Lab 2: Machine Learning Attacks on a PUF

Derek Caprio and Igli Duro
Department of Computer Science & Engineering
University of South Florida
Tampa, FL

I. Introduction

For this lab, we use TensorFlow and SciKit-Learn machine learning tools to model the behavior of a PUF so that its security is broken. The performance of different models are tested and compared, including the neural network provided in modelPUF.py, SVM, KNN, and Naive Bayes. Furthermore, we test to see how XOR-ing PUF outputs together makes the behavior more difficult to model. Finally, different train/test split ratios are used with the models to demonstrate how this can affect a model's accuracy relative to the PUF as well.

II. Methods

This project provided the challenges_100000.npz file as well as the modelPUF.py file. We used the challenges_100000.npz file with the APUF_CMOD_S7_01.bit bitstream from lab 1 on the CMOD-S7 FPGA to get 100,000 responses from the PUF using the acquisition.py program from lab 1. With 100,000 challenges and responses, We then created the FormatChallengeResponse.py program to build .npz files of challenge-response pairs. All CRPs are saved in hexidecimal and in columns labeled 'challenge' and 'response' to suit the format required by modelPUF.py.

After the CRP files have been created, we are ready to train and evaluate some PUF models. We created our own program, modeling.py and imported the modelPUF.py program provided. After the CRP file is read, a pufModel class is created using the class provided in modelPuf.py. The data is then split into training and testing sets using the train_test_split function also provided in modelPUF.py. The value on this line is changed for different experiments as needed for different split ratios. Finally, the PUF is trained and tested using the functions provided. This test function also provides the final accuracy when complete.

The modeling_graph.py is given the accuracy from three trials at varying testing percentages and graphs the results.

III. Results

A. Reading Check

1. *What makes neural networks good for modeling PUFs?*
Machine learning, and especially neural networks, are ideal methods of modeling complex systems which are difficult to describe. They are able to look at many instances of inputs and outputs of a system and learn on their own, without needing to be explicitly programmed, how to model the system. This makes them well-suited for learning to mimic all the small intricacies which are supposed to make the PUF secure.

2. *Why is it important to split the data into separate training, validation and testing sets?*
A model should always be tested on an unseen part of the dataset to get an understanding of its performance on unseen data in the real world. This explains why the test set is separated off from the rest of the data. However, while a programmer is training their model, they would like to be able to try something, make some changes, and maybe try something else to see what helps and what doesn't. For these mini-tests, the validation set is used. The training set is used by the model many times to learn the features of the data.

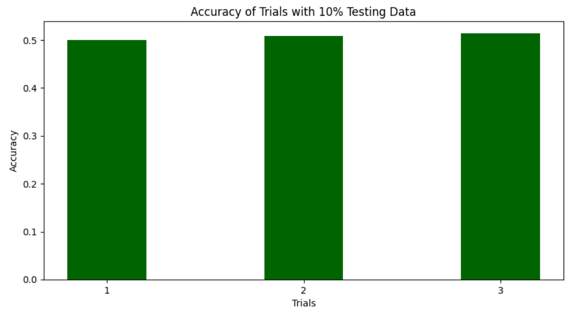
3. *What features/data are used by the neural network to model the PUF?*
The neural network learns the features of the inherent minor differences in chips which result from the manufacturing process. This is often called the chip's fingerprint.

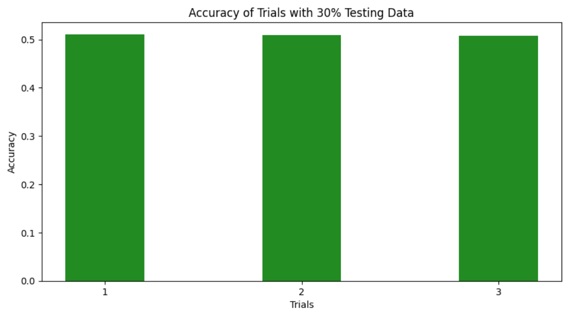
B. Analysis of Results

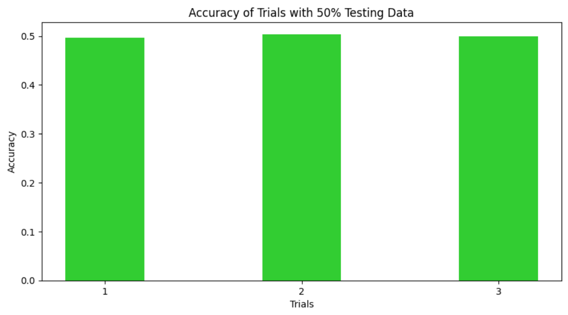
Table I shows the results of the neural network provided in modelPUF.py across several different trials and with several different train/test split ratios. As can be seen, the accuracy when modeling the PUF is a very poor roughly 50% average with very little variance regardless of the train/test split. These results are not what would be expected. We will go over this in Section IV.

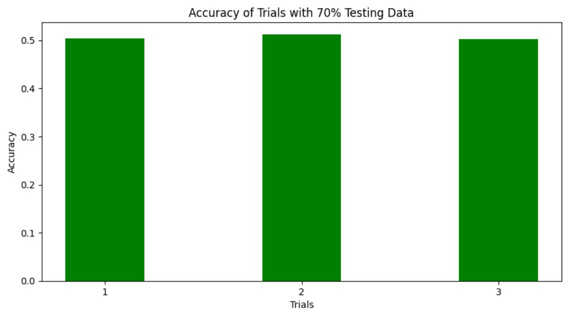
TABLE I
modelPUF Neural Network Results

Test %	**10**	**30**	**50**	**70**
Trial 1	0.5637444	0.5380426	0.5798599	0.5402666
Trial 2	0.5409333	0.5341571	0.5796599	0.5846999
Trial 3	0.5056444	0.5392857	0.5331000	0.5554666
Average	0.5367740	0.5371618	0.5642066	0.5601443









IV. Discussion

As mentioned in Section III, we did not see the results that were expected. In the model for PUFs we got roughly 50% accuracy no matter what, we are aware that many students in class had similar issues. Some reported to gain better accuracy values by modifying the objects in modelPUF.py but we opted to leave it as is and base our testing off the provided file. What we would expect to see is the accuracy's growing larger as the test set gets smaller (training set gets larger) since the model is able to train itself on more data.

V. Conclusion

In this lab, TensorFlow and SciKit-Learn machine learning tools were used to model the behavior of a PUF and break its security. Several train/test split ratios and machine learning classifiers, in addition to adding XOR gates to the PUF outputs, were experimented with. Graphs and tables were created to compare results. However, these results were not what would be expected from proper functionality.