A FIRST STUDY OF THE NEURAL NETWORK APPROACH IN THE RSA CRYPTOSYSTEM

G.C. MELETIOU
T.E.I. of Epirus,
P.O. Box 110,
GR–47100 Arta, Greece.
email: gmelet@teiep.gr

D.K. TASOULIS
Department of Mathematics,
University of Patras Artificial
Intelligence Research Center (UPAIRC),
University of Patras,
GR–26110 Patras, Greece.
email: dtas@math.upatras.gr

M.N. VRAHATIS
Department of Mathematics,
University of Patras Artificial
Intelligence Research Center (UPAIRC),
University of Patras,
GR–26110 Patras, Greece.
email: vrahatis@math.upatras.gr

ABSTRACT
The RSA cryptosystem is supposed to be the first realization of a public key cryptosystem in 1977. Its (computational) security is relied upon the difficulty of factorization. In order to break the RSA cryptosystem it is enough to factorize N where N is the product of two large prime numbers, $N = p \cdot q$. This is equivalent to calculate $\varphi(N) = (p-1)(q-1)$ where φ is the Euler function. In this paper Neural Networks are trained in order the function to be computed.

KEY WORDS
Artificial Neural Networks, Factorization, Public Key Cryptography, RSA Cryptosystem

1 INTRODUCTION

1.1 CRYPTOSYSTEMS

Cryptography is both a very difficult area of mathematics and a key tecknolodgy for the information society. Its importance is related to recent development of electronic transactions, e-commerce, e-bussiness and other. Actually cryptography is an old art and a young science.

The basic objective of cryptography is to enable two persons to comunicate over an insecure chanel in such a way an opponent cannot understand what is being said.

In general there are two well known types of cryptosystems which are mentioned in what follows.

The two people which comunicate are usally reffered as “Alice” and “Bob”.

PRIVATE KEY CRYPTOSYSTEMS
They are also referred as classical, symmetric, single-key, secret key. Alice and Bob want to communicate over an unsecure (public) channel. Alice generates a key k and transmits it to Bob with the help of a “special” secure (private) channel which is used only for key transmission. Then Alice composes the message x and encrypts it with the help of the key k. The cryptogram $y = E_k(x)$ is transmitted over the insecure channel. Bob decrypts the message and recovers the original $x = D_k(y) = D_k(E_k(x))$. The key k is supposed to be a symmetric key since its owner is able to do both : encryption and decryption. The disadvantage of the private key cryptosysetm is the requirement for a private secure channel for key transmission.

PUBLIC KEY CRYPTOSYSTEMS
They are also referred as non-symmetric, double key. Alice and Bob want to communicate over an unsecure (public) channel. Bob obtains a couple of keys (d, e). Then he puts the key e (the public key) on a public domain, however he keeps key d (the private key) secret. It is computationally impossible (there is no efficient algorithm) for obtaining d from e. Then Alice composes the message and encrypts it with the public key e of Bob. The cryptogram y is computed as $y = E(x) = f_e(x)$ and transmitted to Bob. Bob uses his private key d to decrypt the cryptogram as $y = D(E(x)) = f_d(f_e(x))$. Since only Bob owns d he is the only user who can recover x from y.

1.2 THE RSA

DESCRIPTION
The well known *RSA cryptosystem* [18] is supposed to be the first realization of a public key system (proposed in 1977 by Rivest, Shamir, Adleman). Since then several public key systems [2, 3, 7, 13, 16] have been proposed, whose security rests on hard computational problems [16, 3]. Since RSA is a public key cryptosystem it cannot provide unconditional security. Its security is based on the difficulty of factoring large integers. The mathematical (algebraic - number theoretical) structure which is used is the ring $\mathbb{Z}_N$, where N is the product of two odd large prime numbers, $N = p \cdot q$. The original messages and the cryptograms are elements of $\mathbb{Z}_N$ that is integers modulo N (every message is divided into blocks, each block becomes a sequence of bits, that is a number $x,\ 0 \leqslant x \leqslant N-1$). The encryption and the decryption procedures are modular exponentiations.

IMPLEMENTATION
Two large odd primes are chosen [20]. Their product $N = p \cdot q$ is computed. Then the quantity $\varphi(N) = (p-1)(q-1)$ is computed. The next step is to find e, d. It is not hard to find an integer d which is relatively prime to $\varphi(N)$, that is $\gcd(\varphi(N), d) = 1$. For example any prime number greater than $\max(p, q)$ will do. Euclidean Algorithm is used to compute e from d and $\varphi(N)$. Since $\gcd(\varphi(N), d) = 1$ coefficients e, s are calculated s.t. $e \cdot d + s \cdot \varphi(N) = 1$. The above equation implies that $e \cdot d \equiv 1 \operatorname{mod}(\varphi(N))$.

ENCRYPTION PROCESS
Alice composes a message in order to transmit it to Bob. The public key e is used in order the message to be encrypted. The encryption process is just a modular exponentiation:

$$x \to x^e \operatorname{mod}(N)$$

It is well-known [6] that exponentiation modulo N is computable with at most multiplications $\operatorname{mod}(N)$ and only 3 words of memory. It is enough to write e in the binary form and compute.

DECRYPTION PROCESS
It is again modular exponentiation:

$$y \to y^d \operatorname{mod}(N) \equiv (x^e)^d \operatorname{mod}(N) \equiv x \operatorname{mod}(N)$$

In order to be able to decrypt we have to have information on d.

ATTACKS
An opponent wants to obtain the original message x from the cryptogram y. He has complete description of the structure of the system, therefore he is familiar with the equation $y \equiv x^e \operatorname{mod}(N)$. Actually oppnent knows y, N and of course e, since e is the public key of the receiver. We state the following well-known attacks:

1. Brute Force. Given $y \equiv x^e \operatorname{mod}(N)$ try all possible keys $0 \leqslant d \leqslant \varphi(N)$ to obtain $x \equiv y^d \operatorname{mod}(N)$. However since $\varphi(N) \propto N$ it is impossible.
2. Finding $\varphi(N)$ and computing d s.t.
$$d \cdot e \equiv 1 \operatorname{mod}(\varphi(N))$$
(Euclidean Algorithm).
3. Finding d directly and computing $x \equiv y^d \operatorname{mod}(N)$.
4. Factoring $N = p \cdot q$, deriving $\varphi(N) = (p-1)(q-1)$ and calculating d as above.

Once the opponent knows d he can calculate $e \cdot d - 1$, which is a multiple of $\varphi(N)$. Miller [14] has shown that N can be factored using any multiple of $\varphi(N)$. Therefore determining d is the same hard as computing $\varphi(N)$ or as factoring N.

1.3 THE PROBLEM

In this paper Neural Network techniques are used in order the Euler φ function to be computed. The inputs are integers N of the form $N = p \cdot q$, where p, q odd primes. The outputs are values of $\varphi(N)$.

1.4 NEURAL NETWORKS

Computing in the past was based on the concept of programmed computing in which algorithms were designed and subsequently implemented using the currently dominant architecture. An alternative view, inspired from biological systems, was proposed with Artificial Neural Networks. As it is well known biological system functionality is based on interconnections of specialized physical cells called neurons. *Artificial Neural Networks* (ANNs) simulating that procedure are a mathematical model with the ability to learn, adapt, generalize or to cluster and organize data. All these operations are based on parallel processing of data and can be quite advantageous and fast, compared to other techniques.

Trying to give a strict definition we could say that an ANN is a structure composed of a number of interconnected units (artificial neurons). Each unit has an input/output characteristic 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 possibly external inputs. Although “hand crafting” of the network is possible, the networks usually develops an overall functionality through one or more forms of training. As it is obvious this definition defines a diverse family of networks. The overall functionality is determined by the network topology, the training algorithm used and the neuron characteristics. One very important type of ANNs are the Feedforward ones. in this kind of networks, all paths lead to one direction. Furthermore the neurons can be disjointedly split in formulations that are called layers. Especially in the so called *Multilayer Feedforward Networks*, the inputs form an input layer, while the output neurons form the output layer. All other neurons are assigned to a number of hidden layers. Each neuron in a layer is fully connected to all other neurons in the next layer. This structure renders it possible to describe networks of this kind with a series of integers that represent the number of neurons at each layer. For example a network with a topology 4-5-5-1 is a network with 4 inputs at the input layer, two hidden layers with 5 neurons each and one output layer with one neuron.

The operation of such networks consists of iterative steps. At the beginning the states of the input layer neurons are assigned to generally real inputs, and the remaining hidden and output layer neurons are passive. In the next step the neurons from the first hidden layer collect and sum their inputs and compute their output. This procedure is propagated to the following layers until the final outputs of the network are computed.

The computational power of neural networks is based on the fact that they can adapt to a specific problem. It has also been proven [5, 22] that standard feedforward networks with only a single hidden layer can aproximate any continuous function uniformly on any compact set and any measurable function to any desired degree of accurancy. This implies that any lack of success in applications must arise from inadequate learning, insufficient number of hidden units or the lack of a deterministic relationship between input and target.

The training process of feedforward networks is based on patterns for which the desired output is known a priori. We can define a training set T of P patterns as follows:

$$T = \left\{ (x_k, d_k) \,\middle|\, \begin{array}{l} x_k = (x_{k1}, ..., x_{kn}) \\ d_k = (d_{k1}, ..., d_{km}) \end{array} \;,\quad k = 1, \ldots, P \right\},$$

where $x_k \in \mathbb{R}^n$ is the input vector of the kth training pattern and $d_k \in \mathbb{R}^m$ is the vector of the desired output of the specific pattern. The aim of adaption is to assign to the free parameters of the network W, values such that the output ofjhghAF the network based on that set of weights will be the desired one (at the beginning the weights are assigned random values), i.e. it holds that:

$$y(W, x_k) = d_k, \qquad k = 1, \ldots, P.$$

The adaption procedure starts by presenting all the patterns to the network and computing a total error function E defined as:

$$E = \sum_{k=1}^{P} (E_k),$$

where E_k is the partial network error with respect of the kth training pattern, and is computed by summing the squared discrepancies between the actual network outputs and the desired values of the kth training pattern, thus:

$$E_k = \frac{1}{2} \sum_{i=1}^{m} \Big(y_i(W, x_k) - d_{ki} \Big)^2 .$$

Each full pass of all the network patterns is called a *training epoch*. If the adaption method succeeds in minimizing the total error function then it is obvious that its aim has been fulfilled. Thus the problem is a non-trivial minimization problem. One very popular method for doing this task is the *Back Propagation* method, which is based on the well-known steepest descent method. The Back Propagation learning process applies small iterative steps which correspond to the training epochs. At each epoch t the method updates the weight values by the relation:

$$w_{ji}^t = w_{ji}^{t-1} - \Delta w_{ji},$$

where w_{ij}^t corresponds to the weight of the connection from neuron i to the non-input neuron j at epoch t while Δw_{ij} corresponds to the increment of the weights. The latter is proportional to the gradient of the error function $E(w)$ at the weights w^{t-1}:

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}} (w^{t-1}),$$

where η, $0 \leqslant \eta \leqslant 1$ is called learning rate and measures the influence of the gradient, computed at the specific epoch, on the general adaption. To compute the gradient $\frac{\partial E}{\partial w_{ji}}(w^{t-1})$ firstly the sum rule is used to simplify the procedure to the computation of the sums of gradients of the partial error functions:

$$\frac{\partial E}{\partial w_{ji}} (w^{t-1}) = \sum_{k=1}^{P} \frac{\partial E_k}{\partial w_{ji}}.$$

At a next step the rule for the composite function derivative can be used:

$$\frac{\partial E_k}{\partial w_{ji}} = \frac{\partial E_k}{\partial y_j} \frac{\partial y_j}{\partial \xi_j} \frac{\partial \xi_j}{\partial w_{ji}},$$

where:

$$\xi_j = \sum_{i \in j_m} w_{ji} y_j,$$

and y_j is the output of neuron j. Obviously the $\frac{\partial y_j}{\partial \xi_j}$ and $\frac{\partial \xi_j}{\partial w_{ji}}$ can be easily computed and they depend on the activation function. The remaining partial derivative $\frac{\partial E_k}{\partial y_j}$ is computed at the output layer and its value is back-propagated through the network to the input neurons (the method derives its name from this procedure).

The whole process is repeated until the overall error value drops below some pre-determined threshold. At this point we say that the network has learned the problem "well enough". Notice that, the network will never exactly learn the ideal function, but rather it will asymptotically approach it. The total number of epochs required can defined as the speed of the method. For more sophisticated techniques see for example [4, 8, 9, 10, 12, 17, 21].

2 EMPIRICAL TESTS

The empirical tests were made using a Neural Network C++ Interface built under Linux Operating system using the g++ compiler. The training methods used were the

- Standard Back Propagation (BP) [19],
- Back Propagation with Variable Stepsize (BPVS) [10],
- Resilient Back Propagation (RPROP) [17] and
- On-Line Adaptive Back Propagation (OABP) [9].

All the methods were extensively tested with a wide range of parameters. None of them however showed significant differences in the results from the others, except of course from standard back propagation, which encountered big difficulties in training most of the times. The only difference was relative to speed measures were RPROP managed to be the fastest, and BPVS in many cases managed to train the network when no other method could to do so.

Regarding the network architecture, as it is well-known, the problem of choosing the "optimal" network architecture for a given problem is very difficult and remains an open problem in our days. This is also the case in our approach. To find a small enough network we proceed as follows. Starting with a network with total number of weighs smaller than the considered data we reduce its architecture as much as it is permitted by the training method used. On the other hand, by decreasing the network we observed that all the considered training methods yielded to suboptimal solutions due to local minima affect. To alleviate this occasional convergence to local minima we applied recent proposed techniques. In particular we applied the *deflection technique* [11] and the *function "stretching"* method [15].

In our approach a large variety of network topologies were used. Different topologies showed great differences in the results, but after an extensive testing we came to the conclusion that networks with 2 hidden layers were much easier to train. Trying to aproximate the RSA mapping ($p \cdot q \rightarrow (p-1)(q-1)$), input patterns were number $N = p \cdot q$ where p and q primes, and target patterns where the $\varphi(N) = (p-1)(q-1)$ numbers. A crucial part of the whole procedure was the normalization step of the training process. This step takes place before training the network, and aims to transform the data in such a way that the network will find it easier to adapt to. So when the numbers that were presented to the network were lower or equal than M, the space $S = [-1, 1]$, was split in M sub-spaces. So numbers in the data were transformed to analogous ones in the space S. At the same time, the network output was transformed to an integer number within $[0, M]$ using the inverse operation.

To test the network performance two different measures were considered. The first measure, which we call it *complete measure* and we denote it by μ_0, indicated the percentage of training data, for which the network was able to compute the exact target value. This was not sufficient enough for the network performance indicator. The fact that the network output was restricted within the range $[-1, 1]$, played a significant role. Very small differences in output, rendered the network unable to find the exact target but to be very close to it. So, using as a second measure, which we call it as *near measure* and we denote it by $\mu_{\pm}$, the percentage of the data for which the difference between desired and actual output does not exceed ± 2 of the real target, gives a better understanding of the network performance. It was also made clear that the second measure showed the real capability of the network, since when the training procedure is continued long enough the first measure kept rising to reach the second one.

It is important to be mentioned that the "near" measure $\mu_{\pm}$ plays a significant role since the computation of the function can be verified, by solving a second degree polynomial equation since $\varphi(N) = N - p - q + 1$.

In Table 1 we can see the result fo networks trained with data lower than 100, or $N, \varphi(N) \leqslant 100$.

Trying larger prime numbers the networks maintain the ability to adapt to the training sets without a significant chage to the network topology. This is illustrated in Tables 2, 3.

Topology	Epochs	μ_0	μ_2
1-2-1	25000	15%	80%
1-5-1	25000	30%	70%
1-2-3-1	25000	100%	100%

Table 1. Results for networks trained with $N = p \cdot q \leqslant 100$.

Topology	Epochs	μ_0	$\mu_{\pm 2}$	$\mu_{\pm 5}$	$\mu_{\pm 10}$	$\mu_{\pm 20}$
1-2-3-1	40000	3%	10%	20%	30%	60%
1-3-5-1	40000	5%	20%	40%	60%	80%
1-5-6-1	40000	5%	30%	55%	65%	100%

Table 2. Results for networks trained with $N = p \cdot q \leqslant 1000$.

Topology	Epochs	μ_0	$\mu_{\pm 2}$	$\mu_{\pm 5}$	$\mu_{\pm 10}$	$\mu_{\pm 20}$
1-3-5-1	80000	2%	10%	25%	50%	70%
1-5-5-1	80000	3%	15%	35%	65%	90%
1-7-8-1	80000	5%	15%	35%	70%	100%

Table 3. Results for networks trained with $N = p \cdot q \leqslant 10000$.

Keaping apart from the training process a portion of the data set (33%) and using it as a Test set, gives us the results exhibited in Tables 4,5. As, it is obvious the networks are able not only, to adapt to the train data but to achieve very good results to the test sets.

Topology	Epochs	μ_0	$\mu_{\pm 2}$	$\mu_{\pm 5}$	$\mu_{\pm 10}$	$\mu_{\pm 20}$	TYPE
1-3-5-1	60000	5%	20%	40%	60%	80%	Train Set
1-3-5-1	60000	5%	20%	40%	50%	80%	Test Set
1-7-8-1	50000	6%	20%	50%	70%	100%	Train Set
1-7-8-1	50000	5%	20%	50%	70%	90%	Test Set

Table 4. Results for networks trained with a 66% of the data set with $N = p \cdot q \leqslant 1000$.

Topology	Epochs	μ_0	$\mu_{\pm 2}$	$\mu_{\pm 5}$	$\mu_{\pm 10}$	$\mu_{\pm 20}$	TYPE
1-5-5-1	80000	3%	15%	35%	65%	90%	Train Set
1-5-5-1	80000	5%	20%	40%	60%	90%	Test Set

Table 5. Results for networks trained with a 66% of the data set with $N = p \cdot q \leqslant 10000$.

3 CONCLUDING REMARKS

In the present paper, a first study of the neural network approach has been attempted to encounter the RSA Problem. Our experience with this attempt is that it is possible to train feedforward neural networks to tackle this very difficult task. Naturally, this being our first attempt towards this direction, we strongly believe that there are numerous issues remaining in order to obtain a comprehensive view of the ability of Neural Networks to simulate the suggested functions.

Very small prime numbers were selected in order to have an extensive study related to networks' architecture and their ability for various efficient training methods. In particular we have used various training methods such as the Standard Back Propagation (BP) [19], the Back Propagation with Variable Stepsize (BPVS) [10], the Resilient Back Propagation (RPROP) [17] and the On-Line Adaptive Back Propagation (OABP) [9]. All these methods have been extensively tested with a wide range of parameters. Thus we have concluded that the training method does not play a significant role in tackling the particular problem. On the other hand a crucial role is being played by the network architecture and the normalization portion of the training algorithm used. We have succeeded in reducing the network architecture as much as it was permitted by each training method considered, by applying two recent proposed techniques the deflection [11] and the function "stretching" one [15].

In general, the problem can be considered solved if the measure μ_0 is 100% and the architectural topology is small enough (its complexity is a polynomial of $\log(N)$). On the other hand, measures like $\mu_{\pm}$ seem promising in the sense that although the exact target might not be accomplished, the output of the networks is indeed close to that. Thus, with a predetermined number of trial and error procedures the problem can be resolved.

Here we have considered only artificial feedforward neural networks. In a future correspondence we intent to apply various other networks and learning techniques such as Learning Rate Adaptation methods [12, 21], Discrete neural networks [1], Self-Organized Map algorithm [8], Recurrent networks and Radial Basis Function networks [4].

In conclusion we think that the neural network approach in problems related to RSA cryptosystem is promising although many problems have to be resolved and many further work needs to be done.

References

[1] B. Boutsinas and M.N. Vrahatis, Artificial Nonmonotonic Neural Networks, *Artif. Intelligence*, 132, 2001,1–38.

[2] W. Diffie and M. Hellman, New Directions in Cryptography, *IEEE Trans. I.T.*, 22, 1976, 644–654.

[3] T. El Gamal, A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms, *IEEE Trans. I.T.*, 31, 1985, 469–472.

[4] S. Haykin, *Neural Networks*, (New York: Macmillan College Publishing Company, 1999).

[5] K. Hornik, Multilayer Feedforward Networks are Universal Approximators, *Neural Networks*, 2, 1989, 359–366.

[6] D.E. Knuth, *The art of Computer Programming, Vol. II: Seminumerical Algorithms*, p. 399, (Reading, Massachusetts: Addison-Wesley, 1969).

[7] N. Koblitz, Elliptic Curve Cryptosystems, *Math. Comp.*, 48, 1997, 203–209.

[8] T. Kohonen, *Self–Organized Maps*, (Berlin: Springer, 1997).

[9] G.D. Magoulas, V.P. Plagianakos and M.N. Vrahatis, Adaptive Stepsize Algorithms for On-line Training of Neural Networks, *Nonlinear Analysis T.M.A.*, 47(5), 2001, 3425–3430.

[10] G.D. Magoulas, M.N. Vrahatis and G.S. Androulakis, Effective Backpropagation Training with Variable Stepsize, *Neural Networks*, 10(1), 1997, 69–82.

[11] G.D. Magoulas, M.N. Vrahatis and G.S. Androulakis, On the Alleviation of the Problem of Local Minima in Backpropagation, *Nonlinear Analysis T.M.A.*, 30(7), 1997, 4545–4550.

[12] G.D. Magoulas, M.N. Vrahatis and G.S. Androulakis, Increasing the Convergence Rate of the Error Backpropagation Algorithm by Learning Rate Adaptation Methods, *Neural Computation*, 11(7), 1999, 1769–1796.

[13] R.C. Merkle and M.E. Hellman, Hiding Information and Signatures in Trapdoor Knapsacks, *IEEE Trans. I.T.*, 24, 1978, 525–530.

[14] G.L Miller, Rieman's Hypothesis and Tests for Primality, *Journal of Computer and System Sciences*, 13, 1976, 300–317.

[15] K.E. Parsopoulos, V.P. Plagianakos, G.D. Magoulas and M.N. Vrahatis, Objective Function "Stretching" to Alleviate Convergence to Local Minima, *Nonlinear Analysis T.M.A.*, 47(5), 2001, 3419–3424.

[16] S.C. Pohlig and M. Hellman, An Improved Algorithm for Computing Logarithms over $GF(p)$ and its Cryptographic Significance, *IEEE Trans. I.T.*, 24, 1978, 106–110.

[17] M. Riedmiller and H. Braun, A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm, *Proceedings of the IEEE International Conference on Neural Networks*, San Francisco, CA, 1993, 586–591.

[18] R. Rivest, A. Shamir and L. Adlemann, A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, *Commun. ACM*, 21, 1978, 120–126.

[19] D.E. Rumelhart, G.E. Hinton and R.J. Williams, Learning Internal Representations by Error Propagation, In D.E. Rumelhart and J.L. McClelland (Eds.) *Parallel distributed processing: Explorations in the microstructure of cognition*, Cambridge, MA, MIT Press, 1, 1986, 318–362.

[20] R. Solovay and V. Strassen, A Fast Monte Carlo Test for Primality, *SIAM J. Comput.*, 6(1), 1977, 84–85.

[21] M.N. Vrahatis, G.S. Androulakis, J.N. Lambrinos and G.D. Magoulas, A Class of Gradient Unconstrained Minimization Algorithms with Adaptive Stepsize, *J. Comput. Appl. Math.*, 114(2), 2000, 367–386.

[22] H. White, Connectionist Nonparametric Regression: Multilayer Feedforward Networks Can Learn Arbitary Mappings, Neural Networks, 2, 1989, 359–366.