CS771 : Introduction to Machine Learning Assignment - 1
Team: OnePUF

Nishant Gupta
220722
Department of Civil Engineering
IIT Kanpur
nishantg22@iitk.ac.in

Anaswar K B
220138
Dept. of Computer Science and Engineering
IIT Kanpur
anaswar22@iitk.ac.in

Garvit Choudhary
220400
Department of Civil Engineering
IIT Kanpur
garvit22@iitk.ac.in

Vanshika Yadav
221168
Dept. of Biological Sciences and Bioengineering
IIT Kanpur
vanshika22@iitk.ac.in

Devansh Bansal
220342
Department of Earth Science
IIT Kanpur
devanshb22@iitk.ac.in

Vaibhav Agarwal
221160
Department of Economics
IIT Kanpur
vaibhava22@iitk.ac.in

Abstract

This report presents our solution to the CS771 mini-project, which has two parts. First, we show how an ML PUF can be broken using a single linear model and evaluate various solvers, analyzing their performance and the effect of hyperparameters. In the second part, we interpret each learned model by generating 256 non-negative delay values that replicate the model's linear behavior in terms of arbiter PUF parameters.

1 Part 1

1.1 Analyzing Simple Arbiter PUF

Consider an Arbiter PUF (not the ML-PUF),

Let t_i^u be the time at which the upper signal leaves the i-th mux.

Let t_i^l be the time at which the lower signal leaves the i-th mux.

All muxes are different so that p_i, r_i, s_i and q_i are distinct.

Now,

$$t_i^u = (1 - c_i) \cdot (t_{i-1}^u + p_i) + c_i \cdot (t_{i-1}^l + s_i)$$
$$t_i^l = (1 - c_i) \cdot (t_{i-1}^l + r_i) + c_i \cdot (t_{i-1}^u + q_i)$$

Preprint. Under review.

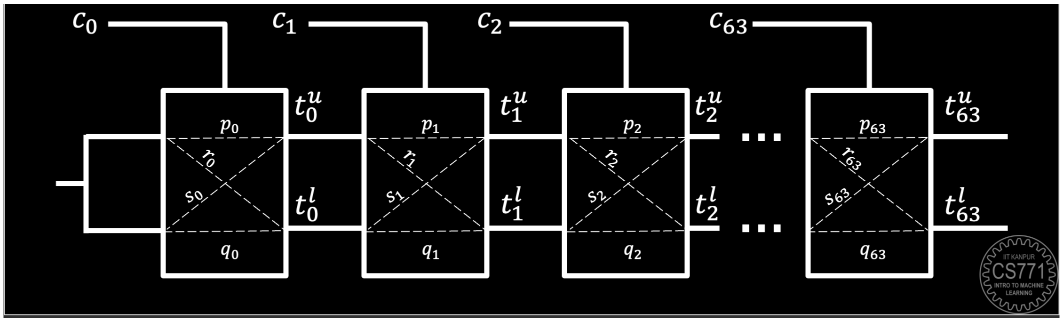

Figure 1: Analysis – Arbiter PUF

Let $\Delta_i = t_i^u - t_i^l$, then:

$$\begin{aligned}\Delta_i &= (1-c_i)\cdot(t_{i-1}^u + p_i - t_{i-1}^l - r_i) + c_i\cdot(t_{i-1}^l + s_i - t_{i-1}^u - q_i)\\ &= (1-2c_i)\cdot\Delta_{i-1} + (q_i - p_i + s_i - r_i)\cdot c_i + (p_i - q_i)\end{aligned}$$

Let $d_i = 1 - 2c_i$, then:

$$\begin{aligned}\Delta_i &= \Delta_{i-1}\cdot d_i + \alpha_i\cdot d_i + \beta_i\\ \alpha_i &= \frac{p_i - q_i + s_i - r_i}{2}, \quad \beta_i = \frac{p_i - q_i - r_i + s_i}{2}\end{aligned}$$

$$\begin{aligned}\Delta_0 &= \alpha_0\cdot d_0 + \beta_0 \quad \text{(since } \Delta_{-1} = 0)\\ \Delta_1 &= \Delta_0\cdot d_1 + \alpha_1\cdot d_1 + \beta_1\end{aligned}$$

Plugging in the value of Δ_0, we get:

$$\Delta_1 = (\alpha_0\cdot d_0)\cdot d_1 + (\alpha_1 + \beta_0)\cdot d_1 + \beta_1$$

$$\Delta_2 = \alpha_0\cdot d_2\cdot d_1\cdot d_0 + (\alpha_1 + \beta_0)\cdot d_2\cdot d_1 + (\alpha_2 + \beta_1)\cdot d_2 + \beta_2$$

Noticing the pattern,

$$\Delta_{n-1} = w_0\cdot x_0 + w_1\cdot x_1 + \cdots + w_{31}\cdot x_{n-1} + \beta_{n-1}$$

Where:

$$\begin{aligned}x_i &= d_i\cdot d_{i+1}\cdot\ldots\cdot d_{n-1}\\ w_0 &= \alpha_0\\ w_i &= \alpha_i + \beta_{i-1} \quad \text{for } i > 0\end{aligned}$$

This implies Δ_{n-1} is a linear function of $x_1, x_2, \ldots, x_{n-1}$.
Similarly, let $S_i = t_i^u + t_i^l$, then:

$$\begin{aligned}S_i &= S_{i-1} + (1-c_i)\cdot(p_i + q_i) + c_i\cdot(r_i + s_i)\\ S_i &= S_{i-1} + \gamma_i\cdot c_i + \delta_i\\ \gamma_i &= r_i + s_i - p_i - q_i, \quad \delta_i = p_i + q_i\end{aligned}$$

Taking $S_{-1} = 0$ and using the above equations we get:

$$S_{n-1} = \gamma_0\cdot c_0 + \gamma_1\cdot c_1 + \ldots + \gamma_{n-1}\cdot c_{n-1} + \delta$$

where $\delta = \sum_{i=0}^{n-1} \delta_i$. This implies S_{n-1} is a linear function of $c_1, c_2, \ldots, c_{n-1}$. Since $d_i = 1 - 2c_i$, S_{n-1} is a linear function of $d_1, d_2, \ldots, d_{n-1}$.

Now, $t_{n-1}^u = \frac{S_{n-1}+\Delta_{n-1}}{2}$ and $t_{n-1}^l = \frac{S_{n-1}-\Delta n-1}{2}$.

So t_{n-1}^u and t_{n-1}^l are linear functions of $d_0, d_1, \ldots, d_{n-1}, x_1, x_2, \ldots, x_{n-1}$. Also note that $d_{n-1} = x_{n-1}$, this will help in reducing dimensionality.

1.2 Cracking ML-PUF

Consider ML-PUF with n-bit challenge. Let T_0^u, T_0^l be the time taken by upper and lower signal of PUF0 and T_1^u, T_1^l be the time taken by upper and lower signal of PUF1.

The final output of ML-PUF is: $\frac{\text{sign}(T_0^u - T_1^u)+1}{2}$ XOR $\frac{\text{sign}(T_0^l - T_1^l)+1}{2}$, which can be written as $\frac{\text{sign}(-1 \cdot (T_0^u - T_1^u) \cdot (T_0^l - T_1^l))+1}{2}$. Now define ϕ as:

$$\begin{aligned}
\phi(\mathbf{c}) &= [d_0 \quad d_1 \quad \cdots \quad d_{n-1} \quad x_{n-2} \quad x_{n-3} \quad \cdots \quad x_0]^\top \\
&= [d_0 \quad d_1 \quad \cdots \quad d_{n-1} \quad d_{n-1}d_{n-2} \quad d_{n-1}d_{n-2}d_{n-3} \quad \cdots \quad d_{n-1}d_{n-2}\ldots d_0]^\top
\end{aligned}$$

$T_0^u, T_0^l, T_1^u, T_1^l$ can be written as $\mathbf{W}^\top \phi(\mathbf{c}) + b$. $T_0^u - T_1^u$ and $T_0^l - T_1^l$ can also be written in the form $\mathbf{W}^\top \phi(\mathbf{c}) + b$.

Therefore, the output of ML-PUF will be of the form $\frac{\text{sign}((\mathbf{W}_1^\top \phi(\mathbf{c})+b_1)\cdot(\mathbf{W}_2^\top \phi(\mathbf{c})+b_2))+1}{2}$. We need to convert this to a linear model,

The final feature map should have terms in the pairwise product of terms in $\phi(\mathbf{c})$ along with the product of b(a constant) with $\phi(\mathbf{c})$. We can ignore constant terms, as we consider a constant b in the final linear model. In addition, note that the terms in the product of b with $\phi(\mathbf{c})$ are already included in the pairwise product of the terms in $\phi(\mathbf{c})$. Also, since the terms in $\phi(\mathbf{c})$ are 1 or -1, the product of a term with itself will give a constant. We can also ignore those. This gives us the final map:

$$\tilde{\phi}(\mathbf{c}) = [\phi(\mathbf{c_0})\phi(\mathbf{c_1}) \quad \phi(\mathbf{c_0})\phi(\mathbf{c_2}) \quad \cdots \quad \phi(\mathbf{c_0})\phi(\mathbf{c_{n-1}}) \quad \phi(\mathbf{c_1})\phi(\mathbf{c_2}) \quad \cdots \quad \phi(\mathbf{c_{n-2}})\phi(\mathbf{c_{n-1}})]^\top$$

2 Part 2

For an ML-PUF with n-bit challenge, $\phi(\mathbf{c})$ has the following terms: $d_0, d_1, \cdots, d_{n-1}$ and $d_{n-1}, d_{n-1}d_{n-2}, \cdots, d_{n-1}d_{n-2}\dots d_0$. Here d_{n-1} appears twice, so the number of terms is $2n-1$.

$\tilde{\phi}(\mathbf{c})$ consist of pairwise product of terms in $\phi(\mathbf{c})$. So the dimensionality of $\tilde{\phi}$ is $\binom{2n-1}{2}$.

For $n = 8$, the dimensionality $\tilde{D}$ will be $\binom{15}{2} = 105$.

3 Part 3

The terms in $\tilde{\phi}(c)$ are monomials in $d_0, d_1, \ldots, d_{n-1}$. Since $d_i = 1 - 2c_i$, the terms are monomials in $c_0, c_1, \ldots, c_{n-1}$. A feature map consisting of all monomials in $c_0, c_1, \ldots, c_{n-1}$ is sufficient to represent this.

To implicitly compute the dot product in this high-dimensional feature space, we should use the **polynomial kernel**, which is defined as:

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^\top \mathbf{y} + r)^d$$

This kernel expands into a sum of monomials of degree up to d, matching our desired feature representation.

- **Degree:** $d = n$, since we want to include monomials of degree up to n.
- **Coefficient:** $r = 1$, to include all lower-degree monomials.

4 Part 4

For a k-bit arbiter PUF, we are given a linear model ($\mathbf{W} \in \mathbb{R}^k, b \in \mathbb{R}$) which predicts the output of the PUF. $\mathbf{W}, b$ are related to the delays of the PUF as follows:

$$w_0 = \alpha_0, \quad w_i = \alpha_i + \beta_{i-1} \quad \text{for } i > 0$$
$$b = \beta_{k-1}$$
$$\alpha_i = \frac{p_i - q_i + r_i - s_i}{2}, \quad \beta_i = \frac{p_i - q_i - r_i + s_i}{2}$$

We need to find one possible set of delays which gives the given linear model.

First, find the values of α_i, β_i for $0 \leq i \leq k-1$. To do this we can simply set $\beta_i = 0$ for $i < k-1$. Which gives us $\alpha_i = w_i$ $(0 \leq i \leq k-1)$ and $\beta_{k-1} = b$. We have obtained a possible set of values of $\alpha_i, \beta_i (0 \leq i \leq k-1)$.

Now, to obtain the values of delays(p_i, q_i, r_i, s_i), set $q_i = 0$ and $s_i = 0$ for all i. This gives us:

$$\alpha_i = \frac{p_i + r_i}{2}, \quad \beta_i = \frac{p_i - r_i}{2}$$

$$p_i = \alpha_i + \beta_i, \quad r_i = \alpha_i - \beta_i$$

We have obtained a set of values of delays which satisfies the given $\mathbf{W}, b$, but we have to ensure that the delays are non negative. For this we will use the following fact: if p_i, q_i, r_i, s_i $(0 \leq i \leq k-1)$ satisfies the constraints, then $p_i + \epsilon, q_i + \epsilon, r_i + \epsilon, s_i + \epsilon$ $(0 \leq i \leq k-1)$ also satisfies the constraints.

Let $\epsilon_0 = -\min_{i=0}^{k-1}(min\{p_i, q_i, r_i, s_i\})$. Then $p_i + \epsilon_0, q_i + \epsilon_0, r_i + \epsilon_0, s_i + \epsilon_0$ $(0 \leq i \leq k-1)$ are a set of non negative delays which satisfy the give constraints.

5 part 5

Code submitted

6 part 6

Code submitted

7 part 7

7.1 changing the `loss` hyperparameter in LinearSVC (hinge vs squared hinge)

The **loss function** defines how the model evaluates its prediction errors during training. The `LinearSVC` implementation from the `sklearn.svm` library offers two choices for the loss function: **hinge** and **squared hinge**.

The observations below uses the following hyperparameters: `C = 1`, `tolerance = 1e-3`, `penalty = L2`, `max_iter = 10000` and `dual = True`

Table 1: Changing Loss Hyperparameters in `LinearSVC`

Loss	Training Time (s)	Accuracy
Hinge	0.541	1.00
Squared Hinge	1.145	1.00

7.2 setting `C` in LinearSVC and LogisticRegression to high/low/medium values

The regularization parameter C in both `LinearSVC` and `LogisticRegression` controls the trade-off between achieving a low training error and enforcing regularization to prevent overfitting.

- **Small** C: Applies stronger regularization. The model prioritizes simplicity and generalization, allowing more misclassifications in training.
- **Large** C: Applies weaker regularization. The model fits the training data more closely, which may lead to overfitting if not tuned properly.

The observations below uses the following hyperparameters:

- **LinearSVC**: `tol=1e-3`, `penalty='l2'`, `max_iter=10000`, `loss='hinge'`, `dual=True`
- **LogisticRegression**:`solver='liblinear'`, `max_iter=1000`, `tol=1e-3`,`penalty='l2'`

Table 2: Changing `C` in LinearSVC and LogisticRegression

Model	C Value	Training Time (s)	Accuracy
LinearSVC	0.01	0.266	0.928
	0.1	0.176	1.000
	1	0.520	1.000
	10	0.537	1.000
	100	0.502	1.000
LogisticRegression	0.01	0.093	0.925
	0.1	0.166	0.998
	1	0.277	1.000
	10	0.483	1.000
	100	0.374	1.000

7.3 changing `tol` in LinearSVC and LogisticRegression to high/low/medium values

The `tol` parameter sets the tolerance for stopping criteria. It determines how small the change in the loss function must be before the optimization algorithm stops.

The observations below uses the following hyperparameters:

- **LinearSVC**: `C=1`, `penalty='l2'`, `max_iter=10000`, `loss='hinge'`, `dual=True`
- **LogisticRegression**:`solver='liblinear'`, `max_iter=1000`, `C=1`,`penalty='l2'`

Table 3: Changing `tol` in LinearSVC and LogisticRegression

Model	`tol` Value	Training Time (s)	Accuracy
LinearSVC	$1e{-}1$	0.184	1.000
	$1e{-}2$	0.390	1.000
	$1e{-}3$	0.684	1.000
	$1e{-}4$	0.856	1.000
	$1e{-}5$	1.102	1.000
LogisticRegression	$1e{-}1$	0.173	1.000
	$1e{-}2$	0.226	1.000
	$1e{-}3$	0.275	1.000
	$1e{-}4$	0.346	1.000
	$1e{-}5$	0.361	1.000

7.4 changing the `penalty` hyperparameter in LinearSVC and LogisticRegression (l2 vs l1)

The `penalty` parameter controls the type of regularization:

- `l2`: Adds squared weights to the loss, encouraging small, smooth coefficients.
- `l1`: Promotes sparsity by driving some weights to zero (feature selection).

The observations below uses the following hyperparameters:

- **LinearSVC:**`C=1`,`loss='squared_hinge'`,`tol=1e-3`, `dual=True/False`,`max_iter=10000`
- **LogisticRegression:** `C=1`, `solver='liblinear'`, `tol=1e-3`, `max_iter=1000`

Table 4: Effect of `penalty` on Training Time and Accuracy

Model	Penalty Type	Training Time (s)	Accuracy
LinearSVC	L1	6.501	1.000
	L2	1.290	1.000
LogisticRegression	L1	0.779	1.000
	L2	0.256	1.000

Note: In `LinearSVC`, the valid combination of `penalty`, `dual` are:

- For `penalty = 'l2'`: `dual = True`
- For `penalty = 'l1'`: `dual = False`