ICIC Express Letters ICIC International ©2019 ISSN 1881-803X
Volume **13**, Number **11**, November **2019** pp. 1057–1064

A MIRRORING NEURAL NETWORK APPROACH FOR ENCRYPTION/DECRYPTION OF DATA

Raed Abu Zitar and Hanan Hussain

College of Engineering and Information Technology
Ajman University
P.O.Box 346, Ajman, United Arab Emirates
r.abuzitar@ajman.ac.ae

Received May 2019; accepted August 2019

Abstract. *This paper presents a novel and simple encryption technique that provides very efficient and highly illusive encrypted data. A feedforward Neural Network (NN) that uses gradient descent methods is used in generating the encrypted data and the keys. The idea is about using the output of the hidden layer as the encrypted data. The resulting weights (hidden and output layers) are considered as the keys that are used for encryption/decryption at the sending and receiving ends. The input data is processed at the binary level format. Training is done for the neural network one time and the network is used for several transmissions of encrypted data until it is retrained for new keys. The training takes short time on multipurpose commercial computer to achieve very illusive encrypted data. Simulations and testing that have been achieved so far are encouraging for further future research.*

Keywords: Neural network, Encryption, Hidden weights, Output weights

1. **Introduction.** Securing a network for digitally stored or transmitted data is an ultimate need for all organizations in the world. There is everlasting need to keep enhancing encryption and protection. Many encryption techniques have been proposed in [1-12,15-18]. They use different models that implement different processing techniques as stream based or block based models. Different key based diffusion and confusion operations were proposed. Mesh-based technique and directive operators are also used in [1]. However, these methods lack the needed confusion due to its quasi-linear nature and might be exploited by hackers. The mapping can be more deceptive if it was nonlinear. On the other hand, the proposed diffusion in this paper has connectionist nonlinear and complex nature that makes it even difficult for predicting the nature of the encryption even if the keys were known. Additional information at both ends should be shared regarding the architecture of the NN and the nature of the activation functions used to complete the process. All parts of the key (i.e., weights) work collectively and in parallel in generating the encrypted/decrypted data. Neural networks with Jordan training method were used in generating keys for encryption [19]. A neural network was trained as sequential machine for encoding process. However, sequential machines have a problem of delay, especially for long input sequences. Other researchers used the neural network to generate encryption keys and they repeated the training process at the receiving end to generate the decryption keys [20]. In that sense, they did not take advantage of the compactness and the mirroring capabilities of the neural networks.

In summary, the proposed technique offers the following contributions.

1) More effective nonlinear mapping is used to overcome any embedded exploitable linearity in the mappings (provided by nonlinear activations of the neurons).

DOI: 10.24507/icicel.13.11.1057

2) More complexity is added with the parallel interconnected computational method used (provided by neural networks).

3) More confusion is generated since the processing is done with real type data (text is converted to binary and then to real values (floating point) that go operations such as multiplication and addition).

4) It uses a block-based cipher rather than stream based one and keys are re-generated for every block.

5) The keys used for encryption are different from keys used for decryption. This unsymmetrical key generation is an additional security feature.

6) When the relatively time-consuming training is finished it is followed by transmission. The created keys are used in generating the encrypted signal and recovering the original data at the receiving end. The cost of complexity is very low (It is a single loop for every neuron using single layer at both ends).

The organization of the paper is summarized as follows.

a) Section 2 describes the encryption/decryption model being proposed.

b) Section 3 discusses the weights and inputs/outputs.

c) Section 4 presents the encryption/decryption process.

d) Section 5 shows discussions.

e) Section 6 gives conclusions and future work.

The paper presents detailed description for the steps of the algorithm with examples. The different layers of encryption/decryption and neural mapping will make it extremely difficult for hackers to decode the encrypted text. The paper presents decryption of the encrypted plain text that is retrieved 100% at the receiver end.

2. Encryption/Decryption Model Details.

A two-layer feedforward Neural Network (NN) (8×8) is trained to function as a mirror. Every input vector fed to the NN gives an identical output vector. The output of the trained NN should be an equal vector to its input binary values. This is done for all possible 256 binary input values. The idea here is to train this NN on 256 possible 8-bit binary vectors using a gradient descent learning algorithm. After training is complete with almost zero accumulated error, the hidden layer of the trained NN is used to encode (encrypt) input vectors from the input plain text (that is already converted to binary). The binary values from input stream are grouped into strings of eight bits and fed to the hidden layer. During this stage of encryption the binary input vector of 8 bits coming from the original block is replaced by the 8-bit output of the hidden-layer neurons. The output of this layer should have no correlations what so ever with the input vectors. Only, the weights of the output layer are sent as a key to the receiving end. At the receiving end, those weights are used in a single layer NN to function in a similar way to the output layer in the original trained NN. The encrypted values in this case will be the inputs to this single layer and the outputs of this layer will be similar to the original input vectors that were used as inputs for the hidden layer at the transmitter end. This process of encryption/decryption is done for every group of 8 bits streaming to the hidden layer NN at the transmitting end and the decryption is done at a similar stage at the receiving end as explained earlier. Please see Figure 1, which depicts the process of encryption/decryption.

The training for the two-layer NN is done one time. After training is complete, the network is separated into two parts. The hidden layer is used in encrypting 8-bit groups from the input stream, and the output layer is used in decryption at the receiving end. The training for the two-layer NN could be repeated with new initialization for the weights whenever there is need for that. The encryption/decryption processes are not time consuming at all. The training only takes almost one second if it is done on a multipurpose dual core computer. Of course, the identical input/target vector pairs that are used in the training are the same 256 possible binary inputs/targets used every time. This will

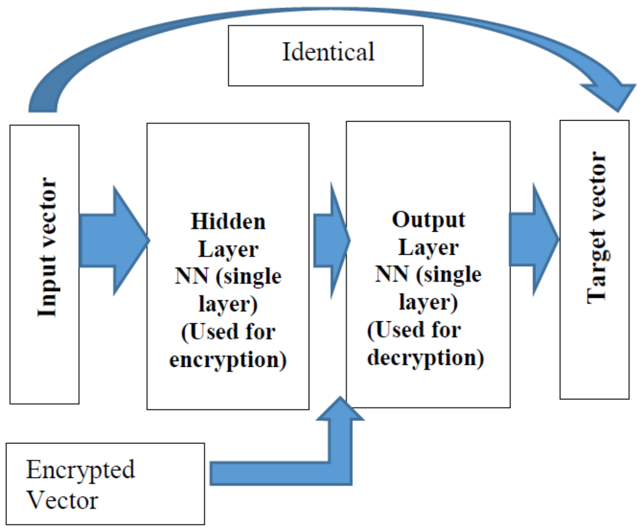
Identical
Input vector
Hidden Layer NN (single layer) (Used for encryption)
Output Layer NN (single layer) (Used for decryption)
Target vector
Encrypted Vector

Figure 1. The mirror two-layered NN used for encryption/decryption

finish the stage of training. The average CPU time for this is 1.0313 seconds using Matlab 14 on Core i5 7th generation computer with 8GB RAM. That could be minimized using more efficient hardware and using assembly language instead of Mat lab 14 since we are dealing with binary values (future work). For every new block of data being transmitted (10K bits), a new training phase could be invoked and a new set of weights (for the output layer) can be generated and sent along with the encrypted data. The weights for the output layer are sent with the encoded data at the beginning of the encrypted block. The whole process will be repeated as long as there is data transmission. More details regarding the NN properties are shown below.

Network Properties:

Number of layers = 2
Inputs = 8
One hidden layer
Number of neurons in hidden layer is 8.
Transfer function is TANSIG.
Number of neurons in output layer is 8.
Network type: Feed-forward backpropagation
Training function: Levenberg-Marquardt backpropagation algorithm (Trainlm)
Adaption learning function: Gradient descent with momentum weight and bias learning function (Learngdm)
Performance function: Mean squared error
Training ratio: 0.7
Validation ratio: 0.15
Test ratio: 0.15

Training algorithm used here is Bayesian regularization backpropagation, please see Figure 2 and Figure 3. In Figure 3, both curves for "Train" and "Test" totally match.

Network training updates the weight and the bias values according to Levenberg-Marquardt optimization. It minimizes a combination of squared errors and weights, and then determines the correct combination so as to produce a network that generalizes outputs similar to targets. This training process is called Bayesian regularization. In our network, the training stops when the performance gradient falls below minimum gradient error value. In Figure 2, mu is the control parameter for the algorithm used to train the

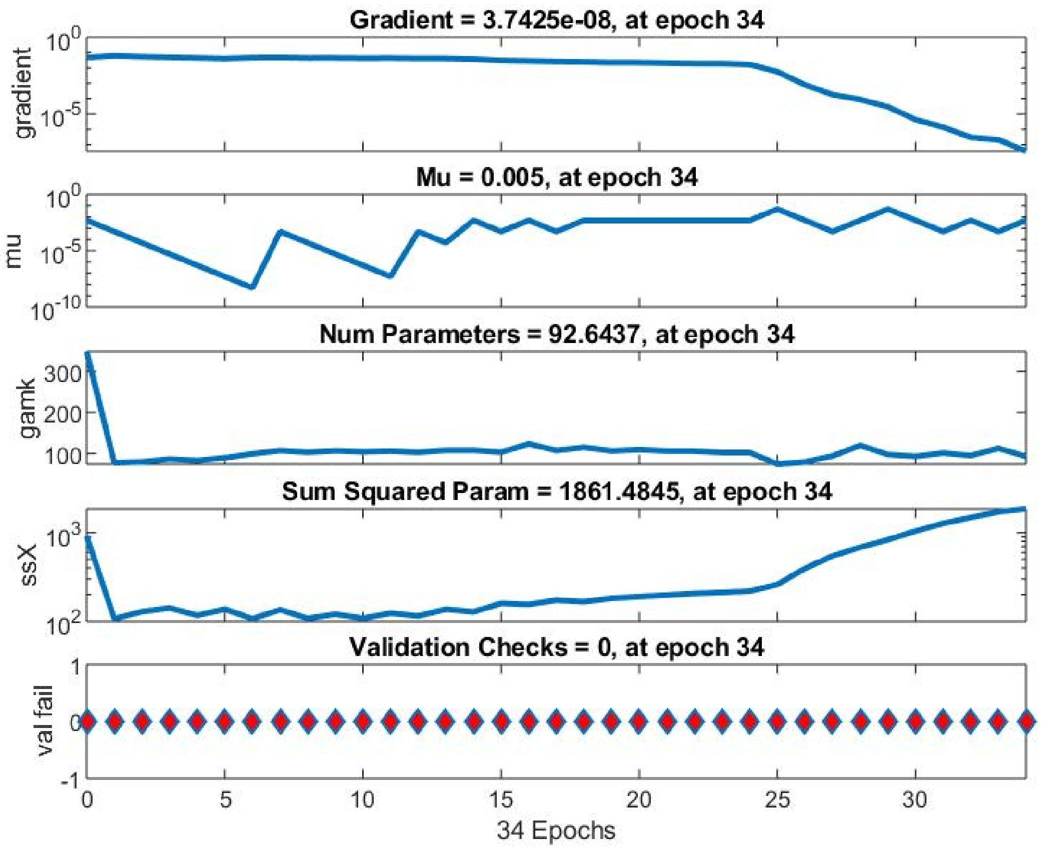

FIGURE 2. Training and validation parameters

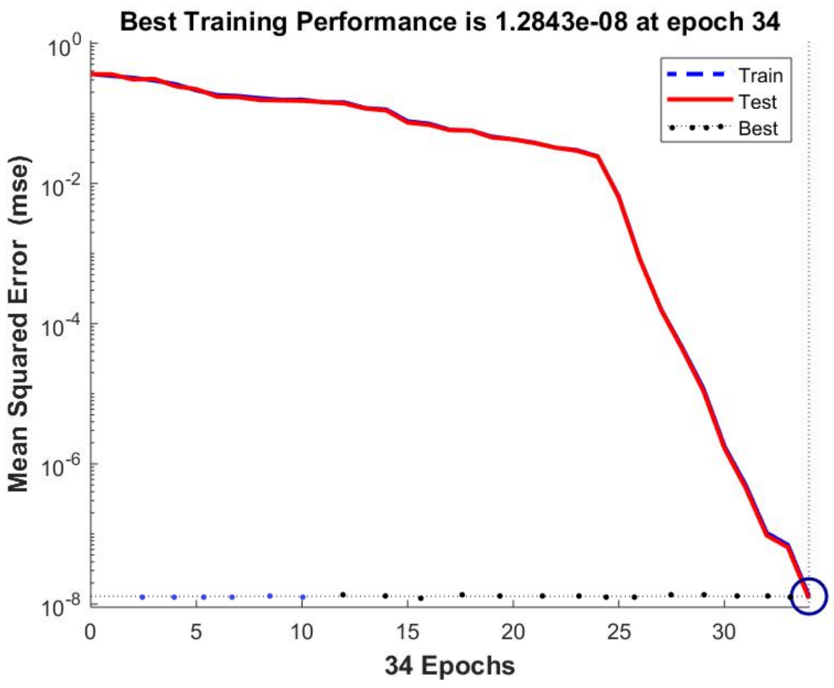

FIGURE 3. Training error versus epochs (an epoch is a whole batch of input vectors)

neural network. *Gamk* is the effective number of parameters. *ssX* is the Sum squared Parameter. *Valfail* is number of failures in validation checks which was zero for all epochs [13].

3. **Binary Weights & Input/Output.** Maximum value for weight found was less than 23, so it requires only 5 bits as exponent. Precision values are limited to 6 bits weights, and the key can be represented as a 12 bits vector as follows:

Sign	Exponent					Precision					
$+/-$	1	2	3	4	5	1	2	3	4	5	6

The Appendix (Tables 1, 2 and 3) shows the weights of both layers as a training outcome example. All the encrypted values are associated with weights and biases shown in those tables. At each new phase of training, new sets of weights (new keys) will be generated.

4. **The Encryption/Decryption Process.** Training is completed using a whole set of binary values from 0 to 255 as inputs and identical targets (mirroring). This would cover all possible binary values of characters that could be generated with 8 bits. The weights in the hidden layer are used as a key to encrypt the inputs and the weights in the output layer are used as a different key for decrypting the encrypted inputs. When a block is packed for transmission it is accompanied with the binary values of the output layer weights to be used as the decryption key. Those weights consist of $8 \times 8 = 64$ values. If each weight is encoded with 12 bits, as mentioned above, then the total decryption key has $8 \times 8 \times 12 = 768$ bits. With proper compression for the binary values of the key, the key length can be reduced by $1 : 11$ ratio [14]. On average, the key length will be around 70 bits and that is acceptable if a block of data with length 10K bits are processed.

For more security and confusion in the sent data, the whole process is repeated for every 100 blocks of data being transmitted.

A new decryption key will be generated and accompanied with the next 100 blocks of encrypted data. All will be packed and sent in a similar way to the previous phase. In this case we guarantee that the same key will not be used for more than 100 blocks of data. If the transmission rate is 1M bits per second (10K $\times$ 100) and the total time needed to transmit the 100 blocks is 1 second, then in this 1 second (assuming we are using our Mat lab 14 and the 5i dual core computer) will be enough to generate another key. The new encrypted block will be pipelined with the previous block. Other processing times are negligible and can be discarded.

5. **Discussions.** The high non-linearity and the unpredictability of resulting weights of the NN result in highly elusive encrypted codes and unpredictable keys. The keys, however, are modified every 100 blocks of data transmitted. This is done with no waste of time as transmission process and new training of the NN will be going on in parallel. The standard deviation of randomly selected blocks of data is around 100. This is an acceptable value knowing that the decimal value of transmitted data (encoded/encrypted) varies from 1 to 256 (8 bits). Random test for data correlation gives values of -0.3 based

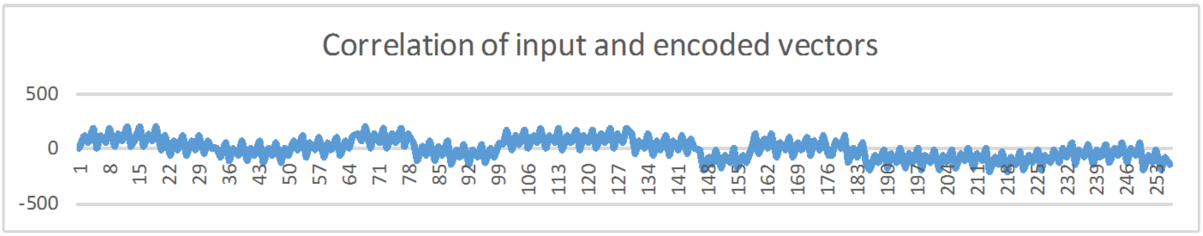

Figure 4. Correlation of a sample between input values and encrypted values

on the decimal values of the encoded data. Figure 4 shows the relation between the decimal values of text (binary) and the encoded (encrypted) data. The decimal input values of the 256 possible characters show no correlation/relation at all with the associated encrypted values. The high nonlinearity and no-association pattern makes it difficult for any hacker to break the encryption. The correlation factor at different segments (every 10 characters) showed different values between $[-1]$ and $[+1]$. In general, the approach is compact and very simple, the ability to re-generate new keys and their highly nonlinear implementation by the neural network is the reason of producing very illusive encrypted data. The key used for encryption is different from the key used for decryption. The key used for decryption is used at the receiver end only. This approach is direct comparing to other existing methods that use boxes, meshes, and tables for different mappings. No exhausting shuffling is used. Shuffling, mapping, and encoding are implemented implicitly and in parallel by the hidden layer of the NN. The output layer of the NN is reversing what the hidden layer has done. The 8 bits input is processed in parallel due to the parallel nature of the NN. This implies more efficiency and collectivity in generating the encoded data. Figure 2 and Figure 3 summarize the training process. The validation has zero failures and the NN was acting like a mirror perfectly. More efficient training techniques could be used to minimize training time. In this method, no keys are directly generated based on random numbers generators. Expanded keys used in many methods are also indirectly based on those random number generators. Those generators are pseudo in nature and their output behaviours are cyclic and predictable. Although the weights in the NN are initially generated randomly, the encoded data do not depend directly on those keys. All weights are incrementally updated during the training (learning) process. Moreover, their output is passed through a highly nonlinear activation function with variable characteristics. The encrypted data cannot be reversed even if the encryption key is known.

6. Conclusions and Future Work.

Although many researchers have strong believes that Neural Networks (NN) are not appropriate at all for encryptions and decryptions, in this work are showing the applicability of NN in efficient encryption/decryption processes. With appropriate selection of architecture and training parameters and with the availability of reasonable hardware, blocks of text or input data can be efficiently encrypted in a manner very competent to traditional methods. Future work will focus on using faster and more efficient techniques for training the NN such as [21-28]. Hybrid encryption or another stage of encryption could be added, more layers of NN to minimize any correlation between the plain text (inputs) and the encrypted data can be tested.

REFERENCES

[1] M. J. Al-Muhammed and R. A. Zitar, Dynamic text encryption, *International Journal of Security and Its Applications (IJSIA)*, vol.11, no.11, pp.13-30, 2017.

[2] A. Bogdanov, F. Mendel, F. Regazzoni, V. Rijmen and E. Tischhauser, ALE: AES-based lightweight authenticated encryption, in *Fast Software Encryption*, S. Moriai (ed.), Berlin, Heidelberg, Springer, 2014.

[3] L. R. Knuden, Dynamic encryption, *Journal of Cyber Security and Mobility*, vol.3, pp.357-370, 2015.

[4] N. Mathur and R. Bansode, AES based text encryption using 12 rounds with dynamic key selection, *Procedia Computer Science*, vol.79, pp.1036-1043, 2016.

[5] J. Daemen and V. Rijmen, *The Design of Rijndael: AES – The Advanced Encryption Standard*, Springer, Berlin, German, 2002.

[6] T. Nie and T. Zhang, A study of DES and Blowsh encryption algorithm, *Proc. of IEEE Region 10 Conference*, Singapore, 2009.

[7] M. J. Al-Muhammed and R. A. Zitar, Lookback random-based text encryption technique, *Journal of King Saud University – Computer and Information Sciences*, doi: https://doi.org/10.1016/j.jksuci.2017.10.002, 2017.

[8] P. Patil, P. Narayankar, D. G. Narayan and S. M. Meena, A comprehensive evaluation of cryptographic algorithms: DES, 3DES, AES, RSA and Blowfish, *Procedia Computer Science*, vol.78, pp.617-624, 2016.
[9] *NIST Special Publication 800-67 Recommendation for the Triple Data Encryption Algorithm (T-DEA) Block Cipher Revision 1*, Gaithersburg, MD, USA, 2012.
[10] W. Stallings, *Cryptography and Network Security: Principles and Practice (7th Edition)*, Pearson, 2016.
[11] R. Anderson, E. Biham and L. Knudsen, *Serpent: A Proposal for the Advanced Encryption Standard*, http://www.cl.cam.ac.uk/rja14/Papers/serpent.pdf, 2018.
[12] C. Burwick, D. Coppersmith, E. D'Avignon, R. Gennaro, S. Halevi, C. Jutla and N. Zunic, *The MARS Encryption Algorithm*, IBM, 1999.
[13] *https://www.eui.eu/ServicesAndAdmin/ComputingService/Software/GuideMatLab*, 2019.
[14] M. Isenburg and J. Snoeyink, Binary compression rates for ASCII formats, *Proc. of Web3D Symposium'03*, pp.173-178, 2003.
[15] *Online Random Key Generator Service*, https://randomkeygen.com, 2019.
[16] J. J. Soto, Randomness testing of the AES candidate algorithms, http://csrc.nist.gov/archive/aes/round1/r1-rand.pdf, 2018.
[17] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray and S. Vo, *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*, 2001.
[18] J. Soto, Randomness testing of the AES candidate algorithms, *NIST IR 6390*, 1999.
[19] T. Komal, R. Ashutosh, R. Roshan and S. M. Nalawade, Encryption and decryption using artificial neural network, *International Advanced Research Journal in Science, Engineering and Technology*, vol.2, no.4, 2015.
[20] N. Rathee, R. Sachdeva, V. Dalel and Y. Jaie, A novel approach for cryptography using artificial neural networks, *International Journal of Innovative Research in Computer and Communication Engineering*, vol.4, no.4, 2016.
[21] R. A. Zitar and A. K. Al-Jabali, Towards general neural network model for glucose/insulin in diabetics-II, *Informatica: An International Journal of Computing and Informatics*, vol.29, 2005.
[22] A. F. Nuseirat and R. A. Zitar, Trajectory path planning using hybrid reinforcement and back propagation through time training, *International Journal of Cybernetics and Systems*, vol.34, no.8, 2003.
[23] R. A. Zitar and M. H. Hassoun, Neurocontrollers trained with rule extracted by a genetic assisted reinforcement learning system, *IEEE Trans. Neural Networks*, vol.6, no.4, pp.859-879, 1995.
[24] M. M. Al-Tahrawi and R. A. Zitar, Polynomial networks versus other techniques in text categorization, *International Journal of Pattern Recognition and Artificial Intelligence*, vol.22, no.2, pp.295-322, 2008.
[25] R. A. Zitar and A. Hamdan, Spam detection using genetic based arti_cial immune system: A review and a model, *Artificial Intelligence Review*, 2011.
[26] R. A. Zitar, Optimum gripper using ant colony intelligence, *Industrial Robot Journal*, vol.23, no.1, 2004.
[27] R. A. Zitar and A. M. Al-Fahed Nuseirat, A theoretical approach of an intelligent robot gripper to grasp polygon shaped object, *International Journal of Intelligent and Robotic Systems*, vol.31, pp.397-422, 2001.
[28] A. M. Al-Fahed Nuseirat and R. A. Zitar, Neural network approach to firm grip in the presence of small slips, *International Journal of Robotic Systems*, vol.18, no.6, pp.305-315, 2001.

Appendix

Table 1. Weights of the hidden layer-keys (example)

Neuron 1	**Neuron 2**	**Neuron 3**	**Neuron 4**	**Neuron 5**	**Neuron 6**	**Neuron 7**	**Neuron 8**
−0.001111	−0.010000	−0.001100	−0.010011	−0.010010	−0.001010	−10110.01001	0.001
−0.001100	−0.001110	−0.000100	−0.010001	10110.0100	−0.010000	0.010001	0.001101
−0.000111	−0.001100	10110.010010	−0.010010	−0.001010	−0.001111	0.001111	0.000111
−0.001110	−0.010010	−0.000111	−0.010011	−0.010000	−0.001011	0.001011	10110.011
−0.001100	−0.001110	−0.00111	−0.010100	−0.010010	10110.010100	0.001011	0.001001
0.100011	0.011000	0.010010	10001.110110	0.001111	−0.010011	0.001101	0.011101
−0.001111	10110.001110	−0.001010	−0.010010	−0.010011	−0.0011	0.001101	0.01
10110.010110	−0.010010	−0.000110	−0.010010	−0.010000	0.001101	0.01001	0.001101

Table 2. Weights of the output layer-keys (example)

Neuron 1	**Neuron 2**	**Neuron 3**	**Neuron 4**	**Neuron 5**	**Neuron 6**	**Neuron 7**	**Neuron 8**
0.00011	0.000111	0.000110	0.00011	0.000111	0.000111	0.00011	100.11011
0.00011	0.001000	0.000110	0.000101	0.00011	0.00011	100.11101	0.00011
0.001001	0.001001	101.000011	0.001001	0.001000	0.000111	0.001001	0.001001
0.000010	0.000111	0.000111	0.000001	0.000001	11.01111	0.0000011	0.000010
0.000111	101.00001	0.000111	0.000111	0.000111	0.000101	0.001000	0.001000
0.001000	0.001000	0.001000	0.001000	101.0001	0.000100	0.001000	0.001000
100.111000	0.000110	0.000101	0.000110	0.000111	0.000100	0.000111	0.000101
0.00011	0.000011	0.000011	100.100111	0.000111	0.000011	0.000100	0.000011

Table 3. Values of the bias in both layers-keys

B1	**B2**
−0.001	−1010.001111
−0.000111	−1010.01
−0.001001	−1010.010101
−0.000101	−1010.0101
−0.000111	−1010.010001
−0.001	−1001.0011
−0.000111	−1010.001011
−0.000111	−1010.01001