Apply watermarking algorithm to LLM-based text summarization model

Xuefei Jiang
School of Computer Science, University College Dublin, Dublin, Ireland
sc.xfjiang@gmail.com

1 Introduction

In this experiment, the primary focus is on two main aspects:

1. The development of an abstractive summarization model based on the T5 architecture [2].
2. The application of the watermarking algorithm as described by [1], with a subsequent discussion on its effects on the performance of the original summarization model.

2 Method

The abstractive summarization task inherently aligns with the sequence-to-sequence paradigm, making the encoder-decoder LLM a natural selection. Consequently, the T5 model is chosen. Different variants of the T5 model, along with their corresponding numbers of parameters, are depicted in Table 1.

model	#parameters
Tiny	16M
Mini	31M
Small	60M
Base	220M
Large	738M
3B	3B
11B	11B

Table 1: Different versions of the T5 model

I have two A100 (40G) GPUs without NVLink or PCIE connection. I aimed to employ the largest feasible models to harness the full capabilities of my GPUs. The suitable hyper-parameters for this experiment and the corresponding GPU memory usage are outlined in Table 2, which closely approaches the limit of my devices.

model	batch size	#parameters
T5 Base	32	39507MB
T5 Large	8	35357MB

Table 2: GPU memory usage

The developed code successfully works on both datasets:

- CNN-DailyMail News Dataset
- News Summary Dataset

The core statistics of the two datasets are illustrated in Table 3.

model	CNN-DailyMail News	News Summary
# train instances	287113	3611
# test instances	11490	903

Table 3: Dataset statistics

However, I have found that conducting experiments on the CNN-DailyMail News Dataset is significantly more time-consuming compared to the News Summary Dataset, especially when using the T5-large model. At the time of writing this report, the experiments on the CNN-DailyMail News Dataset are still in progress. Therefore, for the purpose of this report, I will only present and discuss the results obtained from experiments that are already finished.

The watermarking algorithm has been incorporated into the T5-based summarization model, serving as a LogitsProcessor to adjust the output probabilities of the language model. Core Python files from the repository A Watermark for Large Language Models were included in my repo as modules to apply the watermarking algorithm.

3 Results

Table 4 and 5 illustrates how watermarking affects the performance of the summarization model. As the results indicate, the application of watermarking unsurprisingly lowers the performance. This reduction in effectiveness is attributable to the watermarking algorithm's adjustment of the output probabilities of the language model. By doing so, it constrains the choices available to the language model, thus negatively impacting its performance.

In addition to watermarking, an investigation into how model size influences summarization performance was conducted. The findings reveal that a larger model size enhances the performance of the summarization.

	ROUGE-1	ROUGE-2	ROUGE-L
T5-base without watermarking	0.4832	0.2642	0.3631
T5-base with watermarking	0.4616	0.2321	0.3345
T5-large without watermarking	**0.4901**	**0.2697**	**0.3632**
T5-large with watermarking	0.4780	0.2401	0.3413

Table 4: Comparison of ROUGE scores for different models on News Summary Dataset

	ROUGE-1	ROUGE-2	ROUGE-L
T5-base without watermarking	still training	still training	still training
T5-base with watermarking	still training	still training	still training
T5-large without watermarking	still training	still training	still training
T5-large with watermarking	still training	still training	still training

Table 5: Comparison of ROUGE scores for different models on CNN-DailyMail News Dataset

However, although the results presented above are consistent with expectations, there are some aspects that remain perplexing. For instance, in all experiments, the training converges very early, as illustrated in Figure 1, indicating that the model does not extract as much information from the substantial volume of data as anticipated. Several possible reasons for this, from my perspective, include:

- A bottleneck in the model itself, which may need to be refined.
- The choice of unsuitable hyperparameters.

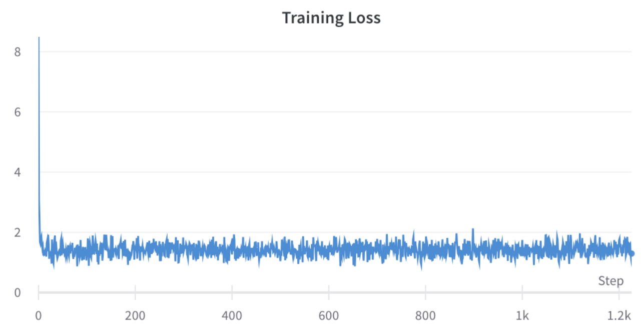

Figure 1: train loss

Appendix: Watermark Algorithm

Warkmarking is the process of embedding signals into generated text that are invisible to humans but can be identified algorithmically by watermark detection algorithms. In this section, I will discuss the algorithms provided in [1].

Algorithm 1 Text Generation with Hard Red List

```
Input: prompt, $s^{(-N_p)} \cdots s^{(-1)}$
for $t = 0, 1, \cdots$ do
    1. Apply the language model to prior tokens $s^{(-N_p)} \cdots s^{(t-1)}$ to get a probability vector $p^{(t)}$ over the vocabulary.
    2. Compute a hash of token $s^{(t-1)}$, and use it to seed a random number generator.
    3. Using this seed, randomly partition the vocabulary into a "green list" $G$ and a "red list" $R$ of equal size.
    4. Sample $s^{(t)}$ from $G$ , never generating any token in the red list.
end for
```

Figure 2: Algorithm 1

In Algorithm 1, given the prompt $s^{(-N_p)} \ldots, s^{(-1)}$, a probabilistic language model is employed to predict the subsequent word in a sequential manner. However, at each time step, the probability of a certain group of words (termed as RED LIST) is manually set to zero, ensuring they never appear at that specific time step. Conversely, only a select portion of the vocabulary (referred to as the GREEN LIST) is allowed to appear. This approach essentially embeds statistical features within the model, which can subsequently be identified by a detection algorithm.

The detection of watermarks operates entirely within the framework of statistical hypothesis testing, largely due to the fact that we're adjusting the probability distribution of words at each timestep. For a text sequence of the same length N, a natural language generator would breach the GREEN/RED LIST rule with a probability of 50%, while a watermarked text generator would not. Consequently, the expected number of words that break the GREEN/RED LIST rule should be $N/2$. We can then establish a one-sample z-test to test this expected value (with variance unknown). The testing statistic is as follows:

$$T(x) = \frac{\sqrt{n}\left(\bar{X} - \mu_0\right)}{S} \tag{1}$$

where

- $n = N$
- $\bar{X}$ is the number of words that breach the GREEN/RED LIST rule.
- $\mu_0 = N/2$
- S is the sample variance (however, in the original paper, S is set to the expected variance $N/4$).

Then, we can use this statistic to test if the text sequence is generated by a watermarked text generator. Certainly, we can change the proportion 50% to another value to give the text generator more options to improve the quality of the generated text.

Algorithm 2 Text Generation with Soft Red List

```
Input: prompt, $s^{(-N_p)} \cdots s^{(-1)}$
       green list size, $\gamma \in (0, 1)$
       hardness parameter, $\delta > 0$
for $t = 0, 1, \cdots$ do
    1. Apply the language model to prior tokens
       $s^{(-N_p)} \cdots s^{(t-1)}$ to get a logit vector $l^{(t)}$ over
       the vocabulary.
    2. Compute a hash of token $s^{(t-1)}$, and use it to
       seed a random number generator.
    3. Using this random number generator, randomly
       partition the vocabulary into a "green list" $G$ of
       size $\gamma|V|$, and a "red list" $R$ of size $(1-\gamma)|V|$.
    4. Add $\delta$ to each green list logit. Apply the soft-
       max operator to these modified logits to get a
       probability distribution over the vocabulary.
```

$$\hat{p}_k^{(t)} = \begin{cases} \frac{\exp(l_k^{(t)}+\delta)}{\sum_{i\in R}\exp(l_i^{(t)})+\sum_{i\in G}\exp(l_i^{(t)}+\delta)}, & k \in G \\ \frac{\exp(l_k^{(t)})}{\sum_{i\in R}\exp(l_i^{(t)})+\sum_{i\in G}\exp(l_i^{(t)}+\delta)}, & k \in R. \end{cases}$$

```
    5. Sample the next token, $s^{(t)}$, using the water-
       marked distribution $\hat{p}^{(t)}$.
end for
```

Figure 3: Algorithm 2

Algorithm 2 is a more advanced version of Algorithm 1, but the methodology remains unchanged. Instead of simply setting all probabilities of words in the RED LIST to zero, Algorithm 2 increases the probability of words in the GREEN LIST. Such manipulation of the probability distribution still allows for the detection algorithm to identify watermarked texts.

When I first read this paper, I was more interested in the detection of watermark. However, I noticed that the detection algorithm relies heavily on familiarity with the watermarking algorithm's specifics - for instance, the hashing function utilized to create the RED/GREEN list. This concept doesn't quite sit right with me, given it implies the need for additional information for the detection algorithm to function effectively. I am optimistic that future developments will bring forth a detection algorithm capable of discerning whether a text has been machine-generated, without the need to comprehend the intricacies of the watermarking algorithm. In my view, this back-and-forth dynamic of text generation and detection will persist until we arrive at true AGI. Just brainstorming.

References

[1] J. Kirchenbauer, J. Geiping, Y. Wen, J. Katz, I. Miers, and T. Goldstein. A watermark for large language models. *arXiv preprint arXiv:2301.10226*, 2023.

[2] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.