



## Article

# Using Physics-Informed Neural Networks for Modeling Biological and Epidemiological Dynamical Systems

Amer Farea <sup>1</sup>, Olli Yli-Harja <sup>1,2</sup> and Frank Emmert-Streib <sup>1,\*</sup><sup>1</sup> Predictive Society and Data Analytics Laboratory, Faculty of Information Technology and Communication Sciences, Tampere University, 33720 Tampere, Finland; amer.farea@tuni.fi (A.F.); olli.yli-harja@tuni.fi (O.Y.-H.)<sup>2</sup> Institute for Systems Biology, Seattle, WA 98195, USA

\* Correspondence: frank.emmert-streib@tuni.fi

**Abstract:** Physics-Informed Neural Networks (PINNs) have emerged as a powerful approach for integrating physical laws into a deep learning framework, offering enhanced capabilities for solving complex problems. Despite their potential, the practical implementation of PINNs presents significant challenges. This paper explores the application of PINNs to systems of ordinary differential equations (ODEs), focusing on two key challenges: the forward problem of solution approximation and the inverse problem of parameter estimation. We present three detailed case studies involving dynamical systems for tumor growth, gene expression, and the SIR (Susceptible, Infected, Recovered) model for disease spread. This paper outlines the core principles of PINNs and their integration with physical laws during neural network training. It details the steps involved in implementing PINNs, emphasizing the critical role of network architecture and hyperparameter tuning in achieving optimal performance. Additionally, we provide a Python package, ODE-PINN, to reproduce results for ODE-based systems. Our findings demonstrate that PINNs can yield accurate and consistent solutions, but their performance is highly sensitive to network architecture and hyperparameters tuning. These results underscore the need for customized configurations and robust optimization strategies. Overall, this study confirms the significant potential of PINNs to advance the understanding of dynamical systems in biology and epidemiology.



Academic Editors: Jian Luo, Zheming Gao and Xin Yan

Received: 11 April 2025

Revised: 29 April 2025

Accepted: 13 May 2025

Published: 19 May 2025

**Citation:** Farea, A.; Yli-Harja, O.; Emmert-Streib, F. Using Physics-Informed Neural Networks for Modeling Biological and Epidemiological Dynamical Systems. *Mathematics* **2025**, *13*, 1664. <https://doi.org/10.3390/math13101664>

**Copyright:** © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

**Keywords:** Physics-Informed Neural Networks (PINNs); ordinary differential equation; deep learning; tumors growth model; gene expression model; epidemiological model

**MSC:** 34A55; 34A34; 37M05; 62-08; 62F30; 62P10; 62R07; 92B20; 68T05; 68T07; 82C32

## 1. Introduction

Physics-Informed Neural Networks (PINNs) represent an innovative approach that integrates physical laws directly into the training of neural networks [1–4]. Unlike traditional neural networks [5–10], which rely solely on data-driven methods, PINNs incorporate knowledge of differential equations—such as Ordinary Differential Equations (ODEs) [11,12] or Partial Differential Equations (PDEs) [13,14]—into the neural network (NNs) model. This integration allows PINNs to utilize both empirical data and established physical principles, providing solutions that are consistent with known equations governed by physical laws.

The relevance of PINNs in solving differential equations is significant, particularly in modeling physical phenomena where these equations describe the evolution of quan-

tities over time or space. Examples include heat flow, fluid dynamics [1,15–18], energy systems [19,20], material deformation [21–23], and geophysics [24–26].

Unlike traditional numerical methods such as the finite element method (FEM) or finite difference method (FDM) [27], which can be computationally intensive, especially in high-dimensional problems, PINNs offer a more flexible and data-adaptive approach by embedding the governing physical laws directly into the loss function of a neural network. While classical analytical and numerical techniques remain foundational, their effectiveness diminishes in settings involving sparse data, uncertain parameters, or partially known models. In such cases, PINNs provide a compelling alternative capable of approximating solutions without extensive preprocessing or complex numerical setups [1,3,28].

Many studies have examined the accuracy and computational efficiency of PINNs in comparison to traditional numerical methods such as the Finite Element Method (FEM). For instance, Grossmann et al. [29] conducted a comprehensive comparison and found that although PINNs can offer faster evaluation for some problems, they generally fall short of FEM in terms of overall solution accuracy and computational time. In contrast, Stiasny and Chatzivasileiadis [30] demonstrated that for specific applications like power system dynamics, PINNs can outperform traditional solvers in speed—being 10 to 1000 times faster—while still maintaining sufficient numerical accuracy and stability. However, as noted by Markidis [31], PINNs tend to converge quickly for low-frequency components but require significantly more training to accurately resolve high-frequency features, suggesting that hybrid approaches may be beneficial. Li [32] further emphasized the strengths of PINNs in modeling complex, high-dimensional systems and incorporating physical laws into data-driven models, but also highlighted challenges such as computational overhead and reduced precision in certain scenarios. Overall, while PINNs show strong potential in flexibility and speed for specific problems, traditional methods remain more reliable for achieving high accuracy and efficiency in well-posed, structured settings.

PINNs have demonstrated considerable success in applications related to physical problems. Specifically, they have been effectively used to tackle problems in fluid dynamics, heat transfer, and wave propagation. The advantages of PINNs in these areas include their ability to achieve accurate solutions with relatively sparse data, their flexibility in handling complex geometries and boundary conditions, and their capacity for generalization across various conditions once the model has been trained.

While PINNs were originally designed for and predominantly applied to physics-related problems, their potential extends beyond traditional physics applications. Instead, they can be used for general dynamical systems. However, dynamical systems often involve complex interactions and behaviors that are not fully captured by physics alone. For instance, biological processes such as tumor growth [33] and gene expression [34,35] are governed by differential equations but are influenced by complex, variable factors that are not described by physical laws. Similarly, epidemiological models like the SIR (Susceptible, Infected, Recovered) model for disease spread are described by ODEs without a connection to physics. Still, they can benefit from PINNs to improve prediction accuracy and inform public health strategies [36–39]. Furthermore, adaptive systems, including economic markets and ecological systems, could leverage PINNs to understand dynamic behavior and enhance predictive capabilities.

The aim of this paper is to demonstrate the application of PINNs in the context of Ordinary Differential Equations (ODEs), extending their use beyond traditional physics-based problems. Specifically, we focus on applications in biology, systems biology, and epidemiology, where problems are formulated as systems of ODEs. Through numerical examples, we explore how PINNs can effectively address complex challenges in dynamical systems by approximating solutions to forward problems and estimating parameters in

inverse problems. This work addresses a significant gap in the literature by applying PINNs to broader systems outside of physics.

We concentrate on three models: tumor growth, gene expression, and the SIR model. For tumor growth, we investigate how PINNs can model cancer progression and predict growth patterns using differential equations that represent relevant biological processes. In the case of gene expression, PINNs are used to predict gene activity and interactions through equations capturing biological regulatory mechanisms. For disease spread, as modeled by the SIR (Susceptible, Infected, Recovered) framework, we explore how PINNs can improve our understanding of disease dynamics and enhance parameter estimation for better predictive accuracy. To make the results fully reproducible, the code of our analysis and datasets associated with this paper can be found on GitHub at <https://github.com/AmerFarea/ODE-PINN> (accessed on 10 May 2025).

Overall, this study illustrates the potential of PINNs to advance the understanding and modeling of complex dynamical systems, providing accurate solutions and insights that extend beyond traditional physics applications. Our exploration highlights the practical benefits of PINNs in diverse fields, contributing to the development of more adaptable and sophisticated modeling techniques.

This paper is organized as follows: We start by explaining the motivation for our study. Section 2 provides the theoretical background, covering PINNs, ODE-based dynamical systems, and forward and inverse problems. Section 3 details the experimental setup, including the implementation steps for solving forward and inverse problems to approximate solutions and estimate parameters. Section 4 presents our numerical results, followed by a discussion. Finally, the paper finishes with concluding remarks.

### *Motivation*

When reviewing the literature [4], we found that PINNs have primarily been applied to physical problems. However, PINNs could also be beneficial for biological systems, as many examples involve the use of differential equations [40–42]. This extension is motivated by the need to address increasingly complex real-world problems governed by differential equations but falling outside the realm of physics. By applying PINNs in these new contexts, we can achieve significant benefits and improvements in problem-solving.

First, many contemporary challenges—such as modeling tumor growth, understanding gene expression, and predicting disease spread—are inherently complex and dynamic. These problems often involve non-linear interactions and high-dimensional systems, which can be difficult to capture using conventional methods. While traditional numerical approaches are powerful, they often struggle with scalability and computational efficiency in such scenarios. PINNs offer a robust alternative by directly integrating dynamic laws into the neural network training process, enabling more accurate solutions with less dependence on extensive data and computational resources.

Second, PINNs can enhance the interpretability and reliability of models in fields such as systems biology, network biology and epidemiology. For instance, in cancer research, PINNs can model tumor growth dynamics with the precision needed to predict progression and assess treatment strategies. In genomics, PINNs can elucidate the underlying mechanisms of gene regulation and expression, providing insights that are crucial for understanding genetic disorders and developing targeted therapies. Similarly, in epidemiology, PINNs can refine the SIR model to better predict and manage disease outbreaks, improving public health responses.

From a technical standpoint, PINNs offer the potential for improved parameter estimation in inverse problems. Many dynamical systems involve parameters that are not directly observable, but are crucial for accurate modeling. By applying PINNs, we can estimate

these unknown parameters more effectively by incorporating both dynamic constraints and observational data. This capability is particularly valuable in complex systems where traditional methods may struggle to balance data fit with prediction accuracy.

The ability of PINNs to generalize across various conditions also presents a significant advantage. Once trained, PINNs can adapt to a range of scenarios and provide solutions for new or unseen conditions, making them highly versatile tools for dynamical systems modeling. This adaptability is crucial for addressing the ever-evolving nature of real-world problems, where conditions and parameters may change over time.

Overall, the extension of PINNs to dynamical systems beyond physics represents a significant advancement with the potential to revolutionize how we approach and solve complex real-world problems. By harnessing the power of PINNs, we can achieve more accurate, efficient, and interpretable models that are essential for advancing scientific understanding and addressing pressing challenges across various domains.

## 2. Theoretical Background

### 2.1. Physics-Informed Neural Networks (PINNs)

PINNs represent a significant advancement in the field of artificial intelligence and machine learning by integrating physical laws directly into the training process of neural networks [1–4]. Unlike traditional neural networks that learn solely from data, PINNs incorporate knowledge of underlying physical principles, such as differential equations, into their architecture. This integration allows PINNs to produce solutions that are consistent with known physical laws, improving both accuracy and generalization.

A typical PINN consists of a neural network with an architecture that includes input layers, hidden layers, and output layers. The key innovation of PINNs is the incorporation of physical constraints into the network's loss function. Specifically, during training, PINNs optimize not only the error between the network's predictions and the observed data but also the residuals of the governing differential equations. This approach ensures that the network's predictions adhere to the physical laws described by the differential equations.

The training process involves minimizing a combined loss function, which typically includes a data loss term and a physics loss term. The data loss term measures the discrepancy between the predicted output and the observed data, while the physics loss term measures how well the network satisfies the differential equations that govern the system. By optimizing this combined loss function, PINNs can achieve solutions that respect both empirical data and physical laws.

### 2.2. ODEs-Based Dynamical Systems

Ordinary Differential Equations (ODEs) play a central role in modeling dynamical systems, which describe how quantities change over time or space according to specified rules. ODEs are used to represent a wide range of phenomena, from mechanical systems [43,44] and electrical circuits [45] to biological processes [46] and financial markets [47]. These equations encapsulate the relationships between variables and their rates of change, making them crucial for understanding and predicting system behavior.

In dynamical systems, ODEs provide a mathematical framework for describing how systems evolve over time. For example, in biological systems, ODEs can model tumor growth by capturing how the tumor size changes in response to various factors, such as treatment or natural progression. Similarly, in epidemiology, the SIR (Susceptible, Infected, Recovered) model uses ODEs to describe the spread of infectious diseases within a population. Below is a brief overview of the dynamical system utilized in this study, along with the associated differential equations.

### 1. Tumor Growth Model

In the 1960s, A.K. Laird [33] successfully applied the Gompertz curve [48] to model tumor growth, recognizing that tumors are cellular populations expanding in a confined space with limited nutrients. The Gompertz curve for tumor size  $X(t)$  is expressed as:

$$X(t) = K \exp\left(\log\left(\frac{X(0)}{K}\right) \exp(-\alpha t)\right) \quad (1)$$

where  $X(0)$  is the initial tumor size,  $K$  is the carrying capacity, and  $\alpha$  represents the cell proliferation rate. The curve approaches  $K$  as time progresses, regardless of the initial size. The corresponding differential equation is:

$$\frac{dX}{dt} = \alpha \log\left(\frac{K}{X(t)}\right) X(t) \quad (2)$$

This model captures the decline in proliferation rate due to nutrient competition, similar to logistic growth [49] but with an unbounded rate for small tumors, which can be unrealistic. Recent insights suggest that the Gompertz model might not fit small tumors well and may not account for immune interactions effectively. Fornalski et al. [50] found that while the Gompertz curve fits typical cancer growth patterns, it is less accurate for very early stages, where other models might be more appropriate.

### 2. Gene Expression Model

The dynamics of gene expression can be modeled using differential equations that describe the behavior of mRNA and protein concentrations over time [34,35]. The rate of change of mRNA concentration is given by:

$$\frac{dM}{dt} = k_m \left( \frac{P_N}{M(t)} - \frac{\gamma_m}{k_m} \right) M(t) \quad (3)$$

where  $\frac{dM}{dt}$  represents the rate of change of mRNA concentration  $M(t)$  over time,  $k_m$  is the transcription rate constant indicating how quickly mRNA is produced from the gene, and  $P_N$  denotes the plasmid number or gene copy number affecting the total mRNA production. The term  $\gamma_m$  represents the mRNA degradation rate constant, reflecting the rate at which mRNA molecules degrade. This equation thus models the balance between mRNA production and degradation. Similarly, the dynamics of protein concentration are described by:

$$\frac{dP}{dt} = k_p \cdot \left( \frac{M(t)}{P(t)} - \frac{\gamma_p}{k_p} \right) \cdot P(t) \quad (4)$$

where  $\frac{dP}{dt}$  denotes the rate of change of protein concentration over time,  $k_p$  is the translation rate constant representing how efficiently mRNA is translated into protein, and  $M(t)$  is the concentration of mRNA available for translation. The term  $\gamma_p$  is the protein degradation rate constant, indicating the rate at which proteins degrade. This equation captures how protein levels fluctuate due to the combined effects of translation and degradation.

Such a model is applicable in various domains, including synthetic biology for designing and optimizing genetic circuits, cell biology for understanding gene expression patterns, pharmacology for studying the impact of drugs on gene expression, and genetic engineering for controlling and analyzing the effects of gene modifications on mRNA and protein levels.

### 3. SIR Model (Susceptible, Infected, Recovered)

The SIR model is a foundational epidemiological model that divides the population into three distinct compartments: Susceptible (S), Infected (I), and Recovered (R) [36–39]. Susceptible individuals can become infected through contact with infected individuals, and infected individuals recover over time. Recovered individuals are assumed to gain complete immunity and cannot be reinfected. The differential equations governing this model are:

$$\frac{dS}{dt} = -\beta SI \quad (5)$$

$$\frac{dI}{dt} = \beta SI - \gamma I \quad (6)$$

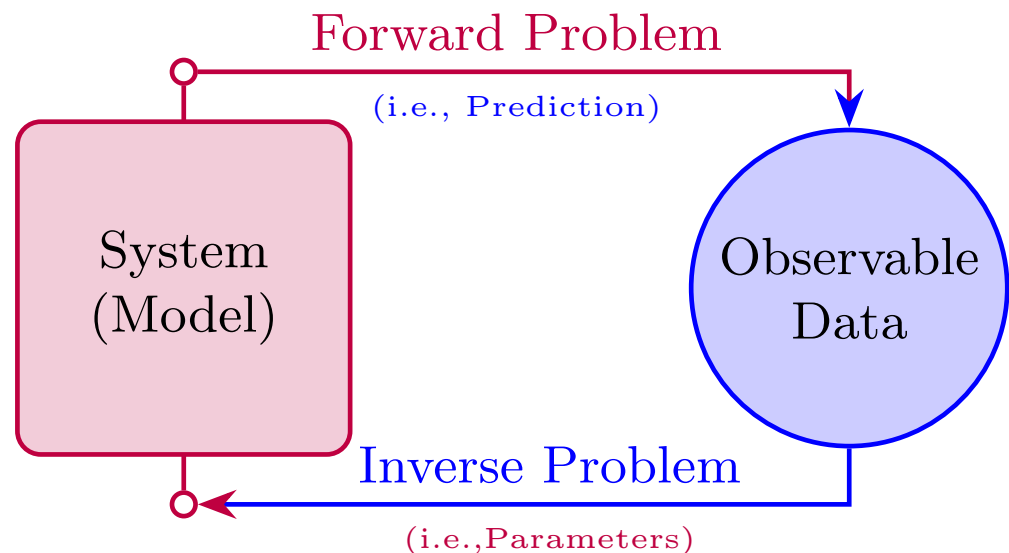
$$\frac{dR}{dt} = \gamma I \quad (7)$$

where the derivatives  $\frac{dS}{dt}$ ,  $\frac{dI}{dt}$ , and  $\frac{dR}{dt}$  describe the rates of change in the numbers of susceptible, infected, and recovered individuals, respectively,  $S(t)$  represents the number of susceptible individuals,  $I(t)$  represents the number of infected individuals,  $R(t)$  represents the number of recovered individuals,  $\beta$  is the infection rate,  $\gamma$  is the recovery rate, and  $N$  is the total population. The constraints for this model are  $S(t), I(t), R(t) \geq 0$  and  $S(t) + I(t) + R(t) = N$ .

The relevance of ODEs to the problems addressed by PINNs lies in their ability to represent the underlying dynamics of the system. PINNs leverage this structure to guide the learning process, enabling the neural network to approximate solutions that adhere to known system behavior.

#### 2.3. Forward and Inverse Problems

PINNs can be employed in a variety of ways [4,51]. Our primary focus here is on using PINNs to tackle both forward and inverse problems, as depicted in Figure 1.

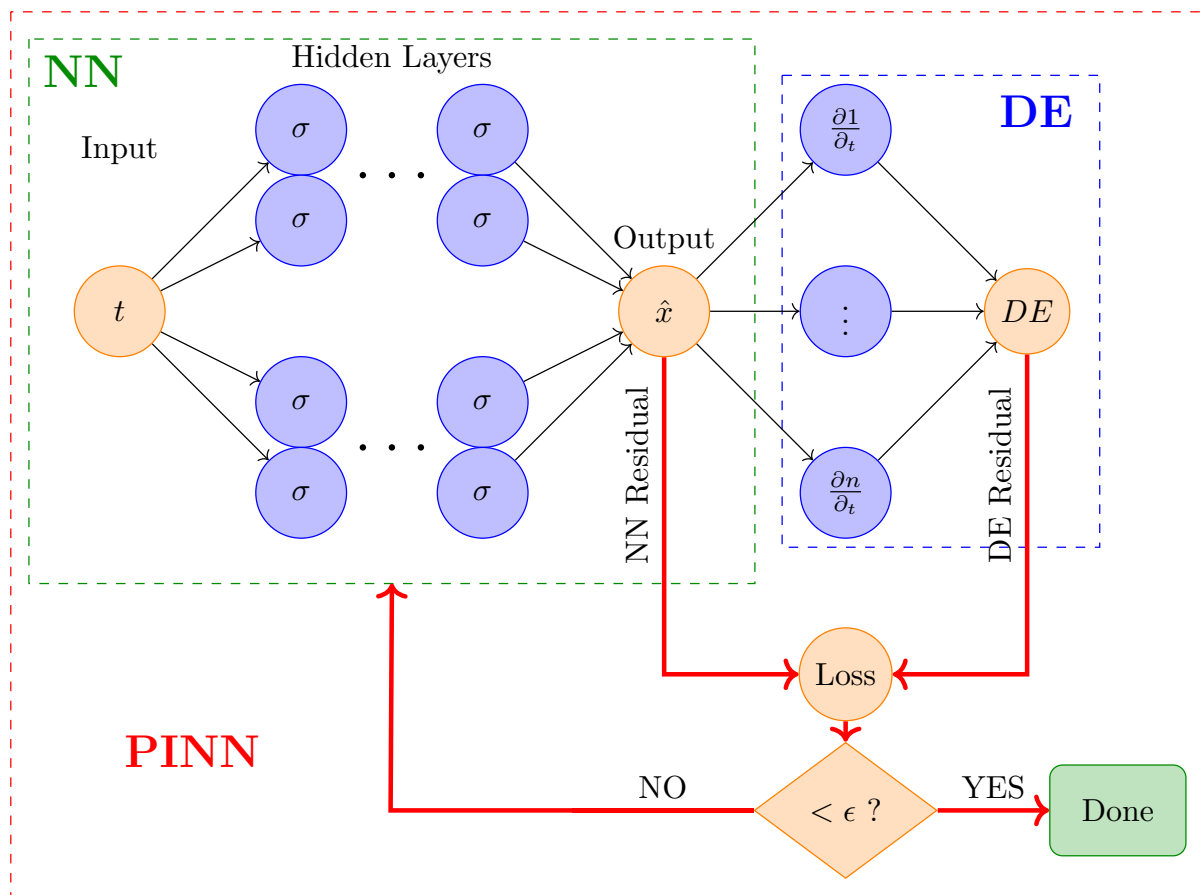


**Figure 1.** This figure illustrates how PINNs address forward problems by approximating ODE/PDE solutions by integrating physical laws into the loss function. It also shows how PINNs estimate unknown parameters in ODEs/PDEs by minimizing the difference between predicted and observed data.

The forward problem entails approximating solutions to ordinary or partial differential equations (DEs) by incorporating the governing physical laws into the loss function, as



shown in Figure 2. It involves predicting the outcome or state of a system given known inputs and parameters. In the context of ODEs, a forward problem requires solving the differential equation to obtain the system's response over time. For instance, in a tumor growth model, the forward problem would involve using ODEs to predict how the tumor size evolves based on initial conditions and parameters. PINNs tackle forward problems by learning to approximate the solution to these differential equations, providing insights into the system's behavior under various conditions.



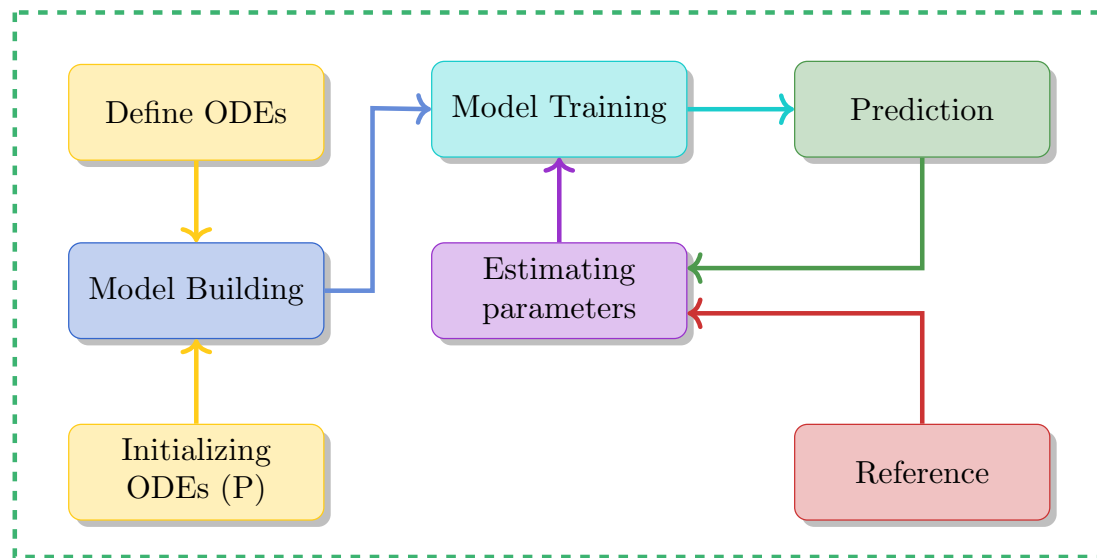
**Figure 2.** In the forward problem, PINN incorporates both the residuals from the Differential Equations (DE) and the data loss into the loss function to improve the accuracy of the Neural Network (NN). Here,  $\epsilon$  represents the permissible margin of error.

On the other hand, PINNs can also be used to solve inverse problems and estimate unknown parameters, see Figure 3. In inverse problems, the goal is to determine unknown parameters or model coefficients from observed data. This is typically achieved by incorporating physical constraints into the loss function, which quantifies the discrepancy between the model's predictions and the observed data, while ensuring that the solutions adhere to the governing equations. This integration ensures that the estimated parameters are not only consistent with the observed data but also adhere to the underlying physical laws.

For example, in the context of a tumor growth model, an inverse problem involves estimating parameters such as the carrying capacity ( $K$ ) and the cell proliferation rate ( $\alpha$ ) from observed data (e.g., tumor size over time). During training, the PINN adjusts these parameters to minimize the difference between the model's predicted tumor growth and the actual observed data, allowing for accurate estimation of the underlying biological dynamics.

To gain a practical understanding of PINNs, the implementations of utilized ODE-based models are outlined next sections. This hands-on approach will allow us to replicate

these examples and build a foundation for applying similar techniques to related problems, which will be outlined.



**Figure 3.** In the inverse problem, PINNs estimate the unknown parameters (P) of ODEs/PDEs by first providing an initial guess or starting values. They then minimize the difference between the predicted results and observed data throughout the training process to determine the true values.

### 3. Experimental Procedures

Utilizing PINN to tackle both forward problems and inverse problems in ODE-based dynamical systems involves several crucial steps, which can be summarized as follows:

#### 3.1. Forward Problems

Implementing a forward solution for a dynamical system using PINNs entails several steps. These steps, detailed in Algorithm 1, can be summarized as follows:

#### Algorithm 1 Approximating Solution for a dynamical system Using PINNs

- 1: **Step 1: Defining the dynamical system**
- 2: INPUT: System parameters and initial condition
- 3: Define the differential equation:  $\frac{dx}{dt} = f(x, t)$
- 4: **Step 2: Defining PINN Model (Network Architecture)**
- 5: Input:  $t$  (time or independent variable)
- 6: Hidden Layers: Specify the number of hidden layers and neurons per layer.
- 7: Activation Function: Choose suitable activation functions (e.g., GELU, Tanh)
- 8: Output:  $\hat{x}(t)$  (state variable or dependent variable)
- 9: **Step 3: Defining Loss Functions**
- 10: **Data Loss:**  $\mathcal{L}_{\text{Data}} = \sum_i |\hat{x}(t_i) - x_i|^2$
- 11: **Physics Loss:**  $\mathcal{L}_{\text{Physics}} = \left| \frac{dx}{dt} - f(x, t) \right|^2$
- 12: **Total Loss:**  $\mathcal{L}_{\text{total}} = \lambda_1 \cdot \mathcal{L}_{\text{Data}} + \lambda_2 \cdot \mathcal{L}_{\text{Physics}}$
- 13: **Step 4: Training PINN Model**
- 14: Initialize Network Parameters
- 15: Use optimizer (e.g., Adam or SGD) to minimize  $\mathcal{L}_{\text{total}}$ :  $\theta^* = \arg \min_{\theta} \mathcal{L}_{\text{total}}(\theta)$
- 16: Backpropagate the loss and update the model parameters.
- 17: Repeat for sufficient epochs until convergence
- 18: **Step 5: Evaluating the Model**
- 19: Compare predicted  $\hat{x}(t)$  with known values, observed data, or analytical solutions if available
- 20:  $x(t) \approx \hat{x}(t; \theta^*)$
- 21: Plot the model predictions against the actual data or reference.



1. In the first step, the differential equation that governs the dynamical system needs to be defined as detailed in Section 2.2.
2. In step 2, the neural network needs to be defined, which requires specifying its architecture, including activation functions and the input range. For example, the input could be time  $t$  in the range  $[0, T]$ , and the output could be the approximate solution for the state variable  $x(t)$ , with appropriate activation functions (e.g., GELU or Tanh).
3. In step 3, the loss functions are defined to guide the training process. Importantly, the total loss function consists of different parts corresponding to individual loss functions, each describing a particular aspect.

This includes:

- **Data Loss:** Measures the differences between predicted values and observed data or exact solution, expressed as

$$\mathcal{L}_{\text{Data}} = \sum_i |\hat{x}(t_i) - x_i|^2 \quad (8)$$

- **Physics Loss:** Ensures that the neural network adheres to the underlying physical law, typically formulated as a differential equation. This is achieved by computing the derivative of the network's output with respect to its input and comparing it to the expected theoretical expression. The loss is defined as:

$$\mathcal{L}_{\text{Physics}} = \left| \frac{dx}{dt} - f(x, t) \right|^2 \quad (9)$$

The derivative  $\frac{d\hat{x}}{dt}$  is computed using PyTorch's automatic differentiation framework. This approach allows for efficient and accurate gradient computation, enabling the enforcement of physical consistency within the training process. Finally, the total loss combines the individual loss functions:

$$\mathcal{L}_{\text{total}} = \lambda_1 \cdot \mathcal{L}_{\text{Data}} + \lambda_2 \cdot \mathcal{L}_{\text{Physics}} \quad (10)$$

where  $\lambda_1$  and  $\lambda_2$  control the trade-off between fitting the data and satisfying the physical constraints.

4. In step 4, the neural network is trained. This involves initializing the network parameters, and using an optimizer like Adam or SGD to minimize the total loss over sufficient epochs until the model converges.
5. In step 5, the model is evaluated. For this, one can assess the performance of the model by comparing the predicted outputs  $\hat{x}(t)$  with actual values  $x(t)$  or analytical solutions, when available.

### 3.2. Inverse Problems

Using PINN to estimate unknown parameters from observable data, a process also known as the inverse problem, for dynamical systems described by ODEs involves several key steps, as summarized in Algorithm 2. These steps can be outlined as follows:

1. **Load Data:** The first step involves reading timepoints  $t$  and corresponding observations  $x$  from the available sources (in our case, a CSV file containing simulated data based on the models' ODE equations) and converting these data into numpy arrays.
2. **Initial Parameter Estimation:** In this step, the ordinary differential equation (ODE) is defined, with  $\theta$  representing the parameters to be estimated and initialized as  $\theta_{\text{init}}$ .

The loss function is formulated to minimize the error between the observed data and the values predicted by the model.

In Step 3, the PINN model is defined by constructing a neural network to approximate the solution of the dynamical system governed by the ODE. The architecture of the network includes an input layer representing  $t$  (the time or independent variable), followed by hidden layers with a specified number of layers and neurons per layer (for example, two layers with 100 units each). The activation function is selected from appropriate options such as GELU or Tanh. Finally, the output layer of the network provides  $\hat{x}(t)$ , the approximated state or dependent variable of the system.

In Step 4, the model parameters, including the network parameters  $\theta$  (e.g., weights, biases) and dynamical system parameters, are optimized through training. An optimizer like Adam or SGD is used to minimize the total loss  $\mathcal{L}_{\text{total}}$ , adjusting both sets of parameters to reduce the error between predictions and actual values. Gradients are computed via backpropagation, and the parameters are updated accordingly. Training continues until the loss converges, signaling that the model has reached an optimal state.

In Step 5: Evaluating the Model, the goal is to assess how well the model's predictions match the actual data or known analytical solutions. This is achieved by comparing the predicted values  $\hat{x}(t; \theta^*)$ , which are the model's outputs based on the optimized parameters  $\theta^*$ , to the observed values or reference data. This comparison is often visualized by plotting the predicted values against the real data, allowing for a clear inspection of how well the model fits. Additionally, the estimated parameters  $\theta$  of the dynamical system are printed, providing the final values of the system's parameters after optimization. This evaluation is important for understanding the model's accuracy and its ability to represent the system's dynamics.

---

**Algorithm 2** Estimating Parameters of a dynamical system Using PINNs

---

- 1: **Step 1: Load Data**
  - 2:     Read timepoints  $t$  and observations  $x$  from the available source.
  - 3:     Convert the data to numpy arrays.
  - 4: **Step 2: Initial Parameter Estimation**
  - 5:     Define the ODE dynamical system along with its initial conditions and initial parameter values  $\theta_{\text{init}}$ .
  - 6:     Define the loss functions:  $\mathcal{L}_{\text{total}} = \lambda_1 \cdot \mathcal{L}_{\text{Data}} + \lambda_2 \cdot \mathcal{L}_{\text{Physics}}$ .
  - 7: **Step 3: Defining PINN Model (Network Architecture)**
  - 8:     Input:  $t$  (time or independent variable)
  - 9:     Hidden Layers: Specify the number of hidden layers and neurons per layer.
  - 10:    Activation Function: Choose suitable activation functions (e.g., GELU, Tanh)
  - 11:    Output:  $\hat{x}(t)$  (state variable or dependent variable)
  - 12:    Specify the parameters  $\theta$  to be estimated as trainable parameters.
  - 13: **Step 4: Training PINN Model**
  - 14:    Initialize Network Parameters
  - 15:    Use optimizer (e.g., Adam or SGD) to minimize  $\mathcal{L}_{\text{total}}$ :  $\theta^* = \arg \min_{\theta} \mathcal{L}_{\text{total}}(\theta)$
  - 16:    Backpropagate the loss and update the NN and dynamical system parameters.
  - 17:    Repeat for sufficient epochs until convergence
  - 18: **Step 5: Evaluating the Model**
  - 19:    Compare predicted  $\hat{x}(t)$  with known values, observed data, or analytical solutions if available
  - 20:     $x(t) \approx \hat{x}(t; \theta^*)$
  - 21:    Plot the model predictions against the actual data or reference.
  - 22:    Print the estimated parameters  $\theta$  of the dynamical system.
-

## 4. Results

In order to make our study reproducible, we start our analysis by providing information about parameter settings. Selecting the appropriate hyperparameters is crucial to optimizing a neural network's performance. The number of hidden layers affects the model's depth; while additional layers enable the model to learn more complex patterns, they also increase the risk of overfitting if the network is excessively deep. Based on experiments with 2, 3, and 4 layers, the optimal number was selected to balance complexity and overfitting, as outlined in Table 1. Hidden dimensions denote the number of neurons in each layer. Larger dimensions allow the model to capture more features but can also lead to overfitting and higher computational costs as well. After trying different values, the chosen dimension aimed to strike a balance between model capacity and efficiency. Activation functions introduce non-linearity into the model. Our experiments with PINNs showed that functions such as *ReLU*, *Tanh*, and *GELU* outperformed *Sigmoid*, likely due to their effectiveness in addressing vanishing gradient issues. These activation functions contributed to better convergence and stability. The Learning Rate governs the step size during optimization. After testing values of  $1 \times 10^{-2}$ ,  $1 \times 10^{-3}$ , and  $\times 10^{-4}$ , it was observed that a rate of  $\times 10^{-2}$  might have caused rapid convergence or oscillation, while  $\times 10^{-4}$  led to slower convergence. The selected value ( $\times 10^{-3}$ ) balanced fast learning with stable convergence. Epochs determine how frequently the model reviews the entire dataset, and Patience helps to stop training early if performance stagnates. The selected values balance comprehensive learning with the prevention of overfitting or excessive training.

In our implementation, the total loss function is defined as in Equation (10), where the weighting parameters  $\lambda_1$  and  $\lambda_2$  control the trade-off between fitting observed data and enforcing physical laws. These values were determined through empirical tuning:  $\lambda_1$  was selected to prioritize data accuracy without overfitting, while  $\lambda_2$  was gradually adjusted to ensure the model accurately captured physical behavior without dominating the loss function. Both parameters were constrained to the range  $[0, 1]$ . The best performance was achieved with  $\lambda_1 = 0.7$  and  $\lambda_2 = 0.3$ . Deviations from this configuration, whether by setting the weights too low or too high, led to an imbalance, resulting in either weak enforcement of physical principles (particularly affecting the estimated parameter values) or diminished data accuracy (i.e., suboptimal data fitting), respectively.

Table 1 gives a summary of three models studied in the following: Tumor Growth (TG), Gene Expression (GE), and an Epidemiological system (ES), the SIR (Susceptible, Infected, Recovered) model. Each model receives a single input, as indicated by the input dimension (INPUT\_DIM) set to 1 across all models. The model architectures differ slightly, with the GE having two hidden layers, while the TG and SIR models are deeper, with three hidden layers. The number of neurons per hidden layer, known as HIDDEN\_DIMS, is 100 for both the TG and SIR models, while the GE model uses 64 neurons, reflecting the computational complexity and capacity tailored to each model's needs.

The output dimensions (OUTPUT\_DIM) differ as well, with the TG model producing a single output, the GE model producing two, and the SIR model generating three outputs, corresponding to the three states in the SIR model. The activation functions used also vary; TG and GE models utilize the GELU activation function, known for its smooth, non-linear properties, while the SIR model employs the Tanh function, suited for modeling smooth transitions between states.

The learning rate, a critical hyperparameter controlling the step size during model optimization, is consistently set to  $\times 10^{-3}$  for all models. Initial conditions, or state variables, are specified uniquely for each model, such as the initial population states for the SIR model (S, I, R) and specific starting levels for the TG and GE models. Each model

also has unique parameters tailored to its dynamics, such as growth rates and carrying capacities, which are crucial for simulating realistic scenarios.

**Table 1.** Parameters settings for the models Tumor Growth (TG), Gene Expression (GE), and the SIR (Susceptible, Infected, Recovered).

	TG	GE	SIR
INPUT_DIM	1	1	1
HIDDEN_LAYERS	3	2	3
HIDDEN_DIMS	64	64	100
OUTPUT_DIM	1	2	3
ACTIVATION_FN	<i>GELU</i>	<i>GELU</i>	<i>Tanh</i>
LEARNING_RATE	$\times 10^{-3}$	$\times 10^{-3}$	$\times 10^{-3}$
STATE_VARIABLES	$X(0) = 1$	$M(0) = 1, P(0) = 1$	$S(0) = 0.99, I(0) = 0.01, R(0) = 0.0$
PARAMS	$\alpha = 0.3, K = 100$	$PN = 50, K_m = 0.7, \gamma_m = 0.8,$ $k_p = 0.6, \gamma_p = 0.6$	$\beta = 0.5, \gamma = 0.05$
SIMULATION_TIME	50	50	100
NUM_SAMPLES	120	100	120
BATCH_SIZE	32	32	32
EPOCHS	1000	3000	4000
EPOCH_INTERVAL	200	250	50
PATIENCE	100	200	50

The simulated data is produced using the models' ODE equations. The time range for simulation differs slightly, with the TG and GE models running for 50 time units, while the SIR model extends to 100. The number of samples used in training is 120 for the TG and SIR models and 100 for the GE model. The generated data (NUM\_SAMPLES) is divided into training and testing sets, with 80% allocated for training and 20% reserved for testing. Here, "training sample" and "test samples" correspond to the true/exact solution. All models are trained with a batch size of 32, meaning that 32 samples are processed before updating the model's parameters.

Training epochs, the number of complete passes through the training data, is set at 1000 for the TG model, 3000 for GE model, and 4000 for the SIR model, likely reflecting the greater complexity or longer convergence time needed for the SIR model. Evaluations are made across all models at every EPOCH\_INTERVAL epoch, as specified by the EPOCH\_INTERVAL. The patience parameter, set to 100 for TG, 200 for GE, and 50 for SIR model, specifies the number of epochs allowed without improvement before early stopping is triggered, preventing overfitting and unnecessary computation.

The same settings were used for both predicting the forward solution and for parameter estimation, with one exception: the number of epochs during parameter estimation differed. It was set to 2000 for the TG model, 15,000 for the GE model, and 10,000 for the SIR model. This variation is due to slight differences in the models and training functions used for each problem (forward vs. inverse). For example, in the inverse problem, the model requires initial values for the parameter being estimated, whereas this is not needed in the forward problem.

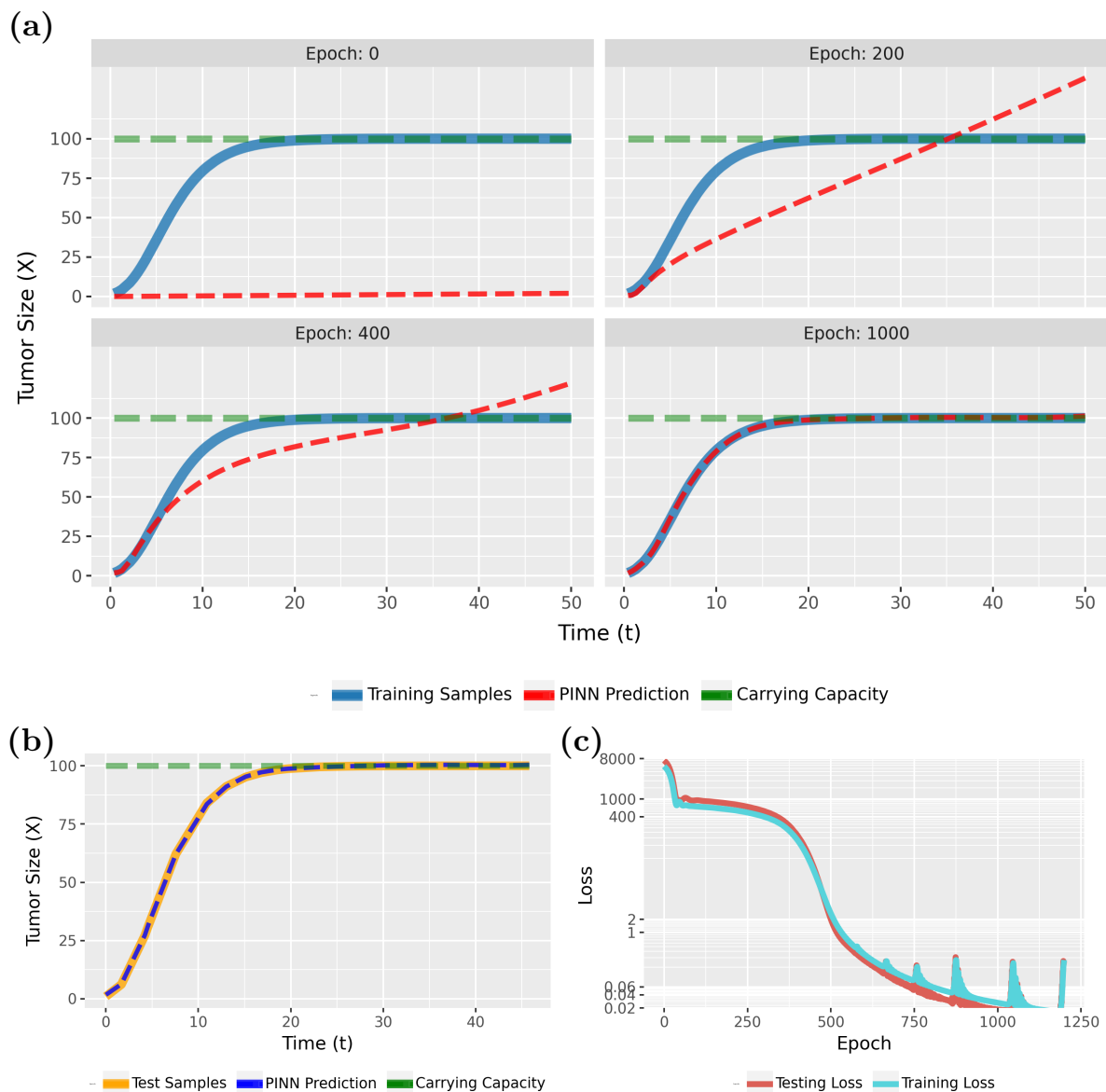
In the parameter estimation process, values for *STATE VARIABLES* and *SIMULATION TIME* were not provided as predefined inputs but had to be extracted from the generated data. The initial parameter values were as follows: for TG,  $\alpha$  and  $K$  were set to [1.0, 10.0]; for GE,  $k_m$ ,  $k_p$ ,  $\gamma_p$ , and  $\gamma_m$  were all set to 1; and for SIR,  $\alpha$  and  $\gamma$  were both set to 1.

Overall, these configurations demonstrate how the models are precisely calibrated to fulfill their respective tasks, balancing complexity, computational efficiency, and the necessity for accurate simulation and prediction.

#### 4.1. Tumor Growth Model

The first model, we analyze is the tumor growth model (TGM) (see Methods section for details). For the TGM, we use a Physics-Informed Neural Network (PINN) to model cancer progression dynamics using the differential Equation (2), which describes the rate at which tumor size changes over time.

Figure 4 compares the PINN predictions (in red) with the actual tumor growth (in blue) modeled by Equation (2). Specifically, in Figure 4a (epoch 0), the initial results are shown, whereas the following three subfigures show the effects of the training for 200, 400, and 1000 epochs. From this one can see that the model rapidly converges to a true model. This is also confirmed by the loss function, shown in Figure 4c.



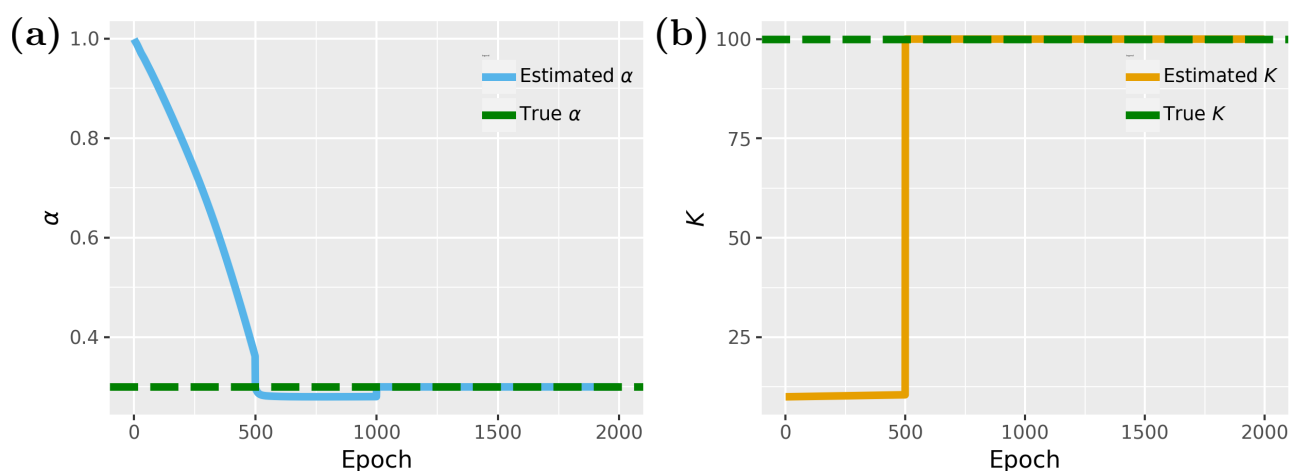
**Figure 4.** Forward solution of the PINN for the tumor growth model: (a) Predictions of the PINN for training data in dependence on different epochs. (b) Predictions of the PINN after training for testing data. (c) Evolution of the training and test loss function.

Notably, the PINN continues to improve for up to 500 epochs before stabilizing. As the loss values approach near-zero, we attempted to highlight this variation by adjusting the Y-axis scale. However, utilizing the early stopping technique, with the patience parameter

specified in Table 1, training will halt if no further improvement in loss accuracy is detected. In this case, early stopping occurred at epoch 1033.

To see that the PINN can generalize to new sample points, we show in Figure 4b results for test data for the converged model. Overall, these results are very similar to Figure 4a (epoch 1000) and indicate excellent generalization capabilities. Overall, our findings show that PINNs could accurately approximate tumor growth curves even from sparse data (see Table 1; a total of 120 samples were used, with 80% allocated for training and 20% for testing), with the predicted patterns closely aligning with the true model.

In Figure 5, we show results for the inverse problem, whereas Figure 5a shows results for the proliferation rate  $\alpha$  and Figure 5b for the carrying capacity  $K$ . For the carrying capacity  $K$  one can see that convergence is achieved for about 500 epochs, while for the proliferation rate  $\alpha$ , 1000 epochs are needed. Interestingly, between 500 and 1000 epochs the model is stuck in local minima close to the optimal value of  $\alpha$  but eventually finds the optimal solution.



**Figure 5.** Inverse problem of the PINN for the tumor growth model which depends on two parameters: (a) Parameter estimation for  $\alpha$ . (b) Parameter estimation for  $K$ .

#### 4.2. Fine Tuning of Hyperparameters

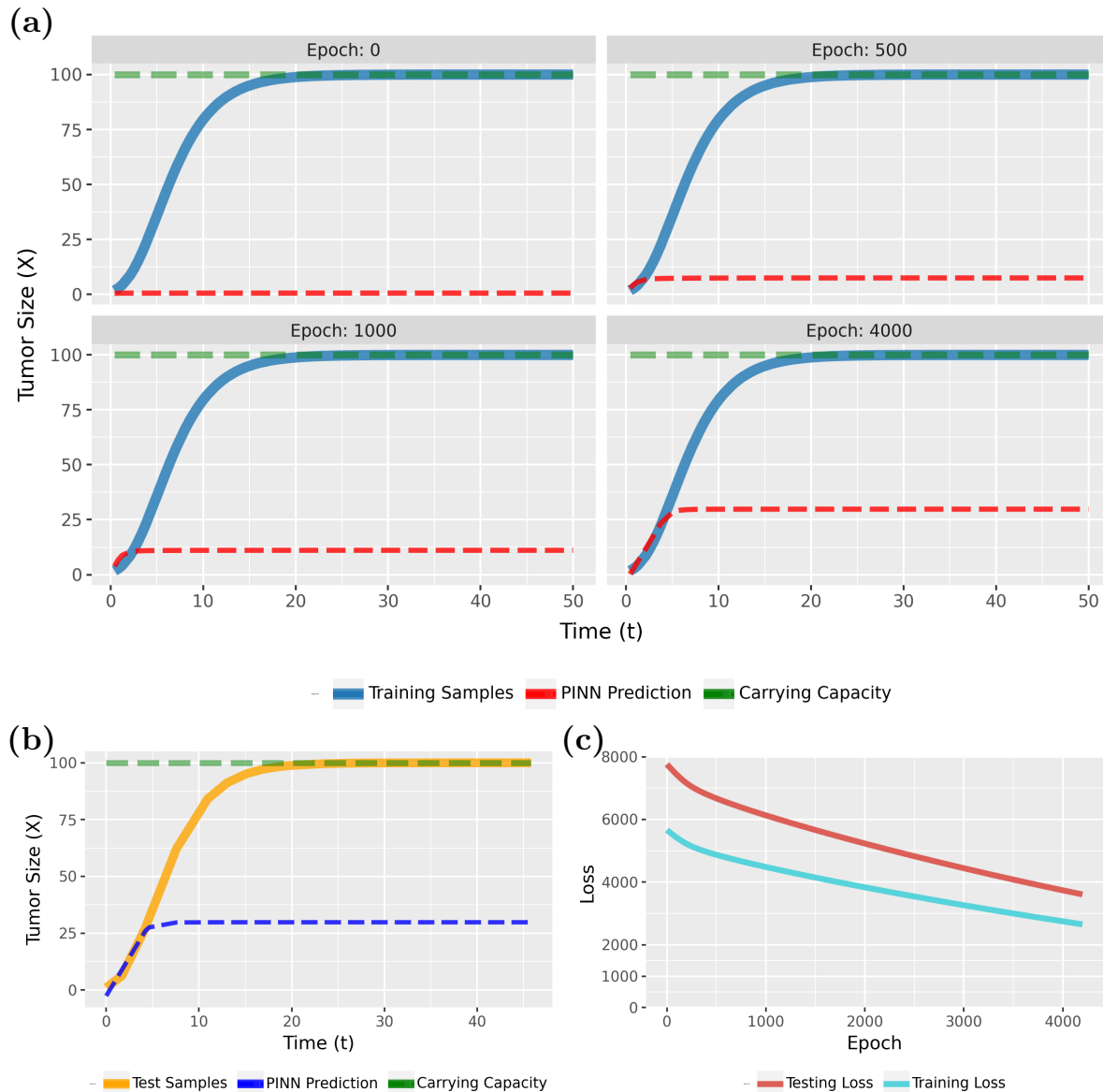
To illustrate how the effectiveness of PINNs relies on the careful fine tuning of the neural network architecture and hyperparameters, as well as the importance of customized configurations tailored to specific system dynamics, we discuss some details of the implementation steps outlined in Section 3 and the fine-tuning considerations discussed in Section 4. For this, we train the PINN model for the TGM as outlined in Table 1, with some modifications to emphasize the impact of hyperparameter tuning on the model's learnability and performance. Specifically, we reduced the number of hidden layers from three to two, with 32 neurons per layer (instead of 64). Additionally, the learning rate was set to  $(1 \times 10^{-4})$ , and the *Sigmoid* activation function was used.

As shown in Figure 6a,b, for the training and testing sets, the model performs poorly, failing to approximate the solution even after 4000 epochs. The loss accuracy, shown in Figure 6c, further highlights the model's poor performance under this configuration. This underscores the critical importance of hyperparameter selection for achieving efficient performance with fewer computational steps. Conversely, using the same configuration from Table 1, the model was able to approximate the solution perfectly within just 1000 epochs, as shown in Figure 4. Overall, this demonstrates the sensitivity of the performance of PINNs to moderate changes in the network design and training parameters.



#### 4.3. Gene Expression Model

The next model for which we illustrate the application of PINNs is a gene expression model (see Methods section for details). This model simulates the regulation of gene activity using a system of ordinary differential equations, as shown in Equations (3) and (4). These equations incorporate various biological interactions and regulatory mechanisms.

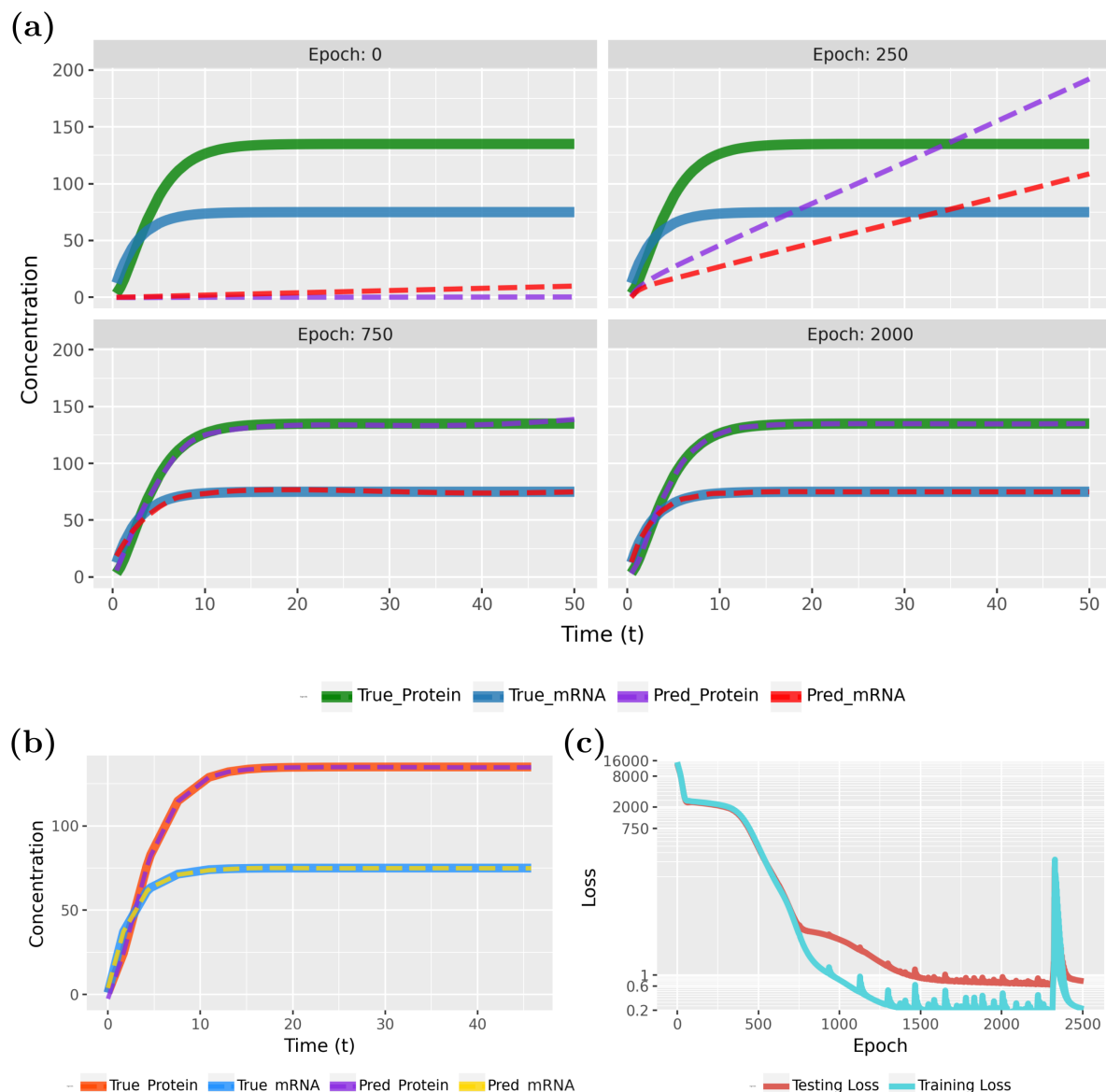


**Figure 6.** Forward solution of the PINN for the tumor growth model with poor configuration: (a) Predictions of the PINN for training data in dependence on different epochs. (b) Predictions of the PINN after training for testing data. (c) Evolution of the training and test loss function.

Figure 7 compares the PINN predictions with the actual synthesis and degradation of mRNAs and proteins. Specifically, in Figure 7a (epoch 0), results for the initialization are shown, whereas the following three subfigures show the effects of the training for 250, 750, and 2000 epochs. From the behavior of the curves, one can see that the model rapidly converges to true model.

The convergence of the model is also confirmed by the loss function, shown in Figure 7c. Notably, the PINN continues to improve for up to 800 epochs before stabilizing. As the loss values approach near-zero, we attempted to highlight this variation by adjusting the Y-axis scale. However, utilizing the early stopping technique, with the

parameters specified in Table 1, training will halt if no further improvement in loss accuracy is detected. In this case, early stopping occurred at epoch 2506.

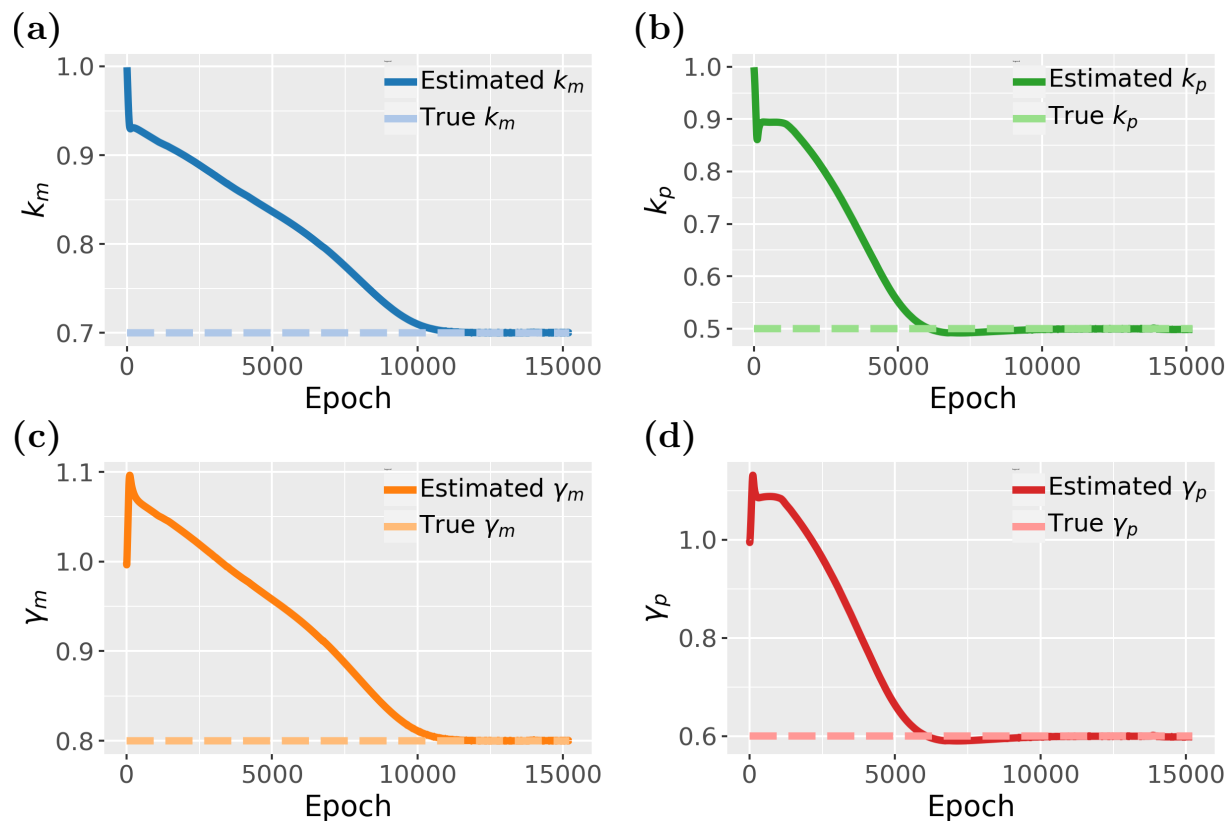


**Figure 7.** Forward solution of the PINN for the gene expression model: (a) Predictions of the PINN for training data in dependence on different epochs. (b) Predictions of the PINN after training for testing data. (c) Evolution of the training and test loss function.

To demonstrate the PINN's ability to generalize to new sample points, we present the test data results for the converged model in Figure 7b. These results highlight the model's generalization capabilities. Our findings indicate that PINNs can accurately approximate gene expression dynamics, including mRNA and protein synthesis and degradation within a cell, even with sparse data. Specifically, we used a total of 100 samples, with 80% allocated for training and 20% for testing (see Table 1), and observed that the predicted patterns closely align with the true model. Figure 8 illustrates the estimated parameters associated with gene regulation, such as  $k_m$ ,  $k_p$ ,  $\gamma_m$ , and  $\gamma_p$ , and their influence on expression levels.

In Figure 8, we study the inverse problem of the gene expression model. In total, the model consists of four parameters  $k_m$ ,  $k_p$ ,  $\gamma_m$ ,  $\gamma_p$  that need to be estimated. From Figure 8, it seems that  $k_p$  and  $\gamma_p$  converges much faster than  $k_m$  and  $\gamma_m$ , however, when looking carefully one notices a slight dip between 5000 and 10,000 epochs. Hence, all four

parameters converge around 10,000 epochs but  $k_p$  and  $\gamma_p$  are, for a time, stuck in local maxima close to the true value before reaching the optimal values. Overall, the results demonstrate that the PINN effectively captures the dynamics of gene expression and also allows the estimation of its parameters.



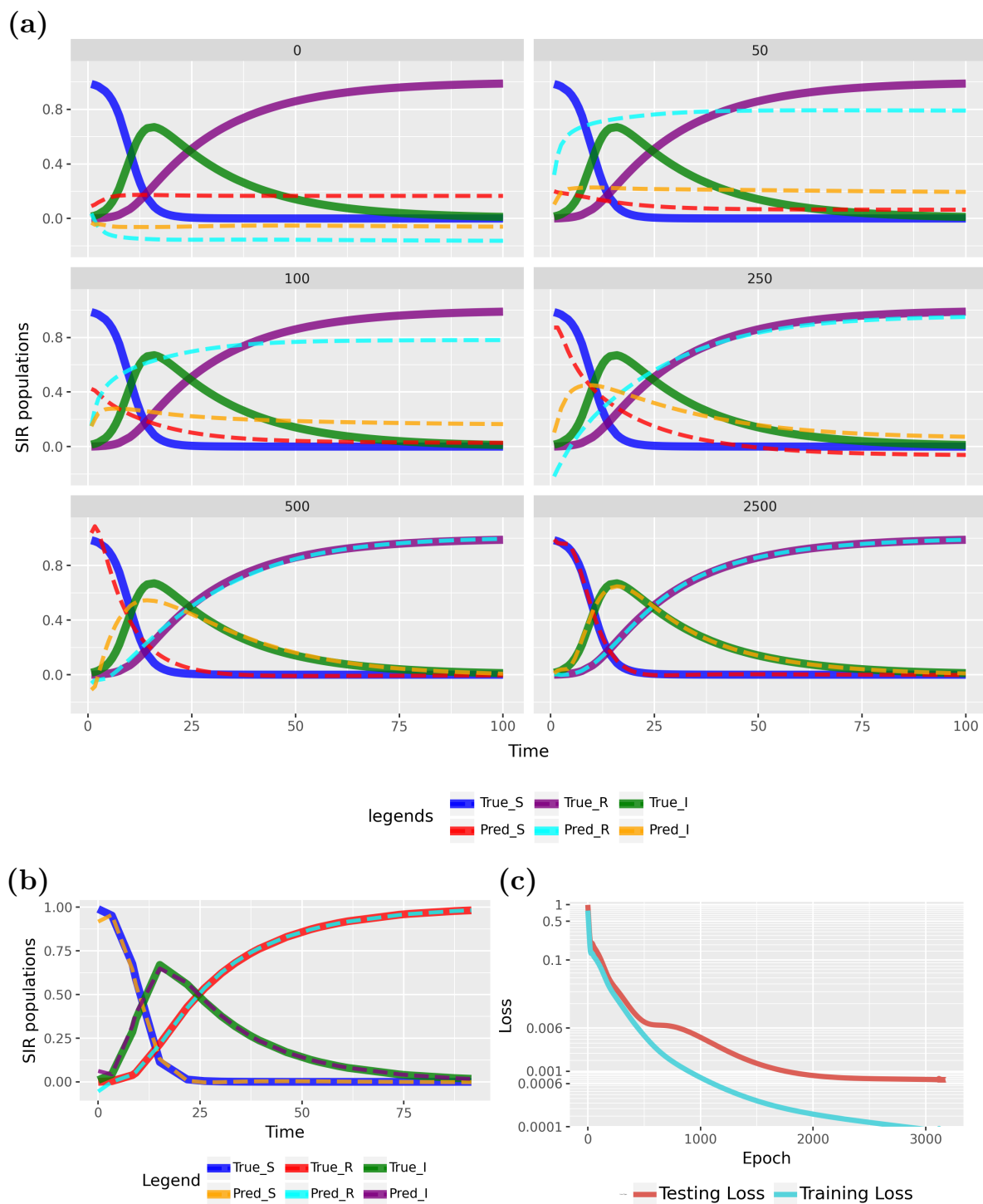
**Figure 8.** Inverse problem of the PINN for the gene expression model: (a) Parameter estimation for  $k_m$ . (b) Parameter estimation for  $k_p$ . (c) Parameter estimation for  $\gamma_m$ . (d) Parameter estimation for  $\gamma_p$ .

#### 4.4. SIR Model

The last model we study is the SIR (Susceptible, Infected, Recovered) model from epidemiology (see Methods section for details). The SIR model describes the spread of an infectious disease using a system of ordinary differential equations (ODEs), shown in Equations (5)–(7). Figure 9 presents a comparison between PINN predictions and the true model.

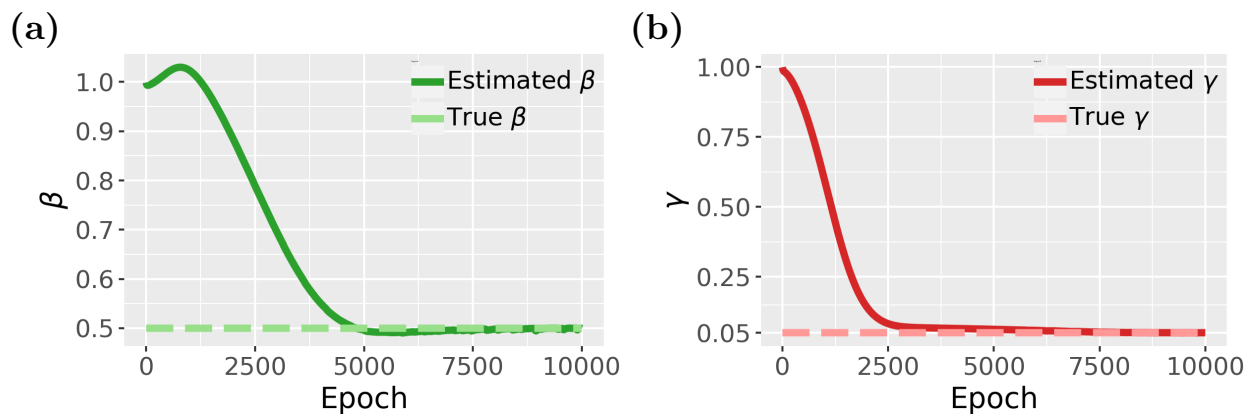
Specifically, Figure 9a shows results from epoch 0 (initialization), while the subsequent five subfigures illustrate the effects of training at epochs 50, 100, 250, 500, and 2500. The model converges rapidly to the true data, as indicated by the loss function in Figure 9c. The PINN continues to improve up to approximately 2000 epochs before stabilizing. To better highlight variations at near-zero loss values, the Y-axis was scaled. Early stopping, as detailed in Table 1, was employed to halt training when no further loss improvement was observed, which occurred at epoch 3175.

To show the generalization ability of the PINN, we show test data results for the converged model in Figure 9b. These results underscore the model's strong generalization capabilities. This demonstrates that a PINN can effectively estimate transmission and recovery rates, with forecasts closely aligning with the true model. Specifically, we used 120 samples, with 80% for training and 20% for testing (see Table 1), and the predicted patterns closely matched the true model.



**Figure 9.** Forward solution of the PINN for the SIR model: (a) Predictions of the PINN for training data in dependence on different epochs. (b) Predictions of the PINN for testing data. (c) Evolution of the training and test loss function.

Finally, in Figure 10, we show results for the inverse problem for the SIR model, where we estimate the two parameters,  $\beta$  and  $\gamma$ . As one can see from Figure 10a,b, both parameters converge to the true parameter values, however, again, only after escaping from local minima (see epochs 5000 to 7500 for  $\beta$  and epochs 2500 to 7000 for  $\gamma$ ).



**Figure 10.** Inverse problem of the PINN for the SIR model: (a) Parameter estimation for  $\beta$ . (b) Parameter estimation for  $\gamma$ .

## 5. Code Availability

To allow the reproduction of our results focusing on systems of ordinary differential equations, we created a Python package called ODE-PINN. The libraries used in our code provide essential functionalities for studying forward and inverse problems for a large variety of systems of ordinary differential equations.

We employ `scipy.integrate.odeint` to solve ordinary differential equations, while `numpy` and `pandas` are used for numerical computations and data handling. To split data into training and testing sets, we use `train_test_split` from `sklearn.model_selection`. Our deep learning models are built with PyTorch, utilizing `torch` as the framework for neural networks. For efficient data loading and batching, we rely on `DataLoader` and `TensorDataset` from `torch.utils.data`. Network architectures are defined using `torch.nn`, and training optimization via `torch.optim`. We apply `xavier_uniform_` from `torch.nn.init` for weight initialization, and `autograd.grad` from `torch.autograd` to compute gradients of outputs with respect to inputs, which is essential for enforcing physics-informed loss functions during training. The `scipy.optimize.minimize` function is also used to solve optimization problems, particularly useful for fine-tuning model parameters or minimizing error metrics in inverse problems. Finally, `plotnine` is used to visualize data and results. The complete code and associated datasets for this manuscript are available on GitHub at <https://github.com/AmerFarea/ODE-PINN> (accessed on 10 May 2025).

## 6. Discussion

This paper investigates the use of PINNs for modeling dynamical systems governed by systems of ODEs, revealing its substantial effectiveness and versatility. Our case studies—tumor growth, gene expression, and disease spread using the SIR model—demonstrate the abilities of a PINN to accurately approximate solutions and estimate parameters with notable precision.

The application of PINNs across these diverse systems underscores their versatility and strength. They excel at integrating physical constraints with observational data, delivering accurate and physically consistent solutions. This is particularly advantageous for complex systems where other methods might struggle.

However, the effectiveness of PINNs is influenced by the sensitivity of network architecture and hyperparameters, which necessitates careful tuning for optimal performance. Challenges such as handling noisy and sparse data, and addressing inverse problems highlight the need for robust regularization techniques and improved methods for parameter estimation.

The detailed configuration, as outlined in the Section 4 section, provides insights into the experimental setups and performance of PINNs across various models. The table outlines key parameters such as network architecture, activation functions, learning rates, and simulation details. For example, the tumor growth model was designed with three hidden layers, each having 64 units, and utilized GELU activation functions. The gene expression model had a similar setup but with two hidden layers, also using GELU activation. In contrast, the SIR model featured a network with three hidden layers of 100 units each and employed Tanh activation functions.

The variations in architecture and parameters highlight the importance of tailoring PINN configurations to the specific characteristics of the dynamical system being modeled. For example, the different activation functions (GELU vs. Tanh) and the adjustments in the number of hidden layers reflect attempts to optimize model performance for diverse types of data and system dynamics. The learning rates and batch sizes were consistent across models, emphasizing a common approach to training stability, though the number of epochs and patience settings varied, suggesting different convergence criteria and training durations tailored to each model's complexity.

These parameter settings underscore the adaptability of PINNs but also point to the sensitivity of their performance to hyperparameters and network design. Careful tuning is essential to achieve optimal results, as shown by the varying results across the different case studies. This highlights both the robustness and the challenges associated with applying PINNs to diverse dynamical systems.

Our results demonstrate that PINNs for systems of ordinary differential equations (ODEs) can effectively model biological mechanisms, systems biology, and epidemiological processes. This is important to highlight because, so far, the primary focus of PINNs has been on partial differential equations (PDEs) for physics-related problems. In contrast, fields outside of physics have been largely understudied. Since ODEs dominate in these fields, we concentrated our study on such dynamical systems to showcase their application.

From a technical standpoint, PDEs differ significantly from ODEs, and consequently, the resulting PINNs also differ. To facilitate the use of ODE-based PINNs, we provide a Python software implementation to complement our presentation. We believe that such applications hold tremendous potential for addressing medical and economic problems, many of which are yet to be explored.

## 7. Conclusions

This paper explores the application of Physics-Informed Neural Networks (PINNs) for modeling dynamical systems described by ordinary differential equations (ODEs), with a focus on biological and epidemiological domains that have received relatively little attention in the PINN literature. We present three detailed case studies—tumor growth, gene expression, and disease spread using the SIR model—demonstrating how PINNs can be effectively used for both forward problem approximation and parameter estimation from synthetic data.

Beyond demonstrating the practical applicability of PINNs in these new contexts, our study also emphasizes important implementation aspects. We show that the performance of PINNs is highly sensitive to the choice of network architecture and training strategy, underscoring the need for problem-specific design and tuning. These insights are valuable for researchers and practitioners aiming to apply PINNs to other real-world systems. In addition to these applied contributions, the paper serves a practical purpose by providing clear examples, structured explanations, and an openly accessible Python library—ODE-PINN—that enables the reproduction and extension of our results. This positions the work



as both a practical demonstration of PINNs in non-traditional domains and a valuable resource for newcomers to the field.

In summary, this study showcases the potential of PINNs for modeling ODE-based systems in biology and epidemiology, emphasizing their role in improving our understanding and management of dynamic systems. By offering practical tools and case studies, we provide resources for further exploration in research.

**Author Contributions:** Methodology, A.F. and F.E.-S.; Writing—original draft, A.F.; Writing—review & editing, A.F., O.Y.-H. and F.E.-S.; Supervision, F.E.-S.; Project administration, F.E.-S. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Data Availability Statement:** The raw data supporting the conclusions of this article will be made available by the authors on request.

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

## References

1. Raissi, M.; Perdikaris, P.; Karniadakis, G.E. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.* **2019**, *378*, 686–707. [\[CrossRef\]](#)
2. Karniadakis, G.E.; Kevrekidis, I.G.; Lu, L.; Perdikaris, P.; Wang, S.; Yang, L. Physics-informed machine learning. *Nat. Rev. Phys.* **2021**, *3*, 422–440. [\[CrossRef\]](#)
3. Cuomo, S.; Di Cola, V.S.; Giampaolo, F.; Rozza, G.; Raissi, M.; Piccialli, F. Scientific machine learning through physics-informed neural networks: Where we are and what's next. *J. Sci. Comput.* **2022**, *92*, 88. [\[CrossRef\]](#)
4. Farea, A.; Yli-Harja, O.; Emmert-Streib, F. Understanding Physics-Informed Neural Networks: Techniques, Applications, Trends, and Challenges. *AI* **2024**, *5*, 1534–1557. [\[CrossRef\]](#)
5. Emmert-Streib, F. A heterosynaptic learning rule for neural networks. *Int. J. Mod. Phys. C* **2006**, *17*, 1501–1520. [\[CrossRef\]](#)
6. Sallam, A.; Al Amery, H.; Al-Qudasi, S.; Al-Ghorbani, S.; Rassem, T.H.; Makbol, N.M. Iris recognition system using convolutional neural network. In Proceedings of the 2021 International Conference on Software Engineering & Computer Systems and 4th International Conference on Computational Science and Information Management (ICSECS-ICOCSIM), Pekan, Malaysia, 24–26 August 2021; pp. 109–114.
7. Radman, A.; Sallam, A.; Suandi, S.A. Deep residual network for face sketch synthesis. *Expert Syst. Appl.* **2022**, *190*, 115980. [\[CrossRef\]](#)
8. Smolander, J.; Dehmer, M.; Emmert-Streib, F. Comparing deep belief networks with support vector machines for classifying gene expression data from complex disorders. *FEBS Open Bio* **2019**, *9*, 1232–1248. [\[CrossRef\]](#)
9. Sallam, A.A.; Amery, H.A.; Saeed, A.Y. Iris recognition system using deep learning techniques. *Int. J. Biom.* **2023**, *15*, 705–725. [\[CrossRef\]](#)
10. Sallam, A.A.; Mohammed, B.A.; Abdulbari, M. A Dorsal Hand Vein Recognition System based on Various Machine and Deep Learning Classification Techniques. In Proceedings of the 2023 3rd International Conference on Computing and Information Technology (ICCIT), Tabuk, Saudi Arabia, 13–14 September 2023; pp. 493–501.
11. Rodrigues, J.A. Using Physics-Informed Neural Networks (PINNs) for Tumor Cell Growth Modeling. *Mathematics* **2024**, *12*, 1195. [\[CrossRef\]](#)
12. Raeisi, E.; Yavuz, M.; Khosravifarsani, M.; Fadaei, Y. Mathematical modeling of interactions between colon cancer and immune system with a deep learning algorithm. *Eur. Phys. J. Plus* **2024**, *139*, 1–16. [\[CrossRef\]](#)
13. Meng, X.; Li, Z.; Zhang, D.; Karniadakis, G.E. PPINN: Parareal physics-informed neural network for time-dependent PDEs. *Comput. Methods Appl. Mech. Eng.* **2020**, *370*, 113250. [\[CrossRef\]](#)
14. Mowlavi, S.; Nabi, S. Optimal control of PDEs using physics-informed neural networks. *J. Comput. Phys.* **2023**, *473*, 111731. [\[CrossRef\]](#)
15. Raissi, M.; Yazdani, A.; Karniadakis, G.E. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science* **2020**, *367*, 1026–1030. [\[CrossRef\]](#)
16. Jagtap, A.D.; Kharazmi, E.; Karniadakis, G.E. Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems. *Comput. Methods Appl. Mech. Eng.* **2020**, *365*, 113028. [\[CrossRef\]](#)
17. Brunton, S.L.; Noack, B.R.; Koumoutsakos, P. Machine learning for fluid mechanics. *Annu. Rev. Fluid Mech.* **2020**, *52*, 477–508. [\[CrossRef\]](#)

18. Sun, L.; Gao, H.; Pan, S.; Wang, J.X. Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data. *Comput. Methods Appl. Mech. Eng.* **2020**, *361*, 112732. [\[CrossRef\]](#)
19. Yucesan, Y.A.; Viana, F.A. A physics-informed neural network for wind turbine main bearing fatigue. *Int. J. Progn. Health Manag.* **2020**, *11*, 1. [\[CrossRef\]](#)
20. Lakshminarayana, S.; Sthapit, S.; Maple, C. Application of physics-informed machine learning techniques for power grid parameter estimation. *Sustainability* **2022**, *14*, 2051. [\[CrossRef\]](#)
21. Pun, G.P.; Batra, R.; Ramprasad, R.; Mishin, Y. Physically informed artificial neural networks for atomistic modeling of materials. *Nat. Commun.* **2019**, *10*, 2339. [\[CrossRef\]](#)
22. Mishin, Y. Machine-learning interatomic potentials for materials science. *Acta Mater.* **2021**, *214*, 116980. [\[CrossRef\]](#)
23. Zhang, E.; Dao, M.; Karniadakis, G.E.; Suresh, S. Analyses of internal structures and defects in materials using physics-informed neural networks. *Sci. Adv.* **2022**, *8*, eabk0644. [\[CrossRef\]](#) [\[PubMed\]](#)
24. Waheed, U.B.; Alkhalifah, T.; Haghighat, E.; Song, C.; Virieux, J. PINNtomo: Seismic tomography using physics-informed neural networks. *arXiv* **2021**, arXiv:2104.01588.
25. Meray, A.; Wang, L.; Kurihana, T.; Mastilovic, I.; Praveen, S.; Xu, Z.; Memarzadeh, M.; Lavin, A.; Wainwright, H. Physics-informed surrogate modeling for supporting climate resilience at groundwater contamination sites. *Comput. Geosci.* **2024**, *183*, 105508. [\[CrossRef\]](#)
26. Perez-Raya, I.; Kandlikar, S.G. Thermal modeling of patient-specific breast cancer with physics-based artificial intelligence. *ASME J. Heat Mass Transf.* **2023**, *145*, 031201. [\[CrossRef\]](#)
27. Xiang, Z.; Peng, W.; Zhou, W.; Yao, W. Hybrid finite difference with the physics-informed neural network for solving PDE in complex geometries. *arXiv* **2022**, arXiv:2202.07926.
28. Raissi, M.; Perdikaris, P.; Karniadakis, G.E. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. *arXiv* **2017**, arXiv:1711.10561.
29. Grossmann, T.G.; Komorowska, U.J.; Latz, J.; Schönlieb, C.B. Can physics-informed neural networks beat the finite element method? *IMA J. Appl. Math.* **2024**, *89*, 143–174. [\[CrossRef\]](#)
30. Stiasny, J.; Chatzivasileiadis, S. Physics-informed neural networks for time-domain simulations: Accuracy, computational cost, and flexibility. *Electr. Power Syst. Res.* **2023**, *224*, 109748. [\[CrossRef\]](#)
31. Markidis, S. The old and the new: Can physics-informed deep-learning replace traditional linear solvers? *Front. Big Data* **2021**, *4*, 669097. [\[CrossRef\]](#)
32. Li, Z. A Review of Physics-Informed Neural Networks. *Appl. Comput. Eng.* **2025**, *133*, 165–173. [\[CrossRef\]](#)
33. Laird, A.K. Dynamics of tumour growth. *Br. J. Cancer* **1964**, *18*, 490. [\[CrossRef\]](#) [\[PubMed\]](#)
34. Polynikis, A.; Hogan, S.; Di Bernardo, M. Comparing different ODE modelling approaches for gene regulatory networks. *J. Theor. Biol.* **2009**, *261*, 511–530. [\[CrossRef\]](#) [\[PubMed\]](#)
35. Bolouri, H.; Davidson, E.H. Modeling transcriptional regulatory networks. *BioEssays* **2002**, *24*, 1118–1129. [\[CrossRef\]](#)
36. Harko, T.; Lobo, F.S.; Mak, M.K. Exact analytical solutions of the Susceptible-Infected-Recovered (SIR) epidemic model and of the SIR model with equal death and birth rates. *Appl. Math. Comput.* **2014**, *236*, 184–194. [\[CrossRef\]](#)
37. Beckley, R.; Weatherspoon, C.; Alexander, M.; Chandler, M.; Johnson, A.; Bhatt, G.S. Modeling epidemics with differential equations. *Tenn. State Univ. Intern. Rep.* **2013**, *32*, 33–34.
38. Kröger, M.; Schlickeiser, R. Analytical solution of the SIR-model for the temporal evolution of epidemics. Part A: Time-independent reproduction factor. *J. Phys. A Math. Theor.* **2020**, *53*, 505601. [\[CrossRef\]](#)
39. Schlickeiser, R.; Kröger, M. Analytical solution of the SIR-model for the temporal evolution of epidemics: Part B. Semi-time case. *J. Phys. A Math. Theor.* **2021**, *54*, 175601. [\[CrossRef\]](#)
40. Taubes, C.H. *Modeling Differential Equations in Biology*; Cambridge University Press: Cambridge, UK, 2008.
41. Brauer, F.; Castillo-Chavez, C.; Castillo-Chavez, C. *Mathematical Models in Population Biology and Epidemiology*; Springer: Berlin/Heidelberg, Germany, 2012; Volume 2.
42. Tyson, J.J.; Novak, B. A dynamical paradigm for molecular cell biology. *Trends Cell Biol.* **2020**, *30*, 504–515. [\[CrossRef\]](#)
43. Natsiavas, S.; Paraskevopoulos, E. A set of ordinary differential equations of motion for constrained mechanical systems. *Nonlinear Dyn.* **2015**, *79*, 1911–1938. [\[CrossRef\]](#)
44. Awrejcewicz, J. *Ordinary Differential Equations and Mechanical Systems*; Springer: Berlin/Heidelberg, Germany, 2014.
45. Perez, J.S.; Conesa, M.; Alhama, I. Solving ordinary differential equations by electrical analogy: A multidisciplinary teaching tool. *Eur. J. Phys.* **2016**, *37*, 065703. [\[CrossRef\]](#)
46. Quach, M.; Brunel, N.; d'Alché Buc, F. Estimating parameters and hidden variables in non-linear state-space models based on ODEs for biological networks inference. *Bioinformatics* **2007**, *23*, 3209–3216. [\[CrossRef\]](#) [\[PubMed\]](#)
47. Neisy, A.; Peymany, M. Financial modeling by ordinary and stochastic differential equations. *World Appl. Sci. J.* **2011**, *13*, 2288–2295.
48. Gompertz, B. On the nature of the function expressive of the law of human mortality, and on a new mode of determining the value of life contingencies. In a letter to Francis Baily, Esq. FRS &c. *Philos. Trans. R. Soc. Lond.* **1825**, *115*, 513–583.

49. Tsoularis, A.; Wallace, J. Analysis of logistic growth models. *Math. Biosci.* **2002**, *179*, 21–55. [[CrossRef](#)]
50. Fornalski, K.; Reszczyńska, J.; Dobrzyński, L.; Wysocki, P.; Janiak, M. Possible Source of the Gompertz Law of Proliferating Cancer Cells: Mechanistic Modeling of Tumor Growth. *Acta Phys. Pol. A* **2020**, *138*, 854–862. [[CrossRef](#)]
51. Kim, S.W.; Kim, I.; Lee, J.; Lee, S. Knowledge Integration into deep learning in dynamical systems: An overview and taxonomy. *J. Mech. Sci. Technol.* **2021**, *35*, 1331–1342. [[CrossRef](#)]

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.