information

MDPI

Article

Neural Network-Based Parameter Estimation in Dynamical Systems

Dimitris Kastoris , Kostas Giotopoulos and Dimitris Papadopoulos *

Department of Management Science and Technology, University of Patras, 26504 Patras, Greece; up1091036@upatras.gr (D.K.); kgiotop@upatras.gr (K.G.)
* Correspondence: dimfpap@upatras.gr

Abstract: Mathematical models are designed to assist decision-making processes across various scientific fields. These models typically contain numerous parameters, the values' estimation of which often comes under analysis when evaluating the strength of these models as management tools. Advanced artificial intelligence software has proven to be highly effective in estimating these parameters. In this research work, we use the Lotka–Volterra model to describe the dynamics of a telecommunication sector in Greece, and then we propose a methodology that employs a feed-forward neural network (NN). The NN is used to estimate the parameter's values of the Lotka–Volterra system, which are later applied to solve the system using a fourth-algebraic-order Runge–Kutta method. The application of the proposed architecture to the specific case study reveals that the model fits well to the experiential data. Furthermore, the results of our method surpassed the other three methods used for comparison, demonstrating its higher accuracy and effectiveness. The implementation of the proposed feed-forward neural network and the fourth-algebraic-order Runge–Kutta method was accomplished using MATLAB.

Keywords: Lotka–Volterra; feed-forward neural network; deep neural network; MATLAB; Runge–Kutta method

Citation: Kastoris, D.; Giotopoulos, K.; Papadopoulos, D. Neural Network-Based Parameter Estimation in Dynamical Systems. *Information* **2024**, *15*, 809. https://doi.org/10.3390/info15120809

Academic Editor: Haridimos Kondylakis

Received: 30 October 2024
Revised: 5 December 2024
Accepted: 11 December 2024
Published: 16 December 2024

1. Introduction

The primary purpose of mathematical modeling is to simplify and explain complex systems that occur in many scientific fields. Our goal is to develop a straightforward model that precisely fits a dataset within a defined margin of error, while also enabling the exploration of specific properties. Our research utilizes a feed-forward neural network to estimate the parameters' values of the Lotka–Volterra model. Neural networks (NNs) have emerged as a adjustable tool for solving complex systems, especially when integrated with experimental data [1]. By training a neural network on such data, the model can reveal hidden patterns and complex relationships that are difficult to capture through traditional approaches. NNs are particularly effective in controlling noisy, nonlinear, or high-dimensional datasets, which improve their ability to estimate parameters and make accurate predictions [2]. When used on experimental datasets, neural networks prove strong generalization capabilities, providing valuable information into system dynamics and enabling the construction of accurate models for a variety of scientific applications [3]. In the case of a Lotka–Volterra system, the parameter values typically represent factors such as competition, mutualism, predation, or growth between interacting species/populations. These parameters must be quantified to effectively apply the model as a decision-making tool in a specific context [4]. The challenge is to accurately quantify these parameters to solve the problem quantitatively and ensure that the model closely fits the raw data. Parameter estimation in dynamical systems like the Lotka–Volterra model faces significant challenges [5]. The inherent nonlinearity complicates parameter identification using traditional linear methods, often resulting in inaccurate or unstable estimates. Moreover, noisy

observational data hinder the computation of reliable derivatives essential for accurate estimation. Furthermore, high-dimensional systems increase computational complexity as the number of parameters scales with system size, necessitating more sophisticated and scalable approaches. In addition, limited data availability risks overfitting, where models capture noise instead of underlying dynamics, reducing generalizability. The proposed method effectively addresses these challenges by leveraging neural networks' ability to model complex nonlinear relationships, providing smooth approximations of state variables that mitigate noise impact. Furthermore, the architecture of the method is designed to be scalable and robust, handling high-dimensional parameter spaces and performing well even with limited datasets. In the literature, there are several techniques that are trying to achieve this target. For instance, Shatalov et al. [6] and Fedatov and Shatalov [7] propose a method that involves using direct integration to acquire goal functions and applying subsequent quadrature rules to determine the values of the unknown parameters. On the other hand, Michalakelis et al. [8] employ genetic algorithm techniques and advanced computer software to solve a nonlinear system.

To demonstrate the practical application of the proposed method, we use a Lotka–Volterra system that describes the dynamics among three competitors in the telecommunication market, relying on a comprehensive set of data. The problem is formulated as a dynamical system containing three nonlinear first-order differential equations, each including both linear and quadratic terms, comparable to the systems discussed by Bazykin [9] and Fay and Greeff [10]. Market concentration has long been a focus of researchers, particularly concerning the number of firms offering specific products or a range of products and services [11]. The structure of a market is crucial in determining market power, business behavior, and overall performance, which in turn facilitates the assessment of the level of competition across different industries. These considerations are especially relevant in the high-technology sector, such as telecommunications. Traditionally, telecommunications were operated as a national monopoly until recent years when market liberalization occurred. This change transformed the market from a monopolistic structure, characterized by important entry barriers, to an oligopolistic or, in some cases, a competitive environment. Therefore, the analysis of this emerged market is basic for identifying its unique characteristics, understanding competitor behaviors, and providing valuable insights for lawmaking and regulatory authorities [11,12].

The proposed neural network architecture is based on a feed-forward neural network with one input layer and one to three hidden layers, and an output layer consists of three nodes. In this structure, the input vector is processed through the hidden layers, where nonlinear transformations are applied, allowing the network to model complex patterns of the experimental data. The three output nodes serve the prediction of multiple target variables simultaneously, making the neural network ideal for tasks that include multiple outputs. Validation applied with a walk-forward algorithm that is suitable for small datasets and for time series data [13]. This architecture enables the model to operate in a data-driven learning mode, producing reliable predictions. Parameter estimation on dynamical systems using neural networks is a relatively new research field (for example, see [3,4]). The specific problem of market modeling is addressed using real data. The purpose of this research is to demonstrate the effectiveness of neural networks (NNs) in comparison with genetic algorithms [8] and other methods, such as integral and logarithmic integral approaches [14], which have already produced results for this specific problem.

2. Overview of the Lotka–Volterra Model Interpretation

The well-known Lotka–Volterra (LV) model is often utilized in order to represent the dynamics of n interacting species within an ecosystem. The LV model is represented by the following system of ordinary differential equations (ODEs):

$$\frac{dx_i}{dt} = x_i\left(a_{i0} + \sum_j a_{ij}x_j\right), \quad i,j = 1,2,\ldots,n, \tag{1}$$

The parameters a_{i0} describe either the intrinsic population growth or the decline of species i in the lack of interactions with other species (positive values indicate growth while negative values indicate decline). Furthermore, the parameters a_{ij} can be positive, negative, or zero, reflecting whether the species interact through predation, rivalry, or mutualism or have no interaction at all. For $n = 3$, the general Lotka–Volterra system of Equation (1) takes the following form:

$$\begin{aligned} \frac{dx}{dt} &= x(a_{10} + a_{11}x + a_{12}y + a_{13}z) \\ \frac{dy}{dt} &= y(a_{20} + a_{21}x + a_{22}y + a_{23}z) \\ \frac{dz}{dt} &= z(a_{30} + a_{31}x + a_{32}y + a_{33}z) \end{aligned} \tag{2}$$

where x, y, z are the three species/competitors. The Lotka–Volterra model is ideal for capturing competitive interactions among companies through its nonlinear dynamics, accurately reflecting real market behaviors [15]. Alternative approaches, such as linear regression, fail to model these complex interdependencies, resulting in oversimplified insights. Agent-based models, while detailed, require extensive computational resources and large datasets, making them less practical for certain applications. Additionally, equilibrium-based models lack the dynamic adaptability necessary to represent evolving market conditions. Thus, the Lotka–Volterra framework provides a balanced and effective means to analyze and estimate parameters in competitive market environments. The modeling approach for the specific telecommunication market has already been proposed and investigated by researchers [8,14].

3. Description of the Method

In this section, we present the proposed methodology for estimating a parameter's value in dynamic systems, which are defined by nonlinear ODEs. The method applies a feed-forward neural network and uses optimization techniques for the determination of the parameter values. The effectiveness of the proposed approach is demonstrated on a Lotka–Volterra system with three interacting competitors, where time-series data are available for each. The numerical results emphasize the efficacy of the method in capturing the system's dynamics.

Valid parameter estimation in dynamic systems is crucial across many scientific and engineering disciplines. Traditional approaches often involve linearization or demand wide prior knowledge of the system's structure, which can be difficult when dealing with nonlinearities and intricate variable interactions. In this study, we present a method that uses the capabilities of neural networks to model complex relationships, coupled with optimization techniques to estimate the parameters of a nonlinear dynamic system. At this point, we present an algorithm to provide a better understanding of the proposed approach (Algorithm 1).

Algorithm 1 Proposed approach

Initialize $t\_data \leftarrow [0, \ldots, 18]^T$, $data \leftarrow [x, y, z]$, normalize $data$
Define NN with x hidden layers using tanh activation
Initialize storage for $validation\_errors$, $params\_all$
for $i = 5$ to 18 **do**
    Train NN on $t\_data[0 : i-1]$, $data\_norm[0 : i-1, :]$
    Perform walk-forward validation on $t\_data[i]$, $data\_norm[i, :]$
    Predict $YPred\_train$, denormalize, compute derivatives
    Estimate parameters via optimization, store in $params\_all$
    Predict $YPred\_val$, denormalize, calculate error, store in $validation\_errors$
end for
Compute average error, plot parameter stability and predictions

In the algorithm, *YPred_train* refers to the neural network's predicted outputs on the training dataset, which are denormalized to their original scale for further analysis. *YPred_val* denotes the predictions made on the validation dataset, used to evaluate the model's performance. The variables $\frac{dx}{dt}$, $\frac{dy}{dt}$, and $\frac{dz}{dt}$ represent the computed derivatives based on the training predictions essential for parameter optimization. Additionally, *validation_errors* stores the error between the predicted and actual values on the validation set, while *params_all* maintains a record of all estimated model parameters throughout the training iterations.

3.1. Data Collection and Normalization

In our application, we use time-series data for the three variables in the system (2), which were collected over a time span of 19 6-month periods. Data for the variables $x(t)$, $y(t)$, and $z(t)$ were stored in the vectors x_{data}, y_{data}, and z_{data}, respectively. For more efficient network training, we normalize the data by subtracting the mean and dividing by the standard deviation of each variable. Normalization involves scaling input data, including time points and state variables, using Z-score normalization. This ensures that all features contribute equally to neural network training [16], enhancing numerical stability and accelerating convergence by preventing larger-scale features from dominating the learning process. The benefits include improved training efficiency, balanced feature contribution, and enhanced performance of activation functions like tanh or sigmoid by keeping inputs within their sensitive ranges. After the neural network generates predictions in the normalized scale, denormalization transforms these outputs back to the original data scale. This step is crucial for accurately interpreting results and computing derivatives in their true scale, which is essential for reliable parameter estimation in the Lotka–Volterra model. Normalization and denormalization are inverse processes using the same statistical parameters (μ and σ), ensuring consistency and enabling the neural network to learn efficiently on scaled data while accurately applying results in their original context.

3.2. Neural Network Architecture

A feed-forward neural network was designed with the following architecture:

- Input Layer: A single feature input representing time (t_{data}).
- Fully Connected Hidden Layers: Each with n possible neurons and a hyperbolic tangent activation function (`tanh`). Custom weight and bias initializers were used to ensure small initial weights and zero biases.
- Output Layer: A fully connected layer producing three outputs, corresponding to the predictions of $x(t)$, $y(t)$, and $z(t)$. The regression layer in each output is used to calculate the loss based on the difference between the predicted and normalized true values.

An example of the proposed architecture is presented in Figure 1.

In our approach, the neural network (NN) captures the time evolution of the system, represented by three variables (x_{data}, y_{data}, and z_{data}) based on time (t).

Basic Roles:

1. Prediction of system dynamics: The NN is trained on normalized data (x_{data}, y_{data}, z_{data}) to predict the values of x, y, and z over time (t). After training, it can generate predictions (YPred) for the system based on the time data.
2. Learning the relationships: The NN is developed to approximate the underlying dynamics between the variables (x, y, z) and time (t), allowing the model to capture the relationships between these variables that might be nonlinear or complex.
3. Providing inputs for further analysis: The NN's predictions are denormalized to provide continuous estimates of the system variables, which are then used to compute the derivatives ($\frac{dx}{dt}$, $\frac{dy}{dt}$, $\frac{dz}{dt}$). These derivatives are essential for parameter estimation using `fminunc` in the subsequent optimization process.

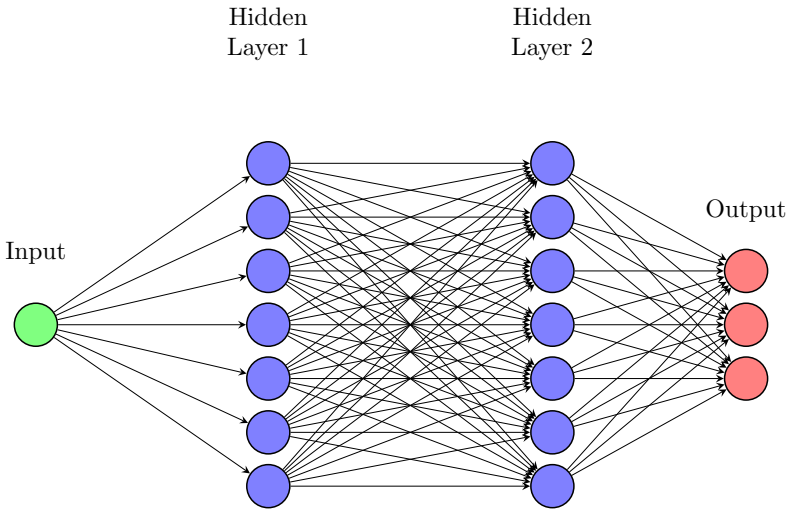

Figure 1. An example of a random feed-forward neural network with one input neuron, three output neurons, and two hidden layers with seven neurons on every layer.

In summary, the NN approximates the time-dependent behavior of the system variables and provides predictions that are crucial for calculating the derivatives needed for optimization.

Feed-Forward Propagation

A feed-forward neural network operates according to the following procedure [17]: For each data point $t^{(m)}$, the learning process begins by setting $r_1^{[0]} = t^{(m)}$. Next, to compute $\mathbf{r}^l$ from $\mathbf{r}^{l-1}$, we follow two key steps. The first step involves calculating an intermediate vector $\mathbf{c}^l$ by performing a matrix multiplication between the previous layer's activations $\mathbf{r}^{l-1}$ and the weight matrix W^l, and then adding the bias vector $\mathbf{b}^l$. This can be represented as follows:

$$c_j^l \equiv \sum_{k=1}^{h^{l-1}} W_{jk}^l r_k^{l-1} + b_j^l., \tag{3}$$

Alternatively, in matrix notation, Equation (3) becomes the following:

$$\mathbf{c}^l = W^l \mathbf{r}^{l-1} + \mathbf{b}^l. \tag{4}$$

The next step involves the application of an activation function σ^l to the values in $\mathbf{c}^l$ to produce the activations $\mathbf{r}^l$ from (4). This step can be expressed as follows:

$$\mathbf{r}^l = \sigma^l(\mathbf{c}^l) = \sigma^l\left(W^l \mathbf{r}^{l-1} + \mathbf{b}^l\right). \tag{5}$$

In summary, for each input data point $t^{(m)}$, we compute both $\mathbf{c}^l$ and the output $\mathbf{r}^l$ by using Equations (4) and (5), respectively. To emphasize the dependence on the input $t^{(m)}$, we can denote the activations and intermediate values as $\mathbf{r}^{l(m)} = \sigma^l(\mathbf{c}^{l(m)})$. Thus, using matrix and vector notation, we can summarize the forward pass across all data points as follows:

$$\begin{aligned} T &= [t_1, \ldots, t_N], \\ c^l &= [\mathbf{c}^{l(1)}, \ldots, \mathbf{c}^{l(N)}], \\ A^l &= [\mathbf{r}^{l(1)}, \ldots, \mathbf{r}^{l(N)}]. \end{aligned} \tag{6}$$

Using the notation of (6), we lead to the following recursive update rules:

$$\begin{aligned} A^0 &= T, \\ c^l &= W^l A^{l-1} + \mathbf{b}^l, \\ A^l &= \sigma^l(c^l), \quad \text{for } l = 1, \ldots, L. \end{aligned} \tag{7}$$

The output layer A^L represents the final output of the network, denoted as follows:

$$N(T, W^L, \mathbf{b}^L) = A^L. \tag{8}$$

Alternatively, in component form, we write the network output as follows:

$$N_j(t^{(m)}, W^L, \mathbf{b}^L) = \pi_j(A^L) = A_j^L(t^{(m)}, W, \mathbf{b}), \tag{9}$$

where π_j in (9) refers to the j-th component of the output vector.

3.3. Training Process, Prediction, and Denormalization

The neural network was trained utilizing the Adam optimization algorithm, running for a total of 10,000 and 20,000 epochs. The initial learning rate was set to 0.01 because different values resulted in quite large deviations, and we also tested three different sets of layers. Furthermore, different activation functions were used, resulting in significant deviations, leading to the conclusion that the tanh activation function should be used.

Walk-forward validation involves iteratively training the model on an expanding training set, including all data up to the current time point, and validating it on the subsequent data point [18]. This approach ensures that the model leverages all available historical information, continuously updating with new data and adapting to evolving patterns. By systematically moving through the dataset in this manner, walk-forward validation provides a realistic assessment of the model's performance, mimicking real-world scenarios where models are retrained as new data become available [13].

To track the training process and ensure convergence, a live plot displaying training progress was employed. Upon completing the training, the network was applied to predict the normalized values of $x(t)$, $y(t)$, and $z(t)$ for the entire time series. These predicted values were subsequently denormalized using the original mean and standard deviation of the dataset.

The proposed method is highly sensitive to the choice of activation functions and the initialization of weights and biases. Utilizing the hyperbolic tangent (tanh) activation function is preferable to others like sigmoid or ReLU because tanh centers data around zero, which accelerates convergence and improves gradient flow. Tanh provides stronger gradients than sigmoid, helping to overcome the vanishing gradient problem, and unlike ReLU, it avoids issues with dead neurons, ensuring consistent learning across all neurons. Initializing weights and biases with small random values is essential to prevent the activation functions from saturating early in training. Small initial weights help ensure that neurons start in a region where the activation function behaves more linearly, promoting stable and efficient gradient descent. This approach prevents exploding gradients and maintains effective learning dynamics. Additionally, small biases help in maintaining symmetry breaking without pushing neurons into saturated states prematurely. Together, the use of tanh and careful initialization of weights and biases enhance the stability, convergence speed, and accuracy of the proposed method, making it well suited for precise parameter estimation in complex dynamical systems.

From Table 1, we select the best architecture for each layer size based on the mean RMSE.

Table 1. Hyper-tuning.

Epochs	Layers	Neurons	RMSE X	RMSE Y	RMSE Z	Mean RMSE
10.000	1	20	0.0126	0.0119	0.0176	0.0140
10.000	2	20	0.0107	0.0109	0.0131	0.0116
10.000	3	20	0.0109	0.0096	0.0129	0.0111
10.000	1	30	0.0134	0.0123	0.0174	0.0144
10.000	2	30	0.0124	0.0116	0.0136	0.0125
10.000	3	30	0.0119	0.0109	0.0144	0.0124
10.000	1	40	0.0123	0.0119	0.0169	0.0137
10.000	2	40	0.0114	0.0107	0.0136	0.0119
10.000	3	40	0.0108	0.0112	0.0128	0.0116
20.000	1	20	0.0141	0.0119	0.0168	0.0143
20.000	2	20	0.0105	0.0104	0.0131	0.0113
20.000	3	20	0.0105	0.0088	0.0128	0.0107
20.000	1	30	0.0142	0.0124	0.0190	0.0152
20.000	2	30	0.0105	0.0113	0.0132	0.0117
20.000	3	30	0.0109	0.0098	0.0137	0.0115
20.000	1	40	0.0144	0.0126	0.0183	0.0151
20.000	2	40	0.0107	0.0110	0.0134	0.0117
20.000	3	40	0.0104	0.0091	0.0130	0.0110

RMSE evaluates the accuracy of predictions by measuring the average error between observed and predicted value based on a validation procedure:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{10}$$

Then, we extract the coefficients for the most efficient architecture corresponding to each layer size and solve the system using the fourth-order Runge–Kutta method. Based on the comparison of the mean RMSE values, we select the best model, which we subsequently compare with other methods and include in the graphs.

3.3.1. Backpropagation

Backpropagation is an algorithm used for training feed-forward neural networks [19]. It involves two main phases:

1. Forward Pass: The input data are passed through the network, and the output is calculated.
2. Backward Pass (Backpropagation): The error (difference between the predicted output and the actual output) is propagated backward through the network. The gradients of the loss function with respect to each weight are calculated, and the weights are updated using these gradients (often via a gradient-based optimization method like Adam or SGD).

Backpropagation relies on the chain rule of calculus to compute the derivatives of the loss function with respect to the weights.

Here is the general mathematical form:

Given that a^L is the output of the network at the final layer L (predicted output), $\mathbf{p}$ is a true label or ground truth, C is the cost function or loss function (e.g., Mean Squared Error or Cross-Entropy Loss), W^l and $\mathbf{b}^l$ are the weights and biases of layer l, $\mathbf{t}^l$ is the

activation at layer l, and $\mathbf{z}^l$ is the weighted input at layer l, where $\mathbf{z}^l = W^l\mathbf{t}^{l-1} + \mathbf{b}^l$ and σ is the activation function (e.g., sigmoid, ReLU, tanh).

1. Loss (Cost) Function:

The loss function C measures the error between the predicted output a^L and the true output $\mathbf{p}$. For a given training example, it can be represented as follows:

$$C = \frac{1}{2}\sum(a^L - \mathbf{p})^2 \tag{11}$$

This is just one example (for Mean Squared Error), but the form of C will vary depending on the task and choice of loss function.

2. Gradients in the Output Layer (Layer L):

The error at the output layer is defined as follows:

$$\delta^L = \nabla_{\mathbf{a}} C \circ \sigma'(\mathbf{z}^L) \tag{12}$$

where $\nabla_{\mathbf{a}} C = (a^L - \mathbf{p})$ in (12) is the derivative of the loss function (11) with respect to the output activations, $\circ$ denotes element-wise multiplication (Hadamard product), and $\sigma'(z^L)$ is the derivative of the activation function at the final layer.

3. Gradients in the Hidden Layers (Layer l):

For the hidden layers, the error is propagated backward using the following:

$$\boldsymbol{\delta}^l = (W^{l+1})^T\boldsymbol{\delta}^{l+1} \circ \sigma'\left(\mathbf{z}^l\right) \tag{13}$$

where $(W^{l+1})^T$ is the transpose of the weight matrix from layer $l+1$, $\boldsymbol{\delta}^{l+1}$ is the error from the next layer, and $\sigma'(\mathbf{z}^l)$ is the derivative of the activation function for layer l.

4. Gradients with Respect to Weights and Biases:

Once the errors $\boldsymbol{\delta}^l$ in (13) are calculated for each layer, the gradients of the loss with respect to the weights and biases are computed as follows:

$$\begin{aligned} \frac{\partial C}{\partial W^l} &= \boldsymbol{\delta}^l(\mathbf{t}^{l-1})^T, \\ \frac{\partial C}{\partial \mathbf{b}^l} &= \boldsymbol{\delta}^l. \end{aligned} \tag{14}$$

5. Weight and Bias Updates (with learning rate η):

$$\begin{aligned} W^l &= W^l - \eta\frac{\partial C}{\partial W^l}, \\ \mathbf{b}^l &= \mathbf{b}^l - \eta\frac{\partial C}{\partial \mathbf{b}^l}. \end{aligned} \tag{15}$$

3.3.2. Adam Optimizer

Based on the optimal learning parameters W^* and $\mathbf{b}^*$ from (14) and (15), we obtain the approximate solutions of the dynamical system. In our study, the Adaptive Moment Estimation (Adam) algorithm was employed to update these parameters. The update process follows the following rules:

$$\begin{aligned} M_w^k &= \beta_1 M_w^{k-1} + (1-\beta_1)\nabla J_1(W^k), \\ M_b^k &= \beta_1 M_b^{k-1} + (1-\beta_1)\nabla J_1(\mathbf{b}^k), \\ V_w^k &= \beta_2 V_w^{k-1} + (1-\beta_2)(\nabla J_1(W^k))^2, \\ V_b^k &= \beta_2 V_b^{k-1} + (1-\beta_2)(\nabla J_1(\mathbf{b}^k))^2, \end{aligned} \tag{16}$$

where β_1 and $\beta_2 \in [0,1)$ are the decay rates for the moment estimates. The corrected moment estimates for the weights and biases are then given by the following:

$$\hat{M}_w^k = \frac{M_w^k}{1-\beta_1^k}, \quad \hat{V}_w^k = \frac{V_w^k}{1-\beta_2^k},$$
$$\hat{M}_b^k = \frac{M_b^k}{1-\beta_1^k}, \quad \hat{V}_b^k = \frac{V_b^k}{1-\beta_2^k}. \tag{17}$$

Finally, the weights and biases are updated as follows:

$$W^{k+1} = W^k - \eta \frac{\hat{M}_w^k}{\sqrt{\hat{V}_w^k} + \epsilon},$$
$$\mathbf{b}^{k+1} = \mathbf{b}^k - \eta \frac{\hat{M}_b^k}{\sqrt{\hat{V}_b^k} + \epsilon}. \tag{18}$$

In Equation (18), η is the learning rate, ϵ is a small constant to prevent division by zero, and M_w, M_b, V_w, and V_b are the first and second moment vectors initialized to zero. Commonly used values for the hyperparameters include $\eta = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$, as suggested by the literature [20].

3.4. Prediction Process After Training

Once the neural network has been trained, we use the time data as the input. This input is a one-dimensional array where each value corresponds to a specific time point, representing the system's behavior at that instant. Essentially, the input is a sequence of time points that the network encountered during training. The trained neural network is then employed to predict the normalized output values for the variables $x(t)$, $y(t)$, and $z(t)$ over the entire time series. In Matlab R2024a, this is achieved by using the `predict` function with the time data as input. The network processes each time step individually and produces a corresponding set of three outputs, one for each variable (x, y, and z). Since the network's predictions are in their normalized form (because the training data were normalized before input), we must convert them back to their original scale. This is performed by reversing the normalization, applying the mean and standard deviation that were initially used to normalize the data. The final output is a matrix, where each row corresponds to a specific time point and the columns represent the denormalized predictions for $x(t)$, $y(t)$, and $z(t)$ at that point in time. These predictions can be directly compared with the actual measured values or used for further computations, such as determining derivatives to estimate system parameters.

3.5. Parameter Estimation via Optimization

The predicted values were utilized to calculate the numerical derivatives with respect to time, applying finite difference methods. These derivatives were then related to the parameters of the dynamic system through a set of nonlinear equations. The general form of these equations is as follows:

$$\frac{dx}{dt} = x(t) \cdot (a_{10} + a_{11} \cdot x(t) + a_{12} \cdot y(t) + a_{13} \cdot z(t))$$
$$\frac{dy}{dt} = y(t) \cdot (a_{20} + a_{21} \cdot x(t) + a_{22} \cdot y(t) + a_{23} \cdot z(t)) \tag{19}$$
$$\frac{dz}{dt} = z(t) \cdot (a_{30} + a_{31} \cdot x(t) + a_{32} \cdot y(t) + a_{33} \cdot z(t))$$

In Equation (20), the time derivatives of $x(t)$, $y(t)$, and $z(t)$ are expressed in terms of the unknown parameters a_{10} through a_{33} and the values of the variables at each time step.

The parameters a_{10} to a_{33} were determined by minimizing the sum of the squared differences between the left-hand and right-hand sides of the system of equations. This minimization was carried out using the quasi-Newton optimization technique, specifically the `fminunc` function in MATLAB. The mathematical formulation for minimizing the sum of squared differences between the left-hand and right-hand sides of the system of Equation (19) can be expressed as follows:

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{N} \left(\frac{dx(t_i)}{dt} - x(t_i) \cdot (a_{10} + a_{11} \cdot x(t_i) + a_{12} \cdot y(t_i) + a_{13} \cdot z(t_i)) \right)^2 \\ & + \sum_{i=1}^{N} \left(\frac{dy(t_i)}{dt} - y(t_i) \cdot (a_{20} + a_{21} \cdot x(t_i) + a_{22} \cdot y(t_i) + a_{23} \cdot z(t_i)) \right)^2 \\ & + \sum_{i=1}^{N} \left(\frac{dz(t_i)}{dt} - z(t_i) \cdot (a_{30} + a_{31} \cdot x(t_i) + a_{32} \cdot y(t_i) + a_{33} \cdot z(t_i)) \right)^2 \end{aligned} \tag{20}$$

where

- N is the number of time points;
- t_i represents each time point;
- $\frac{dx(t_i)}{dt}, \frac{dy(t_i)}{dt}, \frac{dz(t_i)}{dt}$ are the time derivatives of the variables x, y, z;
- $a_{10}, a_{11}, \ldots, a_{33}$ are the parameters to be estimated.

As mentioned earlier, we employed an iterative training approach for the neural network, in which the NN was trained multiple times using datasets of increasing size. Specifically, for each iteration i from 5 to 18, the NN was trained using the first i data points (from $t = 0$ to $t = i - 1$). This approach resulted in training datasets ranging from 5 to 18 data points. This method allows us to assess the model's performance as more data become available.

In our study, N represents the number of data points used in the least squares problem, corresponding to the number of time instances at which observations are available after computing derivatives. Although the training procedure uses a maximum of 18 data points (from $t = 0$ to $t = 17$), the computation of derivatives using finite differences reduced the dataset to 17 data points (N = 17) for the least squares problem.

Mathematical Formulation of Quasi-Newton Method

The quasi-Newton optimization method is a numerical approach designed for minimizing functions, particularly those that are not necessarily quadratic. It simulates the behavior of Newton's method but circumvents the direct computation of the Hessian matrix by iteratively refining an estimate of it. MATLAB's `fminunc` function typically utilizes the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm [21], or its limited-memory version (L-BFGS), to perform the optimization.

Let $f(\mathbf{p})$ be the function to be minimized, where $\mathbf{p}$ is a vector of parameters. The iterative update of the parameter vector $\mathbf{p}$ at the k-th step is given by the following:

$$\mathbf{p}_{k+1} = \mathbf{p}_k - \alpha_k \mathbf{B}_k^{-1} \nabla f(\mathbf{p}_k) \tag{21}$$

where

- $\mathbf{p}_k$ is the parameter vector at iteration k;
- α_k is the step size (also known as the learning rate);
- $\mathbf{B}_k^{-1}$ is an approximation of the inverse Hessian matrix;
- $\nabla f(\mathbf{p}_k)$ is the gradient of the function $f(\mathbf{p})$ evaluated at $\mathbf{p}_k$.

In quasi-Newton methods, instead of calculating the Hessian matrix $\mathbf{H}_k$ directly, the inverse Hessian approximation $\mathbf{B}_k^{-1}$ is updated iteratively using information from the gradient. This update typically follows the following BFGS algorithm:

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \Delta\mathbf{B}_k \tag{22}$$

In Equation (22), $\Delta\mathbf{B}_k$ represents the update to the inverse Hessian matrix, derived from differences between consecutive gradients and parameter values. In essence, the quasi-Newton method employed by `fminunc` works by iteratively adjusting the parameter vector $\mathbf{p}$ in (21) based on estimates of the inverse Hessian matrix and gradient information. This approach helps efficiently locate the minimum of the objective function without the computational cost of directly calculating second-order derivatives.

3.6. Solving the System with the Runge–Kutta Method

After estimating the parameters, they were incorporated back into the Lotka–Volterra system of differential equations to simulate the system's dynamics. To solve these equations, we used the fourth-order Runge–Kutta method [22], a robust and accurate numerical method for solving ordinary differential equations. This method was applied over the same time interval as the original dataset. For solving first-order ordinary differential equations (ODEs), we utilize the Runge–Kutta method, which is known for its accuracy and efficiency [22–24]. However, when dealing with second-order ODEs, the Runge–Kutta–Nyström [25–29] method is preferred as it is specifically designed to handle second-order systems directly, offering improved performance by leveraging the structure of the equations [26].

The solutions obtained from this method were then compared with the experimental data. This comparison was used to assess how well the estimated parameters fit the system's behavior, providing insight into the accuracy of the model. A visual comparison between the simulated trajectories and the data allowed us to evaluate the fit, and error metrics were calculated to quantify the accuracy.

Mathematical Formulation of the Runge–Kutta Method

The Runge–Kutta methods are used for the numerical solution of a first-order ordinary differential equation (ODE) of the following form:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0 \tag{23}$$

The Runge–Kutta method approximates the solution of (23) by using the following iterative scheme:

$$\begin{aligned} y_{n+1} &= y_n + h\sum_{i=1}^{s} b_i k_i, \\ k_i &= f\left(t_n + c_i h, y_n + h\sum_{j=1}^{i-1} a_{ij} k_j\right), \quad \text{for } i = 1, 2, \ldots, s. \end{aligned} \tag{24}$$

where
h is the step size;
s is the number of stages in the method; and
k_i are the stages of the method.
a_{ij}, b_i, and c_i are the coefficients that define a specific Runge–Kutta method.

From the general form of Runge–Kutta methods (24), we can derive the desired algebraic order scheme. The Runge–Kutta method that we used in this case is a fourth-algebraic-order Runge–Kutta method of four stages [22].

Given the initial condition $y(t_0) = y_0$, the value of y at the next time step $t_{n+1} = t_n + h$ is computed as follows:

$$\begin{aligned} y_{n+1} &= y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4), \\ k_1 &= f(t_n, y_n), \\ k_2 &= f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1\right), \\ k_3 &= f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2\right), \\ k_4 &= f(t_n + h, y_n + hk_3). \end{aligned} \tag{25}$$

This iterative process is repeated for each step, producing an approximation to the solution of the differential equation of the general form (23).

4. Example

4.1. Case Study Description

In system analysis, one common challenge is the lack of accessible and reliable datasets. However, a comprehensive historical dataset for three competitors in the Greek mobile telecommunications market was published by Michalakelis et al. [8]. In their work, they proposed a Lotka–Volterra model of competing species to explain the dynamics between these companies. The goal of their research was to predict the potential of maintaining a steady and strong competitive equilibrium in the market, using the available data to analyze market share control in the mobile phone industry over some specific years.

The study examines the historical market penetration and future prospects of the three competing service providers. The dataset covers the years 1995 to 2007, but since the second competitor, y, only entered the market in 1998, the analysis of the three competitors begins with the initial conditions from 1998. Each data point reflects the percentage of the market that each company managed at that time, as shown in Table 2, and serves to demonstrate the proposed methodology, while a and b refer to the observations for the first and second halves of the reference year, respectively. These data were taken from the paper "Lotka–Volterra Model Parameter Estimation Using Experiential Data" by Johanna C. Greeff and P. H. Kloppers [14].

Table 2. Historical market share data for competitors.

Period	Competitor x	Competitor y	Competitor z
1998a	0.52	0.15	0.33
1998b	0.47	0.21	0.32
1999a	0.43	0.27	0.30
1999b	0.40	0.31	0.29
2000a	0.38	0.35	0.28
2000b	0.37	0.36	0.27
2001a	0.36	0.37	0.27
2001b	0.36	0.37	0.27
2002a	0.35	0.38	0.27
2002b	0.36	0.38	0.25
2003a	0.37	0.39	0.24
2003b	0.38	0.39	0.23
2004a	0.39	0.39	0.22
2004b	0.38	0.39	0.23
2005a	0.37	0.39	0.24
2005b	0.38	0.39	0.23
2006a	0.38	0.40	0.22
2006b	0.36	0.39	0.25
2007a	0.34	0.38	0.28

Michalakelis et al. [8] applied this model, based on the principles outlined in Equation (2), where x, y, and z represent the market shares of the three competitors, respectively. To esti-

mate the unknown parameters, they utilized genetic algorithm techniques mentioned as the "Advanced Method" in the subsequent sections. These are their results:

$$\begin{aligned} \frac{dx}{dt} &= x(0.45 - 0.6x - 0.2y - 0.66z), \\ \frac{dy}{dt} &= y(0.86 - 0.02x - 1.8y - 0.59z), \\ \frac{dz}{dt} &= z(0.2 - 0.06x - 0.13y - 0.5z). \end{aligned} \tag{26}$$

Except for the "Advanced Method" that produced the coefficients in system (26), Kloppers and Greeff [30] introduce the integral and log integral techniques as effective tools for addressing complex systems, improving accuracy by taking into account non-uniform system behaviors. These techniques can be applied to parameter estimation challenges in dynamic models such as the Lotka–Volterra system, which is often used in telecommunication modeling.

Using the proposed integral method, the resulting system can be expressed as follows:

$$\begin{aligned} \frac{dx}{dt} &= x(4.1492 - 4.2136x - 3.8654y - 4.5320z), \\ \frac{dy}{dt} &= y(0.9902 - 0.0360x - 2.0410y - 0.7621z), \\ \frac{dz}{dt} &= z(-2.6812 + 2.5223x + 2.8576y + 2.6392z). \end{aligned} \tag{27}$$

and for the log integral method, the system is as follows:

$$\begin{aligned} \frac{dx}{dt} &= x(4.3432 - 4.4069x - 4.0595y - 4.7270z), \\ \frac{dy}{dt} &= y(0.5734 + 0.3298x - 1.5848y - 0.3285z), \\ \frac{dz}{dt} &= z(-3.6170 + 3.4259x + 3.8169y + 3.5911z). \end{aligned} \tag{28}$$

By observing the estimated parameter values in the systems (26)–(28), we notice that there are significant differences among them. This highlights the common understanding that mathematical systems often have multiple solutions. Additionally, the initial problem was viewed as a typical scenario of competing species, where intra-species competition could occur, as evidenced by the incorporation of the terms involving x^2, y^2, and z^2 [30–32]. It is important to mention that the the species y benefits from the existence of the species x, but the reverse is not true. This interaction is known as one-sided mutualism between prey species in the presence of the predator z. In the context of telecommunication service providers in Greece, this can be understood as follows: The provider y entered the market when the providers x and z were already recognized, yet it gradually secured its share of the market. It benefited from a one-sided mutualistic relationship with its more established competitor x, which effectively protected it from the predator z.

Table 3 shows the three best architectures based on the mean RMSE after solving the system with the fourth-order Runge–Kutta method and from where we choose the best one for the graphs and the final results. The results from the three architectures are similar. In terms of time complexity, the architecture with 10,000 epochs is the most efficient; however, in terms of accuracy, the one with 20,000 epochs and two neurons is a little more effective.

Table 3. Model evaluation metrics.

Epochs	Layers	Neurons	Mean RMSE x, y, z
10,000	1	40	0.0112
20,000	2	20	0.0111
20,000	3	20	0.0112

According to the previous analysis, we demonstrate numerical results based on the neural network with two hidden layers and 20 neurons (denoted as FFNN).

Finally, the system with the estimated parameters from the FFNN method is given as follows:

$$\begin{aligned} \frac{dx}{dt} &= x(2.8869 - 3.2245x - 2.6295y - 2.8031z) \\ \frac{dy}{dt} &= y(2.2487 - 1.4324x - 3.2173y - 1.9492z) \\ \frac{dz}{dt} &= z(-1.0575 + 1.5341x + 1.1662y - 0.1452z) \end{aligned} \tag{29}$$

The final estimated parameters from our FFNN model offer valuable insights into the competitive dynamics of the telecommunications market [8].

Intrinsic Growth Rates:

Parameters $a_{10} = 2.8869$ and $a_{20} = 2.2487$ indicate strong natural growth potentials for companies x and y, respectively.

Self-Limitation Effects:

Negative values $a_{11} = -3.2245$ and $a_{21} = -1.4324$ reflect significant internal constraints on x and y, such as market saturation or resource limitations.

Competitive Interactions:

Parameters $a_{12} = -2.6295$, $a_{13} = -2.8031$, $a_{22} = -3.2173$, and $a_{23} = -1.9492$ signify intense competitive pressures among firms, likely due to overlapping services or customer bases.

Declining Growth:

Parameter $a_{30} = -1.0575$ for z suggests a declining natural growth without competition.

Supportive Interactions:

Positive values $a_{31} = 1.5341$ and $a_{32} = 1.1662$ imply that x and y may have a supportive influence on z's growth.

Minimal Constraints:

Parameter $a_{33} = -0.1452$ for z indicates negligible internal constraints.

Overall, these parameters elucidate the balance between intrinsic growth, competitive forces, and internal constraints that shape the strategic landscape of the telecommunications market.

In Figure 2, we show the evolution of the market shares based on the estimated parameter's values derived from the methodology described in Section 3.

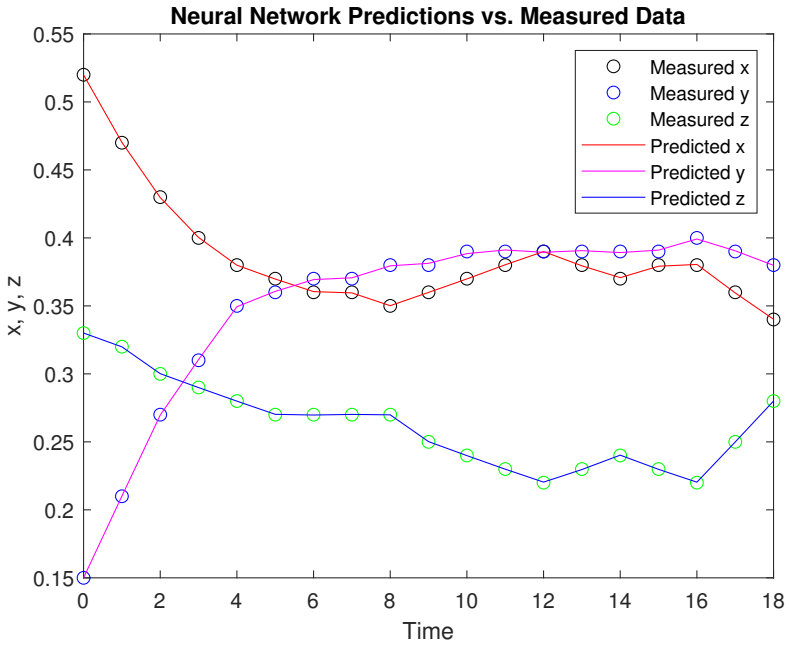

Figure 2. Neural network estimates and the measured data for the three competitors x, y, and z.

4.2. Results and Comparison of the Four Methods

In this section, we present a comparison of the four methods—FFNN, advanced, integral, and log integral—by evaluating their performance on the variables x, y, and z, which represent telecommunications companies. The FFNN method shows the closest alignment with the observed data, while the advanced method offers a reasonable estimate with slightly larger deviations. The integral and log integral methods, though less accurate, still provide meaningful approximations, with the log integral method slightly outperforming the integral method. The four Lotka–Volterra systems were solved by a fourth-order Runge–Kutta method, and to better illustrate these findings, we include diagrams of the results (Figure 3 for competitor x, Figure 4 for competitor y, Figure 5 for competitor z) and a table displaying the Root Mean Squared Error (RMSE) for each method (Table 4).

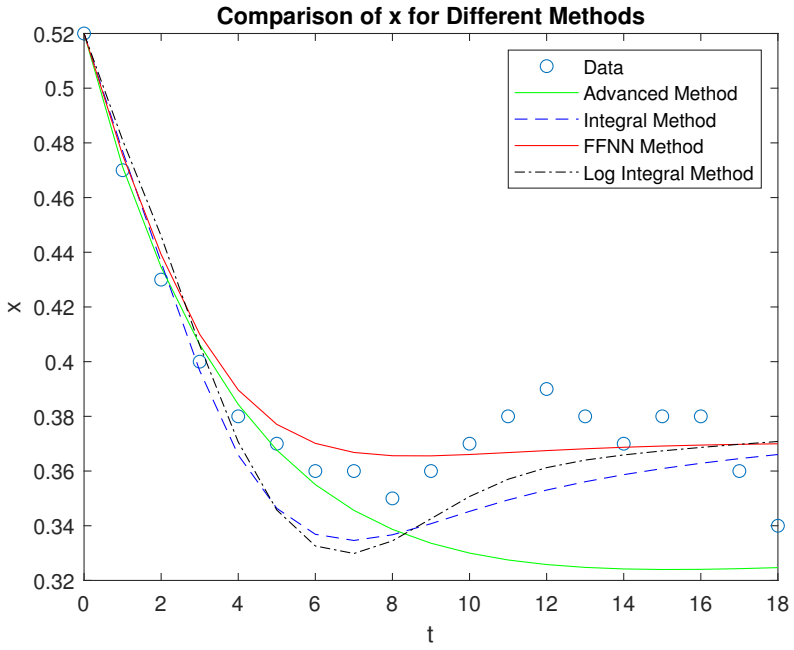

Figure 3. Comparison of the four methods with the measured data for the x competitor.

This comprehensive analysis highlights the superiority of the FFNN method in accurately modeling the behavior of the telecommunications companies.

As observed in Figure 3, after point 4 on the time axis corresponding to the year 2001, the market shares of telecommunications providers have shown minimal fluctuations, suggesting that the market is reaching a state of equilibrium.

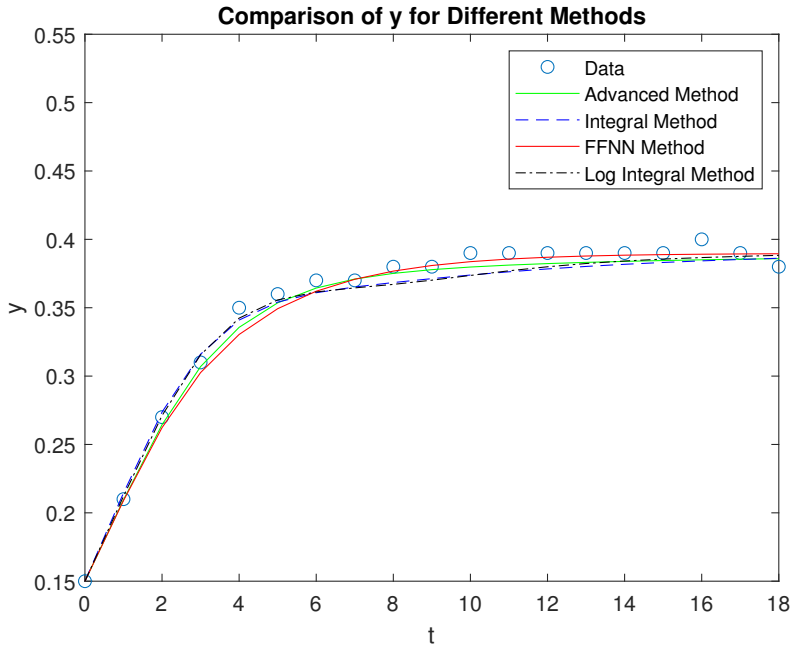

Figure 4. Comparison of the four methods with the measured data for the *y* competitor.

This stability aligns with findings from [33], which examine how a company's response time and strategy to competitors' marketing initiatives can affect market dynamics. Similar to those findings, the introduction of a new product or pricing strategy in oligopolistic markets poses a significant challenge to rivals, often leading to faster and more assertive reactions.

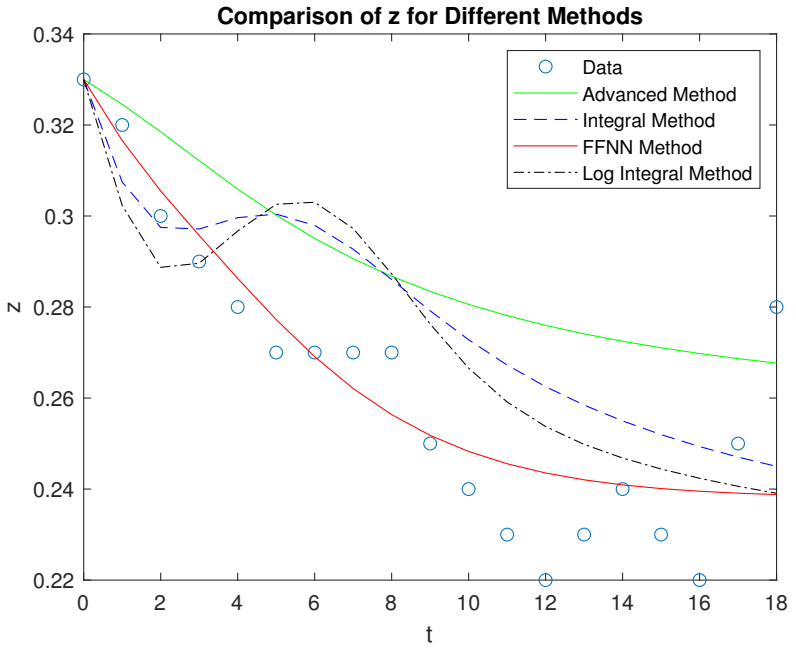

Figure 5. Comparison of the four methods with the measured data for the *z* competitor.

In markets with few competitors, firms that are highly interdependent tend to closely monitor each other's activities, enhancing their ability to respond swiftly. This behavior aligns with the idea that market outcomes, like product sales, are shaped by the interplay of marketing strategies and competitive actions, as discussed in [34].

Comparison of the four methods—FFNN, advanced, integral, and log integral—reveals that the proposed method delivers the most accurate performance across all three competitors (x, y, and z). In particular, it closely aligns with the actual data points, effectively capturing both the trends and the inflection points, especially for the variables x and z. The advanced method, while reasonable, tends to deviate more, particularly in modeling z, where it underestimates the initial drop and overestimates the recovery. The integral and log integral methods show moderate accuracy, but they both diverge from the data significantly in the early phases of x and z, where they struggle to match the initial dynamics.

Table 4. Root Mean Squared Error (RMSE) values for different methods and variables x, y, and z.

Method	RMSE for x	RMSE for y	RMSE for z
FFNN	0.01223	0.00715	0.01401
Advanced	0.03445	0.00719	0.03216
Integral	0.01995	0.00918	0.02493
Log Integral	0.01868	0.00853	0.02318

Overall, the proposed method stands out for its ability to model the competitor's market shares with the least deviation from the actual data, making it the most reliable approach among the four.

5. Discussion

Future research will investigate advanced neural network architectures to enhance parameter estimation in dynamical systems. Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks will be explored for their ability to capture temporal dependencies in time-series data, potentially improving accuracy in dynamic environments. Convolutional Neural Networks (CNNs) may be adapted to identify spatial patterns and handle high-dimensional data more efficiently. Physics-Informed Neural Networks (PINNs) will be examined for integrating physical laws directly into the training process, ensuring that parameter estimates adhere to known system dynamics. Additionally, Autoencoders could be utilized for dimensionality reduction, uncovering latent structures that facilitate more efficient parameter estimation. Ensemble learning methods will be considered to combine multiple models, enhancing the reliability and stability of estimates. By leveraging these diverse neural network approaches, future work aims to build upon the FFNN method, offering more robust, accurate, and scalable solutions for parameter estimation in complex dynamical systems.

6. Conclusions

The work presented in this paper introduces a novel approach for estimating market shares in the telecommunications sector, drawing on principles from population dynamics and ecological modeling. The core assumption is to treat market providers as interacting species competing for a shared resource—the market itself—and to analyze the system's dynamics accordingly. Our method for parameter estimation using a feed-forward neural network demonstrates superior performance compared with advanced, integral, and log-integral methods. This is evident from the lower Root Mean Squared Error (RMSE) values, indicating more accurate predictions. As shown in Table 3, for various architectures, the RMSE values are quite close to each other. In terms of time complexity, the architecture with 10,000 epochs is the most efficient; however, in terms of accuracy, the one with 20,000 epochs and two neurons is a little bit more effective.

Future research also includes developing methodologies based on alternative versions of the Lotka–Volterra model to examine different markets. Additionally, the effectiveness of the proposed approach should be assessed in other high-tech industries that share similar characteristics with the telecommunications sector, such as stringent regulatory requirements. With regulatory restrictions, strict government regulations or licensing requirements limit new competitor's ability to enter the market. These barriers can create a significant challenge for startups or smaller companies, ensuring that only established players or those with substantial resources can compete. Industries like pharmaceuticals, where regulatory approvals for new drugs are stringent, or the energy sector, with heavy government oversight and infrastructure needs, also experiences similar entry barriers as those in telecommunications.

Author Contributions: D.K., K.G. and D.P. conceived the idea, designed and performed the experiments, analyzed the results, drafted the initial manuscript, and revised the final manuscript. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by Grant No. 81845 from the Research Committee of the University of Patras via the "C. CARATHEODORI" program.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in this study are included in the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Giotopoulos, K.C.; Michalopoulos, D.; Vonitsanos, G.; Papadopoulos, D.; Giannoukou, I.; Sioutas, S. Dynamic Workload Management System in the Public Sector. *Information* **2024**, *15*, 335. [CrossRef]
2. Michalopoulos, D.; Karras, A.; Karras, C.; Sioutas, S.; Giotopoulos, K.C. Neuro-Fuzzy Employee Ranking System in the Public Sector. *Front. Artif. Intell. Appl.* **2022**, *358*, 325–333.
3. Lanouette, R.; Thibault, J.; Valade, J.L. Process modeling with neural networks using small experimental datasets. *Comput. Chem. Eng.* **1999**, *23*, 1167–1176. [CrossRef]
4. Livingstone, D.; Manallack, D.; Tetko, I.V. Data modelling with neural networks: Advantages and limitations. *Comput. Aided Mol. Des.* **1997**, *11*, 135–142. [CrossRef] [PubMed]
5. Arnold, A. When Artificial Parameter Evolution Gets Real: Particle Filtering for Time-Varying Parameter Estimation in Deterministic Dynamical Systems. *Inverse Probl.* **2023**, *39*, 14002. [CrossRef]
6. Shatalov, M.; Greeff, J.C.; Fedotov, I.; Joubert, S.V. Parametric identification of the model with one predator and two prey species. In Proceedings of the Technology and its Integration into Mathematics Education Conference (TIME), Buffelspoort, South Africa, 22–26 September 2008; Volume 10, pp. 101–110.
7. Shatalov, M.; Fedatov, I. On identification of dynamical system parameters from experiential data. *Res. Group Math. Inequalities Appl.* **2007**, *10*, 1–9.
8. Michalakelis, C.; Sphicopoulos, T.S.; Varoutas, D. Modelling competition in the telecommunications market based on the concepts of population biology. *Trans. Syst. Man Cybern. Part C Appl. Rev.* **2011**, *41*, 200–210. [CrossRef]
9. Bazykin, A. *Nonlinear Dynamics of Interacting Populations (Series in Neural Systems)*; World Scientific Pub Co Inc.: Singapore, 1998.
10. Fay, T.H.; Greeff, J.C. Lion, wildebeest and zebra: A three species model. *Ecol. Model.* **2006**, *196*, 237–244. [CrossRef]
11. Curry, B.; George, K.D. Industrial concentration—A survey. *J. Ind. Econ.* **1983**, *31*, 203–255. [CrossRef]
12. Tirole, J. *The Theory of Industrial Organization*; MIT Press: Cambridge, MA, USA, 1988.
13. Kouassi, K.H.; Moodley, D. An analysis of deep neural networks for predicting trends in time series data. In *SACAIR CCIS Springer Proceedings*; Springer: Berlin/Heidelberg, Germany, 2020; p. 2009.07943.
14. Kloppers, P.; Greeff, J. Lotka–Volterra model parameter estimation using experiential data. *Appl. Math. Comput.* **2013**, *224*, 817–825. [CrossRef]
15. Olivença, D.V.; Davis, J.D.; Voit, E.O. Inference of dynamic interaction networks: A comparison between Lotka-Volterra and multivariate autoregressive models. *Front. Bioinform.* **2022**, *2*, 1021838. [CrossRef] [PubMed]
16. Huang, L.; Qin, J.; Zhou, Y.; Zhu, F.; Liu, L.; Shao, L. Normalization Techniques in Training DNNs: Methodology, Analysis and Application. *IEEE Trans. Pattern Anal. Mach. Intell.* **2023**, *45*, 10173–10196. [CrossRef] [PubMed]
17. Bebis, G.; Georgiopoulos, M. Feed-forward neural networks. *IEEE Potentials* **1994**, *13*, 27–31. [CrossRef]

18. Bergmeir, C.; Benítez, J.M. On the use of cross-validation for time series predictor evaluation. *Inf. Sci.* **2012**, *191*, 192–213. [CrossRef]
19. Ye, J.C. Artificial Neural Networks and Backpropagation. In *Geometry of Deep Learning. Mathematics in Industry*; Springer: Singapore, 2022; Volume 37, pp. 91–112.
20. Adam: A Method for Stochastic Optimization. Available online: https://www.simiode.org/resources/3892.2014 (accessed on 20 September 2024).
21. Liu, D.C.; Nocedal, J. On the limited memory BFGS method for large scale optimization. *Math. Program.* **1989**, *45*, 503–528. [CrossRef]
22. Butcher, J.C. *Numerical Methods for Ordinary Differential Equations*; John Wiley & Sons: Hoboken, NJ, USA, 2016; Volume 10, pp. 143–331.
23. Tan, D.; Chen, Z. On A General Formula of Fourth Order Runge-Kutta Method. *J. Math. Sci. Math. Educ.* **2012**, *7*, 1–10.
24. Simos, T.E. A family of fifth algebraic order trigonometrically fitted Runge-Kutta methods for the numerical solution of the Schrödinger equation. *Comput. Mater. Sci.* **2005**, *34*, 342–354. [CrossRef]
25. Dormand, J.R.; El-Mikkawy, M.E.A.; Prince, P.J. Families of Runge-Kutta-Nyström formulae. *IMA J. Numer.* **1987**, *7*, 235–250. [CrossRef]
26. Houwen, P.J.V.; Sommeijer, B.P. Explicit Runge-Kutta-Nyström methods with reduced phase errors for computing oscillating solutions. *SIAM J. Numer.* **1987**, *24*, 596–617.
27. Fehlberg, E. *Classical Eight and Lower-Order Runge–Kutta–Nyström Formulas with Stepsize Control for Special Second-Order Differential Equations*; NASA Technical Report 381; NASA: Washington, DC, USA, 1972.
28. Papadopoulos, D.F.; Simos, T.E. A new methodology for the construction of optimized Runge-Kutta-Nyström methods. *Int. J. Mod. Phys. C* **2011**, *22*, 623–634. [CrossRef]
29. Papadopoulos, D.F.; Anastassi, Z.A.; Simos, D.F. The use of phase-lag and amplification error derivatives in the numerical integration of ODEs with oscillating solutions. *AIP Conf. Proc.* **2009**, *1168*, 547–549.
30. Kloppers, P.H.; Greeff, J.C. Estimation of parameters in population models. In Proceedings of the IASTED Technology Conferences, Banff, AB, Canada, 15–17 July 2010; pp. 87–91.
31. Murray, J.D. *Mathematical Biology*; Springer-Verlag: New York, NY, USA, 1993.
32. Starfield, A.M.; Bleloch, A.L. *Building Models for Conservation and Wildlife Management*; Macmillan Publishing Company: Brisbane, Australia, 1986.
33. Bowman, D.; Gatignon, H. Determinants of competitor response time to a new product introduction. *J. Market. Res.* **1995**, *32*, 42–53. [CrossRef]
34. Gatignon, H.; Hanssens, D.M. Modeling marketing interactions with application to salesforce effectiveness. *J. Market. Res.* **1987**, *24*, 247–257. [CrossRef]