Assignment 2

Group Members

Name	Roll Numbers	Email
Ashish	200207	aashish20@iitk.ac.in
Ashish Bairwa	200208	ashishb20@iitk.ac.in
Aman Kumar	200098	amanku20@iitk.ac.in
Sharib Athar	200917	shariba20@iitk.ac.in
N Mohammed sohaib	200611	nmsohaib20@iitk.ac.in
Mahaveer khurkhuriya	200550	mahaveerk20@iitk.ac.in

11 July, 2023

1 Questions and their Answers

1.1 How a sparse CDU PUF can be broken by a single sparse linear model

To break a sparse CDU PUF using a sparse linear model, let's define the CDU PUF system first. We have D CDUs, out of which only S CDUs are active and participate in the response generation process and rest are disconnected.

Let p_i for $i = 0, 1, \ldots, D-1$ be the time delay when the signal passes through i-th CDU and the select bit is 1. Also, let $d_i \in \{0,1\}^D$ for $i = 0, 1, \ldots, D-1$ denote that i-th CDU is connected to the chain. $d_i = 1$ if i-th CDU is connected to the chain and 0 otherwise and let t_i denote the delay when the signal passes by $i-th$ CDU in the PUF (if $i-th$ CDU is disconnected, then t_i is the delay till previous connected CDU). The challenge vector is denoted by $\mathbf{c} \in \{0,1\}^D$, where each element c_i of $\mathbf{c}$ represents the select bit for a CDU. The response of the CDU PUF is the total delay incurred in the active CDUs, which is denoted by t_{D-1}.

Now, time delay when signal passes by the CDUs is given by:

$$t_0 = d_0 \cdot c_0 \cdot p_0$$

$$t_1 = t_0 + d_1 \cdot c_1 \cdot p_1 = d_0 \cdot c_0 \cdot p_0 + d_1 \cdot c_1 \cdot p_1$$

$$t_2 = t_1 + d_2 \cdot c_2 \cdot p_2 = d_0 \cdot c_0 \cdot p_0 + d_1 \cdot c_1 \cdot p_1 + d_2 \cdot c_2 \cdot p_2$$

.....

Similarly, the total delay at the last CDU is given by:

$$t_{D-1} = \Sigma_{i=0}^{D-1} d_i \cdot c_i \cdot p_i$$

Now, let's define the map $\phi : \{0,1\}^D \to \mathbb{R}^D$ as

$$\phi(c) = P \cdot \mathbf{c} = [c_0 \cdot p_0, c_1 \cdot p_1, \ldots, c_{D-1} \cdot p_{D-1}]$$

where P is a diagonal matrix of size DxD with $i-th$ diagonal P_{ii} element equal to p_i (the delay due to $i-th$ CDU). This maps the D-bit 0/1-valued challenge vector to a D-dimensional feature vector.

To break the sparse CDU PUF using a linear model, we need to find a sparse weight vector $\mathbf{w} \in \mathbb{R}^D$ such that $\|\mathbf{w}\|_0 \leq S$, where $\|\mathbf{w}\|_0$ represents the number of non-zero coordinates in $\mathbf{w}$. The weight vector $\mathbf{w}$ should be sparse, meaning it has at most S non-zero coordinates.

Now, let's define $\mathbf{w} \in \mathbb{R}^D$ such that $w_i = d_i$ for $i = 0, 1, \ldots, D-1$. From the design of PUF, we know that at most only S $d_i's$ can be 1 and rest will be 0. So, $\mathbf{w}$ is a sparse vector and satisfies $\|\mathbf{w}\|_0 \leq S$.

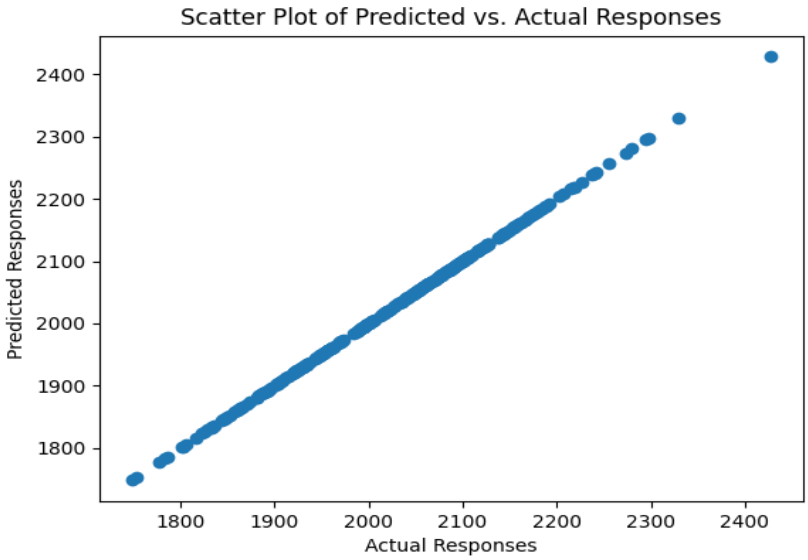
For all challenges $\mathbf{c} \in \{0,1\}^D$,

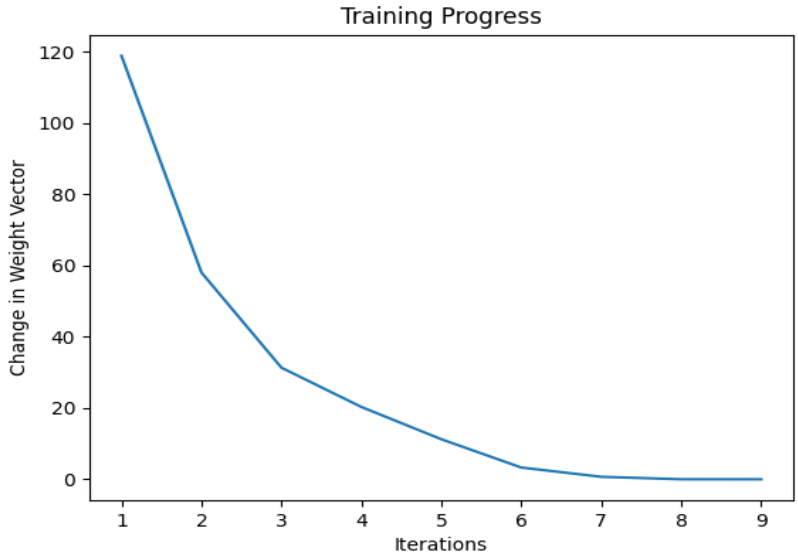
$$\mathbf{w}^\top \phi(\mathbf{c}) = [d_0, d_1, \ldots, d_{D-1}] \cdot [c_0 \cdot p_0, c_1 \cdot p_1, \ldots, c_{D-1} \cdot p_{D-1}] = \Sigma_{i=0}^{D-1} d_i \cdot c_i \cdot p_i$$

which is the exactly equal to the correct response t_{D-1} given by the sparse CDU PUF for the challenge vector $\mathbf{c} \in \{0,1\}^D$ as we have derived above.

Therefore, we have shown that for any S-sparse CDU PUF, there exists an S-sparse linear model $\mathbf{w} \in \mathbb{R}^D$ such that $\|\mathbf{w}\|_0 \leq S$ and $\mathbf{w}^\top \phi(\mathbf{c})$ gives the correct response for all challenges $\mathbf{c} \in \{0,1\}^D$.

1.2 Code submitted





1.3 Account of which method you tried and why you like this method over others. Describe at least one method other than u tried

We tried the naive technique, relaxation technique as well as projected gradient descent.

After implementing these approaches and testing on a validation set (we performed an 80-20 split on data and took the former as the train and the latter as validation data), we found that the projected gradient descent approach with correction gave lower MAE over the other two. We like projected gradient descent over others because:

1. The naive approach mentioned in the question involves solving the least squares problem without considering the sparsity constraint. After obtaining the solution, it applies hard thresholding to sparsify the solution. While this approach can produce a sparse solution, it is not optimized for sparsity. As a result, it gave a high MAE.

On the other hand, LASSO uses L1 regularization to promote sparsity, but it does not guarantee a sparse solution by itself. Additional steps, such as thresholding, are required to sparsify the LASSO solution.

In contrast, the projected gradient descent approach with correction explicitly includes the sparsity constraint in the optimization algorithm. By using the hard thresholding function within the optimization loop, the algorithm encourages sparsity by setting small coefficients to zero. This allows for a more direct and efficient optimization process tailored to the desired sparsity level.

2. The projected gradient descent approach optimizes the objective function by taking steps in the direction of the negative gradient, ensuring convergence to a local minimum. It leverages sub-gradient computations for the non-differentiable L1 regularization term. On the other hand, LASSO typically uses coordinate descent or proximal gradient descent algorithms to solve the optimization problem. While these methods are effective, we found out that they required more iterations to converge compared to projected gradient descent and also gave higher MAE.

3. The projected gradient descent approach has fewer hyperparameters to tune compared to the Coordinate/Proximal Gradient Descent. In the provided code, the hyperparameters are the learning rate, maximum iterations, and tolerance. Tuning these hyperparameters is relatively straightforward and can be done through a systematic search or using heuristics. LASSO, on the other hand, requires tuning the regularization parameter λ, which adds a layer of complexity in finding the optimal value.

1.4 Describe in detail how you chose the optimal values of the hyperparameters (regularization constant, step length, etc

We came up with the final values of hyperparameters using trial and error to observe convergence in mean absolute error (MAE) and time.

Initially, we used a learning rate (LR) of 0.01 and a maximum iteration (MI) of 1000, resulting in an observed MAE of 30.28. However, upon increasing the maximum iteration, we found that the MAE did not decrease further, indicating convergence for LR = 0.01 at an MI of less than 1000. Subsequently, we decreased the maximum iteration and found that the MAE converged for an MI of 2 with LR = 0.01.

Next, by increasing the LR to 0.05, we obtained an MAE of 27.52, which converged on increasing the MI to 7. Further increasing the LR to 0.1 while keeping the MI at 7, the MAE decreased to 18.38, and it converged at an MI of 9.

Continuing the trend, with an LR of 1, the MAE converged at an MI of 18 with a value of 3.17e-07. This represents the global minimum for MAE, regardless of further increases in LR or MI.

To minimize the time, we observed that changing the hyperparameters resulted in time minimization until a certain value of MI, beyond which the time started to increase.

By increasing the MI at LR=1, we found that the time converged at an MI of 30. Then, increasing the LR to 2, the time converged around an MI of 40. Finally, by further increasing the LR to 5, the time converged at an MI of 50.

These are the final values of the hyperparameters we used, as the time and MAE converged regardless of further changes in LR or MI.