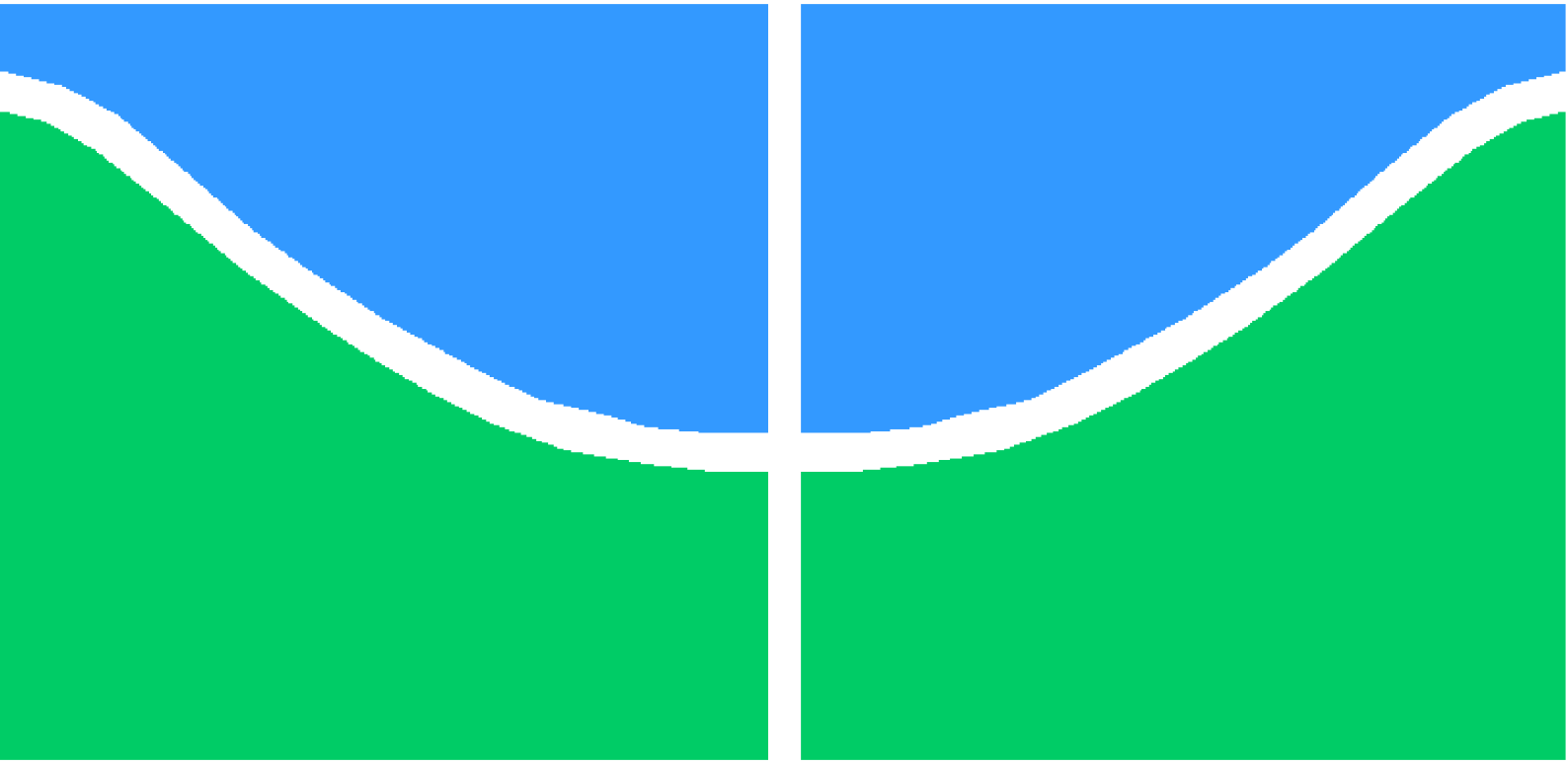


Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Engenharia Eletrônica

Usando Redes Neurais para Estimação de Parâmetros e Solução de Equações Diferenciais Ordinárias que Modelam Crescimento de Tumores

Autor: João Marcelo Almeida de Macedo

Orientador: Dr. Vinicius de Carvalho Rispoli

Brasília, DF

Dezembro de 2023

João Marcelo Almeida de Macedo

Usando Redes Neurais para Estimação de Parâmetros e Solução de Equações Diferenciais Ordinárias que Modelam Crescimento de Tumores

Monografia submetida ao curso de graduação em (Engenharia Eletrônica) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia Eletrônica).

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Dr. Vinicius de Carvalho Rispoli

Brasília, DF

Dezembro de 2023

João Marcelo Almeida de Macedo

Usando Redes Neurais para Estimação de Parâmetros e Solução de Equações Diferenciais Ordinárias que Modelam Crescimento de Tumores/ João Marcelo Almeida de Macedo. – Brasília, DF, Dezembro de 2023-

61 p. : il. (algumas color.) ; 30 cm.

Orientador: Dr. Vinicius de Carvalho Rispoli

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , Dezembro de 2023.

1. Aprendizado de Máquina. 2. Equações diferenciais. I. Dr. Vinicius de Carvalho Rispoli. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Usando Redes Neurais para Estimação de Parâmetros e Solução de Equações Diferenciais Ordinárias que Modelam Crescimento de Tumores

CDU 02:141:005.6

João Marcelo Almeida de Macedo

Usando Redes Neurais para Estimação de Parâmetros e Solução de Equações Diferenciais Ordinárias que Modelam Crescimento de Tumores

Monografia submetida ao curso de graduação em (Engenharia Eletrônica) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia Eletrônica).

Dr. Vinicius de Carvalho Rispoli
Orientador

Dr. Ricardo Ramos Fragelli

Dr. Ronni Geraldo Gomes de Amorim

Brasília, DF
Dezembro de 2023

Agradecimentos

Ao meu orientador Vinicius de Carvalho Rispoli, que desde meus primeiros momentos na faculdade foi minha maior fonte de inspiração e conselhos; Não chegaria até aqui de outra forma.

Ao professor Daniel Costa Araújo pelas incríveis oportunidades concedidas e por acreditar em mim quando eu já não acreditava; Suas ideias me ajudaram a consolidar este trabalho.

Ao professor Ricardo Ramos Fragelli por sua brilhante criatividade que me inspira a continuar passando a corrente do ensino, em busca de tornar nossos arredores um lugar melhor um passo de cada vez.

Ao professor Cristiano Jacques Miosso pela sua inacreditável dedicação ao ensino e cuidado que possui com todos os que participam de sua aula; Levarei para sempre suas conversas inspiradoras e aulas incríveis.

À todos os demais professores que me encantaram com suas aulas e me proporcionaram segurança acerca de minhas habilidades.

À meu amigo Allan Müller, que me ajudou e ensinou com a paciência e a maestria de um grande professor, e que inspirou a concluir este curso, algo que não consigo imaginar sem sua participação nisto.

À todos os meus outros amigos que chegaram até aqui, e também aos que não chegaram. Que todos possamos encontrar a felicidade de nossas maneiras e termos a paz que tanto aguardamos.

À minha família por me incentivar a continuar crescendo e aprendendo mais a cada dia.

‘Todos aqueles que você conheceu
e ainda conhecerá
sabem de algo que você não.”
(Bill Nye)

Resumo

Os estudos sobre as equações diferenciais datam do século XVII e coincidem com a invenção do cálculo diferencial e integral por Newton e Leibniz. Tais equações surgem em diversas aplicações da física, matemática e engenharia, como, por exemplo, na análise de circuitos elétricos e na modelagem do crescimento de tumores. Com relação às suas soluções, pode-se dizer que poucas são aquelas que apresentam soluções analíticas fechadas. A identificação de parâmetros nesse contexto é imprescindível para a correta modelagem dos dados. Nesse sentido, este trabalho se propôs a analisar uma técnica para soluções de equações diferenciais simultaneamente à identificação de parâmetros, utilizando redes neurais. Foram obtidas soluções numéricas e analíticas para equações diferenciais de primeira ordem e sistemas aplicados à área de modelagem de crescimento de tumores, bem como a identificação de suas constantes. Foram utilizados dados experimentais da literatura a respeito do crescimento de tumores mamários em camundongos do modelo FVB/N-Tg(MMTVneu)202Mul/J. Os resultados obtidos mostraram que a técnica é viável, produz soluções precisas e com rapidez. Observou-se, conforme a literatura, que os modelos utilizados para o crescimento de tumores produzem resultados bastante similares entre si.

Palavras-chaves: Redes Neurais. Equações diferenciais. Aprendizado de máquina. Identificação de parâmetros.

Abstract

Studies on the topic of differential equations have been researched since the 17th century, coinciding with the invention of differential and integral calculus by Newton and Leibniz. These equations arise on many applications in the fields of physics, mathematics and engineering, like, for example, in the analysis of electric circuits and in growth tumors modeling. Regarding their solutions, it can be said that only a few of them have closed analytic forms. Parameter estimation in this context can be essential for the correct data fitting. On this topic, this work proposes to analyze a technique aided by neural networks for the simultaneous solving of differential equations and parameter estimation. It will also use experimental data from the literature about mammary tumor growth in the FVB/N-Tg(MMTVneu)202Mul/J mouse model. Final results show that this technique is viable and generates accurate solutions fast. It is also observed that, like previous works concluded, the models utilized in tumor growth modeling all generates similar results.

Key-words: Neural Networks. Differential Equations. Machine Learning. Parameter estimation

Lista de ilustrações

Lista de abreviaturas e siglas

EDO	Equação Diferencial Ordinária
EDP	Equação Diferencial Parcial
API	*Interface* de programação de aplicações.
LSTM	Longa memória de curto-prazo.
LBFGS	Broyden–Fletcher–Goldfarb–Shanno de memória limitada
RLC	Resistor-Indutor-Capacitor
LRC	Indutor-Resistor-Capacitor
PRISM	Proteogenomics Research Institute for Systems Medicine
PK/PD	farmacocinético / farmacodinâmico
TGF	Função de crescimento do tumor

Lista de símbolos

Φ	Solução estimada da EDO
α, β, γ, K	parâmetros
∇	Gradiente
Σ	Somatório
σ	função de ativação sigmoide
∂	Derivada parcial
λ	constante inteira
Λ	somatório das constantes λ
$\prod$	Produtório
Ψ	Função de perda para conjunto de dados

Sumário

Introdução

A inteligência artificial é o ramo da tecnologia que visa automatizar tarefas usualmente realizadas por humanos (CHOLLET, 2018). Este campo viu um crescimento exponencial desde sua primeira concepção real até os dias atuais e não é imprudente estimar que, visto sua intensa importância no tratamento e exploração da vasta quantidade de dados coletadas a cada momento, suas aplicações se tornarão cada vez mais complexas e fundamentais para a sociedade como um todo.

Dentro desse ramo, encontra-se o aprendizado de máquina (do inglês, *Machine Learning*). Ele surge da seguinte questão: "Computadores conseguiriam ir além do que os ordenamos fazer e realizar tarefas por conta própria?" (CHOLLET, 2018). Essa pergunta interessante abre uma alternativa a programação clássica, onde o programador fornece ao computador uma série de instruções a serem executadas, para uma onde as instruções são o objetivo da programação, e não seu meio.

O termo "aprendizado de máquina", entretanto, pode induzir o leitor que a evolução tecnológica permitiu máquinas "pensarem". A verdade é que foi desenvolvido uma classe de algoritmos que permitem aproximar processos bastante complexos de forma muito eficiente, permitindo automatizar tarefas que antes eram atribuídas exclusivamente a humanos (STEVENS; ANTIGA; VIEHMANN, 2020).

Ainda assim, o termo é bastante aceito e popular, e o misticismo por trás deste campo de estudo ajudou a impulsionar curiosos engenheiros para o desenvolvimento e colaboração com esta ciência que se torna cada vez mais sofisticada a cada dia, bem como a questão ética a respeito dos limites que este desenvolvimento deveria ter (LEVINE; RAND; RAHWAN, 2020).

Nesse contexto, o conceito de redes neurais floresceu como uma forma de dar profundidade ao aprendizado de máquina através de múltiplas camadas de "neurônios" artificiais, que recebem um conjunto de dados e extrai padrões muitas vezes invisíveis aos olhos humanos. O primeiro trabalho associado a operações matemáticas para simular a rede neural cerebral ocorreu em 1943 (MCCULLOCH; PITTS, 1943), antes dos primeiros projetos concretos envolvendo inteligências artificiais no geral. Ele se inspirou no conceito do cérebro humano como uma forma de desenvolver um algoritmo que pudesse "aprender" padrões através da autocorreção.

Redes neurais foram teorizadas como solucionadoras de equações diferenciais ordinárias (EDOs) e equações diferenciais parciais (EDPs) pela sua alta capacidade em aproximar funções (DUFERA, 2021; LAGARIS; LIKAS, 1998). Além disso, funções sigmoides já são usadas em redes neurais para aproximar funções contínuas desde 1989

(CYBENKO, 1989). Redes neurais recorrentes, muito usadas em sistemas dependentes do tempo, foram utilizadas em (FRICKE et al., 2020), onde uma célula recorrente especializada foi criada para integração numérica e solução e estimação dos parâmetros da EDO. Há diversos outros trabalhos a respeito, como (GIOVANNI et al., 2020), (PHAM et al., 2020) e (WEN; CHAOLU; WANG, 2022), por exemplo, todos com suas particularidades na teoria e execução de algoritmos para a solução de EDOs.

O trabalho de (LAGARIS; LIKAS, 1998) é o principal alicerce deste, visto que seu método entrega bons resultados com simplicidade e rapidez. Entretanto, o trabalho supracitado não menciona identificação de parâmetros, uma área bastante importante no desenvolvimento de modelos precisos. Trabalhos como (MEHRKANOON; FALCK; SUYKENS, 2012) dividem o campo da estimação de parâmetros entre modelos que assumem valores iniciais aleatórios e refinam a aproximação ao comparar o resultado da rede com o valor esperado e modelos que usam funções de aproximação para obter o resultado com menos custo computacional.

O trabalho de (DUA, 2011) usa redes neurais artificiais para aproximar o modelo do sistema de EDOs e só depois estima os parâmetros a partir da aproximação obtida. (DUA; DUA, 2011) realiza este processo simultaneamente usando a aproximação do trabalho anterior, sendo um outro alicerce deste trabalho, visto que a mesma técnica de empregar a solução da EDO junto à identificação de parâmetros foi utilizada, embora com algumas adaptações. (JAMILI; DUA, 2021) utiliza o processo de modelar a solução usando redes neurais artificiais para aplicar em equações diferenciais parciais. (ROJAS, 2020) aplica o mesmo processo com êxito em um modelo de dano.

No campo das ciências biomédicas, uma área onde as EDOs cumprem um importante papel é na modelagem do crescimento de tumores. Existem diversas equações que tratam de modelar este crescimento sob diversos casos diferentes, e o termo "modelos empíricos" surgiu da observação do comportamento de fatores como volume ou massa, mas sem levar em conta os processos complexos biológicos que podem afetá-los (BAJZER; MARUŠIĆ; VUK-PAVLOVIĆ, 1996). Neste contexto, os modelos logísticos, Von Bertallanfy e Gompertz são EDOs frequentemente citadas em diversos trabalhos que tratam do estudo do crescimento de tumores (BAJZER; MARUŠIĆ; VUK-PAVLOVIĆ, 1996),(MARUSIC; BAJZER, 1993),(KOZIOL; FALLS; SCHNITZER, 2020), (BENZEKRY et al., 2014), (MURPHY; JAAFARI; DOBROVOLNY, 2016), tendo em vista as naturezas simplificadas dos modelos empíricos. Outro modelo mais recente conhecido como o modelo Simeoni e suas adaptações têm se mostrado uma alternativa aos outros métodos utilizados quando há a introdução de quimioterápicos para controlar o crescimento dos tumores no experimento modelado (SIMEONI et al., 2004), (KOZIOL; FALLS; SCHNITZER, 2020), (TATE et al., 2014).

O modelo citado é especialmente interessante visto que se trata de um sistema de

equações. Há poucos trabalhos na literatura utilizando redes neurais para a solução de EDOs simultaneamente à identificação dos parâmetros das mesmas principalmente neste campo da modelagem tumoral, e nesse contexto este trabalho se insere como um estudo a respeito da precisão da solução e identificação de parâmetros por redes neurais das EDOs quando introduzidas a conjuntos de dados (*datasets*) no campo do crescimento de tumores.

Objetivos gerais e específicos

O objetivo geral deste trabalho é o de investigar os parâmetros dos modelos logístico, Gompertz, Von Bertallanfy e Simeoni por meio de redes neurais. Serão utilizadas como referências, para os problemas de controle ótimo a serem resolvidos, dados experimentais do crescimento de tumores mamários em camundongos do tipo FVB/N-Tg(MMTVneu)202Mul/J (KOZIOL; FALLS; SCHNITZER, 2020). Em relação aos objetivos específicos, pretende-se:

- encontrar vantagens e limitações desse tipo de solução quando comparada à soluções analíticas destes mesmos modelos;
- estender a solução das EDOs para intervalos arbitrários, especialmente além do intervalo $[0, 1]$ onde as redes neurais são conhecidas por terem um bom comportamento;
- manter o código simples e ao mesmo tempo robusto;
- Encontrar parâmetros adequados que aproximem a solução das EDOs aos dados experimentais utilizados;
- comparar os resultados qualitativamente através de análise gráfica.

1 Referencial Teórico

1.1 Introdução ao Aprendizado de máquina

Os programas baseados em aprendizado de máquina recebem um conjunto de entradas e saídas e, através de técnicas de correção e retroalimentação, concebem as regras que associam estas entradas a saídas dentro de uma margem aceitável de erro. A programação clássica recebe um conjunto de entradas e instruções para então gerar a saída correspondente.

O nome "aprendizado de máquina" vem do fato que o sistema é treinado ao invés de programado diretamente (CHOLLET, 2018). O sistema busca por alguma relação entre a entrada e a saída utilizando exemplos pré-fornecidos de como deve associá-los. (CHOLLET, 2018) cita três importantes requerimentos para o algoritmo em Aprendizado de máquina:

- Entrada de dados — Se, por exemplo, o objetivo for catálogo de imagens, a entrada de dados seriam figuras.
- Exemplos da saída esperada — Mantendo o mesmo exemplo, as saídas seriam o catálogo em si, como "animal" ou "carro."
- Um meio de quantificar o quão bem o algoritmo está se saindo — O que caracteriza aprendizado de máquina. Através de uma medição do quão bem ele realiza sua tarefa, o sistema pode corrigir-se e ajustar como o algoritmo funciona. Este ajuste é chamado de aprendizado.

Todos os algoritmos de aprendizado de máquina consistem em encontrar transformações que processam dados em representações mais úteis para uma certa tarefa. Estas transformações são mais difíceis de serem concebidas, então o que o sistema faz é procurar em um espaço pré-definido de possibilidades, chamado de espaço de hipótese (CHOLLET, 2018).

Após conceber um conjunto de instruções bem calibrado, ele pode então ser usado para sua tarefa. O aprendizado é um processo sempre contínuo e sua velocidade depende da quantidade de exemplos fornecida, bem como da quantidade de dados processados. Essa simples ideia de procurar por representações úteis dos dados através de uma retroalimentação possibilita a concepção de tecnologias cada vez mais incríveis que vão desde o reconhecimento por voz até direção autônoma de carros.

Uma área do aprendizado de máquina é o aprendizado profundo (do inglês *Deep Learning*), que começou a ganhar força e ser estudado em 1990 e desde então tem crescido exponencialmente para diversas áreas da tecnologia. Este campo esmiúça o processo do aprendizado em múltiplas camadas, das quais se deriva a ideia de "profundo" em seu nome (CHOLLET, 2018). Técnicas em aprendizado profundo modernas envolvem dezenas ou até mesmo centenas de camadas e o aprendizado se dá através das chamadas redes neurais artificiais, inspiradas na neurociência.

No aprendizado profundo, cada camada possui um peso (ou parâmetro) específico, e o aprendizado envolve descobrir qual o valor de cada peso que corresponde a associação entrada-saída fornecida ao sistema. A descoberta destes pesos envolve comparar a saída do algoritmo com a saída real. Essa comparação se dá através da função de perda (CHOLLET, 2018) e seu resultado é retroalimentado diretamente nos pesos através de um otimizador. Essa retroalimentação é a principal parte de uma rede neural e onde de fato o sistema "aprende" com os dados passados. Em seu estado inicial, o valor dos pesos é a princípio aleatório e o sistema tenta otimizá-lo o mais rápido possível para os valores que deixam a função de perda em níveis aceitáveis.

A função de perda se baseia em um critério escolhido pelo engenheiro, uma métrica de sucesso para o algoritmo que irá gerar um valor real que deve ser minimizado para que o sistema se comporte da maneira mais precisa possível (STEVENS; ANTIGA; VIEHMANN, 2020).

O motivo do aprendizado profundo ter se tornado um sucesso imediato é porque além de oferecer uma performance superior no geral, ele automatiza a engenharia de características do sistema (CHOLLET, 2018). Em um processo usual de aprendizado de máquina, a transformação dos dados costuma se dar em representações não-refinadas, geralmente envolvendo uma árvore de decisões ou projeções não-lineares multidimensionais. Caso seja necessário representações complexas, este passo deveria ser realizado por um humano. Com o aprendizado profundo, o sinal de retroalimentação ajusta não apenas o parâmetro por si só, mas também todos os outros modelos que dependem deste parâmetro automaticamente (CHOLLET, 2018).

Algoritmos de aprendizado de máquina precisam de geralmente grandes quantidades de informação para operar. Isso significa que o procedimento padrão é alimentar o modelo com conjuntos de dados para que possam aprender de acordo; Entretanto, isso nem sempre pode ser necessário. Existem três formas de treinamento, chamadas supervisionadas, não supervisionadas e por reforço. No treinamento supervisionado, o conjunto de dados fornece as saídas esperadas para o sistema, e este busca otimizar o modelo de forma a diminuir a diferença entre a saída do sistema e a saída esperada. O treinamento não supervisionado é pouco comum e não é alimentado com conjuntos de dados esperados. Isto pode ser útil para deixar o modelo determinar padrões ocultos em sua entrada

fornecida. O treinamento por reforço não possui uma saída esperada exata, e o aprendizado é baseado em estímulos positivos ou negativos do conjunto de dados. Seu objetivo é aprender sequências de ações de forma a maximizar os reforços positivos (ROJAS, 2020).

Quando o aprendizado profundo envolve apenas poucas camadas em seu algoritmo, ele pode ser chamado de aprendizado raso (do inglês *Shallow Learning*). Este simplifica o consumo de energia e a complexidade dos algoritmos, sendo usado para processos e tarefas mais simples. O aprendizado profundo está incluído dentro das chamadas redes neurais, uma subcategoria de aprendizado de máquina. Os modelos chamados por redes neurais utilizam o algoritmo de aprendizado profundo para poucas camadas escondidas. Este trabalho visa utilizar redes neurais para a solução e estimação de parâmetros de EDOs em um contexto de aplicação real.

1.2 Parâmetros de uma Rede Neural

Todos os sistemas de aprendizado de máquina utilizam tensores como estrutura básica de dados (CHOLLET, 2018). Tensores são vetores multidimensionais, uma generalização de matrizes e vetores para um número não-negativo qualquer de dimensões (STEVENS; ANTIGA; VIEHMANN, 2020). Um tensor de dimensão zero é chamado de escalar, enquanto um tensor de dimensão um é chamado de vetor. O termo tensor costuma ser mais popular quando se trata de três ou mais dimensões.

As bibliotecas de ciência de dados da linguagem *Python*, em especial a *Numpy* ou *PyTorch*, possuem a funcionalidade dos tensores implementadas de forma eficiente em baixo nível para serem utilizadas por uma interface de programação de aplicações (API) alto nível conveniente (STEVENS; ANTIGA; VIEHMANN, 2020). Por padrão, os tensores nestas bibliotecas possuem seus dados em ponto flutuante de 32 bits.

Uma camada é um módulo do processamento de dados que recebe um tensor comportando os dados como entrada, processa estes dados ao transformá-los em informação útil e gera um tensor como saída. Sendo um sistema que recebe uma entrada e gera uma saída, camadas podem ser pensadas como funções compostas em cadeia para formar o sistema (GOODFELLOW; BENGIO; COURVILLE, 2016). Em uma rede neural, o número de camadas representa a profundidade da rede, profundidade esta relacionada ao *deep* (profundo) em *Deep Learning*. Existem diversos tipos de camada para variados formatos de tensores e tipos de dados. Estas camadas possuem pesos ou parâmetros específicos que são calibrados pela própria rede no decorrer do aprendizado.

Hidden layers ou camadas escondidas são aquelas nas quais a rede não mostra sua saída (GOODFELLOW; BENGIO; COURVILLE, 2016). Um sistema de aprendizado profundo pode possuir dezenas ou até mais camadas como esta. Costuma-se chamar a última camada como “camada de saída” (*Output layer*) e a primeira como “camada de

entrada" (*Input layers*). Esta última, por motivos históricos, costuma também ser considerada uma camada escondida, por sua saída também ser oculta (STEVENS; ANTIGA; VIEHMANN, 2020). Modelos que contém uma ou mais camadas escondidas conectadas de forma densa também são chamados de perceptron multicamada (*multilayer perceptron*) (CARNIELLO; VITAL; VALLE, 2021).

A Figura 1 ilustra visualmente duas camadas escondidas entre a entrada e saída. É evidente que no exemplo seu resultado é diretamente enviado para a camada de saída e não há necessidade de sabê-los, visto que, por uma perspectiva humana, um subconjunto de pesos não teria significado se analisado separadamente.

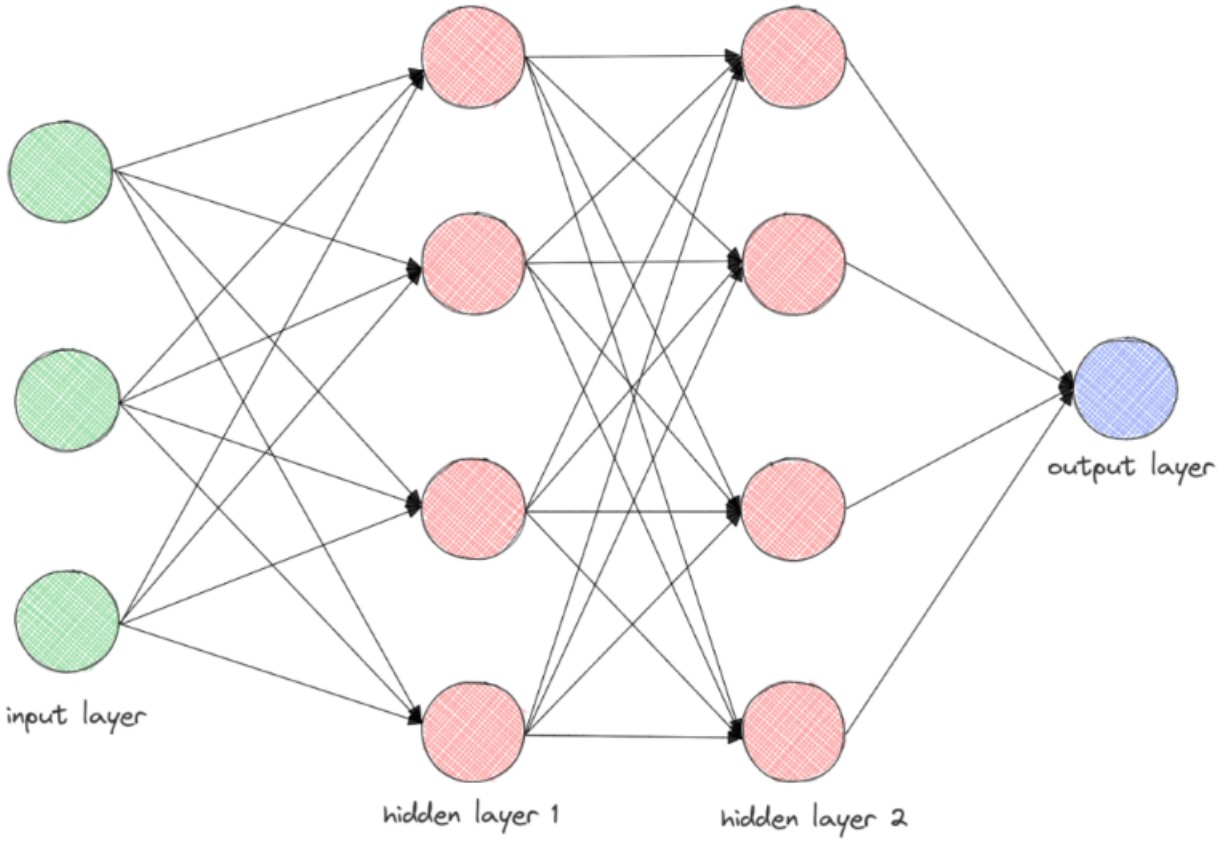


Figura 1 – Ilustração de um modelo básico de rede neural com uma camada de entrada, saída e duas camadas escondidas intermediárias. Modelos mais complexos podem conter dezenas ou centenas de camadas intermediárias. Fonte: (ANTONIADIS, 2022)

Estas camadas são tipicamente vetoriais e cada um destes valores dentro desse vetor são chamados de "neurônios" (*Neurons*) (STEVENS; ANTIGA; VIEHMANN, 2020) ou "unidades" (*Units*) (GOODFELLOW; BENGIO; COURVILLE, 2016), que agem em paralelo representando uma função de vetor para escalar, e na prática realizam uma transformação linear na entrada. A equação 1.1 é uma representação clássica da transformação que os neurônios em uma camada realizam (STEVENS; ANTIGA; VIEHMANN, 2020), onde w e b são os parâmetros a serem refinados com o aprendizado e x é a entrada. O

parâmetro w é muitas vezes chamado de peso e o parâmetro b é conhecido como viés (do inglês, *bias*).

$$o = wx + b \tag{1.1}$$

Se cada neurônio aplica uma transformação linear, as funções de ativação existem para quebrar a linearidade de cada transformação, mediadas pelos pesos que são ativamente atualizados no decorrer do aprendizado. Funções de ativação diferentes geram saídas diferentes, variando desde valores contínuos arbitrários até probabilidades. Sem elas, as camadas só conseguiriam aprender transformações lineares a partir dos dados da entrada, mas isto geraria um espaço de hipóteses muito restrito para o aprendizado e não haveria benefícios em múltiplas camadas, pois todas realizariam operações lineares (CHOLLET, 2018).

Aplicar funções de ativação nas camadas permite que a saída tenha inclinações diferentes em certos valores. Uma composição destas funções permitiria então que a rede aproximasse funções arbitrárias complexas. Na última camada, ela também pode permitir concentrar os valores da saída em um intervalo definido (STEVENS; ANTIGA; VIEHMANN, 2020), útil em casos de classificação, por exemplo. Para fins de exemplificação, a função sigmoide reduz a função para o intervalo (0,1), possui derivadas contínuas e por isso garante uma relação suave entre entrada e saída de um neurônio. A função tanh reduz o intervalo para (-1,1) e é um pouco menos sensível a problemas de saturação. A Figura 2 ilustra algumas funções de ativação presentes em redes neurais.

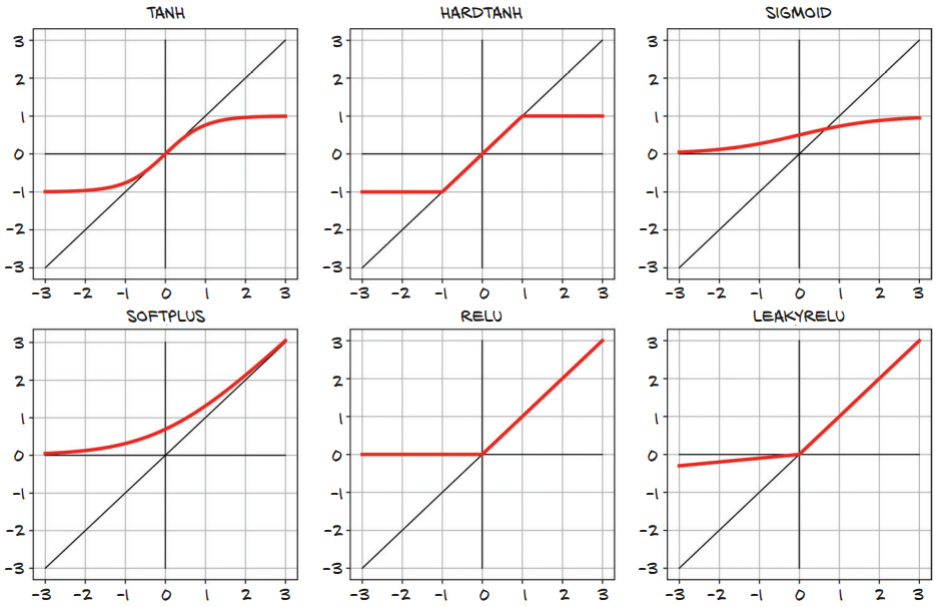


Figura 2 – Gráfico simplificado de algumas funções de ativação. As funções *RELU*, *tanh* e sigmoide são as mais utilizadas. Fonte: (STEVENS; ANTIGA; VIEHMANN, 2020).

A função de perda trata de avaliar a métrica que está sendo aprendida baseada no resultado esperado, gerando um valor escalar que o processo de aprendizado tentará minimizar (STEVENS; ANTIGA; VIEHMANN, 2020). Geralmente, isto envolve alguma variação de tomar a diferença entre o resultado obtido pelo sistema e o resultado esperado. Ela é uma forma de priorizar quais erros consertar durante o treino, ajustando os parâmetros com viés para certas amostras que demonstraram maior urgência que outras com uma menor perda.

A função de otimização determina como os pesos de cada camada e os parâmetros da rede devem ser alterados com base na função de perda. Assim, após cada iteração, a rede é atualizada visando minimizar a função de perda. O otimizador pode atualizar os parâmetros do sistema através da taxa de variação da função de perda em relação aos mesmos. A biblioteca *PyTorch* para o Python mantém um histórico dos tensores que resultam de operações em certo definido tensor e calcula o gradiente da perda automaticamente, muitas vezes facilitando o processo de otimização (PASZKE et al., 2019).

A Figura 3 exibe o fluxo de um modelo em uma rede neural: O sistema recebe uma entrada X, que então passa por camadas onde sofre transformações lineares e pode ser aplicada uma função de ativação. O modelo gera uma resposta que é comparada à resposta esperada através de uma função de perda, que gera um valor de perda. Este valor é passado ao otimizador que define e atualiza as mudanças nos pesos das transformações lineares das camadas. Este processo se repete por todo o treinamento e o esperado é a diminuição da função de perda até níveis aceitáveis.

Existem diversas arquiteturas para redes neurais, como as convolucionais, modulares, LSTM, etc. Este trabalho dará ênfase na arquitetura chamada *perceptron* multicamada, constituída de camadas densas (onde os neurônios de uma camada são conectados a todos os outros das camadas seguintes ou anteriores) e possuindo uma ou mais camadas escondidas.

1.3 Solução Numérica de EDOs com Redes Neurais

Redes neurais rasas ou profundas são ferramentas poderosas para diversas tarefas dentro das ciências biomédicas como, por exemplo, classificação e segmentação de tumores. No entanto, como redes neurais são aproximadores universais e EDOs são inerentemente modelos matemáticos em que se deseja determinar uma função solução, conseguiria ela ser capaz de resolver numericamente equações diferenciais?

A resposta é que sim (CARNIELLO; VITAL; VALLE, 2021). Há bastante teoria envolvendo equações diferenciais e a modelagem por redes neurais, logo seria possível encarar este problema de diversas formas. Neste trabalho iremos utilizar a descrição da solução numérica de um problema de valor inicial baseada em (LAGARIS; LIKAS, 1998).

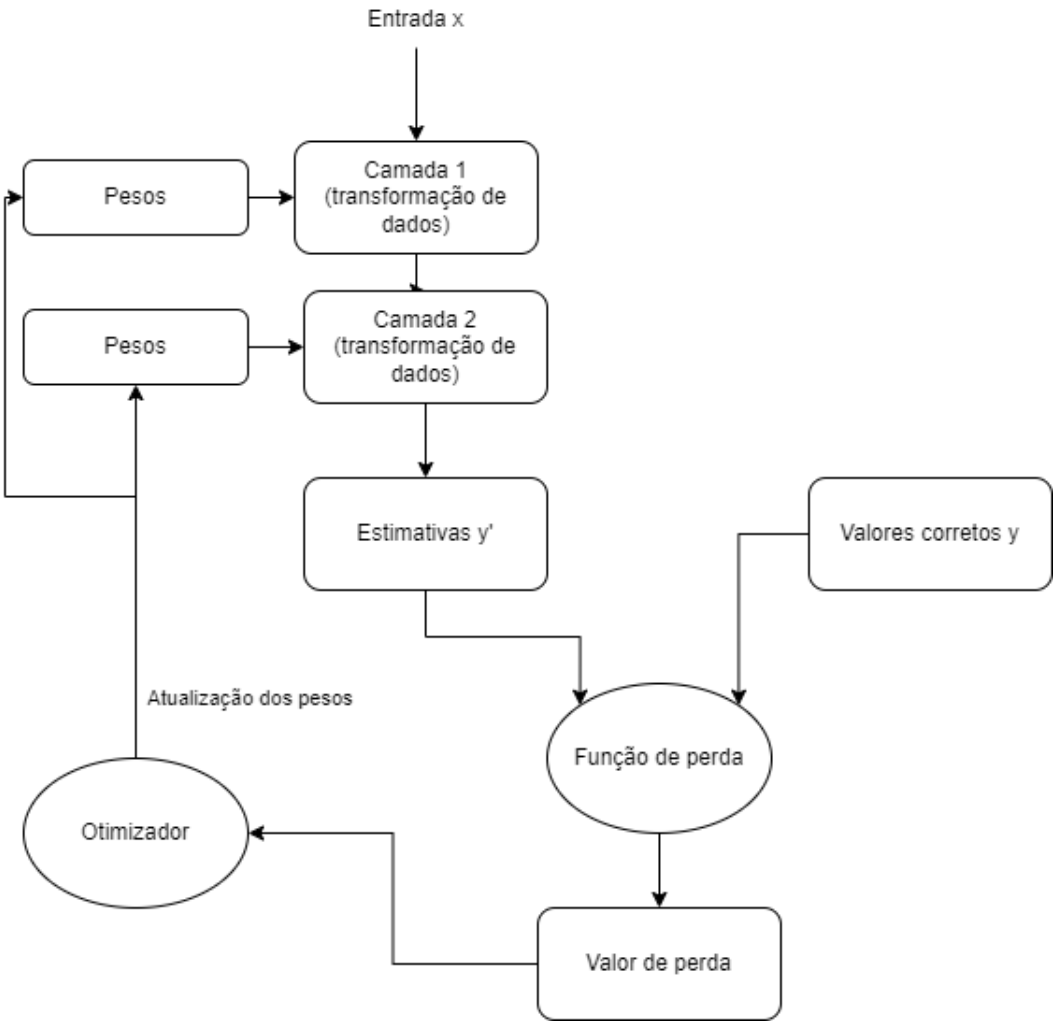


Figura 3 – Relação entre a rede, as camadas, a função de perda e o otimizador. Estes blocos constituem todo o funcionamento de um modelo de aprendizado profundo no geral. Fonte: Elaborado pelo autor.

Parte-se da definição geral de uma equação diferencial:

$$G(\vec{x}, \Phi(\vec{x}), \nabla\Phi(\vec{x}), \nabla^2\Phi(\vec{x})) = 0, \quad \vec{x} \in D \tag{1.2}$$

Esta equação pode estar sujeita a condições iniciais e de contorno, sendo $\vec{x} = (x_1, x_2, x_3, ..., x_n) \in \mathbb{R}^n$, $D \subset \mathbb{R}^n$ o domínio e $\Phi(\vec{x})$ a solução a ser calculada. Utilizando o método da colocação, que assume uma discretização do domínio D e do seu contorno Sem um conjunto $\hat{D}$ e $\hat{S}$, respectivamente. O problema então se torna um sistema de equações sujeito a restrições impostas pelas condições de contorno:

$$G(\vec{x_i}, \Phi(\vec{x_i}), \nabla\Phi(\vec{x_i}), \nabla^2\Phi(\vec{x_i})) = 0, \quad \forall\vec{x_i} \in \hat{D} \tag{1.3}$$

Se $\Phi_t(\vec{x}, \vec{p})$ simboliza uma possível solução com parâmetros ajustáveis p, o problema se transforma na equação 1.4, sujeito às restrições aplicadas pela condição de contorno:

$$\min_{\vec{p}} \sum_{\vec{x_i} \subset \hat{D}} G(\vec{x_i}, \Phi_t(\vec{x_i}; \vec{p}), \nabla\Phi_t(\vec{x_i}; \vec{p}), \nabla^2\Phi_t(\vec{x_i}; \vec{p}))^2 \tag{1.4}$$

A solução Φ_t será buscada através de uma rede neural. O ponto e vírgula separa a entrada $\vec{x}_i$ do parâmetro $\vec{p}$. É preciso então encontrar uma forma de solução que satisfaça as condições de contorno.

Para uma EDO de primeira ordem, uma função apropriada é:

$$\Phi_t(\vec{x}) = A(\vec{x}) + F(\vec{x}, N(\vec{x}; \vec{p})) \tag{1.5}$$

sendo $N(\vec{x}; \vec{p})$ a representação da rede neural de parâmetros $\vec{p}$ e entrada $\vec{x}$. $A(\vec{x})$ satisfaz as condições de contorno e F é construído de modo a não contribuir para estas condições. Uma tentativa bem aceita de solução para a equação 1.5 é (LAGARIS; LIKAS, 1998):

$$\Phi_t(\vec{x}) = A + \vec{x}N(\vec{x}; \vec{p}) \tag{1.6}$$

sendo $N(\vec{x}, \vec{p})$ a saída da rede neural com entrada $\vec{x}$ e pesos $\vec{p}$ e A é a condição inicial $\Phi_t(0)$. Em sua própria construção, ela satisfaz as restrições que o problema de condição inicial pode trazer.

Em relação ao processo de minimização da equação 1.4, a computação do gradiente do erro em relação aos parâmetros da rede envolve não apenas o gradiente da rede neural, mas também o gradiente da derivada da rede em relação às suas entradas (LAGARIS; LIKAS, 1998). Considerando uma entrada $\vec{x} = (x_1, x_2, x_3, ..., x_n)$, a saída da rede contendo uma camada escondida com função de ativação sigmoide de H neurônios e saída linear é:

$$N = \sum_{i=1}^{H} v_i \sigma(z_i) \tag{1.7}$$

sendo $z_i = \sum_{j=1}^{N} w_{ij} x_j + u_i$, representando as transformações lineares da camada escondida, w_{ij} é o peso da camada escondida, v_i é o peso da camada da saída, u_i é o viés da camada escondida e σ é a função de ativação sigmoide. A derivada em relação a entrada então é:

$$\frac{\partial^k N}{\partial x_j^k} = \sum_{i=1}^{H} v_i w_{ij}^k \sigma_i^{(k)} \tag{1.8}$$

Para $\sigma_i = \sigma(z_i)$ e $\sigma^{(k)}$ representando a $k-$ésima derivada da sigmoide. A partir disso, então:

$$\frac{\partial^{\lambda_1}}{\partial x_1^{\lambda_1}} \frac{\partial^{\lambda_2}}{\partial x_2^{\lambda_2}} \frac{\partial^{\lambda_3}}{\partial x_3^{\lambda_3}} ... \frac{\partial^{\lambda_n}}{\partial x_N^{\lambda_n}} N = \sum_{i=1}^{n} v_i P_i \sigma_i^{(\Lambda)} \tag{1.9}$$

sendo $P_i = \prod_{k=1}^{n} w_{ik}^{\lambda_k}$ e $\Lambda = \sum_{i=1}^{n} \lambda_i$. É possível perceber pela equação 1.9 que a derivada da rede em relação a suas entradas é equivalente a uma rede neural N_g com uma camada escondida de mesmos pesos e vieses, com cada peso v_i sendo substituídos por $v_i P_i$. A função de ativação da camada escondida é substituída pela Λ-ésima derivada da sigmoide. O gradiente de N_g em relação aos parâmetros da rede original é obtido por (LAGARIS; LIKAS, 1998):

$$\frac{\partial N_g}{\partial w_{ij}} = x_j v_i P_i \sigma_i^{\Lambda+1} + v_i \lambda_j w_{ij}^{\Lambda_j - 1} \left(\prod_{k=1, k \neq j} w_{ik}^{\lambda_k} \right) \sigma_i^{(\Lambda)} \tag{1.10}$$

Com a derivada definida, qualquer técnica de minimização pode ser utilizada. Neste trabalho, será usado o algoritmo *Limited-memory Broyden–Fletcher–Goldfarb–Shanno* (LBFGS) como otimizador da rede neural.

A função de perda a ser minimizada pela rede neural é:

$$E(\vec{p}) = \sum_i \left(\frac{d\Phi_t(x_i)}{dx} - f(x_i, \Phi_t(x_i)) \right)^2 \tag{1.11}$$

sendo que x_i são pontos do domínio e $f(x_i, \Phi_t(x_i)) = \frac{d\Phi(x)}{dx}$. Como

$$\frac{d\Phi_t(x)}{dx} = N(x; \vec{p}) + x\frac{dN(x, \vec{p})}{dx},$$

é natural calcular o gradiente do erro a partir da equação 1.10. A biblioteca *PyTorch* mantém salvo o gradiente de seus tensores automaticamente (PASZKE et al., 2019), facilitando este processo.

Para o caso de uma EDO de segunda ordem, a mesma estratégia e função de perda é utilizada, com a pequena diferença que $f(x_i, \Phi_t(x_i))$ terá um parâmetro a mais, resultando em $f(x_i, \Phi_t(x_i), \frac{d\Phi_t(x_i)}{dx})$. Mas a solução estimada $\Phi_t(x)$ sofre uma mudança para acomodar uma nova condição inicial:

$$\Phi_t(x) = A + A_2 x + x^2 N(x; \vec{p}), \tag{1.12}$$

onde A representa $\Phi_t(0)$ e A_2 representa $\left.\frac{d\Phi_t(x)}{dx}\right|_{x=0}$. De forma similar, é possível também resolver sistemas de duas EDOs. Consideram-se duas redes neurais N_1 e N_2, cada uma com uma solução estimada própria $\Phi_{t_1}(x)$ e $\Phi_{t_2}(x)$, respectivamente. A estimativa da solução segue a mesma forma da equação 1.6 O erro a ser minimizado é dado então por:

$$E(\vec{p}) = \sum_i \left[\left(\frac{d\Phi_{t_1}(x_i)}{dx} - f(x_i, \Phi_{t_1}(x_i), \Phi_{t_2}(x_i)) \right)^2 + \left(\frac{d\Phi_{t_2}(x_i)}{dx} - g(x_i, \Phi_{t_1}(x_i), \Phi_{t_2}(x_i)) \right)^2 \right] \tag{1.13}$$

1.4 Identificação de Parâmetros

Através da rede neural é possível também realizar a identificação de parâmetros, quando se deseja encontrar a solução de uma EDO que se aproxima de um conjunto de pontos fornecidos. Nesta situação, a identificação de parâmetros ocorrerá simultaneamente ao processo de encontrar a solução da rede. Para que isso ocorra, entretanto, é necessário modificar a função de perda 1.11 e 1.13 para incluir a parcela $\Psi(x)$ que comparará a saída da solução encontrada com o conjunto de dados fornecidos $D(x_D)$, onde x_D são os pontos do domínio do conjunto de dados. Tal parcela está em 1.14 para o caso de uma EDO escalar.

$$\Psi(\vec{x}) = \sum_{i=0}^{N} (\Phi_t(x_{D_i}) - D(x_{D_i}))^2. \tag{1.14}$$

Assim, para a solução do caso escalar, a função de perda a ser minimizada terá a forma:

$$\mathcal{L}(\vec{p}) = \alpha_1 E(\vec{p}) + \alpha_2 \Psi(\vec{x}) \tag{1.15}$$

As constantes α_1 e α_2 determinam os pesos para cada parcela e pode ser útil caso algum conjunto de dados possua um alto desvio-padrão, evitando que a rede produza resultados que fujam da forma da EDO. Neste trabalho, o sistema de EDOs que será numericamente resolvido demandará a inclusão da equação 1.16 na função de perda.

$$\Psi(\vec{x}) = \sum_{i=0}^{N} (\Phi_{t_1}(x_{D_i}) + \Phi_{t_2}(x_{D_i}) - D(x_{D_i}))^2. \tag{1.16}$$

Isso se deve ao fato que a variável dependente do problema que será resolvido é tal que $V(t) = \Phi_{t_1}(t) + \Phi_{t_2}(t)$, com $\Phi_{t_1}(t)$ e $\Phi_{t_2}(t)$ sendo as soluções do sistema.

Descrito dessa forma, a identificação de parâmetros simultânea à solução da EDO pode ser considerada um problema de controle ótimo, visto que se busca controlar (isto é, encontrar os parâmetros) um sistema dinâmico (descrito pela EDO) minimizando uma função objetiva (Função de perda). Este tipo de problema costuma exigir uma base avançada em Teoria de Controle, mas será solucionado aqui como um problema genérico auxiliado pela rede neural.

2 Metodologia

2.1 Dados

O conjunto de dados utilizado no experimento deste trabalho foi extraído do artigo de Koziol, Falls e Schnitzer (2020) com auxílio da ferramenta WebPlotDigitizer[1]. Esta ferramenta permite a extração de um conjunto de dados a partir do gráfico que contenha os pontos de interesse. Por ser uma tarefa manual, entretanto, há um pequeno erro associado ao dado obtido. Este erro é desprezível, no entanto, porque o escopo deste projeto necessita apenas de um conjunto de dados cuja disposição dos pontos obedece um experimento real para fins de comparação e teste da precisão da rede neural.

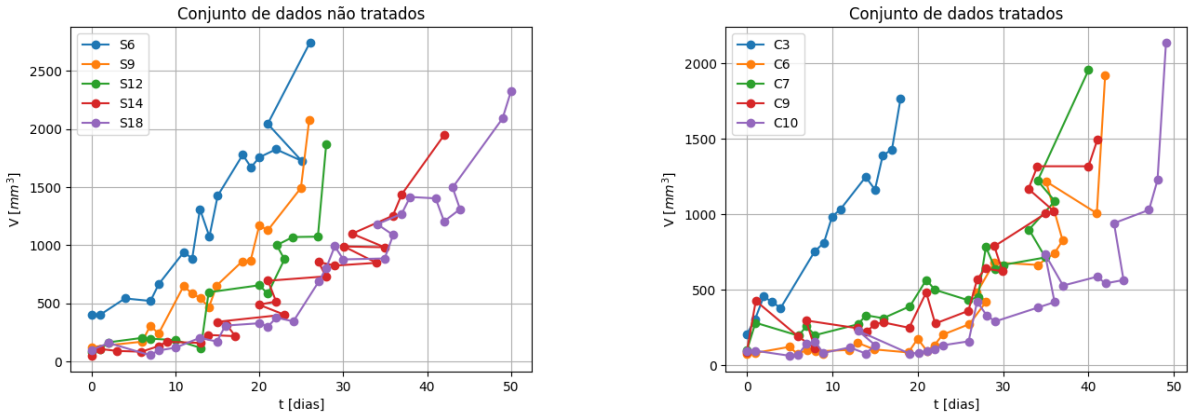
O experimento foi realizado no instituto *Proteogenomics Research Institute for Systems Medicine* (PRISM), onde o objetivo era obter um modelo adequado e reproduzível para tumores mamários em camundongos do tipo FVB/N-Tg(MMTVneu)202Mul/J. Todos os camundongos foram criados e selecionados pelo instituto. O experimento foi realizado a partir da aparição do tumor mamário no dia 0. Dos 40 camundongos envolvidos, 19 receberam uma dose única de cisplatina (5 mg/kg) no dia 0 e os demais não receberam qualquer tratamento. Todos os tumores iniciaram com um volume menor do que $700mm^3$ e a cada dia útil o volume do tumor era medido usando a fórmula

$$V = \frac{\text{tamanho} \times \text{comprimento}^2}{2}.$$

Os camundongos no experimento original foram sacrificados quando os tumores atingiram um volume aproximado de 2500 mm^3 ou por apresentar complicações. O protocolo de eutanásia utilizado no experimento foi aprovado pelo comitê responsável *Institutional Animal Care and Use Committee of Prism* (KOZIOL; FALLS; SCHNITZER, 2020).

Foram escolhidos para o presente trabalho 10 conjuntos de dados do artigo de Koziol, Falls e Schnitzer (2020). A seleção foi aleatória dentre os conjuntos de dados presentes no artigo original e a numeração deles foi mantida neste trabalho para fácil comparação. Após a seleção dos gráficos no artigo, a ferramenta WebPlotDigitizer foi usada para selecionar os pontos e manualmente extraí-los. Os 10 conjuntos de dados selecionados foram divididos em duas partes, tratados e não-tratados, com 5 conjuntos de dados para cada um. Aqui foi adicionada uma nomenclatura com o prefixo S para os experimentos sem tratamento e C para os que houveram tratamento. A Figura 4 permite a visualização dos dados, onde o eixo t é o tempo em dias e o eixo V é o volume em mm^3

[1] https://automeris.io/WebPlotDigitizer/



(a) Experimentos com camundongos não-tratados

(b) Experimentos com camundongos tratados

Figura 4 – Conjunto de dados experimentais utilizados. A imagem da esquerda e da direita referem-se aos experimentos não-tratados e tratados, respectivamente. Dados extraídos da referência (KOZIOL; FALLS; SCHNITZER, 2020). Fonte: Elaborado pelo autor (2023).

2.2 Modelos

Este trabalho usa como referência quatro modelos matemáticos distintos para análise do crescimento de tumores, sendo três deles bem descritos na literatura (BAJZER; MARUŠIĆ; VUK-PAVLOVIĆ, 1996), (MARUSIC; BAJZER, 1993) e um modelo relativamente mais recente conhecido como modelo Simeoni, que obteve bons resultados até então (TATE et al., 2014). Nesta seção vamos apresentar os modelos utilizados.

2.2.1 Equação logística generalizada

O primeiro dos modelos é a equação logística generalizada, desenvolvida para modelar crescimento como uma função sigmoide (BAJZER; MARUŠIĆ; VUK-PAVLOVIĆ, 1996). Este é um modelo bastante comum na literatura e utilizada para modelar os mais diversos tipos de problemas. Ele é dado pelo problema de valor inicial:

$$\frac{dy(t)}{dt} = \alpha y \left(1 - \left(\frac{y}{K}\right)^{\gamma}\right), \quad y(t_0) = y_0, \quad \gamma, \alpha > 0 \tag{2.1}$$

Esta é uma equação separável e sua solução pode ser encontrada de forma analítica como:

$$y(t) = \frac{K}{\left(1 + (-1 + \frac{K}{y_0})^{\gamma} e^{-\alpha\gamma(t-t_0)}\right)^{\frac{1}{\gamma}}} \tag{2.2}$$

Sua principal característica é o decrescimento linear da taxa de crescimento relativo em relação ao volume observado em 2.1. Nesta equação, α é o coeficiente relacionado à cinética da proliferação e K é o volume máximo ou capacidade de carga (BENZEKRY et al., 2014). A Figura 5 mostra uma solução da equação logística generalizada para um

conjunto determinado de parâmetros. Note que a solução tende para o valor de K. A equação logística padrão é um caso especial da equação 2.1 para o caso $\gamma = k = \alpha = 1$.

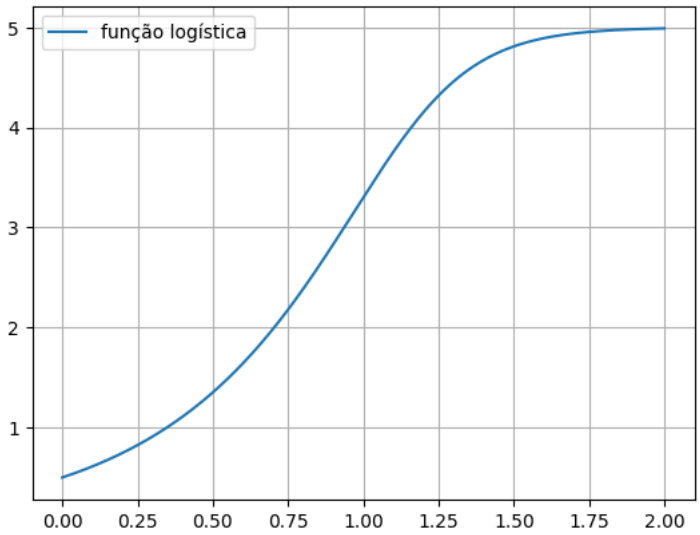


Figura 5 – Solução da equação logística generalizada para $K = 5$, $\alpha = 2$, $\gamma = 3$, $t_0 = 0$ e $y_0 = 0.5$ no intervalo [0,2]. Fonte: Elaborado pelo autor (2023).

A equação logística precisará ser normalizada para apropriadamente ser treinada pela rede neural a um intervalo adequado, visto que as redes neurais trabalham melhor quando os valores da entrada estão em um intervalo de $[0, 1]$ ou $[-1, 1]$ (CHOLLET, 2018). Para realizar as normalizações, considere as substituições $y(t) = \tilde{y}(t) \times y_{\max}$ e $t = \tilde{t} \times t_{\max}$ na equação 2.1. Então fazendo as mudanças de variáveis obtemos:

$$\frac{y_{\max}}{t_{\max}} \frac{d\tilde{y}(t)}{d\tilde{t}} = \alpha \tilde{y}(t) y_{\max} \left(1 - \left(\frac{\tilde{y}(t) y_{\max}}{K}\right)^{\gamma}\right) \tag{2.3}$$

Simplificando e rearranjando os termos da equação 2.3:

$$\frac{d\tilde{y}(t)}{d\tilde{t}} = \alpha t_{\max} \tilde{y}(t) \left(1 - \left(\tilde{y}(t) \frac{y_{\max}}{K}\right)^{\gamma}\right) \tag{2.4}$$

Agora, realizando as substituições $\tilde{\alpha} = \alpha t_{\max}$ e $\tilde{K} = \frac{K}{y_{\max}}$, resta uma versão normalizada da equação 2.1 que pode ser treinada pela rede neural com rápida convergência:

$$\frac{d\tilde{y}(t)}{d\tilde{t}} = \tilde{\alpha} \tilde{y}(t) \left(1 - \left(\frac{\tilde{y}(t)}{\tilde{K}}\right)^{\gamma}\right) \tag{2.5}$$

Com $\tilde{y}(0) = \frac{y(0)}{y_{\max}}$. Os parâmetros da equação 2.5 são facilmente convertidos nos parâmetros da equação 2.1, visto que sabemos as relações entre eles. Isto permite que o treinamento e a identificação de parâmetros ocorra em um domínio bem mais favorável à rede neural.

2.2.2 Equação de Gompertz

Outra equação utilizada na modelagem do crescimento dos tumores é a chamada equação de Gompertz. Este modelo descreve o crescimento com uma curva logarítmica

e foi frequentemente usado antes da mais conhecida função logística (KOZIOL; FALLS; SCHNITZER, 2020), (BENZEKRY et al., 2014). A equação de Gompertz é uma EDO também separável e dada por:

$$\frac{dy(t)}{dt} = ay(t)\ln(\beta y(t)), \quad y(0) = y_0, \quad a < 0, \quad \beta > 0 \tag{2.6}$$

cuja solução é:

$$y(t) = \frac{(\beta y_0)^{e^{at}}}{\beta}. \tag{2.7}$$

O coeficiente a representa o coeficiente da taxa de crescimento, enquanto β é o inverso da capacidade de carga. A Figura 6 ilustra o comportamento da solução para determinados parâmetros. Estes parâmetros fazem a solução tender para $\frac{1}{\beta}$.

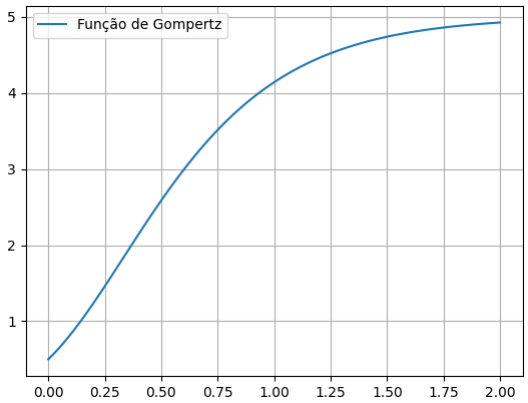


Figura 6 – Solução da equação de Gompertz para $K = 5$, $a = -2.5$, $\beta = 0.2$ e $y_0 = 0.5$ no intervalo [0,2]. Fonte: Elaborado pelo autor (2023).

A normalização desta função ocorre de forma parecida à função logística. As substituições $y(t) = \tilde{y}(t) \times y_{\max}$ e $t = \tilde{t} \times t_{\max}$ na equação 2.6 resultam em:

$$\frac{y_{\max}}{t_{\max}} \frac{d\tilde{y}(t)}{d\tilde{t}} = ay_{\max}\tilde{y}(t)\ln(\beta\tilde{y}(t)y_{\max}) \tag{2.8}$$

Simplificando e rearranjando:

$$\frac{d\tilde{y}(t)}{d\tilde{t}} = at_{\max}\tilde{y}(t)\ln\left(\beta y_{\max}\tilde{y}(t)\right) \tag{2.9}$$

Com as substituições $\tilde{a} = at_{\max}$ e $\tilde{\beta} = \beta y_{\max}$, então a EDO normalizada resultante é:

$$\frac{d\tilde{y}(t)}{d\tilde{t}} = \tilde{a}\tilde{y}(t)\ln\left(\tilde{\beta}\tilde{y}(t)\right), \tag{2.10}$$

sendo $\tilde{y}(0) = \frac{y(0)}{y_{\max}}$.

2.2.3 Equação de Von Bertallanfy

Von Bertallanfy baseou sua equação na ideia de que a taxa de crescimento resultante de um sistema celular vem do equilíbrio entre o crescimento (síntese) e a morte

(destruição) das células (BENZEKRY et al., 2014). O modelo descrito na equação 2.11 então se tornou a forma geral de sua equação, bastante usada para modelar o crescimento de tumores (BAJZER; MARUŠIĆ; VUK-PAVLOVIĆ, 1996).

$$\frac{dy(t)}{dt} = ay^{\gamma} - by, \quad y(0) = y_0, \quad a, b > 0 \tag{2.11}$$

O parâmetro a representa a taxa de síntese e o parâmetro b, a taxa de destruição no sistema. Sua solução está na equação 2.12 (BENZEKRY et al., 2014), e a Figura 7 ilustra esta solução usando um determinado conjunto de parâmetros. É possível notar que a solução tende a $(\frac{a}{b})^3 = 512$ dado as constantes escolhidas.

$$y(t) = \left(\frac{a}{b} + \left({y_0}^{1-\gamma} - \frac{a}{b}\right) e^{-b(1-\gamma)t}\right)^{\frac{1}{1-\gamma}} \tag{2.12}$$

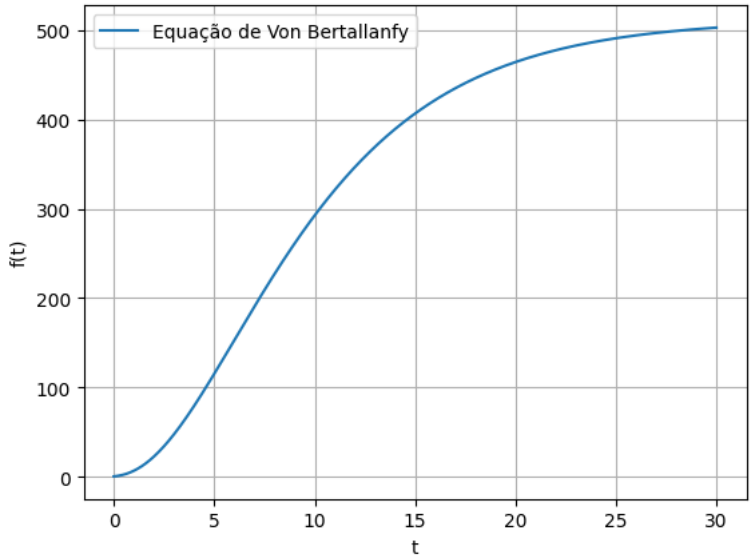


Figura 7 – Solução da equação de Von Bertallanfy para $a = 4$, $b = 0.5$, $\gamma = \frac{2}{3}$ e $y_0 = 0.5$ no intervalo [0,30]. Fonte: Elaborado pelo autor (2023).

A normalização da equação 2.11 acontece como as demais, usando as substituições $y(t) = \tilde{y}(t) \times y_{\max}$ e $t = \tilde{t} \times t_{\max}$:

$$\frac{y_{\max}}{t_{\max}} \frac{d\tilde{y}(t)}{d\tilde{t}} = a\tilde{y(t)}^{\gamma} \times {y_{\max}}^{\gamma} - b\tilde{y(t)} \times y_{\max} \tag{2.13}$$

Simplificando e rearranjando os termos:

$$\frac{d\tilde{y}(t)}{d\tilde{t}} = at_{\max}(y_{\max})^{\gamma-1}\tilde{y}(t)^{\gamma} - bt_{\max}\tilde{y}(t). \tag{2.14}$$

substituindo $\tilde{a} = at_{\max}(y_{\max})^{\gamma-1}$ e $\tilde{b} = bt_{\max}$, chegamos à versão normalizada da equação 2.11, com $\tilde{y_0} = \frac{y_0}{y_{\max}}$:

$$\frac{d\tilde{y}(t)}{d\tilde{t}} = \tilde{a}\tilde{y}^{\gamma} - \tilde{b}\tilde{y}, \quad \tilde{y}(0) = \tilde{y_0} \tag{2.15}$$

2.2.4 Modelo Simeoni

Um modelo do tipo farmacocinético / farmacodinâmico (PK/PD) surgiu de estudos de animais em 2003 (TATE et al., 2014), e tanto este modelo quanto suas adaptações se provaram eficazes na modelagem do crescimento de tumores (MURPHY; JAAFARI; DOBROVOLNY, 2016). Resumido como um sistema de equações, sua principal característica é assumir que a ação dos fármacos não é instantâneo sob quimioterapia; Células de tumor passam por diversos estágios sucessivos de dano por conta dos mecanismos presentes nos fármacos. Isto é descrito como uma série de processos com atrasos em um sistema de EDOs, que podem ter um número arbitrário de estágios (TATE et al., 2014). Este trabalho utiliza uma versão adaptada onde o conjunto arbitrário de EDOs é resumido por um atraso em um único processo como mostrado na equação 2.16:

$$\begin{cases} \frac{dZ_1}{dt} = f_{\text{TGF}}(t) - k_1 c(t) Z_1(t) \\ \frac{dZ_2}{dt} = k_1 c(t) Z_1(t) - k_2 Z_2(t - t_0) \end{cases} \tag{2.16}$$

sendo $f_{\text{TGF}}(t)$ a função do crescimento de tumor (TGF) (KOZIOL; FALLS; SCHNITZER, 2020) dada pela equação 2.17. Os parâmetros λ_0, λ_1 e ϕ governam o comportamento da TGF. As condições iniciais são $Z_1(0) = V_0$ e $Z_2(0) = 0$. k_1 e k_2 são constantes que se referem à taxa nas quais as células são induzidas a destruição, como no caso de k_1, ou danificadas, como no caso de k_2. O termo $c(t)$ é modelado por uma exponencial na forma e^{-bt} e representa a concentração de agentes quimioterapêuticos em Z_1, que por sua vez representa o tumor ativo, aumentando de acordo com a função TGF. $Z_2(t)$ representa então as células danificadas pelo tratamento em qualquer estágio. O volume total do tumor é dado por $V(t) = Z_1(t) + Z_2(t)$ (KOZIOL; FALLS; SCHNITZER, 2020).

$$f_{\text{TGF}}(t) = \frac{\lambda_0 Z_1(t)}{\left(1 + \left(\frac{\lambda_0}{\lambda_1} V(t)\right)^{\phi}\right)^{\frac{1}{\phi}}} \tag{2.17}$$

Esse modelo, diferente dos outros, não possui um platô onde o volume se estabiliza. Ao invés disso, ele se concentra na mudança rápida entre um crescimento exponencial e um linear. Note que, na equação 2.17, se $\left(\frac{\lambda_0}{\lambda_1} V(t)\right)^{\phi}$ for maior do que 1, para um valor de Φ alto o suficiente, esta função pode ser aproximada para um crescimento linear. Caso seja menor do que 1, ela se comportará com um crescimento exponencial.

O modelo Simeoni em geral não possui uma solução analítica, algo comum entre EDOs. Sua solução numérica pode ser obtida de múltiplas formas, inclusive com o auxílio de redes neurais, como no caso deste trabalho. A Figura 8 ilustra o modelo para um conjunto específico de constantes, sendo possível perceber a ausência de um platô na forma da solução. A constante $\phi = 20$ permite que a solução tenha uma fase linear e outra exponencial com transição abrupta (SIMEONI et al., 2004). Os dados coletados consideraram que $t_0 = 0$, e esta mesma constante foi adotada no trabalho.

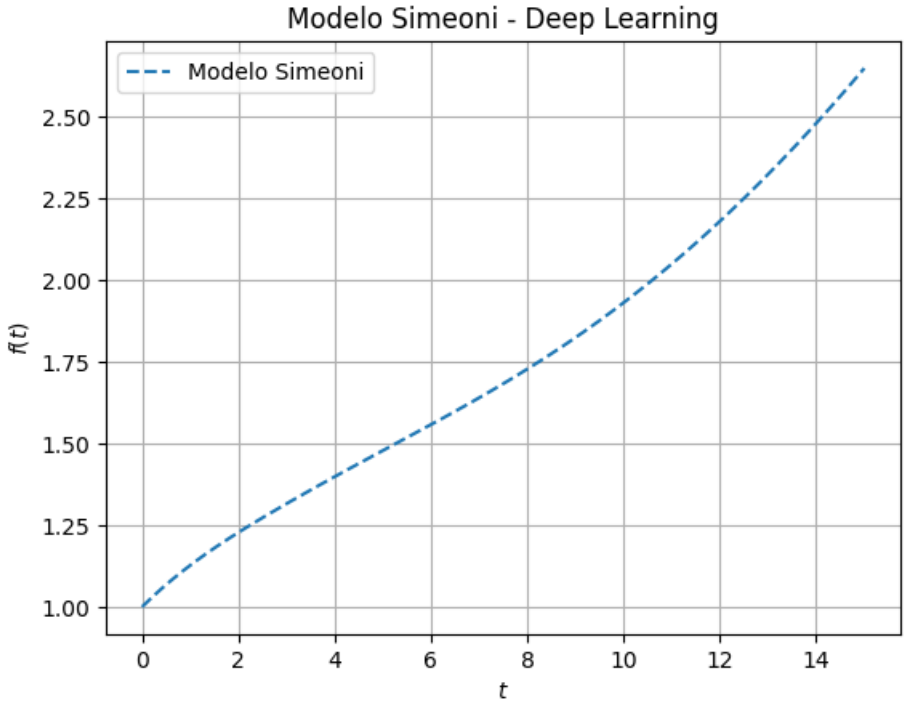


Figura 8 – Solução do modelo Simeoni para $\lambda_0 = 0.146$, $\lambda_1 = 0.334$, $\phi = 20$, $k_1 = 0.469$, $k_2 = 8.4 \times 10^{-4}$, e $y_0 = 1$ no intervalo [0,15]. Fonte: Elaborado pelo autor (2023).

Sua versão normalizada segue um padrão parecido com a das demais equações. As substituições $Z_1 = \tilde{Z_1} \times V_{\max}$, $Z_2 = \tilde{Z_2} \times V_{\max}$, $t = \tilde{t} \times V_{\max}$ e $V = \tilde{V} \times V_{\max}$ são aplicadas em 2.16:

$$\begin{cases} \frac{V_{\max}}{t_{\max}} \frac{d\tilde{Z_1}}{d\tilde{t}} = \frac{\lambda_0 V_{\max} \tilde{Z_1}(t)}{\left(1+\left(\frac{\lambda_0}{\lambda_1} V_{\max} \tilde{V}(t)\right)^{\phi}\right)^{\frac{1}{\phi}}} - k_1 c(t) V_{\max} \tilde{Z_1}(t) \\ \frac{V_{\max}}{t_{\max}} \frac{d\tilde{Z_2}}{d\tilde{t}} = k_1 c(t) V_{\max} \tilde{Z_1}(t) - k_2 V_{\max} \tilde{Z_2}(t - t_0) \end{cases} \tag{2.18}$$

Simplificando e rearranjando a equação 2.18, chega-se em:

$$\begin{cases} \frac{d\tilde{Z_1}}{d\tilde{t}} = \frac{\lambda_0 t_{\max} \tilde{Z_1}(t)}{\left(1+\left(\frac{\lambda_0}{\lambda_1} V_{\max} \tilde{V}(t)\right)^{\phi}\right)^{\frac{1}{\phi}}} - k_1 c(t) t_{\max} \tilde{Z_1}(t) \\ \frac{d\tilde{Z_2}}{d\tilde{t}} = k_1 c(t) t_{\max} \tilde{Z_1}(t) - k_2 t_{\max} \tilde{Z_2}(t - t_0) \end{cases} \tag{2.19}$$

Agora empregando as substituições $\tilde{\lambda_0} = \lambda_0 t_{\max}$, $\tilde{\lambda_1} = \lambda_1 t_{\max} (V_{\max})^{-1}$, $\tilde{k_1} = k_1 t_{\max}$ e $\tilde{k_2} = k_2 t_{\max}$ na equação 2.19:

$$\begin{cases} \frac{d\tilde{Z_1}}{d\tilde{t}} = \frac{\tilde{\lambda_0} \tilde{Z_1}(t)}{\left(1+\left(\frac{\tilde{\lambda_0}}{\tilde{\lambda_1}} \tilde{V}(t)\right)^{\phi}\right)^{\frac{1}{\phi}}} - \tilde{k_1} \tilde{Z_1}(t) \\ \frac{d\tilde{Z_2}}{d\tilde{t}} = \tilde{k_1} c(t) \tilde{Z_1}(t) - \tilde{k_2} \tilde{Z_2}(t - t_0) \end{cases} \tag{2.20}$$

A normalização de $c(t)$ modelado como e^{-bt} se torna $e^{-\tilde{b}t}$, sendo $\tilde{b} = t_{\max} b$.

2.3 Funções de Perda

Tanto para a solução dos modelos quanto para a identificação de parâmetros serão utilizadas funções de perda diferentes para cada uma das situações. Para o modelo logístico a função de perda será dada por:

$$\mathcal{L}_{\text{Logística}} = \sum_i \left(\frac{d\Phi_t(x_i)}{dx} - \alpha\Phi_t \left(1 - \left(\frac{\Phi_t}{K} \right)^{\gamma} \right) \right)^2 + \sum_i (\Phi_t(X_{D_i}) - D(X_{D_i}))^2. \tag{2.21}$$

Para o modelo de Gompertz a função de perda será:

$$\mathcal{L}_{\text{Gompertz}} = \sum_i \left(\frac{d\Phi_t(x_i)}{dx} - a\Phi_t(x_i) \ln(\beta\Phi_t(x_i)) \right)^2 + \sum_i (\Phi_t(X_{D_i}) - D(X_{D_i}))^2. \tag{2.22}$$

No caso do modelo de von Bertallanfy a função de perda tem a forma:

$$\mathcal{L}_{\text{Bertallanfy}} = \sum_i \left(\frac{d\Phi_t(x_i)}{dx} - (a(\Phi_t(x_i))^{\gamma} - b\Phi_t(x_i)) \right)^2 + \sum_i (\Phi_t(X_{D_i}) - D(X_{D_i}))^2. \tag{2.23}$$

Finalmente, para o caso do modelo de Simeoni a função de perda precisará considerar as duas equações do sistema e assumirá a forma:

$$\begin{aligned} \mathcal{L}_{\text{Simeoni}} = & \sum_i \left(\frac{d\Phi_{t_1}(x_i)}{dx} - (f_{\text{TGF}}(x_i) - k_1 c(x_i) Z_1(x_i)) \right)^2 \\ & + \sum_i \left(\frac{d\Phi_{t_2}(x_i)}{dx} - (k_1 c(x_i) Z_1(x_i) - k_2 c(x_i) Z_2(x_i - x_a)) \right)^2 \\ & + 2 \sum_i (\Phi_{t_1}(X_{D_i}) + \Phi_{t_2}(X_{D_i}) - D(X_{D_i}))^2. \end{aligned} \tag{2.24}$$

2.4 Experimento

2.4.1 Ferramentas utilizadas

Este trabalho utilizará a biblioteca *Pytorch* (PASZKE et al., 2019) para a construção do código de uma rede neural capaz de resolver as EDOs necessárias. o formato utilizado para os arquivos foi o *Jupyter Notebook* (KLUYVER et al., 2016). As amostras serão geradas diretamente no código e os hiperparâmetros serão variados conforme a necessidade para se observar o impacto na saída do modelo.

A biblioteca *Pytorch* foi escolhida não apenas pela sua interface amigável, mas também pelas funções especializadas no cálculo dos gradientes de tensores, que no caso de um modelo de regressão como esse, se prova bastante útil e simplifica o código. Outras bibliotecas usuais para ciência de dados do *Python* também serão usadas, como a *Numpy* (HARRIS et al., 2020) e *Matplotlib* (HUNTER, 2007). A biblioteca *pandas* (MCKINNEY

et al., 2010) foi utilizada para importar e exportar os conjuntos de dados trabalhados em *Python*

O teorema da aproximação universal garante que qualquer função contínua pode ser aproximada por uma rede neural com uma camada escondida intermediária (CARNIELLO; VITAL; VALLE, 2021). Isto livra a necessidade de configurar o número de camadas e proporciona melhor foco em fatores como número de neurônios, funções de ativação, etc.

2.4.2 Descrição do Experimento

Para realizar o experimento, primeiro foi necessário a elaboração do código capaz de solucionar a EDO simultaneamente à identificação de parâmetros usando redes neurais, minimizando as funções de perda das equações 2.21, 2.22, 2.23 e 2.24. Com o código pronto, os conjuntos de dados foram extraídos da referência (KOZIOL; FALLS; SCHNITZER, 2020). Para gerar imagens comparativas, o método de Monte Carlo foi aplicado em um experimento realizado com 100 iterações, onde então se coletou cada um dos gráficos finais e calculou-se a média e o desvio-padrão para cada ponto do gráfico. A média dos valores para cada ponto é o que é utilizado na comparação gráfica, e o máximo desvio-padrão é somado e subtraído da média de pontos para exibir a variabilidade da rede neural em solucionar o mesmo problema, visto que com a inicialização randômica dos pesos, os resultados podem nem sempre ser iguais.

Enquanto este estudo estatístico está expresso graficamente para dois conjuntos de dados tratados e dois conjuntos de dados não-tratados, as tabelas geradas com os parâmetros foram obtidos escolhendo a iteração que obteve o menor valor de perda para todos os conjuntos de dados. Não foi possível fazer estudos estatísticos quanto a isso, visto que múltiplas combinações de parâmetros podem gerar respostas similares, e sua média ou desvio-padrão não possuem significado se aplicados à solução analítica.

O código foi construído em etapas. A primeira consistiu na utilização das bibliotecas mencionadas para a elaboração de um programa capaz de solucionar uma EDO arbitrária simples utilizando as técnicas da referência (LAGARIS; LIKAS, 1998). Após a confirmação que o programa desenvolvido conseguia aproximar EDOs com alta precisão, o teste foi realizado nos modelos citados. Porém, o modelo Simeoni exigiu uma reestruturação do código para lidar com as instabilidades sofridas. A normalização das entradas solucionou grande parte deste problema.

Outro ponto relacionado ao modelo Simeoni foi a constante t_0. A captura dos dados da referência (KOZIOL; FALLS; SCHNITZER, 2020) deixava implícito que a constante tinha valor 0. Entretanto, a escolha de utilizar o atraso como forma de modelo deixava incertezas no porquê a constante havia sido tomada como 0. Testes realizados mostravam

que havia uma diferença muito pequena se adicionado um valor de atraso levemente maior e, para manter a consistência com a referência utilizada, foi preferível manter a constante igual a 0. Isto tem o efeito de considerar o modelo Simeoni com apenas duas EDOs, algo que já era contemplado pelo modelo original, que deixa o número de EDOs como arbitrário.

Após corrigir os problemas relacionados ao modelo Simeoni, um conjunto de dados sintético foi inserido para ajustar as funções de perda com a equação 1.14. Após a confirmação de resultados esperados, os conjuntos de dados extraídos da referência (KOZIOL; FALLS; SCHNITZER, 2020) foram utilizados e constantes foram encontradas. Neste período também foi realizada uma otimização dos hiperparâmetros para cada modelo, buscando melhorar a convergência e o tempo de resposta dos modelos.

Diversos testes foram realizados para se configurar os hiperparâmetros da rede neural. Constatou-se que um número maior de neurônios não apresentava diferença perceptível, algo intrínseco à forma de solução utilizando o método da referência (LAGARIS; LIKAS, 1998); Isto significa que uma rede pequena, de menor custo computacional, ainda consegue aproximar EDOs facilmente. Um maior número de pontos de entrada aumentava o tamanho da norma das parcelas da função de perda, prejudicando a convergência da rede.

As possibilidades de otimizadores envolveram LBFGS, Adam e o gradiente descendente estocástico. O melhor absoluto nos testes foi o LBFGS, apresentando convergência quadrática de grande precisão. A taxa de aprendizado na ordem de 10^{-2} auxiliou a estabilidade do treinamento em todos os modelos ao mesmo tempo que proporcionou um baixo número de épocas, agilizando assim a simulação de Monte Carlo.

As funções de ativação em geral tiveram resultados similares. A $ReLU$ não obteve uma precisão satisfatória, mas a tangente hiberbólica e a sigmoide obteram, como mencionado na referência (LAGARIS; LIKAS, 1998). Durante os testes, o modelo de Gompertz obtinha uma maior precisão com a função $\tanh(\cdot)$. As demais continuaram com a recomendação da literatura.

Possibilidades em se usar o erro médio absoluto ao invés do quadrático expresso na literatura foram testadas para a função de perda. Este erro se provou uma alternativa viável principalmente quando colocado na parcela que envolve a identificação de parâmetros. Entretanto, os testes realizados favoreciam o erro quadrático principalmente na questão de estabilidade da rede. Outras funções de erro foram brevemente consideradas, mas descartadas em favor da simplicidade e estabilidade do erro quadrático.

Outro fator que precisou de muitos testes envolveu a normalização da função da perda. A princípio ela começou normalizada em todas as parcelas da função. Entretanto, os treinos não estavam obtendo resultados satisfatórios. Um dos testes envolveu tirar

a normalização apenas da parcela da identificação de parâmetros e a partir daí a rede parecia se encaixar perfeitamente com o conjunto de dados. Entretanto, logo foi constatado que a retirar a normalização daquela parcela fazia com que a rede priorizasse quase que totalmente o encaixe no conjunto de dados. Por isso, a solução da EDO se tornava muito imprecisa e a identificação de parâmetros perdia valor, já que não correspondia de fato à solução. Retirar a normalização da parcela referente à solução da EDO faz com que a rede priorize bem mais a mesma (pois ela possui aproximadamente 500 pontos enquanto a parcela referente ao conjunto de dados possui entre 15 e 30), tornando a identificação de parâmetros bem precisa.

O otimizador detecta que a parcela referente à identificação de parâmetros continua sendo parte integral para a minimização da função de perda e mesmo sem normalização consegue fazer a solução da EDO encaixar-se nos pontos do conjunto de dados. Utilizar o erro médio quadrático na função de perda é o que causa esta detecção, visto que os pontos muito fora da solução tomam a prioridade para serem minimizados internamente, desprezando a necessidade da normalização.

A rede a princípio também sofria com certas instabilidades, no sentido de que muitas vezes a inicialização explodia os parâmetros internos imediatamente. Este problema é suspeito de ser inerente à forma de resolução por redes neurais. Como os pesos são gerados aleatoriamente (e precisam ser assim para uma boa convergência e treinamento), pode ocorrer de valores incômodos causarem gradientes muito altos no início do treinamento. Isso entretanto não impacta durante os testes porque após a inicialização correta, a rede tende a convergir bem e rápida. A solução final minimiza este problema.

Por mais que o trabalho original do modelo Simeoni tenha utilizado $\phi = 20$ na equação 2.17, os testes demonstraram que é possível, através da escolha dos outros parâmetros, utilizar valores de ϕ menores do que este e ainda obter resultados precisos. Por isso, ϕ neste trabalho foi tratado como um parâmetro treinável.

Após a conclusão dos testes, o experimento descrito no início deste subcapítulo foi realizado separadamente para cada modelo. Ao todo, foram feitas 1000 iterações para a extração de todos os dados. A configuração da rede seguiu os hiperparâmetros da Tabela 1 e utilizou todos os conjuntos de dados da Figura 4. Para a comparação gráfica, os conjuntos S12, S6, C9 e C6 foram utilizados.

Logo, foi também comparado os modelos entre si em uma superposição de todos no mesmo gráfico para certos conjuntos de dados. Os parâmetros das iterações que obtiveram o menor valor de perda foram exibidos em tabelas para cada um dos conjuntos de dados.

As configurações dos hiperparâmetros das redes neurais para cada modelo estão descritas na Tabela 1. Estes valores foram obtidos com os testes empíricos e também levando em conta as recomendações da literatura, buscando o melhor balanço entre velo-

cidade de treinamento, precisão e estabilidade.

Tabela 1 – Hiperparâmetros para cada um dos modelos utilizados. Os valores correspondentes ao modelo Simeoni são para uma das duas idênticas redes que compõem a solução. Fonte: Elaborado pelo autor (2023).

Hiperparâmetros	Logística	Gompertz	Von Bertallanfy	Simeoni
Número de neurônios na camada escondida	50	40	30	50
Número de camadas escondidas	1			
Pontos de entrada (excluindo conjunto de dados)	500			
Otimizador	LBFGS			
Taxa de aprendizado	0.01	0.02	0.02	0.01
Função de ativação	sigmoide	tanh	sigmoide	sigmoide
Tamanho de *batch*	todo o conjunto de dados			
Número de épocas	80	80	80	100
Parâmetros treináveis	α, K, γ	a, β	a, b, γ	λ_0, λ_1, k_1, k_2,γ, b

A base para o código utilizado está no apêndice B. Ele conta com uma função para extrair o conjunto de dados de um arquivo em .csv e outra função que normaliza a entrada e a saída do conjunto de dados. O programa gera um vetor igualmente espaçado entre 0 e 1 e então incorpora dentro deste vetor a entrada normalizada do conjunto de dados. Ele lembra os índices que contém os valores referentes ao conjunto de dados para usar posteriormente na função de perda. A rede é criada usando a biblioteca *pytorch* como descrita no capítulo 1.3. As classes retornam a tentativa de solução e também a forma de sua EDO. Isto é enviado para a função de perda implementada como descrito no capítulo 2.3. Após certo número de épocas definido na Tabela 1, a solução é obtida e exibida em um gráfico.

3 Resultados e Discussão

Aqui apresentaremos os resultados e discussões dos experimentos descritos no Capítulo 2.4. Nosso objetivo é fazer a identificação de parâmetros e modelagem do crescimento dos tumores utilizando redes neurais.

Começaremos analisando a convergência da rede neural para os modelos apresentados. A Figura 9 exibe o comportamento da função de perda para todos os quadros modelos utilizando o conjunto de dados S6. O comportamento de todas elas decresce até próximo de zero e se estabiliza lá. Nos testes, concluiu-se que os retornos decrescentes em aumentar o número de épocas passam a não valer a pena a partir do final dos gráficos. É importante destacar que sempre haverá um erro residual aparentemente alto na função de perda, pois ela tenta diminuir duas métricas não-alinhadas diferentes simultaneamente. O resultado, entretanto, não é impactado por isto.

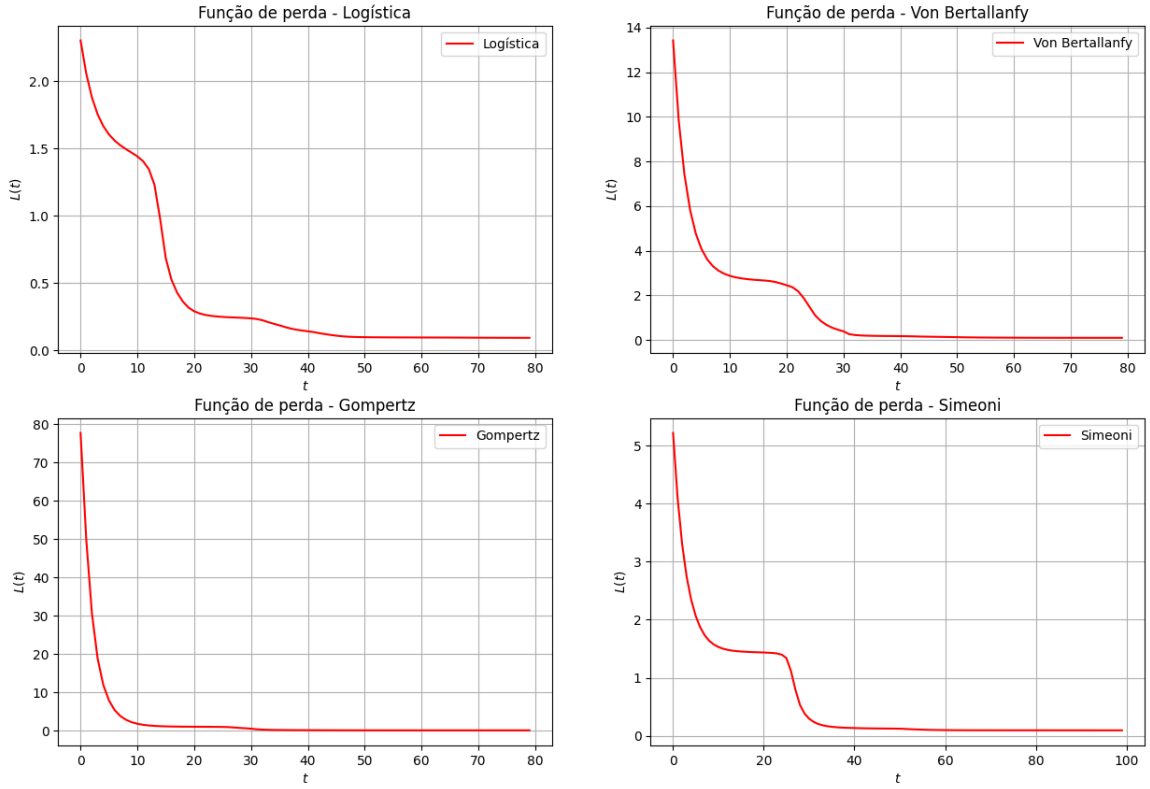


Figura 9 – Função de perda para todos os modelos trabalhados usando o conjunto de dados S6. O tempo é dado em épocas e L(t) representa a função de perda para cada um dos modelos. Fonte: Elaborado pelo autor (2023).

A velocidade do treinamento é bastante satisfatória. Com um processador Intel® Xeon®, o treino da rede neural usando computação em nuvem para o modelo Simeoni dura em média 40 segundos, com os outros modelos levando aproximadamente 25 segundos.

Redes mais complexas como na referência (ROJAS, 2020) podem levar 15 minutos para um treinamento em processadores. Isto se deve não apenas a simplicidade do código, mas também aos métodos utilizados nas referências (LAGARIS; LIKAS, 1998) e (DUA; DUA, 2011): A resolução simultânea permite que a rede utilize um único otimizador e a tentativa de solução acelera a convergência.

Em segundo lugar vamos analisar qualitativamente as soluções obtidas pelas redes neurais. As Figuras 10a e 10b comparam todas as soluções diferentes no conjunto de dados não tratado S12 e S6, respectivamente. Para isto, uma simulação de Monte Carlo de 100 iterações coletou a média e o desvio-padrão máximo do gráfico. A média dos pontos dos gráficos geraram as curvas individuais mostradas na figura e as curvas pontilhadas amarelas são o máximo desvio-padrão que cada reta obteve durante o experimento.

É possível perceber que na Figura 10a, os modelos logístico, Gompertz e Von Bertallanfy possuíram um comportamento bastante similar. O modelo Simeoni foi o menos afetado pelas discrepâncias de certos pontos do conjunto de dados, sendo o mais robusto nos testes realizados, ainda que semelhante aos demais. Todos os outros foram bastante influenciados pelos pontos mais discrepantes do conjunto de dados. A semelhança entre todos se torna mais clara na Figura 10b, onde a superposição neste caso praticamente igualou todas as funções. O conjunto de dados é em grande parte responsável por isto, visto que este parece se comportar de forma mais suave do que os outros.

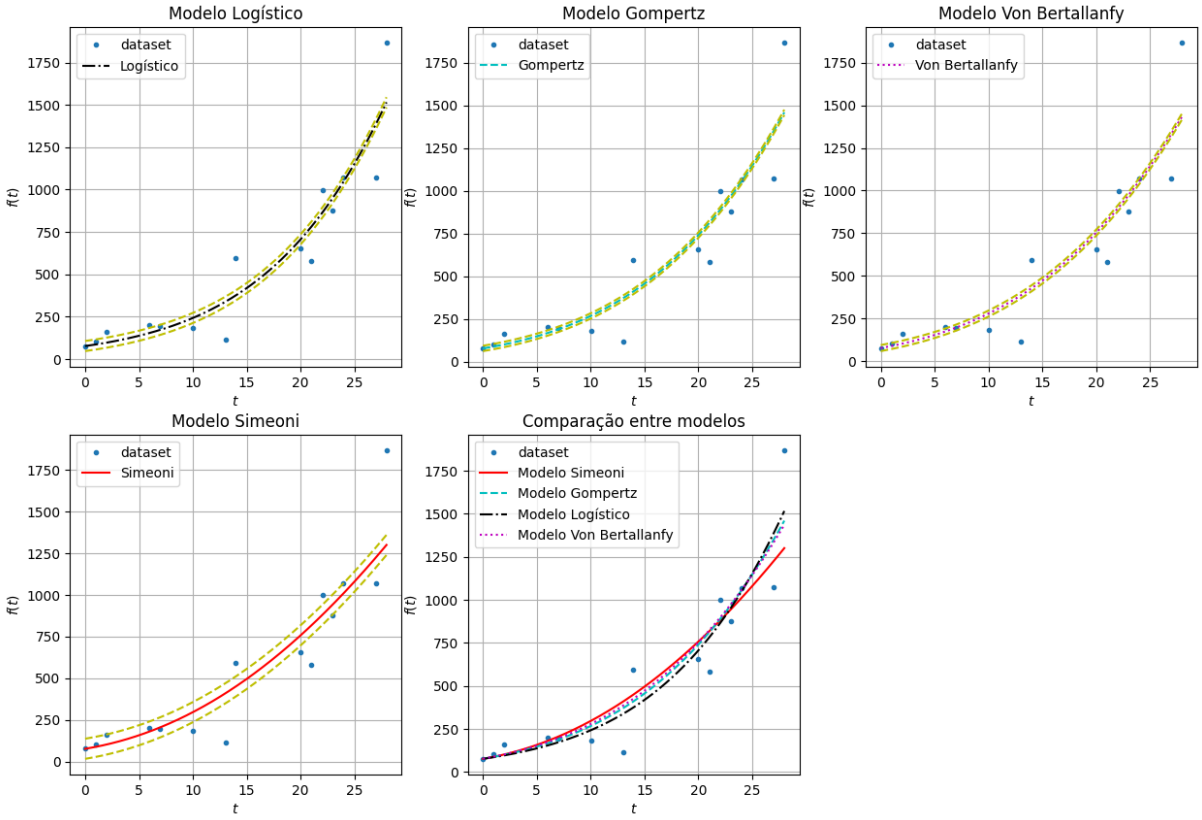
Alguns dos gráficos inclusive apresentaram seu desvio-padrão pequeno o suficiente para se tornar indistinguível no gráfico. Isso significa que a rede está sempre convergindo para o mesmo mínimo local, e essa consistência traz segurança a respeito do método.

As Figuras 11a e 11b comparam as soluções dos conjuntos tratados C9 e C6. A mesma simulação Monte Carlo foi realizada para a obtenção dos dados. Para reduzir o desvio-padrão, os parâmetros das equações tiveram uma inicialização fixa. As mesmas observações na imagem com os dados não-tratados também aparecem aqui: Todos os modelos conseguiram entregar resultados precisos e muito similares entre si.

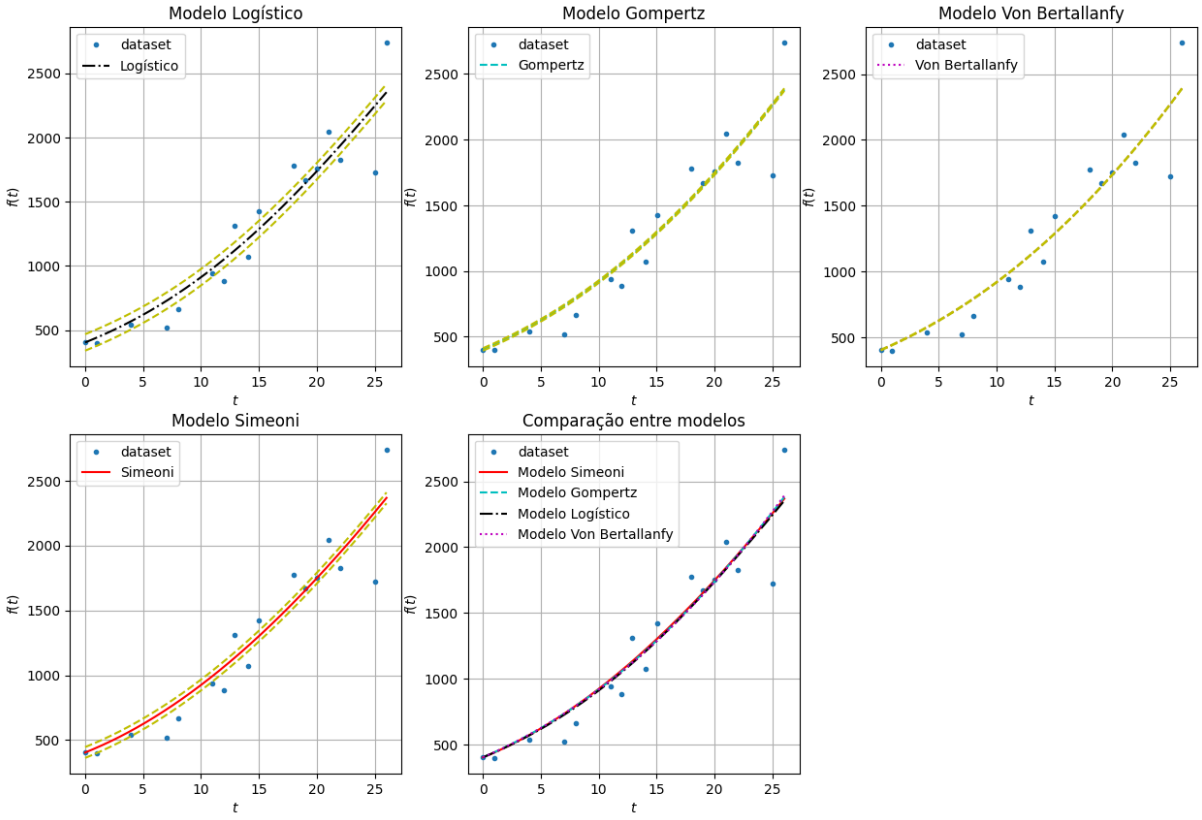
A comparação gráfica para os demais conjunto de dados está localizado no apêndice A e conta com a mesma análise dos apresentados agora. No geral, os resultados foram bastante similares entre si e o método foi satisfatório em todas as aplicações.

Os otimizadores em uma rede neural minimizam a função de perda buscando seus mínimos locais. Fixar as constantes ajuda o experimento a concentrar suas iterações no mesmo mínimo local e produzir resultados mais precisos. A observação da imagem novamente diz que os resultados são similares entre si. Por mais que o modelo Simeoni seja o único que possui comportamentos exponenciais e lineares na mesma função, é possível simular este efeito nos outros modelos com uma escolha adequada de parâmetros.

Para o experimento, a escolha de um modelo sobre o outro é irrelevante visto que

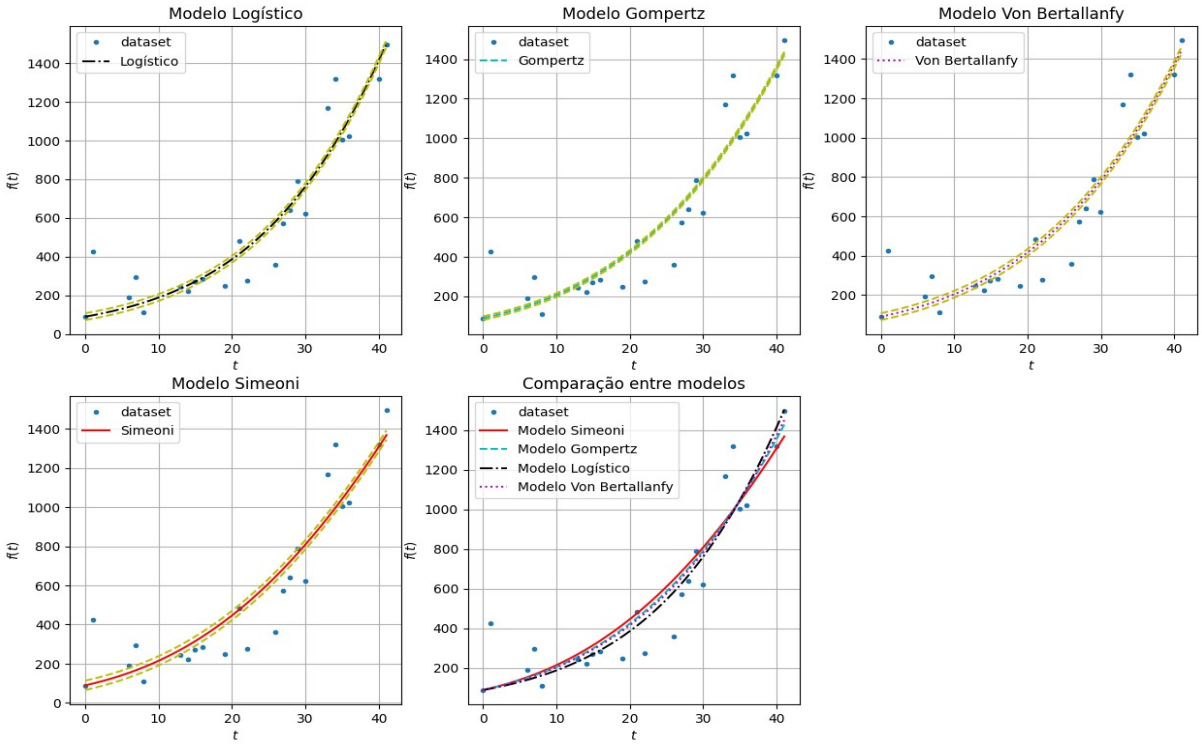


(a) Comparação para o conjunto S12

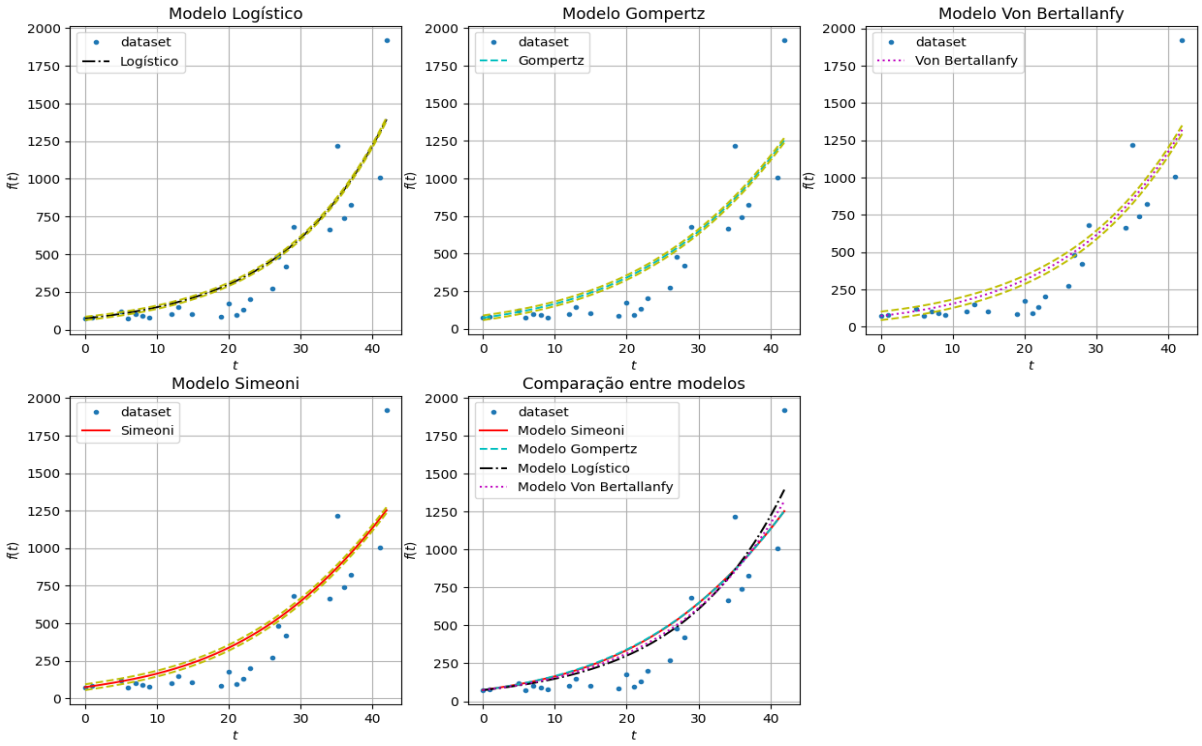


(b) Comparação para o conjunto S6

Figura 10 – Comparação entre os modelos logístico, Gompertz, Von Bertallanfy e Simeoni para os conjuntos de dados S12 e S6. Gráficos individuais representam a média de 100 iterações. As curvas pontilhadas amarelas são o máximo desvio-padrão obtido para cada modelo. O último gráfico mostra a comparação entre cada modelo no mesmo conjunto de dados. Fonte: Elaborado pelo autor (2023).



(a) Comparação para o conjunto C9



(b) Comparação para o conjunto C6

Figura 11 – Comparação entre os modelos logístico, Gompertz, Von Bertallanfy e Simeoni relativo ao conjunto C9 e C6. Cada gráfico individual representa a média de 100 iterações. As curvas pontilhadas amarelas são o máximo desvio-padrão obtido em cada modelo. O último gráfico mostra a comparação direta entre cada modelo no mesmo conjunto de dados C9. As constantes para o experimento foram: Logística: $\alpha = 2, K = \gamma = 1$. Bertallanfy: $a = 2, b = 0.6, \gamma = 1.1$. Gompertz: a = -0.5, b = 0.01. Simeoni: $\lambda_0 = 5, \lambda_1 = 4, k_1 = 1, k_2 = 2, \gamma = 0.7, b = 3$. Fonte: Elaborado pelo autor (2023).

o camundongo morre antes de ser alcançado um platô no crescimento do tumor. Isto significa que os modelos capturam apenas o momento de crescimento dos tumores, o que ainda é bastante útil para compreender sua taxa de crescimento e comportamento sobre a atuação dos fármacos. Qualquer um dos modelos, dado um bom ajuste de parâmetros, pode ser utilizado para isto.

O modelo Simeoni já conta com a ausência do platô no crescimento, diferente dos outros três. Para os conjuntos de dados utilizados, esta diferença é irrelevante, mas em certos cenários experimentais ela pode ser o que diferencia os modelos entre si. O mesmo comportamento foi observado também nas referências (KOZIOL; FALLS; SCHNITZER, 2020) e (SIMEONI et al., 2004).

A identificação de parâmetros resultou em um conjunto de valores para cada conjunto de dados. A Tabela 2 mostra os parâmetros normalizados (internamente calculados pela rede neural) e os parâmetros ajustados (parâmetros reais após o ajuste das constantes) da equação logística. As Tabelas 3, 4 e 5 mostram os parâmetros das equações de Gompertz, Von Bertallanfy e o modelo de Simeoni, respectivamente.

Tabela 2 – Parâmetros normalizados e ajustados para todos os conjuntos de dados da equação logística. Fonte: Elaborado pelo autor (2023).

Conjunto de dados	Parâmetros Normalizados			Parâmetros ajustados		
	$\tilde{\alpha}$	$\tilde{K}$	$\tilde{\gamma}$	α	K	γ
S6	2.333	1.292	1.365	0.0896	3544	1.365
S9	2.760	2.660	3.682	0.106	5525	3.682
S12	3.030	2.631	3.902	0.108	4921	3.902
S14	3.775	2.569	3.953	0.0897	5007	3.950
S18	3.188	2.417	3.730	0.063	5617	3.730
C3	7.489	1.462	0.259	0.416	2581	0.259
C6	2.958	2.302	4.175	0.070	4426	4.175
C7	2.812	4.313	4.155	0.070	8441	4.155
C9	2.944	1.648	2.625	0.071	2468	2.625
C10	2.582	2.793	4.521	0.0526	5976	4.521

As variações nos parâmetros entre conjuntos de dados se dá pela natureza dos mesmos. Certos conjuntos foram obtidos em em um intervalo de 25 dias enquanto outros chegaram a 60 dias. A rede precisa se ajustar a estas mudanças e por isso os valores podem se diferenciar bastante. O ajuste dos parâmetros seguiu as substituições:

Os parâmetros calculados para a equação de Gompertz encontram-se na Tabela 3. Seus resultados foram satisfatórios e convergiram bem para a maior parte dos conjuntos de dados.

Os parâmetros ajustados foram obtidos através das substituições nas equações 3.1, 3.2, 3.3, e 3.4 para a equação logística, Gompertz, Bon Bertallanfy e o modelo Simeoni,

Tabela 3 – Parâmetros normalizados e ajustados para todos os conjuntos de dados da equação de Gompertz. Fonte: Elaborado pelo autor (2023).

Conjuntos de dados	Parâmetros normalizados		Parâmetros ajustados	
	$\tilde{a}$	$\tilde{\beta}$	a	β
S6	-0.642	0.159	-0.0246	5.808×10^{-5}
S9	-0.491	0.0160	-0.0189	7.720×10^{-6}
S12	-0.398	0.00269	-0.0142	1.442×10^{-6}
S14	-0.924	0.0993	-0.0219	5.099×10^{-5}
S18	-0.606	0.0258	-0.0121	1.113×10^{-5}
C3	-1.513	0.621	-0.0841	3.524×10^{-4}
C6	-0.319	4.170×10^{-4}	-7.618×10^{-3}	2.618×10^{-7}
C7	-0.313	7.873×10^{-4}	-7.833×10^{-3}	4.023×10^{-7}
C9	-0.429	0.00524	-0.0104	3.504×10^{-6}
C10	-0.217	4.783×10^{-5}	-4.432×10^{-3}	2.236×10^{-8}

Tabela 4 – Parâmetros normalizados e ajustados para todos os conjuntos de dados da equação de Von Bertallanfy. Fonte: Elaborado pelo autor (2023).

Conjuntos de dados	Parâmetros normalizados			Parâmetros ajustados		
	$\tilde{a}$	$\tilde{b}$	$\tilde{\gamma}$	a	b	γ
S6	1.816	0.576	0.730	0.589	0.0221	0.730
S9	3.285	1.226	0.874	0.329	0.0472	0.874
S12	4.428	2.441	0.879	0.393	0.0873	0.879
S14	1.752	0.0787	0.577	1.021	0.00187	5.776
S18	2.112	0.210	0.715	0.384	0.00420	0.715
C3	1.368	0.426	0.385	7.516	0.0237	0.385
C6	6.136	3.765	0.955	0.204	0.0897	0.955
C7	7.019	5.120	0.930	0.296	0.128	0.930
C9	5.084	2.908	0.914	0.232	0.0708	0.914
C10	2.027	0.236	0.843	0.137	0.00481	0.843

respectivamente.

$$\begin{cases} \alpha = \frac{\tilde{\alpha}}{t_{\max}} \\ K = \tilde{K} y_{\max} \end{cases} \tag{3.1}$$

$$\begin{cases} a = \frac{\tilde{a}}{t_{\max}} \\ \beta = \tilde{\beta} y_{\max} \end{cases} \tag{3.2}$$

$$\begin{cases} a = \frac{\tilde{a}}{t_{\max} {y_{\max}}^{\gamma-1}} \\ b = \frac{\tilde{b}}{y_{\max}} \end{cases} \tag{3.3}$$

Tabela 5 – Parâmetros normalizados e ajustados para todos os conjuntos de dados da equação de Simeoni. Fonte: Elaborado pelo autor (2023).

Conjuntos de dados	Parâmetros normalizados					
	$\tilde{\lambda_0}$	$\tilde{\lambda_1}$	$\tilde{k_1}$	$\tilde{k_2}$	$\tilde{\gamma}$	$\tilde{b}$
S6	3.241	1.245	0.439	2.271	1.177	0
S9	2.638	2.114	0.698	1.378	3.284	0
S12	3.718	3.437	0.878	0.562	0.987	0
S14	6.658	1.108	0.825	6.042	1.323	0
S18	7.289	1.672	0.620	1.011	0.803	0
C3	5.174	1.493	0.197	0.775	0.990	1.00
C6	3.280	3.950	0.339	0.124	1.418	0.454
C7	4.160	2.882	0.103	1.449	0.798	1.141
C9	5.354	4.145	1.296	2.550	0.746	2.872
C10	2.678	1.689	0.228	0.745	3.257	2.267
Conjuntos de dados	Parâmetros ajustados					
	λ_0	λ_1	k_1	k_2	γ	b
S6	0.124	131	0.0169	0.0872	1.177	0
S9	0.101	169	0.0269	0.0531	3.284	0
S12	0.132	229	0.0314	0.0200	0.987	0
S14	0.158	51.3	0.0196	0.143	1.323	0
S18	0.145	77.7	0.0124	0.0202	0.803	0
C3	0.287	146	0.0109	0.0431	0.990	0.0559
C6	0.0782	181	8.093×10^{-3}	2.97×10^{-3}	1.418	0.0108
C7	0.104	141	2.591×10^{-3}	0.0362	0.798	0.0285
C9	0.130	151	0.0315	0.0621	0.746	0.0699
C10	0.0545	73.6	4.649×10^{-3}	0.0152	3.257	0.0462

$$\begin{cases} \lambda_0 = \frac{\tilde{\lambda_0}}{t_{\max}} \\ \lambda_1 = \frac{\tilde{\lambda_1}}{t_{\max}} V_{\max} \\ k_1 = \frac{k_1}{t_{\max}} \\ k_2 = \frac{k_2}{t_{\max}} \\ b = \frac{\tilde{b}}{t_{\max}} \end{cases} \tag{3.4}$$

O conjunto C10 apresentou a maior quantidade de ruído e a rede teve dificuldade de convergência e precisão durante os primeiros testes. Mais uma vez, as inicializações aleatórias dos pesos das camadas e neurônios podem gerar perturbações graves durante os treinos. Infelizmente isto é inerente à solução utilizada, visto que redes neurais precisam de uma inicialização não-uniforme de seus pesos internos para a melhor convergência. Os outros conjuntos de dados, mesmo que menos ruidosos, são bem diferentes entre si e, por isso, exigem constantes que variam bastante entre cada conjunto. Os parâmetros ajustados ainda assim funcionam bem.

A atual utilização de redes neurais fisicamente informadas (que abrange o método

utilizado neste trabalho) ainda precisa de estudos para superar alguns entraves intrínsecos em sua utilização. Dependendo da configuração utilizada para o conjunto de dados, os sistemas internos de otimização da rede podem falhar em convergir de forma apropriada (KRISHNAPRIYAN et al., 2021). Esse caso não aconteceu neste trabalho, mas pode impactar trabalhos futuros.

Qualitativamente, os resultados se assemelham com os da referência (KOZIOL; FALLS; SCHNITZER, 2020), visto que os modelos entregaram resultados similares tanto em experimentos com tratamento ou sem. O modelo Simeoni, mesmo conseguindo alternar entre crescimento exponencial e linear rapidamente, consegue ser simulado pelos outros modelos dependendo das constantes utilizadas.

O método de solução é rápido e obtém resultados satisfatórios tanto para modelos escalares quanto para sistemas. A normalização dos dados se tornou imprescindível aqui. As constantes ajustadas, quando aplicadas à solução analítica dos três modelos clássicos, perfeitamente descrevem as curvas obtidas pela rede neural. No modelo Simeoni, por não possuir solução analítica, as constantes foram validadas por uma outra rede neural que apenas solucionava a EDO com os parâmetros fixos encontrados. O resultado foi idêntico à resolução simultânea das constantes, confirmando assim a precisão das mesmas.

4 Conclusão

Este trabalho apresentou uma forma de solução de EDOs simultaneamente à identificação de parâmetros para diferentes modelos. Os resultados apontam que os modelos são bastante similares entre si, o que abre a oportunidade de escolhê-los sob outros critérios que não sejam a precisão. Por exemplo, a equação logística generalizada foi a que possuiu a menor rede. Em um cenário de complexidade computacional limitada, ela poderia ser uma boa escolha.

Assim como o a referência (KOZIOL; FALLS; SCHNITZER, 2020) sugere, as diferenças intrínsecas que os modelos possuem podem ser superadas com a escolha certa de constantes, e a rede neural as encontra em poucas épocas. O conjunto de dados ruidoso não impacta tanto nos resultados quanto esperado.

A escolha dentre os modelos também é irrelevante dentro do escopo de estudo porque o crescimento do tumor nunca chega a um platô nos experimentos realizados, algo também observado nos experimentos nas referências (SIMEONI et al., 2004) e (KOZIOL; FALLS; SCHNITZER, 2020). O momento do crescimento, então, é a principal característica capturada pelos modelos. O modelo Simeoni já prevê a ausência do platô, o que pode diferenciá-lo dos outros modelos em certas configurações experimentais caso seja permitido que o tumor alcance o platô de seu crescimento.

As técnicas aplicadas entregaram resultados bastante satisfatórios em questão de precisão e velocidade computacional, mesmo se comparado à soluções analíticas. A realização simultânea não prejudicou a precisão dos resultados e agilizou o processo, tornando mais viável a execução do método de Monte Carlo para confirmar a estabilidade da rede.

Após a reestruturação do código e o uso da normalização, os objetivos buscados no trabalho anterior foram alcançados. A normalização permite estender o intervalo ótimo de uma rede neural arbitrariamente e o código refeito é estável e tende a encontrar constantes similares em experimentos repetidos.

As constantes obtidas foram verificadas e, aplicada à solução analítica de cada um dos modelos, condizem perfeitamente com os resultados obtidos. O modelo Simeoni também foi verificado por outra rede neural própria para tal e condiz com os resultados.

Se comparado a redes como (ROJAS, 2020), a rede e métodos utilizados são mais simples sem perder precisão, possui uma menor complexidade computacional e utilizam um menor número de épocas. É possível dizer que a resolução de equações diferenciais por redes neurais é bastante versátil, pois o consumo de recursos computacionais depende da complexidade da equação a ser resolvida. Esta plasticidade é uma boa vantagem desta

forma de solução. Entretanto, ainda há muitas margens para refinar o processo e os métodos de solução envolvendo redes neurais, principalmente quando as equações são mais complexas.

As possibilidades de se trabalhar com a rede neural abrem espaço para a generalização de diversos problemas. Apesar de haver outras formas de soluções tanto para a solução de EDOs quanto para identificação de parâmetros, uma rede neural permite solucioná-las simultaneamente ao mesmo tempo que mantém o mesmo algoritmo para todos os casos deste problema.

Trabalhos futuros acerca deste tema podem envolver testes com o modelo Simeoni na forma de sistemas de 3 ou mais EDOs, novos conjuntos de dados que sejam mais ou menos ruidosos, aplicação das técnicas apresentadas aqui em outra área da biomédica, como a epidemiologia e suas equações diferenciais relacionadas à proliferação de doenças e também o refino da técnica de modo a contornar os problemas intrínsecos das redes neurais demonstrados neste trabalho.

Referências

ANTONIADIS, P. *Hidden Layers in a Neural Network*. 2022. Disponível em: <https://www.baeldung.com/cs/hidden-layers-neural-network>. Citado 2 vezes nas páginas 8 e 19.

BAJZER, Z.; MARUŠIĆ, M.; VUK-PAVLOVIĆ, S. Conceptual frameworks for mathematical modeling of tumor growth dynamics. Elsevier Science Ltd, 1996. Citado 3 vezes nas páginas 14, 27 e 30.

BENZEKRY, S. et al. Classical mathematical models for description and prediction of experimental tumor growth. PLoS Computational Biology, 2014. Citado 4 vezes nas páginas 14, 27, 29 e 30.

CARNIELLO, R.; VITAL, W.; VALLE, M. Universal approximation theorem for tessarine-valued neural networks. In: *Anais do XVIII Encontro Nacional de Inteligência Artificial e Computacional*. Porto Alegre, RS, Brasil: SBC, 2021. p. 233–243. ISSN 2763-9061. Disponível em: <https://sol.sbc.org.br/index.php/eniac/article/view/18256>. Citado 3 vezes nas páginas 19, 21 e 34.

CHOLLET, F. *Deep Learning with Python*. 1. ed. [S.l.]: Manning Publications Company, 2018. Citado 6 vezes nas páginas 13, 16, 17, 18, 20 e 28.

CYBENKO, G. V. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, v. 2, p. 303–314, 1989. Citado na página 14.

DUA, P.; DUA, V. A simultaneous approach for parameter estimation of a system of ordinary differential equations, using artificial neural network approximation. 2011. Citado 2 vezes nas páginas 14 e 39.

DUA, V. An artificial neural network approximation based decomposition approach for parameter estimation of system of ordinary differential equations. v. 35, p. 545–553, 2011. Citado na página 14.

DUFERA, T. T. Deep neural network for system of ordinary differential equations: Vectorized algorithm and simulation. Elsevier, 2021. Citado na página 13.

FRICKE, K. et al. *Python Implementation of Ordinary Differential Equations Solvers using Hybrid Physics-informed Neural Networks*. Zenodo, 2020. Disponível em: <https://github.com/PML-UCF/pinn_ode_tutorial>. Citado na página 14.

GIOVANNI, M. D. et al. Finding multiple solutions of odes with neural networks. 2020. Citado na página 14.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. Citado 2 vezes nas páginas 18 e 19.

HARRIS, C. R. et al. Array programming with NumPy. *Nature*, Springer Science and Business Media LLC, v. 585, n. 7825, p. 357–362, set. 2020. Disponível em: <https://doi.org/10.1038/s41586-020-2649-2>. Citado na página 33.

HUNTER, J. D. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, IEEE COMPUTER SOC, v. 9, n. 3, p. 90–95, 2007. Citado na página 33.

JAMILI, E.; DUA, V. Parameter estimation of partial differential equations using artificial neural network. v. 147, 2021. Citado na página 14.

KLUYVER, T. et al. Jupyter notebooks – a publishing format for reproducible computational workflows. In: LOIZIDES, F.; SCHMIDT, B. (Ed.). *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. [S.l.], 2016. p. 87 – 90. Citado na página 33.

KOZIOL, J.; FALLS, T.; SCHNITZER, J. E. Different ode models of tumor growth can deliver similar results. BMC Cancer, 2020. Citado 12 vezes nas páginas 8, 14, 15, 26, 27, 29, 31, 34, 35, 42, 45 e 46.

KRISHNAPRIYAN, A. S. et al. Characterizing possible failure modes in physics-informed neural networks. In: . [S.l.]: 35th Conference on Neural Information Processing Systems, 2021. Citado na página 45.

LAGARIS, I. E.; LIKAS, A. Artificial neural networks for solving ordinary and partial differential equations. IEEE TRANSACTIONS ON NEURAL NETWORKS, 1998. Citado 7 vezes nas páginas 13, 14, 21, 23, 34, 35 e 39.

LEVINE, Z. E. andSydney; RAND, D. G.; RAHWAN, I. *Who Gets Credit for AI-Generated Art?* Cell Press, 2020. Disponível em: <https://www.nbcnews.com/tech/internet/lensa-ai-artist-controversy-ethics-privacy-rcna60242>. Citado na página 13.

MARUSIC, M.; BAJZER, Z. Generalized two-parameter equation of growth. Journal of Mathematical Analysis and Applications, 1993. Citado 2 vezes nas páginas 14 e 27.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, 1943. Citado na página 13.

MCKINNEY, W. et al. Data structures for statistical computing in python. In: AUSTIN, TX. *Proceedings of the 9th Python in Science Conference*. [S.l.], 2010. v. 445, p. 51–56. Citado na página 34.

MEHRKANOON, S.; FALCK, T.; SUYKENS, J. A. Parameter estimation for time varying dynamical systems using least squares support vector machines. v. 45, 2012. Citado na página 14.

MURPHY, H.; JAAFARI, H.; DOBROVOLNY, H. Differences in predictions of ode models of tumor growth: a cautionary example. BMC Cancer, 2016. Citado 2 vezes nas páginas 14 e 31.

PASZKE, A. et al. Pytorch: An imperative style, high-performance deep learning library. In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019. Disponível em: <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>. Citado 3 vezes nas páginas 21, 24 e 33.

PHAM, B. et al. *Solve Systems of Ordinary Differential Equations Using Deep Neural Networks*. 2020. 42-47 p. Citado na página 14.

ROJAS, C. J. G. *Identificação de parâmetros de um modelo de dano usando uma rede neural informada por leis físicas*. Dissertação (Mestrado) — UNIVERSIDADE ESTADUAL DE CAMPINAS, 2020. Citado 4 vezes nas páginas 14, 18, 39 e 46.

SIMEONI, M. et al. Predictive pharmacokinetic-pharmacodynamic modeling of tumor growth kinetics in xenograft models after administration of anticancer agents. American Association for Cancer Research, 2004. Citado 4 vezes nas páginas 14, 31, 42 e 46.

STEVENS, E.; ANTIGA, L.; VIEHMANN, T. *Deep Learning with PyTorch*. [S.l.]: Manning Publications Co, 2020. ISBN 9781617295263. Citado 7 vezes nas páginas 8, 13, 17, 18, 19, 20 e 21.

TATE, S. et al. Semi-mechanistic pharmacokinetic/pharmacodynamic modeling of the antitumor activity of ly2835219, a new cyclin-dependent kinase 4/6 inhibitor, in mice bearing human tumor xenografts. Clinical Cancer Research, 2014. Citado 3 vezes nas páginas 14, 27 e 31.

WEN, Y.; CHAOLU, T.; WANG, X. Solving the initial value problem of ordinary differential equations by lie group based neural network method. PLoS ONE, 2022. Citado na página 14.

APÊNDICE A – Comparação gráfica entre modelos

A.1 Conjunto de dados S9

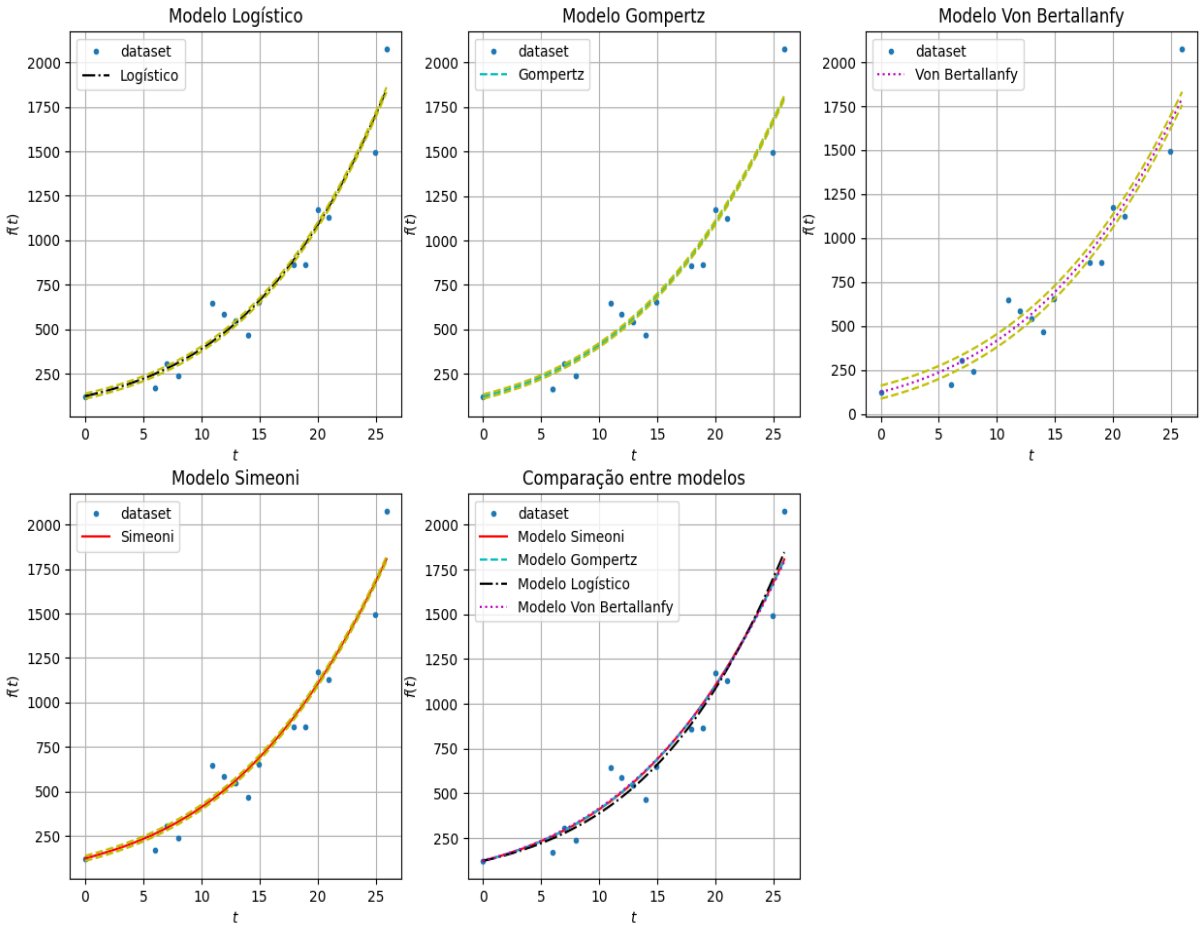


Figura 12 – Comparação gráfica da soluções dos modelos para o conjunto S9. Os parâmetros foram iniciados com base em uma distribuição uniforme entre [0,1]. Fonte: Elaborado pelo autor (2023).

A.2 Conjunto de dados S14

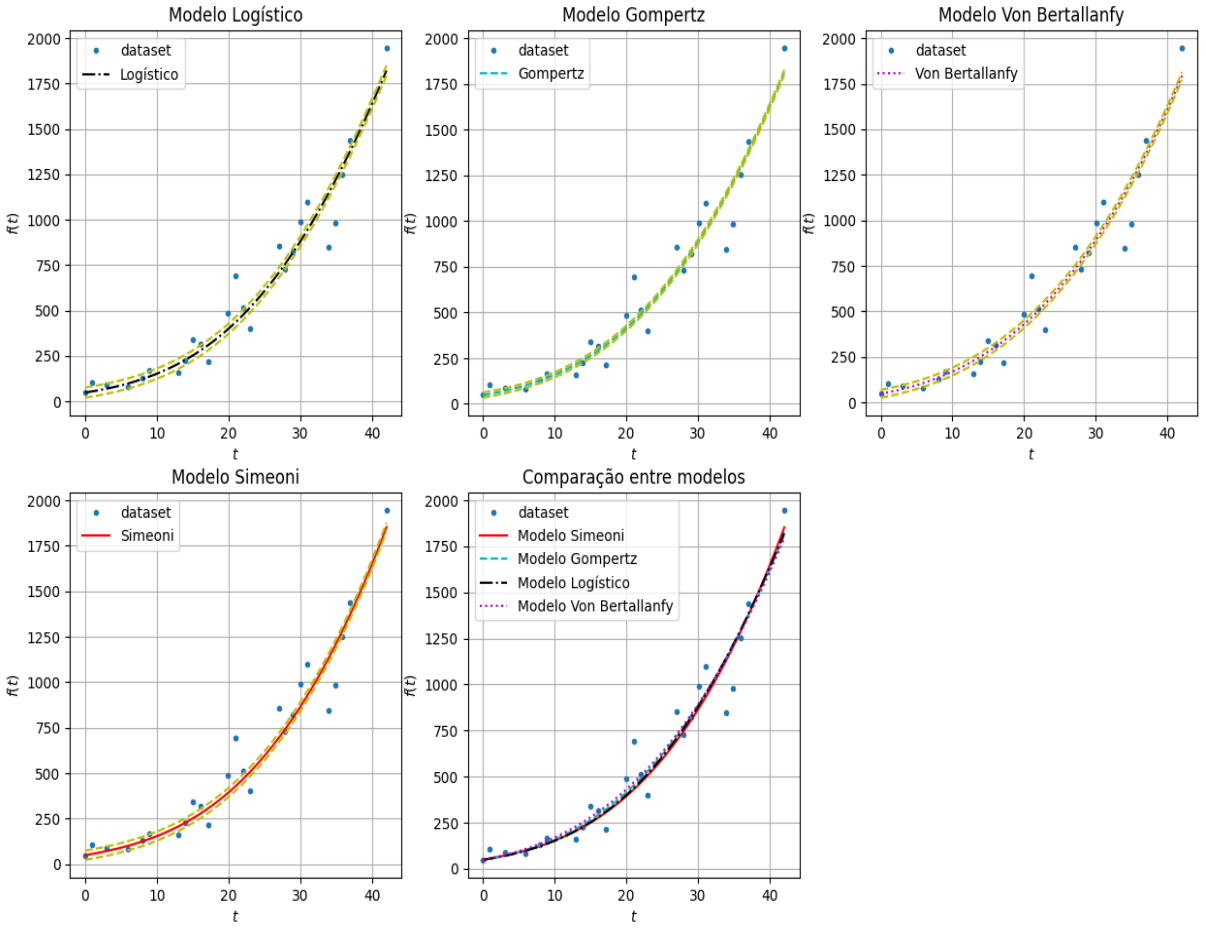


Figura 13 – Comparação gráfica da soluções dos modelos para o conjunto S14. Os parâmetros foram iniciados com base em uma distribuição uniforme entre [0,1]. Fonte: Elaborado pelo autor (2023).

A.3 Conjunto de dados S18

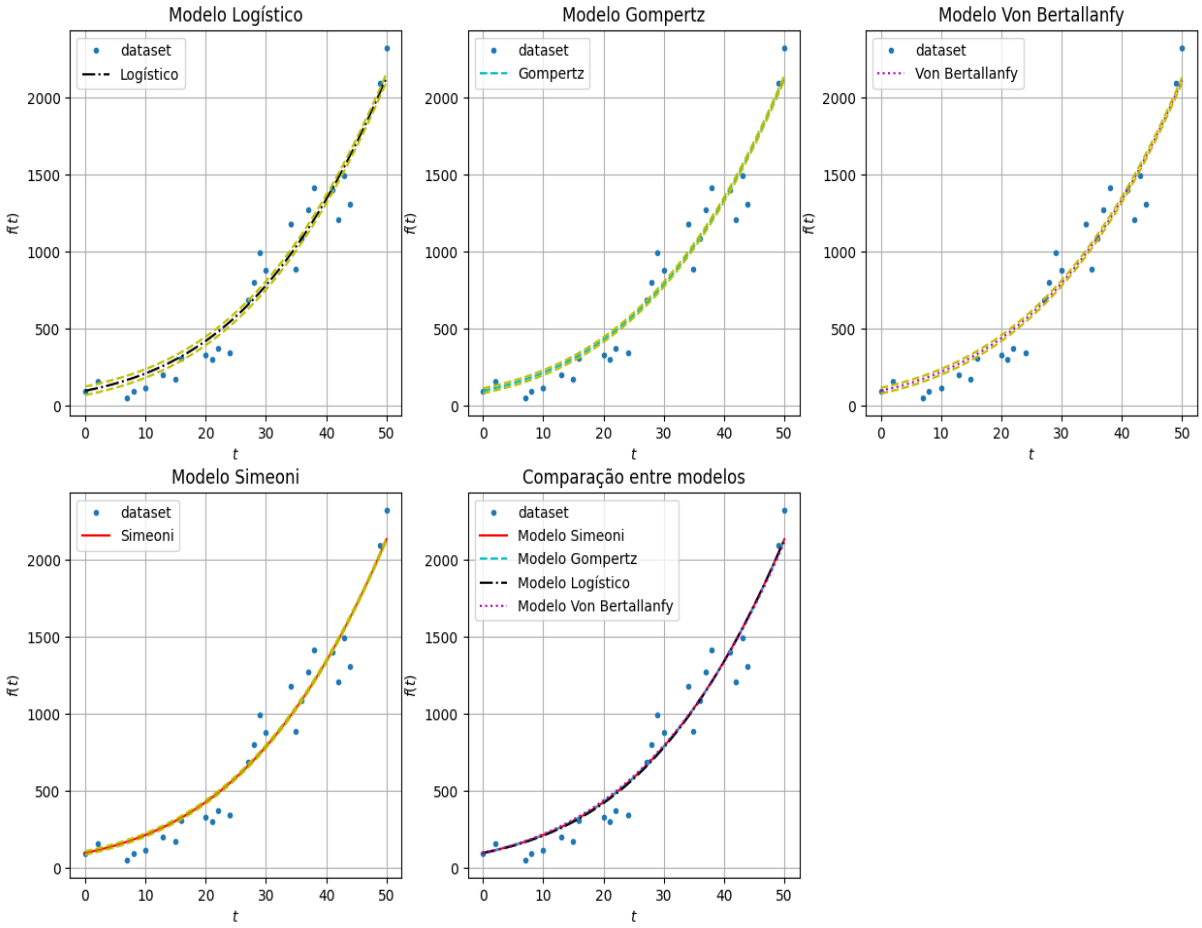


Figura 14 – Comparação gráfica da soluções dos modelos para o conjunto S18. Os parâmetros foram iniciados com base em uma distribuição uniforme entre [0,1]. Fonte: Elaborado pelo autor (2023).

A.4 Conjunto de dados C3

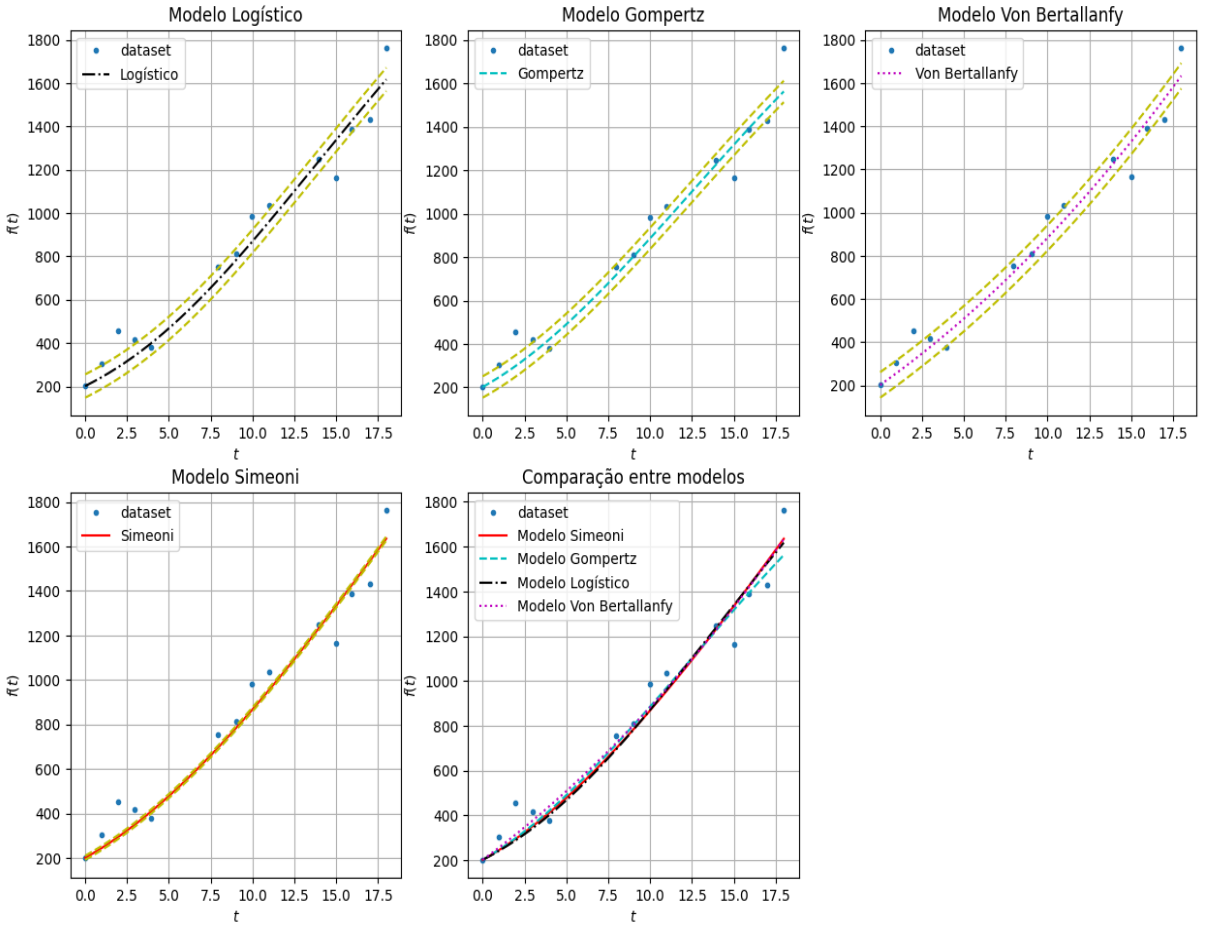


Figura 15 – Comparação gráfica da soluções dos modelos para o conjunto C3. Os parâmetros foram iniciados com base em uma distribuição uniforme entre [0,1]. Fonte: Elaborado pelo autor (2023).

A.5 Conjunto de dados C7

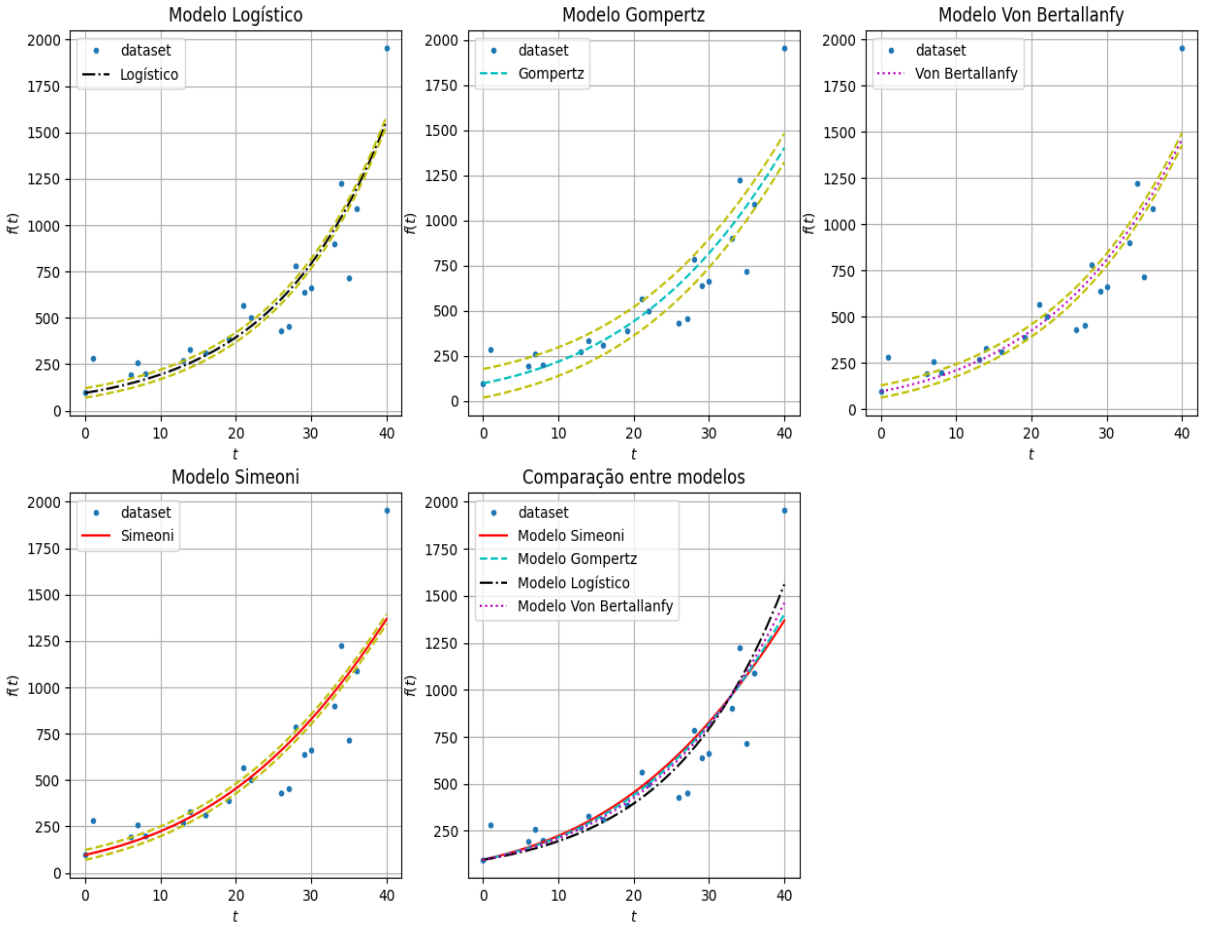


Figura 16 – Comparação gráfica da soluções dos modelos para o conjunto C7. Os parâmetros foram iniciados com base em uma distribuição uniforme entre [0,1]. Fonte: Elaborado pelo autor (2023).

A.6 Conjunto de dados C10

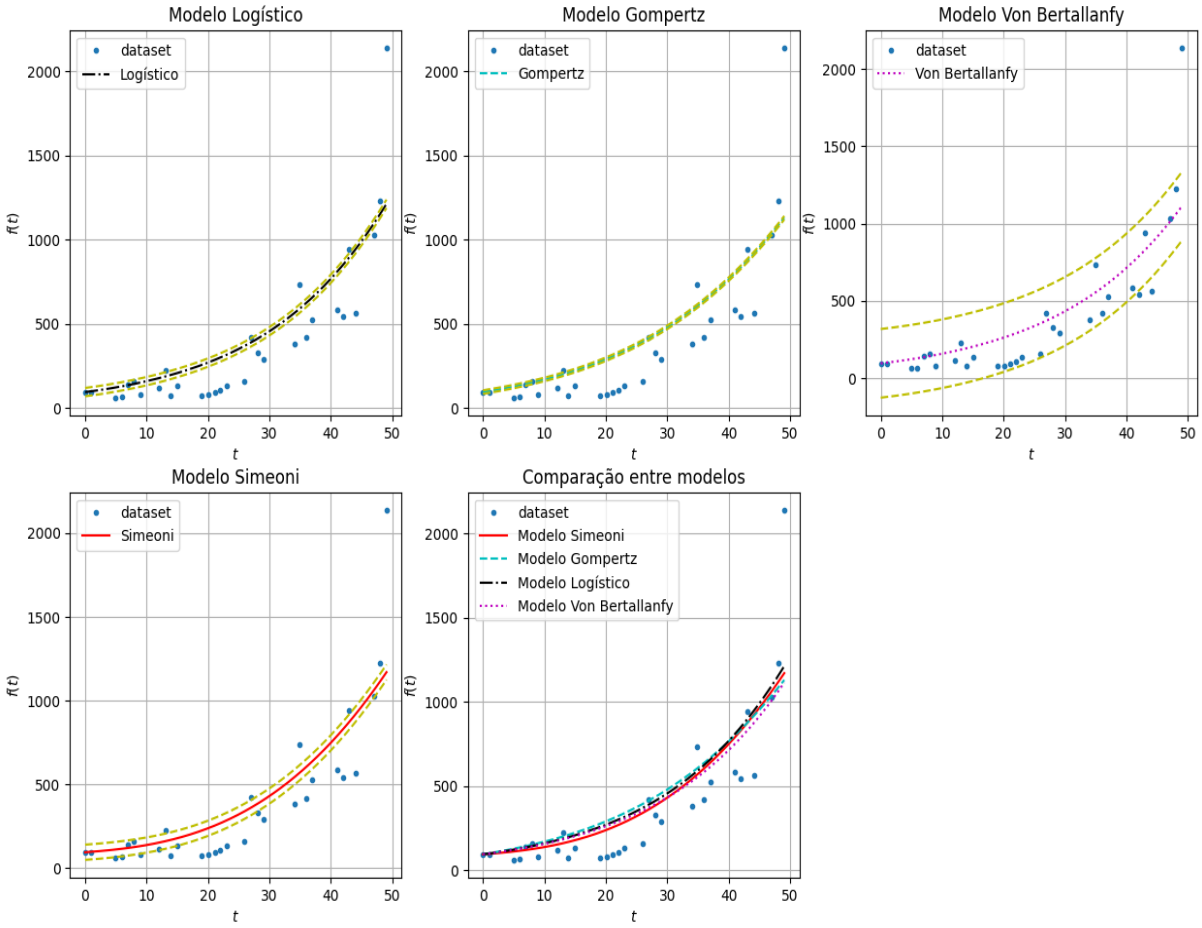


Figura 17 – Comparação gráfica da soluções dos modelos para o conjunto C10. Os parâmetros foram iniciados com base em uma distribuição uniforme entre [0,1]. Fonte: Elaborado pelo autor (2023).

APÊNDICE B – Código solucionador de EDOs

```
import torch
import torch.nn as nn
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
csv = 'Dataset 0_SD_gf6.csv'  #Conjunto de dados a importar
def extrair_dataset(csv):  #funcao para extrair o conjunto de dados
    df = pd.read_csv(csv,header=None)
    df = df.to_numpy()
    x = df[:,0] #Separa o conjunto em valores de x e y
    y=df[:,1]
    return x,y
def normalizacao(x,y,xt,norm=True): #funcao para normalizar o conjunto
#de dados e coloca-lo em xt
    max_x = max(x) #armazena os maximos de x e y
    max_y=max(y)
    if norm:        #normaliza pelos maximos de x e y
      x=x/max_x
      y=y/max_y
    xt = np.unique(np.sort(np.append(xt,x)))  #Insere os valores
    #normalizados em xt
    return x,y,xt,max_x,max_y

def solucionar_edo(modo):
  while True:  #Elimina qualquer falha na inicializacao
    x_dataset,y_dataset = extrair_dataset(csv)
    xt = np.linspace(0,1,500)  #Espaco de treinamento dos pontos da EDO
    x_norm,y_norm,xt,max_x,max_y = normalizacao(x_dataset,y_dataset,xt)
    sorter = np.argsort(xt)
    indices_norm=sorter[np.searchsorted(xt, x_norm, sorter=sorter)]
    #procurando os indices correspondentes ao dataset dentro de
    #xt continuo
    x=torch.Tensor(xt[np.newaxis,:].T) #xt em formato apropriado
    #para treinamento

    A = y_norm[0] #valor inicial Z_1(0)
    if modo == "logistica":
      class NeuralNetwork(nn.Module):    #definicao das classes para
      #cada modelo
```

```
41          def __init__(self):
42              super().__init__()
43              self.N = nn.Sequential(nn.Linear(1, 30), nn.Sigmoid(), nn.
    Linear(30,1, bias=False)) #30 neuronios, 3 parametros
44              self.a = nn.Parameter(data=torch.tensor(np.random.rand()),
    requires_grad=True)
45              self.k = nn.Parameter(data=torch.tensor(np.random.rand()),
    requires_grad=True)
46              self.v = nn.Parameter(data=torch.tensor(np.random.rand()),
    requires_grad=True)
47
48          def forward(self, x):
49              modelo = A + x * self.N(x) #tentativa de solucao
50              return modelo, self.a * modelo * (1 - (modelo/self.k)**self.
    v) #retorna a tentativa de solucao e a EDO
51      elif modo == "bertallanfy":
52        class NeuralNetwork(nn.Module):
53          def __init__(self):
54            super().__init__()
55            self.N = nn.Sequential(nn.Linear(1, 50), nn.Sigmoid(), nn.
    Linear(50,1, bias=False) ) # 50 neuronios, 3 parametros
56            self.a = nn.Parameter(data=torch.tensor(np.random.rand()),
    requires_grad=True)
57            self.b = nn.Parameter(data=torch.tensor(np.random.rand()),
    requires_grad=True)
58            self.v = nn.Parameter(data=torch.tensor(np.random.rand()),
    requires_grad=True)
59          def forward(self, x):
60            modelo = A + x * self.N(x) #tentativa de solucao
61            return modelo, self.a * modelo ** self.v - self.b * modelo #
    retorna a tentativa de solucao e a EDO
62
63      elif modo=="gompertz":
64          class NeuralNetwork(nn.Module):
65            def __init__(self):
66              super().__init__()
67              self.N = nn.Sequential(nn.Linear(1, 40), nn.Tanh(), nn.
    Linear(40,1, bias=False)) # 40 neuronios, 2 parametros
68              self.a = nn.Parameter(data=torch.tensor(- np.random.rand()),
     requires_grad=True)
69              self.b = nn.Parameter(data=torch.tensor(np.random.rand()),
    requires_grad=True)
70
71            def forward(self, x):
72              modelo = A + x * self.N(x) #tentativa de solucao
73              return modelo, self.a * modelo * torch.log(self.b * modelo)
    #retorna a tentativa de solucao e a EDO
```

```
74
75
76      elif modo=="simeoni":
77        class NeuralNetwork(nn.Module):
78          def __init__(self):
79              super().__init__()
80              self.N = nn.Sequential(nn.Linear(1, 50), nn.Sigmoid(), nn.
    Linear(50,1, bias=False)) #50 neuronios
81              self.N2 = nn.Sequential(nn.Linear(1, 50), nn.Sigmoid(), nn.
    Linear(50,1, bias=False)) #50 neuronios
82              self.lambda_0 = nn.Parameter(data=torch.tensor(np.random.
    rand()), requires_grad=True) #6 parametros
83              self.lambda_1 = nn.Parameter(data=torch.tensor(np.random.
    rand()), requires_grad=True)
84              self.k1 = nn.Parameter(data=torch.tensor(np.random.rand()),
    requires_grad=True)
85              self.k2 = nn.Parameter(data=torch.tensor(np.random.rand()),
    requires_grad=True)
86              self.v = nn.Parameter(data=torch.tensor(np.random.rand()),
    requires_grad=True)
87              self.b = nn.Parameter(data=torch.tensor(np.random.rand()),
    requires_grad=True)
88          def forward(self, x):
89              modelo = A + x * self.N(x) #tentativa de solucao da primeira
     EDO
90              modelo2 = 0 + x * self.N2(x) #tentativa de solucao da
    segunda EDO
91              tgf = self.lambda_0*modelo/((1 + (self.lambda_0/self.
    lambda_1 * (modelo + modelo2))**self.v )**(1/self.v))
92              #retorna as tentativas de solucao e as EDOs
93              return modelo,modelo2, tgf - torch.exp(-self.b*x)*self.k1*
    modelo ,  torch.exp(-self.b*x)*self.k1*modelo - self.k2*modelo2
94      if modo == "simeoni":   #funcoes de perda
95        def loss(x):
96          x.requires_grad = True  #permite o grad para o calculo
97          #automatico dos gradientes
98          outputs1 = model(x)[0]
99          outputs2 = model(x)[1]
100         Psi_t_x = torch.autograd.grad(outputs1, x, grad_outputs=torch.
    ones_like(outputs1),
101                                       create_graph=True)[0]      #calculo da
102                                       #derivada de Z1
103         Psi_t_x_2 = torch.autograd.grad(outputs2, x, grad_outputs=torch.
    ones_like(outputs2),
104                                       create_graph=True)[0]      #calculo da
105                                       #derivada de Z2
106         loss1 = (torch.sum( ( Psi_t_x - model(x)[2] )  ** 2))
```

```
107         loss2 = (torch.sum( ( Psi_t_x_2 - model(x)[3] )  ** 2))
108         loss3 = ( torch.sum( (model(x)[0][indices_norm] + model(x)[1][
    indices_norm] - torch.Tensor(y_norm[np.newaxis].T) )**2) )
109         return loss1 +  loss2 +  loss3 #funcao de perda
110       else:
111         def loss(x):
112           x.requires_grad = True
113           outputs = model(x)[0]
114           Psi_t_x = torch.autograd.grad(outputs, x, grad_outputs=torch.
    ones_like(outputs),
115                                         create_graph=True)[0] #calculo da derivada
116           #funcao de perda
117           return  (torch.sum( ( Psi_t_x - model(x)[1] )  ** 2)) + torch.
    sum( ( model(x)[0][indices_norm] - torch.Tensor(y_norm[np.newaxis].T)
    )**2)
118       def closure():
119         optimizer.zero_grad() #grad zerado para nao gerar acumulacoes
120         #erroneas
121         l = loss(x)
122         l.backward()
123         return l
124       model = NeuralNetwork()  #declara a rede neural
125       optimizer = torch.optim.LBFGS(model.parameters(),lr=1e-2)  #
    otimizador
126       for i in range(80):
127           optimizer.step(closure) #atualiza os parametros de acordo com o
    #
128           otimizador
129           if np.isnan(closure().item()):  #caso a inicializacao aleatoria
130           #contenha valores improprios, reiniciar
131             break
132           if i%10==0:
133             print(closure())
134       if np.isnan(closure().item()): #se detectado algum erro estocastico,
       reiniciar
135             continue
136       xx = np.linspace(0, 1, 500)[:, None]
137
138       if modo == "simeoni":
139         with torch.no_grad():
140           yy = ((model(x)[0].numpy() + model(x)[1].numpy())* max_y)[:,0]
    #solucao para Simeoni
141       else:
142         with torch.no_grad():
143           yy = (model(x)[0].numpy() * max_y)[:,0] #Solucao para sistemas
    de 1 ordem
144
```

```
145     fig, ax = plt.subplots(dpi=100)           #plot da solucao
146     ax.plot(x_dataset, y_dataset,'.' ,label='dataset')
147     ax.plot(xt*max_x, yy, '--', label=modo)
148     ax.set_xlabel('$t$')
149     ax.set_ylabel('$f(t)$')
150     if modo =="logistica":
151        titulo = 'Modelo logistico - Solucao por rede neural'
152     else:
153       titulo = 'Modelo ' + modo + " - Solucao por rede neural"
154     ax.set_title(titulo)
155     plt.legend(loc='best');
156     return x_dataset,y_dataset, model
157     break
158
159 x_dataset,y_dataset, model = solucionar_edo("bertallanfy")
160 x_dataset,y_dataset, model = solucionar_edo("logistica")
161 x_dataset,y_dataset, model = solucionar_edo("gompertz")
162 x_dataset,y_dataset, model = solucionar_edo("simeoni")
```

Listing B.1 – Código para solução de EDOs de primeira ou segunda ordem e sistemas com 2 EDOs. Desenvolvido em *Python* através do *Jupyter Notebook*.