Integer Prime Factorization with Deep Learning

Begaiym Murat
Suleyman Demirel University,
Kaskelen, Kazakhstan
begaiym.murat@nu.edu.kz

Shirali Kadyrov
Suleyman Demirel University,
Kaskelen, Kazakhstan
shirali.kadyrov@sdu.edu.kz

Ryszhan Tabarek
Suleyman Demirel University,
Kaskelen, Kazakhstan
181107007@stu.sdu.edu.kz

***Abstract*—Prime factor decomposition is a method that is used in number theory and in cryptography, as well. The security of the message depends on the difficulty of factorization. In other words, to hack the RSA system, factorization of N is needed, where N is a product of two prime (generally large) numbers. This paper analyzes the approaches which are already used to solve the problem, and proposes a new method which is expected to increase the efficiency of prime number factorization with the help of neural networks. The results in this paper can be used to develop and improve the security of cryptosystems.**

Keywords—neural network; prime factorization; LSTM; binary expansion; semiprimes

I. Introduction

Prime factorization, also known as prime number decomposition, is a technique, widely used in mathematics and other sciences as well. According to the fundamental theorem of arithmetic, every positive integer N (N > 1) can be factorized into a product of prime numbers. So far, there is no efficient method found to factorize large numbers, neglecting quantum computations [1]. An attempt was made in 2009, where researchers factorized 232-digit numbers which took hundreds of machines and a couple of years to solve the problem [2]. This leads to the assumption that factoring bigger numbers might take much longer.

Cryptography is an area of study that considers various techniques for secure communication. One of the well-known public key cryptosystems is the Rivest, Shamir and Adleman (RSA) cryptosystem, which is based on semiprime factorization. A natural number N is said to be semiprime if it is the product of two distinct prime numbers. RSA happens to be one of the first public key cryptosystems, which makes the encryption key public, but keeps the decryption key private. This type of system is also called a non-symmetric system, due to the fact that the keys required for encryption and decryption are different. For secure communication with RSA, the receiver generates two large primes numbers p and q, usually taken to be over 400 digits each. These numbers are kept secret, while their product N=pq is made publicly available. So, anyone wishing to send a secure message to the receiver makes use of N to encrypt the message and it can only be decrypted if two primes p and q are known. In particular, the receiver can read the message. The security of a message encrypted by the RSA cryptosystem directly depends on the difficulty of factorizing the given large (in practice is taken to be of 500 digit) natural number to its primes. The paper of M. Mumtaz and L. Ping describes recent attacks on the RSA and gives overview on the challenges [3].

The fundamental of algorithms is based on the factorization problem; therefore, the method plays a significant role in this area. Solving prime factorization problems will make a contribution not only to information security, but also will give benefit in other sciences. For instance, the interest in this problem is high in chemistry. Recently, researchers modelled an algorithm on optimization of the surface energy potential of molecules in a crystal based on computational chemistry. Even though the cases were not realistic, the results show that there is a potential of molecular geometry optimization algorithm in the problem of prime factorization [4]. One more related paper was published recently in 2019. It is about factorizing values by stochastic magnetic tunnel junctions. The results show integer factorization up to 945 using probabilistic computer utilizing probabilistic bits that was implemented with stochastic nanomagnetic devices in a neural-network-inspired electrical circuit operating at room temperature [5].

II. Literature Review

The function N = p×q is known as a one-way function. Evaluating N from the given p and q is not hard, but the inverse of it, that is getting p and q from the given N is much harder. In addition to that, factorizing N is accepted to be as hard as finding out the decryption key in RSA [6]. The first research on factorizing N using neural networks was made in 2002 by G.C. Meletiou et al. The training methods that were used are Standard Back Propagation, Back Propagation with Variable Stepsize, the Resilient Back Propagation and the On-Line Adaptive Back Propagation. They conclude that the training methods do not make a big difference for the results. The actual important factors that affect the efficiency are the approximation methods and network architecture [6].

Another research that made a contribution on solving this problem was made in 2005 [7]. A binary approach together with neural networks was used in this paper. The results showed that this method was able to improve the performance of factorization; so, the efficiency of this problem is a subject to change. The different approximation was used, that is mapping N→p, while in the first paper it was N→φ(N). Both of these mappings can be used for approximation, but the recent one appeared to be more efficient because of its output range and decrease in time needed for validity. One more difference was in the data structure. In that paper, a different structure from decimal form was used, which is a binary form. The assumption that was made to do that was that the binary approach will give a more stable network [7]. The neural network part of these papers is using the feed-forward network, where there is one layer for input, one for the output, and others between them are hidden layers. According to the remarks, using one or two hidden layers gives better results [6, 7]. The methods which were used for training the network include the Resilient Back Propagation, Standard Back Propagation, Back Propagation with Variable Stepsize and On-Line Adaptive Back Propagation. Although there was progress in factorizing not so large numbers, decomposition of larger integers into primes is still far from being efficient.

Our goal in this work is to combine Recurrent Neural Networks with Artificial Neural Networks to improve the deep learning based prime factorization algorithms.

In the next section we introduce the dataset processing and the proposed novel algorithm. In section 4, we provide the results of our experiments. Finally, we end with discussion on limitations and possible future work.

III. METHODOLOGY

A. *Dataset generation*

The dataset needed to train the neural networks was generated using the following algorithms: finding semiprimes of the given number, converting a number from decimal to binary representation and finding the smallest prime factor of a number.

We studied four kinds of dataset, namely the dataset of all semiprimes up to 1,000, 10,000, 100,000, and 1,000,000 respectively. The input values are semiprimes converted into binary vectors and output values (labels) are the smallest of the two prime divisors again converted into binary vectors.

The first part of coding starts with setting the upperbound, then all the numbers from 0 to the value of upperbound are checked for the semiprime property. All the semiprimes are collected and then are converted to binary format and saved in a separate file. When training the neural network, this data is split into train and test parts.

Then comes the part when the neural network finishes training. It provides bunch of digits between zero and one. The smaller prime factors of the semiprimes are converted to binary format and got ready beforehands. In order to check how well the data is trained, the output numbers are rounded first and then compared to the expected values.

B. *The proposed algorithm*

In this section we introduce our proposed model architecture in dealing with prime factorization.

The model should take as an input variable a semiprime n which is the product of two primes p, q with p < q and it needs to output the prime factor p. As mentioned above, the networks are expected to work well when these integers n and p are converted into binary expansion. Each binary expansion is regarded as a vector of dimension equal to the number of binary digits. Clearly Artificial Neural Networks are effective in dealing with vectors as is done in [6, 7]. One shortcoming of ANNs is that the model regards each entry of the vector as independent of the other entries to begin with. However, clearly our binary expansion is an ordered sequence and hence it can be regarded as meaningful words. Thus, it makes sense to include in the model Recurrent Neural Networks (RNN) as a common tool in natural language processing. To this end, we use Long Short-Term Memory (LSTM) networks which can learn the dependencies between the entries of the input variables. Detailed information about LSTM networks can be found in the paper of A. Sherstinsky [8]. Our proposed model summary is provided in Table 1 with 2,127,262 trainable parameters and 2,248 non-trainable parameters.

TABLE 1. MODEL SUMMARY

Layer (type)	Output Shape	Param #
LSTM	(None, 1, 128)	76288
Batch Normalization	(None, 1, 128)	512
Dropout	(None, 1, 128)	0
LSTM	(None, 1, 256)	394240
Batch Normalization	(None, 1, 256)	1024
Dropout	(None, 1, 256)	0
LSTM	(None, 512)	1574912
Batch Normalization	(None, 512)	2048
Dropout	(None, 512)	0

Dense	(None, 128)	65664
Batch Normalization	(None, 128)	512
Dropout	(None, 128)	0
Dense	(None, 100)	12900
Batch Normalization	(None, 100)	400
Dropout	(None, 100)	0
Dense	(None, 10)	1010

To be more precise, the proposed network topology consists of three LSTM layers with output units 128, 256, and 512 respectively. These RNN layers are then followed by two hidden dense layers with 128 and 100 output units and ReLU activations. To avoid overfitting, each layer is followed by batch normalization and dropouts with probabilities 30% except the last one has the probability of 40%. Finally, sigmoid activation function is used in the output layer. The model is then compiled with Binary Cross Entropy loss function

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i log(p(y_i)) + (1-y_i)\, log(1-p(y_i)),$$

RMSprop optimizer and Area Under the Curve (AUC) accuracy metric.

RMSprop is an extension of the Adagrad method where the learning rate is divided by an exponentially decaying average of all squared gradients. This optimizer uses plain momentum.

C. *Accuracy metrics*

It is very common to evaluate the performance of a binary classifier using the area under the receiver operating characteristic (ROC) curve, AUC in short. The performance is regarded better if AUC is closer to 1. Thus, in our model training we use the AUC accuracy metric available in Keras.

On the other hand, to check the performance of our proposed model on both training and test sets after the network weight trainings are over we use metrics β_i's $0 \leq i \leq 4$ similar to [7]. Here, β_0 is called the complete measure which shows how well the neural network has learned by estimating the percentage of the results exactly matching the desired output. In order to see the results which were close to the label, the near measures β_i's are used $1 \leq i \leq 4$. More specifically, β_i indicates the percentage of the data for which at most i bits are wrong. These measures give some additional overview which is found to be useful when assessing the model used on the network.

IV. Results of the Experiment

In this section we state the findings of the experiments. As mentioned in the previous section, we consider four different scenarios, namely, the factorization problem for semiprimes of sizes less than 1,000, 10,000, 100,000, and 1,000,000 respectively and report the accuracy metrics $\beta_i's$ for each case. For the sake of comparison, we provide the results of Jansen and Nakayama from [7].

Table 2 provides results for the semiprimes less than 1000 with 33.3% of the data allocated for validation. As we see that the complete metric is $\beta_0 = 0.46$ for our model while it was 0.24 in [7].

TABLE 2. Model Performances, N < 1,000

Model	Type	β_0	β_1	β_2	β_3	β_4
Proposed RNN-ANN	Train (66.6%)	67%	90%	98%	100%	100%
	Test (33.3%)	46%	63%	89%	100%	100%
ANN (Jansen& Nakayama)	Train (66.6%)	61%	93%	100%	100%	100%
	Test (33.3%)	24%	48%	84%	100%	NA

With the increase of the size of our semiprimes, the model performance drop 10% with complete metric giving $\beta_0 = 0.36$ as given in Table 3. Surprisingly, the performance in [7] is improved to $\beta_0 = 0.53$ with the increase of the semiprimes up to 10,000.

TABLE 3. Model Performances N < 10,000

Model	Type	β_0	β_1	β_2	β_3	β_4
Proposed RNN-ANN	Train (66.6%)	60%	85%	97%	100%	100%
	Test (33.3%)	46%	61%	79%	95%	99%
ANN (Jansen& Nakayama)	Train (66.6%)	64%	81%	93%	98%	NA
	Test (33.3%)	53%	69%	83%	94%	NA

As in [7] we provide two tables for the results with semiprimes less than 100,000, see Table 4 and Table 5. While in the first table, only 33.3% data were allocated for the validation, it was 66.5% for the latter.

TABLE 4. Model Performances, N < 100,000

Model	Type	β_0	β_1	β_2	β_3	β_4
Proposed RNN-ANN	Train (66.6%)	40%	62%	80%	92%	98%
	Test (33.3%)	33%	54%	75%	90%	97%
ANN (Jansen& Nakayama)	Train (66.6%)	49%	63%	77%	89%	97%
	Test (33.3%)	45%	58%	72%	86%	95%

In terms of the performances on validation data, we do not see too much difference while there is a readily seen difference on train sets. In particular, with the decrease of the train set to test set ratio there is overfitting in our model.

TABLE 5. MODEL PERFORMANCE, $N < 100{,}000$

Model	Type	β_0	β_1	β_2	β_3	β_4
Proposed RNN-ANN	Train (33.3%)	65%	89%	98%	99%	100%
	Test (66.6%)	34%	50%	70%	87%	96%
ANN (Jansen& Nakayama)	Train (33.3%)	51%	66%	81%	92%	98%
	Test (66.6%)	44%	57%	72%	86%	95%

Finally we turn our attention to the factorization of semiprimes with sizes up to 1,000,000. To highlight the importance of regularization steps we first train our model without batch normalizations and dropouts. From Fig. 1, we see overfitting. This in turn gives very high performance in the train set while underperforming for the test set as shown in Table 6.

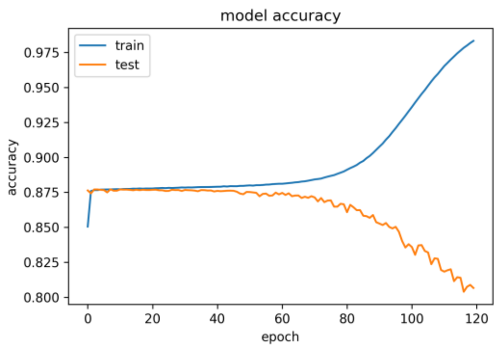

Figure 1. AUC graph without regularizations

As the experimental results from Table1 and Table 2 suggest, the test performance does not seem to change much when the train to test ratio is increased. For the semiprimes up to 1,000,000 we allocate only 10% of the data for training and remaining 90% left for validation see Table 6 and Table 7.

TABLE 6. MODEL PERFORMANCE WITHOUT REGULARIZATION, $N < 1{,}000{,}000$

Type	β_0	β_1	β_2	β_3	β_4
Train (10%)	98%	100%	100%	100%	100%
Test (90%)	25%	36%	53%	72%	86%

In the next, we report the results when regularizations are used as shown in Table 1 to overcome the overfitting. Fig. 2 shows the progress of the AUC metric during the training with 120 epochs and loss function graph is provided in Fig. 3. We can clearly see the effect of regularizations.

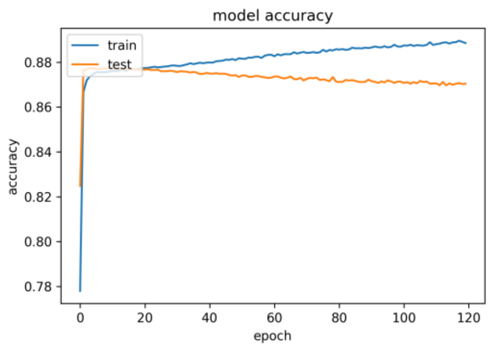

Figure 2. AUC graph with regularizations

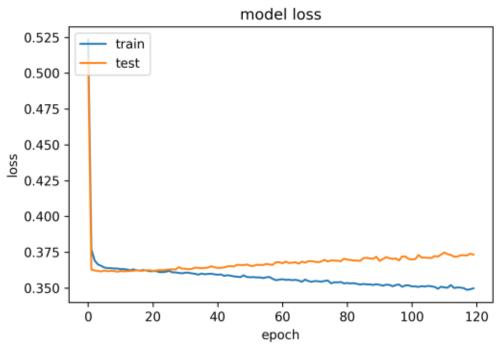

Figure 3. Binary cross entropy loss in 120 epochs for the model

From Table 7, we see that our model outperforms [7] with the accuracy reaching to $\beta_0 = 0.36$.

TABLE 7. MODEL PERFORMANCE WITH REGULARIZATION, N < 1,000,000

Model	Type	β_0	β_1	β_2	β_3	β_4
Proposed RNN-ANN	Train (10%)	38%	53%	70%	83%	93%
	Test (90%)	36%	50%	68%	82%	92%
ANN (Jansen& Nakayama)	Train (10%)	33%	50%	68%	84%	93%
	Test (90%)	28%	42%	60%	77%	89%

V. DISCUSSION AND CONCLUSION

Earlier, B. Jansen and K. Nakayama examined the ability of neural networks to learn factorizing semiprime numbers using a binary approach. In this paper, we address the same problem using binary approach but with different settings and architecture. To be more specific, artificial neural networks were trained together with recurrent neural network LSTM. Analyzing the results provided in the previous section, it is easy to see that the RNN-ANN approach proposed in this paper shows promising results with reaching out close to 100% accuracy on train sets. While the model outperforms [6] with complete metric accuracy of 36% compared to 28% in [7] as shown in Table 7 for semiprimes up to 1,000,000 it is clearly far from being satisfactory. We note that there are classical approaches [9] to the factorization problem such as the Pollard Rho algorithm based on generating numbers with many prime divisors. These algorithms can handle the factorization of , say fifteen digit, numbers in a short amount of time with almost 100% accuracy. However, for large numbers, say numbers with one hundred digits, these algorithms mostly fail unless one of the prime divisors is small.

In any case, we can say that deep learning approaches to prime factorization are still far from being satisfactory. Maybe, combining the classical factorization techniques to the machine learning methods can be beneficial and is a subject of the future work.

REFERENCES

[1] Sh. Jiang, K. A. Britt, A. J. McCaskey, T. S. Humble, and S. Kais, “Quantum annealing for prime factorization,” *Scientific Reports 8*, 2018.

[2] T. Kleinjung et al., “Factorization of a 768-bit RSA modulus,” *Advances in Cryptology – CRYPTO 2010*, vol. 6223, 2010.

[3] M. Mumtaz and L. Ping, “Forty years of attacks on the RSA cryptosystem: A brief survey,” *Journal of Discrete Mathematical Sciences and Cryptography*, vol. 22, 2019.

[4] M. Mishra, U. Chaturvedi, and K. K. Shukla “Heuristic algorithm based on molecules optimizing their geometry in a crystal to solve the problem of integer factorization,” *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, vol. 20, July 2015.

[5] W. A. Borders et al., “Integer factorization using stochastic magnetic tunnel junctions,” *Nature*, pp. 390-393, September 2019.

[6] G. C. Meletiou, D.K. Tasoulis, and M. N. Vrahatis, “A first study of the neural network approach in the RSA cryptosystem,” *Sixth IASTED International Conference on Artificial Intelligence and Soft Computing*, pp. 483–488, July 2002.

[7] B. Jansen and K. Nakayama, “Neural networks following a binary approach applied to the integer prime-factorization problem,” *Proceedings. IEEE International Joint Conference on Neural Networks*, vol. 4, 2005.

[8] A. Sherstinsky, “Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network,” *Physica D: Nonlinear Phenomena*, vol. 404, March 2020.

[9] Kimsanova, G., Ismailova, R. and Sultanov, R., “Comparative Analysis Of Integer Factorization Algorithms Using Cpu And Gpu,” *MANAS Journal of Engineering*, 5(1), pp.52-63, 2017.