CS771: Introduction to Machine Learning
Assignment 1

Mohd Fahad (200591)

Aniket Sandhan (210924)

Harsh Bhati (200408)

Yash Yadav (201150)

Kartikeyan Iyer (200495)

Aditya Pandey (210063)

1 Question (Cross Connection PUF)

While doing some calculations in free time, Melbo realized something funny about arbiter PUFs. Recall that an arbiter PUF is a chain of k multiplexers, each of which either swaps the lines or keeps them intact, depending on what is the challenge bit fed into that multiplexer. The multiplexers each have delays which are hard to replicate but consistent. Let t^u , t^l respectively denote the time for the upper and lower signals to reach the finish line. At the finish line resides an arbiter (usually a flip-flop) which decides which signal reached first, the upper signal or the lower signal. The arbiter then uses this decision to generate the response. Although it is possible for a simple linear model to predict whether the upper signal will reach the finish line first or the lower signal (as we have seen in class), it seems that it is not so simple for a linear model to predict the time taken by the upper signal to reach the finish line, or predict the time taken by the lower signal to reach the finish line. Armed with this realization, Melbo created a new PUF variant and gave it the name **Cross-Connection PUF** or **COCO-PUF** for short.

A COCO-PUF uses 2 arbiter PUFs, say PUF0 and PUF1 – each PUF has its own set of multiplexers with possibly different delays. Given a challenge, it is fed into both the PUFs. However, the way responses are generated is different. Instead of the lower and upper signals of PUF0 competing with each other, Melbo makes the lower signal from PUF0 compete with the lower signal from PUF1 using an arbiter called Arbiter0 to generate a response called Response0. If the signal from PUF0 reaches first, Response0 is 0 else if the signal from PUF1 reaches first, Response0 is 1. Melbo also makes the upper signal from PUF0 compete with the upper signal from PUF1 using a second arbiter called Arbiter1 to generate a response called Response1. If the signal from PUF0 reaches first, Response1 is 0 else if the signal from PUF1 reaches first, Response1 is 1. Thus, on each challenge, the COCO-PUF generates two responses instead of one response.

Melbo wrong! You will do this by showing that there do exist linear models that can perfectly predict the responses of a COCO-PUF and these linear model can be estimated fairly accurately if given enough challenge-response pairs (CRPs).

The Task. The following enumerates 6 parts to the question. Parts 1,2,3,4,6 need to be answered in the PDF file containing your report. Part 5 needs to be answered in the Python file.

Preprint. Under review.

1.1 Question 1 :

1.1.1 Statement :

By giving a detailed mathematical derivation (as given in the lecture slides), show how for a simple arbiter PUF, a linear model can predict the time it takes for the upper signal to reach the finish line. Specifically, give derivations for a map $\phi : \{0,1\}^{32} \rightarrow R^D$ mapping 32-bit 0/1-valued challenge vectors to D-dimensional feature vectors (for some D > 0) so that for any arbiter PUF, there exists a D-dimensional linear model $\mathbf{W} \in R^D$ and a bias term $b \in R$ such that for all CRPs $\mathbf{c} \in \{0,1\}^{32}$, we have $\mathbf{W}^T\phi(\mathbf{c}) + b = t^u(\mathbf{c})$ where $t^u(\mathbf{c})$ is the time it takes for the upper signal to reach the finish line when challenge **c** is input. Remember that $t^u(\mathbf{c})$ is, in general, a non-negative real number (say in milliseconds) and need not be a Boolean bit. **W**, b may depend on the PUF-specific constants such as delays in the multiplexers. However, the map $\phi(\mathbf{c})$ must depend only on **c** (and perhaps universal constants such as 2, $2^{\frac{1}{2}}$ etc). The map ϕ must not use PUF-specific constants such as delays.

1.1.2 Solution Question 1:

We know that for each PUF model, each multiplexer takes time as following for its upper and lower signals to pass :

$$t_2^u = (1 - c_2) * (t_1^u + p_2) + c_2 * (t_1^l + s_2)$$

$$t_2^l = (1 - c_2) * (t_1^l + q_2) + c_2 * (t_1^u + r_2)$$

The approach here is to devide our problem in two models where one gives us the sum of $t^l \& t^u$ and other model gives us their difference.
Let $t^u = x$ & $t^l = y$ if we know both $x - y$ & $x + y$ we can easily get x and y both.

MODEL 1: Based on time difference of upper and lower signals:
Let us use the shorthand $\Delta_i = t_i^u - t_i^l$ to denote the lag.
Note: response is 0 if Δ_{64}< 0 else response is 1
$\Delta_2 = (1-c_2) * (t_1^u + p_2 - t_1^l - q_2) + c2 * (t_1^l + s_2 - t_1^u - r_2)$ = $(1 - 2c_2) * \Delta_1 + (q_2 - p_2 + s_2 - r_2) * c2 + (p_2 - q_2)$

To make notation simpler, let
$d_i \stackrel{\text{def}}{=} (1 - 2c_i)$
$\alpha_i \stackrel{\text{def}}{=} (p_i - q_i + r_i - s_i)/2$
$\beta_i \stackrel{\text{def}}{=} (p_i - q_i - r_i + s_i)/2$
$\Delta_2 = \Delta_1 * d_2 + \alpha_2 * d_2 + \beta_2$

A similar relation holds for any stage:
$\Delta_i = \Delta_{i-1} * d_i + \alpha_i * d_i + \beta_i$

We can safely take Δ_0 = 0 (absorb initial delays into p1, q1, r1, s1).
We can keep going on recursively:
$\Delta_1 = \alpha_1 * d_1 + \beta_1, \quad since\ \Delta_0 = 0,$
$\Delta_2 = \alpha_1 * d_2 + \alpha_2 * d_2 + \beta_2$
$= \alpha_1 * d_2 * d_1 + (\alpha_2 + \beta_1) * d_2 + \beta_2,$
$\Delta_3 = \alpha_1 * d_3 * d_2 * d_1 + (\alpha_2 + \beta_1) * d_3 * d_2 + (\alpha_3 + \beta_2) * d_3 + \beta_3,$
$\Delta_4 = \alpha_1 * d_4 * d_3 * d_2 * d_1 + (\alpha_2 + \beta_1) * d_4 * d_3 * d_2 + (\alpha_3 + \beta_2) * d_4 * d_3 + (\alpha_4 + \beta_3) * d_4 + \beta_4,$

It turns out that

MODEL 2: Based on time summation of upper and lower signals:
Let us use the shorthand $\sigma_i = t_i^u - t_i^l$ to denote the sum

$\sigma_2 = (1-c_2) * (t_1^u + p_2 + t_1^l + q_2) + c2 * (t_1^l + s_2 + t_1^u + r_2)$ = $\sigma_1 + (s_2 + r_2) * c2 + (p_2 + q_2) * (1 - c_2)$

To make notation simpler, let
$d_i \stackrel{\text{def}}{=} (1 - 2c_i)$
$a_i \stackrel{\text{def}}{=} (p_i + q_i - r_i - s_i)/2$
$b_i \stackrel{\text{def}}{=} (p_i + q_i + r_i + s_i)/2$
$\sigma_2 = \sigma_1 + a_2 * d_2 + b_2$

A similar relation holds for any stage:
$\sigma_i = \sigma_{i-1} * d_i + a_i * d_i + b_i$

We can safely take σ_0 = 0 (absorb initial delays into p1, q1, r1, s1).
We can keep going on recursively:
$\sigma_1 = a_1 * d_1 + b_1, \quad since\ \sigma_0 = 0,$
$\sigma_2 = \sigma_1 + a_2 * d_2 + b_2$
$= a_1 * d_1 + a_2 * d_2 + b_2 + b_1,$
$\sigma_3 = a1_1 * d_1 + a_2 * d_2 + a_3 * d_3 + b_1 + b_2 + b_3$

It turns out that

$\Delta_{32} = w^T x + b$	$\sigma_{32} = w^T x + b$
where $x = [x_1, .., x_{32}], w = [w_1, .., w_{32}]$ x & $w \in R^{32}$	where $x = [x_1, .., x_{32}], w = [w_1, .., w_{32}]$ x & $w \in R^{32}$
$x_i = d_i * d_{i+1} * ... * d_{32}$ $= (1 - 2c_i) * (1 - 2c_{i+1})...(1 - 2c_{32})$	$x_i = d_i$ $= (1 - 2c_i)$
$w_1 = \alpha_1$ $w_i = \alpha_i + \beta_{i-1} (for\ i = 2, 3, ..., 32)$ $b = \beta_{32}$	$w_1 = a_1$ $w_i = a_i (for\ i = 2, 3, ..., 32)$ $b = b_1 + b_2 + b_3..... + b_{32}$

Now we have our two models whose summation will give us the time taken by upper signal to reach final arbiter.

$$t_1^u = (\sigma_1 + \Delta_1)/2$$

similarly for any i,

$$t_i^u = (\sigma_i + \Delta_i)/2$$

Now we can clearly write this in terms of a map $\phi(c)$ which gives the relation for
$W^T\ \phi(c) + B = t_u(c)$
Lets define the terms in the final model for $t_u(c)$ as:
$2 * W = [w_1, ..., w3_1, a_1, a_2, a_3, ..., a_{32} + w_{32}]$

where $w_i = \alpha_i + \beta_{i-1}$ from model 1
and $a_i \stackrel{\text{def}}{=} (p_i + q_i - r_i - s_i)/2$ is from the model 2

NOTE: term of c multiplying with w_{32} and a_{32} is the same $(1 - 2 * c_{32})$which will be one term on expansion of linear model

$\phi(c) = [x_1, x_2, ..., x_{31}, d_1, d_2, ..., d_{32}]$

where $x_i = d_i * d_{i+1} * ... * d_{32}$ is from Model 1 and d_i is from Model 2.
lastly,
$2 * B = [\beta_{32} + b_1 + b_2 + ... + b_{32}]$

Thus we have a linear model $W^T \phi(c) + B = t_u(c)$ which gives us the value of $t_u(c)$ for all inputs.

1.2 Question 2 :

1.2.1 Statement :

What dimensionality does the linear model need to have to predict the arrival time of the upper signal for an arbiter PUF? The dimensionality should be stated clearly and separately in your report, and not be implicit or hidden away in some calculations.

1.2.2 Solution Question 2:

The dimensionality of W will be defined by no. of independent terms in $\phi(\mathbf{c})$.

$$\phi(\mathbf{c}) = [x_1, x_2, x_3, ...x_{31}, d_1, d_2, d_3.....d_{32}]$$

Here $x_{32} = d_{32} = (1 - 2 * c_{32})$

Hence they will combine to one tern whose coefficient in W will be given by $a_{32} + w_{32}$

Therefore the total terms are = 32 + 32 - 1 = 63

Hence we get dimensionality of W and D = **63**

1.3 Question 3 :

1.3.1 Statement :

Use the derivation from part 1 to show how a linear model can predict Response0 for a COCO-PUF. As in part 1, give an explicit map $\tilde{\phi} : \{0,1\}^{32} \rightarrow R^{\tilde{D}}$ a corresponding linear model $\tilde{\mathbf{W}} \in R^{\tilde{D}}, \tilde{b} \in R$ that predicts the responses i.e. for all CRPs $\mathbf{c} \in \{0,1\}^{32}$, we have $(1 + sign(\tilde{W}^T \tilde{\phi}(\mathbf{c}) + \tilde{b})/2 = r^0(\mathbf{c})$where $r^0(\mathbf{c})$ is Response0 on the challenge **c**. Similarly, show how a linear model can predict Response1 for a COCO-PUF. As before, your linear model may depend on the delay constants in PUF0 and PUF1 but your map must not use PUF-specific constants such as delays.

1.3.2 Solution Question 3:

We have $t_0^l - t_1^l = \hat{W}^T \phi(c) + \hat{B}$

Where $\hat{W} = W - \tilde{W}$ and $\hat{B} = B - \tilde{B}$
More specifically,

$2 * \hat{W} = [-w_1 + \tilde{w}_1, ..., -w_{31} + \tilde{w}_{31}, a_1 - \tilde{a}_1, a_2 - \tilde{a}_2, a_3 - \tilde{a}_3, ..., a_{32} - \tilde{a}_{32} - w_{32} + \tilde{w}_{32}]$

where $w_i = \alpha_i + \beta_{i-1}$ from model 1
and $a_i \stackrel{\text{def}}{=} (p_i + q_i - r_i - s_i)/2$ is from the model 2

Similarly $\phi(c) = [x_1, x_2, ..., x_{31}, d_1, d_2, ..., d_{32}]$

where $x_i = d_i * d_{i+1} * ... * d_{32}$ is from Model 1 and d_i is from Model 2.
Now simply Response0 will be $(1 + sign(\hat{W}^T \phi(c) + \hat{B}))/2$

Using the derivation from the first question, we can easily calculate the Response 1.

1.) Firstly, we will calculate the time taken by the upper signal of the PUF-1 to reach the arbiter1.Let's call it t_1^u

2.) Similarly we will calculate the time taken by the upper signal of the PUF-0. to reach the arbiter 1.Let's call it t_0^u

3.) To find the Response-1, we will calculate the difference of both these times i.e(t_0^u - t_1^u)

If this difference is negative, response-1 will be 0,else 1.

Response=((1 + sign(t_0^u - t_1^u))/2)

1.4 Question 4 :

1.4.1 Statement :

What dimensionality do your need the linear model to have to predict Response0 and Response1 for a COCO-PUF? This may be the same or different from the dimensionality you needed to predict the arrival times for the upper signal in a simple arbiter PUF. The dimensionality should be stated clearly and separately in your report, and not be implicit or hidden away in some calculations.

1.4.2 Solution Question 4:

The dimensionality of W will be defined by no. of independent terms in $\phi(\mathbf{c})$.

$$\phi(\mathbf{c}) = [x_1, x_2, x_3, ...x_{31}, d_1, d_2, d_3.....d_{32}]$$

Here $x_{32} = d_{32} = (1 - 2 * c_{32})$

Hence for both the Responce0 and Responce1 the dimensionality is 63

D=63

1.5 Question 6 :

1.5.1 Statement :

Report outcomes of experiments with both the sklearn.svm.LinearSVC and sklearn.linear model.LogisticRegression methods when used to learn the linear model. In particular, report how various hyperparameters affected training time and test accuracy using tables and/or charts. Report these experiments with both LinearSVC and Logisti- cRegression methods even if your own submission uses just one of these methods or some totally different linear model learning method (e.g. RidgeClassifier) In particular, you must report how at least 2 of the following affect training time and test accuracy:

1. changing the loss hyperparameter in LinearSVC (hinge vs squared hinge)
2. setting C in LinearSVC and LogisticRegression to high/low/medium values
3. changing tol in LinearSVC and LogisticRegression to high/low/medium values
4. changing the penalty (regularization) hyperparameter in LinearSVC and Logisti- cRegression (l2 vs l1)

You may of course perform and report all the above experiments and/or additional experiments not mentioned above (e.g. changing the solver, max_iter etc) but reporting at least 2 of the above experiments is mandatory.

1.5.2 Solution Question 6:

b) LinearSVC: Training Times, Mapping Times and Test Accuracies

C Value	Training Time(s)	Mapping Time(s)	Test Accuracy (acc0)	Test Accuracy (acc1)
Low (C = 0.1)	10.2265	1.9662	0.9787	0.9893
Medium (C =1)	11.5534	1.4720	0.9819	0.9927
High (C = 10)	14.5535	1.5053	0.9865	0.9932

Logistic Regression: Training Times, Mapping Times and Test Accuracies

C Value	Training Time(s)	Mapping Time(s)	Test Accuracy (acc0)	Test Accuracy (acc1)
Low (C = 0.1)	10.5976	1.5538	0.9781	0.9898
Medium (C =1)	16.5940	1.5318	0.9803	0.9926
High (C = 10)	24.1651	1.5277	0.9849	0.9937

c) LinearSVC: Training Times, Mapping Times and Test Accuracies

C Value	Tol	Training Time(s)	Mapping Time(s)	Test Accuracy (acc0)	Test Accuracy(acc1)
10	0.0001	14.5535	1.5053	0.9865	0.9932
10	0.001	14.4903	1.5433	0.9864	0.9934
10	0.01	14.6919	1.5256	0.9867	0.9936

Logistic Regression: Training Times, Mapping Times and Test Accuracies

C Value	Tol	Training Time(s)	Mapping Time(s)	Test Accuracy (acc0)	Test Accuracy(acc1)
10	0.0001	24.1651	1.5277	0.9849	0.9937
10	0.001	23.8034	1.6001	0.9849	0.9937
10	0.01	23.0892	1.5826	0.9849	0.9937