Explaining the Paper: Stealing Part of a Production Language Model

Ricard Santiago Raigada García

September 12, 2024

Contents

1 Introduction

Currently, large language models have become significant in people's lives. In fact, some are increasingly dependent on the functionalities provided by these models. On the contrary, the areas of research in Adversarial Machine Learning (ML) do not seem to follow the same growing dynamic. This puts models such as GPT, PaLM-2, etc. in jeopardy. This type of attack allows an adversary to recover confidential information from a model, such as internal parameters or structure, in a relatively simple way. Using the application interface to make calls to the public API.

Recently, Google researchers have published a scientific article in which they describe a language model extraction attack. The main idea is to recover the projection layer of embeddings of transformer models by making queries to the public API. Among the mathematical tools used, the Singular Value Decomposition (SVD) and range analysis. All of this is explored in bias extraction attacks such as logit-bias and logit-vector.

The motivation of this article is to extend the wonderful work done to an audience with some knowledge of ML, but who do not necessarily need to be experts in the field to be able to understand the research.

2 Logits and Logprobs: Fundamental Concepts

Previously, I have mentioned a couple of types of attacks described in the research. I will start by briefly reviewing the concept of logits. In neural networks, particularly in language models, when we mention logits we are actually referring to the output values of the last linear layer of the network just before applying the desired activation function.

Particularly, in the research, we discuss the `softmax` activation function. If z is a vector of logits of dimension l, then the logits are the direct inputs to the softmax function.

$$z = \mathbf{W} \cdot \mathbf{x} \tag{1}$$

where l refers to the vocabulary size or the number of classes. $\mathbf{W}$ is the weight matrix of the final projection layer with dimensionality $l \times h$. h is the vector of hidden states, and hence of dimension h. The logits z_i for each token i in the vocabulary represent the linear activations used to compute the probabilities associated with each token.

3 Softmax function

Returning to the softmax function referred to above, let's start by recalling that it is a type of activation function that converts logits into probabilities. To accomplish this, it normalizes the vector of logits so that the sum equals one. The probability that the token y_i is of class k, given the logit vector $\mathbf{z}$, mathematically speaking, is:

$$\mathbf{P}(y_i = k|\mathbf{z}) = \frac{\exp(z_k)}{\sum_{j=1}^{l} \exp(z_j)} \tag{2}$$

The logarithmic probabilities or `logprobs` are the natural logarithms of the probabilities after applying the softmax function, so:

$$\mathbf{P}(y_i = k|\mathbf{z}) = z_k - \log(\sum_{l}^{j=1} \exp(z_j)) \tag{3}$$

4 Integrating Logits and Logprobs in the Context of the Attack

With this brief introductory review, it is now easier to understand how the attacks are performed. First, let's discuss the extraction attack for logit vector APIs. The APIs of the language model interfaces that provide direct access to the logits, the attacker can obtain the vector z in any query he makes.

The security breach occurs in the direct access to the vector z, since this makes it possible to perform the calculation of the probabilities before being transformed by the softmax function. That is, the attacker uses the original values to perform, for example, an SVD to extract the hidden dimension h and reconstruct the matrix $\mathbf{W}$.

Secondly, when the API does not provide direct access to the logits, but to the logprobs; the attacker must manipulate the bias of logits b to infer the underlying logits. As discussed above, the logprobs are derived directly from the logits through the softmax function. In this task, the attacker changes the bias of a particular token to see the variability of the logprobs, and deduce the differences between the logits of different tokens, giving him the possibility to reconstruct the complete vector of logits.

The third case is where the attacker has no access to either logits or logprobs. To do so, he must use other techniques such as binary search to perform a manipulation of the model until a specific token becomes the most probable. That is, it indirectly reveals the structure of the underlying logits.

5 Transformative Language Models

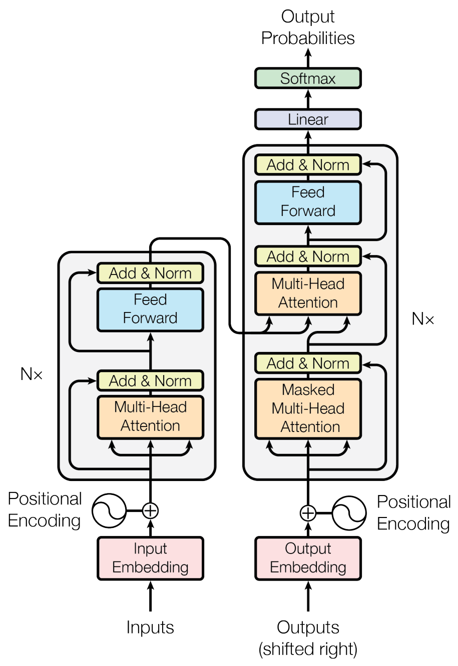

Figure 1: The Transformer - model architecture. Vaswani et al., 2017

Transformer models process sequences of tokens, predicting the next token in the sequence. This architecture follows the attention mechanism Vaswani et al., 2017 to compute dependency between tokens without resorting to a sequential type structure like Recurrent Neural Networks (RNN). The attention mechanism along with the transformer architecture has made it very effective in Natural Language Processing (NLP) tasks.

The transformer architecture has two important components, which can be seen in figure 1. The encoder and the decoder. The two modules have layers that apply attention and a fully connected (feed-forward) network.

5.1 Encoder

In the encoder, there is a stack of N identical layers. Each layer has two sublayer consisting of Multi-Head Attention and Feed-Forward Network.

The Multi-Head Attention has multiple attention heads in parallel that learn different representations of the input. That is, each of the heads performs a different projection of the inputs. The attention function is expressed as:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V} \tag{4}$$

$\mathbf{Q}$ represents the queries, $\mathbf{K}$ the keys and $\mathbf{V}$ the values. $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ are matrices derived from the inputs where d_k is the dimensionality of the keys.

After the attention mechanism, the output passes through a two-layer neural network with an activation function **ReLU**. This process makes the transformation non-linear, improving the model's ability to capture complex relationships. Each sublayer in the encoder has a residual connection, allowing the original signals to flow through the network without being completely altered by the transformation.

5.2 Decoder

The decoder also has a stack of N layers. But the decoder has an additional layer of multi-head attention on top of the encoder output. That is, it is the mechanism for the decoder to use information from the encoder to generate the output sequence. The decoder has masked attention, which prevents future positions from being seen by the network when it generates an output sequentially. This is used to preserve the autoregressiveness of the model, where the prediction of a token only depends on the sequence of previous tokens.

5.3 Logits and Softmax Computation

At the final part of the decoder, a linear layer is applied that projects the outputs into a logit space of dimension **l** (**l** is the vocabulary size). It is represented as:

$$\mathbf{z} = \mathbf{W} \cdot \mathbf{h} \tag{5}$$

where $\mathbf{W}$ is the projection weight matrix and $\mathbf{h}$ is the hidden state vector. The softmax function is applied to $\mathbf{z}$ to convert these logits into a probability distribution over the possible tokens. The formula for computing the probability of a given token y z is:

$$P(y = k \mid \mathbf{z}) = \frac{\exp(z_k)}{\sum_{j=1}^{l} \exp(z_j)} \tag{6}$$

where z_k is the logit value for the token k, and the sum in the denominator is over all logits z_j for the possible tokens.

Since the softmax function converts the vector of logits $\mathbf{z}$ into a probability distribution, where the sum in the denominator is over all possible logits. It is obvious that the total probability of all possible tokens is equal to 1.

$$\begin{aligned}
\sum_{k=1}^{l} P(y=k \mid \mathbf{z}) &= \sum_{k=1}^{l} \frac{\exp(z_k)}{\sum_{j=1}^{l} \exp(z_j)} \\
&= \frac{1}{\sum_{j=1}^{l} \exp(z_j)} \sum_{k=1}^{l} \exp(z_k) \\
&= \frac{\sum_{k=1}^{l} \exp(z_k)}{\sum_{j=1}^{l} \exp(z_j)} \\
&= 1
\end{aligned}$$

6 Singular Value Decomposition (SVD)

This is a fundamental technique of linear algebra that decomposes any matrix $\mathbf{A} \in \mathcal{M}_{m\times n}(\mathbb{R})$ in three matrices: $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}^T$. The SVD is essential to analyze the range of the logit matrix $\mathbf{Q}$ used in the attack.

$$U = \left[\begin{array}{cccc|ccc}
u_{11} & u_{21} & \cdots & u_{n1} & u_{n+1,1} & \cdots & u_{m1} \\
u_{12} & u_{22} & \cdots & u_{n2} & u_{n+1,2} & \cdots & u_{m2} \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
u_{1m} & u_{2m} & \cdots & u_{nm} & u_{n+1,m} & \cdots & u_{mm}
\end{array}\right] \in \mathbb{M}_{m\times m}(\mathbb{R})$$

$$V = \begin{bmatrix}
v_{11} & v_{21} & \cdots & v_{n1} \\
v_{12} & v_{22} & \cdots & v_{n2} \\
\vdots & \vdots & \ddots & \vdots \\
v_{1n} & v_{2n} & \cdots & v_{nn}
\end{bmatrix} \in \mathbb{M}_{n\times n}(\mathbb{R})$$

$$\Sigma = \left[\begin{array}{cccc}
\sigma_1 & 0 & \cdots & 0 \\
0 & \sigma_2 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & \sigma_n \\
\hline
0 & 0 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 0
\end{array}\right] \in \mathbb{M}_{m\times n}(\mathbb{R})$$

$\mathbf{U}$ is a square orthogonal matrix of order m, $\mathbf{V}$ is a square orthogonal matrix of order n, and $\mathbf{\Sigma}$ is a diagonal matrix. The i-values correspond to the singular values and are listed in descending order. There are several key properties of SVD, however the most interesting ones for this case are that the σ_i values are unique, the number of non-zero singular values is the rank of $\mathbf{A}$, and the largest singular value of $\mathbf{A}$ is equal to the spectral norm of $\mathbf{A}$.

SVD is used to decompose the matrix $\mathbf{Q}$ containing the remaining logits from multiple queries. This allows the hidden dimension $\mathbf{h}$ to be discovered and provides a basis for approximating the projection matrix $\mathbf{W}$.

6.1 Rank and Dimensionality

The amount of information that a language model can represent can be known from the rank of a matrix. The rank is the maximum number of linearly independent columns or rows. For the case of the projection matrix $\mathbf{W}$, the rank determines how many directions in the logit space can be occupied by the hidden states from the analysis of the column space of $\mathbf{Q}$ allows the hidden dimension $\mathbf{h}$ to be inferred.

7 Extraction Attacks and Query Interfaces

The extraction attack, as discussed, is based on interacting with the public API that exposes part of the model's functionality. The outputs displayed on the screen (e.g., ChatGPT website) or console (terminal output from the model API) are based on logits or logarithmic probabilities, in general. The research performs different types of attacks depending on the access to the model.

7.1 Query Interfaces and Logit-Vector APIs

Attacks on APIs give direct access to the logit vector. Extraction is based on the recovery of the projection matrix $\mathbf{W}$ from the model responses. A query generates a logit vector that represents the linear activations before applying the softmax function. The objective of this attack is to reconstruct the matrix $\mathbf{W}$ from decomposition techniques such as SVD.

The logit vector $\mathbf{z}$ is defined as in the equation 5, where $\mathbf{W}$ is the weight matrix of dimension $l \times h$, and $\mathbf{h}$ is the hidden state vector of dimension h. To extract the matrix $\mathbf{W}$, the attacker performs multiple queries on the model, varying the token prefixes to modify $\mathbf{h}$. This results in a logit matrix $\mathbf{Q}$ where each column corresponds to a logit vector $\mathbf{z}_i$ resulting from a separate query:

$$\mathbf{Q} = \mathbf{W} \cdot \mathbf{H} \tag{7}$$

where $\mathbf{H}$ is a hidden state matrix collected from multiple queries. By decomposing $\mathbf{Q}$ using SVD, the attacker can identify the hidden dimension h and approximate $\mathbf{W}$ to a linear transformation.

The attacker applies SVD to $\mathbf{Q}$:

$$\mathbf{Q} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T, \tag{8}$$

where $\mathbf{U}$ contains the left-singular vectors approximating $\mathbf{W}$, $\boldsymbol{\Sigma}$ is a diagonal matrix containing the singular values (σ_i), and $\mathbf{V}^T$ is the matrix of right-singular vectors. The rank of $\mathbf{W}$ is inferred from the number of significant singular values in $\boldsymbol{\Sigma}$, providing an accurate estimate of h.

This procedure allows reconstructing $\mathbf{W}$ in the form of

$$\mathbf{W}' = \mathbf{U} \cdot \mathbf{\Sigma} \cdot \mathbf{G}, \tag{9}$$

where $\mathbf{G}$ is an unknown linear transformation. Although $\mathbf{G}$ is not directly recoverable, the structure of $\mathbf{W}'$ is sufficient to largely replicate the behavior of the original model.

7.2 Intuition of the Attack

When an adversary queries the model with different token prefixes, the resulting output vector (vector of logits), although of dimension l (the size of the vocabulary), actually resides in a subspace of dimension h because the embedding projection layer performs a projection from h dimensions. Thus, if enough queries are made to the model, beyond h queries, the new logit vectors will become linearly dependent on the previous ones. This observation can be exploited to compute the dimensionality of the subspace, as I have recommended above.

7.3 Formalization of the Attack

More formally, the attack relies on the following simple mathematical result, formalized as Lemma 4.1 Carlini et al., 2024:

Lema 7.1 (Lema 4.1). *Let $\mathbf{Q}(p_1, \ldots, p_n) \in \mathbb{R}^{l \times n}$ be the matrix whose columns are the answers of the logits API $\mathbf{O}(p_1), \ldots, \mathbf{O}(p_n)$ for different queries. Then, $h \geq rank(\mathbf{Q}(p_1, \ldots, p_n))$. Furthermore, if the matrix with columns $\mathbf{g}_\theta(p_i)$ (para $i = 1, \ldots, n$) has rank h y $\mathbf{W}$ has rank h, so $h = rank(\mathbf{Q}(p_1, \ldots, p_n))$.*

Proof. Let us consider that $\mathbf{Q} = \mathbf{W} \cdot \mathbf{H}$, where $\mathbf{H}$ is a matrix $h \times n$ whose columns are $\mathbf{g}_\theta(p_i)$ for $i = 1, \ldots, n$. Since $\mathbf{Q}$ is the product of $\mathbf{W}$ and $\mathbf{H}$, its range is bounded by the smallest of the ranges of $\mathbf{W}$ and $\mathbf{H}$. Under the assumption that both have rank h, it follows that rank$(\mathbf{Q}) = h$.

It is shown that the dimension h can be estimated by analyzing the range of $\mathbf{Q}$, which can be computed from its singular values.

In practice, the calculations have an accuracy limited by 8- or 16-bit floating point. So it cannot be directly assumed that the rank is the number of linearly independent rows or columns. Therefore, the numerical range of $\mathbf{Q}$ is considered, determined by the order of its singular values ($\sigma_1 \geq \cdots \geq \sigma_n$) and looking for the largest multiplicative jump $\sigma_i / \ sigma_{i+1}$ between consecutive values. Which indicates the point where the singular values represent the effective dimension.

7.3.1 Algorithm 1: Hidden Dimension Extraction Attack

Algorithm 1 Hidden Dimension Extraction Attack

```
1: Initialize $n$ to a value greater than $h$ (an initial guess).
2: Initialize an empty array $\mathbf{Q} = \mathbf{0}_{n \times l}$.
3: for $i = 1$ hasta $n$ do
4:     Generate a random prefix $p_i$ and compute $\mathbf{Q}_i$ as the model response to $p_i$.
5: end for
6: Compute the singular values $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$ of $\mathbf{Q}$.
7: Identify the largest jump in the singular values and use it to estimate $h$.
```

7.3.2 Experimental Results

To visualize this attack, the researchers conducted an experiment Carlini et al., 2024 using the Pythia1.4B model, where an increasing number of queries n were sent to the model and the singular values of $\mathbf{Q}$ were computed. As expected, when $n < h$, the matrix $\mathbf{Q}$ has full rank and the singular values are non-trivial. However, once $n > h$, the significant singular values stabilize at exactly h, allowing h to be identified as the model's hidden dimension.

7.4 Complete Extraction of the Final Layer Up to Symmetries

After determining the hidden dimension h, the next goal is to extend the attack to recover the projection matrix $\mathbf{W}$ that projects from the h-dimensional hidden space to the l-dimensional logits space.

7.4.1 Decomposition of the Matrix Q

Since $\mathbf{Q}$ has already been defined as the logits matrix obtained through queries to the model, it is rewritten using SVD (equation 8).

Lema 7.2 (Lemma 4.2). *Under the logits API threat model, and under the same assumptions as Lemma 4.1:*

The described method recovers a matrix $\tilde{\mathbf{W}} = \mathbf{W} \cdot \mathbf{G}$ for some matrix $\mathbf{G} \in \mathbb{R}^{h \times h}$. With the additional assumption that $\mathbf{g}_\theta(p)$ is a transformer with residual connections, it is impossible to extract $\mathbf{W}$ exactly.

Proof. Consider that $\mathbf{Q} = \mathbf{W} \cdot \mathbf{H}$, where $\mathbf{H}$ is the hidden state matrix. Since SVD decomposes $\mathbf{Q}$ into $\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T$, it can be rewritten as:

$$\mathbf{W} \cdot \mathbf{H} = \mathbf{U} \cdot \boldsymbol{\Sigma} \cdot \mathbf{V}^T. \tag{10}$$

This implies that $\mathbf{W} \cdot \mathbf{H}$ and $\mathbf{U} \cdot \boldsymbol{\Sigma} \cdot \mathbf{V}^T$ are equivalent up to a linear transformation $\mathbf{G}$, where $\mathbf{U} \cdot \boldsymbol{\Sigma} = \mathbf{W} \cdot \mathbf{G}$. Therefore, $\mathbf{W}$ can be recovered, though with ambiguity due to the matrix $\mathbf{G}$, which is inherent to the symmetries in the linear projection.

This method was tested on a range of models, including GPT-2, Pythia, and LLaMA Carlini et al., 2024, to evaluate the effectiveness of the attack in reconstructing $\mathbf{W}$. The results showed that the extracted matrix $\tilde{\mathbf{W}} = \mathbf{U} \cdot \mathbf{\Sigma}$ has a significantly low root-mean-square (RMS) error compared to the original matrix $\mathbf{W}$, indicating high accuracy in recovery. The only notable exception was GPT-2 Small, where the RMS was slightly higher due to the observed lower effective dimensionality.

7.5 Logit-Bias Manipulation and Exploitation

An extension of the attack involves manipulating logit-bias, a feature that some APIs expose to adjust the probability of certain tokens before applying softmax. This manipulation allows the attacker to modify the ranking of token probabilities and, therefore, infer information about the underlying logits.

Suppose we want to determine the logit value associated with a specific token t_k. In APIs where it is possible to adjust a real bias b_k that is added to the logits before applying the softmax function, the modified logit becomes:

$$z'_k = z_k + b_k. \tag{11}$$

By making iterative adjustments to the bias b_k and observing how the token probability changes, the attacker can be approximated z_k with high precision. This process is repeated for various tokens, allowing the reconstruction of critical parts of the weight matrix $\mathbf{W}$.

When a bias B is applied to a specific token i, the observed value is:

$$y_i^B = z_i + B - \log\left(\sum_{j=1}^{l} \exp(z_j + b_j)\right), \tag{12}$$

where z_i are the original logits and B is the bias applied to z_i. By manipulating B and analyzing changes in y_i^B, the attacker can infer the differences between the original logits.

The adaptation of the attack involves making multiple queries to the model with different biases and analyzing how the logprobs change. This technique allows the attacker to infer the original logits from the model's modified responses, providing a way to recover critical information about the weight matrix $\mathbf{W}$.

7.6 Logprob-Free Attacks: Limitations and Strategies

When direct access to logits and manipulation of logit-bias are not possible, one must assume a Logprob-Free attack. This type of attack relies solely on observing the logprobs of the response. A creative attacker can still extract information from the model.

One approach is to perform a binary search on the logits through indirect manipulation of the inputs. This can be done by slightly altering the context of the target token through successive queries, inducing a change in the logprob that reveals the approximate range of the underlying logit. This method could be used for targeted attacks with less precision required in the reconstruction of $\mathbf{W}$.

8 Security and Defense Strategies

The research proposes certain security measures that, for example, OpenAI has implemented. Among them, it suggests eliminating or limiting the ability to manipulate logit bias, adding noise to logits, and monitoring query patterns to detect extraction attempts.

9 Conclusions

The discussed extraction attacks demonstrate that, with sufficient queries and mathematical analysis, the security of language models can be compromised, even when APIs are restrictive. This highlights the importance of the field of adversarial machine learning.

Bibliography

Carlini, N., Paleka, D., Dj, K. (, Dvijotham,) Steinke, T., Hayase, J., Feder Cooper, A., Lee, K., Jagielski, M., Nasr, M., Conmy, A., Yona, I., Wallace, E., Rolnick, D., & Tramèr, F. (2024). Stealing Part of a Production Language Model. https://arxiv.org/abs/2403.06634v2

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention Is All You Need. *Advances in Neural Information Processing Systems*, *2017-December*, 5999–6009. https://arxiv.org/abs/1706.03762v7