `--title` is specified.

4.5 Discord Bot Setup

Using this program requires the admin to "first create a Discord Bot account" [31]. This section will step through stages in the "Creating a Bot Account" and "Inviting your bot" guides in the Discord.py documentation [31].

1. Login to the Discord website: `https://discordapp.com/`.
2. Navigate to the Developer Portal under the `Developers` tab, and make sure the `Applications` section is open.
3. Click the `New Application` button.
4. Enter a name for the application then click the `Create` button.
5. Navigate to the `Bot` section from the sidebar.
6. Click the `Add Bot` button then click the `Yes, do it!` button on the popup.
7. Click the `Copy` button under the `Token` heading. The API Key for the bot will now be in the clipboard.
8. Create a file called `apikey.txt` in the same directory as the `bot.py` file. Paste the API Key into it.
9. On the web page navigate to the `OAuth2` section from the sidebar.
10. Tick the `bot` checkbox under the `scopes` heading.
11. Tick the `Send Messages` and `Manage Messages` checkboxes under the `Bot Permissions` heading.
12. Copy and open the URL in the field under the `Scopes` heading. This should navigate to a new page.
13. Select the server to add the bot to then press the `Continue` button, then press the `Authorize` button.

4.6 Discord Bot UI

The user can interact with the Discord Bot by sending a message that matches a command recognised by the bot. These messages must be sent in a Discord server that also contains the bot. All commands must begin with the `;;` prefix.

4.6.1 Downloading a Fresh Model

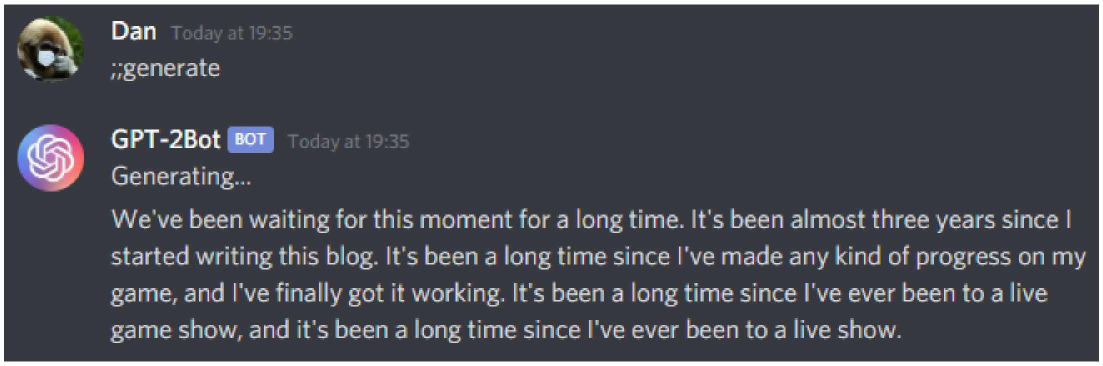
The `;;download_model` command can be used to download one of the four fresh models (124M, 355M, 774M, and 1558M). If used without an argument, it downloads the 124M model by default. However, any of the four models can be downloaded by specifying their model name.

Sending the message `;;download_model` results in the 124M model being downloaded. Sending the message `;;download_model 355M` results in the 355M model being downloaded. This works with any of the four model names. If the model name is valid the message `Model downloaded` will be sent by the bot after downloading the model, otherwise the message `ERROR: Invalid argument` will be sent by the bot and no model will be downloaded.

4.6.2 Generating a Sample

The `;;generate` command is used to generate samples using a downloaded model. The model used is determined by the model named in the configuration.

Sending the message `;;generate` results in the message `Generating...` sent by the bot before the next message which contains the sample generated by the bot. This sample has no prompt as none was provided by the user. Sending `;;generate` but with any amount of text written following it in the same message will result in the same functionality, but with the text provided being used as a prompt to start the sample with.



Dan Today at 19:35
;;generate

GPT-2Bot BOT Today at 19:35
Generating...
We've been waiting for this moment for a long time. It's been almost three years since I started writing this blog. It's been a long time since I've made any kind of progress on my game, and I've finally got it working. It's been a long time since I've ever been to a live game show, and it's been a long time since I've ever been to a live show.

Figure 4.5: An example of the ;;generate command being used without a prompt

4.6.3 Changing the Configuration

The `;;set_model` command is used to change the model used when generating text. However, it can only be one of the four valid fresh model names.

Sending the message `;;set_model` results in the message `ERROR: Argument required` being sent by the bot. Sending the message `;;set_model` followed by a valid fresh model name results in the configuration being updated so that the model name is set to that given by the user. However if the model name given is not a valid fresh model name, the message `ERROR: Argument required` will be sent by the bot.

Chapter 5

Implementation

This chapter will detail how each program is implemented and the process of finetuning the model used in the article generator.

5.1 Article Generator

The program is launched by running `ArticleGenerator.py` with arguments. But if no arguments are provided, the GUI runs instead. This section will first step through how each of the modules is used, both when the program is being used through the CLI and when the program is being used through the GUI.

5.1.1 ArticleGenerator.py

When the article generator runs it first creates a parser. The creation of the parser involves initialising an `ArgumentParser` object from the built in `argparse` package. The usage string to be used in the help message is set, before each of the arguments are added.

Every argument is added as an optional argument, meaning none of the arguments are required when parsing. Each of the arguments added has 2 aliases, a destination variable name (`dest`), and a help message. Some arguments may also have a type, a default value, an an action.

Alias

Each alias provides a way for the user to use the argument. An example of this is where the argument with aliases `-f` and `--filename` can be used with either `-f example.txt` or

`--filename example.txt`.

Destination Variable

The destination variable name is a string that specifies the name of the variable to store the value of the given argument in. An example of this is where the argument with `dest` set to `filename` will have it's value stored in the `filename` variable in the Namespace returned after parsing.

Help Message

The help message is displayed when the optional argument `-h` or `--help` is used when launching the program. An example of this is where the argument with `help` set to `Print the generated sample(s) to console.` will cause this message to be printed next to `-p, --print` when the parser prints the help message.

Type

The type of an argument has been specified via lambdas/functions which attempt to transform the value, but throw an error if the value is not valid. This transformation has only been implemented as just type casting, but the checks of valid values have varied in complexity. These range from ensuring the value is positive, to ensuring the value is the name of a file that exists. If any of these checks fail an `ArgumentTypeError` (from the `argparse` package) is raised to indicate this. Upon raising this Error the help message that is normally printed when using `-h` is printed, alongside a message that specifies which argument the error relates to and why it has been raised.

An example of this is where an argument has its type set to the `existing_filename_type` function. This function checks if the value is the name of a file that exists or not. If it is the value is returned without being transformed, but if it isn't an `ArgumentTypeError` is raised saying the value "is not the name of a file that exists".

Default Value and Action

An argument's default is the value it takes if none has been specified by the user. This default value can be set either explicitly by setting `default` to the desired value, or implicitly via the action. If an argument has no default value then it will default to `None` instead.

An argument's action specifies how the argument is handled by the parser. This can be set by setting `action` to the desired value, but the default value is `'store'`. `'store'` just stores the value in a variable in the Namespace. `'store_true'` is used by one of the arguments to indicate that the argument takes no value, but if the argument is used it stores `True`. This also sets the default value of the argument to `False`.

An example of an action being specified is where the action of an argument with the alias `-p` is set to `'store_true'`. This means if `-p` is one of the arguments used, no value is required after it and instead the value `True` is stored for the respective variable.

Passing Arguments to Generator

Once the arguments are parsed, articles are generated based on which arguments have been parsed. But there are checks made before this.

First the program ensures that at least one way of outputting the articles is specified. This can either be printing to the console or writing to a file. If neither of these is specified, an Exception is thrown with a message explaining this. If the user specified to write the articles to files and the filename of the file is the same as the filename of a specified input file (i.e. `filename`, `title_filename`, or `content_filename`), then an Exception is thrown with a message explaining this.

If these checks are passed to an instance of `Generator`. The method used on the `Generator` instance depends on which arguments were defined by the user. If an argument for `-f` was given `Generator.generate_from_single_file()` is used with the argument for `-f` passed to it, if instead an argument for `-t` was given `Generator.generate_from_files()` is used with the arguments for `-t` and `-c` passed to it, but if instead and argument for `-T` was given `Generator.generate()` is used with the arguments for `-T` and `-C` passed to it.

In all three of the above cases the arguments for `-n`, `-p`, `-o`, and `-w` are also passed to the respective method in `Generate`. If the condition for multiple of these cases is satisfied the earlier case executes, which is information provided in the help message to the user. However if the condition for none of these cases are satisfied, an Exception is raised due to no input being specified.

5.1.2 gui.py

The Gui class

The Gui class respects the singleton design pattern, meaning it should only be initialised once. The constructor of this class should only be used internally, where the `Gui.get_instance()` class method is used to get the instance of the class. This method returns the instance of the class but if the instance doesn't already exist it is first created.

The constructor sets the class variable `__instance` to the object being constructed if one doesn't already exist, otherwise an exception is raised. Then all of the instance variables are initialised to `None` in order to make every instance variable in this object clear, allowing for better maintainability. After this the `Gui.create_gui()` method is called to create the GUI.

The `Gui.create_gui()` method creates the main window using `tkinter` before setting the title and size. The `Gui.create_home()` method is then used to create the home screen. This method creates and populates a tkinter `LabelFrame` object in order to create the home screen.

Objects in which the user inputs a value are stored in an instance variable so that their value can be retrieved when necessary. The `OptionMenu` objects that represent the drop down menus get their functionality by setting their `command` to the relevant method when initialising them. These methods are triggered when the value of a drop down is updated, calling the method with the new value.

If the new value is `'Text'` then the relevant text field becomes editable by changing its state to `'normal'`. Otherwise if there's text in the field, a `messagebox` is used to ask the user if they want to overwrite the contents of the respective text field. If the user chooses to continue a `filedialog` is created where the user can select a file, where the filetypes are Text Files (`*.txt`) and All Files (`*.*`). After the file is selected, the contents of the file is placed into the text field and the text field is made uneditable by setting it's state to `'disabled'`.

The functionality of the submit button is set similarly by setting its `command` to the relevant method when initialising it. This method first checks the required values have been inputted, and performs basic error checking on numerical values, like making sure they can be cast to `int` and ensuring they are not invalid values (e.g. not less than 1). Then it gets an instance of the Generator class by using `Generator.get_instance()` class method and generates articles from it by using the `Generator.generate_as_tuple()`. When calling this method the title, initial content, number of samples, and maximum words per samples, are values that are passed. The return value is then used to create a `SampleViewer` object, which is an inner class of the `Gui` class.

The SampleViewer class

The SampleViewer class represents a window used to display articles to the user. It's constructor takes a list of samples, each with a title and content, where the title is shared amongst the samples. These values are stored and all instance variables are initialised in the constructor in order to make every instance variable in this object clear, allowing for better maintainability.

After this the `SampleViewer.create_window()` method is called, which creates and populates the window used to show the samples. The window is an instance of `Toplevel` from the `tkinter` library and its title is set to the title of the article. `Toplevel.grab_set()` is used so that this window gains focus.

Buttons are created using `SampleViewer.create_buttons()` with their state being updated with `SampleViewer.update_buttons()`. The window also contains a `LabelFrame` containing the article text which is created with `SampleViewer.create_text_frame()` and updated with `SampleViewer.update_sample()`.

In the window the `LabelFrame` contains the two text fields that show the title and content. Below this `LabelFrame` are two buttons with a left arrow and right arrow, and their `command` set to `SampleViewer.previous_sample()` and `SampleViewer.next_sample()` respectively.

These functions change the index of the current sample and ensure it doesn't go out of the bounds of the list of samples, and then use `SampleViewer.update_sample()` to update the currently displayed sample. `SampleViewer.update_sample()` modifies the contents of the text field containing the article content to that of the current sample.

5.1.3 generator.py

The Generator class respects the singleton design pattern, meaning it should only be initialised once. The constructor of this class should only be used internally, where the `Generator.get_instance()` class method is used to get the instance of the class. This method returns the instance of the class but if the instance doesn't already exist it is first created.

Generate Articles

`Generator.generate_as_tuple()` is a static method that takes a string value representing the title and optionally a string value representing the initial content, an integer representing the number of articles of generate, and an integer representing the maximum number of words generated in each article. It then generates a list of samples, each being a list that contains two values, the first value being the title of the article and the second value being the content of the article as returned by the `Gpt2Handler` instance.

Alongside the values specified, the three methods below also each optionally take an integer representing the number of articles to generate, a boolean value representing whether or not to print the article(s), a string value representing the name of the file to write the sample to where `None` or the empty string means not to write to a file, and an integer value representing the maximum number of words generated in each sample.

`Generator.generate_from_single_file()` takes a string representing a filename and reads its content, treating the first line as the title and the seconds line (if any) as the initial content. These are then passed to the `Generator.generate()` method, returning the value it returns. If the file has no second line the empty string is used instead.

`Generator.generate_from_files()` takes a string representing the filename of a file that contains the title, and optionally a string representing the filename of a file that contains the initial content. The first line of each file is read before being passed to the `Generator.generate()` method, and returning the value it returns. If no file is specified for the initial content the empty string is used instead.

`Generator.generate()` takes a string representing a title, and optionally a string representing the initial content. If no initial content was passed, the empty string is used instead. These alongside the number of samples and maximum number of words are passed to `Gpt2Handler.generate_as_tuple()`, which generates a list of samples, each of which is a list where the first value is the title and the second value is the content. Each sample is then converting into a string that consists of the title, followed by a newline character, followed by the content of the sample. If the boolean specifying whether or not to print it to the console is set to `True`, the sample is printed. If a filename is given to write the samples to, the `Generator.write_samples_to_file()` is called with the filename and list of samples passed to it.

Writing to file

`Generator.write_samples_to_file()` is a method that controls how samples are written to file. If there is only 1 sample in the list passed, `Generator.write_sample_to_file()` is called with the filename and sample passed to it. However if there are multiple samples, the filename is modified for each sample so that an increasing number is appended to the filename before the extension.

For example if the filename to write to is `'example.txt'` and 2 samples are to be generated, the first sample is written to a file with the name `'example0.txt'` and the second sample is written to a file with the name `'example1.txt'`. This is done by looping through a list of integers that start at 0 and end before the number of numbers. Each time a sample is written to one of these files, `Generator.write_sample_to_file()` is used to write the current sample to the file with the current filename.

`Generator.write_sample_to_file()` takes a filename and sample. It creates a file with the given filename and writes the sample to it. The `'utf-8'` encoding is used, and `'surrogateescape'` is used to handle errors. This means that if attempting to decode a byte that is to be written to the file, the Unicode byte data will is written out as a string. While this isn't ideal, this results in such errors having a minimal effect when writing the samples.

5.1.4 gpt2handler.py

There are constants stored in this file that dictate certain arguments used when generating samples (`DEFAULT_CONFIG`), and that have lambdas that dictate how these are parsed (`GENERATE_ARGUMENT_PARSER`). These are in the form of dictionaries with keys being the names of arguments of `gpt_2_simple.generate()` and values being the respective value/lambda.

The Gpt2Handler class respects the singleton design pattern, meaning it should only be initialised once. The constructor of this class should only be used internally, where the `Gpt2Handler.get_instance()` class method is used to get the instance of the class. This method returns the instance of the class but if the instance doesn't already exist it is first created.

When the Gpt2Handler instance is created the TensorFlow session is started using `gpt_2_simple.start_tf_sess()`. Then `Gpt2Handler.download_model()` is used to check if the model is downloaded and to download it if it isn't. Then `Gpt2Handler.load_model()` is used to load the model into the TensorFlow session by using `gpt_2_simple.load_gpt2()`.

`Gpt2Handler.generate()` takes a string representing the title and optionally a string representing the initial content, an integer representing the number of samples to generate, and an integer representing the maximum number of words to generate for each sample.

First all newline characters are removed from the initial content, then the title and initial content are used to create a string in the proper format for the finetuned GPT-2 model. This format starts with a start of text token followed by a title token on the next line, the title of the line after, a content token on the following line, and finally the initial content on the last line. This string is used as the `prefix` when generate samples.

For example for the title `'This is an example title'` and initial content `'This is an example initial content'`, the string created will be `'<|startoftext|>\n=====TITLE=====\nThis is an example title\n=====CONTENT=====\nThis is an example initial content'` where `'\n'` represents a newline character.

`Gpt2Handler.parse_generate_arguments()` is then used to convert the values in `DEFAULT_CONFIG` to the correct format. `gpt_2_simple.generate()` is then called with the TensorFlow session, the prefix, the number of samples, the maximum number of words per sample, and the parsed args after being unpacked with the `**` operator. The list of samples generated by the model are then returned by this method.

`Gpt2Handler.generate_as_tuple()` takes the same values as `Gpt2Handler.generate()` and calls `Gpt2Handler.generate()` with them. But then uses `Gpt2Handler.sample_to_tuple()` on each of the generated samples to turn them into a list where the first value is the title and the second value is the content.

`Gpt2Handler.sample_to_tuple()` takes a single sample and takes only the text after the title token. Then any tokens that start with `<|` and end with `|>` are removed before replace all instances of multiple spaces with a single space. After this the content token is split upon, leaving a list where the first value is the title and the second value is the content.

5.2 Discord Bot

This program is launched by running `bot.py`. It is designed to be solely interacted with via Discord.

5.2.1 bot.py

When running this file it first reads the apikey from a file called `'aipkey.txt'`, where if none is found then the program quits. If one was found a `discord.ext.commands.Bot` object is created with the `command_prefix` set to `';;'`. Any discord message must begin with `';;'` followed by the command name (or an alias) to be processed as a command. After this every file that ends with `'.py'` in the cogs folder is loaded into the bot. Once this is complete, the bot is ran with the api key using `client.run()`, passing the api key to it.

5.2.2 cogs/setup.py

This cog is responsible for anything related to the setup of the bot. Currently is only prints `'Bot is ready'` to the console once the bot is ready to receive commands. However this functionality was separated from the other cogs to allow for better expandability in future and ensure that each cog adheres to the single responsibility design principle.

5.2.3 cogs/basiccommands.py

This cog is responsible for basic commands that do not have functional significance to the operation of the bot. Currently it only contains a command called `ping` that when used sends a message containing the network latency of the bot. While this can be used for debugging purposes, it is not a crucial function of the bot. However this functionality was separated from the other cogs to allow for better expandability in future and ensure that each cog adheres to the single responsibility design principle.

5.2.4 cogs/utilities.py

This cog is responsible for commands related to core functional elements of the bot such as unloading/loading/reloading other cogs, updating the bot from the git repository, and shutting the bot down. These commands are called `unload`, `load`, `reload`, `update`, and `stop` respectively. None of them take any arguments.

5.2.5 cogs/gpt2.py

This cog is responsible for the functionality of the bot that is the focus of this project, that is interacting with GPT-2 models. When loaded the constructor runs which loads the configuration from a file called `gpt2.config`, if it doesn't exist the default configuration is used instead. After this the TensorFlow session is started using `gpt_2_simple.start_tf_sess()` and then the model is loaded into the TensorFlow session using `gpt_2_simple.load_gpt2()`.

This file contains multiple constants including a list of valid model names, the default config (a dictionary from config names to their respective default values), and a dictionary from configuration names to lambdas which parse the respective config value.

This cog provides multiple commands to alter the configuration, alongside a command to download a fresh GPT-2 model and a command to generate a sample. This last command is called `;;gpt2_generate` with aliases of `'generate'` and `'gpt2'`, the latter of which was chosen as this command is the focus of this cog and thus the command that will most often be used.

The Generate Command

The generate command is executed by the user sending a message that starts with `;;generate`, followed by the prompt (if any). First a check is made to ensure the model set in the configuration is downloaded by using the `gpt_2_simple.is_gpt2_downloaded()` method. If it isn't downloaded an error message is sent saying this, otherwise the function continues with generating the sample. A message is then sent in response to notify the user that a sample is being generated. Then the configuration is parsed before being passed to `gpt_2_simple.generate()` with the TensorFlow session and the prompt specified by the user. This prompt is used as the prefix to start the generated text. Once this text has been generated, the generated sample is sent as a Discord message.

Auxiliary Commands

This cog also has commands that facilitate generating samples, including commands that download fresh GPT-2 models and commands that modify the configuration. The `;;gpt2_download_model` command with an alias of `'download_model'` downloads the fresh GPT-2 model specified by the configuration. If an argument is passed in the message that triggered this command, that argument is used as the name of the model instead. If this name is not a valid fresh GPT-2 model instead an error message will be sent saying this. If the model is successfully downloaded, a confirmation message is said to notify the user of this.

Configuration Commands

There are 5 commands that modify the configuration. These include commands dedicated to changing a specific value in the config, to commands that can modify the whole configuration. Some of the following commands are:

- `;;gpt2_reset_config`: This command has an alias of `'reset_config'`. If an argument is passed in the message that triggered this command an error message is sent saying this. Otherwise the configuration is set to a copy of `DEFAULT_CONFIG`, and the content of the new configuration is written to `gpt2.config`.
- `;;gpt2_set_config`: This command has aliases of `'set_config'` and `'config'`. If no argument is passed in the message that triggered this command an error message is sent saying this. Otherwise the configuration is set based on the arguments provided

by the user. Each argument is seperated by a space and is expected to be in the format `config=value`, where `config` is the name of a configuration and `value` is the corresponding value. If an invalid `config` or `value` is provided by the user, an error message is sent saying this and no changes are made to the configuration. Only once all checks pass does the configuration get updated.

- `;;gpt2_set_length`: This command has an alias of `'set_length'`. If no argument is passed in the message that triggered this command an error message is sent saying this. If the argument is not an integer or not between 1 and 1023 (inclusive) a different error message is sent saying this. If however these checks have been passed, the `'length'` in config is set to the value given by the argument. This sets the number of words generated samples will be.

- `;;gpt2_set_model`: This command has an alias of `'set_model'`. If no argument is passed in the message that triggered this command an error message is sent saying this. If the argument is not the name of one of the four fresh models a different error message is sent saying this. If however these checks have been passed, the `'model_name'` in config is set to the value given by the argument. This sets the fresh model samples will be generated using.

Chapter 6

Evaluation

6.1 Finetuning the Article Generator Model

When finetuning the model used in the article generator, a Google Colaboratory notebook was utilised which "uses either a Nvidia T4 GPU or an Nvidia K80 GPU" [32]. This notebook is adapted from a publicly available notebook called "Train a GPT-2 Text-Generating Model w/ GPU For Free" [32]. The use of such Google Colaboratory notebooks allows for access to high performance hardware, albeit with time restrictions on how long a session can run for before being reset. "Colab does not publish these limits, in part because they can (and sometimes do) vary quickly" [33].

Before the dataset was finetuned on it was pre-processed to ensure that it was in a plain text format that could be easily replicated by the GPT-2 models. Each article was a string in the following format: a start of text token and a new line character, followed by a title token and a new line character, followed by the title of the article and a new line character, followed by the content token and a new line character, followed by the content of the article and a new line character, followed by an end of text token. Where each of these tokens are as follows:

- Start of text token: `'<|startoftext|>'`
- Title token: `'=====TITLE====='`
- Content token: `'=====CONTENT====='`
- End of text token: `'<|end of text|>'`

Initially all of these were written to a single text file that was to be used for finetuning, however this resulted in an out of memory error when attempting to finetune the 124M model on this dataset using the Google Colaboratory notebook. For this reason the dataset was split into multiple text files, with a new file being created every time a file exceeded 52MB. This soft limit was selected because it alleviated the out of memory error and further increasing it resulted in the error still occurring.

Splitting the text files in this way resulted in 12 files. The process of finetuning the model on all of these files involved first finetuning the fresh 124M model on the first file, then finetuning this model on the next file. This was repeated until all files had been finetuned on. Each time the model was finetuned on a single file the finetuning was for 1000 steps with a learning rate of 1e-4.

6.2 Limitations

Due to the size of these models there can a significant delay when generating samples and articles. This is the case for even the smallest fresh model (124M) but the problem is amplified for the larger model sizes. This can be reduced by ensuring the version of TensorFlow with GPU support is installed on a machine with a GPU that meets the hardware requirements, but the delay will still remain noticeable using modern hardware.

Knowledge distillation "is a compression technique in which a compact model - the student - is trained to reproduce the behaviour of a larger model - the teacher - or an ensemble of models" [34]. This technique has been "applied to distill GPT-2" [35] to produce a model with a size that is "37% less, and is twice as fast as its OpenAI counterpart, while keeping the same generative power" [35]. This model also reduces the risk of encountering an out of memory error due to having insufficient RAM (Random Access Memory) as it even "runs smoothly on an iPhone 7" [35].

In the case for the Discord Bot this model would not have to be finetuned and only needs minor code changes to accommodate for this model as the program expects models to be one of the four fresh models, or a finetuned version of them. For the article generator the model would have to be finetuned such that it produces samples in the expected format, but once this is done it can be used in place of model that is currently used.

Due to the way the model generates text it may start writing a new sample/article before the specified word limit is reached. In this case the text generated for the new sample/article can be truncated, but this results in a smaller output without shortening the time taken to generate the sample.

The small size of the model used in the article generator can lead to problems such as repeated text, and the generated text not making logical sense despite making grammatical sense. These problems can be reduced by using a larger model (after finetuning) but have a heavy resource cost. The gpt-2-simple Python library also comments “Currently, a modern single GPU cannot finetune the 774M GPT-2 model or larger” [36]. But software optimisations may allow for finetuning the 774M model [37].

6.3 Own Research Contributions

When finetuning the fresh models on a dataset the samples generated by the models quickly adhere to the format of the dataset, even when this format may deviate heavily from regular natural language. This will be showcased in the finetuning of the 124M fresh model on 3 datasets with different formats. For each dataset a new fresh model was used instead of using the model finetuned on the previous dataset.

The fresh model generates samples of varying formats common in internet text, with all of these samples being natural language. These samples also include features common in internet text that might not necessarily be natural language, such as links. The only exception to this is the end of text token (`'<|endoftext|>'`) which is used to represent the end of a piece of text in a sample. This is generated as a side effect of the finetuning process where the end of text token is inserted between separate pieces of text.

The first dataset is the dataset of news articles used to finetune the article generator model containing approximately 7,500 articles. The format of this dataset is described in the previous section. When finetuning the model generates a single sample every 200 steps, with the first sample being generated after step 200. After step 400 the second sample is generated, and it was the first sample to include the tokens used in this format. These token were also used in the same order as those in the dataset. Further samples generated after more steps also had this format, including a sample in which there were 2 separate articles as 2 separate sets of tokens were generated in the same sample.

The second dataset is a dataset of cocktail recipes containing approximately 1,000 recipes. Each cocktail recipe is formatted so that the name of the recipe is in the first line followed by a colon and ingredients in the lines below. Each ingredient is on its own line with the name of the ingredient followed by a hyphen and then the quantity of the ingredient. Each of these recipes is separated by an end of text token, as those used in the dataset for the article generator model. When finetuning the second sample which was printed after 400 steps generated recipes of this format.

The third dataset is an exported WhatsApp chat log containing approximately 30,000 messages. This is from a WhatsApp group chat containing 9 members. Each message is in the following format: `DD/MM/YYYY, HH:MM - NAME: MESSAGE`. Where `DD/MM/YYYY` is the date the message was sent, `HH:MM` is the time the message was sent, `NAME` is the first name of the author of the message, and `MESSAGE` is the content of the message. These messages can also include emoji characters, which are characters that did not appear in the original dataset the fresh models were finetuned on. Instead of text WhatsApp messages can be media such as images, videos, audio, and more, such messages have their message content replaced with `'<Media omitted>'`.

The first sample generated when finetuning (after 200 steps) exclusively adhered to the format of messages in the dataset. The date and time of each message generated was the same as or after that of the previous message, with multiple messages often being generated with the exact same time before a message was generated in the next minute. There were also instances of messages with a time a few minutes after the previous message's which were less common, and messages with a time that is hours after the last message's which were even less common. This accurately parallels the time between messages being sent from the chat log.

The names in the generated messages were exclusively names of members in the group chat. With the differing messaging styles of different members seeming to be reflected in the generated messages. One example of this is slang terms, certain vocabulary, and use of emoji in the messages of certain members but not in others, accurately reflecting the words/emojis used in messages from some members that aren't used by others. Another example is media being sent by some members but rarely by others, with this difference in the frequency of media being sent is also accurately reflected in the generated messages, where members that send emojis more frequently have messages generated under their name use emojis more frequently.

One specific example is messaging expressing laughter often being sent after messages of `'<Media omitted>'`. This reflects the commonality of humorous images/videos sent in the WhatsApp chat such as internet memes. Such messages are followed by multiple other members sending messages such as `'lol'`, `'lmao'`, and laughter emojis. Another specific example is the use of nicknames in messages referring to them, including a member's name being shortened and a member's full name being used by another member.

This highlights the ability of even the smallest GPT-2 model being able to generate formats that involve features that heavily deviate from natural language, while retaining the ability to generate natural language where it makes sense. GPT-2 is also able to generate natural language in the style of multiple different authors after being finetuned on text that includes text written by all of them, with these differing writing styles being used in a single sample while still making sense in context.

Originally the 355M model was finetuned on this dataset for 85,000 steps. However this resulted in the model overfitting the dataset and returning completions that were essentially recited from the dataset. The 124M was then used instead to reduce this problem of overfitting. While this helped reduce the problem, overfitting eventually still occurred. One reason for this could be the dataset being too small. Despite the WhatsApp group chat containing more than the 30,000 in the exported dataset, when using the built-in exporting feature to export the chat to a text file, it would only include the 30,000 most recent messages. This led to difficulty in attempting to maximise the quality of generated samples by finetuning for more steps, as the whole chat could not be exported to provide the larger dataset needed.

Chapter 7

Legal, Social, Ethical and Professional Issues

7.1 The Black Box Problem

Many machine learning models can be described as black boxes. “Black-box models take a sequence of query inputs, and return corresponding outputs, while keeping internal states such as model architecture hidden” [38]. This can create obscurity in why any complex machine learning model has given a certain output.

Due to the complex structure of neural networks, it can be difficult to derive meaning from the weights and biases that make it up. This problem is exacerbated in the case of deep neural networks as they have a much larger number of parameters and can have more complex architectures that further frustrate how to conceptually understand how they work.

The very large size of GPT-2 models has contributed to the high quality of the text they generate after learning from the large dataset they were trained on. The large number of parameters has resulted in a lack of precise understanding to why it’s architecture and parameters are conducive to it generating high quality samples, as even the smallest GPT-2 model has 124 million parameters.

In the European Union’s (EU) General Data Protection Regulation (GDPR) recital 71 states “[the data subject should have] the right [...] to obtain an explanation of the decision reached” [39]. This means that even in the case of an artificial intelligence system reaching a conclusion based on a user’s data, the responsible organisation must be able to explain why this decision was reached. However if key components of the system that came to this decision are black box system, not even the organisation would know why this decision has been reached.

As machine learning models and artificial intelligence perform better at increasingly complex tasks, reliance on them can also increase. It is for this reason that there is also an increasing importance to understanding why these systems produce the results they do, in order to increase trust in them and to comply with modern data protection laws.

Aiming to decrease the size of models while maintaining the quality of samples they generate can help reduce this problem. However this alone does not solve the problem of the obscurity complex artificial intelligence systems have.

7.2 Fake News and Propaganda

The article generator model was finetuned on a dataset based on the All The News dataset. While this dataset contained over 200,000 articles from various publications. Over 20,000 of these articles were published by Breitbart, which is the highest number of articles of any single publication in the dataset as shown in Figure 7.1.

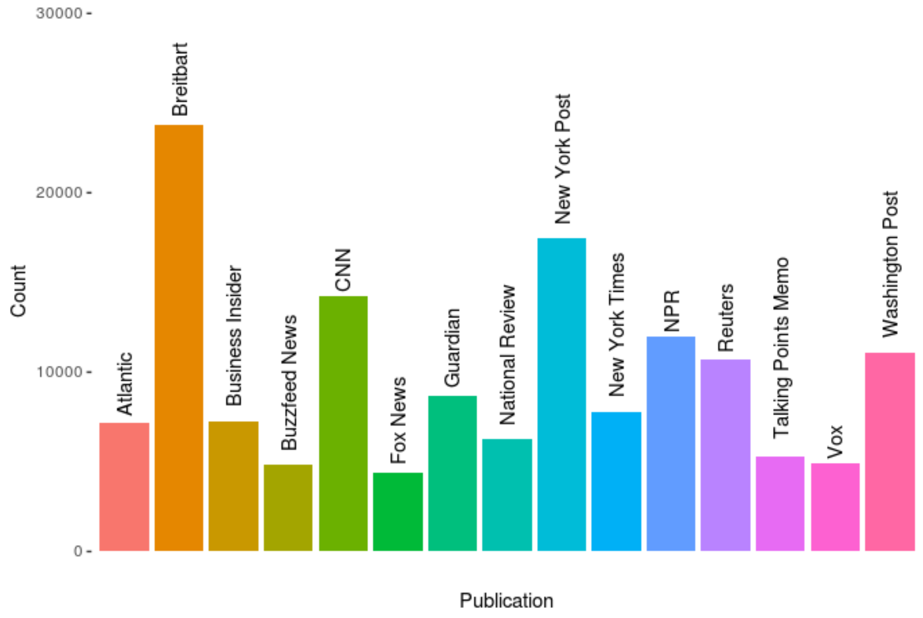


Figure 7.1: Number of articles in All The News by publication[25]

Breitbart is said to be a "far-right website" [40] that has published "material that has been called misogynist, xenophobic and racist" [41] and be a "source of controversy - for liberals and even many traditional conservatives" [41] alongside "viral conspiracy theories" [42].

In a report written by the creators of GPT-2 - OpenAI - it is noted that there is "concern that [GPT-2's] capabilities could lower costs of disinformation campaigns" [3]. While monitoring performed with OpenAI "did not find evidence of GPT-2 direct misuse in publicly-accessible forums" [3], this does not exclude the possibility of such AI systems being misused, as OpenAI "did see evidence of discussion of misuse" [3]. The four ideological positions tested were "white supremacy, Marxism, jihadist Islamism, and anarchism" [43] and it was "demonstrated that it's possible to create models that can generate synthetic propaganda for these ideologies" [43].

One real world example of bias being introduced into AI systems "is Microsoft's "Tay" chatbot, a Twitter bot that replied based on interactions with Twitter users. Internet trolls Tweeted intentionally offensive phrases at Tay, effectively poisoning its dataset [...], resulting in offensive Tweets." [3]

Because of this it was decided that articles published by Breitbart in the All The News dataset would be excluded when generating and formatting the dataset used to finetune the GPT-2 model on. The ideologically extreme sentiment seen from Breitbart has the potential to "poison" the dataset the model is finetuned on, which could introduce this undesirable bias to the model. Removing these articles from the dataset should reduce this bias appearing in articles generated by the model.

However this does not exclude the possibility of GPT-2 models being used for misuse. GPT-2 language models can be used "to generate conditional synthetic text samples of unprecedented quality" [26], including "tasks like question answering, reading comprehension, summarization, and translation" [26]. Finetuning "offers the potential for even more detailed control over generated samples for example, [if finetuned] on the Amazon Reviews dataset [GPT-2 be used to] write reviews conditioned on things like star rating and category." [26] Such a model would allow for easier manipulation of product reviews.

A tool that utilises GPT-2 models and increases their ease of use could make it easier for bad actors to misuse the models. But he range of ways the fictional article generator can be used in misuse has been restricted as its model has been finetuned on a specific dataset of news articles. However it can potentially be used in producing misleading and/or ideological news articles. The risk of this has been reduced by using the 124M model which is smallest available fresh GPT-2 model provided by OpenAI, as OpenAI states "the misuse risk of [the 355M model] is higher than that of [the 124M model]"[26].

Informal testing has shown that it is difficult to introduce specific false information in the generated article beyond the optional human written content to start the article. This further restricts the ability of bad actors using using this tool to generate news articles in order to mislead and spread disinformation.

However this does not eliminate the possibility of a skilled bad actor creating a dataset of ideologically biased or propagandist news articles, and formatting such a dataset to the format used by the article generator tool, then finetuning a larger GPT-2 model on this dataset. This model can then replace the model used in the article generator, allowing it to produce malicious fictional news. However such an attack is unlikely as such a bad actor with the sufficient knowledge would not gain a proportional benefit to training a specific model to be used in this tool.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

This project has aimed to create 2 programs that utilise GPT-2 and allow users to generate text using GPT-2 without technical knowledge in natural language processing or machine learning. This involved designing and implementing a program that uses a finetuned model, alongside a program that can allow users to use any GPT-2 model (fresh or finetuned) via Discord.

This process involved a GPT-2 model being finetuned so that samples produced are in a specified format that could be properly parsed by the program. Despite GPT-2 specialising is natural language generation, it was possible to finetune a fresh model such that generated samples reliably adhere to a desired format.

The existence of libraries that facilitate the use of GPT-2 allows for much more elegant interaction with GPT-2 models. This includes automating downloading models, finetuning models, and using models to generate text as part of a software system.

The propensity for GPT-2 models to learn formats that deviate from natural language shows their capacity to provide utility in applications beyond pure natural language processing. This is especially useful in use cases in which natural language processing is required alongside textual structures that are more formal.

The Discord Bot allows for a very accessible way of using GPT-2 as it allows the user to interact with GPT-2 via Discord. Discord has clients for Windows, Mac, Linux, Android, iOS, and can even be used via a web browser. This means that most users would be able to use the Discord Bot as long as they have an internet connection. The Discord Bot also only needs a single Administrator to set it up then as long as it is running, users can interact with it via any

Discord server it has been added to.

Throughout the duration of this project it has been made clear that language models including GPT-2 are very powerful and have the potential to be utilised to do a lot more than just generating plain text. As more powerful language models are developed this potential will grow, increasing the novel ways in which language models can be used.

8.2 Future Work

While the article generator is capable of generating high quality text, this can be improved by starting with a larger fresh model when finetuning. This program could use the 1558M model as this is the largest model and the original size of GPT-2, or even use larger and more powerful models developed in future. However this could require constraints due to modern hardware to be alleviated via software and/or hardware optimisations. Once such a model has been finetuned, using it to generate text is comparatively much computationally easier but this will also come with performance costs compared to smaller models.

This tool can also be improved by improving the efficiency of the model by using a distilled version of GPT-2 such as the DistilGPT-2 model created by HuggingFace, or even are comparable small language model. The time taken to generate articles could be substantially reduced due to the smaller size of such models, allowing for much better performance. This improved performance could even increase the viability of generating such content directly on smart phones.

The article generator tool could also be built upon to create a web app. This would substantially reduce the hardware and software requirements, as it would allow the tool to be accessed via a web browser. This could also require all processing to be done server side when generating text, the impact of which can be reduced by using a model that uses reduced computational resources.

While the Discord Bot does have functionality that allows it to use any finetuned GPT-2 model the administrator wishes, this aspect of the bot could be further developed to improve the usability of this feature and streamline the process of adding these models to the bot e.g. allowing the bot to download models via a provided link. This could even allow for users to add models to an instance of the bot they have access to.

References

[1] R. Bellman, "Dynamic programming," *Science*, vol. 153, no. 3731, pp. 34–37, 1966.

[2] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI Blog*, vol. 1, no. 8, 2019.

[3] I. Solaiman, M. Brundage, J. Clark, A. Askell, A. Herbert-Voss, J. Wu, A. Radford, G. Krueger, J. W. Kim, S. Kreps, M. McCain, A. Newhouse, J. Blazakis, K. McGuffie, and J. Wang, "Release strategies and the social impacts of language models," 2019.

[4] Minimaxir, "gpt-2-simple." `https://github.com/minimaxir/gpt-2-simple/commit/a4da3ff0f054523d8cf85d98ae42cdd2d8dc7007`, 2019.

[5] Discord, "Discord jobs and company information." `https://discordapp.com/company`, Mar 2020.

[6] K. Andrej, "The unreasonable effectiveness of recurrent neural networks." `https://karpathy.github.io/2015/05/21/rnn-effectiveness/`, May 2015.

[7] J. Alammar, "Visualizing a neural machine translation model (mechanics of seq2seq models with attention)." `https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/`, May 2018.

[8] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," 2014.

[9] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," 2014.

[10] D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” 2014.

[11] M.-T. Luong, H. Pham, and C. D. Manning, “Effective approaches to attention-based neural machine translation,” 2015.

[12] J. Alammar, “The illustrated transformer.” `http://jalammar.github.io/illustrated-transformer/`, Jun 2018.

[13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” 2017.

[14] Minimaxir, “minimaxir/gpt-2-simple/readme.md.” `https://github.com/minimaxir/gpt-2-simple/blob/3018ced23fc0494d59f74ac2ac7b0bf0be95bb07/README.md`, Jan 2020.

[15] TensorFlow, “tf.random.set_random_seed: Tensorflow core r1.15.” `https://www.tensorflow.org/versions/r1.15/api_docs/python/tf/compat/v1/set_random_seed`, Jan 2020.

[16] Openai, “openai/gpt-2/interactive_conditional_samples.py.” `https://github.com/openai/gpt-2/blob/0574c5708b094bfa0b0f6dfe3fd284d9a045acd9/src/interactive_conditional_samples.py`, Jan 2020.

[17] bybaes20, “I made my ai write my english paper.” `https://cymetric1.wordpress.com/2019/10/22/fun-with-gpt-2-i-made-my-ai-to-write-my-english-paper/`, Nov 2019.

[18] TensorFlow, “tf.train.adamoptimizer: Tensorflow core r1.15.” `https://www.tensorflow.org/versions/r1.15/api_docs/python/tf/compat/v1/train/AdamOptimizer`, Mar 2020.

[19] TensorFlow, “tf.train.gradientdescentoptimizer: Tensorflow core r1.15.” `https://www.tensorflow.org/versions/r1.15/api_docs/python/tf/train/GradientDescentOptimizer`, Mar 2020.

[20] R. Haleva, “What is gradient accumulation in deep learning?.” `https://towardsdatascience.com/what-is-gradient-accumulation-in-deep-learning-ec034122cfa`, Jan 2020.

[21] T. Chen, B. Xu, C. Zhang, and C. Guestrin, "Training deep nets with sublinear memory cost," 2016.

[22] Minimaxir, "minimaxir/gpt-2-simple/memory_saving_gradients.py." https://github.com/minimaxir/gpt-2-simple/blob/af3bca4a48ea8fec5cbae040155f3a495a2df387/gpt_2_simple/src/memory_saving_gradients.py, Jul 2019.

[23] Rapptz, "Rapptz/discord.py." https://github.com/Rapptz/discord.py/tree/ade8d03f546c915d8015439da8aa05ffe18a6184, Nov 2019.

[24] Rapptz, "Commands." https://discordpy.readthedocs.io/en/v1.2.5/ext/commands/commands.html, Nov 2019.

[25] A. Thompson, "All the news." https://www.kaggle.com/snapcrack/all-the-news/version/4, Aug 2017.

[26] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Better language models and their implications." https://openai.com/blog/better-language-models/, Dec 2019.

[27] A. King, "Talk to transformer." https://talktotransformer.com/, Jul 2019.

[28] HuggingFace, "Write with transformer." https://transformer.huggingface.co/.

[29] TensorFlow, "Gpu support: Tensorflow." https://www.tensorflow.org/install/gpu, Mar 2020.

[30] Rapptz, "Cogs." https://discordpy.readthedocs.io/en/v1.2.5/ext/commands/cogs.html, Nov 2019.

[31] Rapptz, "Creating a bot account." https://discordpy.readthedocs.io/en/v1.2.5/discord.html, Nov 2019.

[32] Minimaxir, "Train a gpt-2 text-generating model w/ gpu for free." https://colab.research.google.com/drive/1VLG8e7YSEwypxU-noRNhsv5dW4NfTGce, Nov 2019.

[33] Google, "Colaboratory." https://research.google.com/colaboratory/faq.html, Apr 2020.

[34] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," 2019.

[35] HuggingFace, “Distilgpt-2 model checkpoint.” https://transformer.huggingface.co/model/distil-gpt2, Apr 2020.

[36] Minimaxir, “minimaxir/gpt-2-simple.” https://github.com/minimaxir/gpt-2-simple/blob/master/gpt_2_simple/gpt_2.py, Feb 2020.

[37] Minimaxir, “Oom error with new 774m model when running in colab · issue #108 · minimaxir/gpt-2-simple.” https://github.com/minimaxir/gpt-2-simple/issues/108, Dec 2019.

[38] S. J. Oh, M. Augustin, B. Schiele, and M. Fritz, “Towards reverse-engineering black-box neural networks,” 2017.

[39] “Lex access to european union law.” https://eur-lex.europa.eu/eli/reg/2016/679/oj, Apr 2016.

[40] D. Weigel, “Is trump’s new chief strategist a racist? critics say so..” https://www.washingtonpost.com/politics/is-trumps-new-chief-strategist-a-racist-critics-say-so/2016/11/14/b72e2ab0-aa9d-11e6-a31b-4b6397e625d0_story.html, Nov 2016.

[41] M. Grynbaum and J. Herrman, “Breitbart rises from outlier to potent voice in campaign.” https://www.nytimes.com/2016/08/27/business/media/breitbart-news-presidential-race.html, Aug 2016.

[42] L. Robertson, “Trump’s isis conspiracy theory.” https://www.factcheck.org/2016/06/trumps-isis-conspiracy-theory/, Nov 2016.

[43] I. Solaiman, J. Clark, and M. Brundage, “Gpt-2: 1.5b release.” https://openai.com/blog/gpt-2-1-5b-release/, Nov 2019.

Appendix A

Extra Information

List of Figures

Appendix B

User Guide

B.1 GPT2 Article Generator

An application to allow for generating news articles using OpenAI[1]'s GPT-2 text generator[2]. The model used for this was further trained on All The News[3], a dataset of over 200,000 news articles by components.one[4].

B.1.1 Setup

The repository can be cloned as normal:

```
shell git clone https://github.com/DanTm99/gpt2-article-generator.git
```

The model this program uses is hosted on Google Drive and can be downloaded from here[5]. The contents of this archive should be extracted to the `gpt2-article-generator` folder so that the `checkpoint` is in the `gpt2-article-generator` folder.

Navigate into the folder:

```
shell cd gpt2-article-generator
```

To use this with your GPU you must have and NVIDIA GPU with a CUDA Compute Capability 3.5 or higher.

If you have the required hardware you must install the required software on your system as shown here[6].

Install the required packages as normal to use this with GPU support:

[1]<https://openai.com>
[2]<https://openai.com/blog/better-language-models/>
[3]<https://www.kaggle.com/snapcrack/all-the-news>
[4]<https://components.one/>
[5]<https://drive.google.com/open?id=1Lmh7JBRkbC0jEvGtoZwVL3OPT8PIt9qm>
[6]<https://www.tensorflow.org/install/gpu#software_requirements>

```
shell pip3 install -r requirements.txt
```

To use this without GPU support use the following command instead:

```
shell pip3 install -r requirements-no-gpu.txt
```

B.1.2 Usage

To open the GUI use the following command:

```
shell python3 ArticleGenerator.py
```

This application can also be used via the command line. For detailed help use the following command:

```
shell python3 ArticleGenerator.py -h
```

B.2 GPT2 Bot

A simple discord bot written in Python that utilises existing Python libraries to allow for simple interaction with OpenAI[7]'s GPT-2 text generator[8].

B.2.1 Setup

Clone the repository and navigate into it:

`shell git clone https://github.com/DanTm99/gpt2-bot.git cd gpt2-bot`

To use this with your GPU you must have and NVIDIA GPU with a CUDA Compute Capability 3.5 or higher.

If you have the required hardware you must install the required software on your system as shown here[9].

Install the required packages as normal to use this with GPU support:

`shell pip3 install -r requirements.txt`

To use this without GPU support use the following command instead:

`shell pip3 install -r requirements-no-gpu.txt`

Create `apikey.txt` containing the api key for your bot:

`shell echo "[API_KEY]" > apikey.txt` Replace `[API_KEY]` with your api key.

B.2.2 Usage

Run `bot.py` to start the bot:

`shell python3 bot.py`

This bot responds to commands sent to any Discord server it's a part of.

By default messages must start with `;;` to be recognised as a command. This can be changed by changing `COMMAND_PREFIX` in `bot.py`.

`;;download_model` downloads the GPT-2 model and must be used to generate text.

`;;generate [prompt]` generates text that starts with an optional prompt.

[7]<https://openai.com>
[8]<https://openai.com/blog/better-language-models/>
[9]<https://www.tensorflow.org/install/gpu#software_requirements>

Appendix C

Source Code

I verify that I am the sole author of the programs contained in this folder, except where explicitly stated to the contrary.

Danyaal Khan

April 21, 2020

C.1 Discord Bot

C.1.1 bot.py

```python
import os

from discord.ext import commands

API_KEY_FILENAME = 'apikey.txt'
COMMAND_PREFIX = ';;'
api_key = None

# Read API key from file
if not api_key:
    try:
        with open(API_KEY_FILENAME, 'r') as file:
            api_key = file.readline().rstrip()
    except FileNotFoundError:
        print(f'ERROR: API key file {API_KEY_FILENAME} not found')
        exit(0)

client = commands.Bot(command_prefix=COMMAND_PREFIX)


# Load all .py files in cog folder
for filename in os.listdir('./cogs'):
    if filename.endswith('.py'):
        client.load_extension(f'cogs.{filename[:-3]}')
        print(f'Loaded {filename[:-3]}')

client.run(api_key)
```

C.1.2 cogs/setup.py

```
import discord
from discord.ext import commands


class Setup(commands.Cog):

    def __init__(self, client):
        self.client = client

    @commands.Cog.listener()
    async def on_ready(self):
        print('Bot is ready')


def setup(client):
    client.add_cog(Setup(client))
```

C.1.3 cogs/basiccommands.py

```
import discord
from discord.ext import commands


class BasicCommands(commands.Cog):

    def __init__(self, client):
        self.client = client

    @commands.command()
    async def ping(self, ctx, *, arg=None):
        print('Command ping triggered')
        if arg:
            await ctx.send("ERROR: Argument not allowed")
        else:
            await ctx.send(f'Pong! {round(self.client.latency * 1000)}ms')


def setup(client):
    client.add_cog(BasicCommands(client))
```

C.1.4 cogs/utilities.py

```python
import subprocess

from discord.ext import commands


class Utilities(commands.Cog):

    def __init__(self, client):
        self.client = client

    @commands.command()
    async def load(self, ctx, extension):
        self.client.load_extension(f'cogs.{extension}')
        print(f'Loaded {extension}')
        await ctx.send(f'Loaded {extension}')

    @commands.command()
    async def unload(self, ctx, extension):
        self.client.unload_extension(f'cogs.{extension}')
        print(f'Unloaded {extension}')
        await ctx.send(f'Unloaded {extension}')

    @commands.command()
    async def reload(self, ctx, extension):
        self.client.unload_extension(f'cogs.{extension}')
        print(f'Unloaded {extension}')
        self.client.load_extension(f'cogs.{extension}')
        print(f'Loaded {extension}')
        await ctx.send(f'Reloaded {extension}')

    @commands.command()
    async def update(self, ctx, *, arg=None):
        print('Command update triggered')
        await ctx.send('Updating and shutting down...')
        subprocess.call('git pull')
        await ctx.bot.logout()

    @commands.command()
    async def stop(self, ctx, *, arg=None):
        print('Command stop triggered')
        await ctx.send('Shutting down...')
        await ctx.bot.logout()


def setup(client):
    client.add_cog(Utilities(client))
```

C.1.5 cogs/gpt2.py

```python
import logging
import os

from discord.ext import commands
```

```

# Source: https://github.com/tensorflow/tensorflow/issues/27023#issuecomment-475544248
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'  # Used to disable TensorFlow printing debug messages
# Source: https://github.com/tensorflow/tensorflow/issues/8340#issuecomment-332212742
logging.getLogger('tensorflow').disabled = True  # Used to disable TensorFlow printing warning messages

import gpt_2_simple as gpt2

CONFIG_PATH = 'gpt2.config'
DEFAULT_PROMPTS_PATH = 'default_prompts.txt'
VALID_DEFAULT_MODELS = ['124M', '355M', '774M', '1558M']
DEFAULT_CONFIG = {
    'model_name': '124M',
    'length': '100',
    'temperature': '0.7',
    'top_k': '0',  # How many previous words to consider when generating a new word. 0 means unlimited
    'top_p': '0.9',
    'include_prefix': 'True'
}
# A dictionary from argument names to a lambda that determines how to parse the string representing its value
CONFIG_KEY_PARSER = {
    'model_name': lambda s: s,
    'length': lambda i: int(i),
    'temperature': lambda f: float(f),
    'top_k': lambda i: int(i),
    'top_p': lambda f: float(f),
    'include_prefix': lambda b: b == 'True',
}


def parse_generate_arguments(arguments):
    """
    Convert the config into the correct format for arguments for the generate function
    The conversion for each argument is determined by the argument's name and CONFIG_KEY_PARSER
    :param arguments: A dictionary from an argument name to a string representing the value for that argument
    :return: A dictionary from an argument name to the value to be passed for that argument
    """
    return_value = {}
    for key in arguments:
        return_value[key] = CONFIG_KEY_PARSER[key](arguments[key])

    return return_value


def is_valid_config(config):
    """
    :param config: A string representing the name of a config
    :return: Whether or not the config is recognised by this program
    """
    return config in CONFIG_KEY_PARSER


def is_valid_config_value(config, value):
    """
    :param config: A string representing the name of a config
    :param value: A string representing a value for the config
    :return: Whether or not the the value for a recognised config is in the correct format
    """
```

```python
    try:
        CONFIG_KEY_PARSER[config](value)
    except ValueError or KeyError:
        return False
    return True


def read_default_prompts():
    """
    Read the default prompts from a file with contents in the form model_name=prompt on each line.
    :return: A dictionary with model names as keys and the prompts as values, both being strings.
    """
    if os.path.exists(DEFAULT_PROMPTS_PATH):
        with open(DEFAULT_PROMPTS_PATH, 'r') as file:
            return {key: value for [key, value] in (line.rstrip().split('=') for line in file)}
    else:
        return {}


def write_dictionary(dictionary, path):
    """
    Write the contents of a dictionary to a file with contents in the form key=value on each line.
    The keys and values must be a type that can be converted to a from a string.
    If a file already exists in the filepath it will be overwritten.
    :param dictionary: The dictionary to write to file
    :param path: The path of the file to write to
    """
    with open(path, 'w+') as file:
        file.truncate()  # Erase contents of config file
        for key in dictionary:
            file.write(f'{key}={dictionary[key]}\n')


class Gpt2(commands.Cog):

    def __init__(self, client):
        """
        Read the config from the file at the path CONFIG_PATH, if the config file is not in the expected format or
        contains a config that isn't recognised, the config is loaded from the dictionary DEFAULT_CONFIG.

        Initialise a TensorFlow session for gpt2 then load the GPT2 model as determined by the config.

        NOTE: If the config is modified after the DEFAULT_CONFIG has been loaded it, it will be overwritten.
        """
        self.client = client
        self.config = {}
        self.load_config(False)
        self.default_prompts = read_default_prompts()

        self.sess = gpt2.start_tf_sess()
        try:
            gpt2.load_gpt2(self.sess, model_name=self.config['model_name'])
        except ValueError:
            self.sess = gpt2.reset_session(self.sess)
            gpt2.load_gpt2(self.sess, model_name=self.config['model_name'])

    @commands.command(aliases=['generate', 'gpt2'])
    async def gpt2_generate(self, ctx, *, arg=''):
        """
        Generate a text sample from a given prompt using GPT-2.
        The arguments and their values for the generation is determined by the config.
        :param arg: The prompt to generate the text sample on
        """
        print('Command gpt2_generate triggered')
        if gpt2.is_gpt2_downloaded(model_name=self.config['model_name']):
            generate_args = parse_generate_arguments(self.config)
            await ctx.send("Generating...")
            sample = gpt2.generate(self.sess, prefix=arg, return_as_list=True, **generate_args)[0]
            await ctx.send(sample)
        else:
            await ctx.send(f"ERROR: Model {self.config['model_name']} is not downloaded")

    @commands.command(aliases=['set_model'])
    async def gpt2_set_model(self, ctx, *, arg=None):
        """
        Set the name of the GPT-2 model in the config by setting model_name if it's a valid model name.
        :param arg: The value to set model_name to
        """
        print('Command gpt2_set_model triggered')
        if arg:
            if arg in VALID_DEFAULT_MODELS:
                self.update_config(model_name=arg)
            else:
                await ctx.send(f"ERROR: Invalid model name {arg}")
        else:
            await ctx.send("ERROR: Argument required")

    @commands.command(aliases=['set_length'])
    async def gpt2_set_length(self, ctx, *, arg=None):
        """
        Set the length of the samples produced by GPT-2 when producing samples.
        The value represents the number of tokens (i.e. words) each produced sample will contain.
        :param arg: The value to set length to. This must be a positive integer
        """
        print('Command gpt2_set_length triggered')
        if arg:
            try:
                i = int(arg)
                assert (i > 0) and (i < 1024)
            except ValueError or AssertionError:
                ctx.send("ERROR: Argument must be a positive integer number")
            self.update_config(length=arg)
        else:
            await ctx.send("ERROR: Argument required")

    @commands.command(aliases=['set_config', 'config'])
    async def gpt2_set_config(self, ctx, *, arg=None):
        print('Command gpt2_set_config triggered')
        if arg:
            configs = {key: value for [key, value] in (a.split('=') for a in arg.split(' '))}
            for config in configs:
                if not is_valid_config(config):  # Check if the config name exists
                    await ctx.send(f"ERROR: Invalid config name {config}")
                    return
                elif not is_valid_config_value(config, configs[config]):
                    await ctx.send(f"ERROR: Invalid config {config}={configs[config]}")
                    return
                self.update_config(**configs)
            else:
                await ctx.send("ERROR: Argument required")

    @commands.command(aliases=['download_model'])
    async def gpt2_download_model(self, ctx, *, arg=None):
        print('Command gpt2_download_model triggered')

        if arg:
            if arg in VALID_DEFAULT_MODELS:
                gpt2.download_gpt2(model_name=arg)
                await ctx.send("Model downloaded")
            else:
                await ctx.send("ERROR: Invalid argument")
        else:  # If no model name is provided, download the one in the config
            model_name = self.config['model_name']
            if model_name in VALID_DEFAULT_MODELS:
                gpt2.download_gpt2(model_name=model_name)
            else:
                await ctx.send("ERROR: Invalid model_name in config")

    @commands.command(aliases=['reset_config'])
    async def gpt2_reset_config(self, ctx, *, arg=None):
        print('Command gpt2_reset_config triggered')
        if arg:
            await ctx.send('ERROR: Argument not allowed')
        else:
            self.reset_config()
            await ctx.send('Config reset')

    @commands.command(aliases=['custom'])
    async def gpt2_custom(self, ctx, model_name=None, *, arg=None):
        print('Command gpt2_custom triggered')
        if model_name:
            if not arg:
                if model_name in self.default_prompts:
                    arg = self.default_prompts[model_name]
                else:
                    await ctx.send('ERROR: Prompt required')
                    return
            generate_args = parse_generate_arguments(self.config)
            generate_args['model_name'] = model_name
            generate_args['include_prefix'] = False
            sample = gpt2.generate(self.sess, prefix=arg, return_as_list=True, **generate_args)[0]
            await ctx.send(sample)
        else:
            await ctx.send('ERROR: Argument required')

    @commands.command(aliases=['default_prompt', 'prompt'])
    async def gpt2_set_default_prompt(self, ctx, model_name=None, *, arg=None):
        if model_name:
            if arg:
                self.default_prompts[model_name] = arg
                write_dictionary(self.default_prompts, DEFAULT_PROMPTS_PATH)
            else:
                await ctx.send("ERROR: Default prompt cannot be blank")
        else:
```

```python
            await ctx.send("ERROR: Missing model name")

    def is_model_downloaded(self):
        model_name = self.config['model_name']
        return os.path.exists(f'models/{model_name}')

    def update_config(self, write=True, **kwargs):
        for key in kwargs:
            if is_valid_config(key):
                value = kwargs[key]
                if is_valid_config_value(key, value):
                    self.config[key] = value
                else:
                    print(f'ERROR: Invalid config {key}={value}')
            else:
                print(f'ERROR: Invalid config key {key}')

        if write:
            write_dictionary(self.config, CONFIG_PATH)

    def reset_config(self, write=True):
        self.config = DEFAULT_CONFIG.copy()

        if write:
            write_dictionary(self.config, CONFIG_PATH)

    def load_config(self, write_if_default=True):
        if os.path.exists(CONFIG_PATH):
            with open(CONFIG_PATH, 'r') as file:
                for line in file:
                    key, value = line.rstrip().split('=')
                    if is_valid_config_value(key, value):
                        self.config[key] = value
                    else:
                        self.reset_config(write_if_default)  # Load default config instead
                        return
        else:
            self.reset_config(write_if_default)


def setup(client):
    client.add_cog(Gpt2(client))
```

C.2 Article Generator

C.2.1 ArticleGenerator.py

```python
import argparse
import os


def positive_int_type(value):
    """Raise an error if the value is not a positive integer. Return it otherwise."""
    try:
        i = int(value)
        assert i > 0
    except (ValueError, AssertionError):
        raise argparse.ArgumentTypeError(f'{value} is not a positive int.')
    return i


def bound_positive_int_type(value, max_value):
    """Raise an error if the value is not a positive integer and less than the max_value. Return it otherwise."""
    try:
        i = int(value)
        assert i > 0
    except (ValueError, AssertionError):
        raise argparse.ArgumentTypeError(f'{value} is not a positive int.')

    if i > max_value:
        raise argparse.ArgumentTypeError(f'{value} is greater than than {max_value}.')
    return i


def existing_filename_type(value):
    """Raise an error if the provided value is not the name of an existing file. Return it otherwise."""
    if not os.path.isfile(value):
        raise argparse.ArgumentTypeError(f'{value} is not the name of a file that exists.')
    return value


def not_existing_filename_type(value):
    """Raise an error if the provided value is the name of an existing file. Return it otherwise."""
    if os.path.isfile(value):
        raise argparse.ArgumentTypeError(f'{value} is not the name of a file that does not exists.')
    return value


def is_default_args(namespace):
    """Return true if all of the arguments in the namespace are the default value. Return false otherwise."""
    return (namespace.content is None) and (namespace.content_filename is None) and (namespace.filename is None) and \
           (not namespace.print) and (namespace.num_samples == 1) and (namespace.num_words == 1023) and \
           (namespace.output_filename is None) and (namespace.title is None) and (namespace.title_filename is None)


def create_parser():
    """Create and return a parser with a usage string and all the arguments this program can take."""
    usage_str = """
        USAGE:      python ArticleGenerator.py <options>
                    NOTE: The order of the options does not matter.
        EXAMPLES:   (1) python ArticleGenerator.py
                        - Open the ArticleGenerator GUI
                    (2) python ArticleGenerator.py -f example.txt -o sample.txt -n 3
                    OR  python ArticleGenerator.py --filename example.txt --output-filename sample.txt --num_samples 3
                        - Generate 3 articles with the title and initial content specified in \'example.txt\' and
                        write each of them to \'sample1.txt\', \'sample2.txt\', and \'sample3.txt\' respectively.
                    (3) python ArticleGenerator.py -T 'Example title' -C 'Example content' -p
                    OR  python ArticleGenerator.py -title 'Example title' -content 'Example content' -print
                        - Generate 1 article with the title being 'Example title' and the content being
                        'Example content' and print it to the console.
        """
    parser = argparse.ArgumentParser(usage_str)
    parser.add_argument('-f', '--filename', dest='filename', type=existing_filename_type,
                        help='Use the title and initial content in a file with the filename specified by FILENAME. '
                             'The file should contain the title in the first line, and initial content (if any) in the '
                             'second line.')
    parser.add_argument('-o', '--output_filename', dest='output_filename',
                        type=not_existing_filename_type,
                        help='Write the generated sample to a new file with the filename specified by OUTPUT_FILENAME. '
                             'The file must not already exist. The sample number is appended to the filename before '
                             'the extension if multiple samples are to be generated.')
    parser.add_argument('-p', '--print', dest='print', action='store_true',
                        help='Print the generated sample(s) to console.')
    parser.add_argument('-n', '--num_samples', dest='num_samples', default=1, type=positive_int_type,
                        help='Generate a number of samples equal to NUM_SAMPLES. It must be a positive integer. '
                             'Default: 1')
    parser.add_argument('-w', '--num_words', dest='num_words', default=1023,
                        type=lambda i: bound_positive_int_type(i, 1023),
                        help='Generate samples with a number of words that is not greater than NUM_WORDS. It must be a '
                             'positive integer that does not exceed 1023. Default: 1023')
    parser.add_argument('-t', '--title_filename', dest='title_filename', type=existing_filename_type,
                        help='Use the text in the first line of the file specified by TITLE_FILENAME as the title. '
                             'This will be ignored if a filename for \'--filename\' is specified.')
    parser.add_argument('-c', '--content_filename', dest='content_filename', type=existing_filename_type,
                        help='Use the text in the first line of the file specified by CONTENT_FILENAME as the initial '
                             'content. This will be ignored if no filename for \'--title-filename\' is specified.')
    parser.add_argument('-T', '--title', dest='title',
                        help='Use the text specified by TITLE as the title. This will be ignored if a filename '
                             'for \'--filename\' or \'--title_filename\' is specified.')
    parser.add_argument('-C', '--content', dest='content',
                        help='Use the text specified by CONTENT as the initial content. This will be ignored if '
                             'no title for \'--title\' is specified.')
    return parser


def parse_arguments():
    """Create a parser and use it to parse the arguments given by Python, then return the parsed arguments."""
    parser = create_parser()
    parsed_args = parser.parse_args()
    return parsed_args


if __name__ == '__main__':
    args = parse_arguments()
    from generator import Generator

    gen = Generator.get_instance()

    if is_default_args(args):
        gen.launch_gui()
    elif not args.output_filename and not args.print:
        raise Exception('Output has not been set to either console or an output file.\nUse the -h argument for help.')
    elif args.output_filename and (args.output_filename in [args.filename, args.title_filename, args.content_filename]):
        raise Exception('Output filename cannot be the same as an input filename.\nUse the -h argument for help.')
    elif args.filename:
        gen.generate_from_single_file(args.filename, args.num_samples, args.print, args.output_filename, args.num_words)
    elif args.title_filename:
        gen.generate_from_files(args.title_filename, args.content_filename, args.num_samples, args.print,
                                args.output_filename, args.num_words)
    elif args.title:
        gen.generate(args.title, args.content, args.num_samples, args.print, args.output_filename, args.num_words)
    else:
        raise Exception('Input filename, input title filename, or input title have been set.\n'
                        'Use the -h argument for help.')
```

C.2.2 generator.py

```
import os

from gpt2handler import Gpt2Handler


class Generator:
    """This class respects the singleton design pattern and facilitates communication between the other components."""
    __instance = None

    @classmethod
    def get_instance(cls):
        """Return the instance of this class. If it doesn't exist construct it first."""
        if cls.__instance is None:
            cls()
        return cls.__instance

    def __init__(self):
        """Initialise a Generator instance if there is none. For internal use only."""
        if Generator.__instance is None:
            Generator.__instance = self
        else:
            raise Exception("Attempted initialisation of singleton class Gui.")

        Gpt2Handler.get_instance()  # Create instance of Gpt2Handler

    @staticmethod
    def launch_gui():
        """Get the instance of the GUI and start it."""
        from gui import Gui
        Gui.get_instance().start()

    @staticmethod
    def generate_as_tuple(title, initial_content='', num_samples=1, num_words=1023):
        """Use gpt2 to generate an article as a tuple then return it."""
        return Gpt2Handler.get_instance().generate_as_tuple(title, initial_content, num_samples, num_words)

    def generate_from_single_file(self,
                                  input_filename,
                                  num_samples=1,
                                  print_output=False,
                                  output_file=None,
                                  num_words=1023):
        """Read the title and initial content from a single file then use gpt2 to generate an article
        and return it as a single string."""
        with open(input_filename, 'r', errors='surrogateescape') as f:
            file_contents = f.readlines()

        title = file_contents[0].rstrip()
```

```python
        initial_content = '' if len(file_contents) < 2 else file_contents[1].rstrip()

        return self.generate(title, initial_content, num_samples, print_output, output_file, num_words)

    def generate_from_files(self,
                            title_filename,
                            content_filename=None,
                            num_samples=1,
                            print_output=False,
                            output_file=None,
                            num_words=1023):
        """Read the title from a file and initial content from another file then use gpt2 to generate an article
        and return it as a single string."""
        with open(title_filename, 'r', errors='surrogateescape') as title_file:
            title = title_file.readline().rstrip()

        if content_filename:
            with open(content_filename, 'r', errors='surrogateescape') as content_file:
                initial_content = content_file.readline().rstrip()
        else:
            initial_content = ''

        return self.generate(title, initial_content, num_samples, print_output, output_file, num_words)

    def generate(self,
                 title,
                 initial_content=None,
                 num_samples=1,
                 print_output=False,
                 output_file=None,
                 num_words=1023):
        """Use gpt2 to generate an article based on a given title and initial content."""
        if not initial_content:
            initial_content = ''
        samples = Gpt2Handler.get_instance().generate_as_tuple(title, initial_content, num_samples, num_words)
        samples_str = [sample[0] + '\n' + sample[1] for sample in samples]

        if print_output:  # Print each article to the console is specified to
            for sample in samples_str:
                print(sample)
        if output_file:  # Write each of the samples to their own file if a base filename is specified
            self.write_samples_to_file(output_file, samples_str)

        return samples_str

    def write_samples_to_file(self, filename, samples):
        """Write the given samples to a file. If there is more than one, write each to its own file."""
        if len(samples) == 1:
            self.write_sample_to_file(filename, samples[0])
        else:
            base, extension = os.path.splitext(filename)
            for i in range(len(samples)):
                new_filename = base + str(i) + extension
                self.write_sample_to_file(new_filename, samples[i])

    def write_sample_to_file(self, filename, sample):
        """Write a given sample to a file specified by te filename."""
        with open(filename, 'w+', errors='surrogateescape', encoding='utf-8') as f:
            f.write(sample)
```

C.2.3 gpt2handler.py

```
import logging
import os

# Source: https://github.com/tensorflow/tensorflow/issues/27023#issuecomment-475544248
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'  # Used to disable TensorFlow printing debug messages
# Source: https://github.com/tensorflow/tensorflow/issues/8340#issuecomment-332212742
logging.getLogger('tensorflow').disabled = True  # Used to disable TensorFlow printing warning messages

import gpt_2_simple as gpt2
import re

DEFAULT_CONFIG = {
    'model_name': '124M',
    'run_name': '124M_article_generator_model',  # The name of the model
    'top_k': '0',  # How many previous words to consider when generating a new word. 0 means unlimited
    'include_prefix': 'True',
    'return_as_list': 'True',
    'truncate': '<|endoftext|><|startoftext|>'  # Truncate the sample where it contains this substring
}
# A dictionary from argument names to a lambda that determines how to parse the string representing its value
GENERATE_ARGUMENT_PARSER = {
    'model_name': lambda s: s,
    'run_name': lambda s: s,
    'temperature': lambda f: float(f),
    'top_k': lambda i: int(i),
    'top_p': lambda f: float(f),
    'include_prefix': lambda b: b == 'True',
    'return_as_list': lambda b: b == 'True',
    'truncate': lambda s: s
}


class Gpt2Handler:
    """This class respects the singleton design pattern and handles interacting with gpt2 to generate text."""
    __instance = None

    @classmethod
    def get_instance(cls):
        """Return the instance of this class. If it doesn't exist construct it first."""
        if cls.__instance is None:
            cls()
        return cls.__instance

    def __init__(self):
        """Initialise a Gpt2Handler instance if there is none. For internal use only."""
        if Gpt2Handler.__instance is None:
            Gpt2Handler.__instance = self
        else:
            raise Exception("Attempted initialisation of singleton class Generator.")

        # Start the TensorFlow session and load the model into it.
```

```python
        self.sess = gpt2.start_tf_sess()
        self.run_name = DEFAULT_CONFIG['run_name']
        self.download_model()
        self.load_model()

    def download_model(self):
        """Download the 124M gpt2 model if it is not downloaded"""
        if not gpt2.is_gpt2_downloaded():
            gpt2.download_gpt2()

    def load_model(self):
        """Load the gpt2 model. If it has already been loaded, reset it first."""
        try:
            gpt2.load_gpt2(self.sess, run_name=self.run_name)
        except FileNotFoundError:
            raise Exception(f'Model is missing. Place \'{self.run_name}\' in the checkpoint folder and try again.')

    def generate(self, title, initial_content='', num_samples=1, num_words=1023):
        """Generate a sample with the specified title and initial content."""
        initial_content = initial_content.replace('\n', ' ')  # Remove newlines
        # Convert the input into the correct format for the model
        prefix = '<|startoftext|>\n' \
                 + ('=' * 5) + 'TITLE' + ('=' * 5) + '\n' + title + '\n' \
                 + ('=' * 5) + 'CONTENT' + ('=' * 5) + '\n' + initial_content

        generate_args = self.parse_generate_arguments(DEFAULT_CONFIG)
        samples = gpt2.generate(self.sess, prefix=prefix, nsamples=num_samples, length=num_words, **generate_args)

        return samples

    def generate_as_tuple(self, title, initial_content='', num_samples=1, num_words=1023):
        """Generate a sample as a tuple in the form [title, content] with the specified title and initial content."""
        return [self.sample_to_tuple(sample) for sample in
                self.generate(title, initial_content, num_samples, num_words)]

    @staticmethod
    def sample_to_tuple(sample):
        """Take a sample and return a list where the first value is the title and the second value is the content."""
        # Remove the startoftext token and the title header
        no_title_header = sample.split('<|startoftext|>\n' + ('=' * 5) + 'TITLE' + ('=' * 5) + '\n')[1]
        # Remove any remaining tokens using a regex that matches substrings that start with '<|' and end with '|>'
        no_tokens = re.sub('<\\|[^|>]*\\|>', '', no_title_header)
        # Replace multiple adjacent spaces with a single space
        no_repeating_spaces = re.sub(' +', ' ', no_tokens)
        # Convert the sample into a list consisting of the title and sample without the sample header
        split_sample = no_repeating_spaces.split('\n' + ('=' * 5) + 'CONTENT' + ('=' * 5) + '\n')[:2]
        return split_sample

    @staticmethod
    def parse_generate_arguments(arguments):
        """Convert generate arguments from string to the correct respective types using GENERATE_ARGUMENT_PARSER."""
        return_value = {}
        for key in arguments:
            return_value[key] = GENERATE_ARGUMENT_PARSER[key](arguments[key])

        return return_value
```

C.2.4 gui.py

```python
import tkinter as tk
from tkinter import filedialog
from tkinter import messagebox

from generator import Generator


class Gui:
    """This class respects the singleton design pattern and is responsible for the GUI of the program."""
    __instance = None

    # Constants
    MIN_WINDOW_WIDTH = 600
    MIN_WINDOW_HEIGHT = 300
    ROOT_PAD_X = 2
    ROOT_PAD_Y = 2
    HOME_PAD_X = 2
    HOME_PAD_Y = 2

    @classmethod
    def get_instance(cls):
        """Return the instance of this class. If it doesn't exist construct it first."""
        if cls.__instance is None:
            cls()
        return cls.__instance

    def __init__(self):
        """Initialise a Gui instance if there is none. For internal use only."""
        if Gui.__instance is None:
            Gui.__instance = self
        else:
            raise Exception("Attempted initialisation of singleton class Gui.")

        # Initialise the instance variables for this object
        self.root = None

        # Home
        self.home = None
        self.title_option = None
        self.title_text = None
        self.initial_content_option = None
        self.initial_content_text = None
        self.number_of_samples = None
        self.words_per_sample = None

        self.create_gui()

    def start(self):
        """Launch the gui window."""
        self.root.mainloop()

    def create_gui(self):
        """Create the gui window and populate it with the relevant components."""
        # Window
        self.root = tk.Tk()
        self.root.title("Article Generator")
        self.root.minsize(width=self.MIN_WINDOW_WIDTH, height=self.MIN_WINDOW_HEIGHT)
        self.root.resizable(False, False)

        self.create_home()

    def create_home(self):
        """Create the home screen and populate it with the relevant components."""
        self.home = tk.LabelFrame(self.root, padx=self.ROOT_PAD_X, pady=self.ROOT_PAD_Y, borderwidth=0,
                                  highlightthickness=0)
        self.home.pack()

        # Title
        # Label
        title_label = tk.Label(self.home, text="Title:")
        title_label.grid(row=0, column=0, padx=self.HOME_PAD_X, pady=self.HOME_PAD_Y, sticky=tk.W)
        # Option Menu
        self.title_option = tk.StringVar(value='Text')
        title_option_menu = tk.OptionMenu(self.home, self.title_option, 'Text', 'File',
                                          command=self.on_title_option_menu_update)
        title_option_menu.grid(row=0, column=1, padx=self.HOME_PAD_X, pady=self.HOME_PAD_Y, sticky=tk.W)
        # Text Entry
        self.title_text = tk.Text(self.home, width=50, height=1, font=("Helvetica", 10))
        self.title_text.grid(row=0, column=2, padx=self.HOME_PAD_X, pady=self.HOME_PAD_Y, sticky=tk.W)
        # Do nothing when the user tries to enter a newline character
        self.title_text.bind('<Return>', lambda x: 'break')

        # Initial Content
        # Label
        initial_content_label = tk.Label(self.home, text="Initial Content:")
        initial_content_label.grid(row=1, column=0, padx=self.HOME_PAD_X, pady=self.HOME_PAD_Y, sticky=tk.W)
        # Dropdown
        self.initial_content_option = tk.StringVar(value='Text')
        initial_content_option_menu = tk.OptionMenu(self.home, self.initial_content_option, 'Text', 'File',
                                                    command=self.on_initial_content_option_menu_update)
        initial_content_option_menu.grid(row=1, column=1, padx=self.HOME_PAD_X, pady=self.HOME_PAD_Y, sticky=tk.W)
        # Text Entry
        self.initial_content_text = tk.Text(self.home, width=50, height=10, font=("Helvetica", 10), wrap=tk.WORD)
        self.initial_content_text.grid(row=1, column=2, padx=self.HOME_PAD_X, pady=self.HOME_PAD_Y, sticky=tk.W)

        # Number of Samples
        # Label
        number_of_samples_label = tk.Label(self.home, text="Number of Samples:")
        number_of_samples_label.grid(row=2, column=0, padx=self.HOME_PAD_X, pady=self.HOME_PAD_Y, sticky=tk.W)
        # Spinbox
        self.number_of_samples = tk.IntVar(value=1)
        number_of_samples_spinbox = tk.Spinbox(self.home, from_=1, to=99, width=9, textvariable=self.number_of_samples)
        number_of_samples_spinbox.grid(row=2, column=1, columnspan=2, padx=self.HOME_PAD_X, pady=self.HOME_PAD_Y,
                                       sticky=tk.W)

        # Words Per Sample
        # Label
        words_per_sample_label = tk.Label(self.home, text="Max Words Per Sample:")
        words_per_sample_label.grid(row=3, column=0, padx=self.HOME_PAD_X, pady=self.HOME_PAD_Y, sticky=tk.W)
        # Spinbox
        self.words_per_sample = tk.IntVar(value=1023)
        words_per_sample_spinbox = tk.Spinbox(self.home, from_=1, to=1023, width=9, textvariable=self.words_per_sample)
        words_per_sample_spinbox.grid(row=3, column=1, columnspan=2, padx=self.HOME_PAD_X, pady=self.HOME_PAD_Y,
                                      sticky=tk.W)

        # Generate Button
        generate_button = tk.Button(self.home, text='Generate', command=self.submit)
        generate_button.grid(row=4, column=1, columnspan=2, padx=self.HOME_PAD_X, pady=self.HOME_PAD_Y)

    def submit(self):
        """Submit the text in the fields to gpt2 and launch a window displaying the generated articles."""
        try:
            number_of_samples = int(self.number_of_samples.get())  # Ensure the number of samples is an int
            assert number_of_samples > 0  # Ensure the number of samples is greater than 0
        except (ValueError, AssertionError):  # Display a dialog box to the user to notify them of the error
            messagebox.showerror('Invalid Value', 'Number of samples must be a positive number.')
            return

        try:
            words_per_sample = int(self.words_per_sample.get())  # Ensure the words per sample is an int
            # Ensure the words per sample is between 1 and 1023 (inclusive)
            assert (words_per_sample > 0) and (words_per_sample < 1024)
        except (ValueError, AssertionError):  # Display a dialog box to the user to notify them of the error
            messagebox.showerror('Invalid Value', 'Words per sample must be a whole number between 1 and 1023.')
            return

        title = self.title_text.get('1.0', tk.END).rstrip()  # Retrieve the text from the title field
        if len(title) == 0:  # Ensure text has be inputted in the title field
            messagebox.showerror('Invalid Title', 'Title must not be blank.')
            return

        # Retrieve the text from the initial content field
        initial_content = self.initial_content_text.get('1.0', tk.END).rstrip()

        # Generate samples based on the user input
        samples = Generator.get_instance().generate_as_tuple(title, initial_content, number_of_samples,
                                                             words_per_sample)

        # Display the generated samples
        sample_viewer = Gui.SampleViewer(samples)
        sample_viewer.start()

    def on_title_option_menu_update(self, value):
        """Update the contents of the title text based on the respective option menu value."""
        self.on_option_menu_update(value, self.title_option, self.title_text)

    def on_initial_content_option_menu_update(self, value):
        """Update the contents of the initial content text based on the respective option menu value."""
        self.on_option_menu_update(value, self.initial_content_option, self.initial_content_text)

    def on_option_menu_update(self, value, option_menu, text_field):
        """Update the contents of the text field based on the new value of the option menu.
        If 'File' is selected the user will be prompted to select a file to source the new text from."""
        if value == 'Text':
            text_field.config(state='normal')
        else:
            if len(text_field.get('1.0', tk.END)) > 1:
                response = messagebox.askyesno('Continue?', 'The content of the file will overwrite the contents of the'
                                                            ' field.\nWould you like to continue?')
                if not response:  # If the user responded with No
                    option_menu.set('Text')
                    text_field.config(state='normal')
                    return

            filename = filedialog.askopenfilename(initialdir='/', title='Open',
                                                  filetypes=(('Text Files (*.txt)', '*.txt'),
                                                             ('All Files (*.*)', '*.*')))
            if filename:
                with open(filename, 'r', errors='surrogateescape') as f:
                    title = f.readline().rstrip()
                text_field.config(state='normal')
                text_field.delete('1.0', tk.END)  # Clear the contents of the field
                text_field.insert('1.0', title)
                text_field.config(state='disabled')
            else:  # If the user did not select a file
                option_menu.set('Text')
                text_field.config(state='normal')

    class SampleViewer:
        """This inner class is responsible for displaying articles to the user."""
        # Constants
        WINDOW_PAD_X = 2
        WINDOW_PAD_Y = 2
        TEXT_FRAME_PAD_X = 2
        TEXT_FRAME_PAD_Y = 2

        def __init__(self, samples):
            """Initialise the instance variables for this object, then create and populate the sample viewer window."""
            self.window = None
            self.left_button = None

            self.title = samples[0][0]
            self.samples = [sample[1] for sample in samples]
            self.right_button = None

            self.title_text = None
            self.sample_text = None
            self.current_sample_index = 0

            self.text_frame = None

            self.create_window()

        def create_window(self):
            """Create and populate the sample viewer window."""
            self.window = tk.Toplevel()
            self.window.title(self.title)
            self.window.resizable(False, False)
            self.window.grab_set()

            self.create_buttons()
            self.update_buttons()

            self.create_text_frame()
            self.update_sample()

        def create_text_frame(self):
            """Create and populate the frame responsible for displaying the text."""
            self.text_frame = tk.LabelFrame(self.window, borderwidth=0, highlightthickness=0)
            self.text_frame.grid(row=0, column=0, columnspan=2, padx=self.WINDOW_PAD_X, pady=self.WINDOW_PAD_Y)

            self.title_text = tk.Text(self.text_frame, width=100, height=1, font=("Helvetica", 10),
                                      padx=self.TEXT_FRAME_PAD_X, pady=self.TEXT_FRAME_PAD_Y)
            self.title_text.insert(tk.END, self.title)
            self.title_text.configure(state='disabled')

            self.title_text.grid(row=0, column=0)

            self.sample_text = tk.Text(self.text_frame, width=100, height=20, font=("Helvetica", 10), wrap=tk.WORD,
                                       padx=self.TEXT_FRAME_PAD_X, pady=self.TEXT_FRAME_PAD_Y, state='disabled')
            self.sample_text.grid(row=1, column=0)

        def create_buttons(self):
            """Create the buttons and populate them in the relevant window."""
            self.left_button = tk.Button(self.window, text='<', command=self.previous_sample)
            self.left_button.grid(row=1, column=0, padx=self.WINDOW_PAD_X, pady=self.WINDOW_PAD_Y, sticky=tk.E)
            self.right_button = tk.Button(self.window, text='>', command=self.next_sample)
            self.right_button.grid(row=1, column=1, padx=self.WINDOW_PAD_X, pady=self.WINDOW_PAD_Y, sticky=tk.W)

        def previous_sample(self):
            """Change the current sample displayed to the previous sample."""
            if self.current_sample_index == 0:
                messagebox.showerror('Error', 'First sample reached. No previous sample exists.')

            self.current_sample_index -= 1
            self.update_sample()
            self.update_buttons()

        def next_sample(self):
            """Change the current sample displayed to the next sample."""
            if self.current_sample_index == (len(self.samples) - 1):
                messagebox.showerror('Error', 'Last sample reached. No next sample exists.')

            self.current_sample_index += 1
            self.update_sample()
            self.update_buttons()

        def update_sample(self):
            """Update the currently displayed sample to the currently selected sample."""
            self.sample_text.configure(state='normal')
            self.sample_text.delete("1.0", "end")
            self.sample_text.insert(tk.END, self.samples[self.current_sample_index])
            self.sample_text.configure(state='disabled')

        def update_buttons(self):
            """Disable/enable the buttons depending on the relative position of the currently selected sample."""
            if self.current_sample_index == 0:
                self.left_button.config(state='disabled')
            else:
                self.left_button.config(state='normal')

            if self.current_sample_index == (len(self.samples) - 1):
                self.right_button.config(state='disabled')
            else:
                self.right_button.config(state='normal')

        def start(self):
            """Launch the gui window."""
            self.window.mainloop()
```