SpotGPT: Variational Autoencoder-based Machine-Generated Text Detection

Ahmed Abdellatif **Achinth Bharadwaj**
University of British Columbia
{achinth, ahmedab}@student.ubc.ca

Abstract

Generative Pre-trained Transformers (GPTs) are a new iteration of large language models (LLMs), trained on large amounts of text data using a variety of supervised, unsupervised and semi-supervised modalities. As the proliferation of machine-generated text has increased exponentially since late 2022, distinguishing between human-generated text (HGT) and machine-generated text (MGT) has become a crucial responsibility. Some methods have been developed for detecting machine-generated text (e.g., GPTZero, DetectGPT), yet none have used Bayesian- or probalility-based approaches to approximate GPT-generated text. This project proposes the use of a Variational Autoencoder (VAE) to help in understanding underlying distribution of MGT by GPTs. We compare our results with a baseline Text-to-Text (T5) transformer, while rigorously validating our models and our accompanying probability-ratio-based scoring with data generated by ChatGPT to see if our methods are capable of achieving MGT-to-HGT separability.

1 Introduction

The misuse of generative pre-trained transformers (GPTs) has become a growing concern in many fields - primarily in journalism, academic integrity, and social media misinformation campaigns. GPTs are powerful language models that can generate high-quality text, making it difficult to distinguish between human-generated text (HGT) and machine-generated text (MGT). This has led to an uptake in use of GPTs for unethical purposes such as academic fraud and the spread of false information.

Since the explosion of ChatGPT's use in late November 2022, novel research methods have been developed, with the creation of tools such as GPTZero by Edward Tian and the Princeton NLP lab, and DetectGPT (Mitchell et al., 2023). The latter is a zero-shot method which relies on probability curvature of the transformer and text pertubation to estimate a body of text that has been generated by a GPT-based model. However, we were unable to find any research projects or literature which use unsupervised learning or probablistic assumptions with available training data that were published in recent discourse.

1.1 Related Work and Literature Review

Several studies have been conducted on the topic of machine-generated text detection.

The aforementioned GPTZero uses a classification model that predicts whether a document was written by a large language model, providing predictions on a sentence, paragraph, and document level. The classifier returns a document-level score, completely-generated-prob, that specifies the probability the entire document was AI-generated. It works robustly across a range of AI language models, including but not limited to ChatGPT, GPT-3, GPT-2, LLaMA, and AI services based on those models. [7]

DetectGPT, proposed by Mitchell et al. (2023) is a method for detecting machine-generated text based on analyzing the probability distribution of the text generated by a language model and computing the curvature of the probability distribution function. The method achieves state-of-the-art performance on several benchmark datasets and can be used for a wide range of applications, such as identifying fake news, detecting plagiarism, and improving the reliability of automated content generation systems. [10]

Raffel et al. (2020) proposed a text-to-text transformer model called T5, which achieved state-of-the-art results on various natural language processing tasks. In addition, the authors demonstrated the ability of the model to generate coherent and grammatical text that can pass human evaluation. [11]

In addition, Grover, a neural network-based model capable of generating high-quality fake news articles, was proposed as a benchmark for evaluating the performance of machine-generated text detection methods (Zellers et al., 2019). Grover was trained on a large corpus of real news articles and was shown to generate articles that could fool human judges into thinking they were written by real journalists. [14]

1.2 Contribution

Culminating in our research, we have implemented **spotGPT** in its beta form - a variational autoencoder-based scoring system to assess whether some body of text has been generated by GPT or through some other source. The model and other associated research have all been conducted extensively in PyTorch. We also aim to add to the plethora of research in the field of machine-generated text detection, with the hope that our studies will further the quality of models capable of classifying MGT from HGT. The code for this project, along with all other research-associated notebooks can be found at this GitHub repository made available here

2 Background

2.1 Variational Autoencoders

Variational autoencoders (VAEs) are a type of deep generative model that learn to encode and decode high-dimensional data, in our case - text signals. Unlike traditional autoencoders that learn a deterministic mapping from input to output, VAEs learn a probabilistic mapping that can generate new data samples from the learned distribution. This property makes VAEs particularly useful for tasks such as augmentation, and representation learning.

The main goal of a VAE is to use an encoder network to map the input data x to a lower-dimensional latent representation z, and a decoder network to map the latent representation back to the original feature space. Compared to standard autoencoders, VAEs incorporate the probabilistic variable z as a latent learned one to approximate a prior distribution $p(z)$. The encoder network produces the mean and variance of the approximate posterior distribution $p(z|x)$, which is then used to sample a latent representation z. The decoder network then maps the sampled latent representation back to the original data space.

The objective function of a VAE as below:

$$\mathcal{L}_{VAE} = -\mathbb{E}_{q(z|x)}[\log p(x|z)] + KL(q(z|x)||p(z|x))$$

includes two terms: a reconstruction loss that measures the difference between the reconstructed signal and the input, and a regularizer which encourages the learned posterior distribution to be close as possible to the prior. The regularizer is implemented using the Kullback-Leibler (KL) divergence function between the approximate posterior and the prior distributions.

where $\mathbb{E}_{q(z|x)}[\log p(x|z)]$ our the reconstruction loss, and $KL(q(z|x)||p(z))$ is the KL divergence between the approximate posterior distribution and the prior distribution. By optimizing this objective function, the VAE will learn to encode and decode complex high-dimensional data and be able to generate new samples from the learned distribution.

2.2 OpenAI's `logprobs` endpoint and candidate model

OpenAI's logprobs API endpoint is a service provided by OpenAI's language models that allows users to query the probability distribution of words word in a given context. Owing to [10] DetectGPT's work, we were able to use their functionality to extend the endpoint from merely returing log-probabilities to the entire log-probability of some candidate text and threshold. Fortunately, with the type of model that was used in the creation of our dataset (as referred in section 4.1), we were also afforded the customizability of choosing that very model to sample our data against.

3 Hypothesis

Text generated by natural language algorithms are, given some maximum likelihood estimator, the most likely body of text given their inputs. This means - given some likelihood function (and its parameters), the generated output is likely to sit at some optimum of their respective objective functions. Human- generated text (HGT) does not obey such strict rules of maxima and do not exhibit perfect function- approximable behaviour. Exploiting these two tendencies allows us learn the parameters of the text- generating distribution through some function approximator. If we can empirically verify our hypothesis, we can then use the parameters of these functions to reconstruct our data as see if it is likely to be some body of MGT or otherwise.

We propose a scoring metric $PR(x, \overline{x})$ - the ratio of probabilities between two texts as follows:

$$PR(x, \overline{x}) = \frac{\exp(\mathcal{LP}(x))}{\exp(\mathcal{LP}(\overline{x}))} = \exp(\mathcal{LP}(x) - \mathcal{LP}(\overline{x}))$$

where $\mathcal{LP}(i)$ is the log-probability of the candidate text i and $\overline{x}$ is the text generated by our VAE. If our VAE is correctly and faithfully able to capture the underlying distributions that GPTs are using to model their outputs, sending MGT through our VAE should give us a very good reconstruction - since our model should have learned the latent embedding representation of MGT - and therefore, the probability ratio would be higher than the alternate case, since HGT will reconstruct poorly when forced to be re-encoded under the distribution of an MGT. The goal to assess this model is separability - to see if the density of scores for MGTs and HGTs respectively are separated as opposed to being intermingled among one another. If we achieve separability through our scores - ie. more probability ratio scores of either type of text tend to one region of the number line, we can assert that the model has done a good job of being able learn the latent parameters of MGT and that HGT indeed cannot be modelled through the science of maximum likelihoods.

4 Methods

In order to better assess the quality of our results generated by our potential VAE, an effort was made to compare our candidate model against a pre-existing large language model, which is then fine-tuned on machine-generated text. The larger scope was also tailored to assess performance of pre-existing LLMs that predate modern GPTs and their use cases in classifying between machine generated text and human generated text.

4.1 Dataset

GPT Wiki Intro [2] is a dataset that was made to train models to build HGT v/s MGT classifiers in January 2023. The above dataset contains Wikipedia introductions and machine-generated introductions for 150000 topics. All of the data present in this dataset was generated by OpenAI's `text-curie-001` model of their GPT-3 family, which we specify in our `get_log_probability` function, enabling us to sample out candidate texts against the model itself.

4.2 spotGPT-VAE

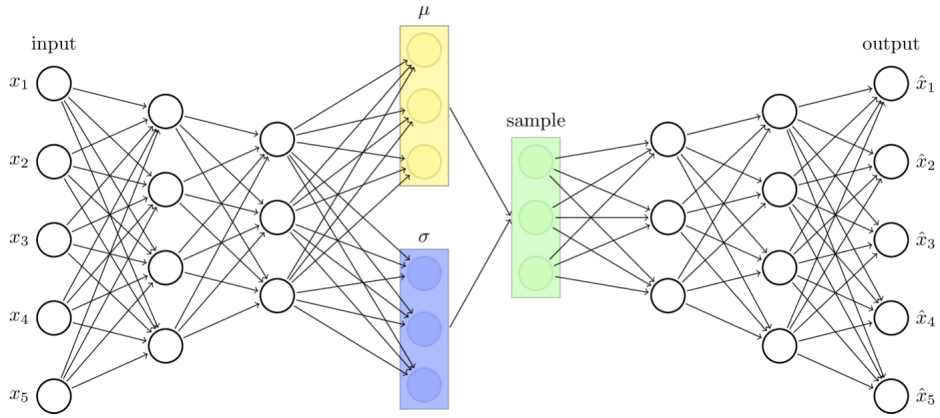

Figure 1: Standard Variational Autoencoder (source: Towards Data Science)

This subsection will discuss the parameters that we have chose for our model, along with other decisions that have been taken for the model's training. Our architecture was inspired and implemented from a Medium article from two years ago (A., Vivek, 2021) [1] and from faithful PyTorch implementations from (Bowman et. al., 2016) [4] made available from https://paperswithcode.com

Embeddings:We have chosen a 300-dimensional word embedding to convert our inputs into a representation for the VAE to learn. Once we finished tokenizing our sentences, GLoVe (Global Vectors for Word Representation) was chosen as our main embedding vocabulary since it utilizes both global and local information to capture the meaning of a word in a comprehensive fashion. By examining sparse and dense features in a corpus of text, we want to be able to see if machine-generated text has certain kinds of words used, and to test id there are strong heuristic-based associations in the body of the text.

Encoder and Decoder: Our encoder and decoder component are both long short-term memory networks with two hidden layers, each of size 256. Our encoder has three hidden layers and is a bidirectional network, while the decoder has four hidden layers, but is a unidiretional LSTM, with each layer also being of size 256. We chose an LSTM over a potential RNN framework as LSTMs have the capacity to selectively remember strongly associated information given the context, which RNNs are relatively worse at performing. Since our objective is to harness patterns across global and local features of the corpus, we opted for the LSTM approach.

Dropout: A common method to combat overfitting; utilizing the dropout layer to mask the activation of neurons should help the network approximate the true underlying function parameters. We have chosen a relatively hefty parameter - with a dropout rate of about 25%

Loss function: To better approximate the reconstruction loss component of the total objective, we use the cross-entropy error function. The benefit of using CE-loss is that when we provide the reconstruction and original input to this part of the objective function, the error will reflect how well we have reconstructed the candidate text input x.

We decided to punish the KL loss component overfitting by adjusting its contribution to the loss function with a small constant regularizer, thus causing it to decrease a a much slower rate. As a result, the distribution that we estimate is not a perfect replica of the ground truth we have. By getting a distribution that isn't a perfect memorization of the ground-truth, we will be able to have better out-of-sample performance.

The final loss function of the model is:

$$Loss(x, \overline{x}, z) = 10 * CE(x, \overline{x}) + (10^3) * D_{KL}(q(z|x)||p(z|x))$$

where $CE(x, \overline{x})$ is the cross-entropy loss between the candidate text and the reconstruction, and the D_{KL} is the KL-divergence loss term between the latent distribution and the ground truth.

Optimizer: Our chosen optimizer was the conventional `Adam` optimizer, in order to efficiently get our model to learn the ideal learning rate parameter. We chose our rate parameter to be a sufficiently small one: 10^{-4}.

4.3 Transformer-based Language Model

The T5-small model is a variant of the T5 (Transformer-based Language Model) architecture, which is a state-of-the-art natural language processing model developed by Google. T5-small is one of the smaller versions of the T5 model family, and it has approximately 60 million parameters, compared to the largest T5-3B model, which has over 3 billion parameters. The model operates on tokenized text inputs and outputs. It takes in a sequence of tokens as input and produces a sequence of tokens as output, using an encoder-decoder architecture with attention mechanisms to capture contextual dependencies between the input and output. (Raffel et al., 2020) [11].

Despite its smaller size, the T5-small model is still capable of performing a wide range of natural language processing tasks, such as language translation, summarization, question answering, and more. It has been pre-trained on a large collection of text data but can be fine-tuned on specific downstream tasks with relatively small amounts of task-specific data, which we leverage with the task to "summarize: ". We referenced Hugging Face's Google Collab notebook titled "Fine-tune a pre-trained model" to do so with the "GPT Wiki Into" data.[5]

We have leveraged a subset of the VAE's data, prefixing each human-generated text (`wiki_intro`) sentence with "summarize: " as the input text and the machine-generated text (`generated_intro`) as the target text, to fine-tune the model. We used this model to reconstruct the test data and compare its scores with the VAEs.

4.4 Approach and Pipeline

The approach of scoring occurs in four major steps. We have currently tested functionality of sentences of sizes approximating sub-or-at-200 words per training example. We enumerate a step-by-step approach as to how the pipeline will work for scoring some candidate text x.

- Once loading our spotGPT-VAE (or in the case of our control - the fine-tuned T5 model), we ensure that we have some candidate text x that we pass into our model in question.
- We expect the model to return a reconstruction of the original candidate text, named $\overline{x}$ that we use to compare against our original candidate text x
- After receiving our texts x and $\overline{x}$, we pass in both of them into our likelihood generator function, which in turn, calls the OpenAI `logprobs` endpoint.
- Having returend two scores - one for the log-probability for the candidate text x and our reconstructed text $\overline{x}$, we then take the difference of the model and raise it by the natural exponent to give us the ratio of probabilities between the two texts.

As an extension to our hypothesis, we aim to assess if the source model in question is indeed able to detect its own samples, or at best - texts which are extremely close in their reconstruction of the model's own samples. The worse the embedding-based reconstruction, the worse the scoring will be and vice-versa.

5 Results

Due to foreseen constraints with access to GPUs and cost management for both OpenAI's API and processing power use for our research, we were severely limited in our capacity to test variations of our variational autoencoder. Upon inspection of the metrics as above along with being conscious of the bias-variance tradeoff, we decided to choose the model with three encoder layers, four decoder layers and a dropout rate of ¼ as it had the best validation loss performance among the similarity (KL divergence) loss axis.

After fitting our model with the chosen hyperparameters, we inspected the training and validation of the model at each epoch to ensure our loss was both decreasing and converging toward a small number - indicating that we have learned our paramaters well without overfitting. One of our most

Table 1: Validation Loss terms of VAEs under hyperparameters

Encoder Size	Decoder Size	Dropout Rate	reconstruction loss	similarity loss
2	4	1/3	0.0022365366308590	0.616309699033898
3	3	0.3	0.0018959746761314	0.005056404183185
3	4	0.25	0.0014953296829458	0.000648323851217

severe constraints was GPU costs and access to high-RAM computing power, which forced us to have a smaller validation set. Our dataset was randomly split into 85% for training purposes and the remaining 15% for the blind validation of spotGPT-VAE. As a result of being conscious of GPU training costs, we trained our model for 25 epochs, which sufficiently allowed us to reach convergence - as below.

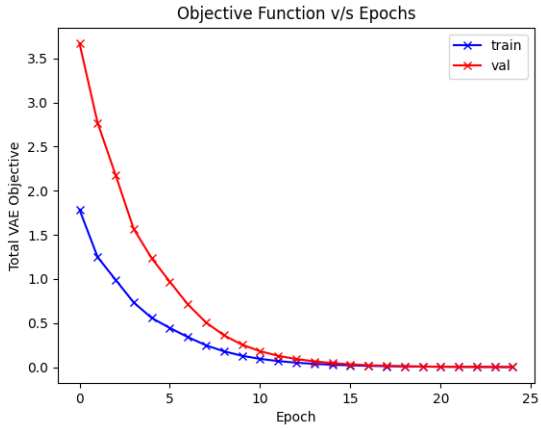

Figure 2: VAE Objective Function performance

Our KL divergence - our similarity loss between our prior and our posterior - also exhibits a similar downward trend. At earlier epochs, we see a very large loss term that trends lower and lower the more we tune the weights of our model. There is a slight increase in the value of the loss term at the fifth epoch (with both the training and validation sets), which could indicate the model having been stuck at some local optima,, before the optimizer potentially adapted its parameters for better performance. Fortunately, we achieve good loss values from the functions above, and we can show our model has reached a global optima for us to start pulling samples from.

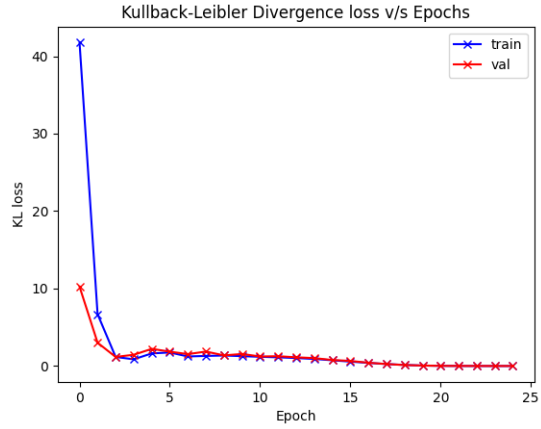

Figure 3: KL Loss Function performance

We then took 1000 randomly sampled datapoints - 500 machine generated datapoints ad 500 human generated ones - from our validation set in order to to start producing probability ratios. Once again - in the interest of the cost of OpenAI's Completion endpoint polling, we made the conscious decision to restrict the number of validation examples to a small enough number to show the distribution of how the scores would look like. We sampled 500 topics from our validation set and took their respective MGT corpuses and HGT corpuses and joined them as our candidate texts. We made the decision to have a topic-to-type split to ensure as little variation as possible in the underlying content and meaning of the candidate texts we were using to give better pairwise interpretability; since it would be highly variable to compare HGT of one topic to MGT of another.

With our candidate texts in hand, we passed them through spotGPT's VAE to recieve our reconstructed texts (via forward passes through the model), and called our functions (which called OpenAI's logprobs completion to get the probability ratios between the candidate text and its reconstruction. We performed the same process for our control, with the noticeable difference being that the model we forward-passed through in this case was the fine-tuned T5 transformer model. A visualization of the distribution of probability ratios has been made available in the figures below.

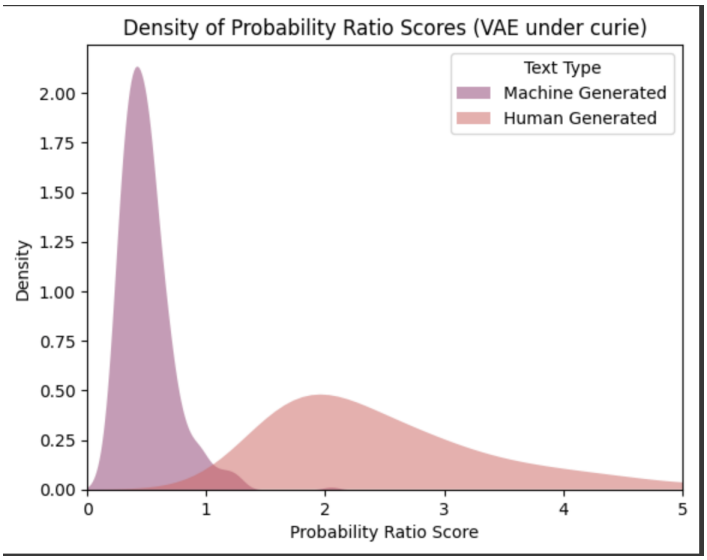

Figure 4: Distribution of probability ratios for spotGPT-VAE

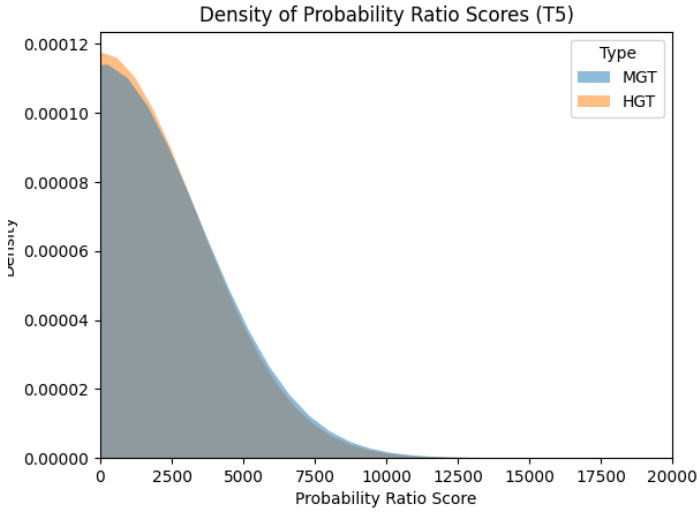

Figure 5: Distribution of probability ratios for T5 transformer

By showing the the distribution of probability ratios as a density chart, we are visually able to assess whether or not we have achieved separability between scored of MGTs and HGTs. Based on our experiments that we have run with spotGPT's VAE - we have achieved good separability between MGT scores and HGT scores. Most HGT scores tend toward zero and smaller values with a much larger density, while MGT scores are much more wide in their distribution. The highest HGT score does not exceed 1.5 in our shrunk test set (with a small but notable outlier cluster around 2). With respect to the MGT distribution, even though we have a higher amount of variance, we also see that there is very little overlap between the two distributions, and the bulk of the data

In contrast, our control T5 model behaved as exactly as expected - with very poor separability since the model would create exceptionally poor reconstructions of machine-generated text. A score close to zero implies that there is a large margin of reconstruction from the original text. Irrespective of the kind of text that we reconstructed, most of the datapoints have exceptionally low probability scores and the distributions of probability scores are fundamentally the same. This pipeline is unable to draw the boundary between MGT and HGT.

To connect with our original hypothesis in section 3, the test set distribution seems to agree with our original hypothesis - that the distribution of probability ratio scores is poor or candidate HGTs and is higher for candidate MGTs given their respective reconstructions.

6 Discussion

Our approach has several limitations that should be considered when interpreting the results.

Firstly, the use of Hugging Face's "GPT-wiki-intro" dataset has several drawbacks, including a limited domain coverage due to it primarily being sourced from Wikipedia, a sample size of 150,000, and a lack of diversity. While the dataset provides a useful starting point for language modeling tasks on the operational scale available to us, its limitations may impact the generalizability of our findings to other datasets or models. Due to limited computing power, we were unable to train on a much larger dataset without sacrificing the flexibility of the model's complexity. Using an 85-15 train-test split of the data on this VAE, we were able to train the model hyper-parameters overnight successfully.

Conversely, in the process of fine-tuning the T5-small model, we had to cut down to a sample of only 1,500 observations in total. We could only afford 1 epoch with a batch size of 1 due to limited computational resources, as any batch size greater than this would cause the program to crash and one epoch ran in 2 hours. Using a small number of epochs and a small batch size may limit the ability of the model to learn the features of the training data, leading to suboptimal performance on the test data. Additionally, the small batch size slowed down the training process significantly as well as increased the risk of overfitting.

Finally, the limited number of observations scored using the openAI API is another notable limitation of our study. Due to the cost of each API call, we could only score 450 observations reconstructed by the T5 model and 1000 observations reconstructed by the VAE. This limits the statistical power of our analysis and raises concerns about the implications of our findings on a wider range of NLP data.

While our results have significant implications on Natural Language Processing through insights ranging from evaluating the quality of generated text, detecting the propagation of false or misleading information and, preventing plagiarism, its limitations should be taken into account when interpreting the results. Natural language data must be diverse in language, culture, genre, demographic, length, quality, and far more. That being said, future studies should consider using larger datasets and more powerful computational resources to overcome some of these limitations and provide a more comprehensive evaluation of the spotGPT scoring metric. It is also worth noting that the human generated text scores produced a much tighter distribution than those of the machine generated text, with clearly separable means between the two classes. Defining probability of text being machine generated with less variability could also be means for future exploration.

Given the current pipeline's inability to draw a boundary between MGT and HGT, another potential exploration for future reference could be to incorporate a linear classifier that is specifically designed to optimize true positives and true negatives simultaneously. In the context of detecting plagiarism or authenticity, though, false negatives could be more costly than false positives because classifying plagiarized content as authentic would allow plagiarism to go undetected, potentially leading to academic or legal consequences. Classifying authentic content as plagiarized may lead to a temporary

inconvenience, such as having to manually verify the authenticity of the text, but it is a small price to pay for ensuring the credibility and integrity of the content. Such a linear boundary should take this into consideration, or be weighted in accordance to the applications targeted for the given model.

References

[1] Vivek A. Generating new sentences from an input sentence using variational autoencoders, Jul 2021.

[2] Aaditya Bhat. Gpt-wiki-intro (revision 0e458f5), 2023.

[3] Hareesh Bahuleyan, Lili Mou, Olga Vechtomova, and Pascal Poupart. Variational attention for sequence-to-sequence models, 2018.

[4] Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space, 2016.

[5] Hugging Face. Training a language model with transformers in pytorch. `https://colab.research.google.com/github/huggingface/notebooks/blob/main/transformers_doc/en/pytorch/training.ipynb`, Accessed: 2023.

[6] Le Fang, Chunyuan Li, Jianfeng Gao, Wen Dong, and Changyou Chen. Implicit deep latent variable models for text generation, November 2019.

[7] GPTZero. GPTZERO FAQ. `https://gptzero.me/faq`. Accessed: Apr. 27, 2023.

[8] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[9] Stephanie Lin, Jacob Hilton, and Owain Evans. TruthfulQA: Measuring how models mimic human falsehoods, May 2022.

[10] Eric Mitchell, Yoonho Lee, Alexander Khazatsky, Christopher D. Manning, and Chelsea Finn. Detectgpt: Zero-shot machine-generated text detection using probability curvature, 2023.

[11] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2020.

[12] Stanislau Semeniuta, Aliaksei Severyn, and Erhardt Barth. A hybrid convolutional variational autoencoder for text generation, 2017.

[13] Wenlin Wang, Zhe Gan, Hongteng Xu, Ruiyi Zhang, Guoyin Wang, Dinghan Shen, Changyou Chen, and Lawrence Carin. Topic-guided variational autoencoders for text generation, 2019.

[14] Rowan Zellers, Ari Holtzman, Hannah Rashkin, Yonatan Bisk, Ali Farhadi, Franziska Roesner, and Yejin Choi. Defending against neural fake news, 2020.