CS-433: EPFL Machine Learning
STLM: Steganography in Text using Language Models

Philipp Eisenhardt, Younes Belkada, Mark Tropin
Advisor: Prakhar Gupta

I. Introduction

Steganography is the practise of hiding a secret message (image, text) inside another message. The most common application of that is in image, where a secret image containing a message inside is hidden in the original image.

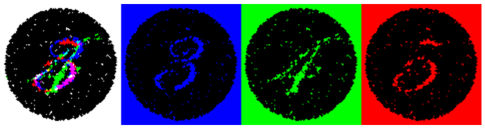

Figure 1. The same image viewed by white, blue, green and red lights reveals different hidden numbers.

In recent years, we have seen significant advances in the field of text generation using Deep Learning. Such models take some preconditioning text as an input and the model, called *Language Models*, will generate a text in the same context as the preconditioning text. Can a secret message be hidden inside of another message (called *cover text*)? This paper presents an approach achieving this using Language Models.

II. Related Work

A. Language Models

Language models for text generation have seen an increasing interest in the recent years. The most famous ones are the *GPT2* models[1] created by *OpenAI*. These models are able to generate a synthetic text given a preconditioning. Other models exists such as BERT[2] or RoBERTa[3]. To access different language models, the Transformers library provided by *Huggingface AI* is available. The advantage of this library is the unified interface that is provided. Through this, the adaption of new language models is significantly easier. Due to the long training process, pretrained models can be utilized[4].

The basic method of a transformer based language model can be split into two parts. At first the input sequence is tokenized using an encoder. Using this tokenized input, the next token is predicted using a neural network.

Using the new tokens, the output text predicted based upon the previous tokens and must be decoded. The result of this decoding is the desired output text. The training of the neural network can be conducted using unsupervised and supervised learning approaches. Especially the unsupervised approaches in GPT2 have lead to siginificant improvements regarding the quality of the language model [1] [5].

B. Steganography through Text

The approach of using steganography by applying a LSTM neural network has previously been applied in [6]. Here, the strings were first compressed and embedded. Through the embedding step, the size of the text is minimized. The cover text subsequently is generated using a modified LSTM model. The method already creates overall decent cover texts with respect to being detected as machine generated, however, it is not utilizing the improvements made through the generation of transformer based language models. Furthermore, [7] utilized the modern language Model of GPT2 medium. There, a *Huffman encoding* is applied to encode the secret text. This is an entropy based approach aiming at minimizing the size of the encoded sequence [8].

III. Proposed Method

The method developed by us differs from existing approaches in [7] in respect to the encoding. Whilst [7] use a Huffman encoding of the secret message, the proposed method utilizes the ranks of the next words given the previous text as the encoding. The protocol can be summarized as follows :

- Alice generates the ranks of her secret text using a LM. The ranks refer to how likely it is that the next word in the secret texts follow the previous sequence.1, 2
- Alice creates a cover text using the obtained ranks and the shared preconditioning.
- Bob receives the cover text
- Bob uses the shared LM and preconditioning to recover the ranks of the cover text
- Bob uses these ranks to reproduce the secret text

Using this protocol it shall be ensured that the encrypted message is both secure in regards to being decrypted by an adversarial attacker and at the same time looks like a normal text such that it does not appear suspicious.

A. Generation of the ranks

In order to reproduce the ranks of the secret text a preconditioning of the Language model LM needs to be chosen. Usually, we take a very small preconditioning such as a dot followed with a space or a small sentence in the same *context* as the secret text. After tokenizing the secret text the formula1 is applied at each token t and the preconditioning tokens p_t :

$$rank_t = arg(sort(LM(p_t)), LM(t|p_t)) \quad (1)$$

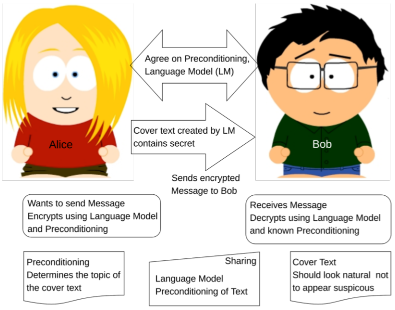

Figure 2. Overview of the encryption system

Then we append the token t to the previous tokens p_t.

B. Creation of the cover text

Given a specific preconditioning for the cover text p_c and the ranks of the secrets $ranks$, we generate the next word by taking the i-th most likely word for each i in the given ranks. The next token t is computed by the formula :

$$t = sort(LM(p_c))_i \tag{2}$$

The token t is appended to the previous tokens p_c. In practice, the generated cover text stops before the generated sentence ends. In III-D an approach to deal with this issue is proposed and implemented.

C. Decryption of the cover text

Here Bob receives the cover text. In order to compute back the ranks, for every token in the cover text, we apply the formula 1 and appending the current token to the previous ones at every step.

Bob uses these ranks to generate the secret text, using the secret precondioning the same way as in equation 2

D. Sentence completion

Using the ranks only, the cover text might end without the sentence being finished. This, however, makes the generated text easy to detect. Thus, the model goes to open end text generation until the final sentence is finished.

To recognize, where the text is finished, a token needs to be reserved that corresponds to the text being finished. This will be embedded into the ranks. As not everything should be offset by one, we decided to only offset ranks greater than two by one. Once the text is finished, the reserved rank three is appended and the model goes to open end text generation until the text ending is reached.

This can be seen from the following example:

Without Completion	1	3	4	20	0			
With Completion	1	4	5	21	1	3	0	0

For the open end text generation, rank one will continuously be used as this leads to the smoothest text.

IV. Model Variants

For the developed process, the user is able to decide between different available models. We propose:

- GPT2 - *small*, *medium*, *large*, *xlarge*
- BERT
- RoBERTa

Although the last 2 models are not used for text generation[1], we have an implementation for them to be able to compare the performance between *GPT2* and *BERT*.

V. Adversarial attacks

One of the main challenges behind *secret exchanging* protocols is to be robust against attacks, especially attacks called *Man In The Middle* (MITM) attacks. Since the model type, the preconditioning of the secret text and the cover text are the shared secrets from Alice and Bob, we assume that it is not possible to retrieve those secrets.

This implies that the only way of attacking the protocol would require discovering the *shared secrets* of Alice and Bob, which means that the security of the communication depends on Alice and Bob (they have to agree on a common secret), not on the protocol itself. This property is known as *Kerckhoffs's principle*, where the adversary knows the system but does not know the secret key.

From a MITM perspective, if the latest can intercept the cover text and be able to predict that the cover text is *fake* (aka generated by a Language Model), then it would be a threat for the communicators. Our goal is to determine which configuration leads to a better protection against this kind of attacks.

VI. Adversarial detection

As it was pointed out in the previous section, we present a steganographic approach that satisfies Kerckhoffs's principle, which implies that without knowing the shared secret (type of the model + preconditioning text), an adversary would be unable to uncover the secret text. Nevertheless, another type of attack is possible: an adversary might intercept a message (sent from Alice to Bob) and classify it as being benign/non-steganographic or steganographic.

This vulnerability can be seen as a threat not to the confidentiality of the conversation (since the content of the messages is not disclosed), but to the privacy of Alice and Bob (the adversary reveals the fact that a steganographic exchange is taking place).

A. Framework

The robustness of our approach to adversarial detection will be tested in the following manner:

The adversarial detector is modeled as a binary classifier that is trained on steganographic samples and benign samples that contain no embedded steganography. These two datasets were prepared in the following way: first, the benign

Corpus Size	Model Type	LSTM	FastText	RoBERTa
100	GPT2	0.7273	0.4286	1.0000
	BERT	1.0000	0.6667	1.0000
	RoBERTa	1.0000	0.7619	1.0000
500	GPT2	0.9020	0.5644	1.0000
	BERT	1.0000	0.5149	1.0000
	RoBERTa	1.0000	0.7327	1.0000
1000	GPT2	0.8709	0.6412	0.9834
	BERT	1.0000	0.6944	1.0000
	RoBERTa	1.0000	0.7907	1.0000

Table I
STEGANOGRAPHY DETECTION USING THREE CLASSIFIERS ON THE CNN-DAILYMAIL CORPUS

samples were taken from the CNN-DailyMail corpus. For every benign sample, its final k (= length of the secret text) words were removed and regenerated using one of the proposed methods. These samples were then labeled (benign samples as 0, steganographic samples as 1) and split into training and testing sets.

The three types of classifiers that were used to model adversarial detection are:

- FastText, a baseline classifier used in various NLP applications
- an LSTM classifier, comprised of an embedding layer, an LSTM layer and a fully connected logistic layer
- a Transformer-based classifier, comprised of a RoBERTa-type model with a fully connected logistic layer

B. A note on preprocessing

Before presenting the computational results, it is important to note why our adversarial detector modeling does not employ any kind of preprocessing. The reason for that is two-fold:

- Our initial experiments on the Brown corpus have shown that removing punctuation, removing leading and trailing white-space and converting all letters to lowercase leads to a negligible fluctuation in adversarial robustness.
- This kind of preprocessing is unrealistic for more advanced language models like GPT2, which not only take punctuation and white-space into account but are actually quite dependent on these small textual details.

C. Experimental results

This subsection presents the experimental results for adversarial steganography detection. The rows of Table I identify the corpus size and the type of model used by Alice to generate the steganographic text (for more details on the models employed, see Section IV). The columns correspond to the classifier models used by the adversary to detect the presence of steganography in a given text.

D. Conclusions: link to GANs

The experimental results presented in Subsection VI-C confirm our initial assumptions about the models that we employ for steganographic text generation: unlike BERT and RoBERTa, GPT2 is specifically tailored to text generation, which allows it to generate more realistic texts which are therefore harder to detect. Moreover, these results correlate with the complexity of the three classifier models.

Apart from these architectural differences, we can also observe the change in accuracy as a function of the corpus size, where increasing the number of samples in the dataset leads to significant improvements in accuracy.

This particular property implies that Alice and Bob have an *information advantage* over the adversary that deteriorates over time: having observed few steganographic messages, the adversary will find it hard to train a detection model. However, as the number of his observations increases, he will be able to detect steganographic messages more reliably.

Nevertheless, in order to minimize the success rate of the adversary, Alice and Bob may find it useful to conduct the detection experiment themselves. By testing their models locally, they will be able to select the model that fools most classifiers (or at least produces the smallest accuracies) and then use this model for secure communication. This can be seen as a *GAN-like* approach to model selection, where Alice and Bob train the classifier (discriminator) and choose the model (generator) which approximates the true distribution (natural language) the best.

VII. EVALUATION

In this section we present how we evaluate our protocol. First, we present the calculation of the smoothness of the generated cover texts with different models. We then focus on computing the perplexity score from the generated corpus from **1000** articles from the *DailyMail corpus* generated with **1000** different preconditioning randomly taken from the *Brown corpus*. Our generated testing corpus is publicly available **here**.

A. Smoothness of the generated ranks

A way to understand the smoothness of the generated ranks will be to plot the ranks of the generated tokens [3]. If the model that we use generates a text with low ranks, we then have a high chance to generate a very consistent cover text. On the other hand if our model generates a cover text with unstable high ranks, the text will probably do not make sense. Below is a toy example, we generate of a secret text which is few lines from an article using *GPT2-medium* and *BERT*.

Therefore, there is an evidence that GPT2 generates smoother ranks, which leads to more consistent cover texts than BERT and RoBERTa.

B. Consistency of the generated texts

In addition to all the features presented above, another way to evaluate our model it to compare the distribution of

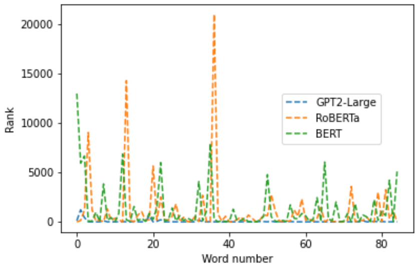

Figure 3. Generated ranks with the different models

the words from the generated cover texts. The cover text would be considered as consistent if the distribution of the generated words is close to the distribution of the words in the original texts. Using 1000 generated texts by the four models, the occurance of the 20 most common words from the secret texts have been plotted.

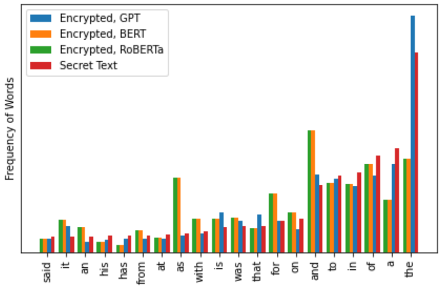

Figure 4. Word Distribution

It can be seen that the distribution of GPT2 large is very similar to the distribution from the secret texts. However, for BERT and RoBERTa the deviations are significantly larger.

C. Perplexity score

One way to evaluate our models is to compute its perplexity score. The perplexity score is a metric to measure the correctness of the output with the following formula :

$$p = \exp \frac{1}{N} \sum_{i=0}^{N} p(w_i|w_{j<i})$$

Model variant	$RoBERTa$	$BERT$	$GPT2_{large}$
Cover text	244.09	5.96	**4.577**
Raw text	11.94	2.32	4.506

For each metric, we evaluate the cover text generated by the corresponding model using the latest model. As expected, cover texts generated with GPT2 lead to lower perplexity scores. This corresponds to a more consistent and more logical cover text. As the difference between cover texts and humanly generated texts is lowest for GPT2, this is most natural.

VIII. Conclusion and Future work

We have managed to implement a protocol for a new approach of steganography in text using Language Models and word ranks. The protocol deployed with *GPT2* seems to be the most efficient since it is robust against the attacks [VI-C], and the distribution of the generated words seems to be really close to the distribution of the words from the raw text [VII-B]. In addition to that, we showed that cover texts generated with *GPT2* have the lowest perplexity score amongst the other models by also having a score which is close to the raw testing set's score [VII-C]. Below we present the discussions regarding the possible issues and the future work.

A. Recovering issues

When using the proposed model, problems in retrieving the correct message can occur. This is due to the way the decoding works. For instance the tokens [326, 220, 39608] and [326, 308, 2959] of Open AI's GPT2-medium both correspond to the text " that grey". However, as only the second token series is retrieved by the encoding of the words, this can lead to problems. To mitigate this risk, one can compare the tokens, which are produced by the model, to tokens one would get by encoding the output text. If they differ, the aforementioned problem will occur which means, a reliable encryption is still possible. As the problem is due to the tokenization, it was out of scope for this project.

B. Deployment issues

This protocol is computationally highly expensive. In order to deploy the protocol we may need an extremely powerful and expensive server. If we have the equipment, it would be realistic to have a server that runs the protocol.

However, some recent works have been done for fast inference applied to language models [9], in the future we may be able to run this protocol with a reasonable computational time.

C. Future work

We saw that the texts generated by *BERT* and *RoBERTa* do not really make sense, and could be easily detected as fake text by classifiers or even humans. This is due to the fact that these models have not been designed for text generation due to their bi-directional architecture. Thus, they do not generate text as well as GPT2. There are some possibilities to make BERT generate consistent texts by fine-tuning it to a specific context and with some post processing techniques [10][11]. For future work, we may be interested to make BERT generate more consistent texts by applying some techniques presented before.

If possible, we would also experiment the protocol and evaluate it using GPT3 (which is available only for commercial use) and other state-of-the-art models that will appear in the future.

REFERENCES

[1] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, “Language models are unsupervised multitask learners,” *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” *arXiv preprint arXiv:1810.04805*, 2018.

[3] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, “Roberta: A robustly optimized bert pretraining approach,” *arXiv preprint arXiv:1907.11692*, 2019.

[4] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz *et al.*, “Huggingface’s transformers: State-of-the-art natural language processing,” *ArXiv*, pp. arXiv–1910, 2019.

[5] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[6] T. Fang, M. Jaggi, and K. Argyraki, “Generating steganographic text with lstms,” *arXiv preprint arXiv:1705.10742*, 2017.

[7] Z. M. Ziegler, Y. Deng, and A. M. Rush, “Neural linguistic steganography,” *arXiv preprint arXiv:1909.01496*, 2019.

[8] D. A. Huffman, “A method for the construction of minimum-redundancy codes,” *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[9] Y. J. Kim and H. H. Awadalla, “Fastformers: Highly efficient transformer models for natural language understanding,” *arXiv preprint arXiv:2010.13382*, 2020.

[10] P. Mishra, “Natural language generation using bert,” 2020. [Online]. Available: https://medium.com/intel-student-ambassadors/natural-language-generation-using-bert-df6d863c3f52

[11] ——, “Natural language generation using bert.” [Online]. Available: https://github.com/prakhar21/Writing-with-BERT