CS771 - Assignment 1

Group: - Binary Brains
Group Members: -
201018 - Sumit Kumar
200073 - Akash Birthal
200439 - Hemant Singh Meena
201131 - Vishal Singh
200529 - Kuldeep Gupta
200922 - Shashvat Singham

Abstract

This report addresses the problem of breaking a sparse Conditional Delay Unit (CDU) Physical Unclonable Function (PUF) using a single sparse linear model. The mathematical derivation shows that for any sparse CDU PUF, there exists a sparse linear model that accurately predicts the responses. The task involves learning a sparse linear model using the provided data and implementing a code solution using only the numpy library. Various methods for solving the problem are explored, and the optimal hyperparameters are determined through experimentation. The report presents the findings and experiences in tackling this challenging problem.

1 Problem 1.1 (Sparse PUFs).

To demonstrate how a sparse CDU PUF can be broken by a single sparse linear model, we provide a detailed mathematical derivation. Our goal is to establish the existence of a mapping $\varphi : 0,1^D \to \mathbb{R}^D$ that maps D-bit 0/1-valued challenge vectors to D-dimensional feature vectors. Additionally, we aim to show that for any S-sparse CDU PUF, there exists an S-sparse linear model, denoted as $w \in \mathbb{R}^D$, such that $||w||_0 \leq S$, meaning w has at most S non-zero coordinates. Furthermore, we want to demonstrate that for all challenges $c \in 0,1^D$, the expression $w^\top \varphi(c)$ provides the correct response. It is important to note that no bias term, not even a hidden one, is allowed in the linear model.

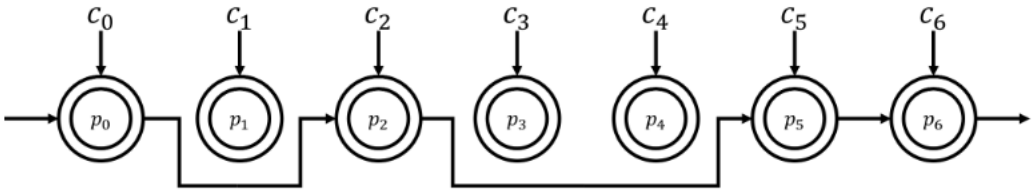

Figure 1: Sparse CDU PUF

1.1 Derivation

The linear model for the given PUF construction can be derived as follows:

Let $c_1, c_2, \ldots, c_D$ be the select bits representing the D CDUs in the pipeline, where $c_i \in \{0, 1\}$.

Let $p_1, p_2, \ldots, p_D$ be the delay values (in microseconds) associated with the CDUs.

The response of the PUF, denoted as R, is the total delay incurred, expressed in microseconds.

Considering the select bits of the active CDUs, we can express the response as:

$$R = \sum_{i=1}^{D} c_i \cdot p_i$$

However, to make the problem challenging, Melbo introduces dummy/irrelevant CDUs into the pipeline. Let S be the number of active CDUs (where $S < D$), and the remaining $D - S$ CDUs are disconnected from the chain.

To represent the active or not active state of the CDUs, we introduce an additional binary variable d_i, where $d_i = 1$ if the i-th CDU is active and $d_i = 0$ if it is disconnected.

Now, the response can be rewritten as:

$$R = \sum_{i=1}^{D} c_i \cdot p_i \cdot d_i$$

Since the select bits of the disconnected CDUs have no effect, their corresponding terms in the above equation will be multiplied by $d_i = 0$ and can be ignored.

Hence, the simplified linear model for the PUF response is:

$$R = \sum_{i=1}^{S} c_i \cdot p_i \cdot d_i$$

This linear model allows us to analyze and study the behavior of the PUF based on the select bits, delay values, and the active or not active state of the CDUs.

$$R = \sum_{i=1}^{S} c_i \cdot p_i \cdot d_i$$

$$R = \sum_{i=1}^{S} p_i \cdot d_i \cdot c_i$$

$$R = \sum_{i=1}^{S} w_i \cdot c_i$$

Here, $w_i = p_i \cdot d_i$ represents the weight associated with the i-th select bit.

Finally, we can express the linear model in the desired form:

$$R = \mathbf{W}^T \cdot \mathbf{c}$$

Where $\mathbf{W} = [w_1, w_2, \ldots, w_n]$ is the weight vector and $\mathbf{c} = [c_1, c_2, \ldots, c_n]$ is the select bit vector.

Therefore, the linear model in the form of W transpose multiplied by the select bit vector is:

$$R = \mathbf{W}^T \cdot \mathbf{c}$$

This form allows us to represent the PUF response R using a weight vector $\mathbf{W}$ and the select bit vector $\mathbf{c}$, enabling analysis and prediction based on the select bits and their corresponding weights.

Hence, The matrix product of w^T and x_i can be expressed as: The matrix product of w^T and x_i can be expressed as:

$$R = w^T \cdot x_i = \begin{bmatrix} w_1 & w_2 & \ldots & w_d \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}$$

The resulting matrix product will be a scalar value R, given by:

$$R = w_1 \cdot x_1 + w_2 \cdot x_2 + \cdots + w_d \cdot x_d$$

2 Problem 1.2 (The Code)

```
def HT( v, k ):
    t = np.zeros_like( v )
    if k < 1:
        return t
    else:
        ind = np.argsort( abs( v ) )[ -k: ]
        t[ ind ] = v[ ind ]
        return t

def my_fit( X_trn, y_trn ):
    w= np.zeros_like(X_trn[0])
    v= np.ones_like(X_trn[0])
    t=0.3   #step length
    lamda=0.01   #regularization constant
    for j in range(100):
        for i in range(1600):
            sgt = lamda*v +(1/1600 * X_trn[i]) * ( np.dot(X_trn
                [i],w.T)- y_trn[i] )
            w = w - t * sgt
        w= HT(w, 512)   # our model sparsity is 512
        t = t/1.02
    return w
```

3 Problem 1.3 (Different Methods)

We have submitted the final code on the basis of the Relaxation Technique. But we tried all three methods.
The Naive Technique is less effective than Relaxation Technique or Projected Gradient Descent. Its accuracy was coming around **76%**, and its mae was about 540. But, this method was very fast in terms of time complexity.
We also tried Projected Gradient Method. It's very effective in terms of mae and accuracy. But, time complexity was high due to the calculation of the gradient of the loss function. However, its mae was around **100**, and accuracy was **95%,** using Gradient Descent Method with projection step at the end of each iteration.

Using only **PGD,** makes the algorithm very slow, for that reason, we used the Correctin method as mentioned in the assignment, in which we compute $HT(z_t, S)$ as shown above, but instead of using it to update w_{t+1}, we find out the non-zero indices in $HT(z_t, S)$ (let's call this index set $I \subset [D]$, noting that $|I| \leq S$ by design).

Next, we solve a least squares problem using the data X_I, y, where $X_I \in \mathbb{R}^{n \times D}$ is a matrix prepared by zeroing out all columns of X that are not in the index set I.

Then, we update w_{t+1} by setting $w_{t+1,i} = u_i$ if $i \in I$, and $w_{t+1,j} = 0$ if $j \notin I$.

After correction, its time complexity decrease, but not as much as we are getting in the Relaxation technique.
We finally used **Relaxation Technique(SGD)**. We are updating the model with the help of Stochastic variants although the number of iterations in this method was very high than the Projected Gradient Descent Method. But, in terms of time complexity and accuracy, it was very effective than the Projected Gradient Descent Method.

4 Problem 1.4(Hperparameter and step length)

To choose the optimal values for the hyperparameters in the stochastic gradient method, such as the **regularization constant** (λ) and **step length** (t), the following approach was used:

1. Regularization Constant (λ):

- The value of λ was set to **0.01**.
- This value was chosen based on prior knowledge or domain expertise, indicating that a small regularization constant would be suitable for the problem at hand.
- No grid search or other optimization technique was used for λ.

2. Step Length (t):

- The initial step length was set to **0.3**.
- The step length was updated within the iterations using a diminishing factor.
- Specifically, the step length (t) was divided by 1.02 after each iteration, causing it to gradually decrease.
- This diminishing step length strategy helps to refine the optimization process and reach a potentially better solution over time.
- No specific criterion or optimization technique was used for selecting the initial or updated step lengths.