Jestr Journal of Engineering Science and Technology Review 8 (2) (2015) 152 - 156
Special Issue on Synchronization and Control of Chaos: Theory, Methods and Applications

JOURNAL OF
**Engineering Science and Technology Review**

www.jestr.org

Research Article

# Neural Synchronization Using Genetic Algorithm for Secure Key Establishment

**Daxing Wang***

*School of Mathematics and Finance, Chuzhou University, Chuzhou, Anhui, P.R. China.*

Received 1 July 2014; Revised 1 November 2014; Accepted 14 November 2014

***Abstract***

Two neural networks that are trained on their mutual output synchronize to an identical time dependant weight vector. This novel phenomenon can be used for creation of a secure cryptographic secret-key using a public channel. Neural cryptography is a way to create shared secret key. Key generation in Tree Parity Machine neural network is done by mutual learning. Neural networks here receive common inputs and exchange their outputs. Adjusting discrete weights according to a suitable learning rule then leads to full synchronization in a finite number of steps and these identical weights are the secret key needed for encryption. A faster synchronization of the neural network has been achieved by generating the optimal weights for the sender and receiver from a genetic process. Here the best fit weight vector is found using a genetic algorithm. In this paper the performance of the genetic algorithm has been analysed by varying the number of hidden and input neurons.

*Keywords:* Neural synchronization, neural cryptography, key exchange, genetic algorithm, tree parity machine.

## 1. Introduction

The problem of key exchange is one of the main concern of classical cryptography and it was extensively studied in classical cryptography. The first published key exchange protocol was based on number theory and it is known by Diffie-Hellman key exchange protocol. While it depends on the difficulties of computing discrete logarithms, it is vulnerable to a man-in-the-middle attack [1]. Moreover, it is computationally intensive. The man-in-the-middle attack is solved by authentication mechanisms. Alternatively, two networks trained on their outputs are able to achieve the same objective by means of mutual learning bringing about what is known as neural cryptography [2,3]. The advantage of neural cryptography is that the algorithm used in generating a common key is very simple and fast with the network used in training can be a simple perception or Tree Parity Machine (TPM). While the simple perception manages to synchronize fast, it is insecure [4]. Therefore, it is preferred to use the TPM due to its higher security although it needs more time to synchronize.

In neural cryptography, both the communicating networks receive an identical input vector, generate an output bit and are trained on their mutual output. This leads to synchronization by mutual learning. The synaptic weights of the two networks converge to a common identical weight vector [4]. Thus, the generated identical weight vectors are shared and used as a secret cryptographic key Synchronization of the Tree Parity Machine using genetic algorithm is described in [5]. A best fit weight vector found using a genetic algorithm is then taken as initial weights to train the network using the feed forward process. Optimal weights lead to faster convergence of the neural network than the random weights which can be further improved by increasing the synaptic depth $D$ of the neural network, but this leads to an increase in the number of iterations. However the process of synchronization also depends on the number of neurons in the hidden layer and the input layer. In this paper the performance of the genetic algorithm is further analysed by varying the number of neurons in the input and the hidden layer.

The paper is organized as follows. Generation of secret keys using optimal weights is described in Section 2. Section 3, describes the security attack followed by the results in Section 4.

## 2. Generation of Secret Keys Using Optimal Weights

The genetic algorithm [6] offers an alternative approach for the opponent, which is not based on optimizing the prediction of the internal representation, but on an evolutionary algorithm. The efficiency of the genetic attack mostly depends on the algorithm which selects the fittest neural networks. In the ideal case the Tree Parity Machine, which has the same sequence of internal representations as $A$ is never discarded. This algorithm has been used to generate the best fit input weight vector for the tree parity machine

* E-mail address: volos@physics.auth.gr
ISSN: 1791-2377 

[7]. The generation of optimal weights is achieved as follows:

To start with, the initial population weight value is taken as set of random numbers in the range [-*D*, *D*]. The fitness function is assumed to be parabolic and is defined as,

$$f(x) = \mathrm{sgn}(x) \cdot D + g(x) \tag{1}$$

Where $g(x)$ is defined as,

$$\mathrm{g}(x) = \begin{cases} 0 & x < -D \\ x^2 & -D \le x < D \\ 0 & x \ge D \end{cases} \tag{2}$$

Based on the fitness value corresponding to each weight, the most fitted set of weights W are identified using Roulette Wheel selection method. Crossover and Mutation is then performed on the elements in *W* to obtain the optimal weights using Crossover rate ($P_c$ = 0.8) and Mutation rate ($P_m$ = 0.01). This completes one cycle of GA process. This process is repeated till coincident weights are obtained in the successive iterations, which are used as initial weights in range [-*D*, *D*] for the tree parity machine shown in Fig. 1. Repetition of the following steps leads to the synchronization of the TPM (Tree Parity Machine).

Initialize weight values $\vec{w}_i$, obtained from GA process in range [-*D*, *D*]. Execute the following steps until the full synchronization achieved.

Step 1: Generate random input vector.

Step 2: Compute the values of the hidden neurons.

$$\sigma_i = \vec{w}_i \vec{x}_i \tag{3}$$

Step 3: Compute the value of the output neuron.

$$\tau = \pi\sigma_i \tag{4}$$

Step 4: Compare the values τ of both tree parity machines.

i. Outputs are different go to Step 1
ii. Outputs are same: one of the following learning rules is applied to update the weights.
Hebbian learning rule [8]:

$$w_i = w_i + \sigma_i x_i \theta(\sigma_i \tau)\theta(\tau^A \tau^B) \tag{5}$$

Random walk:

$$w_i = w_i + x_i \theta(\sigma_i \tau)\theta(\tau^A \tau^B) \tag{6}$$

In this case, only the hidden unit $\sigma_i$ which is identical to $\tau$ changes its weights. In case the updated weights are not in the range [-*D*, *D*] they are redefined as follows:

$$w_i = \begin{cases} -\mathrm{sgn}(w_i)D, & |w_i| > D \\ w_i, & otherwise \end{cases} \tag{7}$$

When synchronization is achieved, the weights $\vec{w}_i$ of both tree parity machines are same and they become the secret keys for communication between A and B. In this work we have shown that the process of synchronization can be further improved by varying the number of hidden neurons.

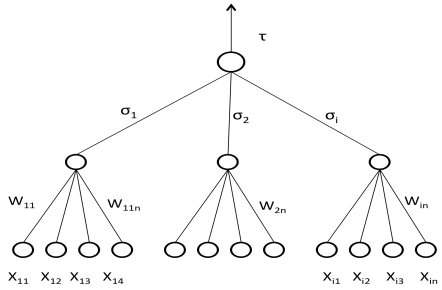



**Fig. 1.** Tree Parity Machine.

## 3. Security Attacks

Synchronization of Tree Parity Machines by mutual learning only works if they receive a common sequence of input vectors. This effect can be used to implement an authentication mechanism for the neural key-exchange protocol. For that purpose each partner uses a separate, but identical pseudo-random number generator. As these devices are initialized with a secret seed state shared by *A* and *B*, they produce exactly the same sequence of bits, which is then used to generate the input vectors $\vec{x}_i$ needed during the synchronization process. By doing so *A* and *B* can synchronize their neural networks without transmitting input values over the public channel. Of course, an opponent *E* does not know the secret seed state. Therefore *E* is unable to synchronize due to the lack of information about the input vectors. Even an active man-in-the-middle attack does not help in this situation, although it is always successful for public inputs. Consequently, reaching full synchronization proves that both participants know the secret seed state. Thus *A* and *B* can authenticate each other by performing this variant of the neural key exchange. The use of genetic algorithms can fasten the process of synchroniza-tion of genuine parties, so that many attacks, which are generally successful due to lengthy synchronization process, can be avoided. This makes the brute force attack very difficult. So in this paper we have discussed the Regular flipping Attack (RFA) and Majority Flipping Attack (MFA) strategy [9].

In RFA the attacker imitates one of the parties, but in the step in which his output disagrees with the imitated party's output, he negates ("flips") the sign of one of his hidden units. The unit, which is most likely to be wrong, is the one with the minimal absolute value of the local field therefore that is the unit which is flipped. In the genetic algorithm RFA attack can be further weakened by increasing *D*. The strategy of MFA [9,10], is to use the attackers as a group rather than as individuals who cooperate and work. All weights are updated according to the majority's result. This "team –work" approach improves the attackers performance. We define the attacker to be successful if he is able to guess 98% of the weights.

## 4. Results and Analysis

1) Generation of keys:

As described in generation of keys using optimal weights section keys are the synchronized set of weights obtained from the TPM network. The number of learning steps which required to achieve this synchronization depends on the parameters *N*, *K*, *D*. Analysis has been carried out to investigate the number of iterations needed for synchronization by varying the range of weights (*D*) and by varying the number of neurons in the hidden layer (*K*) as well as the input layer (*N*).

2) Key generation with varying weight ranges and varying hidden neurons:

The above algorithm was analysed for varying weight ranges noted by *D*, and by varying number of hidden neurons *K*. The number of input neurons is kept fixed as $N = 100$. The average number of learning steps required to achieve weight synchronization with varying *K* and *D* by the sender and receiver is illustrated in Fig. 2. We could see that the number of learning steps is directly proportional to the weight range *D*, which would decelerate the process of synchronization and increase the probability of attack. However, along with increase in *D* if the number of hidden neurons (*K*) is also increased then it helps to accelerate the synchronization process as *D* is increased. Finally this brings down the success probability of an attack. The success probability of RFA and MFA with 100 attackers is shown in Fig. 3, where it is seen that both in RFA and MFA the success of attack reduces as the number of hidden neuron increases.

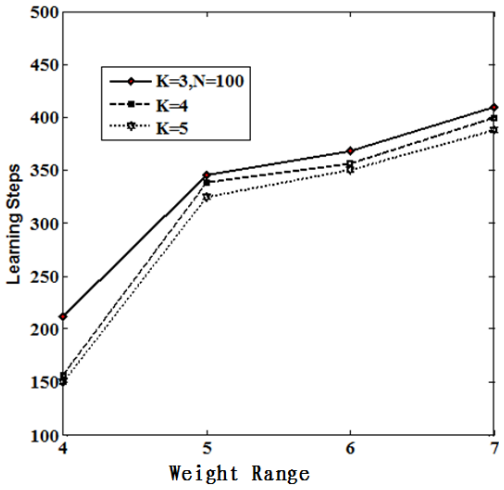
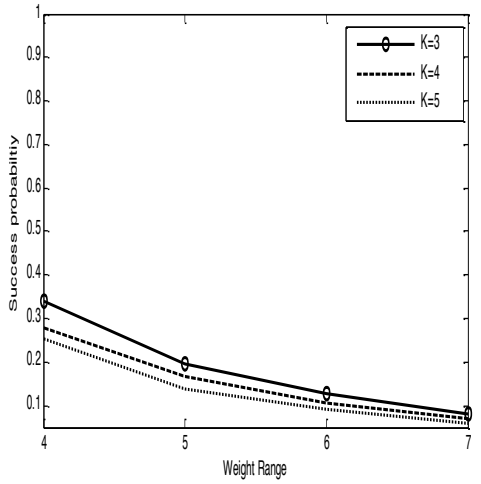


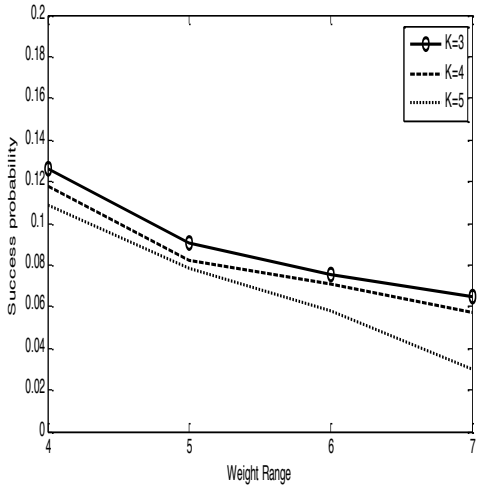

**Fig. 2.** The average synchronization steps with different *K* and *D*.

3) Key generation with varying weight ranges and varying Input neurons:

Tree Parity Machines, which are used by partners and attackers in neural cryptography, are multilayer feed-forward networks. Their general structure is shown in Fig. 1. As in other neural networks the weighted sum over the current input values is used to determine the output of the hidden units. The tree parity machine was also analysed by varying the weight range *D* and the input neurons represented by *N*. Here the number of hidden neurons *K* is kept fixed as 4. The average learning steps for sender and receiver to achieve weight synchronization with varying *N* and *D* is illustrated in Fig. 4. It is seen that the number of learning steps considerably increase as *N* increases, due to which the probability of success of MFA and RFA increases as seen in Figs. 5(a) & (b).




(a)




(b)

**Fig. 3.** (a) Success rate of MFA for $M = 10$ attackers and (b) Success probability of RFA for $M = 100$ attackers.

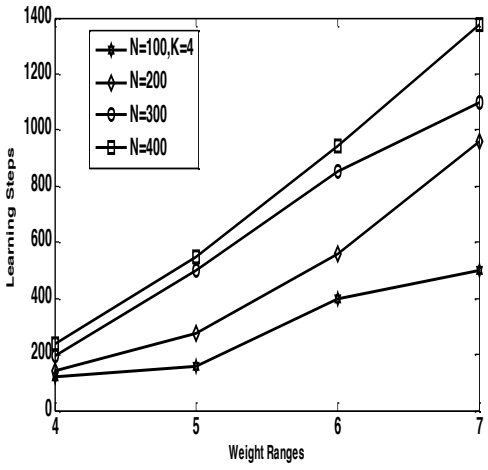



**Fig. 4.** The average synchronization steps with different *N* and *D*.

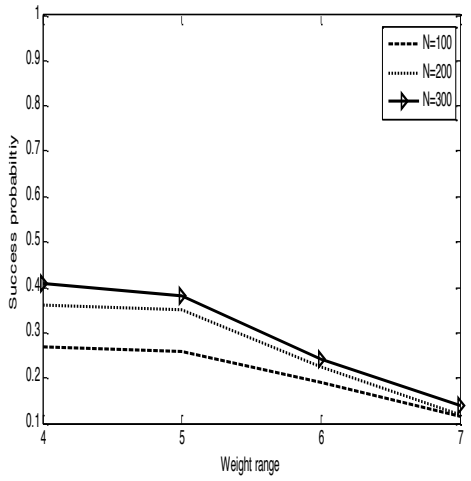



Figure 5a: Success probability of MFA for M=100 attackers

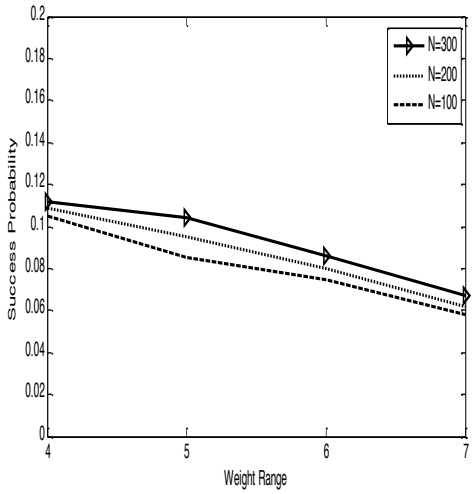



**Fig. 5.** (a) Success probability of MFA for $M$ = 100 attackers and (b) Success probability of RFA for $M$ = 100 attackers.

From the above results it shows that increasing the hidden neurons brings faster convergence and more security.

4) Security analysis:

The above results of experiment show that majority flipping attack failed when the keys are generated using a genetic algorithm. From Figs. 2 & 4 it can be seen that the performance of genetic algorithm can be further enhanced to get a secure key by increasing either the number of hidden neurons or the input neurons. But if we also need to obtain faster convergence then increasing the number of hidden neurons gives better results. The success probability of RFA and MFA with 10 or 100 attackers is shown in Fig. 3, where it is seen that both in RFA and MFA the success of attack reduces as the number of hidden neuron increases.

Figure 5 shows that increasing the hidden neurons bring faster convergence and more security. Figure 6 shows the success probability of MFA with and without GA (that is using random weights) [10]. For $K$ = 3, $N$ = 100 and $M$ = 100 attackers. A considerable decrease in the probabilities is indicated here.

5) Key Stream Generation:

In cryptography, a key stream is a series of flow combined with the encryption and decryption of data flowed with the plaintext data. Key stream generator is designed by a short random key (also known as the actual key or seed key) to generate a long key stream. Once the key stream is generated an XOR operation is performed with the keys and the encoded plain text to obtain the encrypted text. Stream cipher is a symmetric key encryption where each bit of data is encrypted with each bit of key. The Crypto key used for encryption is changed dynamically so that the cipher text produced is secure and difficult to crack.

The above algorithm can be implemented with the feedback mechanism [11] to generate the key stream of desired length. Here the synchronized weights of the previous iteration would become the input vector which is defined as follows:

$$x_i = \begin{cases} -1, & W < 0 \\ 1, & W > 0 \end{cases} \tag{8}$$

New set of optimal weights $W$ are generated using genetic algorithm. The TPM is synchronized for this new set of inputs and weights and a fresh set of synchronized weights is obtained which is appended to the previous set of synchronized weights to obtain a key of desired length. Thus, the process will terminate after generating the final key stream.

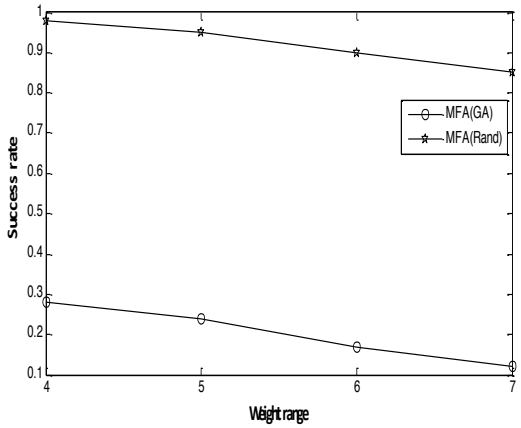



**Fig. 6.** Success probability of MFA with and without GA.

## 5. Conclusion

Compared to the other methods the genetic attack is in a certain way different. First, it is especially successful, if $D$ is small. Second, the genetic attack is a rather complicated algorithm with a lot of parameters. it is necessary to compare all known attack methods in order to estimate the level of security achieved by a certain set of parameters. In this paper the synchronization of the TPM's has been shown to improve by increasing the number of hidden/input neurons. This also helps to bring down the probability of success of both RFA and MFA attack. Neural cryptography therefore promises to revolutionize secure communication by providing security based on the neural networks.

Yet sophisticated attackers which use ensembles of cooperating attackers have a good chance to synchronize. However, advanced algorithms for synchronization, which involve different types of chaotic synchronization seem to be more secure. Such models are subjects of active research, and only the future will tell whether the security of neural network cryptography can compete with number theoretical methods.

**Acknowledgments**

Supported by the Natural Science Research Project of Education Office of Anhui Province (KJ2013B185), the Natural Science Research Project of Chuzhou University (2012kj001Z).

We want to thank the members of our research group, who provided a lot of helpful advice for this paper.

## References

1. W. Stallings, Cryptography and network security, 3rd Edition, Prentice Hall (2003).
2. T. Godhavari, N.R. Alamelu, and R.Soundararajan, Cryptography using neural network, In Proc. of 2005 Annual IEEE INDICON, pp. 258-261 (2005).
3. E. Volna, M. Kotyrba, and V. Kocian, Cryptography based on neural network, ECMS, pp. 386-391 (2012).
4. E. Klein, R. Mislovaty, I. Kanter, A. Ruttor, and W. Kinzel, Synchronization of neural networks by mutual learning and its application to cryptography, In Advances in Neural Information Processing Systems, pp. 689-696 (2004).
5. I. Kanter, W. Kinzel, and E. Kanter, Secure exchange of information by synchronization of neural networks, Europhysics Letters, vol. 57(1), p.141 (2002).
6. D.E. Goldberg and J.H. Holland, Genetic algorithms and machine learning, Machine Learning, vol. 3(2), pp.95-99 (1988).
7. A.M. Allam, H.M. Abbas, M.W. El-Kharashi, Authenticated key exchange protocol using neural cryptography with secret boundaries, In Proc. of the IEEE International Joint Conference on Neural Networks (IJCNN), pp. 1-8 (2013).
8. N. Caporale and Y. Dan, Spike timing-dependent plasticity: A Hebbian learning rule, Annu. Rev. Neurosci., vol. 31, pp. 25-46 (2008).
9. A. Klimov, A. Mityagin, and A. Shamir, Analysis of neural cryptography, In Advances in Cryptology - ASIACRYPT, Springer Berlin Heidelberg. pp. 288-298 (2002).
10. L.N. Shacham, E. Klein, R. Mislovaty, I. Kanter, and W. Kinzel, Cooperating attackers in neural cryptography, Physical Review E, vol. 69(6), pp. 066137 (2004).
11. S. Chakraborty, J. Dalal, and B.Sarkar, Neural synchronization based secret key exchange over public channels: A survey, In Proc. of IEEE International Conference on Signal Propagation and Computer Technology (ICSPCT), pp. 368-375 (2014).