COMPUTATIONAL PROBLEMS OF ELECTRICAL ENGINEERING

Vol. 10, No. 2, 2020

# ENCRYPTION OF TEXT MESSAGES USING MULTILAYER NEURAL NETWORKS

**Volodymyr Brygilevych[1], Nazar Pelypets[2], Vasyl Rabyk[3]**

[1] Institute Of Technical Engineering the State Higher School
of Technology and Economics in Jarosław, Jarosław, Poland,

[2] Department of Sensory and Semiconductor Electronics Ivan Franko National University of Lviv, Ukraine,

[3] Department of RadioPhysics and Computer Technologies Ivan Franko National University of Lviv, Ukraine

*vbrygilevych@pwste.edu.pl, nazarpelypets@ukr.net, vasyl.rabyk@lnu.edu.ua*

**Abstract:** The article considers an algorithm for encrypting / decrypting text messages using multilayer neural networks (MLNN). The algorithm involves three steps: training a neural network based on the training pairs formed from a basic set of characters found in the text; encryption of the message using the weight coefficients of the hidden layers; its decryption using the weight coefficients of the output layer. The conditions necessary for successful encryption / decryption with this algorithm are formed, its limitations are emphasized. The MLNN architecture and training algorithm are described. The results of experimental research done by using the NeuralNet program are given: training the MLNN employing the BP (Sequential), BP (Batch), Rprop, QuickProp methods; an example of encrypting / decrypting a text message.

**Key words:** encryption, decryption, multilayer neural networks, training algorithms, NeuralNet program.

## 1. Introduction

The amount of information transmitted through public communication networks is growing every year. The main task, the importance of which is gaining increase, is the security of this information. Cryptography is one of the important aspects of secure communication, aiming to protect information from unauthorized access. Cryptography provides availability, privacy, and integrity of information.

Cryptographic encryption systems are based on two approaches to the use of keys. In a system with one secret key (symmetric encryption), the key is known only to the sender and receiver of information. The sender encrypts a message (P) with the key (K) to obtain an encrypted message (C) [1]. Having been transmitted over the network, the encrypted message (C) is decrypted using the secret key (K). Asymmetric encryption uses two keys [1]: a public key (K_Pub) to encrypt a message and a private key (K_Prv) to decrypt it.

In recent years, more and more works [2]–[6] on the use of neural networks in cryptography have begun to appear. In such cryptographic data encryption / decryption systems, the secret key is the weights of the neural network, and its architecture. The advantages of such systems are that they are very difficult to break without knowing the methodology underlying these systems.

Feedforward multilayer neural networks for the encryption / decryption of messages are used in [2]. The key in the proposed algorithm is the architecture of the neural network and its weight coefficients. During the encryption phase, the neural network converts 6-bit input sets of a message into 6-bit output sets, which are transmitted over the communication network. When decrypting a message, the neural network inversely converts the received output sets into an input message.

Symmetric data encryption based on counter propagation networks (CPN) is considered in [3]. During encryption, each message character is converted to the ASCII binary format, which is used as a target value for the Grossberg layer and forms a set of input data of the Kohonen layer of the CPN. An encrypted text which together with the target value is transmitted to the receiver is obtained at the output of the Kohonen layer trained. At the decryption stage, the received encrypted text is fed to the input of the Grossberg layer, and the obtained target values are set for the outputs. The Grossberg layer having been trained, each resulting binary value of the ASCII format is converted back to the corresponding character.

In [4], the authors consider the algorithm of asymmetric data encryption. Generation of a private key and encryption are performed on the basis of Boolean algebra using the Permutation and Doping functions. Decryption and generation of a public key scheme employs a MLNN trained by using an inverse error propagation algorithm.

In [5], an algorithm for encrypting 8-bit input data using a clipped Hopfield neural network (CHNN) is proposed. The authors note the simplicity of its architecture and the possibility of expanding the bit size of input data by cascading such networks. The work gives a comparison of this algorithm, implemented on Xilinx FPGA, with the standard symmetric encryption algorithm DES in term of their performance [1].

A symmetric encryption algorithm based on real-time recurrent neural networks (RRNN) is considered in [6]. This algorithm has a relatively simple architecture, allows a variable key length, a variable length of the input block, and increased security to be maintained. The proposed RRNN has a multilayer structure. The dimension of the input layer *X* is twice the dimension of the output layer *Y*. One of the hidden layers consists of only one neuron. The symmetric encryption algorithm works in two stages: key extension and data encryption / decryption.

The main purpose of this work is to implement the algorithm for encrypting / decrypting text messages based on a MLNN, training and testing of the network using the NeuralNet program [7]. In [7], this program is used to recognize handwritten characters. The modification of the program was performed to form from text messages sets of input data of a given dimension, normalize these sets during encryption, and denormalize the output data during decryption. The MLNNs with one or two hidden layers were used in the studies.

**2. MLNN and its training algorithm**

Fig. 1 shows the MLNN used to encrypt / decrypt data. These are fully connected and feedforward neural networks (FNN). The neurons in the first layer are connected to the inputs of the neural network. The number of hidden layers may vary.

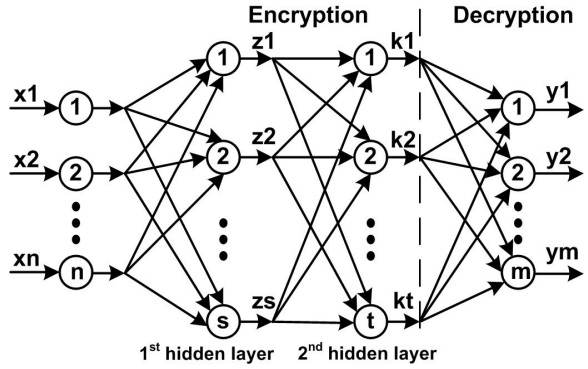



*Fig. 1. MLNN architecture.*

The network consists of an input layer, two hidden layers and an output one. The first hidden layer contains *s* neurons, and the second – *t* neurons. When encrypting / decrypting messages, it was assumed that the number of neurons in the output layer is equal to the number of the inputs ($m = n$). The n number of the network inputs and their bit size may vary.

The number of weights in the MLNN shown in Fig. 1, is defined as:

$$N_w = s(n+1) + t(s+1) + m(t+1). \quad (1)$$

Each neuron in the first hidden layer $z\_in_j$, $j$=1, …, $s$ receives signals $x_i$, $i$=1, …, $n$ from all the network inputs [8]:

$$z\_in_j = \sum_{i=0}^{n} x_i w_{ij}, \quad (2)$$

where $w_{0j}$ stands for the bias of the $j$-th neuron of the first hidden layer. The signals at the outputs of the neurons of the first hidden layer are calculated through the activation function:

$$Z_j = f(z\_in_j), \ \ j = 1, ..., s \quad (3)$$

and are fed to the inputs of neurons of the second hidden layer. Similarly, we determine the signals at the outputs of neurons of the second hidden layer.

The signals at the outputs of neurons in the output layer:

$$Y_j = f(y\_in_j), \ \ j = 1, ..., n, \quad (4)$$

where

$$y\_in_j = \sum_{i=0}^{t} k_i v_{ij}. \quad (5)$$

To train a MLNN, an error backpropagation algorithm (BP) is used [8]. The algorithm includes the following steps: supplying the training pair to the network input and its feedforward propagation through the network; backpropagation of the error associated with this training pair; correction of network weight coefficients according to a certain criterion. This criterion consists in choosing such values of the network weights so that one might obtain the minimum total standard error of the MLNN for all training pairs.

The algorithm of error backpropagation is implemented using a sequential or batch mode. In the sequential mode, the correction of weights is performed after each training pair is presented to the MLNN. The error for the $l$-th training pair is determined by the expression [8]:

$$E(l) = \frac{1}{2} \sum_{j=1}^{m} \left( d_j(l) - y_j(l) \right)^2, \quad (6)$$

where $y_j(l)$, $d_j(l)$ is the actual and target output of the $j$-th neuron of the output layer of the network for the $l$-th training pair.

In the batch mode of the MLNN training, the weight correction is performed after all training pairs of the sequence are presented to the network. One cycle of presenting all training pairs is called an epoch. For the current epoch, the objective function is defined as [8]:

$$E(l)=\frac{1}{2N}\sum_{i=1}^{N}\sum_{j=1}^{m}\left(d_j(l)-y_j(l)\right)^2. \qquad (7)$$

where $N$ is the number of training pairs of the sequence; $l$ stands for the number of the neural network training epoch.

Minimization of objective functions (6), (7) is performed by gradient methods. In particular, in the method of steepest descent, the correction of weights $w_{ij}$ is selected in proportion to the partial derivative of the objective function of the error $E(l)$ with respect to $w_{ij}$:

$$w_{ij}(l+1)=w_{ij}(l)+\Delta w_{ij}(l), \qquad (8)$$

where

$$\Delta w_{ij}(l)=-\eta\frac{\partial E(l)}{\partial w_{ij}}, \qquad (9)$$

$\eta$ represents the neural network training rate coefficient.

Given expression (7), the last expression (9) can be written as:

$$\Delta w_{ij}(l)=-\eta\frac{\partial E(l)}{\partial w_{ij}}=-\frac{\eta}{N}\sum_{r=1}^{N}e_j(r)\frac{\partial e_j(r)}{\partial w_{ij}}, \qquad (10)$$

where the error signal $e_j(r)$ corresponds to the $j$-th neuron for the $r$-th training pair.

The error backpropagation algorithm based on the gradient method that are employed to train a MLNN has a number of disadvantages. These include: long training duration, the possibility of reaching a local minimum during training, the possibility of network paralysis.

The adaptive algorithms RProp, Quickprop are used to speed up the process of MLNN training. For the weights to be corrected in the batch mode, the algorithm Rprop [9] uses only signs of partial derivatives. The QuickProp algorithm [10] prevents looping at the point of a shallow local minimum, which most often occurs during the functioning of a neuron in the saturation domain of the activation function.

### 3. MLNN–based encryption and decryption algorithm

Encryption / decryption of messages based on a MLNN consists of several steps: training the network using the error backpropagation algorithm; data encryption using the weights of the hidden layers obtained in the first step; data decryption using the weight coefficients of the input layer obtained in the first step (see Fig. 1). In order to encrypt / decrypt messages successfully, training the network on training input pairs must be 100 %. The training is to determine the MLNN weight coefficients ($w_{ij}$ – for hidden layers and $v_{ij}$ – for the input layer).

In neural network systems intended for message encryption / decryption, the secret key is:

– network architecture: number of inputs, number of hidden layers and neurons in them, number of neurons in the output layer of the network (Fig. 1);

– matrix of MLNN weight coefficients ($w_{ij}$) for data encryption and decryption ($v_{ij}$);

– initial data for initializing the weight coefficients ($w_{ij}$, $v_{ij}$) of the network during its training;

– bit size of input training pairs during their encryption.

Input data for MLNN training is a set of letters (lowercase and uppercase) of the English alphabet, numbers and punctuation marks in the ASCII format ($N_S$ = 70 characters), for example:

```
'8'=0x38=00111000; 'p'=0x70=01110000

'Z'=0x5A=01011010; '?'=0x3F=00111111
```

Before training the network, a set of training pairs is formed and stored in a file. In order to form it, you need to specify the number of network inputs ($n$) and their bit size ($n_b$). The number of the training pairs is defined as

$$N_{Tr}=\frac{8N_s}{n\cdot n_b}, \qquad (11)$$

where 8 bits are the bit size of each message character in the ASCII format. If in expression (11) $8N_s$ is not divisible evenly by $n\cdot n_b$, then the last training pair is padded with zeros to $n_b$ bits.

Encryption / decryption of text messages was performed using the NeuralNet program [7], written in C # in Microsoft Visual Studio 2013.

The NeuralNet program provides two modes of operation: “Training” – training the neural network on the basis of the input training pairs formed from $N_S$=70 characters; “Work” – encryption or decryption of arbitrary text messages based on the trained MLNN.

Neural network training is performed in one of the implemented methods: BP (sequential), BP (batch), RProp, QuickProp. To configure the NeuralNet program in the training mode, it is necessary to read the file with the input training pairs and specify the following data:

– the activation function steepness ($0<alfa\leq 1$);

– the neural network learning rate ($0<\eta\leq 1$);

– the network training error (*eps*);

– the error (*eps1*) of the neural network outputs against their desired values;

– the initial value for the initialization of network weights;

– the number of inputs $n$ of the network;

–the number of hidden layers;

–the number of neurons in the hidden layers;

–the number of network outputs;

–the number of training pairs;

–the maximum number of network learning epochs.

The choice of initial values of neuron weight coefficients and biases in the network affects the rate of its learning and synchronization of neural networks of the sender and receiver of messages. The weight coefficients initialization procedure implemented allows one to set their initial values to be the same from the interval (-0.5; 0.5) for different MLNN exercises or to change them. This makes it possible to get the same weights of the network after its being trained by the sender and receiver of messages.

The NeuralNet program implements the following criterion for complete training of the neural network using the error backpropagation algorithm: the value of the objective function (expressions (6) or (7)) is less than the specified error of the network training *eps* or the number of epochs is greater than the specified maximum value. When training the neural network, the training results are saved in a text file lerning.txt.

To switch to the encryption or decryption mode using a trained MLNN, go to the "Work" tab of the main interface window and select the appropriate mode. Then download the input file to encrypt the message or the encrypted file to decrypt it. When encrypting, the program will display information on the results of encryption: the number of input pairs; pair recognition accuracy in percentage; the number of unrecognized pairs. When decrypting messages, the received input pairs and characters are displayed in the ASCII format.

Consider an example of encryption and decryption of the following message:

*Multilayer neural networks.*

This message consists of 27 characters that are part of the $N_S$=70 base characters used in MLNN training. Convert it to the ASCII format and create a set of input pairs for encryption. Here is a message in the ASCII code:

```
0x4D 75 6C 74 69 6C 61 79 65 72 20 6E 65
  75 72 61 6C 20 6E 65 74 77 6F 72 6B 73
  2E
```

and the training pairs ($N_{Tr}$=27) for $n$=8 and $n_b$=1:

```
01001101 – 1-a,
01110101 – 2-a,
01101100 – 3-a, …
```

Step 1. Before encrypting this message, train the MLNN by specifying its architecture and generating a training pair file for the set of $N_S$=70 characters:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijkl
mnopqrstuvwxyz0123456789space!.,-?()
```

The training results of a MLNN with different architectures and by different methods are presented in Table 1. Training of all MLNNs was performed by the NeuralNet program with the initialization of identical initial values of weights, errors *eps* = 0.0001, *eps1* = 0.1. Other network parameters – *alfa* = 1.0, η = 0.8. The selected errors were sufficient for 100 % network training. For the 16_8_16_16, 16_12_16_16 network architecture, the training error was *eps* = 0.005. 100 % of its training therewith was also obtained. The bit rate of the input data is selected $n_b$=1. For $n_b$>2 it was not possible to teach MLNN 100 %. As the complexity of the network (architecture 16_8_16_16, 16_12_16_16) increases, so does the time of its training.

*Table 1*

**MLNN training results for a 70-character input basic set**

| Architecture MLNN | Training Algorithm | $N_{Tr}$, $N_W$ | $N_{Epoch}$ |
|---|---|---|---|
| 4_4_4 | BP (Sequential) | 140, 40 | 666 |
| 4_4_4_4 | RProp | 140, 60 | 611 |
| 4_8_4_4 | QuickProp | 140, 96 | 25 |
| 4_12_4_4 | BP (Sequential) | 140, 132 | 426 |
| 8_8_8 | BP (Sequential) | 70, 144 | 2348 |
| 8_8_8_8 | BP (Sequential) | 70, 216 | 2009 |
| 8_12_8_8 | BP (Sequential) | 70, 284 | 2421 |
| 16_16_16 | BP (Sequential) | 35, 544 | 5308 |
| 16_8_16_16 | BP (Batch) | 35, 552 | 238542 |
| 16_12_16_16 | BP (Batch) | 35, 684 | 170186 |

Step 2. Encrypting a message. Suppose a trained network has an 8-8-8 architecture. A message input file is downloaded by the program functioning in its Encryption mode. The values of the outputs of the hidden layer will be the encrypted message. Each input pair is encrypted with 8 output values. The following code is obtained for the 1st input pair:

```
0,968486351114214 0,989043133425048
0,813025661203787 0,011391139946895
0,064313178643986 0,652372929720987
0,789000493082475 0,668199366103064
```

For $N_{Tr}$=27, the dimension of the incoming message is 27 Byte, and the encrypted message is 27·8·8=1728 Byte.

Step 3. Decrypting a message. A public key is also transmitted in the file with the message encrypted. This key specifies the network architecture, its basic parameters and the starting value for the initialization of the network weight coefficients. The receiver of the message reads the public key data from the file and trains the network. In our case, the architecture and parameters of the network are the same as for step 1. After that, the program loads the encrypted data in the Decryption mode and decrypts them. As a result, we get a file in which each character is

represented in the ASCII format. The first 10 characters of the decrypted message are as follows:

```
01001101 → 'M' 01110101 → 'u'
01101100 → 'l' 01110100 → 't'
01101001 → 'i' 01101100 → 'l'
01100001 → 'a' 01111001 → 'y'
01100101 → 'e' 01110010 → 'r'
```

Analyzing the results of decryption, we see that the received message coincides with the incoming message that was encrypted.

**4. Conclusion**

An algorithm for encrypting / decrypting text messages based on an MLNN has been developed. This encryption algorithm, as well as the asymmetric encryption methods, uses public and private keys. The public key, which includes a network architecture, basic parameters for the algorithms of its training, an initial value for the initialization of network weight coefficients, is transmitted together with an encrypted message. The private key is the MLNN weight coefficients obtained after its training with the public key parameters before decrypting the message.

The result of encryption of each input training pair of messages results in the outputs of the last hidden layer (8-byte real numbers). The dimension of the message encrypted in Byte is defined as: $27 \cdot 8 \cdot 8 = 1728$, where $N_{Tr}$ is the number of input pairs formed from the message being encrypted, $t$ is the number of neurons in the last hidden layer. The incoming message dimension: $N_S$ Byte.

For an encrypted message to be decrypted, it is fed to the input of the output layer of the MLNN trained by the receiver. These are the outputs of the output layer where we get the input training pairs in the ASCII format, which are converted into a text message.

A necessary condition for encrypting / decrypting text messages with this MLNN-based algorithm is its 100 % training by both the sender and the receiver of the message. Therewith the weight coefficients of the network in both cases must be the same. This condition affects the choice of the number of bits for input pairs. Thus, the experimental studies have shown the possibility of 100 % network training when choosing the bit size $n_b$=1, 2. For $n_b$>2, it is not possible to train the network with a given error *eps1*. Thus, for $n_b$=4, given the normalization of the input data in the range 0…1, the error *eps1* must be several times less than $1 / (24 - 1) = 0.06$ (6).

The encryption / decryption of text messages using NeuralNet written in C # in Visual Studio 2013 is considered. The results of training the MLNN of different architecture by the BP (Sequential), BP (Batch), Rprop, QuickProp methods for the basic set of input pairs according to the number of necessary epochs are given. In all cases, 100 % network training has been obtained: at the network output we obtain the target value that coincides with the input training pair with an error of *eps1*. An example of encrypting / decrypting a text message has been considered.

**5. References**


[1] B. Schneier, *Applied cryptography: Protocols, Algorithms*, Source Code in C, Triumf, p. 815, 2012.

[2] E. Volna, M. Kotyrba, V. Kocian, and M. Janosek, "Cryptography Based On Neural Network" // in *Proc. 26th European Conference on Modeling and Simulation*, pp. 386–391, 2012.

[3] V. Sagar and K. Kumar, "A Symmetric Key Cryptographic Algorithm Using Counter Propagation Network (CPN)", in *Proc. 2014 ACM International Conference on Information and Communication Technology for Competitive Strategies*, vol. ISBN, no. 978-1-4503-3216-3, 2014.

[4] K. Shihab, "A backpropagation neural network for computer network security", *Journal of Computer Science*, vol. 2, no. 9, pp. 710–715, 2006.

[5] Choi-Kuen Chan, Chi-Kwong Chan, L. P. Lee, L. M. Cheng, *Encryption system based on neural network*, Communications and Multimedia Security Issues of the New Century, Springer, pp. 117–122, 2001.

[6] M. Arvandi, S. Wu, A. Sadeghian, W. W. Melek, and I. Woungang, "Symmetric cipher design using recurrent neural networks", in *Proc. IEEE International Joint Conference on Neural Networks*, pp. 2039–2046, 2006.

[7] V. Bihday, V. Brygilevych, Y. Hychka, N. Pelypets, V. Rabyk, "Recognition of Handwritten Images Using Multilayer Neural Networks IEEE 2019", in *Proc. 11th International Scientific and Practical Conference on Electronics and Information Technologies*, ELIT 2019 – Proceedings.

[8] Simon Haykin, Neural Networks: A Comprehensive Foundation 2nd edition, Prentice Hall, NJ, USA ©1998, 842p, ISBN:0132733501.

[9] M. Riedmiller and H. Brawn, RPROP – a fast adaptive learning algorithms. Technacal Report // Karlsruhe: University Karlsruhe. 1992. http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=6A4F81B00868291D27499A6AADC6C330?doi=10.1.1.52.4576&rep=rep1&type=pdf

[10] S. E. Fahlman, "Faster Learning Variations on Backpropagation: An Empirical Study", in *Proc. 1988 Connectionist Models Summer School*, pp. 38–51, 1988.


## ШИФРУВАННЯ ТЕКСТОВИХ ПОВІДОМЛЕНЬ З ДОПОМОГОЮ БАГАТОШАРОВИХ НЕЙРОННИХ МЕРЕЖ

Володимир Бригілевич, Назар Пелипець, Василь Рабик

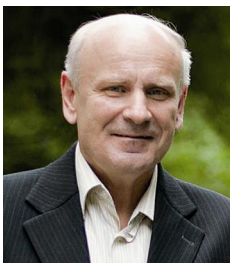
Розглянуто алгоритм шифрування/ дешифрування текстових повідомлень з використанням MLNN, який складається

з трьох кроків: навчання нейронної мережі на основі навчаючих пар, сформованих з базового набору символів, що зустрічаються в тексті; шифрування повідомлення з використанням ваг прихованих шарів; його дешифрування з використанням ваг вихідного шару. Сформовано необхідні умови для успішного шифрування/ дешифрування цим алгоритмом, підкреслено його обмеження. Описано архітектуру і алгоритм навчання MLNN. Приведено експериментальні дослідження з допомогою програми NeuralNet: навчання MLNN методами BP(Sequential), BP(Batch), Rprop, QuickProp; приклад шифрування/ дешифрування текстового повідомлення.

**Volodymyr Brygilevych** – PhD, Associate Professor of the Institute of Technical Engineering at the State Higher School of Technology and Economics in Jarosław, Poland.

Research interests: mathematical modeling of processes in mechatronic sysstems, diagnosis of electronic systems.

**Nazar Pelypets** – master student of the Department of Sensor and Semiconductor Electronics at Ivan Franko National University of Lviv. His research interests are connected with the development and implementation on computers neuron networks and their application for data forcasting, image recognition.



**Vasyl Rabyk** – Ph.D, associate professor of the Department of Radio-Physics and Computer Technologies at Ivan Franko National University of Lviv. He graduated from Ivan Franko State University of Lviv, Faculty of Physics, speciality RadioPhysics and Electronics Scientific and pedagogical interests are asociated with theoretical and practical aspects of diagnosis of electronic circuits, development of electronic devices on microcontrollers, FPGA, application of neuron networks for data forecasting, in cryptography.

*Received: 20.05.2020. Accepted: 25.07.2020.*