PROJECT REPORT

ON

Predicting the Outcome of Unpredictable Microelectronic Circuits

BY

Aditya Patkar

Nantanit Somboon

Carl Ostrenga

Suraj T C

TABLE OF CONTENTS

INTRODUCTION

This report presents the results of our group. We aimed to predict the output of a microelectronic circuit known as a physical unclonable function (PUF). PUFs take advantage of the manufacturing variations of microelectronic devices to produce an unpredictable output, called the "response", to each distinct input value, which is a Boolean vector and is also called the "challenge".

The specific type of PUF being studied is a delay-based PUF that has two delay paths and a comparator that compares the delay in the two paths. The routing of the delay paths in the circuit is dependent on the circuit's input, and the challenge vector provides the select bits for each stage of multiplexers (muxes). At the end of the circuit, the comparator determines the response, which is 1 if the upper signal arrives earlier and 0 otherwise.

The goal of this project was to predict the output of the PUF given a small number of queries. We used optimization theory and techniques to formulate and solve optimization problems to predict the PUF's output to any challenge vector. We worked as a group and tried as many formulations and numerical methods as possible.

In this report, we first provide formulations of different optimization problems involved, the numerical method tried and the evaluation results of predicting the outcome of the PUF.

METHOD 1: Logistic Regression

1. Formulation

We have a 64 stage PUF. The input vector considered is the challenge vector of size 64. We'll call it X. Our feature vector however, will be $\phi(x)$. The predicted output would be $\sigma(w^T\phi(xi))$ where σ is the sigmoid function and w is the weight vector.

The optimization problem involved will be to minimize the negative log likelihood function, defined as,

$$-\log L(w) = -\sum_{i=1}^{m}[y_i \log \sigma(w^T\phi(x_i)) + (1-y_i)\log(1-\sigma(w^T\phi(x_i)))]$$

Thus, the optimization problem is to find the set of weights that minimizes the negative log likelihood function.
Thus the weight vector w is our decision variable. And the objective function is as defined above. This is a convex optimization problem.

2. Numerical Methods

a. Stochastic Gradient Descent (from scratch)

Stochastic gradient descent (SGD) is a commonly used optimization algorithm for logistic regression. In SGD, instead of computing the gradient of the entire dataset, we iterate over the dataset and compute the gradient on a single sample for every step. This allows the optimization process to be more efficient.

The update rule for SGD is:

$w_{t+1} = w_t$ - alpha * gradient

where alpha is the learning rate, and gradient is the gradient of the loss function with respect to the model parameters, computed on the mini-batch of training data. The algorithm iteratively updates the model parameters w using the gradients computed.

For our problem of predicting the outcome of the PUF, following steps were performed.

1. Generate training data using random challenge vectors on the **puf_query** function.
2. Initialize weights (all random numbers)
3. Define learning rate. We used a learning rate of 0.1.
4. Iterate over the training data one at a time.
    a. Calculate $\phi(x_i)$
    b. Get the prediction y_pred using the sigmoid function on the product of w and $\phi(x_i)$
    c. Calculate the error.
    d. Calculate the gradient.
    e. Use the update rule mentioned above to calculate the new weights.
    f. Repeat until data is looped through

b. LBFGS: Limited-memory Broyden-Fletcher-Goldfarb-Shanno (built-in)

We used LBFGS method to calculate the optimal weights which minimizes the negative log likelihood, which is our loss function mentioned above.

This is a quasi-newton method. The algorithm maintains an approximation to the

inverse Hessian matrix of the objective function, which is used to guide the search for the optimal solution. The algorithm starts at an initial point and iteratively moves towards the minimum of the objective function, updating the approximation to the inverse Hessian matrix at each step. The limited memory refers to the fact that it only saves the gradient of last “m” steps unlike BFGS, thus making it faster.

We used the Scikit-Learn library for implementing the LBFGS.

Following steps were performed:

1. Generate training data using random challenge vectors on the **puf_query** function.
2. Import the scikit-learn library and create an object of the LogisticRegression class with the solver defined as “lbfgs”. Set the maximum number of iterations to 1000.
3. Calculate $\phi(x_i)$ and pass it along with the output vector of the training data to the logistic regression solver’s fit method.
4. Get the optimal weights.

3. Evaluation

Solver	Success Rate	EffectiveTraining Time	Training Size
SGD	0.9677	226.83	8350
LBFGS*	**0.9907**	**0.08**	**4500**

*Logistic regression using LBFGS method was our best method in terms of training size and effective training time.

P.T.O.

METHOD 2: Support Vector Machines

1. Formulation

To estimate w, you are using an SVM classifier, which can be formulated as an optimization problem. The SVM tries to find a hyperplane that maximally separates the data points in different classes, while also minimizing the classification error. The optimization problem can be written as:

$$\text{minimize } \frac{1}{2}||w||^2 + C\sum_{i=1}^{m}\xi_i \text{ subject to } \quad y^{(i)}(I(\phi(x^{(i)}))^\top w) \geq 1 - \xi_i, \quad i = 1, \ldots, m \; \xi_i \geq 0, \quad i = 1, \ldots, m$$

Where $y^{(i)}$ is the label, I is an identity function. $\phi(x_i)$ is our feature vector and ε is the correct margin boundary. This function basically maximizes the margin while incurring a penalty when a sample is misclassified or within the margin boundary.

2. Numerical Method: Sequential Minimal Optimization (built-in)

To solve this optimization problem, you are using the built-in function SVC from the Scikit-learn package, which uses the primal formulation of the SVM (shown above) and solves it using a variant of the Sequential Minimal Optimization (SMO) algorithm. As we can see our optimization problem is a QP. SMO is perfect for QP problems.

SMO breaks the overall QP problem into QP sub-problems. It works by selecting a pair of lagrange multipliers at each step and optimizing them while holding all other variables fixed. This optimization can be done analytically. The algorithm then checks if the updated Lagrange multipliers satisfy the KKT conditions, and if not, selects another pair of dual variables to optimize. At each step it updates the SVM to reflect the new optimal values.

3. Evaluation

Success Rate	Effective Training Time	Training Size
0.9919	0.2666	6000

P.T.O.

METHOD 3: Ensemble Learning

1. Formulation

For each ensemble model, we use Log-loss as our measurement for each iteration.

$$log\_loss = \frac{-1}{n}\sum_{i=1}^{n}(y_i \log p_i + (1-y_i)\log(1-p_i)$$

Where p is the probability of prediction being 1 (or 0). This is similar to the loss used in logistic regression. Our optimization problem is to minimize the log loss.

2. Numerical Methods

a. AdaBoost (built-in)

The idea of Adaboost is to train models while paying more attention to inputs with error outputs. With the first model, the inputs that gave incorrect outputs are noted and given more weight before going through the next iteration. The process is repeated until specific conditions are met (loss, number of iterations, scores, etc)
The iterative formula is:
Given $(x_1, y_1), .., (x_m, y_m)$ where $x_i \in$ X , $y_i \in$ {-1, 1} (Here X is phi_x in our problem):
Initialize: $D_1(i) = 1/m$ for i = 1, …,m.
For t=1, …, T:

- train weak learners using distribution D_t.
- Get weak hypothesis $h_t : X -> -1, +1$
- Aim: select h_t with low weighted error:
  $\varepsilon_t = Pr_{i \backsim D_t}[h_t(x_i) \neq y_i]$
- Choose $\alpha_t = 1/2 * ln(1 - \varepsilon/\varepsilon)$
- Update, for i = 1,…,m:

Output final hypothesis

$$H(x) = sign(\sum_{t=1}^{T} \alpha_t * h_t(x))$$

We can see that we first train the weak learner, pick the misclassified samples, and update the weight to make the classifier concentrate more on them. In contrast, decreasing the weight of the samples that are correctly classified.
In this case, we try to use AdaBoost with n_estimators = 50, learning_rate = 1.0

b. Gradient Boost (built-in)

GradientBoost uses similar ideas to AdaBoost. The main difference is GradientBoost sets a target on how a change in prediction affects the error. Essentially, if the next model's prediction can cause a significant drop in error when calculated with the previous model, the model is given a higher value or weight, and vice versa, if the change is small.

Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, (where xi is phi_xi) a differentiable loss function $L(y(F(x))$ which is log loss in our case, number of iterations M.

1. Initialize the model with a constant value
   $F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$
2. for m in M
   a. Compute
      $r_{im} = \frac{[\partial L(y_i, F(x_i))]}{\partial F(x_i)}_{F(x)=F_m-1(x)}$ for i = 1,...,n
   b. Train the base learner using $\{(x_i, r_{im})\}_{i=1}^{n}$
   c. Compute
      $\gamma_m = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i))$
   d. Update model
      $F_m(x) = F_{m-1}(x) + \lambda_m h_m(x)$
3. Output
   $F_m(x)$

3. Evaluation

Ada	Training Size	Success	Effective		Grad	Training Size	Success	Effective
	5000	0.9146	754.33			5000	0.8753	1148.3
	10000	0.9154	746.69			10000	0.8766	1136.6
	20000	0.9116	785.3			20000	0.875	1155.8
	30000	0.918	721.99			30000	0.8805	1103.7

Interestingly, the GradientBoost, which is more complex, has less success rate than AdaBoost.

P.T.O.

METHOD 4: Gradient Descent

1. Formulation

The strategy I employed for determining the response bit will be using gradient descent to perform an unconstrained optimization of ω. The optimization problem to be solved is written below.

Minimize ω e Rn $\sum_{j=1}^{N} \left(\left(\phi_j \cdot \omega_j > 0 \right) - y_j \right)^2$

2. Numerical Method: Gradient Descent

In order to numerically solve this problem, the gradient descent algorithm was constructed from scratch in MATLAB. The main gradient descent function written takes as inputs the following: an integer representing the number of epochs of training, a float learning rate, the 65x1 initial ω vector, an integer m representing the size of the training set, the mx65 phi matrix and y, an mx1 vector of the expected results from each phi sample. The core loop of the gradient descent function, which runs repeatedly till the set number of epochs is reached, first calculates the experimental result that the current ω vector produces by solving the $(\phi \cdot \omega > 0)$ term for the entire phi matrix. Since this is a Boolean operation, the result is an mx1 vector of 0 or 1 depending on if the result of a given index is greater than zero. The difference of the true result, y, and the current experimental result is taken, multiplied by phi elementwise and summed. This gradient value is then multiplied by a factor of the learning rate, and subtracted from ω to make the new weight vector. This is the initial basic gradient descent function before any improvements. Initially, after some bug fixing, with low training data, low number of training rounds, and a learning rate of .1, the model produced an accuracy of ~91%. In order to improve the model's performance and reliability, the data generation, model training and evaluation segments of the code were placed in a loop with training size, epochs and learning rate being declared outside. The loop would run the whole model process ten times and display the average success rate and training time, allowing for rapid tuning.

3. Evaluation

The results of a variety of possible parameter setups are displayed below.

Training Size	Epochs	Learning Rate	Average Success Rate	Average training time
500	5	.1	.907	825
5000	5	.1	.982	76.2
5000	50	.1	.992	0.536
5000	50	.01	.991	0.127
5000	50	.001	**.993**	**0.100**
500	50	.001	.923	668

The most interesting aspect was the importance of averaging results. Since the process terminated when the set number of epochs is reached there is no guarantee that the model produced a weight vector reaching 99% success, which meant that while 99% would often be quickly reached, the model has a high risk of producing sub 99% accurate results. This is the reason more developed models typically use a train test validation split or use a function to check if the model is improving on each epoch rather than a set number of epochs. Training size increases were the most effective change in terms of success rate, but also greatly increased the time to test the model, as dataset generation is a costly action.

P.T.O.

METHOD 5: Stochastic Gradient Descent

1. Formulation

The weight vector w can be estimated using the stochastic gradient descent (SGD) optimization algorithm. The objective of SGD is to minimize a loss function that measures the error between the predicted output and the actual output of the model. In this case, the loss function used is the binary cross-entropy loss, which is commonly used in binary classification problems.

The optimization problem for SGD can be written as follows:

$L(w) = -1/N * \sum [yi * \log(hw(xi)) + (1- yi) * \log(1 - hw(xi))]$

where,

- N is the total number of samples in the dataset
- xi is the feature vector of the i-th sample
- yi is the label of the i-th sample (either 0 or 1)
- hw(xi) is the predicted probability of the i-th sample being labeled as 1, given the model parameters w

The optimization problem is to find the optimal values of the parameters w that minimize the loss function L(w). The optimization problem can be written as:

minimize $L(w) = -1/N * \sum [yi * \log(hw(xi)) + (1- yi) * \log(1 - hw(xi))]$

where,

- w is the parameter vector that we want to find, and hw(xi) is the sigmoid function of the dot product of the parameter vector w and the feature vector xi.

The sigmoid function is defined as:
hw(xi) = 1 / (1 + exp(-wT * xi)) where exp(x) is the exponential function, and wT is the transpose of the parameter vector w.

2. Numerical Method

The optimization problem is solved iteratively using mini-batches of the training data, rather than the entire dataset at once. In each iteration, a mini-batch of m training samples is randomly selected from the dataset. The objective is then to update the parameter vector w using the gradient of the loss function with respect to w computed on the mini-batch. The update rule for the parameter vector is as follows:
w <- w - alpha * (1/m) * sum [(hw(xi) - yi) * xi]
where,

- alpha is the learning rate, which controls the step size in the direction of the gradient,
- m is the size of the mini-batch.

The SGD algorithm continues to update the parameter vector w iteratively until a stopping criterion is met, such as a maximum number of iterations or a minimum improvement in the loss function. The final estimated value of w is then used to predict the class labels for new, unseen data.

3. Evaluation

Success Rate	Effective Training Time	Training Size
0.9921	0.4268	4000

In this experiment, we used Stochastic Gradient Descent (SGD) to train a binary classifier on a synthetic dataset generated by the make_samples function from the PUFGenerator code. The optimization problem was formulated as minimizing the negative log-likelihood of the logistic regression model.

We experimented with different training set sizes ranging from 1000 to 10000 with a step size of 500. For each training set size, we trained the model using SGD for 10 iterations with a learning rate of 0.0001 and batch size of 1. The trained weights were evaluated for their success rate and effective training time.

Our best results were achieved for a **training set size of 4000** with a **success rate of 0.9921** and an **effective training time of 0.4268** seconds. The success rate of the model indicates the percentage of correctly classified instances on the test set. The effective training time is the time taken for the model to achieve the optimal weights.

We plotted two graphs to visualize the relationship between the training set size and the effective training time and success rate. The graph of training set size vs. effective training time shows a positive correlation between the two, indicating that as the training set size increases, the effective training time also increases. However, the increase in effective training time is not linear but rather seems to follow a logarithmic growth pattern. The graph of training set size vs. success rate shows a positive correlation between the two, indicating that as the training set size increases, the success rate of the model also increases. However, there seems to be

a diminishing marginal return on increasing the training set size, as the success rate starts to plateau beyond a certain point.

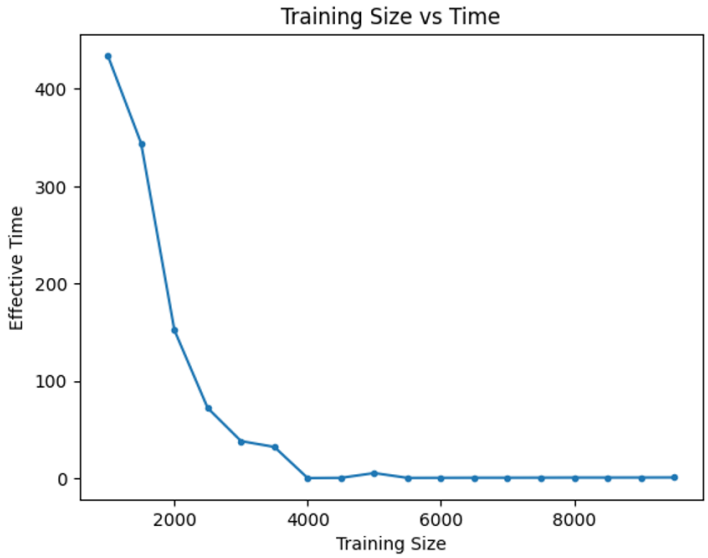

Figure 1: Training Size vs Time for SGD

Figure 2: Training Size vs Success Rate

Other Models Tried:

Model	Training Time (s)	Training Size	Success Rate	Effective Training Time (s)
XGBoost Classifier	2.059	4490	0.895	952.059
Multi Layer Perceptron	0.581	4490	0.9867	33.581

GROUP EVALUATION

1. Responsibilities

- **Aditya**
  - Making GitHub for code, Formatting Report, Creating Template for PPT
  - Methods: Logistic Regression and SVM
  - Coding, writing report and PPT slides for above methods
- **Nantanit**
  - Ensemble methods (AdaBoost and GradientBoost)
  - Coding, writing report and PPT slides for above methods
- **Carl**
  - Gradient Descent method
  - Coding, writing report and PPT slides for above methods
- **Suraj**
  - Methods: Neural Network, XGBoost and Stochastic Gradient Descent
  - Coding, writing report and PPT slides for above methods

2. Evaluation

Written By	**Review**
Suraj	All of us worked closely with each other to ensure that the data was properly preprocessed and that the performance metrics were accurately measured. Overall, each team member was able to contribute effectively to the project, and we were able to achieve a good result through collaboration and sharing of knowledge.
Nantanit	Everyone pitched in their ideas on what method to use, how they should work, and what the project expects from each of us. We all gave our best effort to deliver the project. I am happy to say that the group met every expectation. Be it cooperation, communication, contribution, and so forth.
Aditya	The group worked closely together and everyone played a part, sharing knowledge and ideas to ensure that the project goals were achieved. The collaboration and teamwork were critical to the success of the project, and we are proud of the results that we have achieved. Overall, each member of the team was able to contribute effectively, and the project's success was a testament to our collective efforts.
Carl	The group was very effective, had good communication and strong planning. Everyone worked effectively on their own and pulled their own weight. All of us produced very strong work and I am very proud of what we accomplished.

CONCLUSION

In this report, we have presented various solutions to the problem of estimating the weight vector of a Physical Unclonable Function (PUF).We used classical machine learning methods like Logistic Regression, SVM, SGD along with other methods which use the ensemble method of boosting. We came up with different formulations to achieve the ultimate goal of optimizing the set of weights to crack the internal logic of the delay based PUF.

We have shown that many methods can be used effectively to estimate the weight vector of a PUF, with a high success rate of 99% achieved in our experiments. We have also demonstrated the importance of training set size and effective training time in achieving high accuracy.

Overall, our results suggest that Logistic Regression with LBFGS is the most powerful tool for estimating the weight vector of a PUF, when considering the training size and the training time. Other methods also give good results. The biggest takeaway from this study is that you don’t need a complex overkill model to perform basic tasks. We should always start with less complexity and see if that is enough.