Over-Complicating GPT

Stanford CS224N {Custom} Project

Daniel Yang
Department of Computer Science
Stanford University
`dy92634@stanford.edu`

Abstract

In this paper, I combine controllable text generation (CTG) with unsupervised learning to reword or paraphrase text into either more complex or simpler versions of itself. Being able to change the complexity of sentences marks a giant leap forward in generative models by enabling better adaptation for generative NLP tasks. Due to the lack of labeled text datasets for text difficulty, I will utilize unsupervised training with a custom loss function and a modified generation process to achieve my goals of reconstructing source sentences with either higher or lower complexity. My custom functions attempt to control the word lengths to achieve the CTG goals. Ultimately, this research shows substantial results in controlling text difficulty through custom loss and generate functions within the context of subsupervised learning. However, modifying the generate function compared to the loss function results in better fluency and higher quality results.

1 Introduction

Controllable text generation (CTG) is a rapidly expanding field of natural language processing (NLP). Recently, this topic has drawn substantial attention for its mass research potential. Exploring the intricacies of CTG is vital for the development and performance of advanced generative models (Zhang, 2023). Being able to control specific sentence parameters such as style, sentiment, length, difficulty, etc. enables further enhancements for generated text accessibility and adaptability. The advent of large language models has provided new dimensions and capabilities for CTG, but challenges for efficient training and model accuracy persist (Martínez-Murillo, 2023). In this paper, we focus on controlling language difficulty–consisting of word difficulty, sentence structure, and language syntax–in generative word models. Specifically, we aim to create a generative model that can replicate input English sentences with varying levels of complexity. The potential uses or benefits for these generative models are plentiful; languages can either be simplified (enabling wider accessibility) or increased in complexity (for academic journal purposes). I developed two techniques for CTG enhancements (one utilizing a custom loss function and the second modifying the generation process) within the generative models. The goals for these modification additions is to increase/decrease the frequency of longer words (seen as more complex). These incentives would theoretically dictate the difficulty of the generated text. This paper will evaluate the accuracy metrics of both the methods outlined above and compare them to each other as well as existing studies on unsupervised paraphrasing. Additionally, I have also developed several new unique metrics to evaluate language complexity in any given generated text.

2 Related Work

My work focuses on the GPT2 model for generation, but the methodologies and code from this paper can be easily applied to other language models. The following are papers that guide and provide context for some of my approaches.

- (Hegde and Patil, 2020)[1] This paper also utilizes the GPT2 model for unsupervised paraphrasing. The techniques in Hedge paper used to preprocess the training data were partially adopted in my implementation. While Hedge uses internet posts as its training set, I've utilized a larger corpus from Wikipedia sentences with additional preprocessing steps.
- (Dolan et al., 2004)[2] This paper provides several more benchmarks towards the accuracy of generated text. The paper also utilizes an unsupervised approach for reconstructing sentences and uses Alignment Error Rate (AER) to score its paraphrasing quality. Both Dolan and Hedge[1] aim to reword input text, but my research also includes adding a CTG component to the output.
- (Zhou et al., 2023)[3] A paper that explores CTG through utilizing an augmented corpus. The authors are able to label their target sentences for their desired CTG traits and fine tune the model on this augmented corpus. My research differs from Zhou in several ways. Firstly, I use a traditional unlabeled corpus similar to Hedge[1] and Dolan[2]. All the CTG modifications were done in the decoder and loss function. Secondly, my research focuses on sentence difficutly, which was not one of the tasks covered in the Zhou paper.
- (Gao et al., 2018)[4] This paper aims to achieve a similar goal towards difficulty CTG. The paper attempts to generate questions and answer pairs of varying difficulty. To implement the difficulty control, the paper creates a lookup table that maps difficulty labels to a variable. The variable is combined with the final hidden state within the encoder to initialize the decoder. Furthermore, the model is fed a labeled dataset which was labeled using the SQuAD evaluation [Rajpurkar et al, 2016]. My research uses unsupervised learning with an unlabeled dataset and uses a different modification in the decoder. My work marks a step forward by combining the accurate difficulty generation with large unlabeled datasets.

3 Approach

To increase/decrease our model's output sentence complexity, I chose to increase the frequency of longer words. I fine-tuned a GPT2 model using the hugging face transformers library and the following sections outline the key milestones and file descriptions for all the code I wrote which can be found at `https://github.com/danielyang10237/controllable-text-generation`

3.1 Data Preprocessing (dataparser.py)

The first step would be to format our text corpus to be fed into our model. Our unsupervised technique requires constructing both a source and target sentence. The goal for our model is to reconstruct the target sentence from the source. To ensure the reconstruction process varies the word choice and word ordering, we intentionally corrupt the target sentence to get the source sentence. We remove all stop words and occasionally replace words with synonyms using the wordnet interface from Natural Language Toolkit (NLTK) library. We also by random chance shuffle the word order for certain words.

3.2 Model Training (model.py)

Before training, I removed any inputs that exceed our model's maximum token length. I used the pretrained weights and the GPT2 tokenizer provided by Hugging Face (gpt, 2019). The target and source sentence lines from our preprocessing are appended as follows:

input_tensor = token_start + source_line + token_delimiter + target_line

$token\_start$ and $token\_delimiter$ are unique character sequences that serve to both separate the key sentence parts and tell the model this sentence should be reconstructed.

I also defined a function pack_tensor borrowed from (Kim, 2023). This function essentially packs an input tensor with as many input texts as possible to maximize training speed.

The core component powering the CTG is the custom loss function which penalizes the model for either creating longer or shorter words.

$$H(y, \hat{y}) = -\sum_{c=1}^{C} y_c \log(\hat{y}_c) \tag{1}$$

I took the default cross entropy loss function (equation 1) and added an additional penalty for generating shorter/longer words.

$$H(y, \hat{y}) = -\sum_{c=1}^{C} y_c \log(\hat{y}_c) \pm \sum_{j=1}^{J} P * max(0, z_j - T) \tag{2}$$

Equation 2 shows the final loss function applied to the training. The equation applies a linearly scaled reward/penalty for the lengths of predicted token probabilities. C represents all the potential logits or tokens in our vocabulary. y_c is the actual probability value for token c and $\hat{y_c}$ is the predicted value for token c. j represents the J highest logits from our model predictions. P is the penalty scalar, z_j is the decoded token character length from our best token candidates, and T is the minimum threshold for acceptable decoded word length. We apply the adjusted loss every time a high predicted token exceeds our threshold for word length.

3.3 Text Generation (generate.py)

After reading in prompts, the model will output a reconstructed or paraphrased version of the source in accordance with the CTG goal. We start by constructing our input tensor. First, we take our prompt and applying a version of the scrambling described in subsection 4.1, then we append our $token\_start$ and $token\_delimiter$ to either side. We send this input tensor through the encoder before our softmax (Galassi et al., 2020). For each of the next predicted tokens, we apply a custom sampling function.

$$P_i = \frac{e^{\frac{l_i}{T}}}{\sum_{j=1}^{n} e^{\frac{l_j}{T}}} \tag{3}$$

Equation 3 shows the default probability adjustment for all logits before sampling the next token using softmax with temperature. We want to incentivize the selection of longer words and advanced punctuation.

$$P_i = \frac{e^{\frac{l_i * (1 \pm P * z_i \pm C_i)}{T}}}{\sum_{j=1}^{n} e^{\frac{l_j * (1 \pm P * z_j \pm C_j)}{T}}} \tag{4}$$

Equation 4 represents our modified softmax function that modifies our distribution to either incentivize or disincentive complex words or symbols. l is the logits outputted by our model used to predict the next token. z is the length of the decoded word and P is the scaling factor. C is a positive constant if the decoded token contains an element from our advanced punctuation set and is zero otherwise. After applying our custom modifications to the logits, we randomly sample the next token using the torch multinomial function. Thus, longer and more complex tokens have a higher chance of being selected to be outputted.

3.4 Model Evaluation (evaluate.py)

This file contains all the functions used to evaluate and score my model. The functions can be divided into two groups: functions that evaluate the accuracy and functions that evaluate the difficulty. We have two functions to calculate the ROUGE scores and the BERT scores between our generated text and reference text. The reference text was extracted from Microft's paraphrase text corpus which provides the source text and the corresponding human paraphrased text (preprocessing code found in **paraphraseparser.py**) (Dolan, 2016). In our ROUGE and BERT, functions, I compare the model generated text to the human written ones. The other functions explore the difficulty of generated text. One function calculates the average word length of our generated text, and a second function which counts the frequency of words that are not classified as "simple" (lexicon stored on a set).

4 Experiments

This research required creating three different models and comparing the evaluation metrics between the three. One model serves as the default, with no custom modifications to serve as an additional baseline for our difficulty generation comparisons. The following sections go into depth about our experimental process.

4.1 Data

The text used for training was sourced from wikipedia via Kaggle (Ortman, 2018). Each sentence from the wikipedia page is one line in the .txt file which translates to one input tensor into the model training. There are 7.8 million sentences in entire corpus but only a twentieth of which was used for training. Since all sentences are listed alphabetically, I selected every twentieth sentence to be fed into training. All lines are padded to the max token length which I currently set at 64. The text preprocessing necessary for training is outlined in section 4.1.

4.2 Evaluation method

There are two categories of evaluation methods I used for this research. Accuracy metrics are used to ensure the model quality doesn't deviate too far from the default model when trying to integrate CTG. Difficulty metrics actually evaluate the difficulty of the generated text compared to the default, which prove the efficacy and effectiveness of the CTG custom components.

Accuracy evaluation Our accuracy evaluations compare our generated text to human written paraphrased references provided by the Microsoft research corpus (Dolan, 2016). Every line in the corpus is fed through our model and cross referenced against the provided human translation/paraphrase. For the purposes of this research, I am not too concerned with the accuracy scores. I'm more focused on the difficulty of the generated text and not how well it correlates with provided human rewording. However, it is still important to ensure the model still performs the task at hand to a reasonable degree of quality.

ROUGE-1: ROUGE-1 counts the overlapping unigrams or single words between our model outputs and the human references. The score is calculated by finding the frequency of unigrams in the reference text also present in the generated text. The score ultimately captures the model's ability to retain important words during translation.

ROUGE-2: ROUGE-2 counts the overlapping bigrams or two word sequences between our model outputs and the human references. We count the number of two sequence word grams from the reference text that also appears in our generated text. Naturally, we expect a lower ROUGE-2 score compared to ROUGE-1 because it's more stringent on the word positioning.

ROUGE-L: ROUGE-L finds the longest matching subsequence of words between our generated text and reference text. The longest match subsequence is a series of words that appear in the same order, but not necessarily adjacent to each other. Evaluating our model with ROUGE-L ensures the fluency and flow of our generated sentences.

BERT SCORE: Our BERT score evaluation for this model consists of three components: precision, recall, and F1-score. Our precision (p) is found by measuring the maximum similarity the tokens in the generated has with the tokens in the reference text, which captures the semantic difference. Recall (R) is examining the maximum similarity the reference text tokens have with the generated text tokens. F1-score is ultimately the accumulation of both precision and recall.

Difficulty evaluation *WORD_LEN:* Word_len is a custom evaluation metric where the average word length within a sentence is measured. Longer words correlate with more advanced vocabulary, which would indicate more complex language.

comm_freq_gen: comm_freq_gen is another custom evaluation metric that comprises of measuring the amount of not simplistic words found inside a generated text. I created a set of basic comm_freq_gen words and this metric calculates the frequency of words in the generated text that isn't found in the set

Manual: Lastly, I utilized manual and personal evaluation to estimate the difficulty of the generated text compared to the source/original. The table below contains some samples.

Source text	Despite feeling tired, she went for a run.
Generated text	Despite being tired and feeling exhausted, he went on to run.

Table 1: Higher complexity generation

Source text	I will lend you my car provided that you return it on time.
Generated text	I'll give you a car that will be lent to you as soon as I can.

Table 2: Lower complexity generation

4.3 Experimental details

I trained a total of three models, one using the provided loss function, one with the custom loss function to incentivize longer words and the third with the custom loss function to generate shorter words. Our training process consists of five epochs in total, each one running through the entire training set with batch sizes of 16. My learning rate is set at 0.001 with the initial warmup steps set to 200. The training time for each of our models took around three hours.

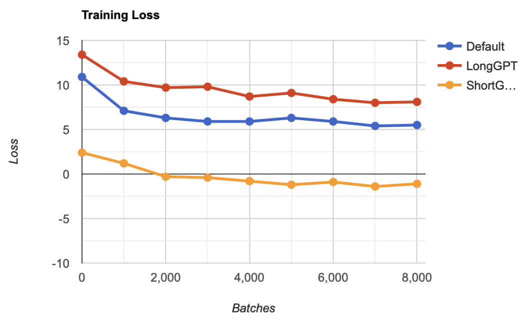
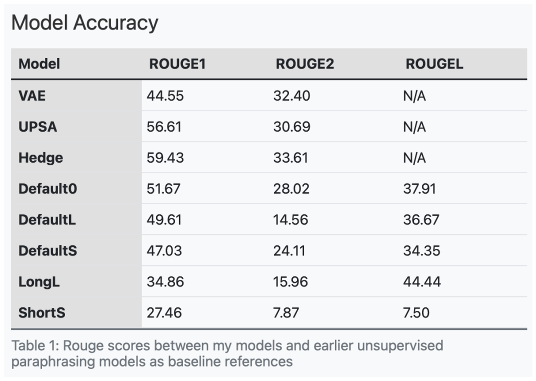


Figure 1: Training of three different loss functions. Long incentivizes longer word generation. Short promotes shorter word generation

4.4 Results

For our accuracy measures on ROUGE and BERT, we compare our models to existing unsupervised models. Our baselines are VAE (Bowman et al., 2016), UPSA (Liu et al., 2020), and Hedge[1] (Hegde and Patil, 2020). I have assembled five models: Default0, DefaultL, DefaultS, LongL, and ShortS.

Default0: Vanilla model, unmodified softmax loss during training
DefaultL: Unmodified softmax loss during training combined with custom generate function for complex sentences
DefaultS: Unmodified softmax loss during training combined with custom generate function for simpler sentences
LongL: Custom softmax loss (training) and custom text generation functions for complex sentences
ShortS: Custom softmax loss (training) and custom text generation functions for simpler sentences



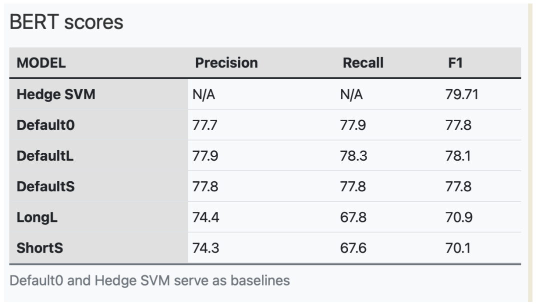
Model Accuracy

Model	ROUGE1	ROUGE2	ROUGEL
VAE	44.55	32.40	N/A
UPSA	56.61	30.69	N/A
Hedge	59.43	33.61	N/A
Default0	51.67	28.02	37.91
DefaultL	49.61	14.56	36.67
DefaultS	47.03	24.11	34.35
LongL	34.86	15.96	44.44
ShortS	27.46	7.87	7.50

Table 1: Rouge scores between my models and earlier unsupervised paraphrasing models as baseline references

Figure 2: Table 3

For our BERT scores, we referance a NB-SVM (trigram) model provided by Hedge as a baseline.



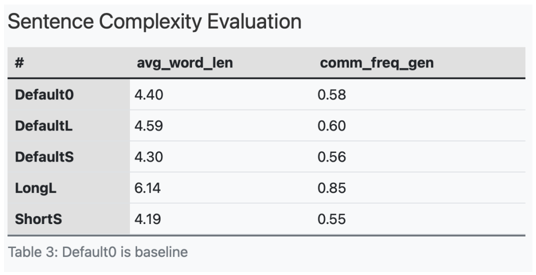
BERT scores

MODEL	Precision	Recall	F1
Hedge SVM	N/A	N/A	79.71
Default0	77.7	77.9	77.8
DefaultL	77.9	78.3	78.1
DefaultS	77.8	77.8	77.8
LongL	74.4	67.8	70.9
ShortS	74.3	67.6	70.1

Default0 and Hedge SVM serve as baselines

Figure 3: Table 4

The following shows our difficulty evaluation results. Our default0 model without any modifications serves as our baseline. The table shows the effects the custom generation and loss functions have on our outputted text.



Sentence Complexity Evaluation

#	avg_word_len	comm_freq_gen
Default0	4.40	0.58
DefaultL	4.59	0.60
DefaultS	4.30	0.56
LongL	6.14	0.85
ShortS	4.19	0.55

Table 3: Default0 is baseline

Figure 4: Table 5

Comments I didn't expect the ROUGE scores for the custom loss function models to be so low. This indicates our paraphrasing deviates too far from the prompt which suggests my corrections for sentence difficulty was too extreme. I was also surprised by the custom loss function model's performance on the sentence complexity. The custom loss function does exceptionally well on creating more complex sentence words but has almost no effect on generating less complex sentences. This indicates there's either more potential in up scaling sentence complexity or the penalties for generating long sentences were not strong enough.

5 Analysis

From table 3, we can see the ROUGE scores obtained from our unmodified training process are comparable to the previous unsupervised learning paraphrasing models. This indicates my approach for pre-processing data and my training process was robust and adequate enough to perform the initial translation task at hand. Any significant differences in accuracy metrics between my models and the baseline is largely due to the design of the model itself. We see a drop in ROUGE scores (table 4) when we switch to our custom trained models. This isn't an inherently a detriment, as the goal of this research paper focuses on implementing CTG. In the event we are trying to manipulate model output, the lower ROUGE states our generated output words are different, which could be a positive sign.

Our BERT scores also see a dip for our custom trained functions, but it's not as significant as ROUGE. BERT measures the semantic similarities in the model outputs. A lower BERT score for our custom trained models suggests these models lose some semantic values in their translations (which can be visually observed in 4.2 and 4.2). This means that the implementation of CTG during training trades off with some of the sentence semantic accuracy. When trying to manipulate the difficulty of the target sentence, we naturally lose some precision and detail.

In regards to the difficulty of the output text, we do see significant differences from our custom functions. Our custom loss function has a more pronounced effect on manipulating our sentence complexity than our custom generate function. This is likely due to a combination of higher rewards/penalties for the loss function compared to the generation sampling and the generation sampling considering less tokens, which narrows its options. Combining both the custom trained model with our custom generate function produced the most extreme results, which indicates both can work hand in hand with each other. Our default0 model serves as the baseline, which is the vanilla GPT2 model without any modifications.

Our custom generation function also appears to reduce the accuracy of our CTG models compared to baseline models from other studies (but less so than our custom training loss). When trying to reword sentences with more than one clause, our model struggles to retain all relevant information when reconstructing the output. Thus, we lose key facts and details. This dip in performance can be largely explained by the use of beams during generation. Beams, which consider future potentially generated tokens after our immediate next token, drastically improve the model's ability to produce coherent non repeating phrases. While models from the other studies are able to capitalize on this tool, my custom generate function omits this feature.

I also observed random sequences of symbols such as "!!!" or "???@" when trying to generate longer words. This would artificially inflate the average word length in our outputted text and is a result of the model constantly being rewarded for appending extra characters to pad its words. Further measures to normalize this behavior should be needed. My results were best using the default unmodified training with the custom generate function. The fluency is preserved while achieving the task at hand. This is likely because choosing the next token to generate doesn't affect the original flow/fluency of the model but still increases the chances longer and more difficult words are chosen. My recommendations would be to focus more on how the model chooses the next token during generation for this CTG task.

6 Conclusion

This research shows CTG on sentence difficulty can be applied to unsupervised learning techniques through both the training and generating process. Some studies have been conducted on controlling the difficulty of generated text, but only through a labeled corpus with supervised training. This study shows the effectiveness of training CTG difficulty models with unlabeled datasets using unsupervised reconstruction techniques.

When trying to linearly penalize/reward the model for a specific CTG task, it's easy to overcompensate, leading to incoherent and unrelated text generation. One big limitation of this study's methods is the incoherent and garbage padding and words the model learns to append to sentences. I often observed unrelated long texts being appended to outputs and the omission of certain key details in the original source text when utilizing a custom loss function. Consequently, we observe much higher accuracy and fluency using the custom generate function with default training. Changing the generation process rather than the training process produces better results across all evaluation data sets.

The outcomes of the adapted softmax loss equation used in this research can likely be drastically improved and used for further research. The linearity scaling I used is somewhat crude and prone to extreme manipulations. This prevents the model from converging faster and pushes bad generation practices. With labeled datasets on text generation that provide both easy and difficult matches for sentences few and far between, exploring the possibilities of unsupervised CTG becomes ever more important.

References

2019. Open source gpt2. Hugging Face.

Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew Dai, Rafal Jozefowicz, and Samy Bengio. 2016. Generating sentences from a continuous space. In *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*, pages 10–21, Berlin, Germany. Association for Computational Linguistics.

Bill Dolan, Chris Quirk, and Chris Brockett. 2004. Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources. In *COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics*, pages 350–356, Geneva, Switzerland. COLING.

William Dolan. 2016. Microsoft research paraphrase corpus.

Andrea Galassi, Marco Lippi, and Paolo Torroni. 2020. Attention in natural language processing. *IEEE transactions on neural networks and learning systems*, 32(10):4291–4308.

Yifan Gao, Lidong Bing, Wang Chen, Michael R Lyu, and Irwin King. 2018. Difficulty controllable generation of reading comprehension questions.

Chaitra Hegde and Shrikumar Patil. 2020. Unsupervised paraphrase generation using pre-trained language models. Online. arXiv preprint arXiv:2006.05477.

Sean Kim. 2023. Npc-gpt — an exploration of large language models in video games. Medium.

Xianggen Liu, Lili Mou, Fandong Meng, Hao Zhou, Jie Zhou, and Sen Song. 2020. Unsupervised paraphrasing by simulated annealing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 302–312, Online. Association for Computational Linguistics.

Iván Martínez-Murillo. 2023. Commonsense knowledge and controllable techniques for an effective and efficient approach to text generation.

Mike Ortman. 2018. Wikipedia sentences corpus. Online. kaggle.

Haolin; Li Shaoyu; Zhou Ming; Song Dawei Zhang, Hanqing; Song. 2023. A survey of controllable text generation using transformer-based pre-trained language models. Online. ACM Computing Surveys 56.3.

Wangchunshu Zhou, Yuchen Eleanor Jiang, Ethan Wilcox, Ryan Cotterell, and Mrinmaya Sachan. 2023. Controlled text generation with natural language instructions. In *International Conference on Machine Learning*, pages 42602–42613. PMLR.