Deep Learning Architectures for Approximating Goldbach's Function in New Regions

Avigail Stekel, Amos Azaria

Computer Science Dept., Ariel University, Israel

***Abstract*—Goldbach conjecture is one of the most famous open mathematical problems. He asserts that: Every even number greater than two is the sum of two prime numbers. The Goldbach function receives an even number and returns the number of different ways to write it as an unordered sum of two prime numbers. We developed a simple multi-layer perceptron that attempts to predict Goldbach's function. This simple model performs well when trained and tested on numbers up to 4 million. However, as expected, the model's performance significantly deteriorates when trained on smaller numbers (up to 4 million) but tested on larger numbers ($4-10$ million).**

To overcome this problem, we present two novel deep learning architectures. In these architectures we introduce two types of multiplication layers, which we believe are more appropriate for solving mathematical relations. We show that both architectures significantly outperform the simple multi-layer perceptron when trained on smaller numbers and tested on larger numbers. We further improve the performance of the deep learning architectures by using a known analytically derived estimation that is used in order to normalize the model's output.

***Index Terms*—Goldbach's function; Deep learning; Out-of-scope inference.**

I. Introduction

In June 1742, the mathematician Christian Goldbach wrote a letter to his friend, Leonard Euler, describing his conjecture that states that every even integer larger than two is a sum of two prime numbers [1]. Since then mathematicians have tried to prove this conjecture or disprove it. Even though more than two hundred and fifty years have passed, the conjecture remains open. To this date, Goldbach's conjecture has been verified manually up to 4×10^{18} [2]. During the past centuries, despite no actual proof being found, there has been some important and significant progress related to this conjecture.

A more general problem is to determine the number of different options there are for a given even number, n, to be written as a sum of two prime numbers. Each such option is called a Goldbach partition. That is, a Goldbach partition is composed of three numbers: two primes, which sum to a given even number, n. This problem is known as the *Goldbach's function*, denoted $G(n)$ [3]. Rephrasing Goldbach's conjecture in terms of Goldbach's function would state that the value of Goldbach's function (for all even numbers greater than 2) is greater than or equal to 1. For example, $G(100) = 6$, because $100 = 3 + 97 = 11 + 89 = 17 + 83 = 29 + 71 = 41 + 59 = 47 + 53$ and there any other. The plot of the Goldbach's function has a form of a comet and is consequently called "Goldbach's comet" [3] (See Figure 1 for Goldbach's function values for all even numbers up to 4×10^6.

Several works ([4], [5]) have suggested different approximations of Goldbach's function, which they derived analytically. Unfortunately, some of these approximations are very far from the actual values taken by Goldbach's function, while others require prime factorization (prime decomposition), which is believed to be an intractable operation on large numbers. In this paper we suggest a different approach to approximate Goldbach's function, by utilizing unique architectures for neural networks (i.e., deep learning).

Deep learning is a hierarchical based approach to approximate complex functions, which most commonly uses neural network architectures that are composed of multiple layers. Training the network is performed by executing grading descent, which adjusts the weights of the network in order to minimize a loss function. Indeed, deep learning has shown success in many different fields [6]. However, it may seem that deep learning is not a suitable approach for this type of problems, as the input to the approximation function is only a single number, and deep learning has shown success in situations in which the input is composed of a large vector or a matrix. We therefore propose a simple, yet powerful concept of translating the number into different bases. Using our approach, a simple multi-layer perceptron performs well when trained on some numbers up to 4 million and tested on other numbers up to 4 million as we show in [7]. However, the simple multi-layer perceptron's performance significantly deteriorates when trained on smaller numbers (up-to 4 million) but tested on larger numbers (4-10 million).

To overcome the problem of the multi-layer perceptron model that does not perform well when tested on a new distribution, we present two novel deep learning architectures. In these architectures we introduce two types of multiplication layers, which we believe are more appropriate for solving mathematical relations. The first architecture includes a multiplication layer, in which some of the neurons are multiplied before serving as an input for the next layer. In the second architecture, an additional layer is included, in which all inputs are activated using a logarithmic function, and then all outputs are activated with an exponential function. This allows the multiplication of any input neurons. We show that both architectures significantly outperform the simple multi-layer perceptron when trained on smaller numbers and tested on larger numbers. We further improve the performance of the deep learning architectures by using a known analytically derived estimation (see Equation 6) that is used in order to normalize the model's output.

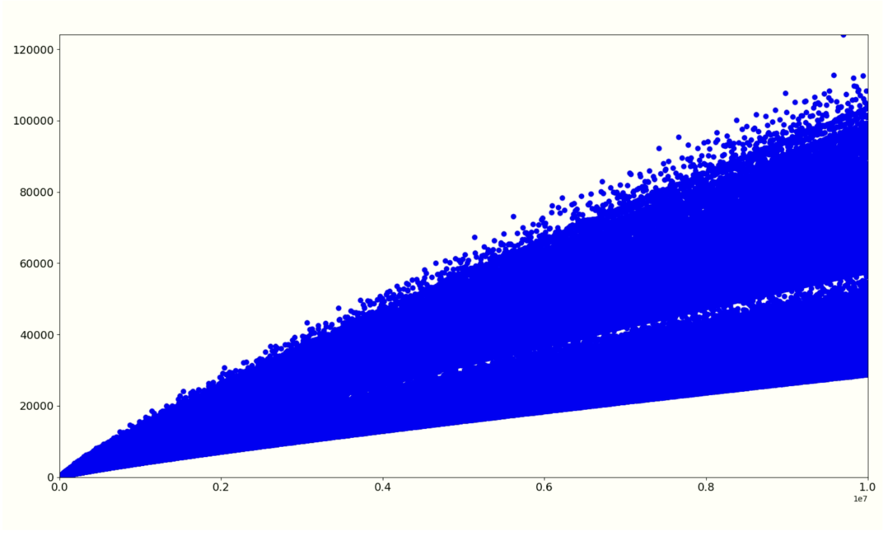

Fig. 1: Goldbach's function values for all even numbers between 4 and 10×10^6. This function is sometimes referred to as Goldbach's comet, due to its shape.

II. Background

Prime and natural numbers have always aroused mathematicians' interest. In 1900 Hillbert made his famous speech at the 2nd International Congress of Mathematics held in Paris, saying there are 23 unsolved problems for mathematicians of the 20th century [8]. One of those math problems was Goldbach's conjecture.

A. Approximations of Goldbach's Function

Goldbach's Conjecture is divided into two conjectures:

1) The 'weak' Goldbach's conjecture states that 'every odd number greater than 5 can be expressed as a sum of three primes'. For example, 11 is the sum of 3, 3 and 5. 21 is the sum of 2, 2 and 17. The weak conjecture was finally proved in 2013 [9].
2) The 'strong' Goldbach's conjecture, which states that 'every *even* integer greater than 2 is a sum of two primes'. The number 6 for example, can be presented with only one pair of prime numbers, $3 + 3$. However, when examining even numbers greater than 12 , there are apparently, at least two pairs of prime numbers that sum to each number. One might assume that the greater the even number, the more different pairs it has, yet by observing different even numbers this assumption does not always hold. For example, while 34 and 36 have 4 Goldbach partitions each, 38 has only 2 Goldbach partitions. There have been multiple attempts to make progress with respect to the strong Goldbach's conjecture, some of which were very recent [10], [11], [12]. However, this conjecture remains open to date.

In this paper we focus on Goldbach's function, which provides the number of Godlbach's partitions an even number has. More formally, let $n \in \mathbb{N}$. Then, Goldbach's function is given by:

$$G(n) = \sum_{\{p,q\} \in \mathbb{P} \times \mathbb{P} \,\wedge\, p \leq q} \mathbb{1}\{p + q = n\} \tag{1}$$

where, $\mathbb{P}$ is the set of all prime numbers, and $\mathbb{1}$ is the indicator function that returns 1 if the expression is *true* and otherwise 0.

Over the years there have been several attempts to find an analytic approximation of Goldbach's function. Hardy and Littlewood [4] proposed the following approximation:

$$G_1(n) = 2 \cdot C_2 \frac{n}{(ln(n))^2} \prod_{p|n} (\frac{p-1}{p-2}) \tag{2}$$

where n is an even number, p denotes all the prime factors of n. C_2 is a number that they refer to as the twin prime constant, which equals:

$$C_2 = \prod_{p \geq 3} \left(1 - \frac{1}{(p-1)^2}\right) \cong 0.6601618158 \tag{3}$$

where p denotes all prime numbers. We note that C_2 is called the twin prime constant because it was previously used in formula developed to estimate the number of twin primes (i.e., two primes, p_1 and p_2 such that $|p_1 - p_2| = 2$) that are smaller than a given number.

While this function was originally proposed as an upper-bound, it is widely used as an approximation. Baker suggested multiplying $G_1(n)$ by $\frac{3}{5}$ to yield a better approximation [5] (we will refer to Baker's approximation as $G_{1'}(n)$). As stated by Hardy & Littlewood [4] the $G_1(n)$ function can only be used as a good approximation when approaching the limit (i.e., for very large numbers). Therefore, Granville [13] provides the following function, which achieves a better approximation for smaller numbers:

$$G_2(n) = C_2 \int_2^{n-2} \frac{dt}{ln(t) \cdot ln(n-t)} \prod_{p|n} (\frac{p-1}{p-2}). \tag{4}$$

Granville [13] shows that $G_2'(n) = G_2(n) \cdot \left(1 - \frac{4}{\sqrt{n}} \prod_{p \geq 3} (1 - \frac{n/p}{p-2})\right)$ yields a better approximation.

Note that $G_1(n)$, $G_{1'}(n)$, $G_2(n)$ and $G_2'(n)$ require factoring n, which is assumed to be a hard problem. Currently, the best known prime factorization algorithm (GNFS) [14] runs in a time complexity of:

$$O\left(exp\left(\left(\sqrt[3]{\frac{64}{9}}+o(1)\right)\sqrt[3]{log(n)}\sqrt[3]{(loglog(n))^2}\right)\right)$$

where n is the number factored. Note that the input size is considered $log(n)$, since the number of digits required to represent n is $log(n)$. We therefore do not consider those approximations in this paper (see [7] for a comparison of a deep learning based method with these methods).

To overcome the factorization requirement, the following approximation of Goldbach's function was proposed [15]:

$$G_3(n)=\frac{n}{(ln(n))^2} \tag{5}$$

This approximation is derived from Gauss' approximation provided in 1793 for the probability of a number being prime. According to Gauss, this probability is given by:

$$f(m)=\frac{m}{ln(m)}$$

Therefore, for an even number n the following may be used as an approximation of its number of Goldbach partitions:

$$\sum_{m=3}^{n/2}\frac{m}{ln(m)}\cdot\frac{n-m}{ln(n-m)}\approx\frac{n}{2ln(n)^2}$$

Note that G_3 is monotone, and thus cannot capture the phenomenon that larger numbers may sometimes have less Goldbach partitions than smaller numbers. The following approximation, which is also monotone, was proposed by Markakis et al. in [16]:

$$G_4(n)=\frac{n}{(ln(n/2))^2}. \tag{6}$$

We note that G_4 was shown to outperform G_3 (see [7]).

III. Deep Learning Based Architectures for Goldbach's Function Approximation

In this section we present several deep learning based models to approximate Goldbach's function values. In previous work we compared the simple deep learning based model (the Multilayer perceptron Basic Model) to other baselines when trained and tested on numbers up to $4M$ [7]. In this paper we focus on the performance of our models when trained on numbers up to $4M$, but tested on larger numbers ($4-10M$).

A. Data Composition

In order to train and evaluate the different methods, we composed a dataset consisting of the number of Goldbach partitions that all even numbers from 4 to 4×10^6 have. To that end, we first computed all prime numbers at that range, and stored them as a list and as a hashmap. For each even number, n, we iterated on all prime numbers (using the list of all primes) that are smaller than or equal to $\frac{n}{2}$. For each of these prime numbers, p, we tested (using the hashmap) whether $n-p$ is a prime number itself. If so, we incremented n's counter by one.

We shuffled the data and split it into a training-set, (90% of the data; 18×10^5 numbers), and the remaining 10% was reserved for the $0-4M$ test-set (2×10^5 numbers).

B. Model Features

From each number we extracted the following 26 features. We converted every number to its binary representation, ternary representation (base 3), quinary representation (base 5) and septenary representation (base 7). The time complexity of computing these base representations for a number n is $O(log(n))$. In practice we computed those representations while composing the data. In order to compute the base-representations, we iteratively divided the number by the required base. We truncated the base representations and used the 6 least significant digits for each representation. The intuition behind using these different representations lies in the fact that these transformations are computationally cheap to extract and that they might allow the model to retrieve underlying information on the number. The first 4 prime numbers $(2,3,5,7)$ were selected as the bases. We used the 6 least significant digits because it is the largest number of digits that allows the biggest base (base 7) to complete at least two cycles with the training data. That is, $2\times7^7<4\times10^6$, but $7^8>4\times10^6$ (and therefore, we cannot use 7 digits). In addition to the representations in the different bases, we added the number itself (divided by 2,000,000, which is the average of the training-set), and the logarithm of the number.

IV. Model Architectures

We used 3 different model architectures:

A. Basic-MLP

This simple model uses a fully connected neural network. The number of neurons was set to 200 on each hidden layer, and the Adam optimizer [17] was used, with a learning rate of 0.001. We used a mini-batch size of 1024 and trained the model for approximately 200 epochs on the data. These values are similar to those we used in previous work [7]. In previous work, [7], we tested several options for finding the optimal number of hidden layers; the model with 5 hidden layers outperformed all other options. We therefore used 5 hidden layers in this paper as well.

For a given number n, the time complexity of generating the features and evaluating our model is $O(log(n))$, which is the best time complexity one could expect from an algorithm that reads the entire input (which requires $O(log(n))$ digits to represent).

We now introduce our novel architectures that were developed for the Goldbach's function approximation in new regions.

B. Multiplication-Layer Model

The multiplication-layer model includes a layer that multiplies pairs of neurons (see Figure 2). Namely, after several fully connected layers (in our case 2) a fraction of the following layer is separated from the rest (in our case 40% of the neurons were separated). The architecture includes random connections between some of the separated neurons; these neurons were multiplied by each other. In practice the model shuffles the rows of an identity matrix, which we denote J.

The group of neurons that were separated, denoted V, is then multiplied, on the right side, by J. The result, $V \times J$, is component-wise multiplied by the original group of separated neurons, V . This process implies that each of the neurons is paired exactly twice (each time with a different neuron). It is possible however, that a neuron will multiply itself, or multiply the same neuron twice.

The motivation behind this architecture was inspired by the analytically derived approximations (G_1 and G_2 functions), which perform relatively well but require prime factorization and are therefore not practical for large numbers. As can be seen, G_1 and G_2 are based on the multiplication of primary factors. We believe that the real solution of the function must include multiplication components. However, a standard neural network cannot multiply two features by each other, but can only approximate such an action. Unfortunately, while such an approximation might perform well when tested on the training-set scope, its performance significantly deteriorates when tested on larger numbers. Therefore, by using the multiplication-layer model there will be actual multiplications between some neurons; this may improve the model performance, especially when predicting numbers larger than the scope of the training-set.

C. Ln-Layer Model

Another architecture introduced in this paper is the ln-layer model (see Figure 3). Similar to the multiplication-layer model, in the ln-Layer model after several fully connected layers, the neurons are divided into two groups. The first group of neurons is fully connected to the next layer (using ReLU activation). The second group is activated first by ReLU and then by the natural logarithm, and its output serves as an input to a separate fully connected layer. That layer is then activated by an exponential function. The two groups are then concatenated and continue with a simple fully connected architecture. Converting the neuron data to natural logarithm and then back by an exponential function enables the network to multiply multiple neurons by each-other.

In addition to the previously introduced architectures, we consider an additional method that make use of Goldbach's function approximation formula developed by Markakis et al. [16], G_4 (see equation 6). This formula was selected because it is the most accurate among the analytically derived formulas that do not require prime factorization.

D. Normalization of the Result

We normalize the result by G_4. To this end, each of the previously presented models (basic model, multiplication-layer model and ln-layer model), is trained to predict the value of Goldbach's function divided by G_4 (rather than simply predicting the value of the Goldbach's function). In addition to the obvious benefit from normalizing the prediction value, dividing by G_4 will likely improve the performance of all models when tested on values that are not in the same distribution area. This is because the model learns the relation between Goldbach's function and G_4; we believe that this relation is less prone to changes (than the actual Goldbach's function) as the number grows. We denote this method by adding '+n' to the original model (that is, the Basic-MLP model with normalized result is denoted Basic-MLP+n, and the Multiplication-layer and Ln-layer models are denoted Multiplication-layer+n and Ln-layer+n, respectively).

E. Results

Test range	**Basic-MLP**	**Multiplication-layer**	**Ln-layer**
4-5	0.4	**0.2**	0.29
5-6	1.58	**1.19**	1.58
6-7	7.65	**4.67**	6.43
7-8	32.53	**12.6**	18.49
8-9	96.42	**23.8**	46.3
9-10	178.88	**38.71**	107.75

TABLE I: MSE of the basic-MLP model, Multiplication-layer model and the Ln-layer model. Note that all models were trained on numbers up to 4 million and were tested on numbers between 4 and 10 million. **All the numbers are in millions**

Table I presents the performance of our models in comparison to G_4 (equation 6), in terms of mean squared error (MSE). All models were trained on numbers up to 4 million, and were tested on numbers up to 10 million. The numbers in the overlapping part (i.e. up to 4 million) were carefully split into a training-set (90% of the data) and a test-set (10% of the data). As expected, the MSE grows as the numbers grow. As we hypothesised, both the Multiplication-layer and the Ln-layer models outperformed the basic-MLP. Somewhat surprisingly, the Multiplication-layer outperformed the Ln-layer model.

Table II presents the MSE of the models normalized by G_4. As expected, the performance of all the methods significantly improved (see Figure 4). Interestingly, the Ln-layer+n model, gained the lead, and outperformed both basic-MLP+n and multiplication-layer+n. However, we would like to note that the derivation of the G_4 formula is not trivial, and therefore, the Multiplication-layer architecture may be used in different domains, in which such an analytically derived function (such as G_4) is not available.

We compare the performance of the Multiplication-layer and the Ln-layer+n models with the performance of the basic MLP model and the G_4 function (which does not require factorization). As depicted by Table III, the Multiplcation and the Ln-layer+n methods outperform the basic MLP and G_4 at all ranges. The Multiplciation model, which does not use any analytically derived formula, outperforms the Ln-layer+n on the 0-4M range (on the test-set), and achieves very close performance in the 4-5M range.

In addition, we test the performance of the Ln-Layer+n model when trained on a subset of the training set. Recall that the training set is composed of 90% of the even numbers between 0 to four million. Table IV presents the performance of the Ln-Layer+n when trained on 100%, 50%, 10% and 1% of the training set. As expected, the performance of the model is much better when trained on a larger data-set; however, even with only 1% of the training set, the Ln-Layer+n performs

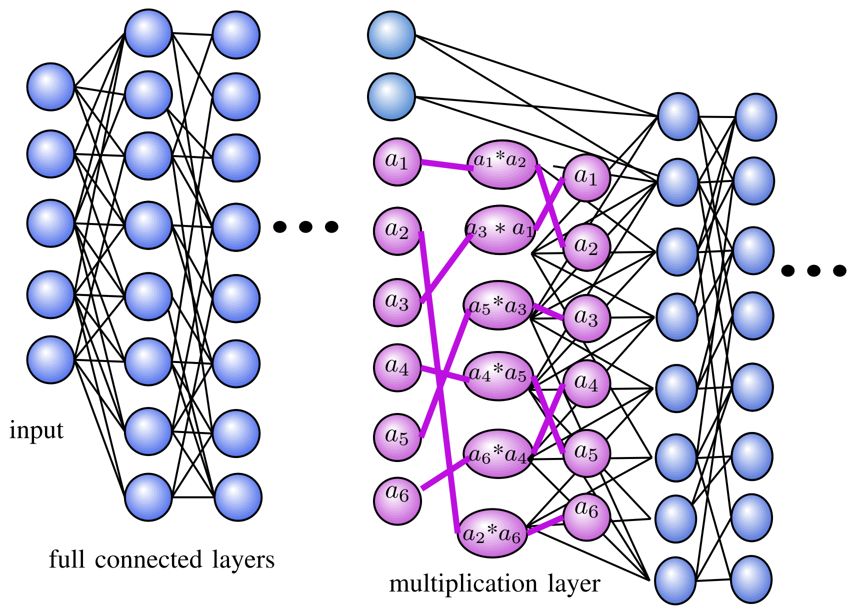

Fig. 2: An illustration of the multiplication architecture. The arcs between the purple neurons ($a_1 - a_6$) demonstrate the multiplication. Note that each of the neurons is paired exactly twice with another neuron. The other neurons (which appear in blue) follow a standard fully connected architecture.

Test range	**Basic-MLP + n**	**Multiplication-layer + n**	**Ln-layer + n**
4-5	**0.17**	0.21	0.19
5-6	0.46	0.47	**0.37**
6-7	1.47	2.08	**1.41**
7-8	4.15	8.5	**3.72**
8-9	10.54	20.95	**6.54**
9-10	23	35.16	**10.79**

TABLE II: MSE of models normalized by G_4. **All numbers are in millions**

Test range	**Multiplication**	**Ln-layer + n**	**basic MLP**	G_4
0-4 (test-set)	**0.034**	0.052	0.1	22.5
4-5	0.2	**0.19**	0.4	73
5-6	1.19	**0.37**	1.58	105.6
6-7	4.67	**1.41**	7.65	140.69
7-8	12.6	**3.72**	32.53	180.14
8-9	23.8	**6.54**	96.42	223.55
9-10	38.71	**10.79**	178.88	270.89

TABLE III: MSE of the Multiplication-layer model and the Ln-layer+n model compared to the MSE of G_4. **All the numbers are in millions**.

Test range	100% **of training set**	50% **of training set**	10% **of training set**	1% **of training set**	G_4
0-4 (test-set)	**0.052**	0.13	0.2	1.8	22.5
4-5	**0.19**	0.28	0.4	7.1	73
5-6	**0.37**	0.76	1.2	11.3	105.6
6-7	**1.41**	2.1	3.7	20.1	140.6
7-8	**3.72**	5.13	8.18	29.6	180.1
8-9	**6.54**	10.4	15.2	48.8	223.5
9-10	**10.79**	17.96	23.6	67.3	270.8

TABLE IV: MSE of the Ln-layer+n model when trained on 100%, 10%, and 1% of the training set, which is composed of 90% of the even numbers between 0-4 million, compared to the MSE of G_4.

significantly better than G_4 at all ranges. Finally, Table V presents the average number of Goldbach partitions for each of the ranges, the mean absolute error (MAE) values of our selected model (ln-layer+n) along with the MAE of G_4, and the error rate of our selected model. We note that the error rate is less than 5% also in the higher ranges.

V. Discussion

As stated in the introduction, Goldbach's conjecture was verified up to 4×10^{18}. This verification was performed by

Test range	Average no. of partitions	Ln-layer + n MAE	Ln-layer + n error rate	G_4
0-4 (test-set)	10.7	0.13	1.2%	3.4
4-5	22.0	0.25	1.1%	7.1
5-6	26.2	0.4	1.5%	8.5
6-7	30.3	0.7	2.4%	9.8
7-8	34.3	1.1	3.3%	11.1
8-9	38.2	1.4	3.8%	12.4
9-10	42.1	1.9	4.5%	13.6

TABLE V: The average number of Goldbach partitions for each of the ranges in thousands, the mean absolute error (MAE) values of our selected model (ln-layer+n) along with the MAE of G_4 in thousands, and the error rate of our selected model.

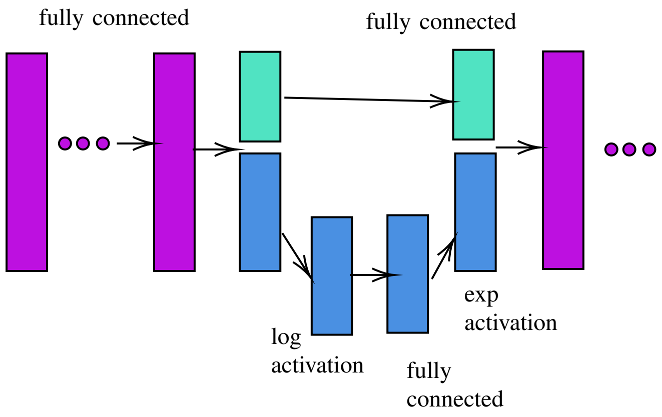

Fig. 3: An illustration of the ln-layer architecture. One of the layers is divided into two sets of neurons. The green set is fully connected to the next layer, while the blue set is first activated by the natural logarithm, then fully connected and activated by an exponent.

using an exhaustive search (see also [18] for an implementation requiring minimal space). Our approximation model may allow a selective search method in which Goldbach's conjecture can be verified only for suspicious numbers according to our model. In other words, only numbers that our model predicts will have a very low number of partitions. This approach can also be used to find numbers that violate the lower-bound proposed by [15]. The methods presented in this paper may allow training on smaller numbers while performing the selective search on larger numbers.

In previous work [7], we used the 10 least significant digits of each base presentation. This model performed well when trained and tested on numbers up to 4 million. However, when this model was tested on large numbers we observed a large gap between its error on $5-6$ million and $6-7$ million. Namely, when tested on numbers between 4 and 5 million the error was 0.35 million, when tested on $5-6$ million it was 0.77 million, but when tested on $6-7$ million, it rose to 128.68 million. This gap is attributed to the fact that in the $0-4$ million region (the training data), the 9th and 10th digits in septenary (base-7) were constantly 0, and the 8th digit in septenary increased monotonically. This is because the number $4,000,000$ is written as $45,666,544$ in septenary (with 0 in the 9th and 10th digits). This caused the weights associated with the 9th and 10th septenary digits to remain with their initial assigned values (noise). Furthermore, since the 8th digit of base-7 increased monotonically, the trained model usually assigned higher values to numbers that had a higher value in the 8th septenary digit. That is, the model did not see any large numbers with a low value, or a 0, in the 8th septenary digit. Therefore, when tested on numbers with 0 in the 8th septenary digit, the model predicted a small value (despite the given number being large). For example, 6.5 million is written as $106,151,303$ in septenary, and therefore has a 0 in the 8th digit location. To overcome this problem and avoid this type of over-fitting, in this paper, we used only the 6 least significant bits. Therefore, in all models presented in this paper, the error rose gradually. For example, in the basic-MLP model presented in this paper, which has an architecture similar to the model presented in [7], the MSE was 0.4 for $4-5$ million, 1.58 for $5-6$ million, and 7.65 for $6-7$ million.

In this paper though we do not attempt to use a deep neural network (DNN) to actually prove a theorem (which is a completely different field), we believe that DNN can be used to help solving open mathematical problems in different ways. The first method is by finding counterexamples (e.g. using selective search) and the second by analyzing the features that the DNN considers when computing its values. In our example we show that using different base representations (which can be computed inexpensively) is very useful for approximating Goldbach's function, (as opposed to the expensive prime factorization computation which was previously known). Future advancements in explainable neural networks, might help shed additional light on this problem when applied to our results. While deep learning has shown great success in many different fields [19], [20], [21], we believe that the success shown in this paper related to an open mathematical problem in number theory, is a big step and should not be disregarded as being merely another deep learning application. Our work may lead to a new paradigm of using deep learning (or machine learning in general) to solve mathematical problems such as prime factorization, friendly numbers, finding prime twins and many similar problems, which may currently seem out of the scope of deep learning methods.

VI. Conclusions

Goldbach's conjecture and Goldbach's function have remained open mathematical questions for over two and a half centuries. There have been several analytic attempts to approximate Goldbach's function, but unfortunately, these approximations either do not work well in practice or require prime factorization (prime decomposition) which is a hard

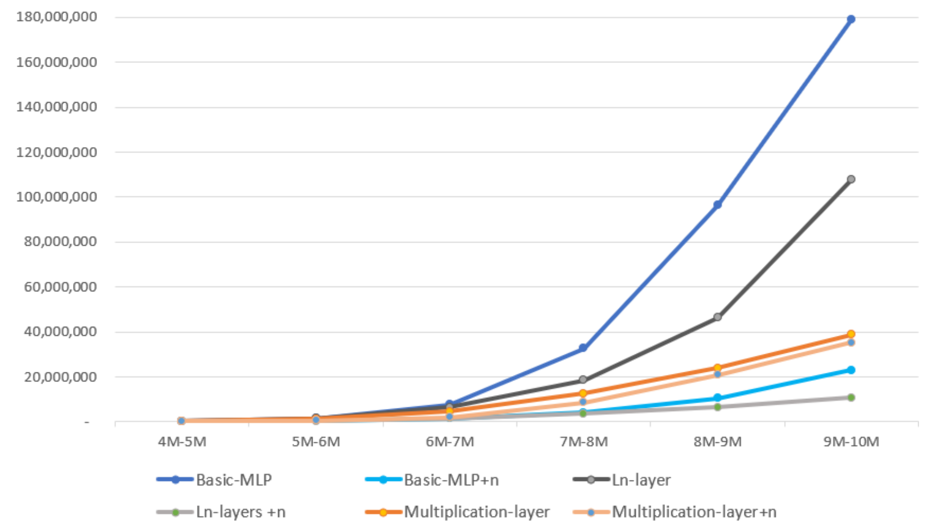

Fig. 4: This plot compares the MSE of the three models (Basic-MLP, Multiplication-layer and Ln-layer) with the MSE of their normalized versions (lower is better). As depicted in the figure, all models gain from this normalization.

problem. In previous work, we developed a basic multi-layer perceptron and show that this simple model performs well when trained and tested on numbers up to 4 million. However, the model's performance significantly deteriorates when trained on smaller numbers (up-to 4 million) but tested on larger numbers ($4-10$ million).

To overcome this problem, in this work we presented two novel deep learning architectures. In these architectures we introduced two types of multiplication layers; in the first architecture some of the neurons of a specific layer are multiplied by each-other (the Multiplication-layer model). In the second architecture a log activation is performed at the output of a set of neurons that is followed by a fully connected layer with an exponential activation (the Ln-layer model). We showed that both architectures significantly outperform the basic multi-layer perceptron when trained on smaller numbers and tested on larger numbers. We further improved the performance of the deep learning architectures by normalizing the model's output by a known analytically derived estimation (G_4).

References

[1] C. Goldbach, Letter to L, Euler 7 (1742) 1.

[2] T. Oliveira e Silva, S. Herzog, S. Pardi, Empirical verification of the even Goldbach conjecture and computation of prime gaps up to 4×10^{18}, Mathematics of Computation 83 (288) (2014) 2033–2060.

[3] H. F. Fliegel, D. S. Robertson, Goldbach's comet: the numbers related to Goldbach's conjecture, Journal of Recreational Mathematics 21 (1) (1989) 1–7.

[4] G. H. Hardy, J. E. Littlewood, Some problems of diophantine approximation: The lattice-points of a right-angled triangle, Proceedings of the London Mathematical Society 2 (1) (1922) 15–36.

[5] J. Baker, Excel and the Goldbach comet, Spreadsheets in Education (eJSiE) 2 (2) (2007) 2.

[6] I. Goodfellow, Y. Bengio, A. Courville, Y. Bengio, Deep learning, Vol. 1, MIT press Cambridge, 2016.

[7] A. Stekel, M. Shukrun, A. Azaria, Goldbach's function approximation using deep learning, in: 2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI), IEEE, 2018, pp. 502–507.

[8] Y. Wang, The Goldbach Conjecture, Vol. 4, World scientific, 2002.

[9] H. A. Helfgott, The ternary Goldbach conjecture is true, arXiv preprint arXiv:1312.7748 (2013) 1–79.

[10] I. O. Bado, New discovery on goldbach, International Journal of Progressive Sciences and Technologies 13 (2) (2019) 216–221.

[11] C. Liu, A study of relationship among goldbach conjecture, twin prime and fibonacci number., IJ Network Security 19 (3) (2017) 406–412.

[12] A. Berdondini, The importance of finding the upper bounds for prime gaps in order to solve the twin primes conjecture and the goldbach conjecture, arXiv preprint arXiv:2002.07174 (2020) 1–10.

[13] A. Granville, Refinements of goldbach's conjecture, and the generalized riemann hypothesis, Functiones et Approximatio Commentarii Mathematici 37 (1) (2007) 159–173.

[14] J. P. Buhler, H. W. Lenstra, C. Pomerance, Factoring integers with the number field sieve, in: The development of the number field sieve, Springer, 1993, pp. 50–94.

[15] C. Provatidis, E. Markakis, N. Markakis, Rule of thumb bounds in Goldbach's conjecture, American Journal of Mathematical Analysis 1 (1) (2013) 8–13.

[16] E. Markakis, C. Provatidis, N. Markakis, Some issues on Goldbach conjecture, Number Theory 29 (2012) 1–30.

[17] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, Proceedings of the 3rd International Conference on Learning Representations (ICLR) (2014) 1–15.

[18] J. Richstein, Computing the number of goldbach partitions up to 5 10 8, in: International Algorithmic Number Theory Symposium, Springer, 2000, pp. 475–490.

[19] Y. Lv, Y. Duan, W. Kang, Z. Li, F.-Y. Wang, Traffic flow prediction with big data: a deep learning approach, IEEE Transactions on Intelligent Transportation Systems 16 (2) (2015) 865–873.

[20] A. A. Cruz-Roa, J. E. A. Ovalle, A. Madabhushi, F. A. G. Osorio, A deep learning architecture for image representation, visual interpretability and automated basal-cell carcinoma cancer detection, in: International Conference on Medical Image Computing and Computer-Assisted Intervention, Springer, 2013, pp. 403–410.

[21] B. Alipanahi, A. Delong, M. T. Weirauch, B. J. Frey, Predicting the sequence specificities of dna-and rna-binding proteins by deep learning, Nature biotechnology 33 (8) (2015) 831.