Indistinguishability Adversary under Ciphertext Only Attack

Sandeep Chatterjee

[1]Indian Statistical Institute Kolkata

sandeepchatterjee66@gmail.com

Abstract. *In this work, we introduce an approach for classifying binary sequences by using transformer-based architectures with specialized preprocessing. Given a binary input sequence, we split it into halves and perform an XOR operation to differentiating bit patterns. We employ both BERT and GPT-2 models to predict classes from the resulting bitwise-processed sequences, adapting them through efficient parameter fine-tuning. This preprocessing and fine-tuning together reduces computational overhead while maintaining accuracy. Experimental results demonstrate the effectiveness of this approach on a binary classification task, with little advantage. Our findings open avenues for applying transformers in low-dimensional, binary-based classification scenarios, offering efficient, adaptable solutions for real-world applications. In future work, we can analyze if the advantage is increasing function in terms of training data and not advantage is not a neglible function in terms of security parameter (bit size).*

1. Introduction

In cryptology, ensuring that ciphertext remains indistinguishable from random data is a cornerstone of cryptographic security, often quantified through the notion of computationally secure ciphers. A cipher is considered *computationally secure* if it is infeasible for an adversary to break the encryption within polynomial time concerning a security parameter λ. This security paradigm implies that given a ciphertext C, an adversary cannot distinguish C from a random string R with probability significantly greater than $\frac{1}{2}$.

Formally, let $\mathcal{A}$ denote an adversary and $E_k(M)$ represent the encryption of a message M with key k. A distinguishing attack aims to determine whether a given string is an encryption of M or a random string R. If the probability of $\mathcal{A}$ correctly identifying $E_k(M)$ as the encryption, rather than random data, exceeds $\frac{1}{2}$ by a non-negligible factor, this indicates a cryptographic weakness. This probability can be expressed as:

$$\Pr\left[\mathcal{A}(E_k(M)) = 1\right] - \Pr\left[\mathcal{A}(R) = 1\right] > \epsilon(\lambda)$$

where $\epsilon(\lambda)$ is a negligible function of the security parameter λ. In contrast, for a computationally secure cipher, $\epsilon(\lambda)$ approaches zero as λ increases, ensuring that the distinguishing probability remains close to $\frac{1}{2}$.

Building upon this framework, our work introduces an approach for distinguishing and classifying binary sequences based on transformer architectures. Given an input binary sequence B, we split it into two equal halves B_1 and B_2, then apply a bitwise XOR operation:

$$X = B_1 \oplus B_2$$

resulting in a transformed sequence X, which emphasizes distinguishing bit patterns within B. This XOR-preprocessed sequence X serves as input to transformer-based models, specifically BERT and GPT-2, for binary classification tasks.

To make use of the model's capabilities efficiently in commonly available GPUs, we employ fine-tune the parameters in a computationally efficient manner. The transformer models then predict the class label based on the XOR-preprocessed binary sequence. By representing the original binary input in this differentiated bit-pattern space, the models can capture nuanced relationships, reducing the need for extensive model parameters while maintaining accuracy.

In experiments, we evaluate our method's efficacy on binary classification tasks, observing that transformer-based models can generalize effectively to the task with minimal accuracy trade-offs. This efficient approach not only demonstrates potential in resource-constrained environments but also opens possibilities for applying transformer-based methods to other low-dimensional, binary-encoded data classifications.

Our findings raise questions for further research: as we increase the size of the training dataset, does this advantage persist, and could this preprocessing technique be extended to enhance security in distinguishing attacks? Future work could examine whether the observed advantage scales as a function of the training data size, potentially offsetting $\epsilon(\lambda)$ and achieving even greater resilience in the classification of encrypted or compressed binary sequences.

2. Our Work

3. Methodology

In this section, we present our approach for classifying binary sequences using transformer-based architectures, particularly BERT and GPT-2, with specialized preprocessing. Our approach involves a novel preprocessing step where binary sequences are split into two halves and an XOR operation is applied. The transformed sequence is then fed into a transformer model for binary classification. We also utilize Low-Rank Adaptation (LoRA) to efficiently fine-tune the parameters of the transformer models, thereby reducing computational overhead.

3.1. Preprocessing Binary Sequences

Let $B = b_1 b_2 \dots b_n$ denote a binary sequence, where each $b_i \in \{0, 1\}$ represents a bit of the sequence, and n is the length of the binary sequence. In our work, we focus on sequences with length $n = 1024$, which are split into two equal parts, denoted B_1 and B_2, each of length 512:

$$B_1 = b_1 b_2 \dots b_{512}, \quad B_2 = b_{513} b_{514} \dots b_{1024}.$$

The goal of this preprocessing step is to introduce a distinct separation between the two halves of the sequence in order to emphasize structural differences. We achieve this by applying the XOR operation on the two halves, producing a new sequence:

$$X = B_1 \oplus B_2,$$

where $\oplus$ denotes the bitwise XOR operation. This operation results in a new sequence X of length 512, representing the bitwise difference between the two halves. The XOR transformation serves to reduce redundancy and may help in highlighting distinguishing patterns in the sequence.

The transformed sequence X is then padded to ensure uniform input length, as required by the transformer architecture. This preprocessing step is central to the model's ability to classify binary sequences effectively, as it captures distinct patterns in the sequence structure that would be otherwise difficult to identify directly.

3.2. Transformer Model Architecture

We employ two different transformer-based models, namely BERT and GPT-2 , to perform binary classification on the preprocessed sequences. In both cases, the model architecture consists of a series of attention layers followed by fully connected layers for classification.

3.2.1. BERT Model

The BERT architecture, introduced by Devlin et al., utilizes a bidirectional attention mechanism, which enables the model to learn contextualized representations of the input sequence. The key idea behind BERT is the use of masked language modeling (MLM) during pretraining, which learns to predict missing words (or tokens) based on their surrounding context. For sequence classification tasks, we adapt BERT by adding a classification head on top of the encoder layers.

Let $X = (x_1, x_2, \ldots, x_{512})$ represent the tokenized and padded sequence. BERT applies multiple layers of self-attention to model dependencies between all pairs of tokens in the sequence. Specifically, the self-attention mechanism computes a set of attention weights α_{ij} for each pair of tokens (x_i, x_j):

$$\alpha_{ij} = \frac{\exp\left(\frac{Q_i K_j^T}{\sqrt{d}}\right)}{\sum_{k=1}^{n} \exp\left(\frac{Q_i K_k^T}{\sqrt{d}}\right)},$$

where Q_i and K_j are the query and key vectors corresponding to tokens x_i and x_j, respectively, and d is the dimensionality of the token embeddings.

The query and key vectors are computed from the input sequence using the following linear transformations:

$$Q_i = W_q x_i + b_q, \quad K_j = W_k x_j + b_k,$$

where W_q and W_k are the weight matrices, and b_q and b_k are the bias terms.

After passing through the transformer layers, the final hidden state corresponding to the '[CLS]' token, denoted h_{CLS}, is used as the input to a fully connected layer for binary classification:

$$y = \sigma(W h_{\text{CLS}} + b),$$

where W is the weight matrix, b is the bias term, and σ is the sigmoid function, ensuring that the output is in the range $[0, 1]$.

3.2.2. GPT-2 Model

GPT-2, in contrast to BERT, is an autoregressive language model that generates sequences one token at a time by conditioning on previous tokens. For sequence classification, we adapt GPT-2 by adding a classification head after the transformer blocks.

Given an input sequence $X = (x_1, x_2, \ldots, x_{512})$, the model uses causal self-attention to predict the next token based on the preceding tokens. The attention mechanism in GPT-2 is similar to BERT's, except it only attends to previous tokens in the sequence. The autoregressive property of GPT-2 ensures that the model generates one token at a time, conditioned on previous tokens, by computing attention weights:

$$\alpha_{ij} = \frac{\exp\left(\frac{Q_i K_j^T}{\sqrt{d}}\right)}{\sum_{k=1}^{i} \exp\left(\frac{Q_i K_k^T}{\sqrt{d}}\right)},$$

where Q_i and K_j are the query and key vectors corresponding to the tokens x_i and x_j, with $i \geq j$.

The output of the final transformer block is then passed through a linear layer for binary classification:

$$y = \sigma(W h_{\text{last}} + b),$$

where h_{last} is the final hidden state of the last token in the sequence, and W and b are learned parameters.

3.3. Low-Rank Adaptation

To efficiently fine-tune the transformer models while minimizing computational overhead, we utilize Low-Rank Adaptation. We use the low-rank matrices into the attention layers of the transformer model, allowing for efficient parameter updates while keeping the number of trainable parameters small.

Given the attention mechanism's query and key matrices Q and K in the standard transformer, LoRA introduces low-rank matrices A and B such that:

$$Q' = Q + \alpha \cdot A, \quad K' = K + \alpha \cdot B,$$

where α is a scaling factor, and A and B have low rank. This modification allows for efficient updates during fine-tuning, as the rank of A and B is much smaller than the original Q and K matrices, significantly reducing the number of parameters that need to be trained.

By applying LoRA, we can efficiently fine-tune both BERT and GPT-2 models with minimal computational overhead, thus making the approach suitable for scenarios where computational resources are constrained.

3.4. Binary Classification Task

Finally, we apply our transformer-based models to a binary classification task. After preprocessing the binary sequences using the XOR operation, we tokenize the resulting sequences and feed them into the BERT or GPT-2 model. The models then predict a

binary class label, indicating whether the input sequence belongs to one class or another. We use the binary cross-entropy loss function to train the models, defined as:

$$\mathcal{L}(y, \hat{y}) = -\left(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})\right),$$

where y is the true label and $\hat{y}$ is the predicted probability.

4. GPT-2 Model

In this section, we describe the implementation of the GPT-2 model for sequence classification and discuss the training results, including performance metrics, model architecture, and possible improvements.

4.1. Model Architecture

The model employed is based on `GPT2ForSequenceClassification`, a variant of the GPT-2 transformer model fine-tuned for classification tasks. The model architecture is structured as follows:

- **Base Model**: The `GPT2ForSequenceClassification` model is built on the GPT-2 transformer architecture, which includes:
  - **Embedding Layers**: The token embeddings (`wte`) and positional embeddings (`wpe`) are used to map input tokens to dense vectors and capture position information for each token in the sequence.
  - **Transformer Blocks**: The model contains 12 stacked transformer blocks, each comprising:
    - `LayerNorm`: Layer normalization applied to the input of each block.
    - `Attention Mechanism`: Multi-head attention layer with attention weights, adapted through the LoRA technique.
    - `Feedforward Network (MLP)`: A two-layer feedforward neural network with a GELU activation function.
  - **Classification Head**: A linear output layer (`score`) that maps the hidden states of the transformer to logits for classification.
- **LoRA Modifications**: Low-Rank Adaptation (LoRA) is applied to the attention layers (`attn`) and feedforward networks to introduce low-rank matrices (`lora_A`, `lora_B`) for efficient fine-tuning. This modification helps reduce the number of trainable parameters by only updating a small subset of the model's parameters.

4.2. Training Process

The model is fine-tuned using a standard supervised training loop. We use the following setup:

- **Epochs**: The model is trained for 15 epochs.
- **Loss Function**: The loss function used is categorical cross-entropy, which is common for classification tasks.
- **Validation Accuracy**: After each epoch, the validation accuracy is calculated to monitor the model's performance on unseen data.
- **Best Model Saving**: The model weights are saved whenever the validation accuracy surpasses the previous best value.

The following results were recorded during training:

- **Epoch 1/15**: Loss = 0.728, Validation Accuracy = 48.5%
- **Epoch 2/15**: Loss = 0.705, Validation Accuracy = 53.0%
- **Epoch 3/15**: Loss = 0.706, Validation Accuracy = 47.0%
- **Epoch 4/15**: Loss = 0.698, Validation Accuracy = 49.75%
- **Epoch 5/15**: Loss = 0.694, Validation Accuracy = 49.75%
- **Epoch 6/15**: Loss = 0.694, Validation Accuracy = 49.0%
- **Epoch 7/15**: Loss = 0.696, Validation Accuracy = 49.75%
- **Epoch 8/15**: Loss = 0.694, Validation Accuracy = 51.25%
- **Epoch 9/15**: Loss = 0.691, Validation Accuracy = 49.25%
- **Epoch 10/15**: Loss = 0.692, Validation Accuracy = 49.0%
- **Epoch 11/15**: Loss = 0.678, Validation Accuracy = 49.5%
- **Epoch 12/15**: Loss = 0.684, Validation Accuracy = 47.5%
- **Epoch 13/15**: Loss = 0.686, Validation Accuracy = 48.5%
- **Epoch 14/15**: Loss = 0.680, Validation Accuracy = 46.25%
- **Epoch 15/15**: Loss = 0.679, Validation Accuracy = 46.75%

The training results show that the validation accuracy fluctuates between 46.25% and 53.0%, with no significant improvement in performance after the second epoch.

4.3. Model Weights and Parameters

The model architecture includes the LoRA adaptations to the attention layers, which include low-rank matrices (`lora_A`, `lora_B`) in addition to the standard weights of the transformer. After training, the model weights are loaded using `torch.load`, and the model is ready for inference.

`PeftModel` is used to encapsulate the GPT-2 model with LoRA adaptations, as shown in the following:

```
PeftModel(
  (base_model): LoraModel(
    (model): GPT2ForSequenceClassification(
      (transformer): GPT2Model(
        (wte): Embedding(50257, 768)
        (wpe): Embedding(1024, 768)
        ...
      )
      (score): Linear(in_features=768, out_features=2, bias=False)
    )
  )
)
```

4.4. Analysis of Results

The training results indicate that the model struggled to achieve high validation accuracy, with the highest accuracy observed being 53.0%. This fluctuation in performance can be attributed to several factors:

- **Learning Rate**: The learning rate might not be optimal for the task, as evident by the fluctuations in the validation accuracy.

- **Model Complexity**: While GPT-2 is a powerful model, the sequence classification task may not have provided sufficient complexity or data to allow for significant improvement in accuracy.
- **Insufficient Training**: The model may require more epochs for the weights to converge fully, or additional fine-tuning on a more suitable dataset.

5. BERT model

The BERT (Bidirectional Encoder Representations from Transformers) model is a pre-trained language model designed for a wide range of NLP tasks. It is based on the Transformer architecture, using a bidirectional attention mechanism to capture context from both directions. We employ the `BertForSequenceClassification` variant of BERT, which is fine-tuned on a downstream task for binary classification.

6. Model Architecture

The model architecture used in this experiment is as follows:

- **Embedding Layer:** The input to BERT is tokenized and passed through word, position, and token type embeddings. The input sequence is first converted into a vector of size 768, which represents the embedding of each token.
- **Encoder:** The model uses 12 layers of Transformer encoders. Each layer consists of self-attention followed by a feed-forward network. The self-attention mechanism in BERT attends to each word in the sequence from both directions.
- **Classifier:** The output of the final encoder layer is passed to a classification layer. This layer has two main parameters:
  - `classifier.bias`
  - `classifier.weight`

7. Model Training and Results

The model is trained for 20 epochs, with the following results:

- **Epoch 1:** Loss = 0.7051, Validation Accuracy = 41.5%
- **Epoch 2:** Loss = 0.6985, Validation Accuracy = 58.5%
- **Epoch 3:** Loss = 0.6950, Validation Accuracy = 41.5%
- **Epoch 4:** Loss = 0.6967, Validation Accuracy = 41.5%
- **Epoch 5:** Loss = 0.6979, Validation Accuracy = 41.5%
- **Epoch 6:** Loss = 0.6976, Validation Accuracy = 58.5%
- **Epoch 7:** Loss = 0.6938, Validation Accuracy = 41.5%
- **Epoch 8:** Loss = 0.6955, Validation Accuracy = 41.5%
- **Epoch 9:** Loss = 0.6936, Validation Accuracy = 41.5%
- **Epoch 10:** Loss = 0.6969, Validation Accuracy = 41.5%
- **Epoch 11:** Loss = 0.6976, Validation Accuracy = 58.5%
- **Epoch 12:** Loss = 0.6931, Validation Accuracy = 41.5%
- **Epoch 13:** Loss = 0.6930, Validation Accuracy = 41.5%
- **Epoch 14:** Loss = 0.6973, Validation Accuracy = 58.5%
- **Epoch 15:** Loss = 0.6963, Validation Accuracy = 41.5%
- **Epoch 16:** Loss = 0.6942, Validation Accuracy = 41.5%
- **Epoch 17:** Loss = 0.6923, Validation Accuracy = 41.5%

- **Epoch 18:** Loss = 0.6942, Validation Accuracy = 41.5%
- **Epoch 19:** Loss = 0.6971, Validation Accuracy = 41.5%
- **Epoch 20:** Loss = 0.6941, Validation Accuracy = 41.5%

The model demonstrates fluctuating accuracy, ranging from 41.5% to 58.5%. The best accuracy was recorded at 58.5% in Epochs 2, 6, 11, and 14. Despite this, the model does not show consistent improvement, indicating that further fine-tuning is needed.

8. Model Loading and Parameters

After training, the best model is saved based on validation accuracy. The loaded model configuration includes the following parameters:

- `BertForSequenceClassification` from the Hugging Face Transformers library.
- `classifier.bias` and `classifier.weight` parameters, which were initialized during training.
- `LoraModel` which applies low-rank adaptation (LoRA) to the self-attention layers.

8.1. Future Work and Improvements

Given the limitations observed in the results, the following improvements are proposed:

- **Hyperparameter Optimization**: A systematic grid search or random search approach can be employed to find the optimal learning rate, batch size, and LoRA rank for better performance.
- **Longer Training**: More training epochs, or using early stopping mechanisms, could help prevent overfitting and improve generalization.
- **Data Augmentation**: Incorporating more diverse or larger datasets might help the model generalize better to unseen sequences.
- **Advanced Regularization**: Techniques like dropout, weight decay, or gradient clipping might improve model stability during training.

8.2. Conclusion

In this work, we have demonstrated the adaptation of transformer for cipher text distinguishing classification. Despite some initial success in fine-tuning the model, the validation accuracy remained suboptimal, indicating that further work is required to improve the model's performance for classification tasks. Future exploration of hyperparameters, training duration, and model configurations can help overcome the observed limitations.

The BERT model shows some improvement during training, but its performance on validation data is inconsistent. Further fine-tuning and hyperparameter tuning may be necessary to improve its accuracy. Moreover, integrating LoRA with BERT may enhance its ability to generalize with fewer parameters.