Machine Learning Approach to Integer Prime Factorisation

Preprint · September 2022

DOI: 10.13140/RG.2.2.25570.81602

2 authors, including:

Ryan Nene
University of Southern California
2 PUBLICATIONS **0** CITATIONS

SEE PROFILE

Machine Learning Approach to Integer Prime Factorisation

Ryan Nene

Advisor: Suleyman Uludag

September 10, 2022

Abstract

Modern cryptanalysis has placed a great deal of focus on the RSA cryptosystem, one of the more popular public-key encryption algorithms used for communication between hosts. The resulting work pertains to computational short-cuts which might allow for the encryption to be broken in a practical amount of time. This paper focuses on the neural-network based approach to breaking this encryption through integer-prime factorisation. While factorisation is not efficiently possible algorithmically, a neural network is capable of guessing the smallest prime factor p in less time. An LSTM model proposed in previous works is recreated in this paper, such that from an input, N, the smallest prime factor p can be predicted. The limitations of this model are discussed, along with challenges present in the data set, and an extension to the domain of N is attempted and the results are examined.

Contents

1 Introduction

The RSA cryptosystem is an algorithm by which asymmetric key cryptography can be used to enforce end-to-end encryption. As the name would suggest, the algorithm is based in two different keys, often denoted as e, and d. The derivation of these keys is reliant on the generation and subsequent multiplication of two random prime numbers, p and q, to produce N. In the functioning of the algorithm, however, the number N is never hidden from the public. One would think that, as a result, the algorithm would be ineffective, however, the security of the algorithm relies on the one-way nature of $N = p * q$. N does not give much useful information as to what its prime factors are, not to mention that, as per recommendations from the NIST, N will generally be a 2048 bit integer or larger [1]. In turn, the ability to compute or find the prime factors of N is considered impractical for modern computers, as it would either necessitate an exhaustive search or more complicated mathematical methods. The methods explored in this paper aim to resolve this issue by adopting a machine learning approach involving the model put forward in Murat et al. The major advantage to the usage of this method is the avoidance of heavy computations by instead training a neural network to "guess", or predict, the smallest prime factor p given the semi-prime N, a semi-prime being the product of two primes, p and q.

Successfully approaching this problem in particular can strongly guide information security; any algorithm that can find these factors in polynomial time would thus create the need for stronger cryptosystems. Such factorisation algorithms also have other uses outside of the field of cybersecurity, primarily in physics [2].

2 Background

2.1: The RSA Cryptosystem

A cryptographic system is defined as any mathematical system that is used to transform or alter information in such a manner that the information cannot be read by those who are not the intended recipient. [3]. The RSA cryptosystem utilizes modular arithmetic and number theory in order to create strength in its encryption, as well as the Diffie-Hellman algorithm for asymmetric key encryption. As per the algorithm, for any host on a network, the host will generated a private key, D, and a public key, E. The public key is shared with other hosts on the network; the private key is known only to the original host. The public key D can therefore be used by a sender to convert any plaintext message, M, into ciphertext, C. An important characteristic of the algorithm is that the application of D is the inverse function of E, such that D(E(M)) = M. Thus, D can then be used by the receiver in order to convert C to M. This algorithm can be represented as follows:

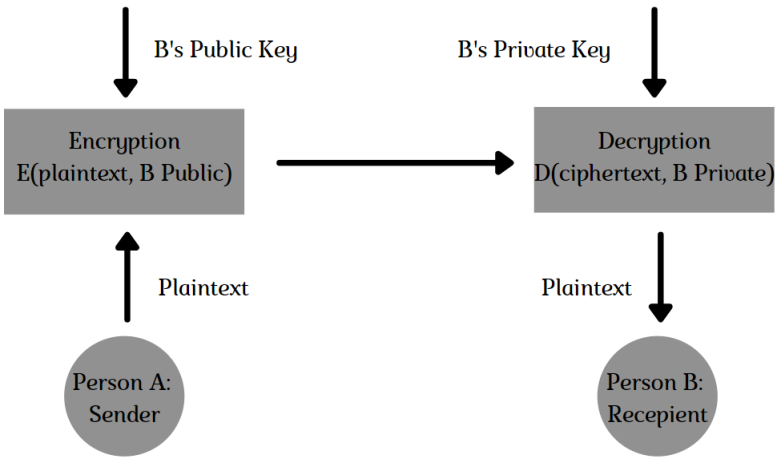


Figure 1: Asymmetric Key Encryption through Diffie-Hellman Algorithm

The RSA (Rivest-Shamir-Adleman) algorithm specifically accomplishes the generation of these two numbers based on the random primes p and q, the product of which is N. The algorithm next applies Euler's totient function, represented as $\phi(n)$, which is a function which returns the count of numbers $n \leq N$, such that the greatest common divisor between N and n is 1, or rather, that they are relatively prime. An important property to note is that for a prime number is relatively prime to all numbers except itself. Thus, $\phi(p) = (p-1)$. Secondly, the totient function is multiplicative; if a = bc, then $\phi(a) = \phi(b) * \phi(c)$. Hence, for a semi-prime N, $\phi(N) = \phi(p) * \phi(q) = (p-1)(q-1)$. Using this, a random integer for E is chosen such that it is relatively prime to $\phi(N)$. The multiplicative inverse of $E \mod \phi(N)$ is then taken to be D. The public and private key are generated thereby, and E and N are shared across the network [4].

The strength of the RSA algorithm is derived from the lack of information that is given by N. The security of ciphertexts generated by applying E to the plaintext message would be compromised if either D, p, or q were known. However, the problem of finding any of the above numbers requires the factorisation of N, which itself presents a difficult problem. The difficulty of this factorisation increases massively as N increases. In a typical encryption process, p and q are chosen such that N has 2048 binary digits. Thus, the factorisation of N becomes an infeasible task for all modern computers, and so, the encryption remains secure. [3]. As a result, cracking this problem has become the interest of cryptanalysts in recent years. Numerous approaches have been formulated using a variety of methods, however, these methods often become computationally strenuous, requiring large amounts of time in order to complete the task. Thus, except for the advent of quantum computing, there is little means of factorising N with practical efficiency.

2.2: Deep Learning, LSTM-RNN Layers

Deep learning is an extension of machine learning, which describes the field of study concerned with computer-based learning models, spanning from simpler regression algorithms to deep learning. In the case of this matter, deep learning is the more relevant type. Deep learning is concerned with neural-networks. A neural network is best explained as a miniature, specialised brain. It is a set of layers of neurons of varying type, with each neuron taking input from all neurons in the previous layer, and outputting to each neuron in the next layer. Each of these inputs from the previous layer has a specific weight and bias associated with it, that is, if the output of a neuron is a, then the input to a given neuron is equal to $(w * a) + b$, where w represents the weight, and b represents a constant bias. These values are changed through the process of back propagation, in which a gradient descent is performed on a high-dimensional cost function. Through this back-propagation, the model is thus capable of getting better at accomplishing the task at hand. As a result of the incredibly dynamic nature in which neural networks can perceive and handle data, they are able to perform tasks which would otherwise be considered very difficult algorithmically, such as image classification, object recognition, as well as tasks involving generation, as is the case with generative adversarial networks, or recurrent neural networks, the latter of which is used in the implementation.

An RNN refers to a recurrent neural network, a type of network which stores an embedding state between time steps. The RNN is thus capable of "remembering" features from its previous output, however, this short-term memory is only persistent for one time step. The LSTM is the logical extension to this operation, making inclusion of a forget and concatenate gate for the embedded state. The network is thereby capable of retaining specific memories for longer — hence the name, Long Short-Term Memory [5]. The purpose for its usage in this context discussed later.

3 Related Work

The amount of research into implementations utilizing the neural network approach to this factorisation problem is rather limited. As stated in the introduction, the implementation discussed in this paper is based heavily on papers by B. Jansen and B. Murat et al. Aside from these two papers, there is a limited amount of research into the relevance of neural networks in a factorisation based attack of the RSA cryptosystem. The first paper on the subject, published

in 2002, outlines the rudimentary background as well as some experimentation, and employs a variety of smaller feed forward artifical neural networks (ANNs) of varying size [6]. An important distinction in the methods used by G. Meletiou et al. was that the model output was to predict $\phi(N)$, which is equivalent to $(p-1)(q-1)$ as mentioned. However, the models' overall performance was poor relative to the accuracy seen in the later works, reaching a maximum of approximately 5-6% for $N < 10,000$. In any case, the most important findings of this paper were that the type of backpropagation used was irrelevant to the outcomes of the model with regards to accuracy, as is noted by Murat.

As for the remaining papers, the specifics of the approaches outlined vary from one another – B. Jansen proposed a single layer ANN based implementation, which demonstrated reasonable effectiveness in solving the problem, however, the far larger LSTM-based model proposed by Murat demonstrated higher effectiveness for all trials. The major reasoning behind the usage of an LSTM is that more information could potentially be gained through the usage of the memory of the nodes, with the reasoning that parsing the binary digits which constitute the input semi-prime could be treated as though it were a language comprehension task, a specialty of LSTM networks. This approach thus proved more effective than the prior single-layer networks for all ranges of N up to 1,000,000. For this reason, said implementation was reproduced for purposes of experimentation and extension in this paper, although certain model parameters and hyperparameters needed to be inferred or tested.

The most important difference in approach was the binary approach proposed by Jansen. The binary approach describes the conversion of numerical semi-primes and primes into binary ones, thus entirely changing the manner in which the model handles the input and output data. As noted in said paper, this changes the problem from a rather difficult numerical guessing problem into a classification problem, wherein the model would be tasked with determining simply whether an output bit in the set should be on (1) or off (0). This was said to give the model better stability, allowing the model to converge upon a cost minima, and thus, fit with much higher accuracy to the data. Additionally, . This same technique was used once more in the LSTM based model.

Even still, by far the largest challenge in this problem, as mentioned in the introductory section, is the lack of useful information in the semi-prime itself. Given that any data set of semi primes will include random numerical data, so to speak, there is no real trend nor structure within the data set that might allow for data preprocessing or principal component analysis. As a result, the problem of efficiency persists: while the model is nonetheless capable of capturing some pattern in the data, as evidenced by its accuracy, the size of the overall data set itself prevents efficiency in the model's operation. As such, this problem is discussed as part of the experimentation.

4 Implementation

The implementation used for this particular purpose was a close recreation of the model proposed in the Murat et al. paper. This was due to the fact that it typically showed better performance when compared to the single hidden layer artificial neural networks used in Jansen's work.

In order to generate the data set, a series of different algorithms were used. First, was the sieve of Eratosthenes, an algorithm by which prime numbers can be filtered out from a set [a,b]. The algorithm works by calculating the divisibility of every number in the data set against all other numbers in the set. Initially, for the purposes of this implementation, the set of [0,1000] was input into the algorithm, as this would give all primes with a maximum product of 1,000,000. This range would be changed for experimentation. Once this set of primes was generated, the product of every possible pair of primes was calculated to generate the list of semiprimes. The calculated semiprime and smallest prime factor of the semiprime were converted to binary, then stored in parallel python dictionaries, where they could then be appended to a pd.DataFrame() object. This way, the input data and its label were related through their indices. A slice of these indices could then be taken to limit the size of N.

It should be noted, however, that 2 was excluded as a prime. This was because 2 is even,

and so, any semiprime with the smallest prime 2 would be even, which is a known property of even numbers in multiplication. The resulting semi-prime, in binary form, must therefore have its 2^0 bit off, as only in this way can it be even. Thus, factoring said semi-prime becomes trivial, as pointed out in [7], and so, there is no need for such numbers to be included in the data set. Additionally, given that there cannot be other even primes, 2 is the only prime that might require exclusion. This reduces the overall semiprime set size, and so, reduces the computation time needed for training and testing of the model.

The model was then created using the TensorFlow library for python on a Windows 10 operating system. Data preprocessing and storage was done through Pandas and Numpy. An input shape of (1,20) was given to the model due to the binary size of the semiprimes, as for $N < 1,000,000$, the maximum size of N would be 20 binary digits. Similarly, the maximum size of the primes was 10 digits, hence the final output shape of 10 neurons, although these values were not hard-coded such that the input and training data sets could be expanded to larger values of N. To form predictions using the model, the raw prediction by the model would be rounded, thus giving a set of 10 digits of either 1 or 0.

Evaluation of the model was done through 3 metrics: Area under Curve (AUC, metric of categorical accuracy), loss (through Binary Cross Entropy loss function), and bit accuracy, which describes the rate of a certain number of errors in the model's predictions, with 0 meaning a totally accurate prediction. The model was thus compiled with the AUC accuracy metric, aforementioned loss metric and the RMSprop back propagation algorithm for reasons stated in [2].

The model architecture and parameters are described below.

Layer (type)	Output Shape	# Params	Args
LSTM	(None, 1, 128)	76288	input_shape = (1,20)
Batch Normalization	(None, 1, 128)	512	
Dropout	(None, 1, 128)	0	rate = 0.3
LSTM	(None, 1, 256)	394240	
Batch Normalization	(None, 1, 256)	1024	
Dropout	(None, 1, 256)	0	rate = 0.3
LSTM	(None, 1, 512)	1574912	
Batch Normalization	(None, 1, 512)	2048	
Dropout	(None, 1, 512)	0	rate = 0.3
Dense	(None, 128)	65664	activation="relu"
Batch Normalization	(None, 128)	512	
Dropout	(None, 128)	0	rate = 0.3
Dense	(None, 100)	12900	activation="relu"
Batch Normalization	(None, 100)	400	
Dropout	(None, 100)	0	rate = 0.4
Dense	(None, 10)	1010	activation = 'sigmoid'

Figure 2: Model Architecture and Arguments

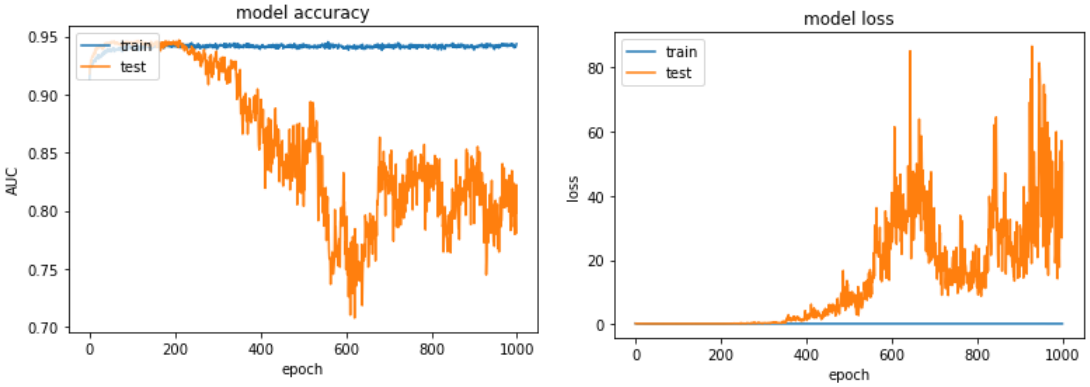
5 Experimentation and Discussion

5.1: Experimentation

Experimentation with this model first began with the tuning of hyper-parameters of the model. The main reason for this was due to the lack of available information in the papers outlining an implementation. Additionally, perfecting such hyper-parameters is one of the simplest steps that can be taken towards efficiently and accurately predicting prime factors. Batch size, the fraction of training data set to be used to train the model in a given epoch, as well as the train-test split, which describes how the full dataset is to be split into a train and test set, were the first variables that were investigated for $N < 1,000,000$. In [2, 7], the typical train-test set used a 66% train, 33% test split for $N < 100,000$. This differs significantly from the 10%, 90% split used for $N < 1,000,000$ data set specifically. In practice, however, the reason behind this is rather evident in the resulting training of the model.

As can be seen in the figure above, model performance is spectacularly underwhelming while using the typical train-test split, and is greatly improved using the 10%-90% split. The reason for this is likely the model's tendency to over fit to the data set, which describes the behaviour of the model becoming incredibly accurate for the training set, but rather inaccurate in the test set. This behaviour is common in models containing more parameters, as there is an increased capability in such case for the model to become too specific to the training set provided. The effect of this overfitting becomes evident in the accuracy and loss metrics for the model, regardless of the value of N.

Combatting this overfitting or underfitting of the model would thus become a matter of properly tuning the number of epochs for which it would run. In [2], the number of epochs is not specified for all trials; only 120 epochs is specified for $N < 1,000,000$. As such, to test the optimal number of epochs, the model was run for 1000 epochs. The results at $N < 1,000,000$ are rather clearly evident of approximately 200 to 300 epochs being optimum, although it must be noted that this number decreases or increases for different domains of N.



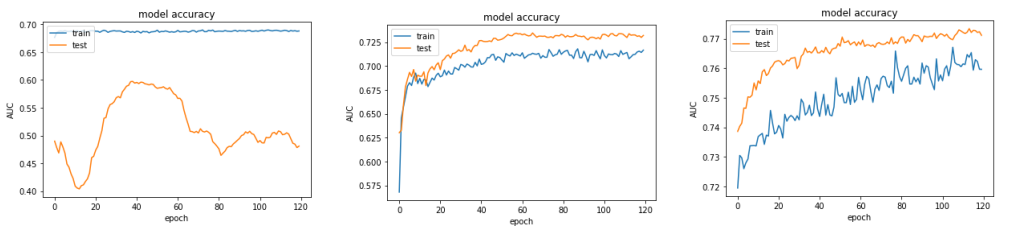
(a) Model Accuracy (b) Model Loss

Figure 3: Model Accuracy and Loss, $N < 1,000,000$, 1000 Epochs

Issues with model accuracy were also strongly related to the batch size. Batch size, b, refers to the proportion, $1/b$, of the training set to be used in each iteration. In the case of this specific implementation, the effect of batch size on the overall accuracy and error of the model was obvious. Testing for 120 epochs, $N < 1,000,000$, the following graphs were generated, which show that a batch size of 4 proved optimal, as shown in 4,5.

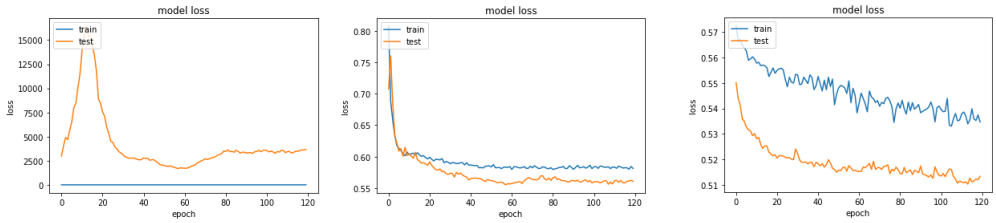
With the hyperparameters in place, reproduction of the results produced by Murat et al. proceeded. The model was trained and tested on data sets of $N < 1,000$, $N < 10,000$ at a batch size of 4 for 120 epochs; this data is shown in 6. The domain on which the algorithm was trained was also further extended to $N < 100,000,000$. This would require that the input shape be changed to (1,27), and the output shape be changed to 14 digits. The relevant data is shown in 7.

Curiously, when given a larger output space, the performance of the model at $N < 1,000,000$ appears to improve. Figure 8 shows the histogram plot of the number of errors and their rates after the model was trained for 1000 epochs on $N < 100,000,000$.



(a) Model AUC, Batch Size 1 (b) Model AUC, Batch Size 2 (c) Model AUC, Batch Size 4

Figure 4: Model Accuracy for Batch Size b = 1,2,4, 120 Epochs

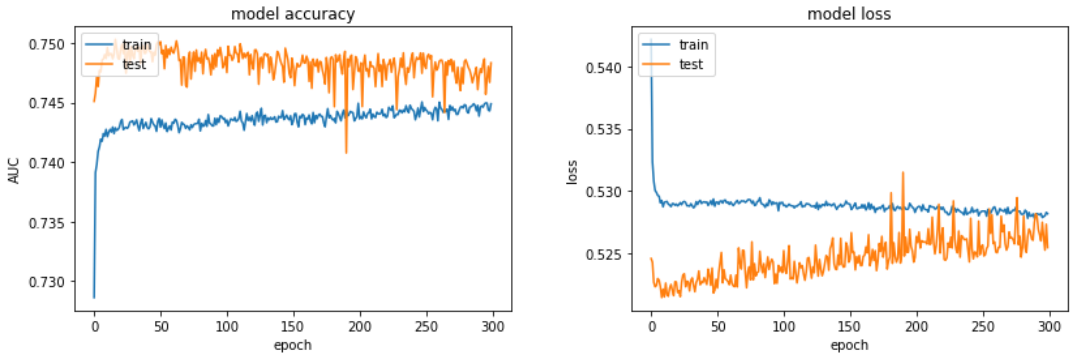


(a) Model Loss, Batch Size 1 (b) Model Loss, Batch Size 2 (c) Model Loss, Batch Size 4

Figure 5: Model Loss, Batch Size b = 1,2,4

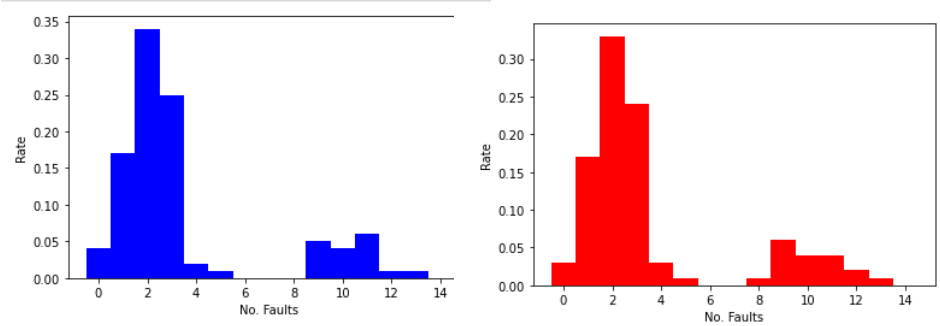
	N<1000	120 epochs							
	0	1	2	3	4	5	6	7	8
Train (66%)	0.02	0.06	0.15	0.25	0.24	0.17	0.08	0.02	0.01
Test (33%)	0.01	0.08	0.14	0.23	0.23	0.17	0.1	0.02	0.01
	N<10000	120 epochs							
	0	1	2	3	4	5	6	7	8
Train (66%)	0.01	0.05	0.15	0.23	0.25	0.19	0.08	0.03	0.01
Test (33%)	0.01	0.06	0.15	0.23	0.24	0.2	0.08	0.03	0.01

Figure 6: No. Errors and Rate



(a) Model Accuracy (b) Model Loss

Figure 7: Model Performance, $N < 100,000,000$, 300 Epochs



(a) Training Set (b) Test Set

Figure 8: Model Accuracy in No. Errors and Rate

5.2 Discussion

As shown above, the recreated implementation is capable of accurately producing the smallest prime factor for a given semiprime at a non-negligible rate, however, it is strange that the results would vary so differently from those shown in [2]. Given that the implementation directly replicates the conditions, parameters, and architecture used in said paper, it is not certain what exactly may be causing this discrepancy in performance, and it is very much possible that this is due to an issue with the gradient descent algorithm converging to a local minima, as opposed to a global one. However, encounters with this problem were never mentioned by Murat et al, and the need for avoiding the problem was similarly dismissed. In any case, however, through the usage of this implementation the limitation of computational speed becomes very much apparent. In testing for large values of N, the training process could often take multiple days in order to complete, owing to the complexity and sheer size of the data set itself.

In this respect, certain aspects of the data's structure were investigated using the Sklearn PCA algorithm, as well as the creation of a two-dimensional T-SNE plot. Using the MLE algorithm, an algorithm included in the sklearn module which attempts to calculate the number of principal features in a data set, it was seen that there was no means by which to reduce the dimensions of the data set, as each feature had an explained covariance of 7% to 2%. As a result, without a reliable and/or effective manner of decreasng the size of the data set, the task becomes far more computationally difficult for any processor, impeding this approach to factorisation. A t-SNE plot was also used to analyse the structure of the data. The t-SNE plot is a tool used in data analysis which gives an impression of the relation between different data points, thus giving a sense of structure [8]. Depicted in 9, the plot reveals the lack of structure and resulting random dispersion of the data points, both of which contribute to the issue of a lack of information.

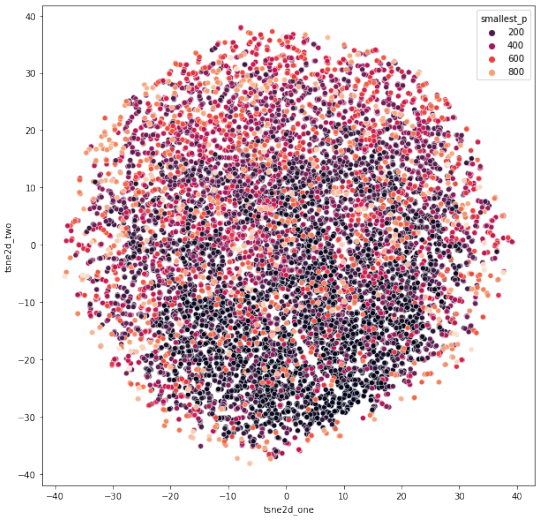


Figure 9: T-SNE Plot of Semi Prime Data Set, Colored by Smallest Prime Size

Another major issue shown in the data is the prevalence of both underfitting and overfitting, as can be seen in the graphs. Given the lack of information that is provided by the data set to any sort of algorithm, it can be seen that underfitting would occur at almost any level. This effect has been shown to increase as N increases, as can be seen in data for $N < 100,000,000$. Generally, this has to do with the number of alterable parameters in the model, which determines the flexibility of the outcome [9].The model used is reasonably complex such that it is able to fit itself to the function, however, it may be the case that this complexity falls short for larger values of N, especially given the incredibly complex nature of the data set, as shown above. On the other hand, as would be the case for any network, the model has a tendency to overfit to the dataset, eventually reaching a point during training wherein the performance on the training set is far better than on the test set. It stands to reason that the lack of structure in this particular data set causes the model to become sensitive to a pattern present in the random training set, but missing in the test data set, creating exaggerated overfitting. This issue may be solvable

through the implementation of Early Stopping, an algorithm which uses recent data of the model's performance on the validation set to decide, given a specified tolerance, when training should stop. While this is subject to some error, it has been demonstrated to be reasonably effective in preventing overfitting [10]

6 Conclusion

Machine learning is proving to be a somewhat capable means of tackling the integer factorisation problem. As demonstrated in the referenced papers, as well as in the experimentation described above, the LSTM-based model, alongside others, is able to predict the smallest prime factor for relatively large semi primes with some accuracy. Even still, it is clear that the approach is still lacking, and is oft outperformed by other existing approaches to the problem, such as in the example of [11]. Nonetheless, the major benefit of the deep learning approach is the avoided computation time taken in such methods.

Given the results of the experimentation with the LSTM-based network, it is rather clear that the model architecture is yet to be perfected. As is well known in machine learning, more complicated neural networks often carry the risk of overfitting to their training data-set, which is very much observable in the data produced by this particular model. That is not to say that a simpler model would work better, however. As discussed, the required sensitivity in outcomes of a model built for this purpose would need to be quite high given the complexity of the data set. It may well be the case that recent machine learning projects, such as OpenAI's GPT-3 or Google's Deep Mind may be more capable of such a task due to their sensitivity to input data. Further investigation into their use in this problem is thus a possibility.

Additional work in analyzing the data set will also become important in the problem, especially as the range of N used and the number of features grows. Lowering the dimensionality of the data set, although difficult, will make the training process as well as the model itself far more efficient, and thus, would be an important step in cultivating this approach to the factorisation problem.

References

[1] E. Barker and Q. Dang, "Recommendation for key management part 3: Application-specific key management guidance," *NIST Special Publication 800-57*, p. 60, Jan 2015.

[2] B. Murat, S. Kadyrov, and R. Tabarek, "Integer prime factorization with deep learning," 2020.

[3] M. E. Hellman, "The mathematics of public-key cryptography," *Scientific American*, vol. 241, no. 2, p. 146–157, Aug 1979.

[4] D. Luciano and G. Prichett, "Cryptology: From caesar ciphers to public-key cryptosystems," *The College Mathematics Journal*, vol. 18, no. 1, p. 2–17, 1987.

[5] A. Sherstinsky, "Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network," *CoRR*, vol. abs/1808.03314, 2018. [Online]. Available: http://arxiv.org/abs/1808.03314

[6] G. Meletiou, D. Tasoulis, and M. Vrahatis, "A first study of the neural network approach to the rsa cryptosystem," 07 2002, pp. 483–488.

[7] K. N. Boris Jansen, "Neural networks following a binary approach applied to the integer prime-factorization problem," *2005 IEEE International Joint Conference on Neural Networks*, 2005.

[8] L. van der Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research 9*, p. 2579–2605, Nov 2008.

[9] W. Koehrsen, "Overfitting vs. underfitting: A complete example," *Towards Data Science*, 2018.

[10] H. Jabbar and R. Z. Khan, "Methods to avoid over-fitting and under-fitting in supervised machine learning (comparative study)," *Computer Science, Communication and Instrumentation Devices*, vol. 70, 2015.

[11] F. Boudot, P. Gaudry, A. Guillevec, N. Heninger, E. Thomé, and P. Zimmerman, "Sympa.inria.fr," 2020. [Online]. Available: https://sympa.inria.fr/sympa/arc/cado-nfs/2020-02/msg00001.html