Topic-based Text Generation in a Specified Author's Style

ECE4179: Neural Networks and Deep Learning Project
Semester 2, 2021
Elizabeth Chai (2881166) & Harrison Broadbent (31481205)

Introduction

Pre-trained language models are commonly used for style transfer and predictive text (i.e. text autocompletion and grammar correction). Applications of text generation on a specified topic in the style of a specific author has been done extensively with Recurrent Neural Networks (RNN), but less so with recent natural language (NLP) state-of-the-art (SOTA): transformers (2017) [1], such as OpenAi's GPT-2 (2019) [2]. Transformers are a non-recurrent alternative to RNNs, aiming to solve sequence to sequence tasks in parallel, handling long-range dependencies. GPT-2 has proven to be very powerful in its capability for generating fluent, grammatically correct and well-formatted text.

The project aimed to leverage GPT-2 to create paragraphs of text based on a specified topic in the style of a specified author. Most style transfer and text generation work has been based on supervised learning methods (i.e. translating Shakespeare to modern English [3]), since parallel datasets already exist (original Shakespeare text paired with a modern English translation). However, since parallel datasets for most authors do not exist, an unsupervised approach was taken. This approach involved paraphrasing inputs to generate synthetic data, which removes the requirement for pre-existing parallel data. Two different methods of approaching this problem were explored.

The project's models were trained on text extracted from the works of specific target authors (books, song lyrics etc.), with additional synthetic data generated using various paraphrasing techniques. During inference, the models were provided with an input prompt, and then tasked with generating some output text.

The project result has potential use in generating various textual works, such as essays, blog posts, speeches or dialogue for chatbots, in the style of a certain author, music artist, or public figure. This would allow greater personalization and humanization of AI generated works.

Transformers

GPT-2, the Generative Pre-trained Transformer 2, is based on a transformer architecture, which uses attention instead of a recurrence-based network architecture [2]. Attention is a mechanism that helps capture relationships between words. It makes transformers faster to train and able to run in parallel, which also removes the issue of vanishing gradients.

A transformer has an encoder and decoder block (Figure 1). The encoder block uses multi-headed attention which captures how relevant a particular word is compared to other words in a sentence. The feedforward block of the encoder is applied to every attention vector and transforms vectors into an acceptable form for the next layer, which is another encoder or decoder. The decoder is passed in the expected outputs. Masked-multi-headed attention is used to pass a context into the block while hiding the next output (so the network cannot use it). This output is combined with the output from the encoder block, where the relationship between them is calculated via the multi-headed attention. The linear layer uses softmax to transform outputs into a conditional probability distribution, and the highest probability is the resulting predicted word. Decoding methods are used as the sequence of possible results is large [4].

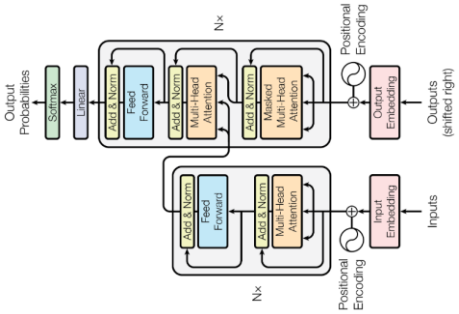

Figure 1: Single transformer block of encoder and decoder [1]

Related Work

Attention is All You Need

GPT-2 is a transformer-based language model with 1.5 billion parameters, trained on a dataset of 8 million web pages. The model's objective is to predict the next word from the previous words in a text [2]. It is an implementation of transformers from the paper "Attention is All You Need" by Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan Gomez, Łukasz Kaiser and Illia Polosukhin [1].

The paper introduces the architecture of transformers, where an encoder-decoder configuration is used with attention, removing the requirement for recurrence and convolutions. Transformers are the current SOTA for NLP. The paper is very clear in explaining the architecture of a transformer block and how the attention mechanism works. This approach was found to be interesting and clever, as it removes the dependency on memory and allows for parallelisation. The paper also details useful results and future applications. This paper is related to the project, as the project uses the transformer architecture described in the paper.

Hugging Face Transformers

Hugging Face Transformers is a library of general purpose architectures for natural language understanding (NLU) and natural language generation (NLG), which include pre-trained models like the GPT-2 Tokenizer and Models [5]. The library allows developers to easily code up a transformer model and train it with required parameters, hence is why this work is related to the project.

An implementation of text generation using GPT-2 and Hugging Face transformers has been written by [15], which consists of a text generation tutorial using GPT-2 and Hugging Face Transformers. It provides a good start on generating text based on an input topic or line, and can be extended further to generate longer text on a specific topic. This can be used as a baseline, however more related work will need to be found on how to transfer generated text into specific author style in an unsupervised fashion. A suggestion was to transfer-learn using some text of the author. The same tokenizer for the author dictionary is used, and by training it, certain words have more weight and higher probability when the next word is being chosen. The Hugging Face Transformers library was explored further for transfer learning and fine-tuning.

Evaluating Creative Language Generation with RNN

The paper 'Evaluating Creative Language Generation with RNN' describes an implementation of RNN text generation to generate new verses in the style of rap music given as input [8]. This RNN approach is described in [7, 8], and an implementation found on GitHub. This is very similar to the project aims, however it uses a RNN and not a transformer. It is also able to output only a few words as a line, as opposed to whole paragraphs. An example implementation is provided on Kaggle [6], for the application of a poetry generator, where text from poets and lyricists are used to train an RNN, and text is generated. This paper and its implementations were useful in providing an example of author style transfer for a specific topic. However, it implements an RNN framework instead of a transformer-based one.

Paraphrasing with Large Language Models

An implementation of transformer style transfer from text is found in the paper 'Paraphrasing with Large Language Models' [9]. The article is useful as it describes the fine-tuning technique for GPT-2 based on any text file. The fine-tuned model can then be used to perform paraphrasing tasks on a variety of texts and subjects and to generate paraphrases at a sentence and paragraph level. The paper proposes a system that generates paraphrased examples in an autoregressive fashion using a neural network, without need for top-k word or beam search. It details using a GPT-2 model and fine tuning on a supervised dataset of pre-made paraphrase examples. The paper also provided a useful explanation of scoring, where generated sentences can be compared to original datasets by using average scores, i.e. USE (Universal Sentence Encoder), ROUGE-L and BLEU.

An implementation of this paper is provided by the article 'Paraphrasing and Style Transfer with GPT-2 [3]. The article described text generation and paraphrasing Shakespeare via fine-tuning the GPT-2 Hugging Face Transformer. The article was useful as it showed how code can be tweaked to finetune GPT-2 based on any text file, where one must first define a model trainer with tokenized values, dataset, trainer, then test with a new text file. The model can then be used to generate paraphrased text in the style of input text file (i.e Shakespeare) based on input string/topic. The project aims to extend from this baseline by generating more text from the input topic (rather than paraphrasing the input sentence), as well as allow for more styles (authors) other than Shakespeare.

Unsupervised Paraphrase Generation Using Pre-Trained Language Models

The paper 'Unsupervised Paraphrase Generation Using Pre-Trained Language Models' proposes a novel unsupervised paraphrasing technique using GPT-2 [10]. The model is trained on domain specific independent sentences at hand and then used to generate paraphrases without suffering from domain shift. An unsupervised paraphrasing task is posed as a sentence reconstruction task from corrupted input. That is, from the sentence, omit all stop words to form a corrupted sentence, scramble and reorder, Source S, and the original sentence is used as Target T. GPT-2 was then used to generate Target sentence given Source, i.e. P(T|S). This paper was useful as it provided a framework for training a transformer in an unsupervised setting. This was a useful framework to fine tune a model on specific text from an author (see Method 1).

Reformulating Unsupervised Style Transfer as Paraphrase Generation

Another unsupervised approach to text style transfer is detailed in the paper titled 'Reformulating Unsupervised Style Transfer as Paraphrase Generation' [11]. This method uses a second language model, pre-trained on the task of paraphrasing text. The paraphraser is used to generate a body of synthetic data, which maps back onto the input dataset. The aim of this process is to synthesise a high-quality body of training data in an unsupervised manner, with the implication that the stylised output should better represent the semantics of the underlying text compared to other methods. This paraphrasing method is different to [10]; this implementation is quite interesting and has a lot of potential (see Method 2). Despite this approach appearing rather convoluted, example code has been provided on GitHub. The paper also describes how the quality of style transfer and text generation models can be evaluated, in which semantic similarity, transfer accuracy and fluency were used.

Deep Learning for Text Style Transfer Survey

Other NLP models and papers were skimmed for more ideas, these include surveys of SOTA NLP [12, 13]. The 'Deep Learning for Text Style Transfer Survey' describes that style transfer models should distinguish between style and content and also highlights different focuses of text style transfer [12].

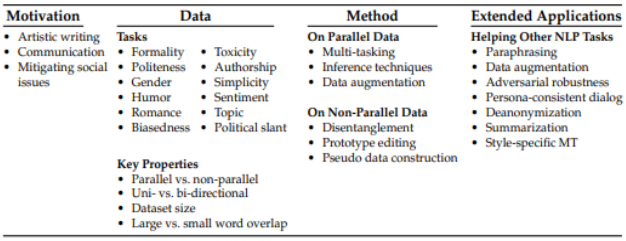

Figure 2: Current implementations of style transfer models [12]

The 'Style Transfer in Text' survey lists a bunch of supervised, unsupervised and semi-supervised text style transfer papers and code implementations [13]. It was a useful resource to find more examples of style transfer and see how they were done. These surveys were useful to provide more insight into current unsupervised text generation based on an author's style, and related works that provide building blocks for the project.

Approach

Pre-trained GPT-2 models used extensively, along with additional architecture to fine-tune for a specific author and then generate text from a specified topic, in an unsupervised setting. Although this has been done before for RNNs, this type of text generation has not been attempted for the current state-of-the-art model, the transformer. Additionally, text generation and style transfer tasks have been completed by a transformer both with supervised and unsupervised methods independently, but never together.

Using natural language models involves developing meaningful representations of words, and for RNNs and transformers, this is done by using word embeddings and embedding space, or tokenizers. An embedding space is an open space of every word mapped and assigned a particular vector of numbers (as computers understand numbers and not words). Words are given word embeddings, such that words with similar meanings are grouped or are present close to each other [4]. Fine-tuning the GPT-2 model with author-style text data allows the word embeddings to update, allowing for author-relevant text. From this, the model can be used to generate text that sounds like the author.

Text generation training requires inputs and expected output sequences. The model then uses the current words in the input to generate or 'predict' the next word and updates its weights based on the loss from the output [4]. For this project, training data consisted of text from an author and the input was some starting phrase, to prompt the model. However, in many cases there exists no expected output text for a specific topic from the author. For example, there is no writing from Emily Dickinson about computers and phones. To remedy this, various methods can be employed to artificially create outputs, through various paraphrasing and text-augmentation techniques. Some examples are removing words, rearranging the order of the words, or generating entirely new paraphrases of input phrases. These methods all aim to retain the semantic meaning of the input text, but with some meaningful alterations for a deep learning model to train on.

Decoding methods for the prediction of the next word can vary the output of the transformer. Such decoding techniques include greedy search, beam search, top-k sampling, top-p sampling and combinations of the aforementioned [14]. Each decoding technique has different hyperparameters that can be changed to affect the text generated.

Quality of style transfer and text generation can usually be evaluated by semantic similarity, transfer accuracy, fluency and human evaluation were used, similar to the 'Reformulating Unsupervised Style Transfer as Paraphrase Generation' paper [11]. However additional language models or labels are required for the former three.

From this basis of knowledge, two unsupervised approaches were devised:

1. Method 1 involved fine-tuning a GPT-2 model using text from a specific author. The original text was 'corrupted' (i.e. paraphrasing: word removal, scrambling), and used as the input training data 'Source', while the original text was used as the expected output 'Target', similar to the 'Unsupervised Paraphrase Generation using Pre-trained Language Model' paper [10]. The model was trained to generate the Target sequence given the Source ($P(Target|Source)$), thus updating the word embeddings and attention vectors. The model can then be used to generate text (i.e. predict the next word) from a specified topic. A Hugging Face Transformers pipeline was used to tokenize, fine-tune and generate text.
2. Method 2 involved reproducing the two-step architecture mentioned above (Related Work, [11]) and training it on a different set of author texts. This method used a second language model, pre-trained to paraphrase text. The paraphraser model was used to generate a body of synthetic data, which mapped back onto the input dataset. The first language model was then trained to map from the paraphrased text (unstyled text) onto the author's text (styled text). The underlying assumption in this process was that the paraphrased text should resemble generic "unstyled" text, and then over the course of training, the model can be trained to paraphrase arbitrary "unstyled" text into the style of the target author.

Method 1

1. Import libraries and set seed
2. Load author-specific text and create Source sentences via corruption (remove stopwords, scramble and remove random words)
3. Create GPT-2 trained tokenizer
4. Create datasets (load & tokenize text, append Source (input) and Target (output) sentences, scramble the order of training sentences)
5. Define GPT-2 model
6. Fine-tune model (specify epochs, training batch size, learning rate, weight decay and seed)
7. Generate topic sentence (encode sentence using tokenizer)
8. Pass encoded sentence into the trained model (specify decoding method (and decoding-method-specific parameters), maximum output length, early stopping, number of repeated words)
9. Decode and print returned output result from model
10. Evaluate model performance

These steps are detailed more below and the code can be found in a link in the Appendix:

Step 1: Specialised libraries that were used include NumPy, PyTorch, transformers and datasets from Hugging Face and nltk (Natural Language Toolkit) for stopword removal. A seed was set to ensure the training and results were reproducible.

Step 2: A corpus from a specific author was loaded in and saved as sequences that were split by newlines. These sequences (Target) were then corrupted (Source) by removing stopwords, scrambled and removing random words. The percentage that the words were scrambled or removed can be modified, however, they were kept the same throughout for simplicity. The corrupted sequences were saved to a file

Step 3: A pre-trained GPT-2 tokenizer was created by using the Hugging Face library. A special separation token was created, which was used to separate the Source and Target.

Step 4: The full dataset of Source and Target sequences was created by loading in both saved corpora and tokenizing. The tokenizer created in Step 3 was used to convert words in sequences into numbers that a computer understands. Training sequences were formatted by appending the Source, separation token and Target together. The order of the sequences was then scrambled before training.

Step 5: A pre-trained (Hugging Face) GPT-2 model was defined. The 'gpt2-large' model was used as it has 36 layers and 774 million parameters. This allows for better text prediction and deep generation potential. The structure of a single layer (encoder and decoder) is seen in Figure 1 and for multiple layers, encoders and decoders are stacked respectively.

Step 6: The model was fine-tuned by specifying a Hugging Face trainer. The number of epochs, training batch size, learning rate, weight decay and seed can be varied to change the behaviour of model training. Training batch size was kept constant at 2 due to memory issues. The parameters (and attention weights) of each layer in the transformer encoder and decoder blocks are learned by backpropagation, much like any deep neural network. However, with the use of attention, vanishing gradients are avoided. The use of residual connections helps the network train, allowing the gradients to flow through the networks directly. Layer normalisation reduces the training time necessary (much like batch normalisation).

Step 7: The user can specify their desired input topic, i.e. 'The weather is nice today'. This topic sequence was then encoded with the same tokenizer.

Step 8. The encoded topic sequence was then fed into the model, where an output paragraph was generated.

Generation of a whole sequence is based on a conditional probability distribution:

$$P(w_{1:T}|W_0) = \prod_{t=1}^{T} P(w_t|w_{1:t-1}, W_0) \text{ ,with } w_{1:0} = \emptyset,$$

Where w is the predicted output sequence, W0 is the topic sequence, T is the length of the sequence and t is the corresponding timestep of T. The length of the output can be specified by the maximum output length. Other specifications include decoding method (and decoding method-specific parameters), early stopping, number of repeated words. Various decoding techniques, such as beam search, top-k, top-p and a combination of top-k and top-p were trialled. Beam search chooses the next word by only keeping track of the num_beams most likely results, top-k sampling filters for the k most likely words and redistributes probabilities and top-p (nucleus) sampling chooses the smallest possible set of words whose cumulative probability exceeds a probability p.

Step 9: The result is then decoded and displayed.

Step 10: The decoded result can then be evaluated. Due to time constraints, no evaluation model was created, so only human evaluation was performed.

Method 2

1. Prepare and encode the training dataset
2. Generate synthetic data using pre-trained a paraphrase model
3. Train the inverse-paraphraser model
4. Use the inverse-paraphraser model to generate new, stylised text.

The first step for this method was to prepare the data for the model. Input data, in the form of text files, was first run through the RoBERTa [16] encoder to translate files into byte-pair encodings (BPE), to represent word embeddings. These files are then passed through Fairseq [17] to generate binary files, which then get used for training. Exact details for preparing a dataset are provided in the project GitHub repository (see Appendix).

Once a dataset had been prepared, it was then paraphrased using a pre-trained GPT-2 paraphraser model. This creates a synthetic set of data, which lies in parallel to the original dataset. Text from the parallel dataset corresponds to paraphrased versions of the text in the original dataset. In doing this, the initial dataset gets complimented with paraphrased data and created a mapping between paraphrased "unstyled" text, and the original "styled" text.

The next step was to re-train another GPT-2 model to rephrase the paraphrased "unstyled" text into the original "styled" text. In doing so, the model was taught to reformulate text into a target style and to recognize mappings between "unstyled" and "styled" text.

Finally, this model, dubbed an "inverse-paraphraser", can be used to reformulate input text into that of the target style. See the example outputs below (Analysis of Results). It's important to note that this model performs a task slightly different to the project aim — rather than generate arbitrary length text, it paraphrases the input text into the target style. While it is possible to tweak things to generate longer outputs, it was found that the model produces more sensible and sophisticated outputs when restricted in this manner, likely due to the nature of the training process (training to convert from paraphrased → original text).

The code for this approach is available at the GitHub repository linked in the Appendix.

Dataset

GPT-2 was used as a starting point for text generation. It is a pre-trained transformer-based language model with 1.5 billion parameters, trained on a dataset of 8 million web pages. Its objective is to predict the next word, given all the previous words within some text [2].

Additional text data from selected authors were found from Kaggle [6]. This formed an initial body of text in the style of specific authors and music artists. This was complemented with additional text files from several authors, scraped from texts in the public domain. Collection methodology includes downloading the text files from Kaggle and copying text from popular works by the author into text files, filtering and processing where appropriate.

In total, over 300,000 lines of text across 22 different authors or musicians were collected. This was further augmented through the various paraphrasing techniques used, which are detailed above (see Method 1, Method 2), to generate sets of parallel data for each of the 22 corpora. It is important to note that only a couple of the 22 collections were ultimately used — over the course of the project only a handful of different style-transfer models were trained (and each style-transfer model uses only a single dataset).

Analysis of Results

Method 1

For fine-tuning of the model, the number of epochs, learning rate and weight decay must be specified. Training batch size was kept constant at 2 due to memory constraints and an RNG seed was declared constant for reproducibility. Weight decay was kept at 0. More epochs made the model longer to train, and text generation results did not improve, as the model was already quite deep. Training for epochs between 1-3 generally causes poor text generation due to a 'warm-up' effect of transformers. A learning rate around 6.25e-5 is commonly found in literature, and this has to be balanced with the warm-up effect. Hence, a learning rate of 6.25e-5 and epochs of 5 was used. However, during training, some difficulty was encountered when trying to use paraphrased text, where the model was unable to calculate the loss between the target and source phrase. The decoding method for the transformer can also be specified, where beam search, top-k, top-p and a combination of top-k and top-p were trialled. Each method has specific parameters which can be changed. For text generation, the maximum output length was kept constant at 500 words.

For beam search, the number of beams specified could be changed. After some experimentation, 5 beams were found to be a good balance of word variation and relevance. A parameter of no repeats of 2 words was specified, so words would not be repeated more than 2 times in a row (which counteracts a common drawback of beam search). For the topic of 'The weather is nice today' the following output was returned using beam search (only the first few lines are shown, see Appendix for full output):

"The weather is nice today, but it's going to be a long day." "I'm just glad we're not in the middle of a snowstorm. It would have been a lot worse if we were in a blizzard. We could have gotten stuck in there for a while, and I don't think we'd be able to get out of there.

This text is understandable and the story is coherent. However, it grows repetitive about the same kind of feelings and thoughts.

For the top-k sampling decoding method, a value of k of 50 was used, as this resulted in a relatively quick generation time, but balanced with a variation of words. For the topic of 'The weather is nice today', the first few lines of output using top-k decoding are:

"The weather is nice. We had a lovely walk on the beach at midnight and at about 3 a.m. when the moon came up I knew you were going to be here. It's a beautiful day. It's sunny outside. It's a beautiful day for swimming and snorkelling."

This story is less coherent, jumping around from different people and places. But for each mini-story in the generation, the text is quite descriptive. The first paragraph manages to stay on topic.

For the top-p sampling decoding method, a p-value of 92% was chosen, as values above 0.9 were commonly chosen. Adding the 0.02 resulted in a slightly more coherent and well-flowing generated text, however, it is hard to discriminate qualitatively. For the topic of 'The weather is nice today', the following output was returned using top-p decoding:

The weather is nice, so we're going to try to do the best we can to make sure our prospects are good enough to try to sign him.

This story is coherent, however, the context of the weather is quickly lost.

Since top-k of 50 resulted in descriptive, on-topic text and top-p of 92% resulted in a coherent story, both were trialled together:

The weather is nice and nice today! Just perfect for a warm start to the year. I got home at about 9:00pm this afternoon to find a package waiting for me! The first thing I noticed was that it was a huge box and I wasn't quite sure how it was made but when I opened the box I saw that it wasn't just a package, but a giant treat... Mascarpone!

It is a pretty coherent and funny story about Mascarpone.

From the above, it is evident that none of the decoding methods are perfect, however, it seems that the final combination of top-p and top-k sampling subjectively resulted in the best text generation. More generation examples are shown in the Appendix for 'Neural networks and deep learning' and 'Covid vaccinations in Australia' using the same parameters. However, some outputs drift from the topic choice. To combat this, different decoding methods or varying attention weights could be trialled.

Method 2

Example outputs from two separate models are shown below. The first model was trained on 25,000 lines of song lyrics from the rapper Eminem. The second model was trained on 35,000 lines of text extracted from the works of Leo Tolstoy.

The final hyperparameters used to train the inverse-paraphrasing model were a learning rate of 5e-5 using the Adam optimizer, trained for 3 epochs with a batch size of 5 phrases. Gradient accumulation was also used, with the interval set to 2. Some experimentation had to be done with the top_p parameter, a parameter passed to the GPT-2 model, to ensure the outputs were grammatically and syntactically correct but still mimicked the style and vocabulary of the target author or artist.

As mentioned previously, the model outputs were evaluated subjectively, and you can see below that they are quite interesting. Given the same input sentences, the two models produced varied and distinct outputs.

The Eminem model employs a less formal register and uses slang and abbreviations, as well as ellipses of certain verbs ("they just cool" instead of "they are just cool", for example), mimicking the style of modern rap music. It also draws the connection between unhappy families and police records, which was a common theme in some of Eminem's early works.

In contrast, the Tolstoy model generates text in a more formal and literate register, using words with greater lexical complexity ("really cool" vs "truly marvellous"), and more complex sentence constructions ("one must..."), indicative of the literary context which it draws from.

Comparing these two models gives an idea of their ability to transfer style, and overall it has been quite successful in doing so — both models are able to wield a vernacular appropriate to the target author, and more interestingly, both use appropriate sentence structures for the target style. Certainly, the models are not perfect — they do have a propensity to ignore some parts of the input context (see the final Eminem example, which ignores the first half of the input phrase).

They also require the top_p setting to be quite high, which allows them to draw on a wide range of vocabulary, but this comes at the expense of coherence. All the examples from below are direct outputs of the language models, however, these outputs were manually selected from 5-10 generations for each input phrase — many outputs were incoherent or nonsensical. While clearly, this was not ideal, overall the results of this method can still be considered a success. The model was able to generate some outputs that clearly demonstrate a transfer of style into that of the target author, meeting the project aims.

Artist: Eminem

INPUT: I love learning about Neural networks and deep learning because they are really cool
OUTPUT: Got me on the deep learning train and love 'cause they kinda remind me of nervous systems
OUTPUT: Just love learning about Neural Networks and deep learning 'cause they just cool stuff

INPUT: All happy families are alike. Each unhappy family is unhappy in its own way
OUTPUT: Every unhappy family's like a police record.

Author: Leo Tolstoy

INPUT: I love learning about neural networks and deep learning because they are really cool
OUTPUT: I love learning about Neural Networks and deep learning because they are truly marvellous. But one must always remember one's soul.

INPUT: All happy families are alike. Each unhappy family is unhappy in its own way
OUTPUT: All families are alike in their misery. Each unhappy family is unhappy by its own peculiar propensities.

Conclusion and future work

The project met its aims, as GPT-2 was leveraged to generate text based on a specified topic, in the style of a specified author. This was done by fine-tuning pre-trained GPT-2 models and using unsupervised paraphrasing approaches to generate synthetic parallel training data.

Method 1 was able to generate interesting text of considerable length, however, it struggled to map the author's style onto the output text. Method 2 was more successful at transferring the author's writing style into the generated text, but it was limited to paraphrasing input text, rather than generating arbitrary-length samples. Both methods had their benefits and drawbacks, and future work could be done to integrate them together so that paragraphs of text could be generated while sticking close to the target style.

To quantitatively evaluate these models, future work could include the implementation of evaluation models to use semantic similarity (BLEU), transfer accuracy and fluency. Additional work would be to train the model on more authors and explore more diverse paraphrasing techniques such as different data augmentations. With modification, it would be possible to further refine the style of the output, specifying tone and setting (casual or formal, and speech, dialogue, story, blog post or essay), making this technology useful in a broader context.

Appendix

GitHub Repository (basic dataset + code)

Method 1 Code
Method 1 Text Generation

Method 2 Code
Method 2 Google Drive (trained models + datasets)

Project Video

References

[1] A. Vaswani *et al.*, "Attention Is All You Need," *arXiv:1706.03762 [cs]*, Dec. 2017, Accessed: Sep. 14, 2021. [Online]. Available: http://arxiv.org/abs/1706.03762

[2] "Better Language Models and Their Implications," *OpenAI*, Sep. 14, 2021. https://openai.com/blog/better-language-models/ (accessed Sep. 14, 2021).

[3] DataChef, "Paraphrasing and Style Transfer with GPT-2," Jul. 30, 2021. https://datachef.co/blog/paraphrasing-with-gpt2/ (accessed Sep. 14, 2021).

[4] U. Ankit, "Transformer Neural Network: Step-By-Step Breakdown of the Beast.," *Medium*, Sep. 15, 2021. https://towardsdatascience.com/transformer-neural-network-step-by-step-breakdown-of-the-beast-b3e096dc857f (accessed Sep. 14, 2021).

[5] "Transformers." https://huggingface.co/transformers/index.html (accessed Sep. 14, 2021).

[6] "Poetry Generator (RNN Markov)." https://kaggle.com/paultimothymooney/poetry-generator-rnn-markov (accessed Sep. 14, 2021).

[7] P. Potash, A. Romanov, and A. Rumshisky, "GhostWriter: Using an LSTM for Automatic Rap Lyric Generation," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, Lisbon, Portugal, 2015, vol. Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pp. 1919–1924. doi: 10.18653/v1/D15-1221.

[8] P. Potash, A. Romanov, and A. Rumshisky, "Evaluating Creative Language Generation: The Case of Rap Lyric Ghostwriting," *arXiv:1612.03205v1 [cs.CL]*, Dec. 2016, Accessed: Sep. 14, 2021. [Online]. Available: https://arxiv.org/pdf/1612.03205.pdf

[9] S. Witteveen and M. Andrews, "Paraphrasing with Large Language Models," *arXiv:1911.09661v1 [cs]*, Nov. 2019, Accessed: Sep. 14, 2021. [Online]. Available: https://arxiv.org/pdf/1911.09661.pdf

[10] C. Hegde and S. Patil, "Unsupervised Paraphrase Generation using Pre-trained Language Models," *arXiv:2006.05477v1 [cs]*, Jun. 2020, Accessed: Sep. 14, 2021. [Online]. Available: https://arxiv.org/abs/2006.05477v1

[11] K. Krishna, J. Wieting, and M. Iyyer, "Reformulating Unsupervised Style Transfer as Paraphrase Generation," *arXiv:2010.05700 [cs]*, Oct. 2020, Accessed: Sep. 25, 2021. [Online]. Available: http://arxiv.org/abs/2010.05700

[12] D. Jin, Z. Jin, Z. Hu, O. Vechtomova, and R. Mihalcea, "Deep Learning for Text Style Transfer: A Survey," *arXiv:2011.00416 [cs]*, Apr. 2021, Accessed: Sep. 20, 2021. [Online]. Available: http://arxiv.org/abs/2011.00416

[13] Zhenxin Fu, "A Paper List for Style Transfer in Text," *GitHub*, Oct. 06, 2021. https://github.com/fuzhenxin/Style-Transfer-in-Text (accessed Sep. 14, 2021).

[14] "How to generate text: using different decoding methods for language generation with Transformers." https://huggingface.co/blog/how-to-generate (accessed Sep. 20, 2021).

[15] M. Bhikadiya 💡, "AI Writer ✍️: Text Generation Using GPT-2 & 🤗Transformers," Analytics Vidhya, Apr. 16, 2021. https://medium.com/analytics-vidhya/ai-writer-text-generation-using-gpt-2-transformers-4c33d8c52d5a (accessed Sep. 13, 2021).

[16] 'Fairseq'. https://github.com/pytorch/fairseq (accessed Oct. 20, 2021).

[17] 'RoBERTa'. https://huggingface.co/transformers/model_doc/model_doc/roberta.html (accessed Oct. 20, 2021).