PREDICTING ZEROS OF THE RIEMANN ZETA-FUNCTION USING MACHINE LEARNING MODELS

JENNIFER KAMPE, HUAN QIN AND ARTEM VYSOGORETS

ABSTRACT. In this paper, we evaluate the predictive performance of Neural Network Regression in locating non-trivial zeros of the Riemann Zeta-function. The problem of predicting zeros is reformulated as a time-series prediction task. The initial pool of candidate features is identified via a theoretical study of the properties of the Zeta-function as well as by consulting previous studies. Model design is implemented as a six stage process consisting of: (i) input selection, (ii) data splitting, (iii) model architecture selection, (iv) model structure selection, (v) model calibration, and (vi) model validation. The range of Zeta-function zeros used is 1001 to 100,000; model training is performed on data with 79,200 observations (80%) for each of the 50 feature variables selected. Multilayer Perceptron (MLP) and Recurrent Neural Network (RNN) architectures are chosen and implemented; these two neural networks are evaluated against a Support Vector Regression (SVR). SVR, MLP, and RNN models yield predictive accuracies of 0.9824, 0.9907, and 0.9953 respectively, measured as test data R-squared. The RNN outputs superior predictions of the actual ζ-function zeros with residuals typically not exceeding 1% of the separation between consecutive values.

1. INTRODUCTION

The Riemann Zeta-function (ζ-function) is one of the most celebrated examples of Dirichlet L-functions and has been extensively studied by mathematicians for almost 200 years. This captivating function connects two seemingly unrelated branches of mathematics—number theory and complex analysis, and has further applications in physics and probability. The importance of the ζ-function resides in its profound links to prime numbers and its potential to provide key insights into their distribution. Of particular interest is the conjecture that all non-trivial zeros of the ζ-function lie on the *critical line*: $Re(s) = \frac{1}{2}$. This problem is widely known as *Riemann Hypothesis*. All computations up to the present have confirmed the hypothesis; yet, a formal proof remains elusive, while an archive of incorrect proofs grows regularly [45].

Given the complexity of the problem, non-parametric machine learning offers an alternate means of exploring the Riemann ζ-function. Artifical Neural Networks (ANNs) constitute a powerful class of machine learning algorithms inspired by the interactions between neurons in biological brains. By concatenating multiple layers of simple transformations, neural networks are capable of representing highly complicated functions. Although first introduced by neurophysiologist Warren McCulloch and logician Walter Pitts in the early 1940's, Artifical Neural Networks did not initially experience widespread adoption due to the computational cost of implementation [25]. In recent years, however, as technological progress has made sufficient computational resources available, ANNs have begun to attract the attention of researchers from a wide spectrum of disciplines.

There has been a great deal of research into the suitability of neural networks for implementing various classes of real-valued functions. *Cybenko's theorem* provides a key result in this area, demonstrating theoretically that any continuous real-valued function can be modeled to the desired accuracy using a multi-layer neural network with sigmoid activation function [11]. While the ζ-function itself is not real valued, the closely related Z-function in fact is, suggesting a possible application of Cybenko's Theorem. Similar theorems have

Key words and phrases. Riemann Zeta-function, Riemann Hypothesis, Neural Network Regression.

been proven for radial basis neural networks [30, 33]. Mhaskar gives a constructive proof for creating optimal single hidden layer neural network approximations of sufficiently differentiable real analytic functions [27]. Petersen and Voigtlaender [32] extend this result to the implementation of piece-wise continuous functions on $\mathbb{R}^d$ via neural networks using a rectified linear unit (ReLU) activation function; they present a theoretical lower for optimal depth and complexity, and verify this lower bound experimentally.

While there has been much work justifying the application of neural networks to real-valued functions, it is unclear how these results might generalize to complex functions such as the ζ-function. While complex-valued neural networks have great potential to expand research in this area, the implementation of gradient descent is significantly complicated by non-analytic derivatives associated with complex cost functions [38]. In this study, we avoid these complications by transforming the modeling of the complex ζ-function into an equivalent real-valued time-series. Because there is ample precedent for time-series forecasting in the realm of neural networks [12, 10, 47], this allows us to draw on a much deeper body of study.

At present, there is only one known application of neural networks to the task of modeling the ζ-function: Shanker [40] uses an unspecified neural network regression to predict the locations of ζ-function zeros. While Shanker's model successfully replicates the mean and standard deviation of the target data, standard validation metrics are not presented. As a result, it is impossible to compare Shanker's neural network to other available techniques for estimating the zeros of the ζ-function.

In the paper that follows, we build upon the work of [40] by adapting multiple neural network architectures to predict the locations the of non-trivial ζ-function zeros. We expand upon this work by enlarging the feature set to take greater advantage of the relationship between the Riemann-Siegel Z-function and the ζ-function. Additionally, we introduce the use of Support Vector Machine Regression (SVM/SVR) for the task. While Support Vector Machine—another important class of machine learning algorithms—have not, to our knowledge, been applied to the study of the ζ-function, there is strong precedent for their use in time-series analysis [43, 28]. Thus, SVM is an obvious candidate for the reformulated ζ-function problem. We evaluate relative and absolute performance metrics for all of the above models.

2. Background

In this section, the needed background for discussing the Riemann Zeta-function and relevant machine learning techniques is provided. In section 2.1, we introduce the Riemann ζ-function and justify the use of Z-function for studying its behavior along the critical line. Additionally, we introduce several related variables which will reappear in our feature set. In section 2.2, we provide a brief introduction to the theory and implementation of Artificial Neural Networks and Support Vector Machines.

2.1. **Introduction to the Riemann Zeta-function.**

2.1.1. *Definition and Key Properties.*

First introduced by Bernhard Riemann in 1859, for $s \in \mathbb{C}$ with $Re(s) > 1$, the Riemann Zeta-function $\zeta(s)$ is defined as the convergent series

$$\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s}. \tag{2.1}$$

More than 100 years earlier, Leonhard Euler studied a real-variable prototype of this function with $s \in \mathbb{R}$, proving that $\zeta(2) = 1 + 1/4 + 1/9 + 1/16 \ldots = \pi^2/6$. In doing so, Euler solved what is known as the Basel

problem; he later provided expressions for the value of the ζ-function at all positive even integers. In 1748, Euler made another important discovery—the *Euler's product* formula, providing the first link between the ζ-function and prime numbers [31]:

$$\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s} = \prod_{p} \left(1 - \frac{1}{p^s}\right)^{-1}, \tag{2.2}$$

where the product is taken over all prime numbers p.

The next breakthrough came in 1859 with the publication of Riemann's paper "On the Number of Primes Less Than a Given Magnitude." Whereas the original series definition of the Riemann ζ-function is valid only in the half plane $Re(s) > 1$, Riemann *analytically continued* it to a meromorphic function on $\mathbb{C}$ with a simple pole at $s = 1$ and obtained the following functional equation [2]:

$$\zeta(s) = 2^s \pi^{s-1} \sin\left(\frac{\pi s}{2}\right) \Gamma(1-s)\zeta(1-s). \tag{2.3}$$

It can be seen that ζ-function vanishes at negative even integers, referred to as *trivial zeros*. Riemann hypothesized that all other (*non-trivial*) zeros lie on the critical line $Re(s) = \frac{1}{2}$ (the famed Riemann Hypothesis). It has been rigorously shown that no non-trivial zero lies outside of the *critical strip*, which is given by $0 < Re(s) < 1$ [42].

In order to facilitate further exploration of ζ-function, we turn to the Riemann-Siegel functions, which capture key aspects of the ζ- function along the critical line:

$$\theta(t) \colon = \arg\left(\Gamma\left(\frac{1}{4} + \frac{it}{2}\right)\right) - \frac{\ln \pi}{2} t \tag{2.4}$$

$$Z(t) \colon = e^{i\theta(t)} \zeta\left(\frac{1}{2} + it\right), \tag{2.5}$$

defined for real t. The second of these, the Riemann-Siegel Z-function (2.5) allows us to study the ζ-function via a simpler function $Z(t)$, which is real-valued on the specified domain [18, 26]. Additionally, since $|Z(t)| = \left|\zeta\left(\frac{1}{2} + it\right)\right|$, zeros of Z-function coincide with zeros of the ζ-function on the critical line. Therefore, locating zeros of ζ-function is equivalent to studying the sign changes of $Z(t)$.

A key advantage of working with the Z-function rather than the ζ-function is the relative ease of evaluation. Before Carl Siegel provided the asymptotic expansion of the Z-function in 1932, the *Euler-Maclaurin summation* method was generally used in approximation of the ζ-function. While Euler-Maclaurin summation is more universal and a more accurate tool, evaluation requires the computation of $|t|$ terms. Evaluation via the asymptotic expansion of Z-function, in contrast, requires only $\sqrt{|t|}$ terms. This asymptotic expansion is known as the *Riemann-Siegel formula* [34]:

$$Z(t) = 2 \sum_{n=1}^{N} \frac{\cos(\theta(t) - t\ln(n))}{\sqrt{n}} + R, \tag{2.6}$$

where $N = \lfloor \sqrt{t/2\pi} \rfloor$, and R is a remainder, the bounds of which are provided by Rosser et al. [36].

2.1.2. *Gram Points.*

Given the above properties of the Z-function, we now explore its connection to ζ-function in more detail. Applying Euler's formula to equation (2.5), we separate $\zeta\left(\frac{1}{2} + it\right)$ into its real and imaginary parts, which

we denote as $A(t)$ and $B(t)$, respectively:

$$\zeta\left(\frac{1}{2}+it\right) = Z(t)e^{-i\theta(t)} = Z(t)[\cos(\theta(t)) - i\sin(\theta(t))], \tag{2.7}$$

$$A(t) \colon = Re\left(\zeta\left(\frac{1}{2}+it\right)\right) = Z(t)\cos(\theta(t)), \tag{2.8}$$

$$B(t) \colon = Im\left(\zeta\left(\frac{1}{2}+it\right)\right) = -Z(t)\sin\left(\theta(t)\right). \tag{2.9}$$

Note that both $A(t)$ and $B(t)$ vanish at zeros of the Riemann-Siegel Z-function. Thus, obtaining values of t such that $A(t) = B(t) = 0$ could potentially helpful to locating zeros of the Z-function itself. When $B(t) = 0$, either $Z(t) = 0$ and, hence, $\zeta\left(\frac{1}{2}+it\right) = 0$, or $\sin(\theta(t)) = 0$, which requires $\theta(t)$ to be a multiple of π. This observation inspires the definition of a *Gram point* g_n, which is a unique solution to the equation $\theta(t) = (n-1)\pi$. The existence and uniqueness of such solutions are ensured by the monotonicity of the θ-function [3]. A key observation is that at a Gram point g_n, one has [21]

$$\zeta\left(\frac{1}{2}+ig_n\right) = A(g_n) + iB(g_n) = Z(g_n)\cos((n-1)\pi) = (-1)^{n-1}Z(g_n) \tag{2.10}$$

According to the above equation, if g_n and g_{n+1} are two Gram points such that $A(g_n)$ and $A(g_{n+1})$ have the same sign, then $Z(g_n)Z(g_{n+1}) < 0$, and so $Z(t)$ vanishes in (g_n, g_{n+1}) at least once. Therefore, Gram points allow us to detect intervals which contain roots of the Riemann-Siegel Z-function, and, consequently, non-trivial zeros of the Riemann ζ-function on the critical line. This method was invented by a Danish mathematician Jorgen Gram in 1902, who used it to show that imaginary parts γ_n of the first 15 non-trivial zeros of ζ-function alternate with the first 15 Gram points g_n (i.e. $g_{n-1} < \gamma_n < g_n$). The conjecture that generalizes this pattern is now referred to as *Gram's law*, despite the fact that Gram himself claimed it to be false. In fact, the first violation of this rule was found by Hutchinson [17] and occurs at the 127-th Gram point [21]. While there Gram points may not be a perfect predictor of the zeros of the ζ-function, the still possess a high degree of relevance to the problem.

The importance of Gram points in the context of the ζ-function inspired us to introduce their "dual" sequence. Recall that Gram points are defined to make the imaginary part $B(t)$ of $\zeta\left(\frac{1}{2}+it\right)$ vanish. Although no known sources have utilized it, we investigate a sequence of points that make $\cos(\theta(t))$ (and, consequently, the real part $A(t)$) equal zero. In the absence of any accepted notation for these points, we refer to them as *coGram points* in order to emphasize their similarity with Gram points. Thus, we define n-th coGram point to be a unique solution to $\theta(t) = (n-1)\pi + \pi/2$. This data sequence will play a critical role in the process of selecting features for our regression models.

2.1.3. *The Distance Variable.*

In predicting the imaginary parts γ_n of the non-trivial zeros of ζ-function on the critical line, we translate the problem into a time-series forecasting task. However, this interpretation requires the target variable to be stationary, i.e. it must have constant mean and variance along all indices. Since the sequence $\{\gamma_n\}$ of ζ-function zeros is monotone increasing, failure to "difference" the data could lead to upward bias in accuracy via autocorrelation. As a result, we follow the method of Shanker[40]: instead of predicting zeros γ_n directly, we predict differences $\gamma_n - g_{n-1}$ between zeros and Gram points, which we refer to as the *distance* variable. Because Gram points g_n and zeros γ_n are asymptotically equivalent with $g_n \sim \gamma_n \sim 2n\pi/\ln(n)$ [21], it is indeed possible to obtain a mean and variance stationary time-series by taking differences $\gamma_n - g_{n-1}$. As is

evident in Figure 1, the distance variable can indeed be represented by a time-series with invariant mean (0.3746 for the first 100,000 values).

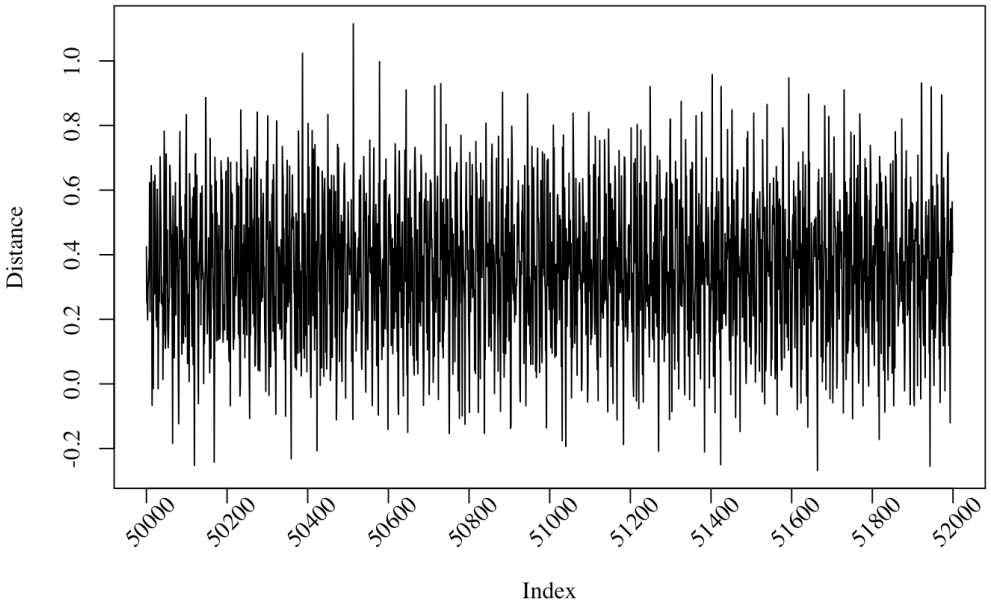

FIGURE 1. A subset of 2000 distance values between $n = 50000$ and $n = 52000$.

Predicting the distance variable is conceptually equivalent to predicting zeros, since the calculation of any finite number of Gram points to a desired accuracy is a relatively easy task [40]. In this study, we calculate first 100,000 Gram points accurate to the 7-th digit after decimal point using the Bisection method in Python. The first 100,000 coGram points are computed in the exact same way and to the same accuracy. In our implementation, the Riemann-Siegel Theta-function is approximated by the first five terms of its asymptotic expansion, which can be derived from the Stirling's expansion of the Gamma function [7]:

$$\theta(t) = \frac{t}{2}\ln\left(\frac{t}{2\pi}\right) - \frac{t}{2} - \frac{\pi}{8} + \frac{1}{48t} + \frac{7}{5760t^3} + \dots \tag{2.11}$$

2.2. **Introduction to Neural Network Regression.**

The non-parametric nature of neural network regression makes it a favorable option for learning tasks in which the functional form is unknown—as is the case with the Riemann ζ-function. In the sections that follow, we give a brief introduction to the architecture of neural networks, highlighting the key considerations for the study of the ζ-function.

Artificial Neural Networks (ANNs) constitute a class of machine learning models inspired by the information processing seen in biological neural networks like the human brain. Neural network models can be divided into two major categories: data classification and Neural Network Regression (NNR). The latter is utilized here in our effort to represent the zeros of the Riemann ζ-function, which we reformulate as a time-series problem using the distance variable. In its application to time-series data, Neural Network Regression

can be thought of as a generalization over multiple classic forecasting models (bilinear, autoregressive, non-parametric, etc.) [40]. Unlike these methods, however, NNR requires few assumptions to be made regarding the nature of the system of interest, which makes it more universal and usually more effective in pattern recognition [9].

2.2.1. *Forward Propagation in a Feedforward and Recurrent Neural Networks.*
The standard architecture of a neural network consists of layers of elementary processing units known as neurons, linked together in a fashion that depends on type and purpose of a particular ANN. The first layer is called an *input layer* and contains as many neurons as there are different features in the input data. Next, data is processed through neurons of custom-sized *hidden layers*. ANNs with two or more hidden layers are called *deep networks*, while their implementation is referred to as *deep learning*. In all cases, the last layer is called an *output layer* and contains one neuron for each feature predicted.

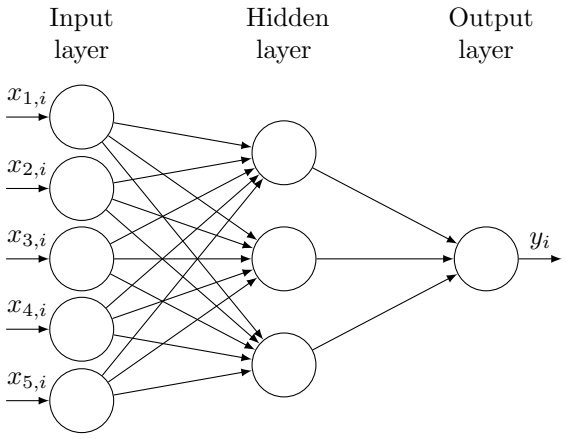

FIGURE 2. Standard architecture of a single layer densely connected MLP.

Upon entering the input layer of a neural network, data is transformed into output by propagating through neurons via connections (edges) between them. The topology of these connections is determined by the type and architecture of an ANN. In *feedforward* neural networks, data propagates strictly towards the output layer, i.e. neurons output exclusively to neurons of subsequent layers.

The simplest and the most common example of a feedforward network is Multilayer Perceptron (MLP)—the first model to be implemented in this paper. It is common practice for MLPs to have all neurons of neighboring layers connected. In other words, in most MLPs any two subsequent layers form a complete bipartite graph with these layers as partitions (Figure 2). Neural networks having such a topology are often referred to as *densely* connected.

The data processing structure of ANNs can be interpreted using analogies from biological neural networks. In human brains, neurons communicate through electrochemical impulses, the intensity of which are determined by the strength of synaptic connections between them. In turn, the strength of input signals determines their ability to *excite* neurons, which have a certain activation threshold (~ 55mA). Neurons fire an output signal to successive neurons if an incoming signal exceeds this threshold, and remain dormant otherwise. Thus, the biological neuron is a switch, which is either on or off. In the context of ANNs, this

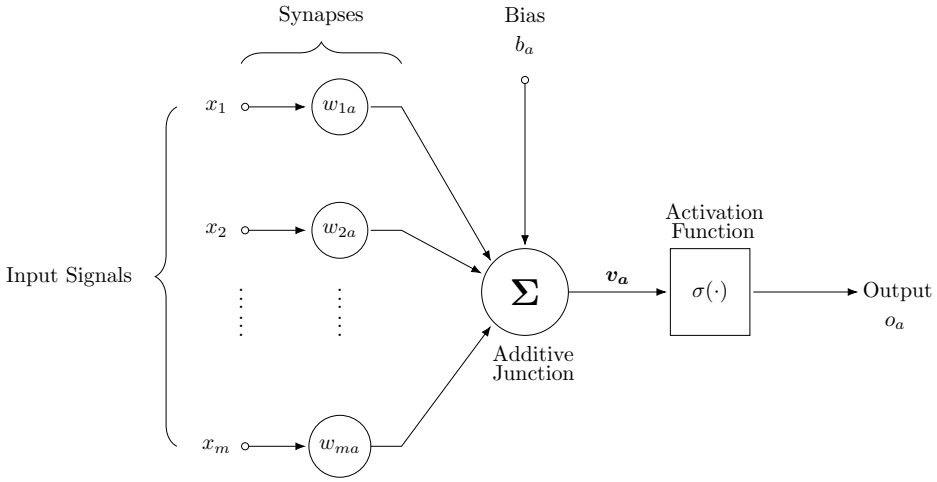

FIGURE 3. Feedforward propagation in a single hidden layer neuron.

behavior is modeled by introducing bias b_a and an activation function $\sigma = h(t)$, where $h(t)$ is a Heaviside step function (also known as *binary* function) activated at $t = 0$.

Consider the a-th neuron in the L-th layer of a given ANN, which we will denote by n_a^L. Suppose that according to the topology and architecture of the ANN in question, n_a^L receives signals from m neurons and sends signals to q other neurons. Then, the neuron n_a^L can be described by two parameters: an m-dimensional vector of *weights* $(w_{1a}, w_{2a}, \ldots, w_{ma})^T$ and a *bias* term b_a. Given that n_a^L receives signals x_1, x_2, …, x_m from neurons $n_1, n_2 \ldots, n_m$ in the previous layer, the signal o_a sent by n_a^L to each of the q output connections is calculated using the following formula [44]:

$$o_a = \sigma_L \left(\sum_{i=1}^{m} w_{ia} x_i - b_a \right), \tag{2.12}$$

where σ_L is an activation function specified for all neurons of layer L, and $\sum_{i=1}^{m} w_{ia} x_i - b_a$ is referred to as the *input signal*. This transformation is depicted in Figure 3.

2.2.2. *Backprogagation in Neural Networks.*

The process of *learning* in artificial neural networks refers to the iterative optimization of the weights and biases described above. This optimization is accomplished via the introduction of a large amount of *training data* to the ANN. The ANN then uses one of the two major learning strategies. *Supervised learning* is implemented by allowing neural networks to access values of the target output variable associated with each observation of the training data. In this learning paradigm, machines "know" the desired output y_i, which allows them to estimate the accuracy of their output $\hat{y}_i$ and alter weights to minimize error function $E = E(y_i - \hat{y}_i)$. On the other hand, in *unsupervised learning*, the networks are given no information about the desired output, and so rather than responding to feedback (e.g., prediction error), such networks attemept to group data with common patterns or features together [44].

Since NNR utilizes supervised learning, we will focus on details of its implementation. In 1974, Paul Werbos developed an algorithm called *backpropagation*, which is now one of the most widely used supervised learning techniques for ANNs. In backpropagation, ANNs minimize an error or *loss* function E through the iterative process of *gradient descent*— adjusting weights proportional to the negative of the partial derivative of E [24]. Thus, the result of each iteration (propagation of a single observation from training data through

the ANN) is the calculation of all partial derivatives $\partial E/\partial w_{ij}$ and transformation $w_{ij} \to w_{ij} - k \cdot \partial E/\partial w_{ij}$ applied to each weight w_{ij}, where k is a positive constant known as *learning rate*.

Through this same procedure, bias terms are also updated, resulting in the simultaneous adjustment of both parameters within a single iteration. A set of iterations marked by the propagation of each observation from training data exactly once is called an *epoch*; *backpropagation* refers to the complete optimization of weights and biases.

Note that the backpropagation algorithm described above is incompatible with ANNs that consist of neurons of purely biological design. In order to fully simulate biological neurons, the Heaviside function should be used. However, since $E = E(y_i - \hat{y_i})$ is necessarily a function of the output $\hat{y_i}$, its partial derivatives do not exist unless a differentiable function is used as the activation function σ. To resolve this incompatibility, modern ANNs violate the "binary-switch" nature of neurons and use continuous non-linear activation functions, such as logistic (sigmoid), hyperbolic tangent (tanh), or rectified linear unit (ReLU).

2.2.3. *Theoretical Comparison of ANNs and SVR.*

Support Vector Machine Regression refers to a second class of algorithms popular for supervised machine learning tasks. SVR algorithms use kernel functions to derive efficient solutions to non-linear learning problems. For an overview of the basic workings of SVR, as well as the key algorithms used today, see Smola [41]. To summarize, the characteristic features of SVR are: (i) the use of a kernel function to transform the data into a higher dimensional feature space where linear separation is possible, and (ii) the use of a convex objective function permitting a global optimal solution via the Lagrangian. Thissen [43] argues that the use of a convex objective function permitting a global optimal solution a key advantage of SVR relative to ANNs. Additionally, convergence on the global optimal solution conveys increased generalizability, reducing the overfitting problem often encountered in ANNs.

As a kernel-based machine, SVR can be characterized as a shallow architecture in that it contains only one layer of trainable coefficients (i.e. features)—analogous to a single layer neural network. Bengio [5] contrasts kernel methods with deep architectures, such as neural networks with multiple hidden layers. He finds shallow architectures to be relatively inefficient for learning tasks which demand the representation of complex functional forms due to the extent of computations required.

Further, Bengio argues that deep architecture neural networks, with their greater number of tunable hyperparameters, offer greater flexibility and improved predictive validity for such functions [5]. However, the potential for more flexible representation comes at the cost of an increased propensity to get stuck in poor local minima and greater risk of overfitting. Thus, there are compelling reasons to consider both methods for the present task.

3. Neural Networks: Application to Riemann Zeta-function

Given the ability of ANNs to provide efficient representations of complex functions, the present study applies these methods to the modeling of the Riemann Zeta-function. Specifically, we design two ANNs and, for comparison, a Support Vector Machine Regression to predict time-series obtained by taking differences $\gamma_n - g_{n-1}$, where γ_n is the imaginary part of the n-th zero located on the critical line and g_{n-1} is the $(n-1)$-th Gram point.

3.1. **Neural Network Design Protocol.** The process of building our models is guided by the protocol proposed by Wu [46]. Our implementation of this protocol consists of six phases: (i) input selection, (ii) data splitting, (iii) model architecture selection, (iv) model structure selection, (v) model calibration, and (vi) model validation. While the data splitting and input selection procedures described below apply to both SVR and ANN regression models, all subsequent stages refer exclusively to ANNs.

In the model-design sections to follow, we perform repeated sensitivity analysis, evaluating the sensitivity of model output to changes in the inputs including features, architecture, and hyperparameters. In order to select the optimal models, and finally, validate our models we rely on the following performance criteria:

(i) The coefficient of determination, R^2:

$$R^2 = 1 - \sum_{i=1}^{n} \frac{\left(y_i - \hat{y}_i\right)^2}{(y_i - \bar{y})^2}$$

where y_i, $i = 1, \ldots, n$ are the true values of the dependent variable and $\hat{y}_i$, $i = 1, \ldots, n$ are the corresponding model predictions. The coefficient of determination is a measure of relative fit that describes the proportion of total variation in the dependent variable which is explained, or eliminated by the model. R^2 ranges from 0 to 1, with $R^2 = 1$ indicating perfect model fit. Thus, a key advantage of this metric that it is easily interpretable even in the absence of additional information about the distribution of the dependent variable.

(ii) The root mean square error, $RMSE$:

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^{n} \left(y_i - \hat{y}_i\right)^2}.$$

The $RMSE$ is a measure of absolute fit which indicates the standard deviation of the residuals, i.e. the prediction errors. It is in the original units of (and should be interpreted in the context of) the dependent variable.

Additionally, the mean and standard deviation of the model-fitted values are compared to the true sample statistics of the dependent variable. This provides an overview of each model's success in terms of replicating the distribution of the dependent variable.

3.2. **Data Splitting.** Whereas linear regression is optimized for inference, the explicit goal of machine learning regression is to make predictions. It is no surprise that predictive performance is the key metric used in the evaluation of machine learning models. In order to assess the generalizability of a model and inspect it for overfitting, it is crucial that a model be tested on independent data, i.e., data different from the training data. This process of randomly dividing data examples into test and training subsets is known as *data splitting*.

Cross-Validation is a method of model evaluation in which the original data sample is partitioned into multiple training sets used for model design, and a test set used for model validation. In k-fold cross validation, a sample is partitioned into k equally sized subsamples and the model is trained k times, with each subsample being used as the test data exactly once. Thus, k-fold cross-validation offers a more robust test of generalizability by permitting all examples to enter into the test data.

Throughout our model design process, 10-fold cross validation is employed in sensitivity analysis. The resultant calibrated models are then evaluated on a 80%/20% data split: a random selection of 80% of the

shuffled data is used for training with the remaining 20% used for testing. Both test and training errors are reported using the metrics above.

3.3. **Input Selection.** Input selection refers to the key phase of model design in which the relevant features, (i.e., variables or regressors) are selected from the entire pool of candidate features. In an efficient neural network, redundant or irrelevant regressors should be associated with near-zero input layer weights as a result of training. Further, a linear transformation equivalent to Principle Component Analysis (PCA) can be performed in the input layer. As a result, it can be said that neural networks can automate feature selection via the learning of input layer weights and biases.

Nonetheless, performing feature selection exogenous to the model may confer strong advantages in terms of performance. It has been observed that the predictive power of a regression with a fixed number of observations decreases when more than a certain number of features are added—an occurrence known as Hughes' Phenomenon [16].

May et al. outline the principal considerations in selecting the optimal subset of features: relevance, computational effort, dimensionality, and training difficulty [23]. *Relevance* refers to the feature's potential to contribute to the prediction of the dependent variable. For example, a good feature cannot be noise with respect to the variable being predicted. Given a set of relevant candidate features, *computational effort* indicates the cost of running the neural network: as the number of variables increases linearly, computational difficulty increases exponentially because of the number of neuron connections. Additionally, as the dimension of the feature space increases, data becomes relatively sparse and may be insufficient to satisfactorily train the great number of connection weights produced—a phenomenon Richard Bellman named as *the curse of dimensionality* [4].

Given our ability to generate (theoretically) limitless amounts of data for the problem at hand, the greatest concern is that of training difficulty. *Training difficulty* refers to slow run time and poor predictive power resulting from the inclusion of redundant or irrelevant features in a neural network. In the simplest terms, it is a waste of time and computational effort to train weights on features with no relation to the outcome variable. Even more concerning than wasted time, however, is the fact that redundant features increase the number of local extrema in the cost function. This, in turn, can cause the process of backpropagation to converge on a poor local minima, producing a model with weak predictive power.

3.3.1. *Assembling the Data.*

While gathering our initial pool of features, we consulted a similar study carried out by Shanker [40]. Based on his work, we include Gram points, values of Z-function at Gram points, as well as terms 2 through 10 of the Riemann-Siegel formula (2.6) evaluated at Gram points. In addition, we add another 10 variables of lagged Gram sequence (lags 1-10), and 10 variables of lagged Z-values evaluated at Gram points (lags 1-10). Because our regression problem is reformulated as a time-series prediction, we also included 25 variables of lagged distance—the target variable (lags 1-25). Additionally, the initial pool of input candidates is enlarged by the following features: coGram sequence along with lags 1-10; Z-function evaluated at coGram points and lags 1-15; Z-function evaluated at successive integers and lags 1-10. In total, this amounts to 94 candidate features. All of the computations of Z-function were carried out using Mathematica programming software, which is capable of computing it to any desired accuracy. The values of the target variable, distance, are

adopted from the Odlyzko's computation of the first 100,000 zeros of the Riemann zeta function, accurate to within $3 \cdot 10^{-9}$ [29].

3.3.2. *Minimum Redundancy Maximum Relevance Feature Selection.*

From the pool of 94 candidate features described above, the final inputs for the regression models are selected based on the dual criteria of minimum redundancy maximum relevance (mRMR). For each candidate variable, relevance is assessed using both model-free and model-based methods. (i) We begin by computing the correlation between each candidate feature and distance, and sort the resulting list according to the absolute value of the correlation coefficient. (ii) Next we run a series of preliminary multilayer neural networks using ten-fold cross validation and compute variable importance using the weights method proposed by Gevrey (2003) [13]. (iii) Finally, we run a Random Forest regression and compute mean decrease in accuracy and mean decrease in node purity for each variable. For the given data, the above measures are largely consistent.

The precise subset of the ranked features with the greatest predictive power is then chosen via forward selection using a multilayer neural network. The optimal subset is identified by minimizing the RMSE, which occurs with feature set of 50 independent variables, which included all 25 distance lags, all 15 lags of the Riemann-Siegel Z-function evaluated at coGram points, 8 terms of the Riemann-Siegel formula evaluated at Gram points, and 1 lag of both Gram and coGram sequences. This set is then analyzed for redundancy using a correlation matrix, which showed that no correlation exceeds 0.5.

3.4. **Topology Selection.** A great variety of neural network topologies are currently used in supervised machine learning. For the purposes of this study, we implement a feedforward Multilayer Perceptron (MLP) and a Recurrent Neural Network (RNN). All models in this study are implemented in the R programming language. The MLP model is constructed using RSNNS package, while the RNN is built in the Keras module using Tensorflow as backend.

Whereas a feedforward neural network can be visualized as a directed acyclic graph [15], a *Recurrent* Neural Network (RNN) can be represented by a directed graph that contains cycles. In neural networks of this type, signals between neurons can follow cyclic paths; this allows RNNs to learn from the same data multiple times before the output is calculated. This feature is implemented via feedback connections, which deliver output signals back to the inputs of neurons. This provides RNNs with "memory", because processed data remains in the network for some period of time. [22, 37]. Thus, in RNNs, weights are applied not just to the current input, but to previous inputs as well. For this reason, RNNs are considered to be the state-of-the-art for sequential data and were thus a clear choice for our second model.

3.5. **Structure Selection.** In this stage we select the optimal numbers of (i) hidden layers and (ii) neurons per layer for each of the neural network topologies above. Note that the dimensions of the input and output data determine the number of neurons in the input and output layers: since our data consists of 50 features and predicts only one variable, distance $\gamma_n - g_{n-1}$, the input and ouptut layers consist of fifty neurons and one neuron, respectively. Thus, it remains only to determine the structure of the hidden layers.

First, we determine the optimal number of hidden layers for each network by analyzing the relationship between network depth and test data RMSE. In both the MLP and the RNN, the number of neurons per layer is fixed at a preliminary ad hoc value of 45, while the number of hidden layers is varied from one to

five. The results of these experiments are recorded in Tables 1 and 2. We find the optimal depth to be two and four layers for the MLP and RNN models, respectively.

Hidden Layers	Test RMSE	Test R^2
1	0.04567	0.96580
2	**0.02610**	**0.98884**
3	0.03054	0.98471
4	0.02682	0.98821
5	0.03239	0.98281

TABLE 1. MLP model: Selection of hidden layers.

Hidden Layers	Test RMSE	Test R^2
1	0.05322	0.94461
2	0.02575	0.98703
3	0.02285	0.98979
4	**0.02182**	**0.99068**
5	0.13216	0.99026

TABLE 2. RNN model: Selection of hidden layers.

Next, the optimal number of neurons in each hidden layer is determined by fixing depth according to the optimal numbers above, while varying the number of neurons per layer. We begin by varying the number neurons per layer in five-point increments; after identifying the optimal interval, we determine the precise value of neurons per layer to minimize test RMSE. The results are illustrated in Figures 4, 5. On the leftmost plots, the shaded areas mark regions around optimal points that require finer inspection, the results of which are shown on the rightmost plots. On the basis of this analysis, it is determined that the optimal MLP model consists of 2 hidden layers with 34 neurons per layer, while the optimal RNN model consists of 4 hidden layers with 55 neurons per layer.

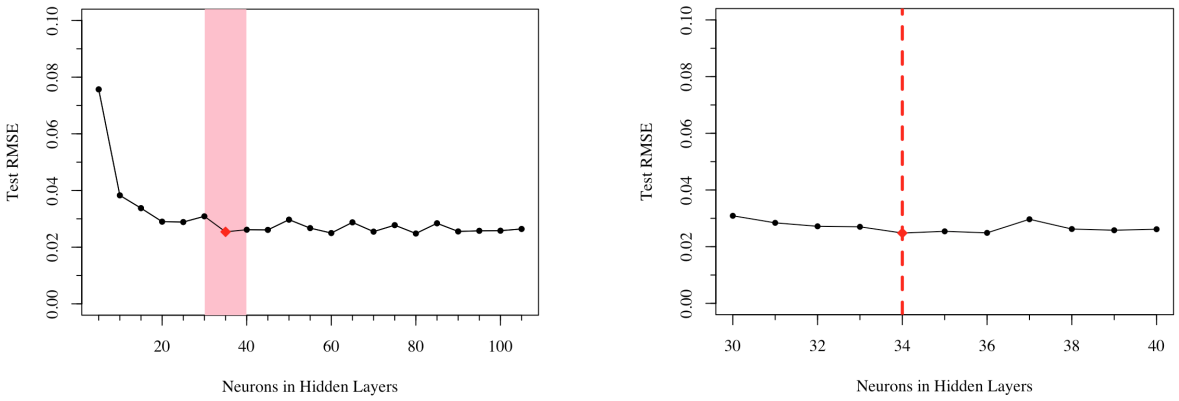

FIGURE 4. Test Error with respect to the number of neurons in each of 2 layers of the MLP.

3.6. **Calibration Stage.** In the calibration stage the neural networks are trained to determine weights. The optimization algorithms, as well as other hyperparameters used, are specified here.

(1) *Optimization algorithm*: Standard backpropagation methods were used for training the MLP model (the default in the R package RSNNS). The RNN model was optimized using ADAM—a stochastic optimization algorithm [19]. In our study, ADAM was compared to several other learning functions and proved to demonstrate better performance. This observation is supported by existing research [20]; stochastic gradient descent was found to converge faster and to a more optimal local minima than standard gradient descent.

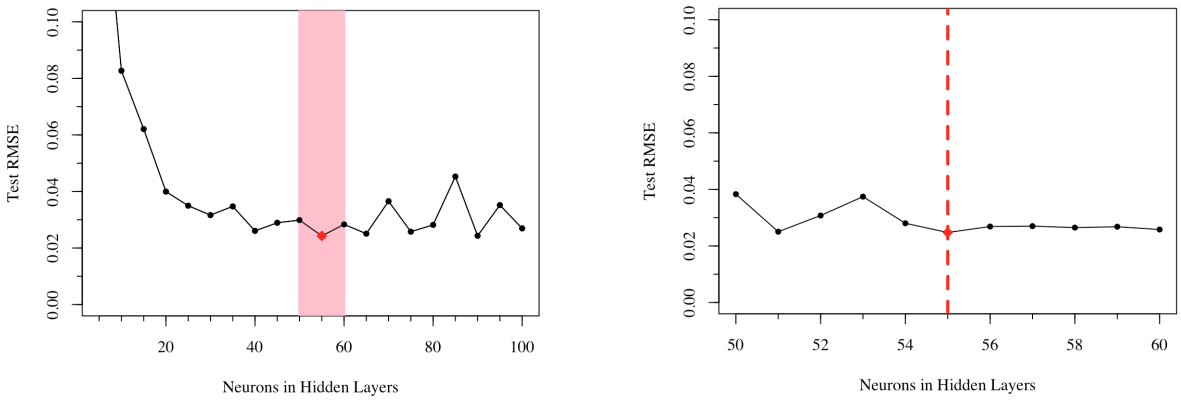

FIGURE 5. Test Error with respect to the number of neurons in each of 4 layers of the RNN.

(2) *Activation function:* Although the sigmoid activation function was considered the standard for many years, it suffers from increased likelihood of a vanishing gradient in deep learning architectures. For this reason, the constant gradient of the ReLU function exhibits more efficient training in many cases [1, 35]. Further, the ReLU offers increased sparsity, i.e., setting non-predictive coefficient weights to zero. Because our RNN model, consisting of four hidden layers, is a deeper network than our MLP, the comparative advantages of the ReLU are greater for the RNN. Thus the sigmoid activation function is used in the MLP model and in the output layer of the RNN model, while the hidden layers of the RNN are equipped with the ReLU activation function.

(3) *Loss function:* The Summed Squared Error (SSE) is an immutable default error function used in RSNNS package [6] and, consequently, is used for MLP. For the RNN model, the Mean Squared Error (MSE) metric was specified.

(4) *Number of epochs:* An *epoch* refers to an event marked by the propagation of full input data through neural network exactly once. In general, additional epochs increase optimization time but result in lower values of the loss function E on training data. However, when sufficiently many epochs are performed, models can exhibit *overfitting*—a state in which the dynamic parameters (i.e. weights and thresholds) are tuned to replicate training data rather than the patterns underlying it. An overfitted model is marked by very low training error, but high test error. In order to prevent overfitting, we record test and train errors produced by running our models with different numbers of epochs. These results are shown in Figure 6.

We observe that both train and test errors show an overall decreasing trend in the specified intervals of epochs (2 to 2^{13} for MLP and 2 to 2^{11} for RNN). Thus, it is not clear that an optimal number of epochs has been identified, indicating a direction for future research. However, there is no evidence of overfitting. In our implementation, the choice of epochs is informed by these results, but constrained by computational limitations resulting in 100 and $1,000$ epochs being utilized for the RNN and MLP, respectively.

(5) *Observations:* In order to avoid any edge effects associated with data points of small indices as well as to accommodate computations of terms of Riemann-Siegel formula, the first 1,000 observations are excluded from the final data set. As a result, the dimensionality of the selected input data is 99,000

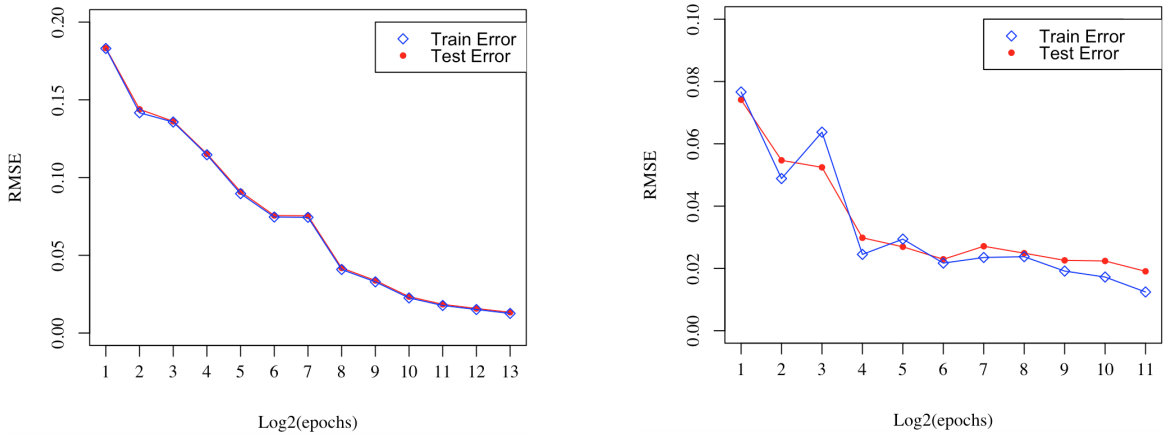

FIGURE 6. Test and train error curves with respect to $\text{Log}_2(epochs)$ for the MLP (left) and the RNN (right) models.

by 50. All data was preprocessed by Min-Max scaling algorithm for improved training efficiency [14]. This data set is divided amongst training and test data in a 80%/20% split.

In order to understand the dynamics of the predictive performance of our models when additional observations are introduced, we construct learning curves seen in Figure 7. We observe that the learning curve for the MLP appears to be sufficiently level for all sample sizes above 60,000. While the RNN does not provide quite as long a plateau, the results nonetheless suggest that an optimal or near optimal test RMSE is achieved by the use of a data set with at least 80,000 observations. Thus, both learning curves verify the sufficiency of the present data set of 99,000 observations.

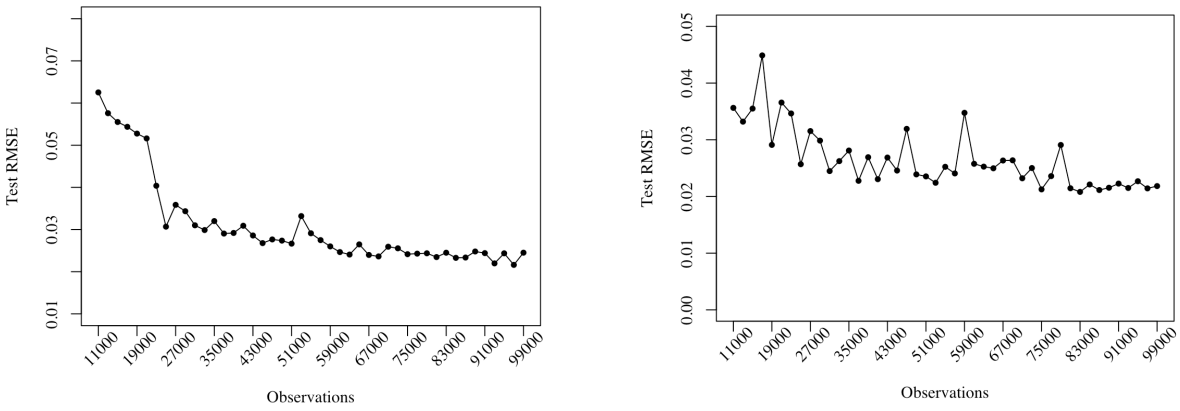

FIGURE 7. Test RMSE curve with respect to the number of observations available to ANNs for the MLP (left) and the RNN (right) models.

4. Model Validation And Predictions

In this section, we summarize implementation of the Multilayer Perceptron, Recurrent Neural Network, and Support Vector Machine regression models for the prediction of distance, $\gamma_n - g_{n-1}$. All three models utilize the above-described data set of 99,000 observations in 50 variables, Min-Max scaled during preprocessing. We evaluate the predictive and replicative accuracies of all three models. Finally, predictions for the distance variable are converted into a list of predicted zeros of the ζ-function.

4.1. **Implementation of the Three Models.**

4.1.1. *Implementation of the Multilayer Perceptron.* The MLP is implemented via the R package RSNNS with two hidden layers consisting of 34 neurons each. We apply a standard 80%/20% split of the randomly shuffled data. The model is then trained on 79,200 training observations in the 50 features selected during the input selection stage for a duration of 1,000 epochs. The remaining 19,800 observations are used as an independent test set to estimate predictive accuracy of the trained model. From these 19,800 data points, 50 observations for distance variable are plotted against the corresponding MLP predictions (Figure 8).

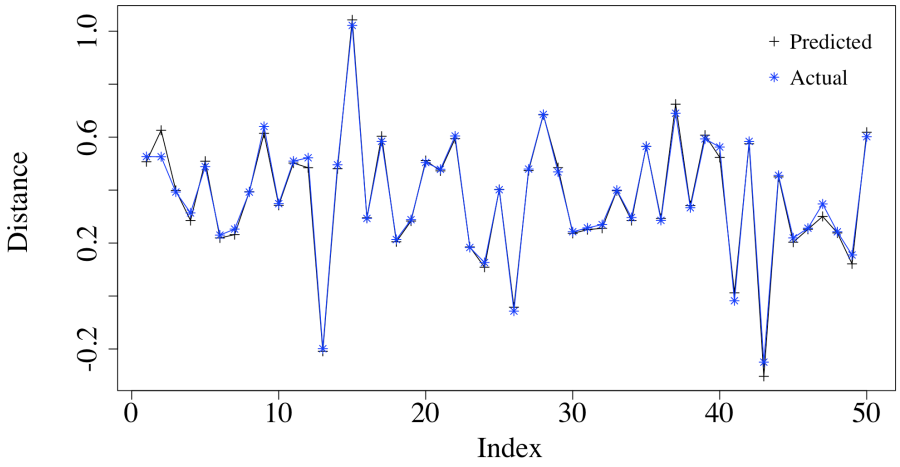

FIGURE 8. MLP predictions versus actual distance $\gamma_n - g_{n-1}$ for 50 randomly selected observations from the test data.

4.1.2. *Implementation of the Recurrent Neural Network.* The RNN model consists of 4 layers with 55 neurons in each, implemented in the Keras module using TensorFlow as a backend. It is trained with the ADAM stochastic gradient descent optimizer for a total of 100 epochs.

Whereas the data was randomized prior to splitting for the MLP model, for the RNN the data is not shuffled. The key advantage of recurrent neural networks is their exceptional ability to extract temporal dependencies within input sequences, thus, shuffling would greatly diminish this benefit[8]. After an 80%/20% split into training and testing sets, our RNN model is trained on 79,200 sequential observations of the same 50 features used in the MLP. A set of 50 consecutive distance observations is chosen randomly from the testing data and plotted against model's predictions (Figure 9).

4.1.3. *Implementation of the SVR Model.* For the purposes of comparison, we implement a Support Vector Machine Regression using the same data set employed in the neural networks above. The selected model utilizes ν-SVR with an RBF kernel given by $e^{\gamma|\boldsymbol{x_i}-\boldsymbol{x_j}|^2}$ where $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$ are features, $\gamma = \frac{1}{2\sigma^2}$, and σ denotes the width parameter. This kernel is advantageous in that it can automatically optimize centers, weights, and thresholds [39]. Final model specification requires the additional optimization of σ, described above and c, a regularization parameter which controls the trade-off between training and test error. We tune this SVR model using 10-fold cross validation with three repeats within a grid search to obtain the optimal values of $\sigma = 0.01$ and $c = 10$.

Data splitting was performed in the same manner as for the MLP: After performing an 80%/20% split of the randomly shuffled data into training and testing sets, our SVR model is trained on 79,200 observations

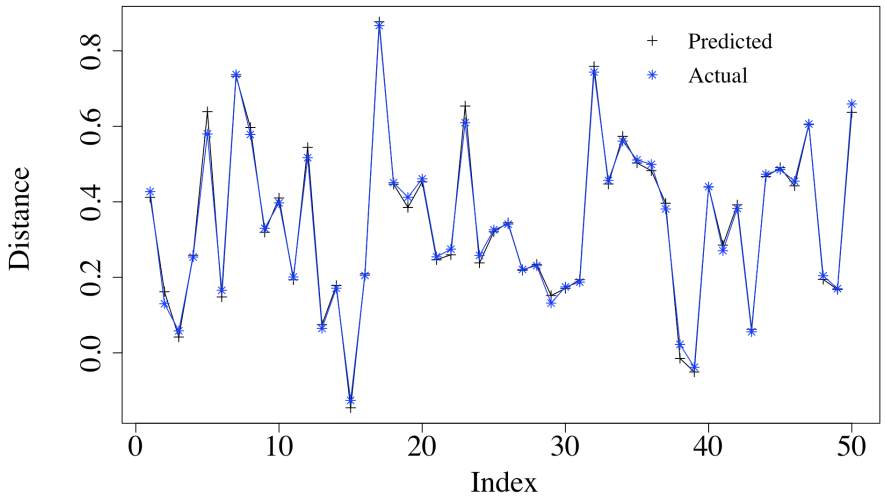

FIGURE 9. RNN predictions versus actual distance $\gamma_n - g_{n-1}$ for 50 consecutive observations from the test data.

in 50 features, centered during pre-processing. Following training, the calibrated SVR model is applied to the reserved test data: a set of 50 consecutive distance observations is chosen from the test data and plotted against the model's predictions (Figure 10).

The optimized model requires 58,427 support vectors, which results in a very high computing cost. As a result, each of the proposed neural networks outperform SVR in terms of efficiency by an order of magnitude. It remains to assess the respective predictive power of the three models. While visual inspection of the plot suggests good fit on test data, we find that this method is relatively inefficient, echoing the conclusions of Bengio (2007) [5].

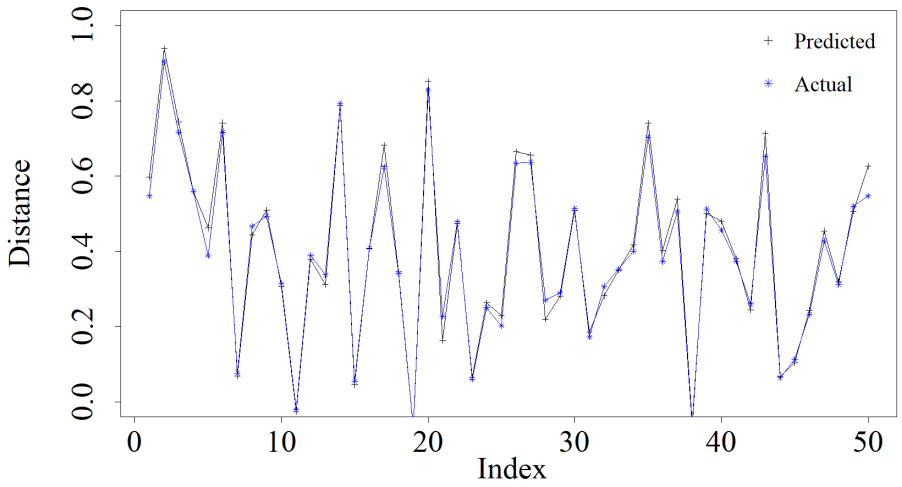

FIGURE 10. SVR predictions versus actual distance $\gamma_n - g_{n-1}$ for 50 randomly selected observations from the test data.

4.2. **Accuracy of the Models Compared.** In this section, we present the performance metrics obtained for all models during the prediction process. These accuracy measurements can be categorized into two

groups—*replicative accuracy* and *predictive accuracy*. Replicative accuracy refers the model's performance on data that was used during training. This should be considered a preliminary assessment of the extent to which a model accurately expresses the underlying patterns in the training data [46].

Predictive accuracy refers to the ability of a model to correctly predict output of the unseen data, given the respective inputs. This measurement essentially estimates the generalizability of the models and thus requires propagation of the test data through the trained neural networks.

The replicative accuracy values of the three models are summarized in Table 3. Since the original data is standardized using Min-Max scaling prior to data splitting, predictions and scaled distance observations must be converted back to the original scaling in order to avoid corrupted error scores. Thus, all of the metrics presented in the tables to follow are calculated with unscaled data and predictions. Note that because RNN was trained and tested on ordered data, Mean and SD (standard deviation) measures are omitted for this model. All three models produce an R-squared over 0.99. Based on the RMSE, we conclude that both neural networks offer superior replicative accuracy relative to SVR, with the RNN offering the highest accuracy.

Model	RMSE	R^2
Support Vector Machine	0.0202	0.9926
Multilayer Perceptron	0.0218	0.9913
Recurrent Neural Network	0.0163	0.9953

Model	Mean	SD
Actual Data	0.3713	0.2347
Support Vector Machine	0.3687	0.2303
Multilayer Perceptron	0.3717	0.2310
Recurrent Neural Network	N/A	N/A

TABLE 3. Replicative accuracy metrics for each of the three regression models.

Model	RMSE	R^2
Support Vector Machine	0.0310	0.9824
Multilayer Perceptron	0.0225	0.9907
Recurrent Neural Network	0.0201	0.9913

Model	Mean	SD
Actual Data	0.3709	0.2337
Support Vector Machine	0.3684	0.2274
Multilayer Perceptron	0.3713	0.2301
Recurrent Neural Network	N/A	N/A

TABLE 4. Predictive accuracy metrics for each of the three regression models.

Moving on to predictive accuracy, we see that all three models display similarly impressive predictive ability on previously unseen data. Both neural networks still produce an R-squared over 0.99, while the SVR R-squared falls to 0.9824 when considering only test data. Consistent with the findings of our replicative accuracy analysis, the Recurrent Neural Network model is most successful in predicting new terms of the distance variable sequence. Predictive accuracy figures are listed in Table 4.

The reliability of the obtained data is supported by the 10-fold cross-validation method for the neural network models. Prior to training the reported models, we calculate average errors of the 10 different models constructed during cross-validation in order to estimate the order of accuracy we should expect. For MLP, the average errors were $\overline{R^2} = 0.9899$, $\overline{RMSE} = 0.0234$; for the RNN model—$\overline{R^2} = 0.9920$, $\overline{RMSE} = 0.0205$. These values are consistent with our final error measurements.

4.3. **Predicted Zeros.** Since the RNN showed the best predictive accuracy (Table 4) among the three models compared, we now present the unscaled predictions of this regression model. In Table 5, we record

a sample of computed predictions of distance $\gamma_n - g_{n-1}$ and zeros γ_n using the first and the last two observations from the test data. For these same observations, we perform the necessary conversion to obtain the corresponding prediction for the zero of the ζ function. The residual error is consistent across observations and is typically below 0.01, Moreover, within the scope of this study, errors do not demonstrate an increase with the height of the zero being predicted

Distance $\gamma_n - g_{n-1}$	Prediction	Actual
$\gamma_{79,201} - g_{79,200}$	0.3249	0.3171
$\gamma_{79,202} - g_{79,201}$	0.4478	0.4386
$\gamma_{98,999} - g_{98,998}$	0.3688	0.3728
$\gamma_{99,000} - g_{98,999}$	0.2752	0.2712

Zero γ_n	Prediction	Actual	Residual
$\gamma_{79,201}$	61531.6300	61531.6222	0.0078
$\gamma_{79,202}$	61532.4366	61532.4274	0.0092
$\gamma_{98,999}$	74920.2557	74920.2598	-0.0041
$\gamma_{99,000}$	74920.8316	74920.8275	0.0041

TABLE 5. Predictions and Actual values of the distance ($\gamma_n - g_{n-1}$) and zero (γ_n) variables obtained by RNN.

5. Conclusion And Future Work

In this study, we apply three regression models—Support Vector Machine, Feed-forward and Recurrent Neural Network–to the task of predicting non-trivial zeros of Riemann zeta-function on the critical line. This problem is reformulated as a time-series prediction task with distance $\gamma_n - g_{n-1}$ as the target variable. The feature space for each of the three models consists of 99,000 observations in 50 predictor sequences. While the RNN model exhibited the best replicative and predictive accuracies, all models were found to output satisfactory predictions. SVR, MLP, and RNN models yield predictive accuracies of 0.9824, 0.9907, and 0.9953 respectively, measured as test data R-squared.

Using the most accurate of these three models, RNN predictions for the distance variable $\gamma_n - g_{n-1}$ variable are converted into predictions of the non-trivial zeros of the ζ-function and compared to the true values. The residual error of these predictions typically does not exceed 0.01 for observations within the test set. Most importantly, residual error appears to be independent of the height of the zero being predicted. Given the time-series transformation, the index of a zero being predicted has no bearing on computational effort. Thus, the finding of sustained accuracy at various heights offers a compelling advantage over existing computational methods, which become costly at high indices.

Future research could involve the implementation of other neural network models (e.g. Long/Short Term Memory, Radial Basis Function, etc.). Additionally, it is important to perform further evaluation of the decay of predictive accuracy as separation between test and training data increases and to extend our analysis to higher index levels. Finally, we hope to explore the possibility of using our predictions to aid in the location of new zeros.

Acknowledgements

We would like to thank our scientific advisor, Dr. Huan Qin, and the director of the San Diego State University REU program, Dr. Vadim Ponomarenko. Without their guidance, support and encouragement, this study would not be possible. Additionally, we extend our gratitude to San Diego State University for the provided facilities and National Science Foundation for funding.

References

[1] G. Alcantara. Empirical analysis of non-linear activation functions for Deep Neural Networks in classification tasks. ArXiv e-prints, Oct. 2017.

[2] A. Alhadbani and F. Stromberg. The riemann zeta function and its analytic continuation. 22:1–47, 01 2017.

[3] J. Arias-de-Reyna. X-Ray of Riemann zeta-function. ArXiv Mathematics e-prints, Sept. 2003.

[4] R. Bellman. Dynamic Programming. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.

[5] Y. Bengio and Y. Lecun. Scaling learning algorithms towards ai, 01 2007.

[6] C. Bergmeir and J. Bentez. Neural networks in r using the stuttgart neural network simulator: Rsnns. Journal of Statistical Software, Articles, 46(7):1–26, 2012.

[7] M. V. Berry and J. P. Keating. A new asymptotic representation for $\zeta(\frac{1}{2} + it)$ and quantum spectral determinants. Proc. Roy. Soc. London Ser. A, 437(1899):151–173, 1992.

[8] N. Boulanger-Lewandowski, G. J. Mysore, and M. Hoffman. Exploiting long-term temporal dependencies in nmf using recurrent neural networks with application to source separation. In 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 6969–6973, May 2014.

[9] G. E. P. Box and D. R. Cox. An analysis of transformations. (With discussion). J. Roy. Statist. Soc. Ser. B, 26:211–252, 1964.

[10] J. T. Connor, R. D. Martin, and L. E. Atlas. Recurrent neural networks and robust time series prediction. IEEE Transactions on Neural Networks, 5(2):240–254, March 1994.

[11] G. Cybenko. Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals, and Systems (MCSS), 2(4):303–314, Dec. 1989.

[12] G. Dorffner. Neural networks for time series processing. Neural Network World, 6:447–468, 1996.

[13] M. Gevrey, I. Dimopoulos, and S. Lek. Review and comparison of methods to study the contribution of variables in artificial neural network models. Ecological Modelling, 160(3):249 – 264, 2003. Modelling the structure of acquatic communities: concepts, methods and problems.

[14] S. Gopal Krishna Patro and K. K. Sahu. Normalization: A Preprocessing Stage. ArXiv e-prints, Mar. 2015.

[15] M. Gori. Machine Learning: A Constraint-Based Approach. Elsevier Science, 2017.

[16] G. Hughes. On the mean accuracy of statistical pattern recognizers. IEEE Transactions on Information Theory, 14(1):55–63, January 1968.

[17] J. I. Hutchinson. On the roots of the Riemann zeta function. Trans. Amer. Math. Soc., 27(1):49–60, 1925.

[18] A. Ivić. Hardy's function $Z(t)$: Results and problems. Tr. Mat. Inst. Steklova, 296(Analiticheskaya i Kombinatornaya Teoriya Chisel):111–122, 2017.

[19] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. ArXiv e-prints, Dec. 2014.

[20] R. Kleinberg, Y. Li, and Y. Yuan. An Alternative View: When Does SGD Escape Local Minima? ArXiv e-prints, Feb. 2018.

[21] M. Korolev. Gram's law and the argument of the Riemann zeta function. Publ. Inst. Math. (Beograd) (N.S.), 92(106):53–78, 2012.

[22] Z. C. Lipton, J. Berkowitz, and C. Elkan. A Critical Review of Recurrent Neural Networks for Sequence Learning. ArXiv e-prints, May 2015.

[23] R. May, G. Dandy, and H. Maier. Review of input variable selection methods for artificial neural networks, 04 2011.

[24] J. L. McClelland and D. E. Rumelhart. Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises. MIT Press, Cambridge, MA, USA, 1988.

[25] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. Bull. Math. Biophys., 5:115–133, 1943.

[26] L. Menici. Zeros of the riemann zeta-function on the critical line. 2012.

[27] H. N. Mhaskar. Neural networks for optimal approximation of smooth and analytic functions. Neural Computation, 8(1):164–177, Jan 1996.

[28] K.-R. Müller, A. J. Smola, G. Rätsch, B. Schökopf, J. Kohlmorgen, and V. Vapnik. Advances in kernel methods. chapter Using Support Vector Machines for Time Series Prediction, pages 243–253. MIT Press, Cambridge, MA, USA, 1999.

[29] A. Odlyzko. The first 100 zeros of the riemann zeta function, accurate to over 1000 decimal places.

[30] J. Park and I. W. Sandberg. Universal approximation using radial-basis-function networks. Neural Computation, 3(2):246–257, June 1991.

[31] R. Pérez-Marco. Notes on the Riemann Hypothesis. ArXiv e-prints, July 2017.

[32] P. Petersen and F. Voigtlaender. Optimal approximation of piecewise smooth functions using deep relu neural networks. Neural Networks, 108:296–330, 2018.

[33] M. Powell. The theory of radial basis function approximation. In W. Light, editor, Advances in Numerical Analysis III, Wavelets, Subdivision Algorithms and Radial Basis Functions, pages 105 – 210. Clarendon Press, Oxford, 1992.

[34] G. R. Pugh. The riemann-siegel formula and large scale computations of the riemann zeta function. 1998.

[35] P. Ramachandran, B. Zoph, and Q. V. Le. Searching for Activation Functions. ArXiv e-prints, Oct. 2017.

[36] J. B. Rosser, J. M. Yohe, and L. Schoenfeld. Rigorous computation and the zeros of the riemann zeta-function. 1968.

[37] H. Salehinejad, S. Sankar, J. Barfett, E. Colak, and S. Valaee. Recent Advances in Recurrent Neural Networks. ArXiv e-prints, Dec. 2018.

[38] S. Scardapane, S. V. Vaerenbergh, A. Hussain, and A. Uncini. Complex-valued neural networks with non-parametric activation functions. CoRR, abs/1802.08026, 2018.

[39] B. Scholkopf, K.-K. Sung, C. J. C. Burges, F. Girosi, P. Niyogi, T. Poggio, and V. Vapnik. Comparing support vector machines with gaussian kernels to radial basis function classifiers. IEEE Transactions on Signal Processing, 45(11):2758–2765, Nov 1997.
[40] O. Shanker. Neural network prediction of Riemann zeta zeros. Adv. Model. Optim., 14(3):717–728, 2012.
[41] A. J. Smola and B. Schölkopf. A tutorial on support vector regression. Stat. Comput., 14(3):199–222, 2004.
[42] E. Stein and R. Shakarchi. Complex Analysis. Princeton lectures in analysis. Princeton University Press, 2010.
[43] U. Thissen, R. van Brakel, A. de Weijer, W. Melssen, and L. Buydens. Using support vector machines for time series prediction. Chemometrics and Intelligent Laboratory Systems, 69(1):35 – 49, 2003.
[44] B. Warner and M. Misra. Understanding neural networks as statistical tools. The American Statistician, 50(4):284–293, 1996.
[45] M. R. Watkins. Proposed proofs of the riemann hypothesis.
[46] W. Wu, G. C. Dandy, and H. R. Maier. Review: Protocol for developing ann models and its application to the assessment of the quality of the ann model development process in drinking water quality modelling. Environ. Model. Softw., 54:108–127, Apr. 2014.
[47] G. Zhang. Time series forecasting using a hybrid arima and neural network model. Neurocomputing, 50:159 – 175, 2003.

JENNIFERKAMPE@GMAIL.COM, AVYSOGORETS@UMASS.EDU, HQIN@SDSU.EDU