A Machine Learning Approach for Cryptanalysis

Kowsic Jayachandiran
Department of Computer Science
Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, NY 14586
kj4401@cs.rit.edu

***Abstract*—The paper introduces a novel approach of using neural networks to perform cryptanalysis on a lightweight cipher like the Simon cipher. The neural network takes in plaintexts and their corresponding ciphertexts to predict the key that was used to encrypt the plaintext. The neural network is tested with various rounds of the cipher system to see how it fares, and is also tested with various configurations of the neural network to determine which configuration could give us the highest accuracy at determining the key correctly.**

***Index Terms*—Cryptanalysis; Neural network; Machine Learning; Simon cipher**

I. Introduction

Cryptanalysis has been an area of great research interest in the past decade owing to advancements in machine learning algorithms, particularly in neural networks. The process of discovering the plaintext from a ciphertext without knowing any information about the system or the key that was used to encrypt the plaintext is called cryptanalysis. Any mode of communication is secure only as long as the cryptographic system that encrypts the messages between the sender and the receiver is strong. Once a third party listening in on the communication channel is able to decipher the encrypted texts, the cipher system is said to have flaws and to be broken. All ciphers are vulnerable to brute-force attacks in that the attackers try to break the cipher system by exploring its key space. Though this takes a lot of time and computational power, it is possible to break the system. The project discussed in this paper deals with a method of breaking a cryptographic system with less work than brute-force search.

The project aims at finding the key of a Simon cipher [1] that was used to encrypt a set of plaintexts. It tries to achieve this by using a machine learning technique called Neural Networks that mimics the working of a neuron in the brain. Just like how the neurons in the brain process information in parallel and arrive at a decision/action, the perceptrons (neuron-equivalent in neural network) learn from the errors made and pass on the information to succeeding perceptrons, thus creating a multi-layered neural network. The Simon cipher, a lightweight block cipher, has a block size of 32 bits and key size of up to 64 bits. Each bit of the input plaintext will be assigned to the perceptrons in the input layer. The hidden layers perform the task of identifying the key.

The method used in the paper is quite different from other papers in this research domain in that it uses neural network for cryptanalysis and also that the network tries to predict the key of a cipher given a dataset of plaintext-ciphertext pairs with quite a large number of keys, while the other papers (mentioned in the related work section) try to identify if neural network can predict the key given the dataset of plaintext-ciphertext pairs for only one key. The method discussed in the paper discusses how it uses two versions of the Simon cipher — one with two rounds of the cipher and another with just one round to determine how well neural networks perform.

The remainder of this paper is structured as follows: various works that have been done in the area of cryptanalysis using machine learning techniques are mentioned in Section II; Section III discusses the generation of dataset and how the neural network was designed to read in the plaintext-ciphertext pair and obtain the predicted key; the results and its evaluations are discussed in Section IV; finally, we conclude and describe the future work in Section V.

II. Related Work

In [2] Chandra, the authors used neural networks to classify the ciphertext based on the algorithm that was used to encrypt it. They had used Cascade Correlation Neural Network and Back Propagation Network to identify the cipher systems. For training they had used ciphertexts obtained from Enhanced RC6, a block cipher, and from SEAL, a stream cipher. They had used different types of datasets with same keys, different keys, same sets of plaintexts, different sets of plaintexts etc. and concluded that cascade correlation worked better than the back propagation method.

Alani MM [3] had come up with an idea to break Data Encryption Standard (DES) cipher using neural network. The author had used the known-plaintext attack to arrive at the plaintext. The algorithm used by the author does not seem to attempt to find the key, but rather tries to directly find the plaintext. Though this approach is not considered to be a cryptographic attack, the work of the author is commendable as the author had designed a neural network for the process of identifying the plaintext using the same plaintext and ciphertext of the same key.

In the paper written by Albassal and Wahdan [4], the authors have described how they were able to use neural networks to break a hypothetical Feistel cipher, called HypCipher. The round function for the HypCipher had been chosen from the Advanced Encryption Standard (AES). The back propagation technique has been used showing success with 2 and 3 rounds of the cipher. An additional hidden layer had been added for 4 rounds. The model was successful in that it used a simple neural network with a simple activation function like the sigmoid function. The authors have proposed to use a distributed system to attack ciphers with more rounds.

III. Methodology

A. Simon Cipher

The project was designed with an intention to observe how well neural networks can work in the area of cryptanalysis. Since this is a first step in this field, the decision was made to use a lightweight cipher that would be easier for the network to analyze and come up with an approximation function for the inputs given. One such lightweight cipher is the Simon cipher. The Simon cipher has an input block size of length 32 bits with a key size of 64 bits in length. Lightweight ciphers generally find their use in devices that have a very restricted hardware resources like in Internet of Things [1]. For the sake of simplicity and to test the neural network's effectiveness in coming up with the approximation function, the Simon cipher had been modified to encrypt the plaintexts with first a single round of the Feistel network and then with two rounds, and the same was followed for decryption as well. The source code for Simon cipher was available online on GitHub and was written by Calvin McCoy [5]. The code was modified to work with one and two rounds of the Simon cipher.

The architecture of a single round of Simon cipher is shown in Figure 1. The 32 bits long plaintext is divided into two blocks of 16 bits in length, which is shown in the figure as x_i and x_{i+1}. The former half is made to go through various left circular shift units, with S^1, S^2, and S^8 representing left circular shift by one, two, and eight bits respectively. A bitwise *AND* is performed on the results of S^1 and S^8 followed by a bitwise *XOR* with the second half of the input bits, x_{i+1}. The result of the previous operation is again XOR-ed with the result from S^2. The Simon cipher, generally, creates a list of key words k_0, k_1,..., k_T from the key that is provided to it, where T is the number of rounds. Since in our case, we are only working with one round and two rounds of the cipher, the key schedules generate only one and two key words respectively. The corresponding key word from the key schedule is then XOR-ed with the result of the last performed operation. In the final step of the round, the result from the last step is swapped with the input block x_{i+1} and then passed on to the next round of the Simon cipher, where the same steps are repeated again.

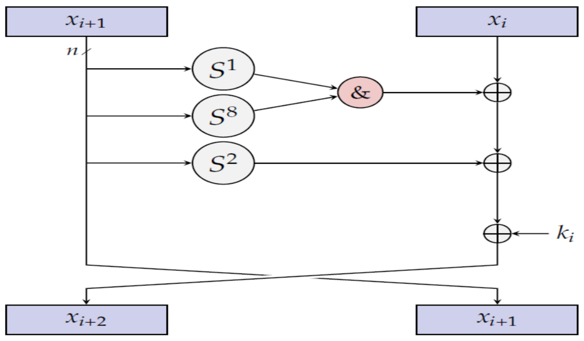
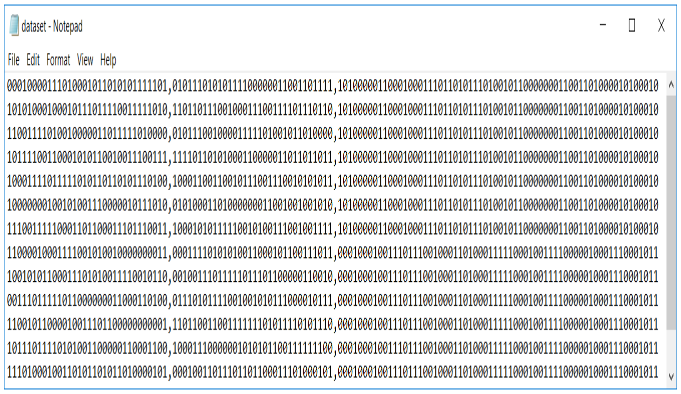


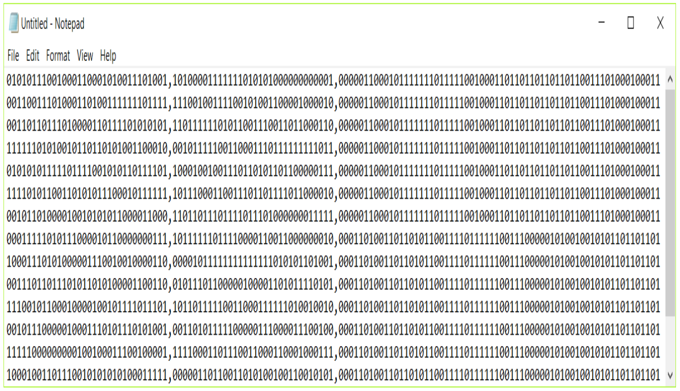
Fig. 1. One round of Simon cipher [1]

B. Data Preparation



```
dataset - Notepad
File Edit Format View Help
00010000111010001011010101111101,01011101010111100000011001101111,1010000011000100011101101011101001011000000011001101000010100010
10101000100010111011110011111010,11011011100100011100111101110110,1010000011000100011101101011101001011000000011001101000010100010
11001111010010000011011111010000,01011100100001111101001011010000,1010000011000100011101101011101001011000000011001101000010100010
10111100110001010110010011100111,11110110101000110000011011011011,1010000011000100011101101011101001011000000011001101000010100010
10001111011111010110110101110100,10001100110010111001110010101011,1010000011000100011101101011101001011000000011001101000010100010
10000000100101001110000010111010,01010001101000000011001001001010,1010000011000100011101101011101001011000000011001101000010100010
11100111110001101100011101110011,10001010111110010100111001001111,1010000011000100011101101011101001011000000011001101000010100010
11000010001111001010010000000011,00011110101010011000101100111011,0001000100111011100100011010001111100010011110000010001110001011
10010101100011101010011110010110,00100111011111011101100000110010,0001000100111011100100011010001111100010011110000010001110001011
00111011111011000000011000110100,01110101111001001010111000010111,0001000100111011100100011010001111100010011110000010001110001011
11001011000010011101100000000001,11011001100111111101011110101110,0001000100111011100100011010001111100010011110000010001110001011
10111011110101001100000110001100,10001110000001010101100111111100,0001000100111011100100011010001111100010011110000010001110001011
11101000100110101101011010000101,00010011011101101100011101000101,0001000100111011100100011010001111100010011110000010001110001011
```

Fig. 2. Dataset sample for one round



```
Untitled - Notepad
File Edit Format View Help
01010111001000110001010011101001,10100001111110101010000000000001,0000011000101111111011111001000110110110110110110011101000100011
00110011101000110100111111101111,11100100111001010011000010000010,0000011000101111111011111001000110110110110110110011101000100011
00110110111010000110111101010101,11011111101011001110011011000110,0000011000101111111011111001000110110110110110110011101000100011
11111010100101101101010011000010,00101111001100011101111111111011,0000011000101111111011111001000110110110110110110011101000100011
01010101111101111001010110111101,10001001001110110101101100000111,0000011000101111111011111001000110110110110110110011101000100011
11110101100110101011100010111111,10111000110011101101111011000010,0000011000101111111011111001000110110110110110110011101000100011
00101101000010010101011000011000,11011011101111011101000000011111,0000011000101111111011111001000110110110110110110011101000100011
00011111010111000010110000000111,10111111011100001100110000000010,0001101001101101011001111011111100111000001010010010101101101101
10001110101000001110010010000110,00001011111111111111010101101001,0001101001101101011001111011111100111000001010010010101101101101
00111011011101011010100001100110,01011101100000100001101011110101,0001101001101101011001111011111100111000001010010010101101101101
11100101100010000100101111011101,10110111100110001111110100100010,0001101001101101011001111011111100111000001010010010101101101101
00101110000010001110101110101001,00110101111100000111000011100100,0001101001101101011001111011111100111000001010010010101101101101
11111000000000100100011100100001,11110001101110011000110001000111,0001101001101101011001111011111100111000001010010010101101101101
10001001101110010101010100011111,00000110110011010100100110010101,0001101001101101011001111011111100111000001010010010101101101101
```

Fig. 3. Dataset sample for two rounds

The training dataset was generated using a random binary digits generator that generated random hexadecimal numbers and were converted to binary representation later. As can be seen in Figure 2 and Figure 3, the datasets were structured such that the first 32 columns consist of the

32-bit plaintexts represented in binary bits; the succeeding 32 columns represented the corresponding 32-bit ciphertexts, also represented in binary bits; the last 64 columns representing the 64-bit binary representation of keys that were used to encrypt the corresponding plaintexts.

Figure 2 shows a sample of dataset records that were generated using only one round of Simon cipher, while Figure 3 shows a sample of records that were generated using two rounds of Simon cipher. Around 5 million records were generated with 1000 keys, where each key encrypted 5000 plaintexts. Each plaintext generated was ensured that it was unique so that the neural network did not have trouble in calculating the loss and adjusting the weights after having witnessed duplicate entries.

C. Keras

Keras is a high-level library written in Python for the purpose of working with neural networks [6]. For this project, Keras has been configured to work on top of Google's TensorFlow [7]. Keras provides options to run the deep learning algorithms on either the Central Processing Unit or the Graphical Processing Unit; the choice, ultimately, is decided by the number of records to be processed in the dataset and the amount of calculations required. In this project's case, there are five million records which need to be handled by the neural network. So, the project required the installation of GPU version of Keras to help with faster processing of records. This version was particularly helpful when the epochs were increased while training the neural network. Keras also required the use of a binary data format called HDF5 that helps save the model after training to ease with the calculations while predicting [8].

Following are the specifications of the machine on which the project was designed and tested:

Operating System: 64-bit Windows 10 Home
Processor: Intel Core i7-6700HQ
Number of cores: 4
Clock Speed: 2.60 GHz
System Memory: 16 GB
Graphics Chip Type: NVIDIA GeForce GTX 960M
Graphics Memory: 12.17 GB

D. Neural Network Design and Training

An overview of the project's neural network architecture is shown in Figure 4. Each neuron in the input layer corresponds to each bit of the plaintext and ciphertext pairs. There would, therefore, be 64 neurons in the input layer. The hidden layers are where the actual computation takes place. The neurons in the hidden layer compute the weights and bias after each iteration. These weights are then assigned to the neurons in the preceding layer depending on the loss error determined at the end of each iteration. The output layer has 64 neurons in total each predicting whether the key bit is a 0 or a 1 based on the computations done in the hidden layers.

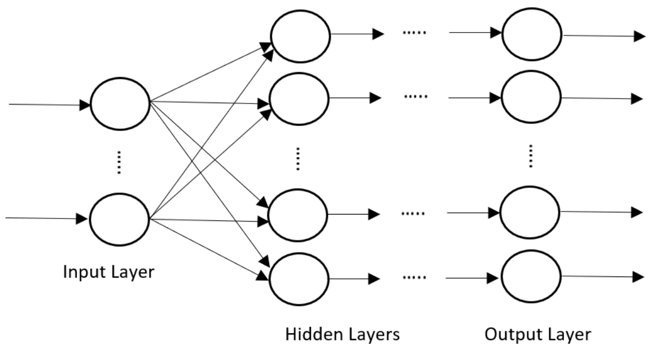


Fig. 4. Neural network architecture

Before proceeding with the neural network design, it is imperative that a few of the terms used in the neural network be defined. A neural network reads the records in a dataset in groups. The number of records in a group is specified by the *batch size* while configuring the neural network. In the case of the neural network designed for the project, the batch size is chosen such that the neural network reads all plaintext-ciphertext pairs of one key at a time, which is 5000 pairs per key. When the neural network completes reading the number of records specified by the batch size both in the forward and backward direction, it is said to have completed one iteration. At the end of each iteration, approximate values for weights and bias are assigned to the neurons. When the iterations are performed through the entire dataset, it is said to have completed one epoch. At the end of one epoch, approximate values for each neuron in the network would have been assigned for each record in the dataset.

There are 1000 keys in the dataset, which makes the total number of iterations per epoch to be 1000 – at the end of each iteration, 5000 records would have been read (specified by the batch size) and hence, 1000 iterations to read 5000000 records. The approximate values assigned to the neurons will only get better with each epoch as the neural network learns at the end of each epoch. The number of epochs is determined on a trial-and-error basis and depends entirely on the type of dataset. In the case of the project, the epoch at which a good accuracy is obtained will tend to be higher as the dataset is huge.

$$\text{Number of iterations per epoch} = \frac{\text{Number of training records}}{\text{Number of batch files}}$$

Number of neural networks: The design of the neural network took quite a lot of time as it involved experimenting with various parameters of the network provided by Keras.

The first approach that was tried was to use multiple neural networks with each neural predicting one bit of the key, resulting in 64 different neural networks. One problem that was faced was the fact that the dataset consists of all 0s and 1s and having multiple networks with the same set of inputs but predicting different outputs did not quite work well. So it was decided to have one single network which handles all kinds of inputs and predicts the keys.

Layers and numbers of neurons: The neural network model is defined as a sequence of layers. The input layer, as mentioned before, has 64 neurons. This step of adding the input layers is mandatory for designing any neural network, as without it the neural network would not know what it is going to be working with. After repeated experimenting, the hidden layers are added to the model one at a time. The number of neurons in each layer is independent of the previous layers, which also needs to be experimented with before arriving at a conclusion as to how many neurons should be placed in a layer. It first started out with just one hidden layer, and when the accuracy was not found to be good, more layers were added. The accuracy is determined by calculating the number of predicted key bits that match with the original key bits. Initially only around 100 neurons were added in each hidden layer, but the accuracy of the model was around 50% which showed that the model was very poor in deciding the key bits. So, the neurons in the hidden layers were gradually increased in each layer to reach 1024 in one of the hidden layers. The number of hidden layers were also increased to smoothen the increase in the number of neurons in successive layers. This helped boost the accuracy and hence the next step in plan was to work on other configurations of the network.

A neuron in one layer is connected to a neuron of the succeeding layer using one of many methods available in Keras. To get the maximum accuracy possible, each neuron in one layer is connected to every neuron in the following layer. Keras provides a method called *Dense* to connect every neuron in the first layer to every neuron in the succeeding layer to improve the assignment of weights and to improve the approximation function.

Weights and bias: The initial weights of the neurons are initialized using the *uniform* distribution method and are updated after each iteration. The activation function determines the output of each neuron in the network. It is a function of the weight and bias assigned to the neuron along with the input that it receives. The output of the activation function is passed on to the next neuron in the network. This step again took a significant amount of time as Keras provided a lot of activation functions that could be used. The network designed until now was tested with various activation functions like softmax, sigmoid, and tanh functions but these functions only brought down the calculating power of the network. It was mentioned on Keras' online documentation [6] that most of the projects on neural networks start out by trying the Linear Rectifier activation function defined as *relu* in Keras. Unlike other activation functions, the *relu* function increased the accuracy by a little factor and hence, it was decided to proceed with this activation function.

$$\text{Activation Function} = \sum(\text{weight} * \text{input}) + \text{bias}$$

Model compilation: TensorFlow then compiles the model using the parameters provided, such as the loss function to be used and the optimizer. The loss function is chosen such that the loss calculated after each iteration is always low. The optimizer assigns the weights to the neurons in the network after each iteration. Keras provides quite a lot of loss functions that could be used like mean squared error, mean absolute error, categorical crossentropy, etc. Various optimizers like Stochastic Gradient Descent (SGD), RMSProp, and Adamax optimizers were tried with the network but the accuracies were lower than 51% for these optimizers.

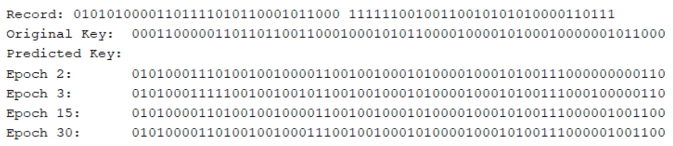
The neural network, after experimenting with all the different possible combinations of optimizer and loss functions, was compiled with the *binary_crossentropy* loss function and *adam* optimizer. The *binary_crossentropy* was chosen as the network required outputs in either of the two classes – 0 or 1. Among all the efficient optimizers available on Keras, *adam* optimizer is used as it is a simple optimizer and is found to be very efficient in assigning weights[9]. It was also able to improve the accuracy of the model by a lot compared to other optimizers.

Model fitting: The next step to follow after compiling is to fit the model. The fit method in Keras takes in various parameters, with the most important of them being the input and output columns from the dataset, epochs, and the batch size of records that need to be iterated. The values for epoch and batch size are to be experimented with until a good accuracy is obtained. Much of the time was spent on trying to figure out the right combination of these factors that affect the neural network. The right number of epoch to be chosen is decided based on when the accuracy of the model does not increase any further. Since the dataset was also huge, it took a considerable time to fit the model to analyze if the chosen configurations were indeed correct or had to be changed to get a better accuracy. For epochs greater than 20, it took close to 4 hours to train the model.

IV. Results

The trained neural network model is then used to predict the key of the plaintext-ciphertext pairs by using the *predict* method provided by Keras. The prediction is based on the approximation functions calculated by the neurons in the hidden layer.

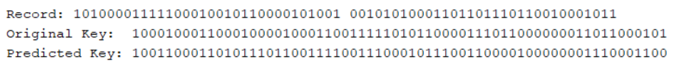
A. *Simon cipher with two rounds*



```
Record: 0101010000110111101011000101100 0 11111100100110010101010000110111
Original Key:  0001100000110110110011000100010101100001000010100010000001011000
Predicted Key:
Epoch 2:       0101000111010010010000110010010001010000100010100111000000000110
Epoch 3:       0101000111110010010010110010010001010000100010100111000100000110
Epoch 15:      0101000011010010010000110010010001010000100010100111000001001100
Epoch 30:      0101000011010010010001110010010001010000100010100111000001001100
```

Fig. 5. Predicted key for sample training record

The results in the Figure 5 shows how the neural network is predicting the bits of the key for a Simon cipher with two rounds. At epoch two, since the neural network had just begun learning the dataset, it is having trouble predicting the bits correctly. When the epoch is increased to three and then consequently to 15 and 30, the key bits vary a little but there is not much improvement. It can be seen clearly that the network is just guessing the bits to be either a 0 or a 1. This result goes on to show that the neural network is not able to come up with an approximation function for the dataset provided, and hence, the poor results. This result could be attributed to the fact that a round in the cipher consists of complex functions like XOR, circular shifts and bitwise ANDs, and the neural network is having trouble figuring out the approximation given a repetition of the same functions in the second round.

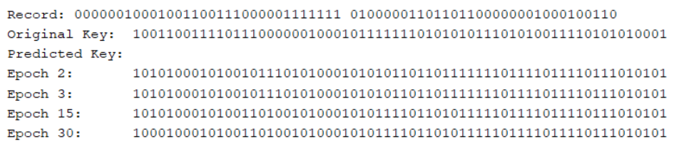


```
Record: 10100001111100010010110000101001 00101010001101101110110010001011
Original Key:  1000100011000100001000110011110101100001110110000000110110001 01
Predicted Key: 1001100011010111011001111001110001011100110000100000001110001100
```

Fig. 6. Predicted key for sample test record

The accuracy of the network is measured by identifying the number of bits predicted by the neural network that are the same as the key bits in the original key. The accuracy rate for this method has its range between 50% and 60%; in the case of the example record in Figure 6, the accuracy was close to 61%. The result shown in the image is obtained at the end of 30 epochs. Each bit in the predicted key is the output from each node of the output layer of neural network. This bad accuracy led to the decision of toning down the number of rounds in Simon cipher which is explained in the following subsection.

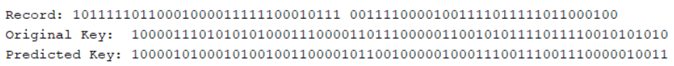
B. *Simon cipher with one round*



```
Record: 00000010001001100111000001111111 01000001101101100000000100010011 0
Original Key:  1001100111101110000000100010111111101010101110101001111010101000 1
Predicted Key:
Epoch 2:       1010100010100101110101000101010110110111111101110111101110101 01
Epoch 3:       1010100010100101110101000101010110110111111101110111101110101 01
Epoch 15:      1010100010100110100101000101011110110101111101110111101110101 01
Epoch 30:      1000100010100110100101000101011110110101111101110111101110101 01
```

Fig. 7. Predicted key for sample training record

As explained in the previous section, the results in Figure 7 shows improvements over higher epochs. The results obtained in the final epoch of Figure 7 show an accuracy of 71%. As discussed earlier, with increase in epochs and iterations, the model has gotten to a point where it has assigned the weights and bias properly to the neurons and hence the approximation function is able to find the outputs correctly to an extent. It was found that the accuracy did not improve any further with increase in epochs after 30. This trained model was then used on the test dataset to predict the key.



```
Record: 10111111011000100001111110001011 1 00111100001001111011111011000100
Original Key:  1000011101010101000111000011011100000110010101111011110010101010
Predicted Key: 1000010100010100100110000101100100000100011100111001110000010011
```

Fig. 8. Predicted key for sample test record

As we could see in the Figure 8, the neural network is able to predict the bits well and we can observe that the accuracy has now jumped to 70% which is a great improvement over the model trained with two rounds. This means that there is a 70% match between the predicted key bits and the original key bits that were used to encrypt the plaintext. Further experiments with other test datasets showed the same results with accuracy ranging between 63% and 74%.

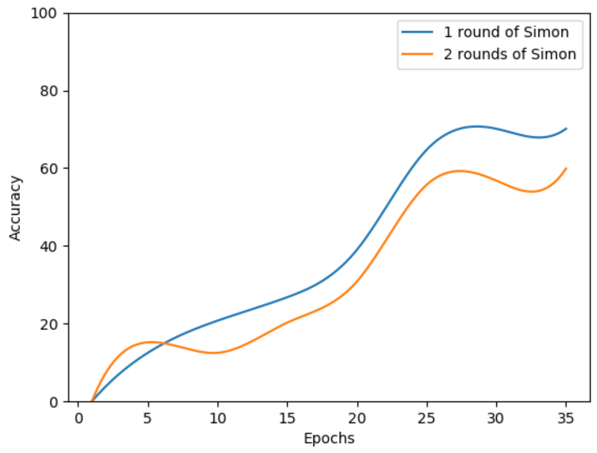


Fig. 9. Plot of Accuracy vs. Epochs

Figure 9 shows the plot of accuracy with increase in epochs. The accuracies were calculated as the maximum of all accuracies obtained after every five epochs of training the dataset. For two rounds of Simon cipher, the accuracy initially seems to increase and then decrease before epoch ten as the learning rate of the neural network is high in the beginning and then keeps decreasing. This is also sometimes referred to as a common problem in deep learning neural networks, called the Exploding Gradient Problem [10], which is the process of increasing learning rate in earlier layers of the neural network. It has been found to decrease later when the epoch increases. After epoch 30, the accuracy seems to have stabilized at approximately 60% and no further increase can be seen. For one round of Simon cipher, the learning rate looks quite smooth and the accuracy seems to increase with increase in epochs until epoch 30. After epoch 30, the accuracy becomes stagnant at approximately 70% and an increase cannot be observed.

The project was initially designed to work with Simon cipher of rounds two and four, but the accuracy of the model was only around 50% to 60% which is very much equivalent to guessing if a key bit is 0 or 1. To check if the neural network could be able to identify the approximation function, the Simon cipher was configured to work with just one round of the Feistel network and the accuracy jumped to the range of 63% to 74%.

V. Conclusion and Future Work

From the results obtained, it is evident that neural networks is a possible solution to many questions we have in the field of cryptanalysis. As such, there has been only a few methods tried in the domain of cryptanalysis since the key space of any complex cipher system is large. The project also demonstrates that the neural network works well with one round of the Simon cipher but does not perform well when it comes to working with higher number of rounds. This is only the first step of many more possibilities that could be explored with neural networks; with more knowledge acquired in the future about neural networks, the number of rounds could be increased and the network would be able to come up with an approximation function provided it has all the required resources for such heavy computations.

An extension of this project in the future could be a design based on fuzzy classifiers that could be used along with the neural networks to yield probabilities for 0 and 1 rather than a hard decision that a neural network does. The fuzzy classifier in itself could possibly do a brute-force search for the key in decreasing order of key probability, resulting in less work than an exhaustive key search.

Acknowledgment

I would like to thank Prof. Alan Kaminsky for guiding me with my project through the semester, and providing invaluable guidance and suggestions. I would also like to thank Prof. Reynold Bailey for helping me improve my writing and presentation skills.

References

[1] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, "The simon and speck families of lightweight block ciphers," Cryptology ePrint Archive, Report 2013/404, 2013, https://eprint.iacr.org/2013/404.

[2] B. Chandra and P. P. Varghese, "Applications of cascade correlation neural networks for cipher system identification," *International Journal of Computer, Electrical, Automation, Control and Information Engineering*, vol. 1, no. 2, pp. 369 – 372, 2007. [Online]. Available: http://waset.org/Publications?p=2

[3] M. M. Alani, "Neuro-cryptanalysis of des," in *World Congress on Internet Security (WorldCIS-2012)*, June 2012, pp. 23–27.

[4] A. M. B. Albassal and A. M. A. Wahdan, "Neural network based cryptanalysis of a feistel type block cipher," in *Electrical, Electronic and Computer Engineering, 2004. ICEEC '04. 2004 International Conference on*, Sept 2004, pp. 231–237.

[5] C. McCoy, "Simon speck ciphers," https://github.com/inmcm/Simon_Speck_Ciphers, 2016.

[6] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[7] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[8] A. Collette, "Hdf5 for python," http://h5py.alfven.org, 2008.

[9] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[10] R. Pascanu, T. Mikolov, and Y. Bengio, "Understanding the exploding gradient problem," *CoRR*, vol. abs/1211.5063, 2012.