A Hybrid-Optimizer-Enhanced Neural Network Method for 
the Security Vulnerability Study of Multiplexer Arbiter PUFs

Khalid T. Mursi1,2 ✦, Bipana Thapaliya1 ✦, and Yu Zhuang1 ✦

1Department of Computer Science, Texas Tech University, Lubbock, Texas, USA
2College of Computer Science and Engineering, University of Jeddah, Jeddah, Saudi Arabia

Email: {khalid.mursi, bipana.thapaliya, yu.zhuang}@ttu.edu; kmursi@uj.edu.sa

Abstract. With the advent of the Internet of Things, security has become indispensable. Physical unclonable functions (PUFs) are emerging as a promising alternative to classical cryptographic algorithms as it provides a lightweight and cost-effective solution for implementing a keyless security mechanism. Before adopting a PUF for real-world applications, a thorough examination of all important properties of PUF is necessary, and security and reliability are two of the important properties. The multiplexer based PUF (MPUF) was recently designed to improve upon the reliability while maintaining a similar resistance to machine learning (ML) attacks as compared with XOR PUFs of certain sizes. Recently, feed-forward neural network (NN) methods were found to be an effective tool for studying PUFs’ security against ML attacks, and a study in 2019 found that some MPUFs are insecure against NN attack methods. In this paper, we try to gain further insight into various factors of NNs that affect the predictive power of NN as PUF attack methods. We investigate a NN that employs different optimizers at different stages of the machine learning process, leading to what is called a hybrid-optimizer-enhanced NN. We implemented the new NN for ML attack of MPUFs, and experimental results have shown it converges faster than a traditional NN with a single optimizer on attacking MPUFs, and the new method also requires less training data as compared with a recent NN-based attack study of MPUFs.

1. Introduction

From commercial and industrial activities to personal usage, the Internet of Things (IoT) has become ingrained in our everyday lives. Along-with the efficiency that it provides, it also raises a great deal of security and privacy issues. Physical unclonable functions (PUF) [1] are emerging as a promising candidate for IoT security because of its unclonability, device-dependent responses, and low resource requirements, unlike the classical cryptographic algorithms that are not adequately lightweight for resource-constrained devices and vulnerable to invasive side-channel attacks [2].

While physically unclonable, studies [3, 4] have shown that PUFs are mathematically clonable in the sense that the responses of the PUF can be predicted accurately using machine learning methods. The arbiter PUF (APUF) [5], a highly simplistic PUF, was found to be vulnerable to machine learning attacks shortly after it was proposed [3, 5, 6]. The XOR APUF (XPUF) [6, 7], out of an effort to improve security over the APUF, was found to fail to resist ML attacks [3, 4, 8, 23] if the number of components APUFs of an XPUF is not large enough. Increasing component APUFs in an XPUF can increase its

** Co-first authors, whose work was partially supported by National Science Foundation under Grant No. CNS-1526055.
security against ML attacks, but the reliability of XPUFs is shown to be decreasing as the number of component APUFs increases [9]. Multiplexer PUF (or MPUF) [10] was proposed to overcome XPUFs’ dilemma in decreased reliability with increased component APUFs for higher security. A security study performed in [10] shows MPUFs can resist linear cryptanalysis attacks. However, a recent study [11] has discovered that the MPUFs are vulnerable to the more powerful deep NN based attacks that exhibited high predictive power in earlier PUF vulnerability studies [4, 8].

In this paper, we dig further into the NN architecture to find ways that can speed up the training process while reducing the amount of required data. For machine learning, three of the major goals are high prediction accuracy, fast training, and a low amount of required training data, and in this paper, we present our work on speeding up training and reducing training data while maintaining high prediction accuracy for MPUF ML attack study. The main contributions of this paper are:

1. A hybrid optimizer for neural network, an optimizer that uses the popular Adam optimizer at beginning and mid-stages of the training and the L-BFGS optimizer at the later stage of the training.
2. New NN parameters, including layers, number of neurons, and activation functions.

The new neural network was implemented, and test results on MPUFs showed significant performance improvement over existing deep NN based attack methods in terms of both training speed and amount of training data when compared with experimental results in [11].

Before proceeding further, we briefly describe the MPUF. An MPUF consists of multiple APUFs. An n-bit APUF, illustrated in Figure 1, consists of 2-pairs of 2-1 multiplexers at each of the n stages of the PUF, with the output of two multiplexers at one stage fed to the two multiplexers at the next stage as input and the two output bits out of the last stage is fed to an arbiter, usually implemented by a D-latch. The two signals pass through the two paths, and slightly different delays incurred at different gates of the multiplexers causes the difference in arrival times of the top and bottom paths arriving at the arbiter [12]. If the top path arrives first, the output is 1, otherwise 0. An (n, k)-MPUF, illustrated in Figure 2, represents an MPUF with n-bit challenges and a MUX with k-selection inputs with a total of combine $2^k + k$ n-bit component APUFs.

The rest of the paper is organized as follows: In section 2, we present the methodology and design of the proposed hybrid optimizer enhanced NN. In section 3, we describe the experimental process, which includes machine learning attack studies. The results are discussed in the same section. Section 4 will be the conclusion.

2. Methodology and design

2.1. Background information on neural network

A feed-forward neural network, stand-alone or embedded in other neural networks (like in CNN), has been extensively used in machine learning studies. For simplicity, from here on, we use the neural network or NN to refer to a feed-forward neural network. For modeling objects with complex input-
output relations, multiple layers of neurons are needed for a neural network, and we call neural networks with 3 or more hidden layers deep neural networks.

When a trained feed-forward NN is used for inference or prediction, information flows in one direction from input to output layer via hidden layers [13]. The neurons at each layer receive input from the previous layer, and the inputs are multiplied by the weight of the connection they travel along. In this way, each unit sums up the input that it receives, and each neuron fires its output. After propagating crossed all hidden layers, it reaches the final output layer that gives the prediction values for the corresponding input to the input layer.

But before a neural network can be used for inference or prediction, it must be trained to meet adequate prediction accuracy on a training dataset. The training procedure is a nonlinear optimization procedure to minimize a loss function that evaluates the difference between the actual observed data and the predicted data. From weights of internal neuron inputs and outputs, the number of layers and the number of neurons at different layers, the choice of activation functions, an optimization algorithm, everything plays a role in the performance of the neural network.

2.2. The proposed hybrid-optimizer neural network
The optimization algorithm, or the optimizer, is central to the training of a neural network. From one perspective, optimization algorithms can be classified into first-order, or linear-convergence methods and higher-order or super-linear-convergence methods. First-order optimizers use the gradient of the loss function with respect to the parameters in each iteration (or epoch) to find the direction for reducing the loss function, e.g., stochastic gradient descent (SGD) and its popular variant Adaptive Moment Estimation (Adam) [14]. Superlinear-convergence methods use the Hessian or approximations of Hessian, in addition to the gradient, e.g., the Newton method and the Broyden–Fletcher–Goldfarb–Shanno method (BFGS), where the BFGS is an approximation to the second-order Newton method by approximating the inverse of the Hessian. The limited memory BFGS (L-BFGS) [15], one of the quasi-Newton methods suitable for large datasets, is an approximation to the BFGS by performing rank-two updates to the inverse Hessian approximation of BFGS. It is known that when the solution of an

![Figure 2. Multiplexer-based Arbiter PUF](image_url)
optimization problem is close to the optimality, the Newton method converges at a quadratic convergence, and the BFGS has a super-linear convergence, both significantly faster than linear convergence. We believe that the L-BFGS, not the linear-convergent stochastic L-BFGS [16], might also have super-linear convergence in many cases. Though of higher convergence, it is possible that each epoch of the Newton method or its approximation method like L-BFGS could incur high computation cost than one epoch of some first-order optimization methods.

From another perspective, optimization algorithms can be classified into stochastic optimization methods like SGD and Adam, and the usual, non-stochastic, optimization methods like gradient descent, the Newton method, and quasi-Newton methods like the BFGS and L-BFGS methods. Non-stochastic optimization methods, like L-BFGS, are known to converge to the local minimum, which may not necessarily be the global minimum. Since a local minimum of a convex function is its global minimum, the L-BFGS will converge surely to the global minimum when the loss function is convex but may not converge to the global minimum when otherwise. On the other hand, stochastic optimization methods always converge to the global minimum whether the loss function is convex or not [17]. Thus, stochastic optimizers are safe choices for convergence. In addition, by partitioning the whole dataset into multiple mini-batches on which a stochastic optimizer iterates through during each epoch, for large datasets, a stochastic optimizer is expected to have higher memory reference performance in accessing data due to small sizes of mini-batches that can fit more easily into cache memories.

From the two perspectives discussed in the two preceding paragraphs, stochastic and non-stochastic optimizers have their comparative advantages and disadvantages. It is known from optimization theory that in a neighborhood of a minimum, including the global minimum, the loss function is approximately a quadratic function. Since a quadratic function is convex, we are motivated to surmise that non-stochastic optimizers like BFGS and L-BFGS will always converge in a neighborhood of the global minimum, that is, convergence to a global minimum is guaranteed when the solution is close to a global minimum. In addition, BFGS is known to have super-linear convergence in a neighborhood of all minima, including the global minimum, the classical L-BFGS, the approximate BFGS is hence expected to have faster convergence than first-order optimizers like the Adam.

Along the line of thought discussed in the preceding paragraph, we are motivated to construct a hybrid optimizer that uses the Adam at the beginning of the optimization process when the solution is far away from the global minimum, but switches to the L-BFGS when the solution is close to the global minimum. The key is to decide when to make the switch, that is, what is the event observable in the training process of a neural network that triggers the switch from Adam to L-BFGS. In this paper for MPUFs, we examined multiple trainings of neural networks, and find that the loss function reaches 0.4 or lower is a good triggering event that is easily observable with little computation cost, and the resulting training procedure with the hybrid optimizer is given below:

```
WHILE loss increases for less than num_patience
  IF loss > 0.4 THEN perform training with Adam
  ELSE perform training with L-BFGS
ENDWHILE
```

2.3. Other network parameters
Activation functions also play a significant role in neural networks. In our problem, we found Sigmoid activation function $g(x) = \frac{1}{1+e^{-x}}$ to be working the best in the output layer [18]. For the hidden layers, the Hyperbolic tangent (tanh) activation function was found to be working better than other activation functions like the rectified linear unit (ReLU) [19] which was shown to perform well in our earlier
studies [4, 8] and was also recognized by others in a recent study [11]. The tanh function is defined as the ratio of the half-difference and half-sum of two exponential functions
\[
\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}
\]
and its value lies in the range of [-1, 1] [18]. This range is well suited for the type of challenge bits transformed by a reverse accumulative product that leads to a linear classification model for the arbiter PUF.

With the new activation functions, accordingly adjusted are the values for the rest of the neural network parameters, such as learning rate, loss function, initializers, number of hidden layers and the number of neurons in each hidden layer, etc. as listed in Tables 1 and 2.

3. Experimental studies

3.1. Simulation setup
In our simulation setup, we generated the challenge-response pairs (CRPs) for multiplexer based PUF using the C programming implementation. The CRPs are generated such that the challenges are randomly drawn from the range \((0, 2^n)\) where \(n\) is the number of bits of a challenge. The arbiter delays for the APUF are considered to be independent and identically distributed, and they follow a standard normal distribution. We take mean as 10 and standard deviation as 0.05. The generated challenge was applied to the PUF circuits as input to receive the corresponding response. The modeling experiments were performed on 64-bits and 128-bits PUF variants. We generated CRPs for different types of MPUFs ((\(n, 3\)), \((n, 4)\), and \((n, 5)\)) for both 64-bits and 128-bits.

3.2. Machine learning attacks
We have implemented the proposed NN with the hybrid optimizer and carried out a machine learning attack experimental study. In the study, the specifications of the NN method that have been used are summarized in Table 1, and the NN architecture is specified in Table 2. For comparison on optimizers, we have used the proposed hybrid optimizer, the Adam optimizer, and the L-BFGS optimizer. The generated CRP dataset was split with a ratio of 80% and 20% as a training set and testing set. Firstly, we trained our network using the training dataset and test the performance of our network using the test data.

| Hyperparameters | Description |
|-----------------|-------------|
| Optimizer       | Adam stop when loss <= 0.40 |
|                 | LBFGS stop when loss <= 0.18 |
| Learning rate   | Adaptive |
| Bias Initializer| Zeros |
| Weight Initializer | Xavier Normal |
| Loss function   | Binary Cross Entropy |
| Hidden layer activation function | Hyperbolic Tanh |
| Output layer activation function | Sigmoid |

| Challenge size | MPUF type | No. of Hidden layers | No. of Neurons |
|----------------|-----------|---------------------|----------------|
| 64/128-bits    | k=3       | 4                   | [16, 32, 32, 16] |
|                | k=4       | 4                   | [32, 64, 64, 32] |
|                | k=5       | 4                   | [32, 64, 64, 32] |
Adam only as an optimizer, L-BFGS only as an optimizer and the proposed hybrid optimizer. We

Our observed performance of Adam can probably be explained by the studies [16, 22] that though the first-order stochastic methods like Adam often make rapid progress early on, the variance of the estimates of the gradient slows their convergence near the optimum. And the fast convergence exhibited after switch confirms that quasi-Newton optimizers can converge much quicker when it is near the global minimum.

Table 3 lists the accuracy, computation time, and the number of epochs during the training for MPUF with selection inputs, $k$, ranging from 3 to 5 for 64-bit and 128-bit input implemented using 3 methods: Adam only as an optimizer, L-BFGS only as an optimizer and the proposed hybrid optimizer. We

| No. of bits | No. of $k$ | HYBRID | L-BFGS only | Adam only |
|-------------|------------|--------|--------------|-----------|
|             |            | Accuracy | Time | Epc. | Accuracy | Time | Epc. | Accuracy | Time | Epc. |
| 64-bit      | 3          | 98.69%   | 21s  | 55   | 98.72%   | 7s   | 13   | 98.01%   | 38s  | 75   |
|             | 4          | 98.51%   | 47s  | 66   | 98.73%   | 65s  | 34   | 97.56%   | 82s  | 151  |
|             | 5          | 98.11%   | 1m 52s | 98  | 97.82%   | 2m 16s | 70  | 96.72%   | 6m 25s | 212  |
| 128-bit     | 3          | 98.47%   | 33s  | 58   | 98.08%   | 18s  | 21   | 97.36%   | 46s  | 101  |
|             | 4          | 98.58%   | 1m 37s | 64  | 98.09%   | 2m 26s | 49  | 98.26%   | 1m 55s | 96   |
|             | 5          | 98.02%   | 4m 41s | 99  | 97.77%   | 7m 16s | 84  | 98.12%   | 5m 26s | 138  |

Table 4. Modeling accuracy of different MPUF types comparison with [11].

| No. of bits | No. of $k$ | HYBRID | Results from [11] |
|-------------|------------|--------|-------------------|
|             |            | CRPs   | Accuracy | Time | CRPs   | Accuracy | Time |
| 64-bit      | 3          | 35K    | 98.69%   | 21s  | 111K   | 98.10%   | 2m 5s |
|             | 4          | 50K    | 98.51%   | 47s  | 176K   | 97.44%   | 4m 31s|
|             | 5          | 100K   | 98.11%   | 1m 52s | 256K   | 97.02%   | 14m 13s|
| 128-bit     | 3          | 50K    | 98.47%   | 33s  | 112K   | 97.50%   | 3m 23s|
|             | 4          | 100K   | 98.58%   | 1m 37s | 184K   | 96.49%   | 16m 10s|
|             | 5          | 200K   | 98.02%   | 4m 41s | 312K   | 96.40%   | 22m 43s|

The implementation code was written in Python 3.7, and the Keras [20] framework with PyTorch [21] as backend. Experiments were executed on a Macbook Pro with 2.6 GHz 6-Core Intel Core i7 and a memory capacity of 16 GB. Serial execution was performed.

3.3. Experimental results
We have listed our experimental results under two tables: Table 3 and Table 4. The first Table lists the comparison of the proposed method with the traditional approach where a single optimizer is used, but with the same NN parameters, whereas the latter lists the comparison of the proposed method with the one used in [11] which has different NN parameters like the activation function, number of hidden layers and neurons, etc. The accuracy of the hybrid method denotes the final accuracy that we receive after using both Adam and L-BFGS optimizer. The threshold that we use for the selection of these optimizers is based on the loss values that the model generates. Adam threshold is set to 0.4, and LBFGS is set to 0.18.

We carefully evaluated the training process of the hybrid optimizer and observed that when the loss value was close to 0.4 but right before switching to L-BFGS, the validation in the training process gives an accuracy of around 90%, meaning it is near the global minimum but has not reached there yet. We also examined the training processes of Adam only network and found that with further epochs under, the training accuracy changes slowly when near 90% and takes a long time to improve after that. But for the hybrid optimizer, as soon as the switch takes place, the L-BFGS optimizer takes very little time to improve accuracy further and find the global minimum. Our observed performance of Adam can probably be explained by the studies [16, 22] that though the first-order stochastic methods like Adam often make rapid progress early on, the variance of the estimates of the gradient slows their convergence near the optimum. And the fast convergence exhibited after switch confirms that quasi-Newton optimizers can converge much quicker when it is near the global minimum.

Table 3 lists the accuracy, computation time, and the number of epochs during the training for MPUF with selection inputs, $k$, ranging from 3 to 5 for 64-bit and 128-bit input implemented using 3 methods: Adam only as an optimizer, L-BFGS only as an optimizer and the proposed hybrid optimizer. We
observe that, when the training size is small, “L-BFGS only” as the optimizer selection performs the best since the second-order optimization algorithm is known to have rapid convergence. However, as the training size and complexity increase, the proposed method seems to outperform the rest. Note that in Table 3, the NN parameters and the number of CRPs used for all three methods are the same.

Table 4 lists the prediction accuracies, computation time, and the number of CRPs required for the proposed attack method and the method used in [11]. The data in the table clearly show that as the component PUF in MPUF increases, the number of CRPs and the time required to attack such PUFs also increase. For all the MPUF types, our method reported much fewer CRPs requirement than in [11], up to 3.5 times less CRPs was required for breaking the 4-MPUF 64-bit, for instance. Besides using a much smaller set of CRPs, the computation time was significantly shorter than [11]. For example, for larger MPUFs like 5-MPUF 64-bits, the CRP size was reduced from 256K to 100K and the computation time from 14 min 13 sec to 1 min 52 sec (7.6 times), which is remarkable. Therefore, the proposed hybrid optimizer implemented on our NN architecture significantly outperforms the previous method in terms of both training time and the size of the training dataset needed for the successful training. Along with the efficiency of the proposed method, the results also verify that the reduction in the time and CRPs show that even larger MPUFs are not secure enough to withstand machine learning attacks.

4. Conclusion

In this paper, we have proposed a new machine learning procedure by using a hybrid-optimizer-enhanced NN. The procedure employs a learning method that utilizes the benefits of the first-order and second-order optimization algorithms and modifications to other NN parameters. Our findings indicate the efficiency of the proposed method, as the model converges much faster, and converges with smaller sets of training data. The proposed method also has found the vulnerability of MPUF circuits to be more severe than previous studies have revealed.

Acknowledgments

Khalid T. Mursi is supported by University of Jeddah (UJ) and the Saudi Arabian Cultural Mission (SACM). The research was partially supported by the National Science Foundation under Grant No. CNS-1526055.

References

[1] B. Gassend, D. Clarke, M. Van Dijk and S. Devadas, "Controlled physical random functions," in 18th Annual Computer Security Applications Conference, 2002. Proceedings., 2002.
[2] C. Herder, M.-D. Yu, F. Koushanfar and S. Devadas, "Physical unclonable functions and applications: A tutorial," Proceedings of the IEEE, vol. 102, p. 1126–1141, 2014.
[3] U. Rührmair, F. Seahke, J. Solter, G. Dror, S. Devadas and J. Schmidhuber, "Modeling attacks on physical unclonable functions," in Proceedings of the 17th ACM conference on Computer and communications security, 2010.
[4] A. O. Aseeri, Y. Zhuang and M. S. Alkatheiri, "A machine learning-based security vulnerability study on xor pufs for resource-constraint internet of things," in 2018 IEEE International Congress on Internet of Things (ICIOT), 2018.
[5] D. Lim, J. W. Lee, B. Gassend, G. E. Suh, M. Van Dijk and S. Devadas, "Extracting secret keys from integrated circuits," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 13, p. 1200–1205, 2005.
[6] U. Rührmair, J. Sölter, F. Seahke, X. Xu, A. Mahmoud, V. Stoyanova, G. Dror, J. Schmidhuber, W. Burleson and S. Devadas, "PUF modeling attacks on simulated and silicon data," IEEE transactions on information forensics and security, vol. 8, p. 1876–1891, 2013.
[7] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in 2007 44th ACM/IEEE Design Automation Conference, 2007.
[8] K. T. Mursi, Y. Zhuang, M. S. Alkatheiri and A. O. Aseeri, "Extensive Examination of XOR Arbiter PUFs as Security Primitives for Resource-Constrained IoT Devices," in 2019 17th
International Conference on Privacy, Security and Trust (PST), 2019.

[9] M.-D. Yu, D. M’Raïhi, I. Verbauwhede and S. Devadas, "A noise bifurcation architecture for linear additive physical functions," in 2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), 2014.

[10] D. P. Sahoo, D. Mukhopadhyay, R. S. Chakraborty and P. H. Nguyen, "A multiplexer-based arbiter PUF composition with enhanced reliability and security," IEEE Transactions on Computers, vol. 67, p. 403–417, 2017.

[11] P. Santikellur, A. Bhattacharyya and R. S. Chakraborty, "Deep Learning Based Model Building Attacks on Arbiter PUF Compositions," 2019.

[12] S. Tajik, E. Dietz, S. Frohmann, J.-P. Seifert, D. Nedospasov, C. Helfmeier, C. Boit and H. Dittrich, "Physical characterization of arbiter PUFs," in International Workshop on Cryptographic Hardware and Embedded Systems, 2014.

[13] A. Shrestha and A. Mahmood, "Review of deep learning algorithms and architectures," IEEE Access, vol. 7, p. 53040–53065, 2019.

[14] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv preprint arXiv:1412.6980, 2014.

[15] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," Mathematical programming, vol. 45, p. 503–528, 1989.

[16] P. Moritz, R. Nishihara and M. Jordan, "A linearly-convergent stochastic L-BFGS algorithm," in Artificial Intelligence and Statistics, 2016.

[17] S. Sun, Z. Cao, H. Zhu and J. Zhao, "A Survey of Optimization Methods from a Machine Learning Perspective," IEEE transactions on cybernetics, 2019.

[18] B. Karlik and A. V. Olgac, "Performance analysis of various activation functions in generalized MLP architectures of neural networks," International Journal of Artificial Intelligence and Expert Systems, vol. 1, p. 111–122, 2011.

[19] X. Glorot, A. Bordes and Y. Bengio, "Deep sparse rectifier neural networks," in Proceedings of the fourteenth international conference on artificial intelligence and statistics, 2011.

[20] F. Chollet, keras, GitHub, 2015.

[21] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga and A. Lerer, "Automatic differentiation in PyTorch," in NIPS-W, 2017.

[22] A. S. Berahas, J. Nocedal and M. Takáč, "A multi-batch L-BFGS method for machine learning," in Advances in Neural Information Processing Systems, 2016.

[23] K. T. Mursi, B. Thapaliya, Y. Zhuang, A. O. Aseeri and M. S. Alkatheiri "A Fast Deep Learning Method for Security Vulnerability Study of XOR PUFs," Electronics, 2020; 9(10):1715.