Neural Networks Following a Binary Approach Applied to the Integer Prime-Factorization Problem

Boris Jansen Kenji Nakayama
Graduate School of Natural Science and Technology
Kanazawa University
Kakuma-machi, Kanazawa, 920-1192, JAPAN
E-mail: boris@leo.ec.t.kanazawa-u.ac.jp, nakayama@t.kanazawa-u.ac.jp

***Abstract*— Nowadays, the integer prime-factorization problem finds its application often in modern cryptography. Artificial Neural Networks (ANNs) have been applied to the integer prime-factorization problem. A composed number N is applied to the ANNs, and one of its prime factors p is obtained as the output. Previously, neural networks dealing with the input and output data in a decimal format have been proposed. However, accuracy is not sufficient. In this paper, a neural network following a binary approach is proposed. The input N as well as the desired output p were expressed in a binary form. The proposed neural network is expected to be more stable, i.e. less sensitive to small errors in the network outputs. Simulations have been performed and the results are compared with the results reported in the previous study. The number of required search times for the true prime number can be well reduced. Furthermore, the probability density function of the training patterns is investigated and the need for different data creation and/or selection techniques is shown.**

I. Introduction

Since ancient times, mathematicians have been fascinated by the integer prime-factorization problem, also known as the prime decomposition problem. In mathematics, more specifically in number theory, the fundamental theorem of arithmetic [1] states that every positive integer greater than 1 can be written as a product of prime numbers in only one unique way[1]. Although it is fairly easy to compute the product of a number of given primes, it appears to be very difficult to decompose a given product into its unique primes. Up to present, there exist no known deterministic or randomized polynomial-time algorithm for finding the factorization of a given composed number [2].

Nowadays, the integer factorization problem finds its application often in modern cryptography. The computational security of many cryptographic systems, including the well-known and widely applied RSA public-key algorithm, relies on the difficulty of factoring large integers. If a fast method for solving the integer factorization problem would be found, then several important cryptosystems would become insecure [2] [3]. Therefore, studies on factoring large integers are very important for the development of more secure cryptosystems.

[1]Ignoring the ordering of the factors

In this paper, first the ability of artificial neural networks (ANNs) in an attempt to solve the integer factorization problem is investigated. A binary approach is proposed, expected to be more stable, i.e. less sensitive to small errors in the network output compared to the previously used decimal approach. Simulations have been performed and results are compared with results reported in the previous independent study. Moreover, instances of larger composites N and consequently larger primes p are investigated. Finally, the probability density of the training patterns is examined and the need for a different way to create or select the training data, in order to solve the integer factorization problem for numbers composed of larger primes, is shown.

II. Problem Description

In this paper, artificial neural networks are applied in order to factor integers N, which are the product of two odd[2] primes p and q, i.e. $N = p \cdot q$. Throughout the article we will assume $p \leq q$. To factor an integer composed of two prime numbers, it is enough to find one of the two primes. The other prime can be obtained though a single division. Here, given N we focus on obtaining the smaller prime p of the two. In other words, we try to approximate the mapping $N \rightarrow p$.

Although it hasn't been applied in our research, it should be mentioned for a later reference that it has been proved [4] that N can be factored using any multiple of $\varphi(N) = (p-1)(q-1)$.

III. Neural Network for Factorization

A. Network Design

The adopted neural networks used in our experiments are multilayer feed-forward networks with a single hidden layer. Although we have experimented with various multilayer feed-forward neural networks, the best results were obtained with networks having only a single hidden layer.

Networks constructed with neurons having a sinusoidal activation function in the hidden layer performed better compared

[2]We have chosen to consider only numbers composed of odd primes for two reasons. First of all, this allowed us to make a straightforward comparison with the results reported in the previous study. Secondly, the case where N is composed of an even prime, resulting in N to be an even number, is trivial anyway, especially in a binary format.

0-7803-9048-2/05/$20.00 ©2005 IEEE

to networks composed of neurons having a hyperbolic tangent activation function in the hidden layer. The network input and output data was expressed in a binary format. Therefore, neurons having a hyperbolic tangent activation function were selected to be used in the output layer.

By setting the binary output target values to ± 0.7 instead of the asymptotic values ± 1.0 of the hyperbolic tangent functions used in the output layer, we removed the tendency of the network driving its free parameters to infinity [5] and increased the overall performance.

The Resilient Back-Propagation (RPROP) [6] learning algorithm was used in order to train the different networks. Its parameters were set to its default, previously proposed values. Standard Back Propagation (BP) [7] tested with a broad range of different parameter settings including some of its variants, e.g. BP with a momentum term and BP operating in batch mode, showed to be incapable of obtaining any satisfactory results.

Weight initialization took place on a so-called *fan-in* basis [5]. All connection weights and biases were uniformly distributed inside the range $[-\frac{2.4}{F_i}, \frac{2.4}{F_i}]$, where F_i is the *fan-in*, i.e. the total number of the inputs of the neuron i in the network. This weight initialization method resulted in a slightly better performance of the network compared to networks using a random weight initialization method where weights were randomly initialised within the interval $[-0.2, 0.2]$ or $[-1.0, 1.0]$.

We have experimented with single neural networks (SNNs) as well as multi-model neural networks (MNNs), where several neural networks do the precision in parallel and the final decision is made by averaging over all outputs. Here, multi-model networks are composed of three independent neural networks and its outputs are decided by taking the average of the outputs of the three single neural networks.

B. Training and Test Data

This being our first study related to the integer factorization problem, we dealt with relatively small composed integers, restricted by an upper bound, and consequently small prime numbers. This enabled us to generate and use all possible patterns $N \rightarrow p$ within the limitation $N < M$. A certain percentage of the data set was used for training, while the remaining part was used to test the performance of the network after learning had converged.

Both the input N and the desired output p were represented in a binary form. During evaluation, any output of an output neuron greater than or equal to zero was considered as an upper bit, while any output less than zero was considered as a lower bit.

IV. Performance Measure

A. Measures for Bit Errors

To evaluate the network performance, two different measures are considered. The first measure, which we call the *binary complete measure* and denoted by β_0, indicates the percentage of the data for which the network produces the exact desired output. The second measure, which we call the *binary near measure* and denoted by β_i where $i \geq 1$, indicates the percentage of the data for which at most i bits are incorrect in the output produced by the network.

We felt the need for this second measure. Whenever the network is unable to produce the exact desired output for a certain input, it does not necessary mean that the network output is useless. If the network output contains just a small number of bit errors, the exact target value can still be found within a predetermined number of trial and error procedures. It is very easy to verify if a certain value is the target value, i.e. a factor of N, because a division of N by the number should result in another whole number with no remainders. Therefore, this second measure, which gives an indication of the distance to the exact desired output, provides a better understanding of the real network performance rather than relying on the binary complete measure alone.

B. Number of Searches for True Prime Number

Assuming k bits are incorrect in a certain output, then by trying all combinations of "flipping" k bits in the incorrect output, the exact target output is sure to be found. The number of existing combinations can be given by:

$$c(k) = \binom{b}{k} = \frac{b!}{k!(b-k)!} \tag{1}$$

where b is the number of output bits. Therefore, the near binary measure β_i for $i \geq 1$ indicates the percentage of data for which the correct output can be found within S_{β_i} trial and error procedures, defined by:

$$S_{\beta_i} = \sum_{l=1}^{i} c(l) \tag{2}$$

However, it can be known that the least significant bit of p will always be equal to one, because N is a product of two *odd* primes. Also in all our performed experiments the ANNs always correctly output this least significant bit. Taking this knowledge into consideration, $c(k)$ given in Eq. (1) can be replaced by:

$$c'(k) = \binom{b-1}{k} = \frac{(b-1)!}{k!(b-1-k)!} \tag{3}$$

Analogously, the maximum number of required trial and error procedures in order to obtain the correct output for data within β_i can be defined by:

$$S'_{\beta_i} = \sum_{l=1}^{i} c'(l) \tag{4}$$

V. Simulations

The neural networks used in our simulations have been developed using the Java Object-Oriented Neural Engine (JOONE), an open source neural net framework implemented in Java [8].

Results are reported for single neural networks (SNNs) and multi-modal neural networks (MNNs) consisting of three independent neural networks. All presented results are averaged over 10 independent runs for each problem instance.

Tables I and II show the results of networks trained on the problem instances $N < 1000$ and $N < 10000$, respectively. Here, 66% of the data set was used for training. This allowed us to make a straightforward comparison with the results reported in the previous study [9].

TABLE I
RESULTS FOR NETWORKS TRAINED WITH 66% OF THE DATA SET WITH $N = p \cdot q \leq 1000$

Topology	Epochs	β_0	β_1	β_2	β_3	Data
10-6-5	1000	61%	93%	100%	100%	Train
SNN		24%	48%	84%	100%	Test
10-6-5	3000	68%	95%	100%	100%	Train
MNN-3	$(3 \cdot 1000)$	26%	50%	89%	100%	Test

TABLE II
RESULTS FOR NETWORKS TRAINED WITH 66% OF THE DATA SET WITH $N = p \cdot q \leq 10000$

Topology	Epochs	β_0	β_1	β_2	β_3	Data
14-20-7	5000	64%	81%	93%	98%	Train
SNN		53%	69%	83%	94%	Test
14-20-7	15000	72%	85%	95%	99%	Train
MNN-3	$(3 \cdot 5000)$	62%	73%	85%	94%	Test

For larger problem instances where $N \leq 100000$, the neural networks are still able to obtain satisfactory results as shown in Table III.

TABLE III
RESULTS FOR NETWORKS TRAINED WITH 66% OF THE DATA SET WITH $N = p \cdot q \leq 100000$

Topology	Epochs	β_0	β_1	β_2	β_3	β_4	Data
17-50-9	10000	49%	63%	77%	89%	97%	Train
SNN		45%	58%	72%	86%	95%	Test
17-50-9	30000	55%	66%	79%	90%	97%	Train
MNN-3	$(3 \cdot 10000)$	51%	61%	73%	86%	96%	Test

Trying smaller training sets, the networks maintain the ability to adapt to the training data and generalize on the test data. This is illustrated in Table IV where neural networks were trained with 33% of the data set, while the remaining part was used for validation.

Finally, the results for the problem instance $N < 1000000$, where only 10% of the data set was used for training, are shown in Table V.

TABLE IV
RESULTS FOR NETWORKS TRAINED WITH 33% OF THE DATA SET WITH $N = p \cdot q \leq 100000$

Topology	Epochs	β_0	β_1	β_2	β_3	β_4	Data
17-50-9	10000	51%	66%	81%	92%	98%	Train
SNN		44%	57%	72%	86%	95%	Test
17-50-9	30000	59%	72%	84%	94%	99%	Train
MNN-3	$(3 \cdot 10000)$	52%	62%	74%	86%	95%	Test

TABLE V
RESULTS FOR NETWORKS TRAINED WITH 10% OF THE DATA SET WITH $N = p \cdot q \leq 1000000$

Topology	Epochs	β_0	β_1	β_2	β_3	β_4	Data
20-100-10	15000	33%	50%	68%	84%	93%	Train
SNN		28%	42%	60%	77%	89%	Test
20-100-10	45000	41%	54%	71%	85%	94%	Train
MNN-3	$(3 \cdot 15000)$	37%	47%	62%	78%	90%	Test

VI. COMPARISON BETWEEN PREVIOUS APPROACH AND CURRENT STUDY

A. Differences in Approach

Previously Meletiou et al. investigated the ability of ANNs to factor integers and reported promising results [9]. Here we address the same problem, however two major differences exist between their study and ours. The differences occur in the approach of solving the integer factorization problem by ANNs, more specifically the differences are in the representation of the data and the function to approximate.

Meletiou et al. dealt with the data in decimal form, where the input data $N < M$ was normalized to the space $S = [-1, 1]$ by splitting it up in M sub-spaces. The network output was transformed again into an integer number within the interval $[0, M]$ using the inverse operation. According to their paper, this normalization step played a crucial role in the whole procedure aiming to transform the data in such a way that the network will find it easier to adapt to. However, in our opinion this normalization step might lead to problems as the problem space, i.e. the upper bound M of N starts to grow. Whenever larger problem instances are considered, the network output range, restricted to $[-1, 1]$, will be divided into more, but smaller sub-spaces during the normalization step. Then, even very small differences in the network output might lead to values far removed from the desired target output after denormalization.

This problem is already shortly addressed in their work by introducing a second performance measure besides the *complete measure*. The *complete measure*, denoted by μ_0, indicates the percentage of the data for which the network is able to compute the exact target value. The second measure, called the *near measure* and denoted by $\mu_{\pm k}$, indicates the percentage of the data for which the difference between the

desired and actual output does not exceed $\pm k$ of the real target. However, introducing this second measure does not solve the problem and is more an admittance of the existence of the problem. As long as the network output is within a small distance from the desired output, this approach is acceptable. However, we believe that as the problem size will start to grow it might be very difficult to maintain this approach.

Therefore, in our research we have chosen to deal with the data in a binary form, expecting that this approach leads to a more stable network. It is our assumption that by applying a binary approach, the danger where small differences in the network output might lead to values far removed from the desired output is greatly reduced, because the restricted output $[-1.0, 1.0]$ of the network output neurons is divided into only two large sub-spaces opposite to M very small sub-spaces as is in the decimal approach.

From a different point of view, applying ANNs dealing with the data in a binary form, the problem can be seen as a classification problem instead of a function approximation problem, where every output neuron has to decide to which class the input belongs.

The second difference is related to the function to approximate. Meletiou et al. focused on approximating the mapping $N \rightarrow \varphi(N)$, while we tried to approximate the mapping $N \rightarrow p$. Both realizations of these mappings enable one to factor integers. However, the possible output range for $N \rightarrow p$ is much smaller than for $N \rightarrow \varphi(N)$ when N is bounded by a certain upper value. Secondly, it takes less work to verify the validity of the network output and to compute the complete factorization of N after obtaining p than after obtaining $\varphi(N)$.

One more difference worth to mention is that Meletiou et al. used two interesting, but hard to use non-straightforward techniques, the so-called *deflection technique* [10] and *function "stretching"* [11] method, to overcome convergence to local minima. We were able to obtain satisfactory results without the use of these kinds of techniques.

B. Comparison of Results

In Tables VI and VII the results reported by Meletiou et al. are shown for the problem instances $N \leq 1000$ and $N \leq 10000$ respectively, where 66% of the data set was used for training.

TABLE VI
RESULTS REPORTED BY MELETIOU ET AL. FOR NETWORKS TRAINED WITH 66% OF THE DATA SET WITH $N = p \cdot q \leq 1000$

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

Comparing those results with the results reported in Tables II and II, it can be easily noticed that our proposed neural networks outperform the networks proposed in the study of Meletiou et al.

TABLE VII
RESULTS REPORTED BY MELETIOU ET AL. FOR NETWORKS TRAINED WITH 66% OF THE DATA SET WITH $N = p \cdot q \leq 10000$

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

First of all, by observing the results for $N \leq 10000$, the values for the (binary) complete measure, that is 64% and 53% for the training data and the test data respectively in case of single neural networks are much higher than the percentages 3% and 5% for the training data and test data respectively reported in the study of Meletiou et al.

Moreover, in an attempt to make a fair comparison of the overall performance of the two different networks, the average number of required trial and error procedures in order to obtain the true prime number for the reported test data will be taking into consideration. Regarding the test data for $N \leq 10000$, Meletiou et al. reported for the successive measures $\mu_{\pm 0}$ and $\mu_{\pm 2}$ 5% and 20% respectively. This means that $20\% - 5\% = 15\%$ of the data is within $\mu_{\pm 2}$ and not within $\mu_{\pm 0}$. Furthermore, for that 15% of the data the true prime is sure to be found within 4 trial and error procedures[3]. Therefore, we will assume that it takes 2 trial and error procedures on average. By accumulating the average number of trial and error procedures for each data set proportional to their size, it is possible to calculate the average number of trail and error procedures for all the reported data:

$$\frac{15 \cdot \frac{4}{2} + 20 \cdot \frac{4+10}{2} + 20 \cdot \frac{10+20}{2} + 30 \cdot \frac{20+40}{2}}{90} = 15.2$$

A similar calculation can be performed on the data reported for our simulations. Considering the test data $N \leq 10000$, we have reported 53% and 69% for the successive binary measures β_0 and β_1 respectively. Therefore, 69% - 53% = 16% of the test data is within β_1 and not within β_0, and for that 16% at most 6 trial and error procedures are required in order to find the true prime according to Eq. 4. Therefore we will assume that it takes half the number, i.e. 3 trial and error procedures on average. The average number of trial and error procedures for all the reported data can be given by:

$$\frac{16 \cdot \frac{6}{2} + 14 \cdot \frac{6+21}{2} + 11 \cdot \frac{21+41}{2}}{94} = 6.1$$

Although these average values for the required number of trial and error procedures have some inaccuracy, because not all the test data (90% and 94%) is used in the calculations, in

[3]The data is within $\mu_{\pm 2}$, therefore the correct output can be obtained within the range of 2 above and 2 below of the actual output resulting in a maximum of 4 trail and error procedures

our opinion these values still give a very good understanding of the performance gain that can be achieved by the newly proposed method. We believe that these results support the choice for a binary approach.

Moreover, a positive side effect is that the number of training epochs in our experiments, 5000 and 15000 for SNNs and MNNs respectively, is much lower compared to 50000-80000 training epochs reported in the study of Meletiou et al., who also applied among others the RPROP learning algorithm.

VII. Probability Density

Whenever N is restricted by a certain upper bound, i.e. $N < M$ and we consider all possible patterns $N \rightarrow p$ within that limitation, then the lower the value of the smaller prime p the higher the density of patterns. The density of patterns w.r.t. p for $N \leq 10000$ is shown in Fig. 1.

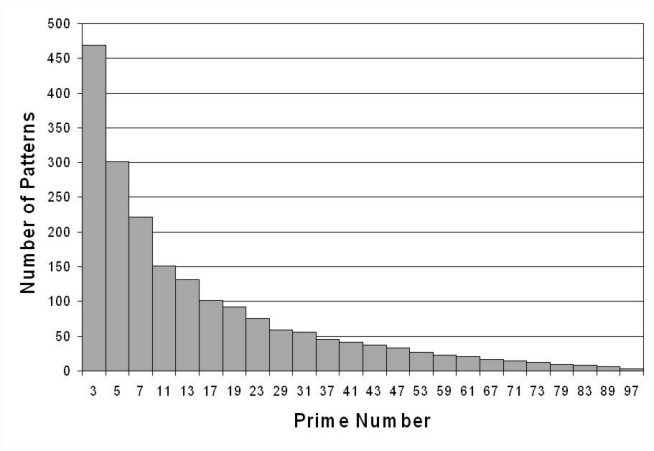

Fig. 1. Pattern density for $N \leq 10000$

After observing the results of our experiments more in detail, we noticed that the networks performed very well on the data patterns having a high density w.r.t. p and that the network performance gradually decreases as the density of the (training) patterns w.r.t. p decreases. This can be seen in Fig. 2, which shows the percentages of bit errors for all patterns w.r.t. p.

This effect, where the performance of the network is in accordance with the density of the training data, is a natural occurrence. However, in some cases, for example in case of an attack on the RSA cryptosystem where keys are usually created using hard, i.e. large prime numbers, it is wishful to obtain a similar or even better performance for N composed of large prime numbers compared to N composed of smaller prime numbers.

How to extend the ability of ANNs in order to solve the integer factorization problem for sparse N composed of larger primes remains an open problem. More research, such as research to different training data creation and/or selection techniques in order to address this problem is required.

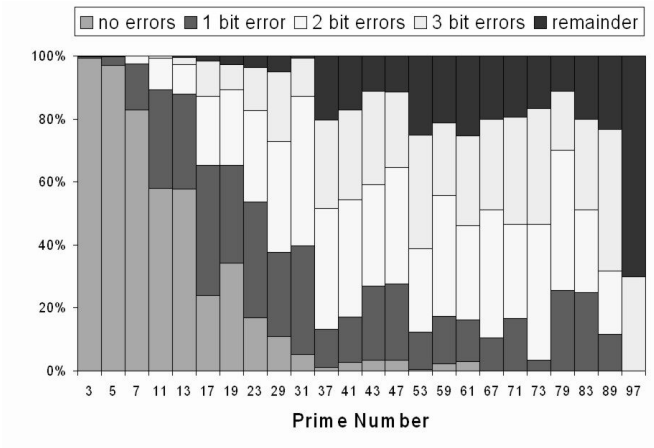

Fig. 2. Percentages of bit errors for $N \leq 10000$

VIII. Conclusions

The ability of artificial neural networks for solving the integer factorization problem has been studied. The integer factorization problem is a very difficult problem and the mapping to approximate, be it $N \rightarrow p$ or $N \rightarrow \varphi(N)$ is a function with a very spiky nature. Nevertheless, artificial neural networks have shown to be able to solve this problem with some accuracy for relatively small N. In this paper, the multilayer neural network has been optimized, proposing a binary expression of the input and the output data and focusing on p, which is the smaller prime of N, to be obtained as the network output. Simulation results have demonstrated usefulness of the proposed approach. Probabilities for the solutions without any bit error and the number of required searches to obtain the true prime number are greatly improved compared to existing methods.

In future research we intent to apply ANNs to larger, more realistic problem instances and to investigate various other network models and related techniques. However, at first future work will be directed towards research addressing the problem as outlined in Sec. VII where we will search for a solution to extend the ability of ANNs for solving the integer factorization problem for sparse N composed of larger prime numbers.

References

[1] R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathematics: A Foundation for Computer Science*, 2nd ed. Massachusetts: Addison-Wesley, 1994.

[2] R. P. Brent, "Recent progress and prospects for integer factorisation algorithms," *Lecture Notes in Computer Science*, vol. 1858, pp. 3+, 2000.

[3] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[4] G. L. Miller, "Riemann's hypothesis and tests for primality," *Journal of Computer and System Sciences*, pp. 300–317, 1976.

[5] S. Haykin, *Neural Networks: A Comprehensive Foundation*. New Jersey: Prentice-Hall, 1994.

[6] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: The RPROP algorithm," in *Proc. of the IEEE International Conference on Neural Networks*, San Francisco, CA, 1993, pp. 586–591.

[7] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," *Parallel Distributed Processing: Exploration in the Microstructure of Cognition*, vol. 1, pp. 318–362, 1986.

[8] P. Marrone. (2005) Java object-oriented neural engine (JOONE). [Online]. Available: http://www.jooneworld.com

[9] G. C. Meletiou, D. K. Tasoulis, and M. N. Vrahatis, "A first study of the neural network approach in the RSA cryptosystem," in 6^{th} *IASTED International Conference Artificial Intelligence and Soft Computing ASC 2002*, Banff, Canada, July 2002, pp. 483–488.

[10] 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.*, vol. 30, no. 7, pp. 4545–4550, 1997.

[11] 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.*, vol. 47, no. 5, pp. 3419–3424, 2001.