Predicting Zeros of the Riemann Zeta Function Using Machine Learning: A Comparative Analysis

Jennifer Kampe and Artem Vysogorets

August 11, 2018

Abstract

In this study, we evaluate the predictive performance of Neural Network Regression in locating non-trivial zeros of the Riemann zeta-function relative to Support Vector Machines Regression. We provide a brief summary of the fundamental properties of the zeta-function and use the analysis therein to identify relevant features and reformulate the given regression problem as a time-series prediction problem. Next, we provide a basic introduction to the architecture and use of Neural Networks. Model design is implemented as a six stage process including: (i) input selection, (ii) data splitting, (iii) model architecture selection, (iv) model structure selection, (v) model calibration, and (vi) model validation. Within each stage, optimization is achieved via available algorithms as well as trial-and-error. The range of zeta-function zeros used is 1001 to 100,000; model training is performed on data with 99,000 observations for each of the 50 feature variables selected. Multilayer Perceptron and Recurrent Neural Network architectures are chosen and implemented in the R programming language. Finally, the replicative and predictive accuracies of the two neural networks are evaluated against those of a Support Vector Machine Regression.

1 Introduction

The Riemann Zeta-function (ζ-function) is one of the most celebrated examples of Dirichlet L-functions that has been extensively studied by mathematicians for almost 200 years. Study of this function connects two seemingly unrelated branches of mathematics—number theory and complex analysis. The importance of ζ-function resides in its profound links to prime numbers that provide key insights into their distribution.

Given the complexity of ζ-function, regression with non-parametric machine learning models offers an alternate means of exploring the Riemann ζ-function. Machine learning regression refers to an ensemble of algorithms used to model dynamic systems by analyzing large sequences of

system parameters or other relevant variables. Two major categories of machine learning regression algorithms are Artificial Neural Networks and Support Vector Machines.

First introduced by neurophysiologist Warren McCulloch and logician Walter Pitts as early as in 1943, Artifical Neural Networks did not initially experience widespread adoption due the computational cost of implementation [29]. 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. At present, however, there is only one known application of the neural networks to the task of modeling the ζ-function: Shanker [39], uses an unspecified Neural Network Regression to predict the locations of ζ-function zeros. While Support Vector Machines 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 [42, 31]. Thus, SVM may be appropriate for purposes of this study.

In the paper that follows, we build upon the work of [39] by adapting different varieties of neural networks to the prediction of ζ-function zeros locate on the critical line. Additionally, we introduce the use of Support Vector Machines Regression for the problem and report relative and absolute performance metrics for all models considered.

2 Introduction to Riemann Zeta Function

First introduced by Bernhard Riemann in 1859, for $s \in \mathbb{C}$ with $\Re(s) > 1$, Riemann ζ-Function $\zeta(s)$ is defined as follows:

$$\zeta(s) = \sum_{n=1}^{\infty} 1/n^s, \tag{1}$$

which converges absolutely in the specified domain and so is well-defined. More than 100 years earlier, Leonhard Euler studied a real-variable prototype of this function (with $s \in \mathbb{R}$), computed $\zeta(2) = 1 + 1/4 + 1/9 + 1/16 \ldots = \pi^2/6$ (Basel problem) and later provided expressions for values of 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 [34]:

$$\zeta(s) = \sum_{n=1}^{\infty} n^{-s} = \prod_{p} (1 - p^{-s})^{-1}, \tag{2}$$

where the product is taken over all prime numbers p. This relation can be shown by constructing a bijection between terms of these two infinite expressions, which exists due to the Fundamental Theorem of Arithmetic. This representation allows us to conclude that no zeros of ζ-function have

real part greater than 1:

$$|\zeta(s)| = \Big|\prod_p \frac{1}{(1-p^{-s})}\Big| = \prod_p \frac{1}{|1-p^{-s}|} \geqslant \prod_p \frac{1}{(1+p^{-\Re(s)})} > 0. \tag{3}$$

Whereas the original series definition of the Riemann ζ-function is valid only in the half plane $\Re(s) > 1$, Riemann extended its domain using a process of *analytic continuation* [4]. In this way, Riemann continued ζ to a meromorphic function on the entire complex plane—except for a simple pole at $s = 1$. Moreover, he constructed the following functional equations for the new extended ζ-function [2]:

$$\pi^{-s/2}\Gamma(\frac{s}{2})\zeta(s) = \pi^{-(1-s)/2}\Gamma(\frac{1-s}{2})\zeta(1-s), \tag{4}$$

$$\zeta(s) = 2^s\pi^{s-1}\sin\Big(\frac{\pi s}{2}\Big)\Gamma(1-s)\zeta(1-s), \tag{5}$$

where Γ is the Gamma-function, defined as $\Gamma(s) = \int_0^\infty x^{z-1}e^{-x}dx$.

The second functional equation (5) can be derived from (4) using Legendre relation $\Gamma(z/2)\Gamma(z/2+1/2) = 2\sqrt(\pi)2^{-z}\Gamma(z)$ with $z = 1-s$, and the property $\sin(z\pi) = \pi\frac{1}{\Gamma(z)}\frac{1}{\Gamma(1-z)}$ with $z = s/2$ [20]. From equation (5) we see that ζ-function vanishes at negative even integers, which Riemann called *trivial zeros*. Later, Riemann hypothesized that all other (*non-trivial*) zeros lie on a *critical line* ($\Re = 1/2$). This conjecture is widely known as *Riemann Hypothesis* and is now on the list of Hilbert's unsolved problems [4]. So far, all computations confirm the Riemann hypothesis. Moreover, it has been rigorously shown that no non-trivial zeros of ζ-function lie outside of a *critical strip*, which is defined by $s \in \mathbb{C} : 0 < \Re(s) < 1$ [41].

A further insight from the functional equations can be found in Riemann's proof of (4) and (5). In these proofs, Riemann used Cauchy's theorem on complex integrals over closed contours [30], which permitted him to obtain an equation of the following form:

$$\zeta(s) = \chi(s)\zeta(s-1), \tag{6}$$

$$\chi(s) = \frac{\Gamma((1-s)/2)}{\Gamma(s/2)}\pi^{s-1/2}. \tag{7}$$

One can observe a certain symmetry in the functional equation (4), as well as in the definition of χ (7). In order to reveal this symmetry, we will now assume that the argument s lies on the critical line ($s = 1/2 + it$). Then, we can rewrite χ-function as follows:

$$\chi(1/2+it) = \frac{\Gamma((1-1/2-it)/2)}{(1/2+it)/2}\pi^{1/2+it-1/2} = \frac{\Gamma(\frac{1}{4}-\frac{it}{2})}{\Gamma(\frac{1}{4}+\frac{it}{2})}\pi^{it}. \tag{8}$$

For reasons that will become apparent presently, we now define the *Riemann Siegel Z-function* and

use (8) to rewrite it in a different form [19]:

$$Z(t) := \zeta(1/2+it)(\chi(1/2+it))^{-1/2} \Rightarrow \tag{9}$$

$$Z(t) = \zeta(1/2+it)\sqrt{\frac{\Gamma(\frac{1}{4}+\frac{it}{2})}{\Gamma(\frac{1}{4}-\frac{it}{2})}}\pi^{-it/2} = \zeta(1/2+it)\pi^{-it/2}\exp(i\arg(\Gamma(\frac{1}{4}+\frac{it}{2}))) \tag{10}$$

Here, we use the fact that for any complex number z, $\arg(z) = \frac{1}{i}\ln\sqrt{z/\overline{z}}$, as well as $\Gamma(\overline{z}) = \overline{\Gamma(z)}$. To further simplify the equation above, it is useful to introduce the *Riemann-Siegel θ-function*:

$$\theta(t) := \arg[\Gamma(\frac{1}{4}+\frac{it}{2})] - \frac{\ln\pi}{2}t \Rightarrow \tag{11}$$

$$Z(t) = e^{i\theta(t)}\zeta(\frac{1}{2}+it). \tag{12}$$

Finally, (12) gives us a concise representation of ζ-function on the critical line:

$$\zeta(\frac{1}{2}+it) = Z(t)e^{-i\theta(t)}. \tag{13}$$

The equation above is important because it allows us to study behavior of ζ-function on the critical line through the Z-function, which itself possesses several useful properties. First, an immediate consequence of (13) is the identity $|Z(t)| = |\zeta(1/2+it)|$. This implies that zeros of Z coincide with zeros of ζ-function on the critical line. Therefore, a change of sign of Z-function may help locate non-trivial zeros of ζ-function. Additionally, it turns out that Z is a real-valued function (for $t \in \mathbb{R}$) [19, 30]. To see this, observe that (8) implies

$$\chi(1/2+it)\chi(1/2-it) = \frac{\Gamma(\frac{1}{4}-\frac{it}{2})}{\Gamma(\frac{1}{4}+\frac{it}{2})}\pi^{it}\cdot\frac{\Gamma(\frac{1}{4}+\frac{it}{2})}{\Gamma(\frac{1}{4}-\frac{it}{2})}\pi^{-it} = 1. \tag{14}$$

Also, from the functional equation (4) and the definition (7) of χ-function we get

$$\zeta(s) = \frac{\pi^{\frac{s-1}{2}}\Gamma(\frac{1-s}{2})\zeta(1-s)}{\pi^{-s/2}\Gamma(s/2)} = \chi(s)\zeta(1-s) \Rightarrow \tag{15}$$

$$Z(t) = \frac{\zeta(1/2+it)}{\sqrt{\chi(1/2+it)}} = \frac{\zeta(1/2-it)}{\sqrt{\chi(1/2-it)}} = \Big(\overline{\frac{\zeta(1/2+it)}{\sqrt{\chi(1/2+it)}}}\Big) = \overline{Z(t)}. \tag{16}$$

To verify the last identity, we need to observe that $\chi(\overline{1/2+it}) = \overline{\chi(1/2+it)}$, which follows from $\Gamma(\overline{s}) = \overline{\Gamma(s)}$ and the definition of χ-function. Along with the property $\zeta(s) = \overline{\zeta(\overline{s})}$ [16], this implies equation (16) and so $Z(t) = \overline{Z(t)}$ follows. Therefore, $Z(t)$ is real-valued for $t \in \mathbb{R}$.

A final advantage of working with the Z-function relative to the ζ-function is that its values are easier to compute. Before Carl Siegel made asymptotic expansion of Z-function available in 1932, *Euler-Maclaurin summation* method was generally used in approximation of ζ-function. Even

though Euler-Maclaurin summation was a more universal and a more accurate tool, it required $|t|$ steps as opposed to $\sqrt{|t|}$ steps needed for evaluation of asymptotic expansion of Z-function, widely known as the *Riemann-Siegel formula* [35]:

$$Z(t) = 2\sum_{n=1}^{N} \frac{\cos(\theta(t) - t\ln(n))}{\sqrt{n}} + R, \tag{17}$$

where $N = \lfloor\sqrt{t/2\pi}\rfloor$, and R is a remainder term.

Given the above properties of the Z-function, it seems natural to explore its connection to the ζ-function (16) in more detail. Let us use the Euler's formula to separate ζ-function into its real and imaginary parts, which we will denote by $A(t)$ and $B(t)$, respectively:

$$\zeta(1/2 + it) = Z(t)e^{-i\theta(t)} = Z(t)[\cos(\theta(t)) - i\sin(\theta(t))] \Rightarrow \tag{18}$$

$$A(t) := \Re(\zeta(1/2 + it)) = Z(t)\cos(\theta(t)); \tag{19}$$

$$B(t) := \Im(\zeta(1/2 + it)) = -Z(t)\sin(\theta(t)). \tag{20}$$

Locating zeros of ζ-function on the critical line is equivalent to finding values of t such that $A(t) = B(t) = 0$. If $B(t_x) = 0$, then either $Z(t_x) = 0$ (hence, $\zeta(1/2 + it_x) = 0$), or $\sin(\theta(t_x)) = 0$, which requires $\theta(t_x)$ to be a multiple of π. This discussion inspires the definition of a *Gram point* g_n, which is a unique solution to the equation $\theta(g_n) = (n-1)\pi$. The existence and uniqueness of such solutions are ensured by the monotonicity of θ-function [3].

A key observation is that for Gram points, we have [23]:

$$\zeta(1/2 + ig_n) = A(g_n) + iB(g_n) = Z(g_n)\cos((n-1)\pi) = (-1)^{n-1}Z(g_n) \tag{21}$$

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, nontrivial zeros of the Riemann ζ-function on the critical strip. This method was invented by a Danish mathematician Jørgen 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 regarding the universality of this pattern is now known as *Gram's law* — despite the fact that Gram himself claimed it to be false. In fact, the first violation of this conjecture was found by Hutchinson [18] and occurs at the 127-th Gram point [23].

It is worth going into some depth regarding Gram points because they do indeed prove useful as regressors of both Neural Networks and Support Vector Machines Regression models of the

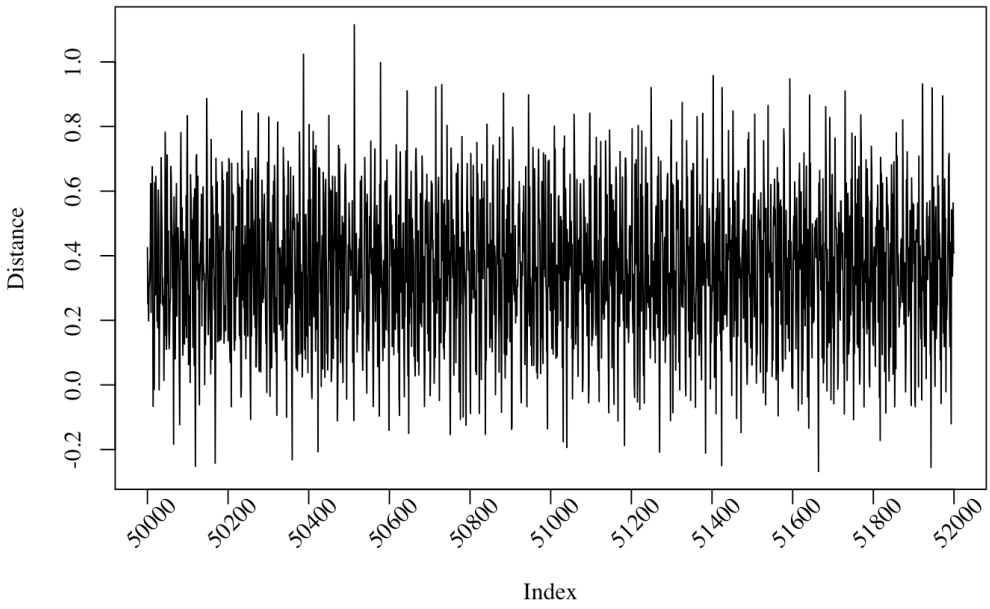

Figure 1: A subset of 2000 Distance values between $n = 50000$ and $n = 52000$.

Riemann ζ zeros. Instead of predicting the imaginary parts γ_n of non-trivial zeros of ζ-function, we predict differences $\gamma_n - g_{n-1}$ between zeros and Gram points, which we refer to as *distance*. This decision was inspired by the fact that Gram points and g_n and zeros γ_n are asymptotically equivalent ($g_n \sim \gamma_n \sim 2n\pi/\ln(n)$ [24]), which allows us to obtain a mean and variance-stationary time-series by taking differences $\gamma_n - g_{n-1}$. Additioanlly, Shanker successfully uses the textitdistance approach in his neural network study of Riemann ζ zeros [39].

Predicting the distance variable is conceptually equivalent to predicting zeros, since calculation of any finite number of Gram points to a desired accuracy is a relatively easy task [39]. In this study, we calculate first 100,000 Gram points to the 7-th digit after decimal point using a custom numerical algorithm implemented in Python. This algorithm utilized the first five terms of the following asymptotic expansion of θ-function, which can be derived from the Stirling's expansion of the Gamma function [9]:

$$\theta(t) = \frac{t}{2}\ln\left(\frac{t}{2\pi}\right) - \frac{t}{2} - \frac{\pi}{8} + \frac{1}{48t} + \frac{7}{5760t^3} + \cdots \tag{22}$$

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). Thus, using distance as the output of our regression algorithms, rather than the zeros themselves which are clearly non-stationary, facilitates the application of time series methods in the models to follow.

Recall that the definition of Gram points was chosen so that the imaginary part $B(t)$ of $\zeta(1/2+$

it) vanishes in equation (18). Although no known sources have utilized the following, we investigate a sequence of points that make $\cos(\theta(t))$ and, consequently, the real part $A(t)$ zero as well. In the absence of any accepted notation for these points, we refer to them as *co-Gram points* in order to emphasize their similarity with Gram points. Thus, we define n-th co-Gram point as a unique solution to $\theta(t) = n\pi + \pi/2$. The first 100,000 co-Gram points were computed using the same numerical algorithm in Python and are used as inputs of our regression models.

3 Introduction to Neural Network Regression

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. ANN models can be divided into two major categories: data classification and Neural Network Regression (NNR). The latter, NNR, is utilized here in our effort to represent the zeros of the Riemann Zeta function. According to Kouam, NNR operates as a generalization over multiple classic forecasting models (bilinear, autoregressive, non-parametric, etc.) [39]. 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 [11]. Thus, the nonparametric nature of NNR makes it a favorable option for learning tasks in which the functional form is unknown - as is the case with the Riemann ζ function.

3.1 Forward Propagation in a Neural Network

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 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*. The last layer is called an *output layer* and contains one neuron for each feature predicted.

Upon entering the input layer of an ANN, 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 *Feed-forward* 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 Feed-forward network is a multilayer perceptron (MLP), which is one of the models implemented in this paper. It is common practice for

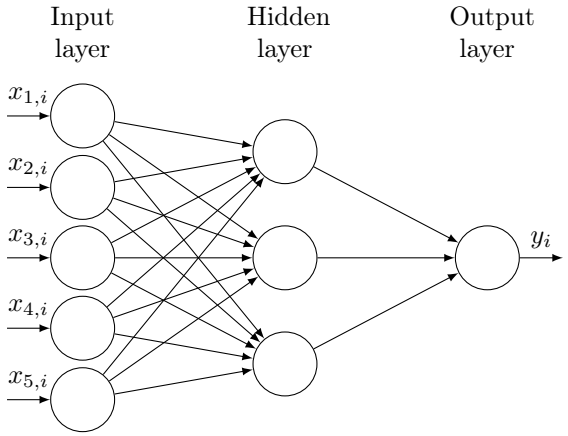

Figure 2: Standard architecture of a single layer densely connected MLP.

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.

Whereas Feed-forward ANNs can be visualized as directed acyclic graphs [15], the recently introduced concept of 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.

As in biological neural networks, data/signal processing in ANNs occurs in neurons. Consider the a-th neuron in the L-th layer of a given ANN, which we will denote by n_a^L. Additionally, suppose that according to the topology and architecture of an ANN, it receives signals from m and sends signals to q other neurons. Then, neuron n_a^L can be described by two parameters: m-dimensional vector of *weights* $[w_{1a}, w_{2a}, \ldots, w_{ma}]^T$ and a *bias* term b_a. Given output signals of neurons $n_1, n_2 \ldots, n_m$ as x_1, x_2, $\ldots$, x_m, signal o_a sent by n_a^L to each of the q output connections is calculated using the following formula [43]:

$$o_a = \sigma_L(\sum_{i=1}^{m} w_{ia}x_i - b_a), \tag{23}$$

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 *input signal*.

The data processing structure described above can be interpreted using analogies from basic principles of biological neural networks. In human brains, neurons communicate through electrochemical impulses, intensity of which is determined by the strength of synaptic connections between them. In turn, strength of input signals determines their ability to *excite* neurons, which have a certain activation threshold ($\sim$ 55mA). Neurons fire an output signal to successive neu-

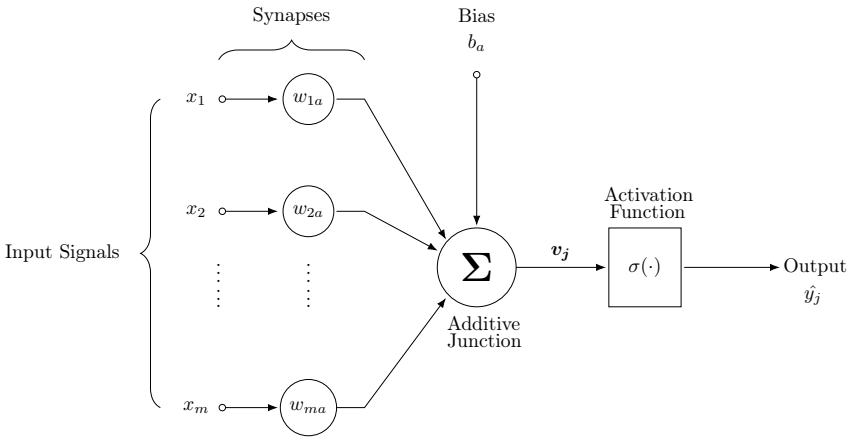

Figure 3: Feedforward propagation in a single hidden layer neuron.

rons if an incoming signal exceeds this threshold, and remain dormant otherwise. Thus, biological neuron is a switch, which is either on or off. In context of ANNs, this behavior is modeled by introducing bias b_a (threshold) and an activation function $\sigma = h(t)$, where $h(t)$ is a Heaviside step function (also known as *binary* function) activated at $t = 0$. The strength of a signal transferred across synapses of neurons n_i and n_j is imitated by weight w_{ij}. Therefore, an m-dimensional weight vector $[w_{1a}, w_{2a}, \ldots, w_{ma}]^T$ refers to a vector of synaptic strengths assigned to each of the m connections coming into neuron n_a^L [Figure 3]. Neurons of this design are referred to as *McCulloch-Pitts neurons*. [5, 25, 43].

3.2 Backprogagation in Neural Networks

The process of *learning* in artificial neural networks refers to the iterative optimization of weights and thresholds (biases) for all neurons. 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 $E = E(y_i - \hat{y}_i)$. In biology, such a learning strategy is called *reinforcement learning*. On the other hand, in *unsupervised learning*, machines are given no information about the desired output, while their goal is to group data with common patterns or features together [43].

Since NNR utilizes the first learning approach, 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 are programmed to minimize error (*loss*) function E by iteratively applying *gradient descent*—a strategy of adjusting

weights proportional to the negative of the partial derivative of E (e.g. mean squared error) taken with respect to each of the weights [28]. Thus, result of each iteration (propagation of a single observation from training data through ANN) is calculation of all partials $\partial E/\partial w_{ij}$ 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*. The above procedure updates biases as well. In practice, bias term b_a is often implemented as an always "on" input connection. In other words, having bias term b_a is equivalent to having $(m+1)$-th input connection of weight b_a from a neuron with constant output 1. Thus, treating bias as a "virtual" connection allows us to apply backpropagation to update both dynamic parameters of all neurons with a single iteration. A set of iterations marked by propagation of each observation from training data exactly once is called an *epoch*.

The algorithm of backpropagation is clever, however, it is incompatible with ANNs that consist of neurons of purely biological design described above. 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. On the other hand, in order to fully simulate biological neurons, Heaviside function should be used. In order to resolve this incompatibility, most of 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).

Even though backpropagation algorithm described above is commonly used in Feed-forward neural networks, there exists a wide spectrum of other optimization techniques, many of which are still based on the concept of gradient descent.

3.3 Theoretical Comparison of ANNs and SVR

Support Vector Machines are a flexible class of supervised learning algorithms that use kernel functions derive efficient solutions to certain nonlinear learning problems. For an overview of the basic workings of SVR, as well as the key algorithms used today, see Smola [40]. 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 [42] argues that the use of a convex objective function permitting a global optimal solution a key advantage of SVR relative to ANN's. Additionally, convergence on the global optimal solution conveys increased generalizability, reducing the overfitting problem often encountered in ANN's.

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 [7] 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 [7]. 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 compare methods of deep and shallow architecture and for .

4 Neural Networks: Application to Riemann Zeta-function

Given the ability of ANN's 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 ANN's and, for comparison, a Support Vector Machines 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.

4.1 Neural Network Design Protocol

The process of building our models is guided by the protocol proposed by Wu [44]. 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, while 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 three 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, $i = 1, \cdots, n$ are the true values of the dependent variable $\hat{y_i}$, $i = 1, \cdots, n$ 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.

4.2 Data Splitting

Whereas linear regression is optimized for inference, the explicit goal of machine learning regression is to make predictions. It is no surprise, then, 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 data is known as *data splitting*.

Cross-Validation is a method of model evaluation in which the original data sample is partitioned into a training set 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 error are reported using the metrics above.

4.3 Input Selection

Input selection refers to the key phase of model design in which the relevant features, (i.e., variables, 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 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 its 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 [17].

May et al. outline the principal considerations in selecting the optimal subset of features: relevance, computational effort, dimensionality, and training difficulty [27]. *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* [6].

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.

4.3.1 Assembling the Data

While gathering our initial pool of features, we consulted a similar study carried out by Shanker [39]. 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 (17) evaluated at Gram points; computation of Gram points is described in section 2. 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 can be thought of a time-series prediction, we also included 25 variables of lagged distance (lags 1-25). Finally, the initial pool of input candidates is enlarged by the following features: co-Gram sequence and the first 10 of its lags; Z-function evaluated at co-Gram points and the first 15 of its lags; Z-function evaluated at successive integers and the

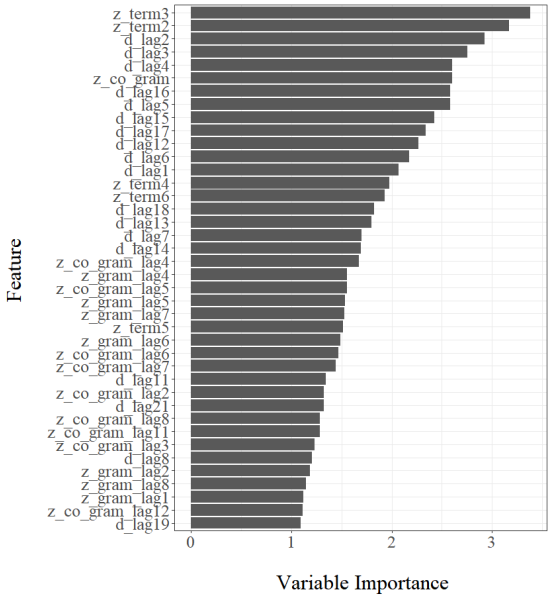

Figure 4: The top 40 most important variables.

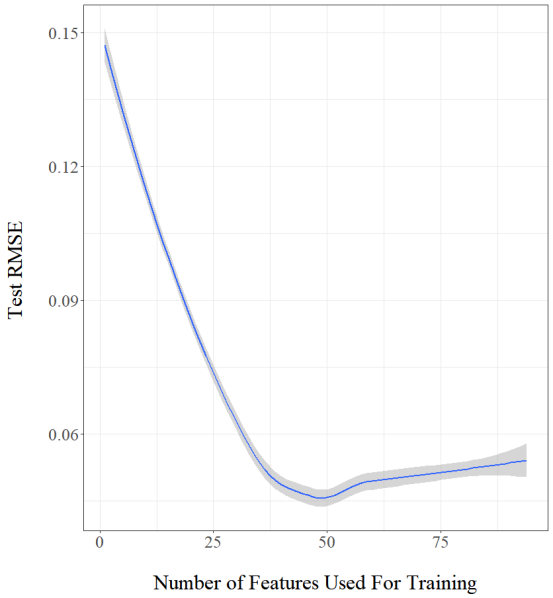

Figure 5: Test error as a function of feature set size.

first 10 of its lags. In total, this amounts to 94 candidate features. All of the computations of Z-function were carried out using Mathematica programming software. The outcome variable, *distance* is derived from the Odlyzko's computation of the first 100,000 zeros of the Riemann zeta function, accurate to within $3 \cdot 10^{-9}$ [33].

Such an emphasis on Z-function was encouraged by the *Cybenko's theorem*, which shows that any continuous real-valued function can theoretically be modeled using MLP with sigmoid activation function to the desired degree of accuracy [12]. Therefore, our hope was to provide sufficient information regarding Z-function, which has close connections to ζ-function, its zeros and, consequently, to the distance variable we wish to predict.

4.3.2 Minimum Redundancy Maximum Relevance Feature Selection

From the of 94 candidate features, we select inputs for our regression models using 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 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. We rank variables in descending order of importance as generated in (ii) above.

Next, the subset of ranked features with the greatest predictive power is chosen via forward selection using a multilayer neural network. The best subset is chosen by minimizing the RMSE—this occurs with feature set of 50 independent variables. Note that Figure 4 displays only the first forty of these for ease of visualization. In this graph, d_lag variables refer to lagged distance sequences; z_gram_lag and $z\_co\_gram\_lag$ variables—to lagged sequences of Z-function evaluated at Gram and co-Gram points, respectively; z_term stands for the sequence of the corresponding term in Riemann-Siegel formula (17) at Gram points. The obtained candidate set is then analyzed for redundancy using a correlation matrix, which showed that within the final subset, no correlation exceeds 0.5. 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, first 1,000 observations were excluded from this final set. As a result, the dimensionality of the selected input data is 99,000 by 50. Before it was fed to the ANN models, data was preprocessed by Min-Max scaling algorithm, which is a common approach that often results in a better predictive performance [14].

4.4 Architecture Selection

There is a great variety of different types of ANNs currently used in machine learning, including Feed-forward, Radial Basis Function (RBF), Kohonen, Recurrent, and Convolutional Neural Networks. Due to insufficient supply of previous works done on the topic of our interest, two most standard NNR architectures—Multilayer Perceptron and Recurrent Neural Network—were selected for this study. In section 3, we briefly described some fundamental characteristics of these two models. While MLP is a Feed-forward neural network, RNN models contain feedback connections, which deliver output signals back to the inputs of neurons. This feature provides Recurrent neural networks with "memory", because processed data remains in the network for some period of time. [26, 37]. Thus, RNNs are capable of learning short and long term dependencies in the input sequences, which makes them a desired tool for time-series prediction.

All models in this study are implemented in R programming language. MLP model is constructed with help of RSNNS package, while RNN Sequential model with dense layers is built in Keras module using Tensorflow as backend.

4.5 Structure Selection

This stage is concerned with selection of the optimal numbers of hidden layers and neurons for the ANN models. As we mentioned in the second section, the dimensionality of input/output data uniquely determines input and output layers. Since our data consists of 50 features and predicts only one variable (distance), input and output layers of ANNs in this study had 50 and 1 neurons, respectively. First, the number of hidden layers of the RNN and the MLP is determined iteratively

by running these models with preliminary numbers of neurons (45 in each layer).

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.

Results of these runs, recorded in tables 1 and 2, imply that 2 and 4 hidden layers are optimal for MLP and RNN models, respectively. Next, the optimal number of neurons in hidden layers of these ANNs is estimated in a similar way [Figures 6, 7]. On the leftmost plots, shaded areas mark regions around optimal points that require finer inspections, the results of which are shown on the rightmost plots. On the basis of this analysis, it is determined that optimal MLP model must have 2 hidden layers with 34 neurons per layer, while structure of the optimal RNN model is 4 hidden layers with 55 neurons per layer.

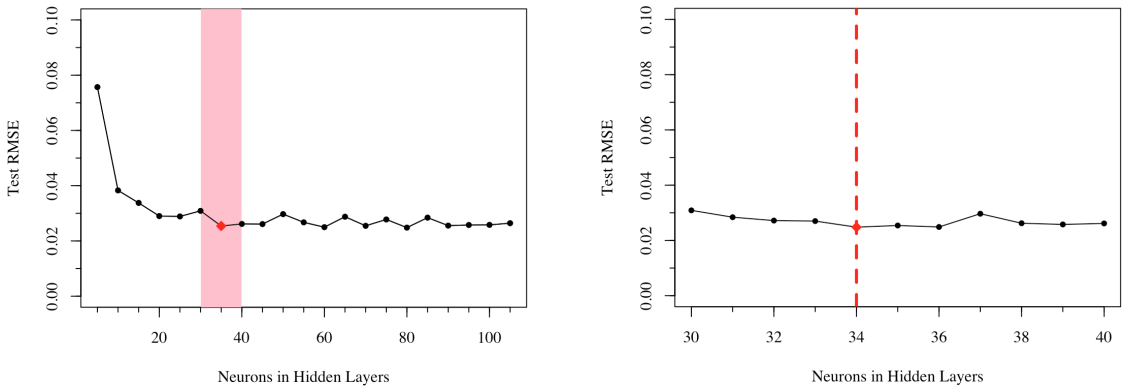

Figure 6: Test Error with respect to the number of neurons in each of 2 layers of the MLP.

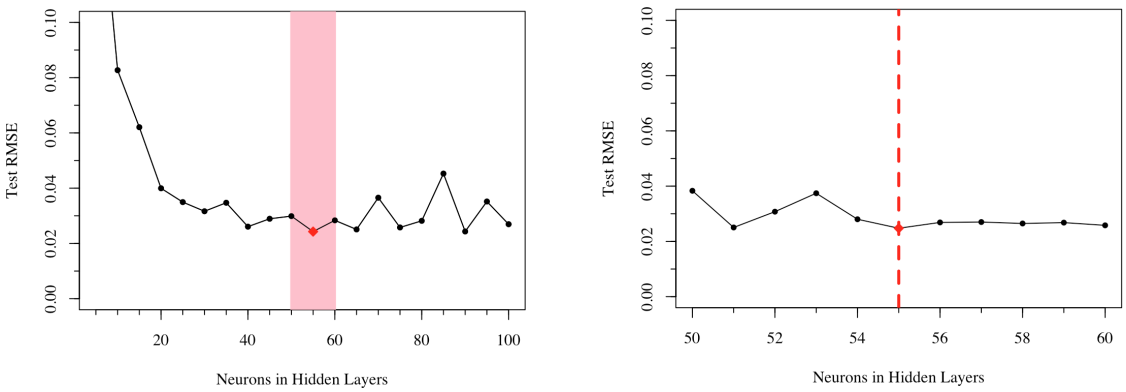

Figure 7: Test Error with respect to the number of neurons in each of 4 layers of the RNN.

4.6 Calibration Stage

The calibration step is concerned with the process of training. Here, we will specify the optimization algorithms used during neural networks training, as well as other hyper-parameters of the models.

1. *Optimization algorithm*: The standard backpropagation method was used for training the MLP model (the default in the R package RSNNS). RNN model was optimized using ADAM—a stochastic optimization algorithm [21]. In our study, ADAM was compared to several other learning functions and proved to demonstrate better performance. This observation is supported by existing research [22]; stochastic gradient descent was found to converge faster and to a more optimal local minima than standard gradient descent.

2. *Activation function:* The sigmoid activation function is used in the MLP model and in the output layer of the RNN model. Hidden layers of the Recurrent neural network use the ReLU activation function. Although the sigmoid function tends to be the standard activation function, multiple sources claim ReLU to exhibit better performance [1, 36]. The optimal combination of these two activation functions in our models is established by trial-and-error.

3. *Loss function:* The Summed Squared Error (SSE) is an immutable default error function used in RSNNS package [8] and, consequently, was used for MLP. For the RNN model, the Mean Squared Error (MSE) metric was specified, which is the closest to the RMSE measure used for model evaluations throughout this study.

4. *Number of epochs:* As mentioned in section 3, an epoch is a period of time marked by a propagation of full input data through neural network exactly once. Additional epochs increase optimization time, which, in general, results in lower values of loss function E on training data. However, when sufficiently many epochs are performed, models can exhibit *overfitting*—a state of dynamic parameters (i.e. weights and thresholds) tuned to replicate training data rather than the patterns underlying it. An overfitted model is marked by very low training error and poor performance on test sets. 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 8.

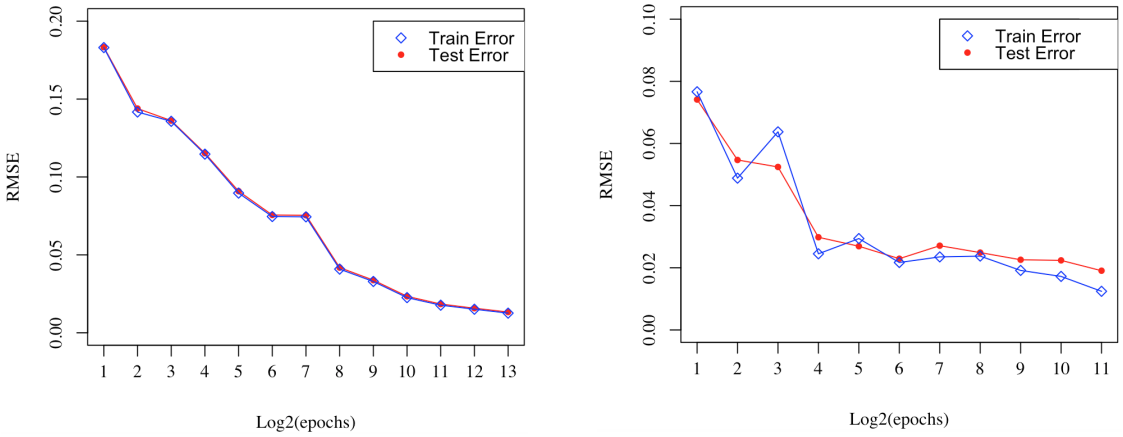

Figure 8: Test and train error curves with respect to $\text{Log}_2(epochs)$ for the MLP (left) and the RNN (right) models.

We observe that both train and test errors have an overall decreasing trend in the specified intervals of epochs (2 to 2^{13} for MLP and 2 to 2^{11} for RNN). In the framework of this study, extending these intervals is not computationally feasible. In fact, running our ANNs with more than 1,000 epochs already requires unreasonably costly computations. Therefore, we choose 100 epochs for the RNN and 1,000 epochs for the MLP model.

5. *Observations:* The overall number of observations that can be made available to our regression models is 99,000. 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 9. We observe that the learning curve appears to be sufficiently level when the full list of 99,000 datapoints are used. Note that these plots are obtained by running preliminary models with 80%/20% split, meaning each model is trained only on 80% of the corresponding number of observations.

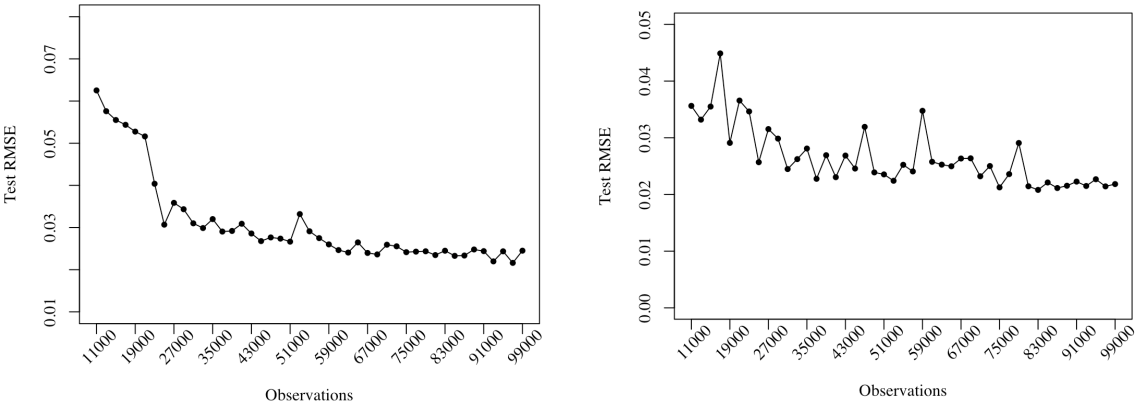

Figure 9: Test RMSE curve with respect to the number of observations available to ANNs for the MLP (left) and the RNN (right) models.

5 Model Validation And Predictions

In this section, we summarize the specifications of our regression models. Next, we evaluate the predictive and replicative accuracies of the Multilayer Perceptron and Recurrent Neural Networks implemented relative to a Support Vector Machines regression using the preselected data of 99,000 observations. Finally, a list of predicted zeros is provided.

5.1 Implementation of the Three Models

5.1.1 Implementation of the SVR Model

The SVM that we implement 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 [38]. 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 $c = 10$. 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 (centered in preprocessing). A set of 50 consecutive distance observations is chosen from the testing data and plotted against model's predictions in Figure 10.

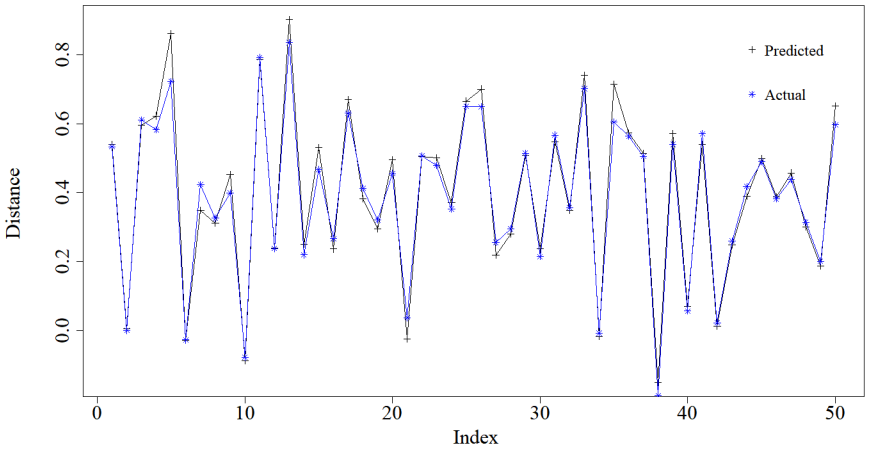

Figure 10: SVR predictions versus actual distance $\gamma_n - g_{n-1}$ for 50 randomly selected observations from the test data.

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) [7].

5.1.2 Implementation of the MLP

During the structure selection stage, we determine that the optimal configuration of the MLP neural network for our purposes has 34 neurons in each of the 2 hidden layers. At the calibration stage, backpropagation optimization algorithm was selected to train our MLP model for 1,000 epochs.

Since we apply a standard 80%/20% split of randomly shuffled data, the MLP model is trained on 79,200 observations of the 50 features selected during the input selection step. The remaining 19,800 observations are used as an independent test set to estimate predictive accuracy of the trained model. From this 19,800 data points, 50 observations for distance variable are plotted against the corresponding MLP predictions [Figure 11].

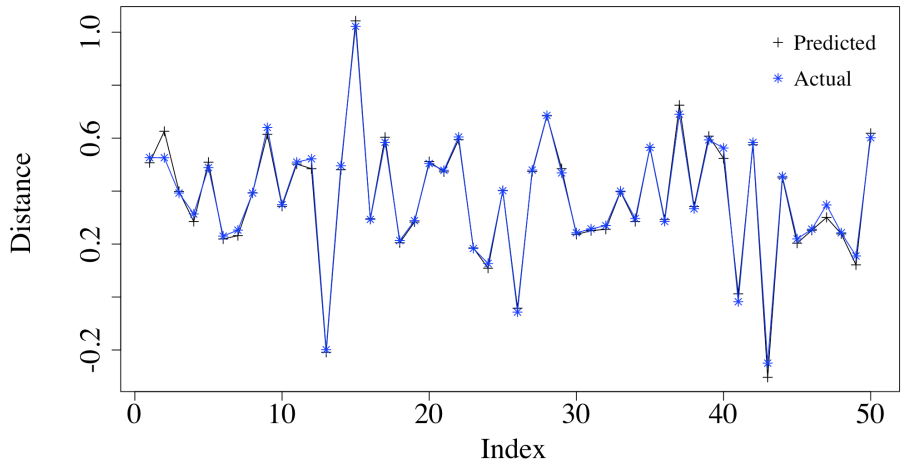

Figure 11: MLP predictions versus actual distance $\gamma_n - g_{n-1}$ for 50 randomly selected observations from the test data.

5.1.3 Implementation of the RNN

Our RNN model consists of 4 layers with 55 neurons in each. It is trained with an ADAM stochastic gradient descent optimizer for a total of 100 epochs. The key advantage of RNNs is their exceptional ability to extract temporal dependencies within input sequences, thus, we do not perform random shuffling of input data prior to feeding it to the model [10]. 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. A set of 50 consecutive distance observations is chosen from the testing data and plotted against model's predictions [Figure 12].

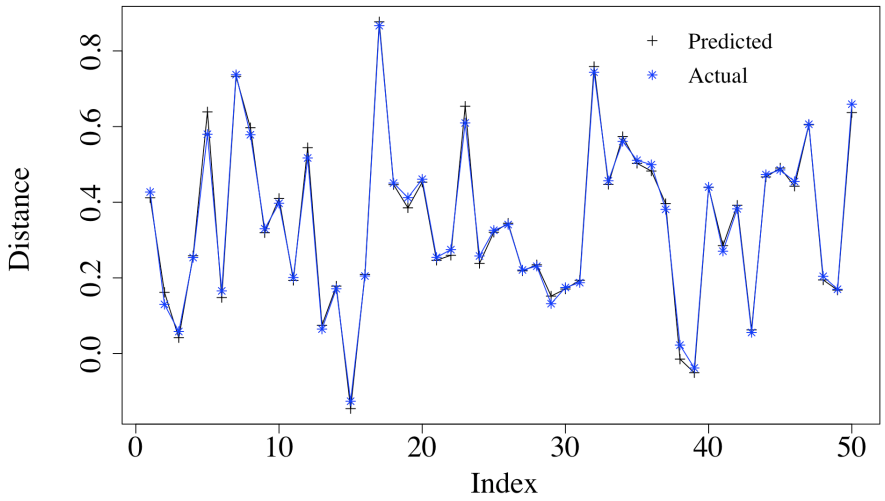

Figure 12: RNN predictions versus actual distance $\gamma_n - g_{n-1}$ for 50 consecutive observations from the test data.

Based on the predictions plots shown in figures 10, 11, 12, we conclude that all three regression models succeeded in generalizing patterns of the input data. As can be seen, they closely mimic the behavior of the unseen (test) distance observations.

5.2 Accuracy of 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 [44].

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 (20% of data set aside during data splitting) 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. Based on the metrics above, we conclude that both neural networks offer superior replicative accuracy relative to SVR.

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	0.3685	0.2304
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	0.0315	0.9817
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	0.3687	0.2266
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.

Consistent with the findings of our replicative accuracy analysis, the Recurrent Neural Network model is most successful in predicting unseen terms of the distance variable sequence. Predictive accuracy figures are listed in table 4. Still, all three regression methods display impressive predictive accuracy.

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. For SVM, cross-validation was not possible due to the unreasonably extensive computations that it required.

5.3 Predicted Zeros

Throughout this study, we use distance variable given by $\gamma_n - g_{n-1}$ as the target output variable of our models. This decision provides the ANN and SVM models with a time-series more suitable for regression. Given that Gram points—the only component of the distance variable except for zeros–are relatively easy to compute, predicting the distance variable is equivalent to predicting imaginary parts γ_n of the non-trivial zeros of ζ-function. In table 5, we provide 4 actual predictions of distance $\gamma_n - g_{n-1}$ and γ_n variables (the first and the last 2 observations from the test data) as compared to actual values. Since the RNN showed the best predictive accuracy (table 4), we present unscaled predictions of this regression model. The residual errors of our predictions are consistent across different observations and typically do not exceed 0.01 within test data.

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.

6 Conclusion And Future Work

In this study, we applied three regression models (Support Vector Machine, Feed-forward and Recurrent Neural Networks) to the prediction of locations of the nontrivial zeros of Riemann zeta-function on the critical line. Basic background on the Riemann zeta-function as well as Neural Networks is provided in the opening sections of this paper. The construction, implementation, and evaluation of our models is thoroughly described in sections 4 and 5.

For the purpose of achieving better predictive performance of our regression models, we choose the target output variable to be distance time-series $\gamma_n - g_{n-1}$, where γ_{n-1} refers to zeros of zeta-function and g_n is the n-th Gram point. We base our input selection on inspection of previous works [39] and the theoretical discussion of Riemann zeta-function given in section 2. Input data is obtained from the computations of Odlyzko [33] (zeros), Mathematica software (Z-function) and our custom numerical algorithm implemented in Python (Gram and co-Gram points). We used mRMR, and Random Forest algorithms to estimate the importance of the variables in predicting our target output. The final input feature space contains 99,000 observations for each of the 50 regressors. We evaluate our models with metrics such as R^2, $RMSE$, as well as by using cross-validation for the neural network models. Some general statistics regarding the set of our predictions (mean, standard deviation) are compared to those of the actual data. While the RNN model exhibited the best replicative and predictive accuracy, all models were found to output satisfactory predictions. The RNN predictions of distance $\gamma_n - g_{n-1}$ variable are converted into predictions of the non-trivial zeros of zeta-function. Typical residual error of these predictions typically does not exceed 0.01 for observations within the test set.

Future research could involve the implementation of other neural network models (e.g. Long/Short Term Memory, Radial Basis Function, etc.). Computational resources available in the framework of this study did not permit us to increase the number of training epochs to the point of overfitting. Therefore, future research could consider looking at larger numbers of epochs in order to select the optimal value. The evaluation stage of our RNN model can be improved by inspecting the decay of its predictive accuracy as separation between test and train data increases. Further, we would like 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.

Appendix A Forward Propagation

Following Nielsen [32], a more detailed explanation of forward propagation is described below.

Each neuron in a given layer transforms each real-valued input it receives, x using a weight and a bias term: $x \to x \cdot w + b$. Note that weight and bias may vary by layer and neuron, thus we denote the weight for the connection running from the kth neuron in the layer $l-1$ to the jth neuron in the layer l as w^l_{jk}. Similarly, b^l_j denotes the bias for the jth neuron in the lth layer.

Next, the neuron sums each these weighted, biased inputs and transforms them into a single output using a smooth activation function, often the logistic sigmoid function (range 0 to 1), denoted σ. We can write the output, or "activation" of the jth neuron in the lth layer as the sum of the weighted, biased activations produced in layer $l-1$:

$$a^l_j = \sigma\big(\sum_k w^l_j a^{l-1}_k + b^l_j\big) \tag{24}$$

In matrix notation, this is equivalent to:

$$\begin{aligned} a^l &= \sigma\big(w^l a^{l-1} + b^l\big) \\ &= \sigma(z^l), \\ \text{where } z &= w^l a^l - 1 + b^l. \end{aligned} \tag{25}$$

This output is then sent to each neuron in the subsequent layer. The output corresponding to a single input, x, is simply the output of the final layer.

Appendix B Backpropagation and Gradient Descent

Nielsen [32] provides the following derivation for the backpropagation algorithm for . We begin by defining prediction error using the quadratic cost function (i.e., the mean squared error):

$$C(w,b) = \frac{1}{2n}\sum_x ||y(x) - a^L(x)||^2, \tag{26}$$

where n is the number of observations. To minimize C with respect to w and b, we use the process of gradient descent. Gradient descent is an optimization algorithm in which a local minimum is identified by taking iterative steps proportional to the negative to the gradient of the function to minimize, here, C. Thus, we must first compute the partials $\frac{\partial C}{\partial b^l_j}$ and $\frac{\partial C}{\partial w^l_{jk}}$. Because both partials relate to the rate of change of C, it is useful to introduce a new variable to denote this quantity:

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} \tag{27}$$

Hence,δ_j^L denotes the errors in the output layer. Using the chain rule, we derive:

$$\begin{aligned} \delta_j^L &= \frac{\partial C}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^l} \\ &= \frac{\partial C}{\partial a_j^L}\sigma\prime(z_j^L) \end{aligned} \tag{28}$$

In matrix form, this is equivalent to

$$\begin{aligned} \delta_j^L &= \nabla_a C \odot \sigma\prime(z_j^L) \\ &= (a^L - y) \odot \sigma\prime(z_j^L), \end{aligned} \tag{29}$$

where $\odot$ is the Hadamard product (i.e. component-wise multiplication of vectors) and $\nabla_a C$ is the vector with jth component $\frac{\partial C}{\partial a_j^L} = (a^L - y)$ for the quadratic cost function.

Next, we derive a recursive formula for error, using the transpose of the weight matrix to move one layer backwards in the construction above:

$$\delta^l = ((w^{l+1})^T \delta^{l+1} \odot \prime(z^l)) \tag{30}$$

Finally, we arrive at the key partials for the gradient of the cost function:

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{31}$$

and

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l \tag{32}$$

New weights and biases are computed by taking small steps proportional to the negative of the gradient of the cost function at a given point.

In total, the algorithm works as follows:

1. Initialization: input training examples x_i, $i = 1, \cdots, m$ and randomly select initial weights, biases

2. Feed-forward: compute $z^l = w^l a^{l-1} + b^l$ and $a^l = \sigma(z^l)$ for each layer excluding the input layer, i.e., for $l = 2, 3, \cdots, L$

3. Backpropagation: compute the error recursively starting with the final layer. For each $l = L-1, L-2, \cdots, 2$, compute $\delta^l = ((w^{l+1})^T \delta^{l+1} \odot \sigma\prime(z^l)$ as in 30.

4. Output: for a given output, compute error terms backwards starting at the final layer and compute the gradient of the cost function as ($\frac{\partial C}{\partial b_j^l} = \delta_j^l$, $\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l$).

5. Gradient descent: for each layer $l = L-1, L-2, \cdots, 2$, update weights and biases according to the rule:

$$w^l \to w^l - \frac{\eta}{m}\sum_x \delta^{x,l}(a^{x,l-1})^T \tag{33}$$

and

$$b^l \to b^l - \frac{\eta}{m}\sum_x \delta^{x,l}, \tag{34}$$

where η is a small positive number, known as the learning constant.

6. Go to step 1 until convergence.

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] A. Awan. On the theory of zeta-functions and l-functions. 2015.

[5] C. A. L. Bailer-Jones, R. Gupta, and H. P. Singh. An introduction to artificial neural networks. 2001.

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

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

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

[9] 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.

[10] 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.

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

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

[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] Y. Heymann. The admissible domain of the non-trivial zeros of the Riemann zeta function. ArXiv e-prints, Apr. 2018.

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

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

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

[20] D. Karayannakis and I. S. Xezonakis. An algorithm for the evaluation of the gamma function and ramifications: Part I. Int. J. Math. Game Theory Algebra, 18(6):435–447 (2010), 2009.

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

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

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

[24] M. Korolev. On small values of the riemann zeta-function at gram points. Matematicheskii Sbornik, 205:67–86, 2014.

[25] D. Kriesel. A Brief Introduction to Neural Networks. 2007.

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

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

[28] 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.

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

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

[31] 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.

[32] M. Nielsen. Neural Networks and Deep Learning. Determination Press, 2015.

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

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

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

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

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

[38] 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.

[39] O. Shanker. Neural network prediction of Riemann zeta zeros. Adv. Model. Optim., 14(3):717–728, 2012.

[40] A. J. Smola and B. Schölkopf. A tutorial on support vector regression. Stat. Comput., 14(3):199–222, 2004.

[41] E. Stein and R. Shakarchi. Complex Analysis. Princeton lectures in analysis. Princeton University Press, 2010.

[42] 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.

[43] B. Warner and M. Misra. Understanding neural networks as statistical tools. The American Statistician, 50(4):284–293, 1996.

[44] 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.