TASK QUARTERLY 9 No 4, 409–414

DATA SECURITY BASED ON NEURAL NETWORKS

KHALED M. G. NOAMAN AND HAMID ABDULLAH JALAB

Faculty of Science, Computer Science Department,
Sana'a University,
P.O. Box 13499, Sana'a, Republic of Yemen
{*drabujeehad,hamidjalab*}*@yahoo.com*

(Received 31 January 2005; revised manuscript received 16 May 2005)

Abstract: The paper is concerned with the study and design of a data security system based on neural networks. Data with different keys were taken as test data, encrypted, decrypted and compared with the original data. The results have confirmed its advantages over other techniques.

Keywords: data security, cryptography, neural networks

1. Introduction

With the introduction of the computer, the need arose for automated tools for protecting files and other information stored thereon. The generic name for tools designed to protect data and to thwart hackers is computer security [1].

By far the most important automated tool for network and communications security is encryption. The essence of the encryption technique is mapping data to a domain in a manner that is sniffing-proof. Two major techniques used in encryption are symmetric and asymmetric encryption.

In symmetric encryption, two parties share a single encryption-decryption key [2]. The sender encrypts the original message (P), which is referred to as plain text, using a key (K) to generate apparently random nonsense, referred to as cipher text (C), *i.e.*:

$$C = \text{Encrypt}(K, P). \tag{1}$$

Once the cipher text is produced, it may be transmitted. Upon receipt, the cipher text can be transformed back to the original plain text by using a decryption algorithm and the same key that was used for encryption, which can be expressed as follows:

$$P = \text{Decrypt}(K, C). \tag{2}$$

In asymmetric encryption, two keys are used, one key for encryption and another, paired key for decryption. One of the keys is kept private by the party that generated the key pair, while the other is made public. Theoretically, the encryption process is such that the message encrypted with the public key can only be decrypted with the

corresponding private key, while the message encrypted with the private key can only be decrypted with the corresponding Public key [3].

In this paper we present an encryption system based on neural networks (NNs). A neural network is used to construct an efficient encryption system by using a permanently changing key. The NN topology is an important issue, as it depends on the application the system is designed for. Consequently, since our application is a computation problem, we have used a multi-layer topology. In the present paper, General Regression Neural Network (GRNN), a simple, one-parameter neural network model, is proposed for the encryption-and-decryption process. Neural networks offer a very powerful and general framework for representing non-linear mapping from several input variables to several output variables. The process to determining the values of these parameters on the basis of a data set is referred to as learning or training, and so the data set is generally referred to as a training set. A neural network can be viewed as suitable choice for the functional forms used for encryption and decryption operations.

2. Design of the proposed system

The encryption function consists of the following three sub-functions, described in the following subsections:

1. the keys are created;
2. the input message is broken into blocks equal to the number of keys and processed, one block at a time, as input to the neural network;
3. the neural network (GRNN) is used to encrypt each block of data producing the encrypted message.

2.1. Creating the keys

First, we have to choose a super-increasing Knapsack with which to encrypt the data. The number of keys chosen is equal to N, so that:

1. the sum of these numbers (keys) must be less than 2^x, where $x = 2^N$, and
2. the numbers cannot be equal ($K_1 \neq K_2 \neq \ldots \neq K_n \neq 0$).

To keep our simulation simple, we have chosen objects of the following numbers: 27, 14, 68, so that $K_1 = 68$, $K_2 = 14$ and $K_3 = 27$ will be our keys.

2.2. Breaking the input data

Let us suppose that M is a set of N-bit initial unipolar data [4], *i.e.*:

$$M_i = \{0,1\}, 0 \leq i \leq N-1. \tag{3}$$

In this model, a 3-bit plain text is entered ($N = 3$) and an 8-bit cipher text is the output (2^N). Now, we need something to encrypt, for example 011010110. First, we break it down into blocks ($N = 3$) the length of our Knapsack, thus: 011 010 110. In the first block (011), there are 1's in the second and third positions. Thus, we take the second and third keys and add them to each other: $0+14+68 = 82$. Repeating the procedure for all the input data blocks yields the encrypted version of 01001000 as 82, as shown in Table 1. The encrypted data will then be transmitted to the recipient.

Table 1. The full pattern of training data sets

Input			**Output**							
P_3	P_2	P_1	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	1	0	0
0	1	0	0	0	0	0	1	1	1	0
0	1	1	0	1	0	1	0	0	1	0
1	0	0	0	0	0	1	1	0	1	1
1	0	1	0	1	0	1	1	1	1	1
1	1	0	0	0	1	0	1	0	0	1
1	1	1	0	1	1	0	1	1	0	1

2.3. The proposed Neural Network-based encryption system

A neural network is a structure (network) composed of a number of interconnected units (artificial neurons). Each unit has a characteristic input/output (I/O) and implements a local computation or function. The output of any unit is determined by its I/O characteristic, its interconnection to other units, and external inputs, if any [5].

In this paper, General Regression Neural Network (GRNN), a simple, one-parameter neural network model, is proposed for encryption and decryption.

General Regression Neural Network (GRNN) was developed by Specht [6] and is a simple yet very effective local approximation based on a neural network, in the sense of estimating a probability distribution function. The main advantages of GRNN are as follows:

- fast learning;
- convergence to optimal regression surface for large numbers of samples;
- effective use with sparse data;
- the capacity to handle non-stationary data.

GRNN uses a standard statistical formula for calculating the conditional mean Y of the scalar random variable y given a measurement X of a vector random variable x. The vector random variable x corresponds to the input of the network and the random variable y corresponds to the network's output. Apart from being used as a static regression technique, GRNN can be used when the data statistics changes over time, is the use derived recently by specifying a time constant and a threshold.

In order to build a GRNN:

1. set the number of input, pattern, and output layer (Processing Elements, or PEs);
2. choose the pattern unit;
3. choose the time constant and the reset factor;
4. set the radius of influence.

This is a simple clustering mechanism which assigns an input vector to a cluster if the cluster center is the cluster center nearest to the input vector AND closer than the radius of influence. Otherwise, the input vector is assigned as the center of a new cluster (if possible).

The implementation of GRNN allows an exponentially decaying σ of the following form:

$$\sigma = \frac{S}{N^{E/M}}\,, \tag{4}$$

where N is the number of pattern units, M is the number of input PEs, and E must lie between 0 or 1.

2.4. Building and training the Neural Network

The problem we face is a computational problem, so we will use a multi-layer GRNN. The GRNN used for encryption consists of three layers. Each layer consists of a number of neurons, depending on the case to be solved. In the encryption process, the input message is divided into 3-bit data sets, while 8-bit sets are produced after the encryption process. The basic GRNN architecture is shown in Figure 1. Each layer consists of the following:

1. an input layer consisting of 3 nodes, which represents the N-bit blocks;
2. a pattern layer of 8 nodes;
3. an output layer of 8 nodes, used to define the decrypted output message.

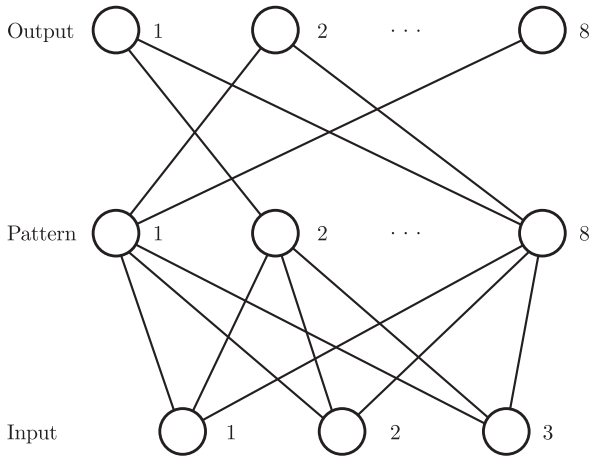

Figure 1. Three-layered GRNN architecture

The other parameters of the GRNN are as follows:

- time constant = 1000.0;
- reset factor = 0.000;
- radius of influence = 0.050;
- σ scale = 1.000;
- σ exponent = 0.500.

A value of 0.5 for E is suitable under most circumstances. A value of 0.0 for E allows to use a constant value for σ (equal to 5).

In order to study the behavior of the NNs, two training sets are used:

- a full pattern of inputs, which consists of all the possible inputs, or
- a half of the pattern mentioned above.

At the beginning of learning the encryption key, was GRNN fed with the valid states as shown in Table 1 for the full pattern and Table 2 for half the input pattern. The training set is repeatedly presented to the network and the weight values are adjusted until the overall error is below a predetermined level of tolerance. After the weight matrix is constructed, the network is tested for encryption.

Table 2. The half-full pattern of training data sets

Input			**Output**							
P_3	P_2	P_1	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	1	1	1	0
1	0	0	0	0	0	1	1	0	1	1
1	1	0	0	0	1	0	1	0	0	1

3. Results and discussion

In order to evaluate the discussed mechanism, the encryption steps of typical digital data are shown below. We have tested the behavior of the neural network described in the previous section, and found that:

1. the neural network works reliably when using the full pattern and absolutely no errors are found in the outputs, as shown in Table 3;
2. the network's performance is poorer when using a half or other part of the full input patterns. Due to numerous errors found in the outputs, the network fails to encrypt the input data, as shown in Table 4.

Table 3. The encryption network results for full pattern of inputs

Test input data			**Outputs**							
P_3	P_2	P_1	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
0	0	0	0.000000	0.018004	0.000325	0.017732	0.035753	0.035063	0.035109	0.018057
0	0	1	0.000000	0.982032	0.000325	0.017732	0.035753	0.964971	0.018050	0.018057
0	1	0	0.000000	0.018004	0.017733	0.000324	0.982373	0.947597	0.947279	0.018057
0	1	1	0.000000	0.982031	0.017733	0.982031	0.017733	0.052370	0.982031	0.018057
1	0	0	0.000000	0.018004	0.017699	0.964386	0.982407	0.018315	0.964703	0.982084
1	0	1	0.000000	0.982032	0.017699	0.964386	0.982407	0.981720	0.964391	0.982084
1	1	0	0.000000	0.018004	0.964455	0.017629	0.999678	0.034964	0.034907	0.982084
1	1	1	0.000000	0.982031	0.964455	0.017629	0.999678	0.965070	0.017945	0.982084

Table 4. The encryption network results for half-full pattern of inputs

Test input data			**Outputs**							
P_3	P_2	P_1	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
0	0	0	0.000000	0.000000	0.072325	0.197732	0.465575	0.195063	0.393109	0.269805
0	1	0	0.000000	0.000000	0.197733	0.072324	0.802373	0.534597	0.606279	0.269057
1	0	0	0.000000	0.000000	0.196862	0.534438	0.803407	0.072231	0.607703	0.731084
1	1	0	0.000000	0.000000	0.534455	0.196629	0.927678	0.196496	0.392907	0.731084

Another test has been performed to investigate the effect of the number of hidden units on the model's convergence. The results of this test are shown in Figure 2,

which demonstrates the variation of errors as a function of the number of neuron numbers in the hidden encryption layer. The errors decrease rapidly to zero at 8 hidden neurons, which indicates that the number of neurons in the hidden layer must be equal to the number of neurons in the output layer.

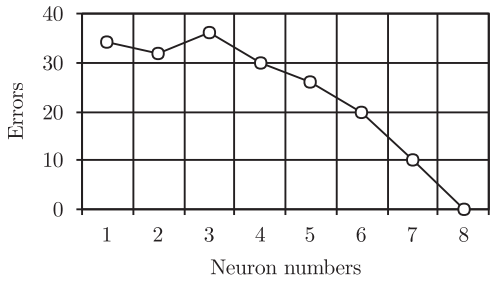

Figure 2. Error variations during training for the encryption processes

At the receiving terminal, the decryption process is the reverse of the encryption process. In order to decrypt the cipher data correctly:

1. the receiver must use the same key numbers to decrypt the data;
2. the data must have reached its intended receiver, as only the receiver knows the correct key numbers necessary to remove the encryption.

The message must have been authentic, because only the sender has the key numbers needed to encrypt the message so that receiver's key numbers will decrypt it correctly.

4. Conclusion

This paper presents an attempt to design an encryption system based on artificial neural networks of the GRNN type which is invariant to the secret keys. The proposed NN has been tested for various numbers of training iterations and for different numbers of hidden neurons, input data. The simulation results have shown a very good result, with relatively better performance than the traditional encryption methods.

References

[1] Stallings W 2002 *Data and Computer Communications*, Prentice Hall of India
[2] Tanenbaum A 1996 *Computer Networks*, Prentice Hall International, Inc.
[3] Stallings W 2002 *Cryptography and Network Security Principles and Practice*, Pearson Edition Asia
[4] Hossein R, Anoloni M and Samee M 2004 *Neural Network in Network Security*, ACIT 2003 Proceeding, pp. 274–281
[5] Schalkoff R 1999 *Artificial Neural Networks*, McGraw-Hill
[6] Specht D F 1991 *IEEE Trans. Neural Networks* **2** (6) 568