



**UNIVERSIDADE FEDERAL DO CEARÁ  
CENTRO DE TECNOLOGIA  
DEPARTAMENTO DE ENGENHARIA QUÍMICA  
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA QUÍMICA**

**RENAN MENDES FROTA**

***PHYSICS-INFORMED NEURAL NETWORKS* APLICADAS À SIMULAÇÃO DE  
BIORREATORES:  
UM ESTUDO DE CASO COM A PRODUÇÃO DE ÁCIDO LÁTICO**

**FORTALEZA**

**2022**

**RENAN MENDES FROTA**

***PHYSICS-INFORMED NEURAL NETWORKS* APLICADAS À SIMULAÇÃO DE  
BIORREATORES:  
UM ESTUDO DE CASO COM A PRODUÇÃO DE ÁCIDO LÁTICO**

Dissertação de Mestrado submetida à coordenação do Programa de Pós-Graduação em Engenharia Química, do Centro de Tecnologia, da Universidade Federal do Ceará, como requisito parcial para obtenção do título de Mestre em Engenharia Química. Área de concentração: Processos Químicos e Bioquímicos. Área de concentração: Processos Químicos e Bioquímicos.

**Orientador:** Prof. Dr. Ivanildo José da Silva Júnior  
**Coorientador:** Prof. Dr. Amaro Gomes Barreto Júnior

**FORTALEZA**

**2022**

## RESUMO

Redes Neurais com Embasamento Físico (PINN, do inglês *Physics-Informed Neural Network*) são uma técnica relativamente recente capaz de simular sistemas com embasamento físico com Redes Neurais (NN, do inglês *Neural Network*). Para isso, os dados necessários para otimização da NN são gerados a partir de equações advindas de modelos matemáticos. PINNs podem substituir métodos numéricos e outras formas de resolver sistemas de equações empregados em Engenharia Química, como processos de produção, purificação e reações químicas. Biorreatores são um desafio natural para PINNs por sua complexidade intrínseca e, portanto, estratégias que busquem reduzir a *loss* (erro) e melhorar a convergência dos resultados à solução correta precisam ser investigados. O processo de produção do ácido láctico foi escolhido como um estudo de caso para investigar técnicas de otimização de PINNs aplicados à simulação de biorreações devido à abundância de literatura científica relacionada, além de suas grandes importâncias técnica e econômica. Os valores ótimos para o tamanho e largura de camada da NN, pesos da função *loss* e fatores (*scalers*) de adimensionalização foram determinados para a produção de ácido láctico por *Lactobacillus casei* em um reator batelada. A adimensionalização do tempo resultou em um erro maior em praticamente todos os casos. O sistema proposto foi capaz de reproduzir com grande fidelidade (comparado com um método numérico alternativo e dados experimentais) o processo estudado, alcançando valores de *loss* inferiores a  $10^{-4}$  em volume constante (reator batelada). Contudo, o modelo se demonstrou deficiente ao reproduzir modelos com variação de volume (CSTR e batelada alimentada). A performance da simulação do CSTR e batelada alimentada pode ser melhorada aplicando  $X_M$  (concentração de biomassa inibitória) como o fator de adimensionalização da concentração da biomassa, e quando o  $V_{max}$  (volume máximo do reator) como o fator de adimensionalização do volume. A concentração de biomassa e o volume de líquido no reator foram determinadas como as variáveis mais relevantes para a redução da *loss* e consequente geração de resultados mais precisos.

**Palavras-chave:** PINN; Ácido Láctico; Simulação; Biorreator; Machine Learning

## ABSTRACT

Physics-Informed Neural Network (PINN) is a relatively new technique capable of simulating physics-based systems with Neural Networks (NN) using mathematical models equations to generate the necessary data for NN optimization. PINNs can replace numerical methods and other techniques in solving systems of equations used by Chemical Engineering, such as production and purification processes, and chemical reactions. Bioreactors pose a challenge for PINNs because of their intrinsic complexity, and thus strategies that improve loss reduction and solution convergence must be studied. Lactic acid production was chosen as a case study for the investigation of PINN optimization strategies for the simulation of bioreactions because of its great economic and technological importance and abundant scientific literature available. The optimal layer deepness and wideness, loss weights and nondimensionalization factors were determined for a batch reactor production of lactic acid by *Lactobacillus casei*. The nondimensionalization of time increased loss in all cases. PINNs were able to reproduce with great fidelity numerical results and experimental data in constant volume (batch reactor) with loss values  $< 10^{-4}$ , but showed poor performance in models with volume variation (CSTR and fed-batch). CSTR and fed-batch performance improved when  $X_M$  (inhibitory biomass concentration) was used as a nondimensionalization scaler of biomass concentration and  $V_{max}$  (reactor maximum volume) as a nondimensionalization scaler of volume. Biomass concentration and reactor liquid content volume were also the most relevant variables to reduce loss values and impact significantly in the final solution.

**Keywords:** PINN; Lactic Acid; Simulation; Bioreactor; Machine Learning

## LISTA DE FIGURAS

Figura 1 - Representação gráfica de uma rede neural simples.....	18
Figura 2 - Representação gráfica de um LTU.....	19
Figura 3 - Representação gráfica de um perceptron.....	20
Figura 4 - Multi-Layer Perceptron.....	22
Figura 5 - Esquema de Funcionamento do GD.....	28
Figura 6 - Esquema de Funcionamento do SGD.....	28
Figura 7 - Esquema de Funcionamento do mini-batch.....	29
Figura 8 - <i>Dropout</i> em rede neural com uma <i>hidden layer</i> .....	46
Figura 9 - Diferentes regimes de dados para PINNs.....	49
Figura 10 - Modelo esquemático do funcionamento de um PINN.....	84

## LISTA DE GRÁFICOS

Gráfico 1 - Funções de ativação em função do valor de $z$ .....	35
Gráfico 2 - Resultados do teste de adimensionalização do tempo para reator batelada.....	91
Gráfico 3 - Loss (teste) ao longo de epochs para o teste <code>layer_size</code> .....	93
Gráfico 4 - Loss para teste <code>weight</code> com 3 camadas com 22 neurônios.....	95
Gráfico 5 - Loss para teste <code>weight</code> com 5 camadas de 32 neurônios.....	96
Gráfico 6 - Comparação entre PINN, Método Numérico e Dados experimentais: Reator Batelada.....	98
Gráfico 7 - Perfis para Batelada, Batelada Alimentada e CSTR em NN com 5 camadas de 32 neurônios.....	100
Gráfico 8 - Loss para Batelada, Batelada Alimentada e CSTR em NN com 5 camadas de 32 neurônios.....	100
Gráfico 9 - Loss para Batelada, Batelada Alimentada e CSTR em NN com 6 camadas de 8 neurônios.....	101
Gráfico 10 - Loss para Batelada, Batelada Alimentada e CSTR em NN com 3 camadas de 90 neurônios.....	102

## LISTA DE TABELAS

Tabela 1 - Parâmetros empregados na equação cinética.....	79
Tabela 2 - Dados experimentais de Altiook (2006).....	81
Tabela 3 - Parâmetros matemáticos para cada reator.....	86

## LISTA DE QUADROS

Quadro 1 - Substratos empregados na produção de LA.....	69
Quadro 2 - Parâmetros de cada caso.....	88
Quadro 3 - Conclusões para cada teste.....	103



## SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO.....</b>	<b>8</b>
<b>2</b>	<b>OBJETIVOS.....</b>	<b>11</b>
<b>2.1</b>	<b>Objetivo Geral.....</b>	<b>11</b>
<b>2.2</b>	<b>Objetivo Específicos.....</b>	<b>11</b>
<b>3</b>	<b>REVISÃO BIBLIOGRÁFICA.....</b>	<b>12</b>
<b>3.1</b>	<b>Machine Learning.....</b>	<b>12</b>
<b>3.1.1</b>	<b><i>Neural Networks</i>.....</b>	<b>16</b>
3.1.1.1	<i>Neurônio.....</i>	17
3.1.1.2	<i>Perceptron.....</i>	18
3.1.1.3	<i>Multi-Layer Perceptron.....</i>	21
3.1.1.4	<i>Back-propagation.....</i>	22
3.1.1.5	<i>Outras Técnicas e Algoritmos de Treino.....</i>	25
3.1.1.6	<i>Funções de Ativação.....</i>	32
3.1.1.7	<i>Funções de Inicialização.....</i>	35
3.1.1.8	<i>Vanishing e exploding gradients.....</i>	41
3.1.1.9	<i>Batch Normalization.....</i>	43
3.1.1.10	<i>Técnicas de regularização.....</i>	45
<b>3.1.2</b>	<b><i>Aplicação em bioprocessos</i>.....</b>	<b>46</b>
<b>3.2</b>	<b>Physics-Informed Neural Network.....</b>	<b>48</b>
<b>3.2.1</b>	<b><i>Solução de equações diferenciais orientada por dados</i>.....</b>	<b>54</b>
<b>3.2.2</b>	<b><i>Descoberta de equações diferenciais orientada por dados</i>.....</b>	<b>55</b>
<b>3.2.3</b>	<b><i>Aplicações</i>.....</b>	<b>56</b>
3.2.3.1	<i>Gerais.....</i>	57
3.2.3.2	<i>Physics-based architecture PINNs.....</i>	60
3.2.3.3	<i>BINN.....</i>	62
3.2.3.4	<i>Falhas e Desafios.....</i>	65
<b>3.3</b>	<b>DeepXDE.....</b>	<b>66</b>
<b>3.4</b>	<b>Engenharia de Processos.....</b>	<b>67</b>
<b>3.4.1</b>	<b><i>Processos estacionários e transientes</i>.....</b>	<b>67</b>
<b>3.4.2</b>	<b><i>Dimensionamento e Simulação de Equipamentos</i>.....</b>	<b>67</b>
<b>3.5</b>	<b>Ácido Lático.....</b>	<b>68</b>

3.5.1	<i>Cinética de produção</i> .....	71
3.6	Modelagem matemática.....	75
3.6.1	<i>Tanque</i> .....	75
3.6.2	<i>Reatores CSTR e Batelada</i> .....	75
3.6.3	<i>Variáveis de Processo</i> .....	76
3.6.4	<i>Modelo</i> .....	77
4	METODOLOGIA.....	78
4.1	Adimensionalização.....	78
4.2	Modelo Matemático da Cinética de Reação.....	79
4.3	Dados Experimentais.....	81
4.4	Modelo Matemático do reator.....	82
4.5	Simulação usando <i>Physics-Informed Neural Networks</i> .....	83
4.6	Configurações de Simulação.....	84
5	RESULTADOS E DISCUSSÃO.....	88
5.1	Testes de Adimensionalização ( $t_s$ e $non\_dim$ ).....	90
5.2	Testes de Largura e Profundidade da Rede ( $layer\_size$ ).....	92
5.3	Testes de Pesos da Função Loss ( $weight$ ).....	93
5.4	Teste Comparativo e de Validação ( $pinn\_numeric\_xp$ ).....	97
5.5	Teste de múltiplas configurações ( $multiple\_config$ ).....	98
6	CONCLUSÃO.....	105
7	REFERÊNCIAS.....	107

## 1 INTRODUÇÃO

Há aproximadamente 70 anos, a Indústria Química foi a primeira das Indústrias da sociedade civil a empregar o uso de computadores, ainda na década de 1950. Desde então, os custos para a aquisição de computadores caiu consideravelmente, enquanto que a capacidade de processamento e armazenamento aumentou de forma expressiva (JOHNS, 2011). Os custos com a aquisição de computadores e softwares, portanto, se justificam não somente por permitirem economizar ao escolher projetos ótimos e realizar centenas de cálculos que seriam muito dispendiosos e praticamente impossíveis de outra forma.

A simulação de processo de Engenharia Química frequentemente emprega métodos numéricos tradicionais, como Ruge-Kutta e Euler Contudo, essas técnicas de solução numérica ainda estão sujeitas a instabilidades e podem ser bastante sensíveis à estratégia de discretização numérica empregada. A precisão pode ser aumentada empregando malhas mais finas, com mais pontos de discretização, mas isso também pode gerar erros no sistema e/ou aumentar consideravelmente o esforço computacional necessário. Além disso, sua implementação requer conhecimento técnico tanto em arquitetura de software e programação quanto em modelagem matemática, além dos conhecimentos específicos da área de interesse. O uso de softwares proprietários, de código fechado ou não acessíveis como Matlab, Aspen Plus e COMSOL Multiphysics é empregado (JANOSKA; BUIJS; VAN GULIK, 2023; LI; XU, 2023) por poder reduzir o esforço necessário, ou tornar o fluxo de trabalho mais agradável ou, ainda, melhorar os resultados obtidos ao fornecer técnicas aprimoradas. Dependendo do modelo a ser realizado, o custo experimental pode ser relativamente elevado e até mesmo proibitivo.

Redes Neurais Artificiais (ANN, do inglês Artificial Neural Network) são aproximadores universais, capazes de representar as mais diversas funções de classificação e regressão numérica. São capazes de representar fenômenos extremamente complexos, mesmo com uma quantidade relativamente reduzida de parâmetros. Contudo, podem ser computacionalmente intensivas e normalmente necessitam de uma grande quantidade de dados (*big data*). Essa aquisição de uma grande quantidade de dados normalmente é muito custosa e, por vezes, proibitiva. Uma grande vantagem das ANNs é que, apesar de seu alto custo para construção e

otimização do modelo (treino), o uso do modelo já pronto é relativamente simples e pode ser feito em computadores menos robustos.

Redes Neurais com Embasamento Físico (PINN, do inglês Physics-Informed Neural Networks) são uma abordagem relativamente recente, que emprega ANN para a solução de sistemas de equações diferenciais, em alternativa aos tradicionais métodos numéricos (RAISSI; PERDIKARIS; KARNIADAKIS, 2019; SANTANA et al., 2022). Nessa abordagem, a capacidade de aproximação universal das ANN é empregada para a solução do sistema de equações, e a grande quantidade de dados necessárias para treino da ANN é contornada ao se empregar modelos matemáticos. Em resumo, a rede é responsável por predizer as variáveis dependentes (saídas da rede neural) com base nas entradas (variáveis independentes). Desse modo, a grande capacidade de predição e solução de problemas das ANN pode ser aplicada à solução de equações diferenciais, ao mesmo tempo em que contorna adequadamente a limitação de dados. Para a otimização de um PINN, uma função erro (frequentemente chamada de *loss*) é minimizada ao se modificar os diferentes parâmetros que relacionam neurônios e camadas da rede neural. Essa técnica é muito relevante em casos onde a solução numérica é extremamente complicada ou custosa.

Reações biológicas são naturalmente um desafio para PINNs, por apresentarem com frequência dependência múltipla entre diversas variáveis e suas respectivas derivadas. Contudo, o uso de PINNs abre caminho para grandes inovações na simulação de bioprocessos, pois pode permitir uma maior precisão e adequação em diversos modelos, bem como facilitar a reprodução de resultados através de compartilhamento dos modelos já otimizados. Como é uma técnica relativamente recente, ainda são necessários esforços no sentido de entender melhor o funcionamento e sua otimização aplicada a bioprocessos já que, embora seja muito relevante, nem sempre é capaz de resolver adequadamente os sistemas simulados (LI; XU, 2023; SANTANA et al., 2022).

O LA (Ácido Lático, do inglês *Lactic Acid*) é uma molécula de grande interesse econômico, com aplicações nas indústrias alimentícia, farmacêutica, têxtil e cosmética (KOMESU; MACIEL; FILHO, 2017; LÓPEZ-GÓMEZ et al., 2019). É ainda empregado na síntese de PLA (Poli-Ácido Lático), matéria-prima de polímeros comumente usados em impressões 3D. Sua produção pode se dar por via fermentativa ou síntese química, sendo a fermentativa usada em cerca de 90% dos

casos. Dentre as principais linhas de pesquisa recentes relacionadas ao LA, destacam-se as pesquisas por diferentes fontes de carbono para a via fermentativa, a redução dos custos de produção e a redução dos custos de purificação.

O trabalho se propôs a investigar técnicas de otimização e racionalização da construção de PINNs tendo a produção de LA por *L. casei* conforme descrito por (ALTIOK; TOKATLI; HARSA, 2006) como um estudo de caso. Para isso, foram estudados diversos parâmetros, hiperparâmetros e configurações em reatores operando nos regimes batelada, batelada-alimentada e contínuo. A biblioteca DeepXDE, em Python, foi empregada para a criação dos modelos PINN (LU et al., 2021). Foram determinadas as condições ótimas para produção de PINNs que pudessem representar adequadamente o processo fermentativo descrito, bem como discutidas as diferenças entre os modelos de operação empregados e seu impacto na capacidade dos PINNs de representar ou não adequadamente as concentrações de interesse (biomassa, produto e substrato) e volume do reator.

## **2 OBJETIVOS**

### **2.1 Objetivo Geral**

O objetivo geral deste trabalho é avaliar diferentes estratégias para o emprego de PINNs (do inglês Physics-Informed Neural Networks, Redes Neurais com Embasamento Físico) em simulações de biorreatores e reações biológicas.

### **2.2 Objetivo Específicos**

- Propor uma metodologia para a melhoria da performance de PINNs na solução de Equações Diferenciais Ordinárias, em específico na simulação da produção de LA em um reator;
- Testar o impacto da adimensionalização das variáveis tempo, volume, e concentrações de biomassa, produto e substrato na performance e erro da simulação de um fermentador por PINN;
- Determinar a relevância dos efeitos de adimensionalização, quantidade de camadas e neurônios por camada na performance de modelos de reator simulados por PINN que alcançam o estado estacionário (batelada e CSTR) e modelos permanentemente transientes (batelada alimentada).

### 3 REVISÃO BIBLIOGRÁFICA

#### 3.1 Machine Learning

ML (Aprendizado de Máquina, do inglês Machine Learning) é um termo geral para denominar a área da ciência que cria algoritmos capazes de executar funções para as quais os algoritmos não foram explicitamente programados. Em outras palavras, esses algoritmos têm a capacidade de aprender de forma implícita, e daí se vem o conceito de Aprendizagem pela Máquina (computador) (GÉRON, 2017). Algoritmos de ML são versáteis, adaptáveis e muitas vezes capazes de representar ou interpretar situações que seriam demasiadamente complexas se feitas de outra forma.

Uma grande desvantagem dos métodos de ML é a necessidade do fornecimento de uma grande quantidade de dados para produção de modelos com boas capacidades de predição. Muitas vezes, não é possível obter uma quantidade tão grande de dados a ponto de ajustar os modelos propostos em condições normais de laboratório, sobretudo na área de engenharia química. Em um estudo com reatores batelada, por exemplo, alguns estudos coletam de 7 a 14 pontos experimentais (ALTIOK; TOKATLI; HARSA, 2006; GUILHERME, [s.d.]), enquanto que muitas vezes são necessário dados ao menos uma ordem de grandeza acima (de 100 a 1000) para poder produzir e testar um modelo de ML.

Ao contrário de modelos matemáticos, que são propostos muitas vezes usando argumentação lógica, as expressões matemáticas derivadas desse raciocínio e, quando necessário, ajustes empíricos (termos que se são usados para tornar o modelo mais preciso matematicamente, mas que não tem embasamento teórico), modelos de machine learning são muito mais dispersos -, no sentido de que podem representar quaisquer fenômenos, inclusive relações não existentes. É, portanto, necessário ressaltar que tais modelos podem ser superajustados com muita facilidade (isto é, o modelo representa perfeitamente o caso de teste, mas não é capaz de generalizar para outros contextos similares porém não idênticos). Assim,

um desafio extra se dá na necessidade de ter uma grande quantidade de dados não só para produzir o modelo (etapa chamada de *train*) mas também para testá-lo (etapa chamada de *test*). Essa separação é crucial para evitar que o modelo *superajuste*, sendo capaz de reproduzir com perfeição os dados de teste, mas incapaz de ser usado para prever outros casos, que é justamente a ideia por trás do emprego de modelos de ML.

A principal função de algoritmos de ML é a generalização de problemas, de forma que conjuntos de informações de entrada (que podem variar de variáveis numéricas a cores, textos ou pixels de imagens) sejam capazes de produzir resultados consistentes de saída (que podem ser quantitativos [como em sistemas de regressão que calculam a concentração dada uma massa e volume] ou qualitativos [como em um sistema de classificação que determinar se uma imagem se trata de um carro ou um animal, por exemplo]). Para isso, o modelo de ML é treinado atualizando seus principais parâmetros (pesos e *biases* no caso de redes neurais, introduzidos nas próximas seções) através de um conjunto de dados. O treino normalmente se dá pela minimização de funções erro (muitas vezes chamadas de *loss* ou *cost*), que variam de acordo com as necessidades do algoritmo. A cada etapa de iteração (de treino) completa se dá o nome de *epoch*.

A *Cross-Validation* (Validação Cruzada) consiste em separar, dentro de um mesmo conjunto de dados, uma parte para ser usada na etapa de *train* da ML, e outra para ser usada na etapa de *test* do modelo treinado. Esses testes e treinos são feitos de modo intercalado, até que se atinja um critério determinado (erro obtido abaixo de um erro mínimo pré-determinado ou um número de iterações [epochs] máximo, por exemplo). Normalmente, a quantidade destinada ao teste é complementar à do treino, no sentido em que ambas somam em 100%. Por exemplo, um conjunto de dados com 1000 pontos e com 400 pontos de teste teria 600 pontos de treino, totalizando o valor original.

A *Cross-Validation* permite, ainda, testar como o modelo se comporta de acordo com a quantidade de dados (proporções 20:80 e 50:50 entre percentual de treino e percentual de teste, por exemplo) e produzir modelos otimizados. Outra vantagem é que a comparação entre o erro médio ou desvio das predições entre a seção destinada a treino e aquela destinada a testes permite-se ter uma noção se o sistema está superajustado aos testes ou ao treino. Um sistema com baixo erro de treino mas alto erro de teste pode indicar, por exemplo, ou que o sistema treinado



está superajustado (e não será capaz de generalizar adequadamente) ou que o número de pontos de dados de teste ou de treino é muito baixo, o que torna a comparação enviesada. Idealmente, o valor de ambos deve ser próximo, pois indica que o sistema tanto foi capaz de reproduzir os dados de treino adequadamente, quanto foi capaz de extrapolar e prever dados com parâmetros de entrada distintos.

Uma das dificuldades ao se trabalhar com ML é a necessidade, além de uma abundância de dados, envolve o balanceamento dos mesmos. Sistemas que contenham muitos dados de determinado tipo podem favorecer previsões com baixo erro mas que, ainda assim, serão modelos inadequados para as situações em que se pretende empregá-los. Por exemplo, consideremos um modelo que tenta prever a presença de bactérias em um meio com base nos gases emitidos ao longo do tempo. Caso sejam alimentados 990 dados para a bactéria tipo A e 10 dados para a bactéria tipo B, o modelo poderá se limitar a prever que em 0% dos casos a bactéria B está presente e ainda assim acertar em 99% das vezes. Para esse tipo de dado, é comum o uso de um *balanceamento*, que pode ocorrer pela duplicação dos dados faltantes (replicar os dados da bactéria B 90 vezes, de modo que o sistema passe a ter medidas balanceadas dos casos A e B) ou remoção dos dados em excesso de A (reduzindo-os de 900 para 20, por exemplo) (GÉRON, 2017).

Outro ponto sensível de ML é na surpreendente grande dependência do fator humano: a filtragem dos dados que serão alimentados no sistema, bem como a escolha de algoritmos adequados e de interpretação adequada dos resultados é crucial para propor modelos que realmente representem o fenômeno estudado, fazendo jus ao esforço empregado. Desta forma, é necessário que a seleção dos dados seja feita não somente por profissionais da área de engenharia de dados ou de aprendizado de máquina, mas também por profissionais da área de coleta e/ou de aplicação desses dados.

Um ponto de grande relevância é que vários modelos de ML, em especial Redes Neurais, podem demorar bastante tempo para serem otimizados. Contudo, uma vez que o modelo esteja construído, ele pode ser executado em frações de segundo (GÉRON, 2017; SANTANA et al., 2022). A implicação dessa característica é que, normalmente, são necessários computadores mais robustos, bem como um esforço computacional e tempo de *treino* maior em comparação com métodos tradicionais de classificação ou regressão. Contudo, os resultados são *data-driven*,

capazes de representar fenômenos sem equacionamento matemático próprio e, mais importante, os modelos gerados são relativamente leves. Em outras palavras, a maior exigência computacional na fase de testes e criação do modelo é compensada pois uma vez que o modelo tenha sido construído e validado, pode ser usado indefinidamente por um custo computacional muito reduzido.

Os algoritmos que empregam ML podem se dividir em basicamente duas grandes categorias: classificação e regressão. Os algoritmos de classificação são responsáveis por prever categorias ou classificações de modelos com base em dados de entrada. Um exemplo clássico são programas capazes de analisar fotografias ou vídeos e determinar as pessoas existentes nessas imagens com base no rosto. Nesse sentido, um algoritmo supervisionado seria aquele que já foi treinado com as pessoas que deve reconhecer, e classifica os rostos detectados como uma dessas pessoas. No mesmo exemplo, um algoritmo não supervisionado seria aquele que detecta todos os rostos de pessoas e os agrupa para determinar as pessoas existentes em um conjunto de fotografias, mas não as rotula. Ou seja, a ideia de supervisão é que as classes ou categorias são informadas previamente, e o algoritmo deve determinar a categoria a cada objeto de interesse analisado pertence. No não supervisionado, o algoritmo deve produzir as possíveis categorias ao mesmo tempo em que já associa cada objeto analisado a uma delas. Um algoritmo de regressão é aquele que produz resultados numéricos. Dentre os principais algoritmos de regressão supervisionados, destacam-se (GÉRON, 2017):

- *k-Nearest Neighbours* (Vizinhos Próximos);
- Regressão Linear;
- Regressão Logística;
- *Support Vector Machines* (SVMs);
- Árvores de decisão (DF, do inglês *Decision Trees*);
- Floresta Randômica (RF, do inglês *Random Forest*);
- Redes Neurais.

Dado que em Engenharia Química comumente se trabalha com dados numéricos, algoritmos de regressão tem um maior interesse imediato. Existem estudos aplicando alguns dos modelos citados à predição de variáveis de efluentes (MATEO PÉREZ et al., 2021), à produção de bio-hidrogênio (PANDEY et al., 2023) e cream cheese (LI et al., 2021), ao tratamento de efluentes (BAGHERZADEH et al.,

2021), à digestão anaeróbia (ANDRADE CRUZ et al., 2022), dentre outros. Todos os modelos aqui apresentados requerem uma quantidade razoável de dados disponíveis para boa performance sem superajuste. Muitas vezes essa abundância de dados não existe na literatura científica, ou por ser muito custosa, ou por ser impraticável, ou pelos estudos que geraram dados terem sido realizados para uso interno de uma empresa ou instituição. Por isso, outras estratégias foram buscadas, como os PINNs.

### **3.1.1 Neural Networks**

ANN (Redes neurais artificiais, do inglês Artificial Neural Networks) são algoritmos altamente adaptáveis (ANDRADE CRUZ et al., 2022), capazes de simular o funcionamento do cérebro humano. Possuem a capacidade de funcionar como aproximadores universais de qualquer função com entradas e saídas (SANTANA et al., 2022). Foram citados pela primeira vez em 1943, com um modelo matemática que tentava reproduzir o comportamento de neurônios tendo como base o cérebro de animais (GÉRON, 2017). O conceito rapidamente ganhou interesse e se difundiu, atraindo a atenção de diversas áreas do conhecimento.

Inicialmente, criou-se uma expectativa muito grande a respeito de redes neurais, o que fez com que o interesse nas mesmas disparasse, mas como não foram capazes de atender às altíssimas expectativas de computadores super inteligentes rapidamente – e uma série de limitações foi sendo encontrada conforme a área era explorada –, foram perdendo espaço lentamente até a década de 80, quando ocorreu um novo crescimento de interesse exponencial. Nesse período, novas técnicas surgiram e expandiram consideravelmente o potencial das ANN. Outro fator relevante para o grande interesse em ANN a partir da década de 90 foi o grande aumento da capacidade de processamento computacional disponível em todos os computadores, em especial àqueles aos quais a maioria dos cientistas tinha acesso: os comuns. A indústria dos jogos também foi uma grande responsável pelos avanços dessa época, uma vez que estimulou a criação de GPUs (Unidades de Processamento Gráfico, do inglês *Graphics Processing Units*) mais potentes (GÉRON, 2017).

A menor unidade de uma rede neural é neurônio, que faz analogia aos neurônios biológicos (humanos). Mantendo a analogia a uma rede neural biológica, as ANN possuem três camadas: o input (entrada), a *hidden layer* (camada secreta) e a *output* (saída). Cada conexão entre neurônios de diferentes camadas possui um *weight* (peso) responsável por balancear a importância entre os diferentes neurônios e camadas para a redução do erro das previsões (ALZUBI; NAYYAR; KUMAR, 2018).

Os principais hiper-parâmetros (parâmetros relacionados à própria estrutura ou taxa de aprendizado) da ANN são (GÉRON, 2017; NGO; LIM, 2021):

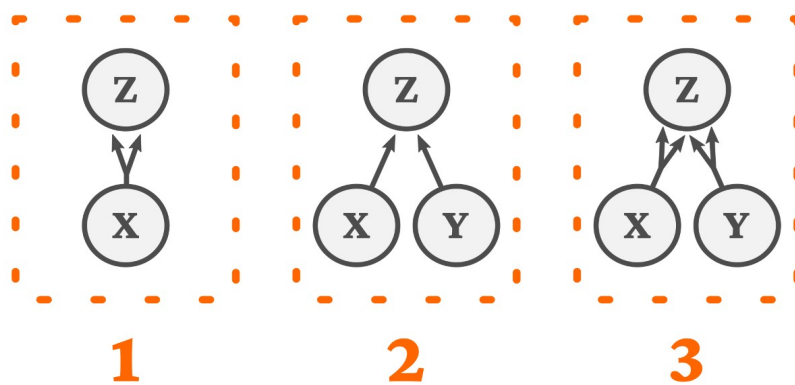
- O número de camadas da NN, também chamado de *Deepness* (profundidade) da *hidden layer*;
- O número de neurônios por camada oculta, também chamado de *Wideness* (largura);
- A função de ativação empregada;
- O número de pontos empregados para o treino (em alguns algoritmos chamadas de *domain*);
- A taxa de aprendizado (normalmente igual a  $1 \times 10^{-3}$ ), que é o valor usado para influenciar o quanto a rede modifica o conjunto de pesos a cada interação com base no erro.

### 3.1.1.1 Neurônio

Um neurônio artificial é tão somente uma função que ativa determinados *outputs* (valores de saída) em função dos *inputs* (valores de entrada) recebidos. Seu comportamento é do tipo binário (como ativado/desativado, liga/desliga, ou verdadeiro/falso) e é capaz de representar os mais diversos comandos lógicos (GÉRON, 2017). A Figura 1 mostra um esquema de um neurônio (Z) capaz de gerar uma saída binária em função das entradas (X e Y). As setas representam as conexões. A seção 1 da figura mostra um neurônio com a chamada função identidade: o neurônio Z está ativo quando o neurônio X também estiver, pois as duas entradas que recebe vem diretamente de X. Da mesma forma, caso X esteja desativado ou com um sinal negativo, Z simplesmente replicará esse resultado. A

seção 2 mostra uma operação lógica do tipo “e” (AND): Z será ativado apenas se X e Y, simultaneamente, tiverem valores também ativados. Todas as outras combinações resultam na saída do neurônio Z como “negativa” ou “desativada”. A seção 3 mostra uma conexão do tipo “ou” (OR): O neurônio Z é ativado se X ou Y ou ambos estiverem ativados, pois é capaz de receber duas entradas de cada um. Mesmo com esquemas simples como esse, é possível criar as mais diversas representações lógicas para representar o sistema de interesse.

Figura 1 - Representação gráfica de uma rede neural simples



Fonte: Adaptado de (GÉRON, 2017)

### 3.1.1.2 Perceptron

Um perceptron é uma das arquiteturas mais simples de Redes Neurais. Criado em 1956, o conceito resistiu a décadas de evolução das NN e ainda permanece bastante relevante até os dias atuais (GARDNER; DORLING, 1998; GÉRON, 2017; ROSENBLATT, 1958). Os neurônios empregados em um perceptron são chamados LTU (Unidade de Limiar Linear, do inglês Linear Threshold Unit). Nos LTUs, as saídas e entradas do neurônio são numéricas, e não binárias – contrastando fortemente com os modelos de neurônios tradicionais, apresentados na seção 3.1.1.1.

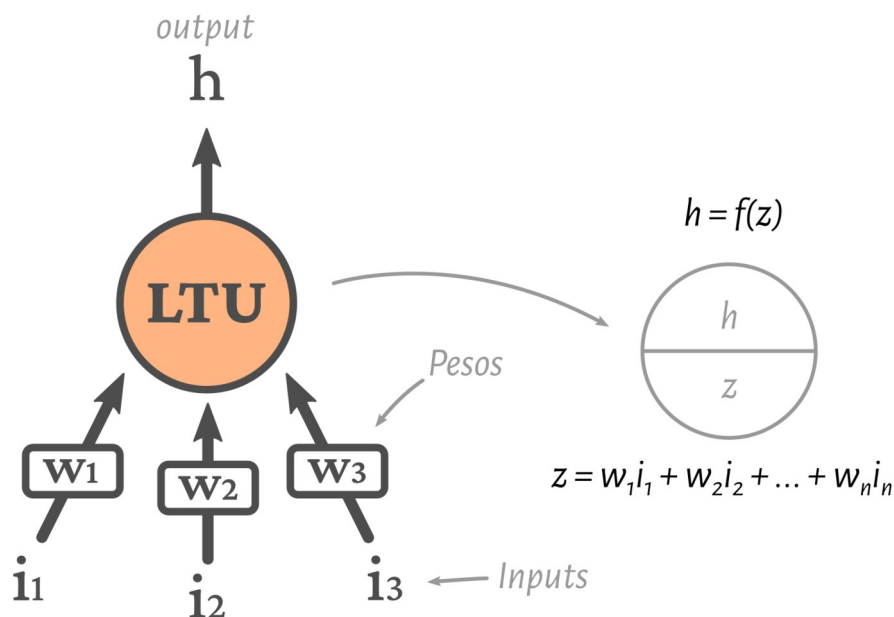
Um LTU soma os valores de entrada ponderados por pesos (*weights*) e aplica uma função para gerar a saída a partir dessa soma. O esquema de funcionamento de um LTU simples é mostrado na Figura 2. A partir de uma série de dados de

entrada (representados por  $i_1$ ,  $i_2$  e  $i_3$ ), é obtido um resultado ou saída. O somatório de pesos multiplicados pelas entradas pode ser definido como  $z = w_1i_1 + w_2i_2 + \dots + w_ni_n$ , conforme indicado na Equação 1. Esse valor é então transformado, através de uma função, em  $h$  – o valor de saída do LTU. O treino de um LTU diz respeito à etapa de otimizar os pesos ( $w_1$ ,  $w_2$ , ...,  $w_n$ ) para que gerem saídas mais próximas das de interesse. Em resumo, são os pesos que determinam o resultado de saída, juntamente com a função de transformação em  $h$ .

$$z = w_1i_1 + w_2i_2 + \dots + w_ni_n = \sum_{m=0}^{m=n} w_m i_m \quad (1)$$

Em que  $w_i$  é o peso atribuído à entrada  $i$  e  $N$  é o número de entradas.

Figura 2 - Representação gráfica de um LTU

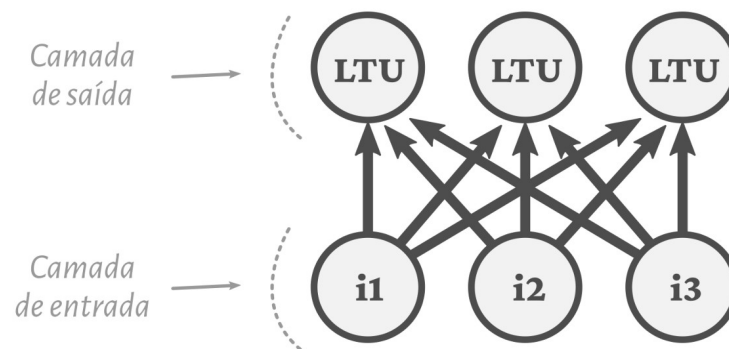


Fonte: Adaptado de (GÉRON, 2017)

Um perceptron é composto por uma única camada de LTUs. Nesse caso, cada neurônio (ou LTU) se conecta a todas as entradas. Normalmente para fazer a transferência de dados entre essas entradas e a NN, é usado o chamado *input neuron* (neurônio de entrada), que apresenta como valor de saída o próprio valor de entrada. Um esquema simples de um perceptron é mostrado na Figura 3. A camada de saída é composta pelo conjunto de perceptrons que calculam os valores de saída

do sistema. Dentre os neurônios de entrada, alguns podem ser um *bias neuron*, isto é: um valor constante, capaz de efetivamente permitir a linearização do modelo. Por exemplo, em uma equação do tipo  $y = a \cdot x + b$ , o *bias* faria um papel semelhante ao do termo “b” de deslocar a reta sem mudar a inclinação.

Figura 3 - Representação gráfica de um perceptron



Fonte: Adaptado de (GÉRON, 2017)

Conforme citado, o aprendizado ou treino de um perceptron consiste basicamente em otimizar os valores dos pesos de suas respectivas entradas. O treino clássico para perceptrons consiste na abordagem de Hebbian, chamada de *Hebbian Learning* (Aprendizado de Hebbian). Mais uma vez é feita uma analogia a neurônios biológicos: neurônios que apresentam a mesma saída devem ter suas conexões (balizadas pelos pesos) reforçadas. O treinamento se dá através da alimentação de um conjunto de *inputs* e comparação com a *output* esperada por vez, em ciclos. Ou seja, é um processo iterativo baseado em um conjunto de dados que contém entradas e as saídas esperadas para cada uma dessas entradas. O procedimento é representado pela Equação 2 (GÉRON, 2017). O peso é atualizado a cada iteração ao se comparar o resultado obtido, o resultado esperado (fornecido pelo conjunto de dados de teste) e o valor do peso na etapa atual (representado pelo sobrescrito “n+1”) e o valor do peso na etapa anterior (representado pelo sobrescrito “n”).

$$w_{i,j}^{n+1} = w_{i,j}^n + \eta(\hat{y}_j - y_j)x_i \quad (2)$$

Em que  $w_{i,j}$  é o peso entre o neurônio de entrada  $i$  e o neurônio de saída  $j$ ;  $x_i$  é o valor de saída do neurônio de entrada  $i$ ;  $\hat{y}_j$  é o valor de saída do neurônio de saída  $j$ ;  $y_i$  é o valor esperado para os dados;  $\eta$  é a taxa de aprendizado (*learning rate*). O sobrescrito  $n$  representa a iteração de “ $w$ ”.

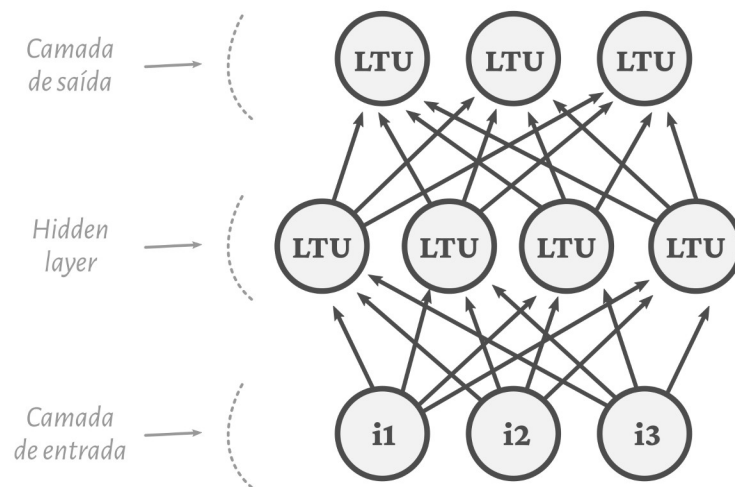
### 3.1.1.3 Multi-Layer Perceptron

Uma das grandes limitações de perceptrons é que são puramente lineares. Por isso, são incapazes de prever resultados de maior complexidade, ou mesmo executar tarefas de classificação mais complexas, como um condicional “ou” exclusivo (XOR). Contudo, muitas dessas limitações podem ser contornadas ao se criar uma série de camadas de perceptrons empilhadas (GÉRON, 2017). Assim, os valores de entrada de uma camada são, na verdade, os de saída da outra, e não somente os valores dos *inputs neurons*. Esse tipo de abordagem é denominada MLP (Perceptron Multicamada, do Inglês *Multi-Layer Perceptron*) e comprovadamente é capaz de aproximar qualquer função continuamente diferenciável com o devido treinamento (GARDNER; DORLING, 1998). O MLP possui ainda boa capacidade de generalização até mesmo para modelos complexos e não lineares.

As camadas internas de perceptrons do MLP são chamadas de *hidden layers* (camadas secretas ou escondidas). Cada perceptron se conecta (recebe os valores de saída) com todos os perceptrons da camada imediatamente anterior, sucessivamente, até que os valores sejam passados para a camada de saída. Quando a NN possui duas ou mais *hidden layers*, ela é chamada de Rede Neural Profunda (DNN, do inglês *Deep Neural Network*). Um esquema de um MLP é mostrado na Figura 4.



Figura 4 - Multi-Layer Perceptron



Fonte: Autoria Própria (2022)

#### 3.1.1.4 Back-propagation

Uma grande dificuldade do uso de MLPs se dá pela complexidade em otimizar os pesos para vários neurônios em camadas distintas. Esse desafio foi estudado ao longo de anos e, ainda antes da década de 90, em 1986, um dos mais clássicos algoritmos foi introduzido: o *back-propagation* (RUMELHART; HINTONT; WILLIAMS, 1986). Toda a descrição matemática e lógica desta seção é baseada no trabalho original de Rumelhart, Hintont e Williams (1986), salvo quando indicado explicitamente o contrário.

Bastante poderoso e relativamente simples, o *back-propagation* se baseia na derivada da função erro ou objetivo do sistema. A entrada de cada perceptron continua sendo uma combinação linear dos valores de saída da camada anterior, e a função de saída é necessariamente não linear. É importante que tanto a função de entrada quanto a de saída sejam continuamente diferenciáveis.

A função erro é definida na Equação 3, enquanto a função de saída,  $y$ , é definida na Equação 4. A função  $y$  não precisa ser idêntica à informada, a equação é apenas para referência.

$$E = \frac{1}{2} \sum_c \sum_j (\hat{y}_{j,c} - y_{j,c})^2 \quad (3)$$

$$y_j = \frac{1}{1 + e^{-x_j}} \quad (4)$$

Em que o subscrito c representa a identificação de cada conjunto de dados de entrada e de saída; o subscrito j identifica cada unidade dos neurônios da camada de saída;  $\hat{y}$  é o valor predito; y é o valor esperado; E é o erro.

A ideia por trás do algoritmo *back-propagation* é empregar a derivada da função erro, de acordo com cada peso do sistema, na atualização do conjunto de pesos de toda a Rede Neural. Para um dado conjunto “c”, primeiramente, calcula-se a derivada do erro em relação a cada peso da camada mais externa (a de saída ou *output*), com a Equação 5 (o subscrito c foi omitido).

$$\frac{\partial E}{\partial y} = \hat{y}_y - y_i \quad (5)$$

Pela regra da cadeia, é possível calcular a derivada do erro em função das entradas (x), conforme indicado pelas Equações 6 e 7:

$$\frac{\partial E}{\partial x_j} = \frac{\partial E}{\partial y_j} \frac{dy_j}{dx_j} \quad (6)$$

$$\frac{\partial E}{\partial x_j} = \frac{\partial E}{\partial y_j} y_j (1 - y_j) \quad (7)$$

A equação 6 é o cerne o método *back-propagation*, pois permite que a mudança em um determinado peso em um neurônio rede seja relacionada diretamente ao erro nas previsões dessa rede. A substituição da Equação 4 em 6 gera a 7. A variação do erro com relação a um peso específico  $w_{ji}$  pode ser obtida relacionando a variação do erro no LTU “i” à entrada de dados (saída do LTU “j”), sendo calculada pela equação 8:

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial x_j} \frac{\partial x_j}{\partial w_{ji}} = \frac{\partial E}{\partial x_j} y_i \quad (8)$$

O efeito causado no erro pela saída do LTU “i” na saída do LTU “j” é, por fim, definida na Equação 9, que pode por sua vez ser generalizada para representar o erro na saída  $y_i$  em relação a todos os pesos de j através de um somatório como na Equação 10:

$$\frac{\partial E}{\partial x_j} \frac{\partial x_j}{\partial y_i} = \frac{\partial E}{\partial x_j} w_{ji} \quad (9)$$

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{\partial E}{\partial x_j} w_{ji} \quad (10)$$

Essas equações demonstram o processo cálculo da variação do erro em função da saída da penúltima camada ( $\partial E/\partial y$ ) em relação à variação do erro em relação à saída na última camada (*output layer*). Para as demais camadas, o procedimento deve ser repetido de forma análoga. Um peso qualquer  $w$  pode ser variado ao ser somado com a relação entre a variação no erro e uma taxa de aprendizado  $\eta$ . A Equação 11 representa uma das formas mais simples de se calcular o valor de  $\Delta w$  para fazer esse processo iterativo de atualização dos pesos.

$$\Delta w = -\eta \frac{\partial E}{\partial w} \quad (11)$$

Essa atualização dos valores do peso, por sua vez, pode ser feita de formas distintas. Uma forma é acumular  $\partial E/\partial y$  ao longo de todo o conjunto de dados (entradas e saídas esperadas) antes de aplicar as alterações aos pesos. A outra é aplicar essas alterações a cada caso, treinando a rede ponto a ponto (RUMELHART; HINTON; WILLIAMS, 1986). Nesse caso, o treino consiste basicamente em uma sequência de ações (GÉRON, 2017):

1. *Forward pass*: É feita uma predição com base nos dados de entrada, com o sistema tal qual está;
2. Erro: O erro é calculado com base nos resultados de saída obtidos e nos que eram esperados;
3. *Reverse pass*: O algoritmo “desce” a rede neural, calculando a contribuição de cada peso, da camada mais externa (saída) para a mais interna (imediatamente após os *input neurons*). O cálculo é feito aplicando a lógica da Equação 10;

4. *Gradient descent*: os pesos das conexões entre neurônios são ajustados de forma análoga ao uso da Equação 11 para a minimização da função erro (explicado em maior detalhes na seção 3.1.1.5);
5. A sequência é repetida por um número determinado de iterações ou até que um erro aceitável seja alcançado.

Salienta-se que a *back-propagation* em si não é responsável por não é o treino ou otimização da NN, mas uma forma de calcular os gradientes da rede. O treino é feito por um algoritmo associado – como o *Gradient Descent* apresentado no parágrafo anterior.

Um ponto de fragilidade desse método é a possibilidade de se alcançar um mínimo local, e não um mínimo global. Dessa forma, a derivada do erro é incapaz de ajustar os pesos para ajustar o sistema e convergir ao mínimo global. Uma possível estratégia para contorná-la consiste em dividir o conjunto de dados em vários grupos menores, e treinar a NN com apenas uma parte dos dados (o chamado *batch-training*) por vez, e interromper o treino quando um erro mínimo for alcançado. Na sequência, o sistema volta a ser treinado com o conjunto de dados seguinte, até que todos os dados tenham sido usados (GARDNER; DORLING, 1998).

Contudo, foi empiricamente constatado que o problema de convergência a mínimos locais ocorre majoritariamente em sistemas onde a quantidade de neurônios e camadas é meramente o suficiente para representação do sistema; isto é: em sistemas com relativamente mais camadas e neurônios, normalmente, o mínimo global é alcançado (RUMELHART; HINTON; WILLIAMS, 1986). Reforçando a robustez do método, o *back-propagation* é tido, na atualidade, como uma das estratégias dominantes para o treino de Redes Neurais Profundas (RAISSI; PERDIKARIS; KARNIADAKIS, 2019).

#### 3.1.1.5 Outras Técnicas e Algoritmos de Treino

O treinamento (ou otimização) de uma NN pode ser uma etapa consideravelmente lenta. A otimização de NNs inicialmente foi feita através da técnica *Gradient Descent*. Posteriormente outras técnicas, como *Stochastic Gradient*

*Descent*, L-BFGS (BYRD et al., 1995) e Adam (KINGMA; BA, 2014) foram introduzidas. Nesta seção são abordados os desafios e vantagens, bem como funcionamento teórico de cada uma dessas técnicas.

O *Gradient Descent* (GD), também chamado de *Batch Gradient Descent*, é um algoritmo de otimização genérico, capaz de obter soluções ótimas em uma ampla gama de problemas de ML (GÉRON, 2017). consiste em empregar o gradiente (i.e. o vetor de derivadas parciais de primeira ordem da função objetivo em relação às variáveis de entrada). Uma LR (taxa de aprendizado, do inglês *Learning Rate*) é responsável por controlar o quanto o erro é empregado para a atualização dos pesos e *biases* da rede neural. Quanto maior a LR mais facilmente a NN é otimizada, mas valores muito elevados podem levar a instabilidades e valores muito baixos a zonas de estagnação, onde a NN praticamente não tem sua *loss* reduzida ao longo de milhares de iterações.

Os valores de entrada (*features*) de uma NN otimizada por GD devem ser escalados de forma a terem ordem de grandezas semelhantes – do contrário, a convergência a um mínimo global pode ser consideravelmente custosa (GÉRON, 2017). A equação 12 representa o modelo básico do GD, que consiste em calcular a variação da função *loss* em função das variáveis de entrada (pesos e *biases*) para atualizá-los.

$$\theta^{i+1} = \theta - \eta \nabla_{\theta} \text{loss}(\theta) \quad (12)$$

Em que  $\theta^{i+1}$  são os valores dos parâmetros  $\theta$  na etapa de iteração seguinte,  $\theta$  representa os parâmetros na etapa atual, *loss* representa a função a ser minimizada,  $\nabla_{\theta}$  é o gradiente de  $\theta$  e  $\eta$  é a LR ou taxa de aprendizado.

Embora seja considerado um método clássico e tenha servido de base para diversos dos algoritmos mais modernos, o GD apresenta uma importante desvantagem: Como todo o conjunto de dados é aplicado para cálculo do gradiente, o tempo de treino da NN pode ser extremamente custoso para grandes conjuntos de dados.

O *Stochastic Gradient Descent* (SGD) é uma variação do *Gradient Descent*, também capaz de minimizar a função *loss* de um modelo de ML a partir de dados de treino. O termo *stochastic* (estocástico) se refere ao fato de que os gradientes do SGD possuem ruído em relação às variáveis de entrada, o que pode ser derivado inclusive do próprio conjunto de dados usados para o aprendizado (GOODFELLOW;

BENGIO; COURVILLE, 2016). O SGD é uma técnica de otimização, e não uma família de modelos (PEDREGOSA et al., 2011) embora ela e as várias técnicas derivadas dela, como a Adam, sejam possivelmente os algoritmos mais empregados para *Deep Learning* (GOODFELLOW; BENGIO; COURVILLE, 2016).

O SGD se difere do GD por usar uma instância aleatória do conjunto de dados, o que faz com que use um único ponto por vez para a otimização (PEDREGOSA et al., 2011). Por processar um conjunto muito reduzido de dados por etapa em relação a seu antecessor, o SGD é consideravelmente mais rápido e menos dispendioso computacionalmente (GÉRON, 2017). Isso também o torna menos propenso a ficar estagnado em mínimos locais.

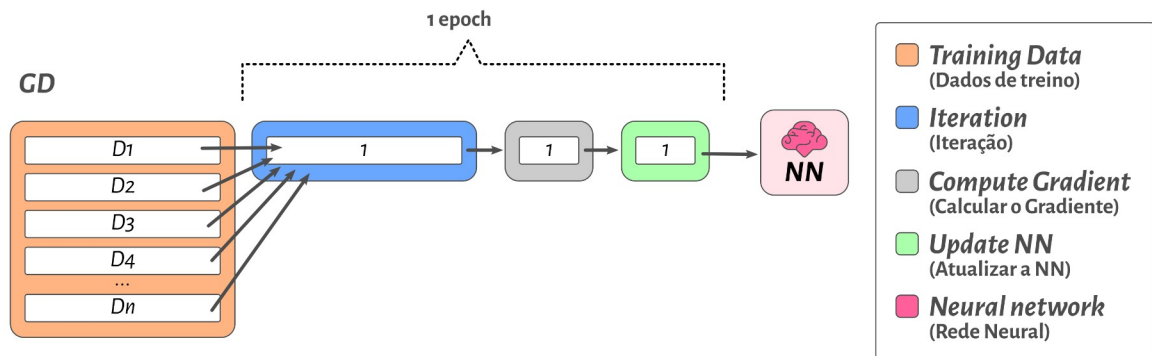
A estratégia baseada em um ponto por vez do SGD, porém, tem um custo: devido à natureza randômica com que o SGD foi concebido, nem todas as etapas de treino necessariamente reduzem o erro. Isto é: etapas subsequentes com aumento da *loss* não significam que o algoritmo esteja divergindo ou que o erro não cairá em iterações seguintes. Assim, é necessário não interromper o processo de treino prematuramente, mas aguardar o alcance de um mínimo global. Uma estratégia mais apropriada para verificar a impossibilidade do SGD de continuar a melhorar o modelo – e, portanto, indicar um possível ponto de “parada” do algoritmo - consiste em salvar o modelo em intervalos regulares (por exemplo, a cada 10.000 iterações) e medir a diferença de erro entre os melhores resultados ou a média dos resultados de ambos intervalos (GÉRON, 2017).

Dentre as principais vantagens do SGD, destacam-se sua eficiência e facilidade de implementação. Dentre as desvantagens, destaca-se a sensibilidade ao dimensionamento das variáveis de entrada – problema também apresentado pelo GD (PEDREGOSA et al., 2011). Este último problema pode ser amenizado através da normalização das variáveis de entrada, de forma que fiquem entre  $[0,1]$  ou entre  $[-1,+1]$ .

Como o SGD atualiza os pesos a cada ponto ou instância de treino, Se aquele ponto, em específico, for um conjunto de entradas e saídas que tenda a ser mais a ser uma exceção do que a ser regido pela mesma “regra” dos demais pontos, ele será, na prática, um ruído. Assim, ele irá aumentar o erro da rede ao invés de diminuir, dificultando o treino da capacidade de generalização da NN. Uma estratégia para contornar isso consiste no emprego de *mini-batches* (ou mini-bateladas). O *mini-batch GD* consiste em acumular o erro para uma parte ( $m$  ou

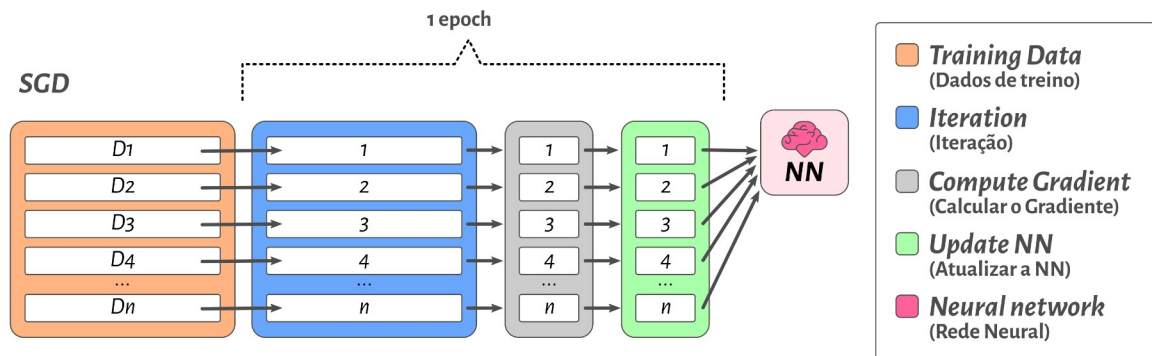
*batch-size*, tamanho da batelada) do conjunto de dados (C) e realizar uma única etapa de atualização dos pesos, considerando os resultados do subconjunto “n” (VASILEV et al., 2019). As Figuras 5 a 7 mostram um diagrama esquemático das diferenças entre as abordagens. Na Figura 7, o modelo de mini-batch foi exemplificado para um tamanho de batelada igual a 2 (dois).

Figura 5 - Esquema de Funcionamento do GD



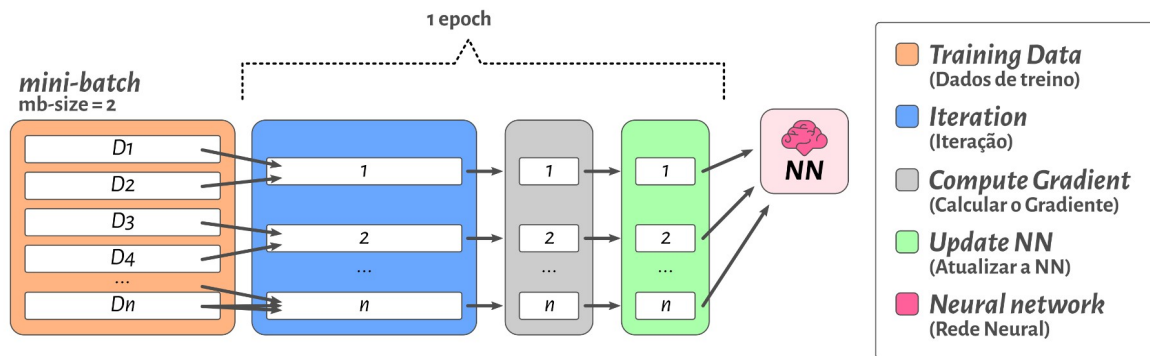
Fonte: Adaptado de (ANTONIADIS, 2023)

Figura 6 - Esquema de Funcionamento do SGD



Fonte: Adaptado de (ANTONIADIS, 2023)

Figura 7 - Esquema de Funcionamento do mini-batch



Fonte: Adaptado de (ANTONIADIS, 2023)

Em resumo:

- O GD emprega todos os pares de dados de treino em uma única etapa de iteração do algoritmo e atualização dos pesos a partir da média dos gradientes. O processo pode ser muito custoso para grandes conjuntos de dados;
- O SGD emprega os pares de dados de treino um por vez, calcula o gradiente e faz a atualização da rede. Se a rede possui  $N$  pares de dados no conjuntos de dados, para cada *epoch* serão realizadas  $N$  iterações. O uso de um único ponto de dados por vez implica em flutuações que podem inviabilizar o alcance do erro mínimo global;
- O mini-batch GD consiste em uma abordagem híbrida entre o GD e o SGD. Em vez de treinar a NN com um conjunto de dados por vez, o treino é feito com  $m$  (onde  $m$  é o tamanho da batelada, ou *batch-size*) instâncias de dados. Em um conjunto de dados com  $N$  pares de dados, uma *epoch* corresponde a  $N/m$  iterações.

O método de Newton é uma técnica de otimização de segunda ordem que emprega a matriz Hessiana. Um grande ponto fraco dessa abordagem é o elevado custo computacional do cálculo da inversa a matriz Hessiana, além de possíveis instabilidades na operação. Os chamados métodos *quasi-Newton* empregam o gradiente para aproximar a matriz Hessiana e sua inversa, o que os torna mais eficientes que o método de Newton tradicional. Os métodos BFGS e L-BFGS estão entre os métodos *quasi-Newton* de maior sucesso na otimização de NNs. O BFGS é



um algoritmo de otimização (BROWNLEE, 2021b) cujo nome deriva da abreviação dos nomes de seus quatro idealizadores – Broyden, Fletcher, Goldfarb e Shanno (NOCEDAL; WRIGHT, 2006). O L-BFGS é uma variação do BFGS, criada para contornar alguns dos problemas que serão abordados em breve. Os dois algoritmos conseguem atingir mínimos locais (BROWNLEE, 2021b) mas muitas vezes não são os mais indicados para encontrar mínimos globais. Assim, é comum que sejam empregados após uma etapa prévia com outro método de otimização (LU et al., 2021).

O BFGS armazena a matriz Hessiana e sua inversa a cada iteração. Assim, a memória ocupada pelo método cresce conforme o processo de otimização avança (NOCEDAL; WRIGHT, 2006; “Numerical Optimization”, 2014). Além disso, o tamanho da matriz é proporcional ao número de parâmetros de entrada. Embora o número de parâmetros de entrada não seja um problema tão aparente em PINNs – que normalmente possuem de 1 a 6 variáveis de entrada -, torna esse modelo proibitivo para outras áreas de *deep learning*, que podem possuir centenas ou milhares de parâmetros de entrada (BROWNLEE, 2021b). A explicação matemática do método é relativamente complexa e está disponível na literatura (NOCEDAL; WRIGHT, 2006).

O L-BFGS foi apresentado em 1995 (BYRD et al., 1995) como uma extensão do BFGS e busca resolver um grande problema do antecessor: seu consumo de memória crescente a cada iteração. Em vez de armazenar todos os valores anteriormente calculados para a matriz, o L-BFGS armazena apenas os  $m$  últimos pares de valores, necessários para determinar a curvatura da função e a direção de alteração dos pesos e *biases* para minimização da *loss*, implicitamente. A cada iteração, novos conjuntos de valores substituem os anteriores. Baixos valores de  $m$  (entre 3 e 20) são o suficiente para obtenção de bons modelos (NOCEDAL; WRIGHT, 2006). O algoritmo é executado não por um determinado número de iterações, como ocorre com o *Adam* e frequentemente com o SGD, mas sim até que um limite mínimo aceitável de erro seja alcançado.

O algoritmo Adam (do inglês *Adaptive Moment Estimation*, ou Estimativa de Momento Adaptativa) emprega várias ideias de outros algoritmos –como *Momentum* e *RMSProp*-, e é uma das grandes referências da atualidade em termos de otimização de NNs por seus bons resultados (GÉRON, 2017; KINGMA; BA, 2014). Considerada uma extensão do SGD (BROWNLEE, 2021a), o algoritmo consiste em

aplicar *momentum* (momento) na otimização. Para isso, são usados dois *momentums*: 1) um baseado no decaimento exponencial da média do gradiente e 2) um baseado no decaimento exponencial da variância para cada variável de entrada (KINGMA; BA, 2014). O método aplica fatores de correção para possibilitar a inicialização de ambos os momentos em zero (GOODFELLOW; BENGIO; COURVILLE, 2016).

O Adam é capaz de acelerar o processo de aprendizado por aplicar um *step size* (tamanho do passo de iteração – conceito análogo à LR) distinto para cada parâmetro de entrada. Cada *step size* é atualizada ao longo do processo de otimização, baseando-se nos gradientes de cada *input* (BROWNLEE, 2021a). Adam possui três principais hiperparâmetros (KINGMA; BA, 2014):

- $\alpha$ : O *step size* inicial. Normalmente seu valor é em torno de 0,001;
- $\beta_1$ : O fator de decaimento do primeiro momento. Tipicamente igual de 0,9.
- $\beta_2$ : O fator de decaimento do segundo momento. Tipicamente igual a 0,999.

A descrição matemática e lógica de Adam descrita a seguir é baseada no trabalho original (KINGMA; BA, 2014). Na primeira etapa, são inicializados o primeiro momento (Equação 13) e o segundo momento (Equação 14):

$$m_0 \leftarrow 0 \quad (13)$$

Em que  $m_0$  é o primeiro momento. A seta para a esquerda representa a atribuição em linguagem computacional.

$$v_0 \leftarrow 0 \quad (14)$$

Em que  $v_0$  é o segundo momento.

A primeira iteração é inicializada em  $t$  (número de iteração) igual a 0 (zero). É então criado um laço de repetição (*loop*) do tipo *while*, que cessará mediante uma condição (convergência dos parâmetros de entrada  $\theta$ ). O loop consiste em repetir as Equações 15 a 21:

$$t \leftarrow t + 1 \quad (15)$$

$$g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1}) \quad (16)$$

$$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \quad (17)$$

$$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2 \quad (18)$$

$$\hat{m}_t \leftarrow \frac{m_t}{1 - \beta_1^t} \quad (19)$$

$$\hat{v}_t \leftarrow \frac{v_t}{1 - \beta_2^t} \quad (20)$$

$$\theta_t \leftarrow \theta_{t-1} - \frac{\alpha \cdot \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \quad (21)$$

Em que os subscritos “t” e “t-1” indicam qual a iteração do respectivo valor marcado pelo subscrito;  $g_t$  armazena os gradientes com relação à função objetivo na iteração de número  $t$ ;  $m_t$  representa o primeiro momento, atualizado com o momento da etapa anterior (t-1);  $v_t$  representa o segundo momento, atualizado em relação ao segundo momento da etapa anterior (t-1);  $g_t^2$  indica a multiplicação, elemento por elemento, de  $g_t$  por  $g_t$ ;  $\theta_t$  corresponde ao conjunto de parâmetros que foram otimizados;  $\epsilon$  é um valor suficientemente pequeno (normalmente  $10^{-8}$ ) que impede a equação de gerar divisões por zero.

Os valores empregados na Equação 21 vem das Equações 19 e 20 e não de 17 e 18 por para que compensem a inicialização em 0 das Equações 13 e 14. Sem essa etapa, os valores seriam enviesados no sentido de 0 pela etapa de inicialização. Após o laço de repetição ser concluído, o valor de  $\theta_t$  é então retornado para a função inicial, encerrando o processo.

O Adam requer relativamente pouca memória (KINGMA; BA, 2014) e é bastante robusto, aceitando diversas faixas de hiperparâmetros (GOODFELLOW; BENGIO; COURVILLE, 2016), o que também contribui para que seja hoje um das ferramentas mais populares de otimização de NNs. A descrição matemática completa de Adam está disponível no trabalho original (KINGMA; BA, 2014).

### 3.1.1.6 Funções de Ativação

A modificação não linear feita nos resultados de saída de cada LTU é chamada de função de ativação. Para atender às necessidades específicas de distintos problemas de classificação e/ou regressão, diversas funções de ativação foram introduzidas ao longo dos anos. A função de ativação empregada e suas particularidades impactam fortemente o treinamento e performance da rede neural (RASAMOELINA et al., 2022). O termo “y” é usado aqui para designar a função de ativação abordada em cada tópico. As principais são definidas a seguir e o Gráfico 1 exibe o valor de saída de cada função de ativação em função da entrada (z):

- **Logística (logistic ou sigmoid):** É a forma mais clássica, e a introduzida originalmente no algoritmo de *back-propagation*, citada na seção 3.1.1.4 e representada pela Equação 22. O gráfico desta função exibe uma curva em “S”, com o valor mínimo 0 ( $y \rightarrow 0$  para  $z \rightarrow -\infty$ ) e o valor máximo +1 ( $y \rightarrow 1$  para  $z \rightarrow +\infty$ ) (GÉRON, 2017). A *sigmoid* pode ser inapropriada para redes com muitas camadas, por conta do ser valor médio (GLOROT; BENGIO, 2010).

$$\text{sigmoid}(z) = \frac{1}{1 + e^{-z}} \quad (22)$$

- **Tangente Hiperbólica (tanh):** O gráfico desta função também possui forma de “S”, mas se diferencia da logística por ter o valor mínimo de -1, e não 0. Por produzir valores sempre entre -1 e 1 ( $y \rightarrow -1$  para  $z \rightarrow -\infty$  e  $y \rightarrow +1$  para  $z \rightarrow +\infty$ ), a função tende a produzir resultados mais normalizados (pelo “centro” da função ser em zero) e pode promover a convergência mais rapidamente (GÉRON, 2017; PANNEERSELVAM, 2021). Contudo, é essa característica de normalizar os valores dentro de um pequeno limite (entre -1 e +1) que causa um dos erros mais icônicos dessa função de ativação. Especialmente em redes muito profundas (com muitas camadas), observa-se o chamado *vanishing gradient* (WANG et al., 2019) - quando os valores de atualização (da função erro) da rede são tão baixos que, eventualmente, a rede neural se torna incapaz de reduzir o erro e o aprendizado é estagnado. A função tangente hiperbólica é dada pela Equação 23:

$$\tanh(z) = \frac{2}{1 + e^{-2z}} - 1 \quad (23)$$

- **ReLU:** Um dos grandes desafios dessa função ocorre em virtude de ser contínua, mas não diferenciável em  $z = 0$ , o que faz com que o *Gradient Descent* não funcione adequadamente em valores baixos. A função ReLU (do inglês *Rectified Linear Units*, ou Unidades Lineares Retificadas) basicamente “filtra” os valores de entrada: todos os valores acima de zero geram o próprio valor, enquanto todos os valores iguais ou abaixo de zero retornam zero, conforme indicado na Equação 24 (RASAMOELINA; ADJAILIA; SINCAK, 2020). Duas vantagens dessa função são o fato de ser computacionalmente menos custosa, além de que, por não determinar valores de topo, evita problemas do tipo *vanishing gradient* (GÉRON, 2017). Contudo, por não limitar valores de topo, pode gerar problemas do tipo *exploding gradient*, quando os valores de atualização do erro são tão altos que a rede se desestabiliza e é incapaz de reduzir o erro nas iterações consecutivas, gerando valores que ultrapassam o máximo do ponto flutuante permitido pela linguagem de programação empregada.

$$ReLU(z) = \max(0, z) \quad (24)$$

- **SELU:** O nome vem do inglês *Scaled Exponential Linear Unit*, ou Unidade Linear Exponencialmente Escalonada. A ideia principal do emprego de SELU é construir Redes Neurais Auto-Normalizadas (SNNs, do inglês *Self-normalizing Neural Networks*). Dentre os requisitos para tal feito, a função de ativação deveria necessariamente 1) poder gerar valores positivos e negativos (para controlar a média do erro), 2) ter regiões de saturação (onde o valor da derivada se aproxima de zero), 3) possuir inclinação superior à variância e 4) possuir uma curva contínua. Dessa forma, a função SELU induz a estabilização de variância, de modo a evitar os problemas *exploding* e *vanishing gradients* (KLAMBAUER et al., 2017). Na prática, isso significa que teoricamente redes com SELU podem possuir mais camadas que redes ativadas por ReLU. Há registros na literatura de que a SELU pode contribuir para uma convergência mais rápida em relação à ReLU (RASAMOELINA et al., 2022). A SELU é definida na Equação 25 (KLAMBAUER et al., 2017).

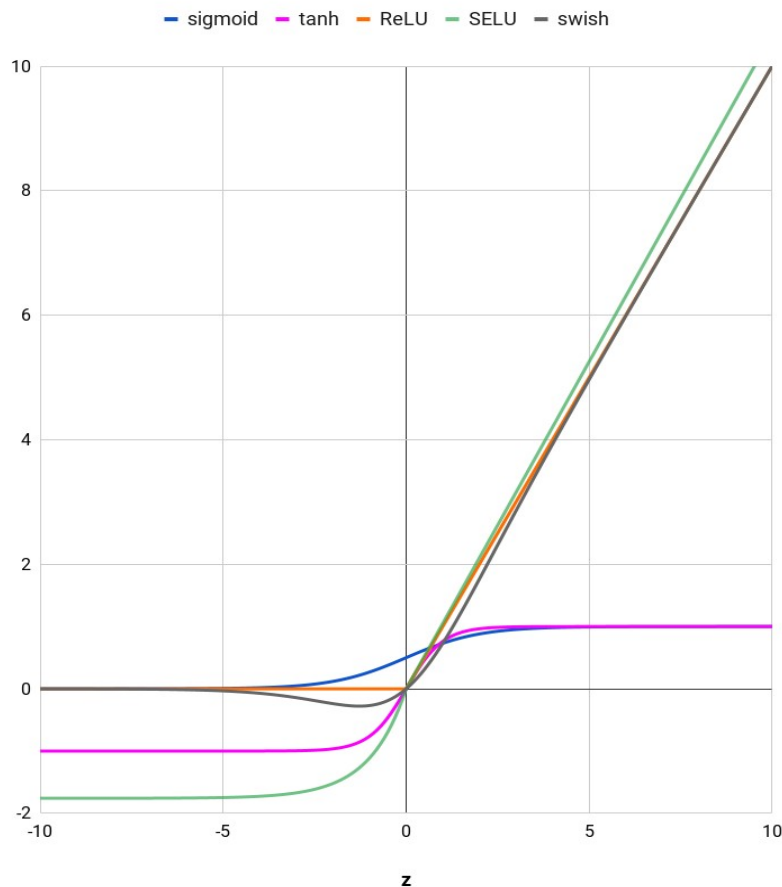
$$SELU(z) = \lambda \begin{cases} z & \text{se } z > 0 \\ \alpha e^z - \alpha & \text{se } z \leq 0 \end{cases} \quad (25)$$

Em que  $\alpha \approx 1,6733$  e  $\lambda \approx 1,0507$ .

- **Swish:** Assim como a função ReLU, a swish pode atingir qualquer valor máximo (não é restringida), mas possui restrições para os valores mínimos que alcança (RAMACHANDRAN; ZOPH; LE, 2017). A função swish é normalmente usada em aplicações complexas, como tradução, e que possuem mais de 40 camadas (BAHETI, [s.d.]). A função swish é definida pela Equação 26 (RASAMOELINA et al., 2022):

$$\text{swish}(z) = \frac{z}{1 + e^{-z}} = z * \text{sigmoid}(z) \quad (26)$$

Gráfico 1 - Funções de ativação em função do valor de z



Fonte: Autoria Própria (2023)

### 3.1.1.7 Funções de Inicialização

Na seção 3.1.1.4, foi discutida a importância da estrutura da rede e do processo de otimização ou treinamento. Esse processo de treino começa a partir da estrutura inicial da rede, e consiste em atualizar os pesos e *biases* para otimizar a rede para a representação do problema em questão. Contudo, a depender dos valores iniciais desses parâmetros, pode ser que o sistema seja incapaz de atingir um valor de mínimo global, atingindo portanto um erro mínimo local (SUTSKEVER; MARTENS; DAHL, 2013). Assim, os valores iniciais dos pesos podem influenciar profundamente nos resultados produzidos pela Rede Neural Artificial (NARKHEDE; BARTAKKE; SUTAONE, 2022). Para contornar esse problema, faz-se necessário o uso das chamadas funções de inicialização, que são responsáveis por definir os valores de pesos e *biases* na rede antes do início do processo de otimização (GÉRON, 2017).

Foi a partir do ano de 2006 que as NNs profundas começaram a se tornar mais difundidas, e isso se deu pelo sucesso da implementação de estratégias de inicialização ou treino (GLOROT; BENGIO, 2010). O objetivo de inicializadores (ou funções de inicialização) é determinar o estado inicial da NN em um bom ponto de erro mínimo local (DAS; BHARGAVA; PORIKLI, 2021). Com a escolha de um bom ponto local, é mais provável que o mínimo global seja alcançado. Além disso, a inicialização correta de pesos é uma das maneiras mais efetivas de se acelerar a etapa de treino da NN (YAM et al., 2002).

Ao longo dos anos de desenvolvimento das NNs e conforme o interesse nelas aumentou, novos algoritmos e técnicas de inicialização foram criadas. As técnicas de inicialização se dividem em basicamente duas categorias: 1) as sem treino prévio – do inglês *pre-training* - (que se subdividem ainda em com inicialização randômica e em com inicialização orientada a dados [ou *data-driven*]) e 2) as técnicas com treino prévio (NARKHEDE; BARTAKKE; SUTAONE, 2022).

Os principais tipos de técnicas de inicialização randômica são a baseada em intervalos (do inglês *Interval based initialization*), a com escalonamento baseado em variância (do inglês *Variance scaling based initialization*) e a abordagem híbrida. A inicialização baseada em intervalos consiste em atribuir os valores de forma randômica. O requisito para tais valores de pesos é estar dentro uma determinada faixa pré-determinada. A inicialização com escalonamento baseado em variância consiste em selecionar os pesos, a um primeiro momento, de forma randômica. Na sequência, eles são modificados de forma que a variância entre as camadas de

entrada e de saída, ou ainda a variância dos gradientes da rede, é mantida a um determinado valor, pré-determinado (NARKHEDE; BARTAKKE; SUTAONE, 2022).

As técnicas de inicialização *data-driven* contrastam por empregarem dados para uma inicialização mais robusta dos parâmetros da NN (DAS; BHALGAT; PORIKLI, 2021). Nelas, o peso das conexões entre camadas é diretamente derivado do conjunto de dados de treino. Uma miríade de técnicas *data-driven* já foi publicada na literatura científica (ALBERTI et al., 2017; GAN et al., 2015; KRÄHENBÜHL et al., 2015; LEHTOKANGAS; SAARINEN, 1998; YAM et al., 2002). De forma geral, essas técnicas convergem de forma mais rápida ou a resultados melhores em comparação com técnicas mais tradicionais (NARKHEDE; BARTAKKE; SUTAONE, 2022). Esse tipo de inicialização pode melhorar consideravelmente tanto a etapa de treino quanto o uso posterior da NN (DAS; BHALGAT; PORIKLI, 2021).

A função de inicialização de Xavier (do inglês *Xavier initialization*) foi introduzida em 2010 (GLOROT; BENGIO, 2010) e faz o ajuste dos pesos a partir da variância (DAS; BHALGAT; PORIKLI, 2021). A função de inicialização de Xavier também é conhecida como função de Glorot (TENSORFLOW, [s.d.]) e é uma das funções de ativação mais empregadas atualmente (GÉRON, 2017). Seu objetivo é inicializar os pesos de tal forma que a variância das funções de ativação (saída de cada LTU) seja a mesma em cada camada. Essa variância constante ajuda a evitar problemas do tipo *exploding gradient* e *vanishing gradient* (NG; KATANFOROOSH, 2022). Na prática, o método consiste na inicialização randômica empregando a Equação 27 para distribuições normais com média 0 e a Equação 28 para distribuições uniformes entre -r e +r. Nas duas equações,  $n_{in}$  é o número de conexões de entrada e  $n_{out}$  o número de conexões de saída (GÉRON, 2017).

$$\sigma = \sqrt{\frac{2}{n_{in} + n_{out}}} \quad (27)$$

Em que  $\sigma$  é a variância

$$r = \sqrt{\frac{6}{n_{in} + n_{out}}} \quad (28)$$

A dedução matemática da inicialização de Glorot ou Xavier é dada pelas Equações 29 a 42, e considera, inicialmente, um regime com função de ativação



linear (Glorot; Bengio, 2010; NG; Katanforoosh, 2022). O procedimento aqui exemplificado assume quatro hipóteses:

- Os pesos e entradas possuem centro em zero;
- Os pesos e entradas são independentes e igualmente distribuídos;
- Os *biases* são inicializados como zero;
- É empregada a função de ativação tanh (tangente hiperbólica), que é aproximadamente linear em valores de entrada ( $z$ ) relativamente pequenos.

$$W_{i,j} \sim U\left[\frac{-1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right] \quad (29)$$

Em que  $W$  é a matriz de pesos,  $W_{ij}$  é o peso de cada LTU e camada,  $U[-a, a]$  é a distribuição uniforme no intervalo  $(-a, a)$  e  $n$  é o tamanho da camada anterior (número de colunas de  $W$ ).

$$\frac{\partial Loss}{\partial s_k^i} = f'(s_k^i) W_k^{i+1} \frac{\partial Loss}{\partial s^{i+1}} \quad (30)$$

$$\frac{\partial Loss}{\partial w_{l,k}^i} = z_l^i \frac{\partial loss}{\partial s_k^i} \quad (31)$$

Em que  $f$  é uma função de ativação simétrica e com derivada igual à unidade em zero ( $f'(0)=1$ ),  $z^i$  é o vetor de ativação da camada  $i$ , e  $s^i$  é o vetor de parâmetros da função de ativação na camada  $i$ , de forma que  $s^i = z^i W^i + b^i$  e  $z^{i+1} = f(s^i)$ .

A hipótese de que a função tem comportamento linear na inicialização, que os pesos foram inicializados de forma independente e de que a variância dos parâmetros de entrada é igual foi empregada para, então, chegar na Equação 32.

$$Var[z^i] = Var[x] \prod_{i'=0}^{i-1} n_{i'} Var[W^{i'}] \quad (32)$$

Em que  $Var[a]$  é a variância de um termo “ $a$ ”,  $x$  representa os parâmetros de entrada e  $n_i$  é o tamanho da camada  $i$ .

A variância  $Var[W^i]$  corresponde à variância compartilhada por todos os pesos na camada  $i$ . Para uma rede com  $d$  camadas, são dadas as Equações 33 e 34:

$$\text{Var}\left[\frac{\partial \text{Loss}}{\partial s^i}\right] = \text{Var}\left[\frac{\partial \text{Loss}}{\partial s^d}\right] \prod_{i'=1}^d n_{i'+1} \text{Var}[W^{i'}] \quad (33)$$

$$\text{Var}\left[\frac{\partial \text{Loss}}{\partial s^i}\right] = \text{Var}\left[\frac{\partial \text{Loss}}{\partial s^d}\right] \prod_{i'=0}^{i-1} n_{i'} \text{Var}[W^{i'}] \prod_{i'=i}^{d-1} n_{i'+1} \text{Var}[W^{i'}] \times \text{Var}[x] \text{Var}\left[\frac{\partial \text{loss}}{\partial s^d}\right] \quad (34)$$

Considerando um mecanismo *back-propagation*, o fluxo de atualização é dado pela Equação 35:

$$\forall (i, i'), \text{Var}\left[\frac{\partial \text{loss}}{\partial s^i}\right] = \text{Var}\left[\frac{\partial \text{loss}}{\partial s^{i'}}\right] \quad (35)$$

As condições apresentadas pela Equação 35 podem, então, ser expressas como as Equações 36 e 37:

$$\forall i, n_i \text{Var}[W^i] = 1 \quad (36)$$

$$\forall i, n_{i+1} \text{Var}[W^i] = 1 \quad (37)$$

Para tornar ambas as condições verdadeiras, podemos determinar a variância  $\text{Var}[W]$  conforme a Equação 38:

$$\forall i, \text{Var}[W^i] = \frac{2}{n_i + n_{i+1}} \quad (38)$$

As condições são satisfeitas para o caso em que todas as camadas tenham a mesma largura (quantidade de neurônios ou LTUs por camada). Se além de ter a mesma largura, for adotada a mesma inicialização para todos os pesos, as Equações 39 e 40 também passam a ser válidas:

$$\forall i, \text{Var}\left[\frac{\partial \text{loss}}{\partial s^i}\right] = [n \text{Var}[W]]^{d-i} \text{Var}[x] \quad (39)$$

$$\forall i, \text{Var}\left[\frac{\partial \text{loss}}{\partial w^i}\right] = [n \text{Var}[W]]^d \text{Var}[x] \text{Var}\left[\frac{\partial \text{loss}}{\partial s^d}\right] \quad (40)$$

Dessa forma, a variância do gradiente e dos pesos é a mesma para todas as camadas. Contudo, a variância do gradiente aplicando *back-propagation* ainda pode dar origem a *vanishing* ou *exploding gradients*, sobretudo em redes muito profundas.

Aplicando a inicialização padrão (Equação 29), obtém-se, a variância indicada pela Equação 41:

$$n\text{Var}[W] = \frac{1}{3} \quad (41)$$

A normalização é de grande importância, pois efeitos de desestabilização se espalham rapidamente pela NN pela natureza multiplicativa do processo. Os autores do trabalho original (Glorot; Bengio, 2010) recomendaram, por fim, o emprego de uma inicialização normalizada e capaz de atender, ainda que de forma aproximada, os critérios mencionados nesta seção (representado na Equação 42, que é equivalente à Equação 28).

$$W \sim U \left[ \frac{-\sqrt{6}}{\sqrt{n_j + n_{j_i}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}} \right] \quad (42)$$

A função de Xavier recomendada para a inicialização de NNs é dada na Equação 43. Todos os *bíases* são inicializados como zero, conforme indicado pela Equação 44 (Katanforoosh; Kunin, 2019).

$$W^l = U \left( \mu = 0, \sigma^2 = \frac{1}{n^{[l-1]}} \right) \quad (43)$$

Em que  $\mu$  é a média da distribuição normal,  $\sigma^2$  é a variância e  $n^{[l-1]}$  é o número do neurônio na camada  $l-1$ .

$$b^{[l]} = 0 \quad (44)$$

Em que  $b$  é o *bias*.

A ideia principal para prevenir *vanishing* e *exploding gradients* através de funções de inicialização consiste em 1) garantir que a média da função de ativação seja zero e 2) garantir que a variância das funções de ativação permaneça a mesma ao longo de todas as camadas (Katanforoosh; Kunin, 2019). A função de Glorot busca, ainda, evitar a saturação excessiva de funções de ativação (o que levaria a *vanishing gradients*) e funções muito lineares (que possuem limitações e muitas vezes são incapazes de produzir bons resultados). Glorot e Bengio (2010)

determinaram que a função de ativação tanh, de forma geral, apresentou melhores resultados do que a sigmoid, por sua simetria em zero, validando a ideia da hipótese 1). O emprego da Equação 29, contudo, apresentou o fenômeno de saturação mesmo com o uso de tanh como função de ativação. O problema se iniciou na primeira camada e foi se propagando para as demais da NN. O estudo original (GLOROT; BENGIO, 2010) foi incapaz de responder o porquê desse comportamento.

### 3.1.1.8 *Vanishing e exploding gradients*

Um dos grandes obstáculos ao se trabalhar com Redes Neurais ao longo da história tem sido não só como determinar o número ótimo de neurônios, camadas ou a função de ativação ideal, mas também como contornar dois grandes problemas já citados nas seções anteriores: *vanishing* e *exploding gradients*. Essas ocorrências são problemas relativamente graves, uma vez que praticamente inibem o aprendizado da NN, e ainda são motivos de discussão e busca por técnicas que possam contorná-las.

Conforme discutido na seção 3.1.1.4, o mecanismo de atualização de pesos e *biases* por *back-propagation* propaga o gradiente do erro (função *loss*) da camada mais externa (*output*) até a camada de entrada (*input layer*). Conforme a *propagation* avança do longo das camadas, o gradiente se torna cada vez menor ao decair exponencialmente, reduzindo consequentemente a alteração que é feita nos pesos. Por conta disso, as camadas das regiões iniciais são desfavorecidas no processo de atualização, podendo permanecer praticamente inalteradas e impossibilitando a NN de ser adequadamente treinada. A esse fenômeno em que o gradiente se reduz a ponto de praticamente não existir, atribui-se o nome de VG (*Vanishing gradients* - em tradução livre, Gradientes que Desaparecem) (GÉRON, 2017).

Os VGs, dificultam não só a determinação do valor de atualização, mas também a direção da atualização (i.e. se os pesos devem aumentar ou diminuir numericamente) (GOODFELLOW; BENGIO; COURVILLE, 2016). Pelo tamanho reduzido e por acontecer em etapas posteriores, os VGs podem ainda contribuir para que os valores mais recentes (das últimas iterações) se sobreponham aos das

mais antigas, o que faz com que a informação que foi alimentada para o treino da rede seja, do ponto de vista prático, parcialmente inutilizada. É um problema relativamente difícil de detectar, pois, ao contrário do que se observa em *exploding gradients*, a NN continua a produzir resultados tecnicamente válidos (VASILEV et al., 2019).

Um exemplo concreto de um modelo que pode favorecer a presença de VGs é quando, devido à semi-normalização causada pela função de ativação (como tanh), valores muito altos ou muito baixos acabam sendo representados pelos valores de topo e de fundo (no caso da função de ativação tanh [tangente hiperbólica], +1 e -1) (WANG et al., 2019). Assim, há uma saturação de valores nessa região e, na prática, o treinamento das camadas iniciais é tão lento e requer tantas iterações que pode, na prática, ser inviável ou quase impossível de obter a convergência devido ao custo computacional. Os VGs já foram observados empiricamente em outros trabalhos, e foram uma das razões (dentre outras discutidas na seção 3.1) pelas quais as NNs, em seus primórdios, perderam um pouco de interesse (GÉRON, 2017). Os VGs puderam ser muito minimizados com a aplicação das recomendações envolvidas no trabalho que deu origem à inicialização de Glorot, descritas na seção 3.1.1.5 (GLOROT; BENGIO, 2010). Nele, foi determinado que dentre as causas que levam a *vanishing gradients*, destacam-se o uso da *sigmoid* como função de ativação combinado à inicialização usando uma distribuição normal com média 0 e desvio padrão 1.

De forma análoga ao comportamento que dá origem ao VG, o EG (Gradiente Explosivo, do inglês *Exploding gradient*) é observado quando os valores são excessivamente grandes. Em vez de resultar em valores de atualização cada vez menores conforme o *back-propagation* avança das camadas de saída para as de entrada, observa-se valores cada vez maiores, tornando a função de treino ou aprendizado instável (VASILEV et al., 2019). Essa instabilidade leva a valores extremamente grandes de atualização, o que por sua vez pode contribuir para manter o treino ainda instável e, quase sempre, levar a números tão elevados que ultrapassam o máximo permitido pela linguagem de programação empregada (LI, 2022).

Em Python, os números de ponto flutuante do tipo *float* podem ter valor máximo de aproximadamente  $1,79 \cdot 10^{308}$ . Quando esse valor é ultrapassado, o sistema gera os chamados NaNs (do inglês *Not a Number*, ou Não é um Número).

Como NaNs não podem ser representados graficamente, normalmente em casos que exibem o erro ao longo de iterações (ou epochs) é comum que seja verificado um aumento súbito do erro, seguido da ausência de valores para as iterações seguintes (por serem NaN e, portanto, impossível de plotá-los graficamente).

A alteração na taxa de aprendizado (LR) pode auxiliar ambos os problemas relacionados aqui citados, pois ajuda a balizar a etapa de atualização. Contudo, EGs são naturalmente mais complicados pois, ao passo em que os valores muito pequenos de VGs podem ser multiplicados por LRs altas para torná-los relevantes, não há como corrigir tão facilmente EGs que tenham atingido valores praticamente infinitos ou NaN. Uma possibilidade para minimização dos EGs envolve o chamado *gradient clipping*, que consiste em determinar valores máximos do gradiente e limitá-lo antes da etapa de atualização dos pesos (GOODFELLOW; BENGIO; COURVILLE, 2016).

Embora a aplicação das recomendações de Glorot e Bengio minimize o aparecimento de VGs e EGs no início do processo de treino, esses problemas ainda podem surgir conforme o treino da NN avança. A chamada *Batch normalization* (normalização em batelada) é capaz de estabilizar o treino para impedir esses fenômenos que afetam negativamente o aprendizado (GÉRON, 2017).

### 3.1.1.9 Batch Normalization

O treino de DNNs, sobretudo devido aos problemas com VGs e EGs, normalmente requer maior atenção na escolha de taxas de aprendizado (LR) e no método de inicialização de parâmetros, o que torna o processo mais custoso computacionalmente e mais lento. Essas modificações para prevenir desvios tornam mais propenso a falhas o treino de redes com saturação em não-linearidades (i.e. termos que alcançam valores de máximo ou mínimo e estagnam de forma não-linear). Esse fenômeno foi nomeado como ICS (Deslocação de Covariável Interna, do inglês *Internal Covariate Shift*) (IOFFE; SZEGEDY, 2015).

Uma das suposições dos métodos de otimização por gradientes é que, de forma geral, para cada atualização, as demais camadas não mudam. Ou seja, cada parâmetro é atualizado considerando que os demais não fossem (GOODFELLOW;

BENGIO; COURVILLE, 2016). Contudo, como essa mudança acontece, a otimização está sempre buscando um ponto em movimento, pois a cada atualização as demais camadas mudaram. Essa é uma outra definição, um pouco mais simples, do que é o ICS.

A BN (Normalização em Batelada, do inglês *Batch Normalization*) é uma técnica que permite o uso de LRs muito maiores (o que acelera o aprendizado) e mais tolerância a diferentes métodos de inicialização (IOFFE; SZEGEDY, 2015). A BN consiste em uma maneira elegante de re-parametrizar uma DNN (BROWNLEE, 2019; GOODFELLOW; BENGIO; COURVILLE, 2016). A forma de operacionalização da BN é dada pelas Equações 45 a 48 (GÉRON, 2017):

$$\mu_B = \frac{1}{m_B} \sum_{i=1}^{m_B} x^{(i)} \quad (45)$$

$$\sigma_B^2 = \frac{1}{m_B} \sum_{i=1}^{m_B} (x^{(i)} - \mu_B)^2 \quad (46)$$

$$\hat{x}^{(i)} = \frac{x^{(i)} - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \quad (47)$$

$$z^{(i)} = y \hat{x}^{(i)} + \beta \quad (48)$$

Em que  $\mu_B$  é a média empírica (calculada empregando toda a *mini-batch* B),  $\sigma_B$  é o desvio padrão empírico (calculado empregando toda a *mini-batch* B),  $m_B$  é o número de instâncias ou pontos na *mini-batch*,  $\hat{x}^{(i)}$  são os parâmetros de entrada normalizados e com centro em zero,  $y$  é o parâmetro de escala da camada,  $\beta$  é o parâmetro de deslocamento (*shifting* ou *offset*) da camada,  $\epsilon$  é um parâmetro para evitar possíveis divisões por zero (normalmente  $\epsilon \sim 10^{-3}$ ) e  $z^{(i)}$  é a saída da BN (a entrada em escala e deslocada).

Ioffe e Szegedy (2015) confirmaram a robustez da BN. Os autores foram capazes de reduzir o número de iterações necessárias em 14 vezes para um algoritmo de classificação de imagens, que ainda assim performou melhor que a versão original. Na época, o feito foi validado, ainda, por apresentar uma performance de validação (erro de 4.9%) superior à de testes feitos diretamente por humanos. A BN atua, ainda, como um regularizador, reduzindo a necessidade de

outras técnicas de regularização, como *Dropout* (GÉRON, 2017; IOFFE; SZEGEDY, 2015).

Contudo, a BN tem algumas desvantagens bastante relevantes. Além de adicionar complexidade ao modelo, a rede gerada faz previsões de forma mais lenta do que redes sem BN (o que é chamado de *Runtime Penalty*, ou penalidade em tempo de execução) devido ao esforço computacional adicional necessário. Para casos em que a velocidade de predição seja de grande importância, portanto, pode ser mais interessante aplicar a inicialização de Glorot a redes tradicionais em detrimento da BN (GÉRON, 2017).

#### 3.1.1.10 Técnicas de regularização

As técnicas de regularização são um conjunto de abordagens que visam evitar o *overfitting* de redes neurais. O *overfitting* acontece quando a rede consegue prever com extrema precisão dados do grupo de treino (*domain*) mas falha com dados de teste (*test*). Na prática, isso significa que a rede está superajustada e não é capaz executar o que talvez seja a principal função de uma rede neural: a generalização de valores de saída com base em valores de entrada. Dentre as técnicas de regularização, destacam-se a *Early Stopping*, a regularização em L e o *Dropout*. O *Dropout* é um dos casos de maior sucesso, e é capaz de melhorar facilmente de 1 a 3% a precisão de algoritmos de classificação, mesmo em casos já bem otimizados (ZHANG; ZHOU; XU, 2021).

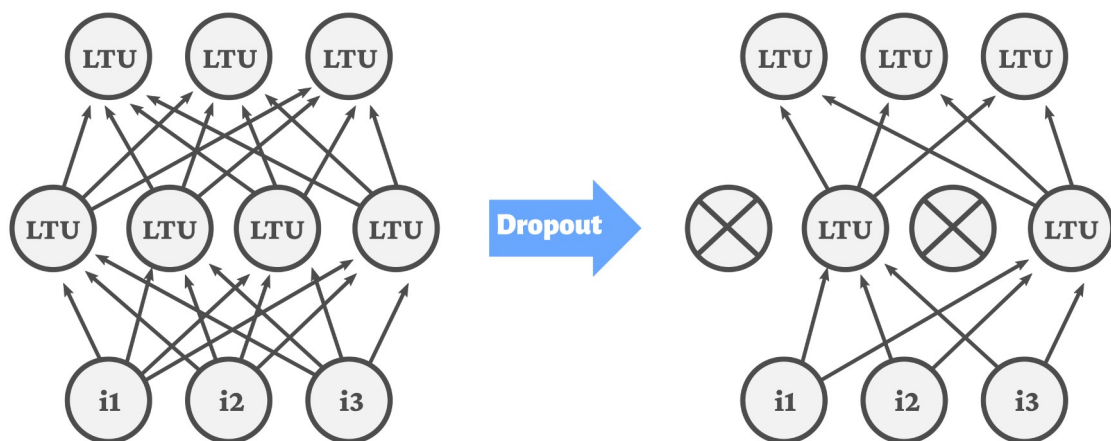
A *Early Stopping* (Parada prematura) consiste em interromper o treino da rede neural. Uma maneira de implementá-la consiste em avaliar o erro da predição da rede a cada número de iterações (por exemplo, a cada 100 *epochs*) e comparar o erro ou *loss* para os dados de treino e os dados de teste. A partir do momento em que o sistema apresentar um desvio consistente, com os resultados de teste apresentando erros maiores que os resultados de treino, é necessário parar o processo de otimização, para evitar o *overfitting*. Embora seja uma técnica relativamente bruta, consegue produzir resultados interessantes. A performance da NN no geral é melhor caso a *Early Stopping* seja combinada com outras técnicas de regularização.



O *Dropout* consiste em, de forma randômica e periódica, desativar uma parte dos neurônios (bem como suas conexões de entrada e de saída), com base em uma probabilidade  $p$ , que vai de 0 a 100 (VASILEV et al., 2019). A cada iteração ou epoch de treino, um conjunto de neurônios (aproximadamente  $p \cdot N/100$ , onde  $N$  é o número total de neurônios) é desativado, e os demais são treinados de forma independente. Essa técnica favorece que os neurônios não se tornem dependentes exclusivamente de um pequeno grupo de neurônios preferenciais como fonte de entrada de informações, mas que consigam cooperar adequadamente com os demais. Além disso, favorece uma certa independência de cada neurônio, tornando-o mais robusto a alterações nos parâmetros de entrada e, conseqüentemente, tornando a NN produzida mais robusta (SRIVASTAVA et al., 2014). A Figura 8 exibe um esquema que exemplifica um *Dropout*. Os círculos com um “X” representam os neurônios que foram desativados (ou *dropados*).

Embora o *Dropout* reduza a taxa de convergência (no sentido de que requer mais iterações para produzir um modelo com baixo erro), a técnica costuma produzir modelos muito melhores e que justificam o maior custo computacional. O Dropout é uma das técnicas de regularização mais populares atualmente por reduzir com efetividade o problema de *overfitting* (GÉRON, 2017; ZHANG; ZHOU; XU, 2021).

Figura 8 - *Dropout* em rede neural com uma *hidden layer*



Fonte: Adaptado de (SRIVASTAVA et al., 2014)

### 3.1.2 Aplicação em bioprocessos

Bioprocessos naturais são, em geral, extraordinariamente complexos e, sob alguns aspectos, impossíveis de modelar e prever. Quando termo “bioprocessos” se refere a processos produzidos pelo homem, contudo, a descrição é um pouco diferente. Nesse contexto, bioprocessos são processos que empregam seres vivos de qualquer natureza ou produtos produzidos por eles (células, enzimas, vírus, etc) para a produção, modificação ou tratamento de determinada substância ou conjunto de substância e que foi realizado propositalmente por ação humana. Dentre os bioprocessos, destacam-se os processos fermentativos, presentes na produção de diversos bens de consumo (como iogurtes, cervejas e queijos) e tratamento de efluentes (LIM et al., 2023).

A representação de bioprocessos através de modelos matemáticos é um desafio que vem sendo superado já há algum tempo (DORAN, 2013), mas ainda existem modelos muito complexos ou microrganismos cuja cinética de crescimento, consumo de substrato ou geração de produto não é representada de forma simples pelas equações comumente empregadas e suas variantes. Os modelos matemáticos que apresentam boa performance na representação desses processos podem ser empregados posteriormente na simulação do processo, com o intuito de reduzir custos, melhorar o impacto ambiental ou reduzir o tempo de produção. Uma alternativa ao uso de modelos matemáticos é o emprego de ANN para a predição das variáveis de saída de interesse (como concentração de células ou de produto) com relação às variáveis de entrada (como o tempo decorrido desde o início do experimento). Contudo, essa alternativa é bastante custosa por causa das exigências em relação à quantidade e qualidade dos dados citada anteriormente.

Diversos modelos de ML já foram aplicados nos mais variados bioprocessos, mas muitos não conseguiram prever adequadamente as variáveis de interesse (LIM et al., 2023). Por conta disso, os métodos de simulação numérico computacionais continuam sendo considerados uma ferramenta bastante robusta e até mesmo a mais adequada para diversos casos, uma vez que não dependem da disponibilidade de *big data*. Contudo, métodos numéricos podem depender significativamente das estratégias de discretização empregadas, e é praticamente impossível dissociar o método de solução numérico do método de discretização. Assim, é difícil comparar diferentes métodos numéricos sem levar em consideração a estratégia de discretização adotada. Outro ponto relevante é que o uso de malhas muito finas (aumentar o número de pontos de discretização) nem sempre resolvem o

problema encontrado em malhas menos finas, bem como podem acarretar em grandes tempos de execução do programa – reflexo do grande esforço computacional necessário. É nesse cenário que surgem as *Physics-Informed Neural Networks*, um híbrido entre redes neurais e solução de modelos matemáticos.

### 3.2 Physics-Informed Neural Network

PINN (Physics-Informed Neural Network) nasce em um contexto onde diversas técnicas de ML não conseguem convergir a resultados apropriados devido à falta da chamada *big data* (grande quantidade de dados). Introduzido à comunidade científica em 2018, o *framework* se vale da otimização do cálculo derivadas graças ao uso de GPUs por ANN para aplicá-las à derivação de equações e expressões matemáticas (RAISSI; PERDIKARIS; KARNIADAKIS, 2019). Embora o trabalho de 2019 tenha sido o grande responsável pela popularização do conceito, um outro estudo, publicado em 1994, abordou conceitos muito semelhantes aos princípios do PINN, e pode ser considerado sob essa ótica o pioneiro da área (DISSANAYAKE; PHAN-THIEN, 1994). Toda a descrição matemática desta seção se baseia no trabalho de Raissi, Perdikaris e Karniadakis (2019) salvo quando explicitamente indicado o contrário.

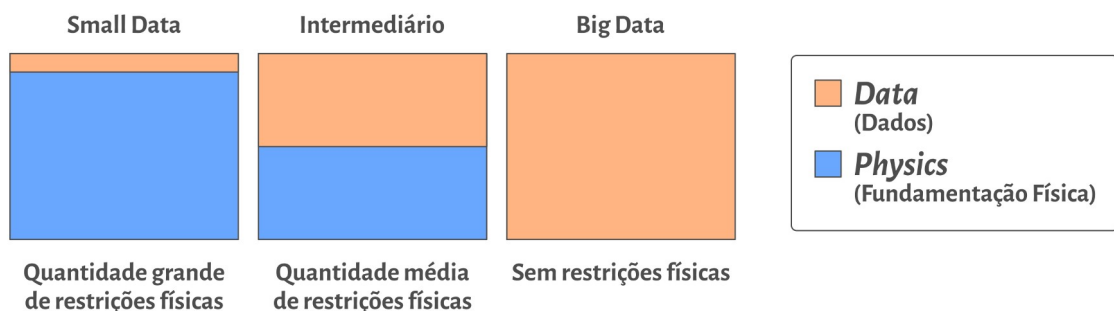
A funcionalidade de aproximadores universais, capazes de representar qualquer função com pontos de entrada e de saída de ANN foi explorada para gerar as variáveis de saída desejadas e suas respectivas derivadas, através da chamada diferenciação automática (AD, do inglês *automatic differentiation*). A AD possui um custo computacional inferior à derivação simbólica e precisão superior à derivação numérica (NGO; LIM, 2021).

A aplicação dos PINNs se dá, basicamente, de duas formas. Uma consiste em resolver equações diferenciais já conhecidas empregando um PINN em substituição a um método numérico, e é chamada de método *forward* por alguns autores. A outra consiste em resolver sistemas de equações em conjunto com incógnitas, descobrindo, dessa forma, NNs que representem essas incógnitas. Isso,

por sua vez, permite simultaneamente resolver uma equação diferencial, estimar seus parâmetros (incógnitas) e posteriormente propor o equacionamento dos parâmetros que foram substituídos por NNs próprias. Esse segundo método é chamado de método *inverse* por alguns autores (NGO; LIM, 2021; RAISSI; PERDIKARIS; KARNIADAKIS, 2019).

Para se entender melhor o conceito de PINN e o motivo de sua criação, é necessário primeiramente introduzir os conceitos de *big* e *small data*. A Figura 9 exibe um esquema visual que explicita a diferença entre os sistemas de dados aqui abordados. No regime *big data*, algoritmos de ML são empregados com uma grande quantidade de dados e, normalmente, nenhum tipo de restrição física – é o modelo clássico de ML. Em um regime *small data*, assume-se que todo o contexto físico teórico é plenamente conhecido, bem como condições de contorno, inicial, e todos os coeficientes e termos das equações diferenciais do sistema.

Figura 9 - Diferentes regimes de dados para PINNs



Fonte: Adaptado de (KARNIADAKIS et al., 2021)

Em um regime intermediário, há uma combinação de uma quantidade de dados menor que o de um regime *big data* com as restrições físicas específicas para o sistema que está sendo representado, mas que normalmente não seriam suficientes, por si só (sem os dados), para representar o sistema. Um dos motivos para a incapacidade de representar o sistema seria, por exemplo, o desconhecimento de alguns termos ou coeficientes das equações diferenciais. Nesse caso, o modelo intermediário é capaz de empregar equações parcialmente conhecidas, em conjunto com dados experimentais medidos à parte, e resolver o sistema de equações enquanto simultaneamente infere os parâmetros e termos que

estavam ausentes do equacionamento. O regime intermediário é encontrado em diversas aplicações, como discutido nas seções 3.4.2 e 3.2.3.3, e é a categoria de maior interesse para diversas aplicações. Os PINNs são especialmente úteis, portanto, para os modelos intermediário e *small data*.

Para que o PINN seja capaz de “aprender” e representar adequadamente as restrições físicas de cada sistema de equações, é necessário que atenda a uma série de pré-requisitos, que variam de sistema para sistema. Um exemplo é um sistema que nunca gere ou calcule valores de massa ou volume menor que zero, que são fisicamente impossíveis. Para isso, se faz necessário introduzir *biases* (vieses ou tendências) no modelo computacional, com o intuito de induzi-lo a embutir as leis físicas ou naturais em sua solução. Os principais *biases* são:

- **Observational biases (vieses de observação):** É introduzido através da modificação ou duplicação de dados com o intuito de aumentar a quantidade de pontos com valores fisicamente coerentes e, assim, direcionar o sistema no sentido correto de aprendizado para que a minimização da função erro seja feita de maneira apropriada. É a maneira mais simples para a introdução de *biases* nos modelos de ML. Para modelos com uma grande quantidade de parâmetros, um grande volume de dados se faz necessário para aplicar esse tipo de *bias*, o que pode torná-lo proibitivo devido a altos custos experimentais ou de simulação;
- **Inductive biases (vieses de indução):** Hipóteses que são aplicadas a partir de ajustes específicos na estrutura de arquitetura do modelo de ML, buscando garantir a obediência a princípios físicos ou restrições matemáticas. Essa abordagem normalmente é limitada a modelos relativamente simples ou com simetria geométrica. É frequentemente aplicada em redes neurais convolucionais e em áreas que façam uso de visão computacional;
- **Learning biases (vieses de aprendizado):** São introduzidos através de modificações na função *loss* (erro) a ser minimizada. Ou seja, explicitamente modificam a função erro a ser minimizada para favorecer a convergência da solução do PINN de modo que esta se situe dentro das restrições físicas existentes. Por conta disso, é dito que o emprego dessa estratégia é baseado em *soft constraints*, ou seja, restrições suaves, uma vez que não existe uma modificação direta na rede para a restrição física, mas sim modificações que auxiliam o sistema a produzir resultados fisicamente coerentes. O uso de *soft*

*constraints* significa que o sistema pode produzir apenas representações físicas aproximadas, mas, por outro lado, torna o modelo mais versátil, e o permite ser aplicado em equações integrais, diferenciais e fracionárias.

Os modelos baseados em *learning biases* contrastam com os baseados em *inductive biases*, portanto, por não proporem uma arquitetura que impõe as restrições físicas do sistema, mas sim uma aproximação através de modificações explícitas que orientam o algoritmo de otimização da NN de forma que produza resultados que respeitem às regras físicas e matemáticas que regem o sistema.

PINNs empregam valores de condições de contorno *soft*, isto é: os valores fisicamente incoerentes citados nos parágrafos anteriores não são impossibilitados através de um código que os proíba, mas através do fornecimento de um conjunto de dados e equações que permitam ao sistema aprender os limites físicos de cada variável. A flexibilidade fornecida pelas condições *soft* permitem que o sistema consiga incorporar informações mais genéricas do sistema de equações ao modelo de ML. Em resumo, essa abordagem funde a solução numérica tradicional (baseadas em modelos) à abordagem de ML (baseada em dados).

PINNs adotam uma abordagem diferente das ANN. A geração de dados se dá através do uso de modelos matemáticos. Uma função *loss* (que representa o erro) é minimizada em cada etapa de iteração (chamada de *epoch* em alguns algoritmos). Em algumas referências, a *loss* também é chamada de *cost* ou função custo (GLOROT; BENGIO, 2010; VASILEV et al., 2019). A função consiste na soma dos desvios entre os valores preditos pela rede neural e aqueles que seriam os corretos (obtidos através do equacionamento matemático). Assim, o PINN integra as informações advindas do sistema de equações na NN através da função *loss*. É através de iterações subsequentes que a rede é capaz de aprender os valores do sistema de equações e resolvê-lo. A partir da minimização do erro das derivadas, portanto, o sistema é capaz de obter também as variáveis. A função de perda (*loss*) é definida pela Equação 49 e  $L_{data}$  e  $L_{PDE}$  pelas Equações 50 e 51:

$$L = \sum w_{data} L_{data} + w_{PDE} L_{PDE} \quad (49)$$

Em que  $L$  é a *total loss* ou função erro,  $w_{data}$  é o peso do erro em relação ao conjunto de dados,  $w_{PDE}$  é peso do erro em relação ao sistema de equações. Reparar que o peso (*weight*) aqui se refere ao peso que a variável  $N$  compõe para a determinação da *loss*, e não ao peso empregado entre neurônios, que é um outro conceito.

$$L_{data} = \frac{1}{N_{data}} \sum_{i=1}^{N_{data}} (\hat{y}_i - y_i)^2 \quad (50)$$

Em que  $\hat{y}_i$  e  $y_i$  são respectivamente o valor predito pela Rede e o valor esperado para o conjunto de dados de entrada “i” e  $N_{data}$  é o número de pontos que foram calculados ou testados.

$$L_{PDE} = \frac{1}{N_{PDE}} \sum_{j=i}^{N_{PDE}} (eq_1, eq_2 \dots eq_N)_j \quad (51)$$

Em que  $N_{PDE}$  é o número de pontos em que o sistema foi predito e usado para comparação e aprimoramento da rede através da *loss* (normalmente chamado de *domain*);  $j$  representa cada ponto individual e o conjunto de variáveis de entrada empregados, naquele ponto, na NN;  $eq_1, eq_2 \dots eq_N$  representam as equações do sistema, organizadas de tal forma que o valor correto seja zero. Por exemplo, caso a equação  $eq_1$  representasse  $dX/dt = X$ , o valor após um lado da equação ser igualado a zero seria  $0 = X - dX/dt$  e o valor de  $eq_1$ , portanto, seria  $eq_1 = X - dX/dt$ .

O conjunto de dados fornecidos, empregado na Equação 50, muitas vezes é composto basicamente pelas condições iniciais e de contorno. O conjunto de dados empregados na Equação 51 normalmente é gerado pelo sistema de forma espaçada para cobrir toda a faixa de valores de interesse para cada variável de entrada. Os valores de  $w_{data}$  e  $w_{PDE}$  na Equação 49 são usados para balancear a relevância de um termo em relação ao outro, caso se faça necessário ou sistema esteja exibindo previsões inapropriadas particularmente nas condições de contorno ou em desrespeito às leis físicas que regem o sistema de equações.

Na prática, um PINN tem certa semelhança com um sistema de integração numérico, mas possui duas grandes vantagens. A primeira é não depender fortemente da discretização dos pontos no tempo ou no espaço (sendo por isso denominado *gridless*). Por isso, a rede PINN, após o treino, é capaz de simular o modelo matemática em graus de resolução diferentes daqueles para as quais foi otimizada, sem necessidade de novas etapas de treino (a resolução aqui citada diz respeito às aproximações infinitesimais de variáveis, sobretudo do espaço e do tempo, comumente representadas respectivamente por “dz” e “dt”). A segunda vantagem é que, uma vez que o modelo tenha sido treinado e otimizado (o que pode

levar um tempo considerável), ele pode ser reutilizado infinitas vezes a um custo computacional ínfimo (MARKIDIS, 2021). Isso poderia permitir, por exemplo, a otimização de modelos em computadores de alta capacidade, mas a execução dos modelos já prontos em computadores comuns.

Diversos fenômenos já foram representados através de PINNs, como fluxos aerodinâmicos (MAO; JAGTAP; KARNIADAKIS, 2020) e processos de adsorção (SANTANA et al., 2022). Desde seu lançamento, a metodologia ganhou muita tração, atraindo a atenção de pesquisadores de diversos países. Os países com mais publicações relacionadas a PINNs nos anos de 2019 a 2022 foram Estados Unidos e China, responsáveis por 29% e 25% do total, respectivamente (LAWAL et al., 2022). O mesmo estudo que levantou esses dados estatísticos mapeou uma série de limitações de PINNs e de alternativas híbridas baseadas em PINNs. Frequentemente, a sensibilidade a configurações das Redes Neurais é uma forte limitação. Outro ponto é a aplicabilidade restrita de cada uma dessas variações. Um dos casos estudou a dinâmica de lençóis freáticos, mas demonstrou-se inapropriado quando o modelo abordou uma maior área (espaço) ou possuía maior complexidade (LAWAL et al., 2022; ZHANG et al., 2022).

O *spectral bias* é uma das dificuldades ao se trabalhar com PINNs é, e se manifesta no treino da NN através de *vanishing gradients* que impedem o erro de ser efetivamente reduzido (KARNIADAKIS et al., 2021). O *spectral bias* é um fenômeno observado empiricamente (RAHAMAN et al., 2019) em que funções menos complexas (CAO et al., 2021) ou de menor frequência são aprendidas ou otimizadas primeiro pelo NN. Outra grande dificuldade é que o design de arquiteturas de NN efetivas normalmente é feito de forma empírica, o que pode requerer muito tempo por parte dos pesquisadores ou desenvolvedores (KARNIADAKIS et al., 2021). Não há uma correlação matemática explícita que determine ou norteie o número mínimo de camadas, neurônios por camada ou iterações para resolver com um erro aceitável determinada equação diferencial por PINN (RAISSI; PERDIKARIS; KARNIADAKIS, 2019).

Um ponto que não é exatamente uma desvantagem, mas um aprimoramento ainda não aplicado, é o uso de derivadas de maior ordem (maiores que 1). *Frameworks* comumente empregados, como PyTorch e TensorFlow não suportam nativamente a derivação de equações de ordens mais altas, o que faz com que a



velocidade de treino seja limitada em relação ao potencial caso essa implementação existisse (KARNIADAKIS et al., 2021).

Embora a aplicação de modelos de ML a biorreações e biorreatores, bem como estações de tratamento de efluentes (ANDRADE CRUZ et al., 2022; BAGHERZADEH et al., 2021; MATEO PÉREZ et al., 2021; MEY et al., 2021) tenha muitos estudos publicados, a aplicação de PINNs a bioprocessos ainda não é tão difundida. Assim, novos esforços, como este, são bem-vindos para identificar os principais méritos e dificuldades do método na representação de bioprocessos. Para solucionar alguns pontos específicos de dificuldades no emprego de PINNs a modelos que representam fenômenos biológicos, foram introduzidos os BINNs, explicados na seção 3.2.3.3. Contudo, ele foi empregado para o estudo de difusão de células em um meio (um fenômeno mais relacionado à transferência de massa e cinética de crescimento celular), e não para a modelagem de um sistema de engenharia, como um reator.

### **3.2.1 Solução de equações diferenciais orientada por dados**

A solução de equações diferenciais através de PINNs busca, a partir de parâmetros conhecidos, determinar os possíveis estados (valores das variáveis de interesse, aqui representadas por “u”) que o sistema pode assumir. Essa forma de uso dos PINNs também é chamada de *forward* (NGO; LIM, 2021). Assim, a ideia de resolver equações diferenciais de forma orientada por dados busca aplicar parâmetros já existentes para determinar os estados do sistema. A Equação 52 descreve matematicamente essa operação (RAISSI; PERDIKARIS; KARNIADAKIS, 2019):

$$u + N[u; \lambda] = 0, \quad x \in \Omega, \quad t \in [0, T] \quad (52)$$

Em que u representa a solução ou estado do sistema; N representa um operador não linear de parâmetro  $\lambda$ ;  $\Omega$  é o domínio do espaço.

Muitos dos problemas que representam Equações Diferenciais Parciais ou Equações Diferenciais Ordinárias podem ser representados através de uma abstração (uma equação generalista). Uma dada variável de interesse, u, com

dependência no espaço e no tempo, pode ser descrita pela Equação 53 (KRISHNAPRIYAN et al., 2021):

$$F(u(x, t)) = 0, x \in \Omega \subset \mathbb{R}^d, t \in [0, T] \quad (53)$$

Em que  $F$  é uma abstração que representa o operador diferencial;  $u$  é a variável de interesse;  $x$  representa o espaço;  $t$  representa o tempo;  $T$  é o tempo máximo (horizonte de tempo);  $\Omega$  é o domínio do espaço.

Para um modelo contínuo, a função  $F$  pode então ser definida conforme a Equação 54 e o valor de  $u$  pode, então, ser aproximado por uma rede neural profunda (RAISSI; PERDIKARIS; KARNIADAKIS, 2019):

$$f \stackrel{\text{def}}{=} u + N[u] \quad (54)$$

A rede pode então ser derivada aplicando a regra da cadeia (conforme demonstrado na seção 3.1.1.4). A função  $f$  é definida de tal forma que possui os mesmos parâmetros de entrada que a solução ou estado que está sendo resolvido, isto é: se  $u = u(x, t)$ , então  $f = f(x, t)$ . Para otimizar a rede, a função objetivo a ser minimizada é definida na Equação 55 (RAISSI; PERDIKARIS; KARNIADAKIS, 2019):

$$MSE = MSE_u + MSE_f \quad (55)$$

Em que

$$MSE_u = \frac{1}{N_u} \sum_{i=1}^{N_u} |u(t_u^i, x_u^i) - u^i|^2 \quad (56)$$

E

$$MSE_f = \frac{1}{N_f} \sum_{i=1}^{N_f} |f(t_f^i, x_f^i)|^2 \quad (57)$$

Em que  $t$  representa o tempo;  $x$  é o espaço; a função erro (loss)  $MSE_u$  representa as condições iniciais e de contorno;  $MSE_f$  é responsável por promover a estrutura da equação (e portanto o *embasamento físico*) em um conjunto de pontos.

### 3.2.2 Descoberta de equações diferenciais orientada por dados

A descoberta de equações diferenciais orientada por dados através de PINNs foi abordada no trabalho original (RAISSI; PERDIKARIS; KARNIADAKIS, 2019). A técnica consiste em determinar valores de parâmetros ou termos diferenciais na equação simultaneamente, enquanto a equação é resolvida para responder a um conjunto de restrições e/ou prever adequadamente um dado conjunto de dados. Essa técnica também é chamada de *inverse*, por partir de parâmetros que inicialmente são incógnitas (NGO; LIM, 2021). De forma semelhante à Equação 52, é inicialmente definida uma função, como descrito na Equação 58:

$$f \stackrel{\text{def}}{=} u_t + N[u; \lambda] \quad (58)$$

Em que  $\lambda$  representa os parâmetros do operador diferencial.

O processo consiste em aproximar  $u_i(t, x)$  por uma NN  $f(t, x)$ , o que gera um PINN. Os parâmetros do operador diferencial,  $\lambda$ , são transformados então em parâmetros do PINN, e então usados para aproximar as equações diferenciais em questão. É então feita a diferenciação da Rede Neural aplicando a regra da cadeia. Um exemplo completo usando a equação de *Navier Stokes* é disponibilizada na literatura científica (RAISSI; PERDIKARIS; KARNIADAKIS, 2019).

Uma importante consideração ao empregar essa técnica se dá no reconhecimento de que fenômenos podem ter mais de uma interpretação física. Por conta disso, a descoberta de equações de forma *data-driven* (orientados por dados) feita a partir de um mesmo conjunto de dados (ou de conjuntos que contenham informações equivalentes) pode produzir modelos distintos, mesmo que produzam resultados extremamente semelhantes. Assim, há uma relevância e uma necessidade no desenvolvimento e aprimoramento de técnicas que permitam a integração e validação de modelos criados dessa forma (KARNIADAKIS et al., 2021).

### 3.2.3 Aplicações

### 3.2.3.1 Gerais

PINNs já foram aplicados a uma série de diferentes modelos e sistemas de equação, envolvendo reatores, reações e modelos de Engenharia Química. Nesta seção, discutimos as principais conclusões de alguns desses trabalhos, bem como obstáculos encontrados ao longo dos estudos e sugestões para trabalhos futuros.

Um estudo de 2021 investigou a aplicação de PINNs na simulação de reações de conversão de CO<sub>2</sub> em metano em um reator de leito fixo (NGO; LIM, 2021). Foram testadas configurações que variaram de 64 a 256 neurônios por camadas, 2 a 10 camadas, 4.000 a 10.000 s de tempo de treino e 1.000 a 30.000 pontos de treino. As funções de ativação sigmoid e tanh foram consideradas em um primeiro momento, e a inicialização de Xavier (seção Erro: Origem da referência não encontrada) foi empregada. A função sigmoid exibiu boa performance apenas para um dos casos estudados enquanto a tanh apresentou um bom desempenho para todos, o que fez com que ela fosse usada nas etapas seguintes.

O tempo de treino da NN foi aproximadamente proporcional ao número de *hidden layers* e ao número de pontos de treino. Curiosamente, o tempo de treino não aumentou significativamente com o aumento do número de neurônios. Os autores atribuíram esse comportamento ao fato de que mais neurônios proporcionaram uma convergência mais rápida, o que contrabalanceou o custo computacional dos mesmos.

O uso de apenas duas camadas secretas foi insuficiente para representar o modelo, mesmo com 256 neurônios por camada. A otimização da NN acabou por gerar modelos que predizem valores constantes para todas as concentrações de substâncias estudadas. Normalmente, essa predição de valores constantes acontece em sistemas que atingem um estado estacionário ou se aproximam assintoticamente de algum ponto, quando a NN é incapaz de fazer predições melhores em diversas iterações. Assim, a estratégia para minimizar o erro acaba envolvendo o uso de um valor idêntico independente dos *inputs*. Isso se dá porque, ao se comparar o erro (*loss*) contra um valor de predição constante e limitado, ele será menor do que o erro gerado por um sistema que faz predições muito distantes das corretas, variando inclusive em ordens de grandeza, por mais que o perfil seja graficamente mais coerente. Ou seja, a predição de valores constantes pelo modelo

ao longo de todo o domínio das variáveis de entrada pode indicar que o modelo não possui complexidade o suficiente (número de camadas ou de neurônios por camada) para representar o fenômeno.

Dentre as combinações de hiperparâmetros investigadas, a que apresentou o menor erro possuía 6 camadas secretas com 256 neurônios por camada. Apesar da tendência geral de que, conforme o número de camadas aumentava, aumentava também a capacidade de aprendizado da NNs, um fenômeno interessante aconteceu em modelos com mais de 6 camadas secretas. As configurações com 8 e 10 camadas, respectivamente, apresentaram erros maiores do que as com 6 camadas (embora consideravelmente menores que os modelos de 4 e 2 camadas).

Em outro teste, Ngo e Lim (2021) variaram os dados de treino de forma que apenas uma determinada região do reator fosse considerada. A confiabilidade dos resultados obtidos (preditos) posteriormente dependeu fortemente da região que foi selecionada para treino. De forma geral, como esperado, o emprego de regiões que mostrem os diferentes comportamentos dos perfis de concentração ao longo do reator foi capaz de produzir melhores resultados. Embora seja uma conclusão aparentemente óbvia, é um tanto importante: apesar da coerência física das previsões advir do equacionamento, a escolha da região de treino influencia fortemente nas variações às quais o sistema será submetido. Por conta disso, o uso exclusivo de regiões de saturação, sobretudo as regiões finais do reator, onde a reação já estava estagnada, fez com que o modelo erroneamente assumisse valores constantes ou de baixa variação em todo o comprimento anterior. Assim, é mais seguro que a região de treino se dê dentro de faixas próximas às quais o modelo treinado será usado para predição.

Um estudo empregou PINNs à simulação de um processo de adsorção de proteínas em leito fixo (SANTANA et al., 2022). Através de *random grid search*, um processo que testa vários conjuntos de hiperparâmetros diferentes, a melhor configuração de NN foi determinada como 6 camadas e 80 neurônios por camada. A minimização da *loss* se deu por 25.000 *epochs* do algoritmo Adam (KINGMA; BA, 2014) (seção 3.1.1.5) seguidas de L-BFGS-B até que o valor de erro estipulado fosse alcançado. O resultado obtido foi comparado com a solução numérica do sistema de equações que representa o sistema através do Método das Linhas.

A função *loss* foi tornada *physics-informed* através da adição do *residual* das equações diferenciais normalizadas (multiplicadas) por  $1.10^{-4}$ . Essa estratégia de

multiplicar os termos por um fator havia sido aplicada em um outro estudo e foi empregada por contribuir para acelerar a convergência. O espaço e o tempo foram adimensionalizados. O erro da solução foi maior em valores de tempo maior (ou seja, mais próximos do tempo máximo considerado) em valores do espaço mais próximos do limite superior (que, devido à adimensionalização, é igual a 1).

Em testes feitos em um mesmo computador, o modelo obtido por PINN foi capaz de realizar previsões do sistema em frações de segundos, enquanto que o modelo por método das linhas precisou de cerca de 37 segundos. Assim, foi validado não só a capacidade do emprego de PINNs a sistemas relativamente complexos (em específico a simulação de adsorção de proteína) como a possibilidade de usá-lo para casos que necessitem de simulações em tempo real ou iteração constante.

Ren, Wu e Weng (2023) estudaram a modelagem de sistemas de produção de gás a partir de biomassa (REN; WU; WENG, 2023). Como o sistema é bastante complexo, envolvendo diversas reações, e a composição da biomassa pode variar consideravelmente, modelá-lo de forma tradicional é desafiador – daí se dá a importância de aplicação de ML. Além disso, o sistema é representado por funções monótonas - se  $x > y$ ,  $f(x) > f(y)$  – o que configura mais uma restrição a ser atendida, e que pode ser difícil para sistemas de ML.

Os PINNs foram empregados especificamente para tentar gerar as restrições físicas necessárias ao sistema. Outra razão para o emprego de PINNs se deu pela disponibilidade de dados limitadas – os autores possuíam 324 pontos, obtidos através de estudos feitos por diversos outros autores. 200 desses pontos foram usados para validação do modelo, restando apenas 124 para otimização e treino. As necessidades de restrição física muito provavelmente não seriam adequadamente “ensinadas” à rede com um conjunto tão pequeno de dados, dado que se tem três variáveis de entrada (temperatura, umidade e *equivalence ratio* [a razão entre o oxigênio fornecido e a quantidade necessária para a combustão completa, segundo estequiometria, do material em questão]) e diversas variáveis de saída (a composição dos gases). 85% dos dados foram usados para treino da NN e 15% para teste.

Para o treino do aspecto de monotonicidade, foi criada uma função cujos valores foram chamados de “sintéticos”. Em vez de prever as saídas esperadas (que não existem) era avaliada a coerência. Se um aumento na entrada era esperado na

saída e isso não ocorria, então a função de monotonicidade contribuía para aumentar a *loss*. Foi através desse aumento da *loss* que o sistema foi ensinado a obedecer à monotonicidade. Como a função monotonicidade criada não era continuamente diferenciável, o modelo tradicional de *back-propagation* não pode ser empregado, e os autores optaram por usar o CMA-ES (Estratégia de Evolução da Adaptação da Matriz de Covariância, do inglês *Covariance Matrix Adaptation Evolution Strategy*).

Os modelos com PINNs superaram significativamente outros cinco modelos de ML analisados pelos autores, e foram capazes de generalizar predições mesmo quando se encontraram fora da faixa de dados empregada no treino. Dos demais cinco modelos analisados, o que mais se aproximou da precisão dos PINNs foi a *Random Forest*. A ANN tradicional, embora fosse tecnicamente o modelo mais próximo do PINN dentre os investigados, foi curiosamente a de pior desempenho – o que reforça a importância e relevância dos PINNs.

É importante destacar que a observação (o modelo PINN conseguiu generalizar adequadamente para predizer com precisão aceitável dados fora da faixa de treino) não se contradiz diretamente com o encontrado por Ngo e Lim (2021): no caso de Ngo e Lim, o sistema foi capaz de predizer valores fora da faixa de dados de treino mas, para isso, o intervalo de treino precisou ser significativo e se localizar em uma faixa que representasse o comportamento do sistema de forma mais “global”, e não um comportamento específico daquela faixa.

### 3.2.3.2 *Physics-based architecture PINNs*

Os PBA-PINNs (PINNs com arquitetura com embasamento físico, do inglês *physics-based architecture - PINNs*) foram propostos em um trabalho com o intuito de permitir não só o treinamento dos pesos da rede neural, mas também a construção da própria arquitetura da rede com embasamento físico (TARKHOV; LAZOVSKAYA; MALYKHINA, 2023). Com isso, é possível racionalizar mais a etapa de desenvolvimento da arquitetura de NN, processo incentivado em um outro trabalho (KRISHNAPRIYAN et al., 2021). Como um estudo de caso, foi empregado o

estudo da explosão térmica de um reator não isotérmico. O equacionamento adimensionalizou os valores de concentração e espaço.

Uma vez que parte do objetivo dos PBA-PINNs era gerar uma rede realtivamente compacta, os pesquisadores propositalmente empregaram poucos neurônios e apenas uma camada secreta - embora o uso de muitos neurônios e camadas secretas permitisse à rede mais flexibilidade e possivelmente melhores resultados. Foram empregados de 3 a 20 neurônios por camada, valores muito inferiores ao de outros estudos com PINNs também aqui abordados. O treinamento foi feito ao longo de 2000 epochs.

O processo de criação do PBA-PINN ocorre em três etapas. Inicialmente é produzido um modelo de baixa fidelidade (através de métodos numéricos), que é então convertido em um modelo de média fidelidade (através de uma função *loss* com embasamento físico) e então em um modelo de alta fidelidade (através de dados obtidos por sensores) na terceira etapa.

A primeira etapa do processo consiste em construir uma ANN através de uma modificação de métodos números clássicos (em específico o método de Euler). Inicialmente, a NN construída aproxima a solução implícita da equação diferencial que representa a reação de interesse. O resultado disso é um PBA, que também pode ser entendido como uma DNN com “N” camadas secretas. A inicialização dos pesos foi aproximada não de forma randômica (como a função de inicialização de Xavier), mas através do método numérico empregado. Consequentemente, essa primeira etapa depende fortemente do método numérico empregado e do número de iterações executadas. A primeira etapa no geral produz modelo que não são capazes de realizar predições tão interessante, mas é responsável por reduzir consideravelmente o tempo de treino necessário. Após a segunda e terceira etapas, o modelo se torna bastante robusto, embora permaneça relativamente simples e bastante compacto.

A segunda etapa introduziu uma função *loss* com as restrições físicas. O modelo produzido na primeira etapa é treinado novamente, mas com a nova função *loss*. O produto desse processo é um modelo PBA-PINN (anteriormente apenas PBA) de média fidelidade. A terceira etapa consistiu em treinar o modelo mais uma vez, agora com dados de alta fidelidade obtidos por sensores. É ressaltado que o emprego de um número reduzido de neurônios e camadas torna a rede bastante compacta e performática, o que permite que as várias etapas de otimização sejam



viáveis. Mais uma vez, os pesos da NN são atualizados através de uma função *loss* atualizada.

O modelo gerado pode ser empregado, ainda, para resolver problemas do tipo inverso (i.e. alguns parâmetros da equação foram obtidos através do emprego de PINNs). Uma curiosidade se deu no fato de que o erro diminui conforme o número de neurônios aumentou até o limite de 10 neurônios, mas subiu em 20 neurônios. Esse padrão de comportamento foi observado também em um outro estudo abordado na seção anterior (NGO; LIM, 2021).

Os autores concluíram que os melhores casos para aplicação do PBA-PINN proposto são aqueles em que é difícil representar ao modelo físico do fenômeno e/ou quando o modelo é representado por equações diferenciais com problemas de valor de contorno. Uma ressalva importante foi que um dos parâmetros precisou ser limitado para que não atingisse valores próximos aos que desestabilizariam a função.

### 3.2.3.3 BINN

Baseado fortemente no PINN, foi introduzido o conceito de BINN (Rede Neural com embasamento biológico, do inglês *Biologically-Informed Neural Network*) em 2020 (LAGERGREN et al., 2020). BINNs se valem da capacidade de aproximação universal das ANNs para a solução de equações que representam fenômenos biológicos. Toda a discussão desta seção é baseada no trabalho de Lagergren et al. (2020), exceto quando explicitamente indicado o contrário.

No estudo de caso que validou o BINN, comportamentos biológicos dinâmicos do fenômeno estudado, como taxa de difusão e crescimento celular, foram representados por redes neurais independentes, o que foi feito para que seja possível representá-los mesmo sem o uso de um equacionamento explícito. Assim como no PINN, a diferenciação do próprio sistema de NN é usada para o cálculo de derivadas. Os valores preditos são então empregados para a construção de uma equação parcial diferencial que represente o sistema. A equação gerada com esses termos tem, então, seu valor calculado comparado com o valor predito pelo sistema.

A diferença entre os termos é usada para gerar a função *loss* e posteriormente minimizar o erro do sistema.

A validação do BINN se deu através da modelagem matemática do fenômeno da migração celular *in vitro* em Python, usando a biblioteca PyTorch 1.2.0. Nesse contexto, no geral, não há uma grande quantidade de dados, e muitas vezes eles possuem bastante ruído, o que pode dificultar consideravelmente o trabalho humano de buscar padrões e relações para propor modelos adequados. A estratégia adotada foi fazer a modelagem baseando-se em dados, auxiliando o trabalho dos cientistas que, por sua vez, irão analisar e validar os modelos gerados.

O procedimento feito pelos autores foi similar ao descrito na seção 3.2.2 (Descoberta de equações diferenciais orientada por dados). Primeiramente, foi definido, conforme Equação 59, um termo que descreve a taxa de mudança da quantidade de interesse (densidade celular, do inglês *cell density*):

$$u_t = (Du_x) + Gu, \quad x \in [x_0, x_f], \quad t \in [t_0, t_f] \quad (59)$$

Em que  $u_t$  é a variação da densidade celular, em função da difusão (representada pelo termo  $D$ ) e do crescimento ou reação celular, representados pelo termo  $G$ .

A Equação 59 descreve a forma tradicional de modelagem do processo de migração celular *in vitro*. Como fenômenos físicos possuem mais de uma interpretação possível (KARNIADAKIS et al., 2021), e existem vários fenômenos biofísicos que podem interferir na dinâmica da equação estudada, testar os diversos modelos possíveis seria um grande desafio. Não obstante, muitas das variáveis de sistemas biológicos são não lineares, o que torna a aplicação de técnicas como PINNs mais dispendiosas nesses casos. Para isso, os pesquisadores definiram a Equação 60:

$$u_t = \mathcal{F}(x, t, u_x, u_{xx}, \dots; \theta), \quad x \in [x_0, x_f], \quad t \in [t_0, t_f] \quad (60)$$

Em que  $u = u(x, t)$  é a quantidade de interesse,  $\theta$  é o vetor de parâmetros tal que  $\theta \in \mathbb{R}^k$ , com condições de contorno e iniciais.

O termo  $\mathcal{F}$  na Equação 60 é uma combinação de termos com relevância biológica. Assumindo que a forma da equação que representa o fenômeno será relativamente similar à de fenômenos semelhantes (Equação 59), podemos então encontrar as funções da difusividade e do crescimento celular,  $D$  e  $G$  respectivamente. Então, em vez de substituir os valores de  $D$  e  $G$  na equação, de

forma mecanística, cada um desses termos é representado por uma Rede Neural própria. Assim, posteriormente, é possível estudar as NNs individuais de forma separada para buscar por padrões e propor equações que representem cada termo de  $F$ , e não apenas  $F$  como um todo. A vantagem dessa forma de modelagem é que as formas não lineares de  $D$  e  $G$  podem ser aprendidas mesmo sem ser explicitadas ou usando um conjunto de possíveis termos e testando-os um a um.

A abordagem adotada empregou o uso simultâneo de dados e da equação de descreve o fenômeno (componente responsável pelas informações biológicas e físicas e, portanto, pela característica *physics informed* da rede). Para garantir a coerência com ambas as fontes de informação, a função *loss* foi personalizada – portanto, as restrições são *soft*, conforme descrito na seção 3.2 – e é demonstrada na Equação 61:

$$L_{total} = L_{GLS} + L_{PDE} + L_{Constr} \quad (61)$$

Em que  $L_{total}$  é função *loss* empregada para minimização do erro;  $L_{GLS}$  é a distância de Quadrados Mínimos Generalizada (do inglês *Generalized Least Squares*) entre o valor predito pela NN e o valor observado experimentalmente;  $L_{PDE}$  é o termo responsável por introduzir o embasamento físico (semelhante à Equação 51) e  $L_{constr}$  é o termo responsável por fornecer informações biológicas à NN.

O termo  $L_{constr}$  na Equação 61 foi adicionado especificamente para a tarefa de transformar o PINN em “*Biologically-Informed*” (BINN). As razões pela adição do termo, disponível no trabalho original (LAGERGREN et al., 2020), foram 1) As taxas de difusão e de crescimento,  $D$  e  $G$ , foram mantidas dentro de faixas  $[D_{min}, D_{max}]$  e  $[G_{min}, G_{max}]$ , respectivamente, de forma que representassem taxas biologicamente possíveis e 2) O termo  $D$  não reduz com o aumento da densidade celular,  $u$ , e o termo  $G$  não aumenta com o aumento da densidade celular. Matematicamente,  $\partial D / \partial u \geq 0$  e  $\partial G / \partial u \leq 0$ . Isso foi representado incluindo os valores que não obedecessem a essas limitações dentro da função  $L_{constr}$ .

Um dos destaques do trabalho diz respeito à necessidade de pré-processamento dos dados. Como os dados de densidade de células, posição e tempo diferiam em muitas ordens de grandeza (de  $10^{-3}$  a  $10^3$ , em diferentes unidades). Como tanto as variáveis de entrada ( $x$  e  $t$ ) quanto a comparação entre as variáveis de saída ( $u$ ) e as de entrada tinham tamanha diferença, a NN foi incapaz

de convergir quando alimentada com dados experimentais. A solução encontrada pelos autores consistiu em modificar as unidades de cada variável, de forma que as variações dos valores numéricos ficassem entre aproximadamente 0 e  $10^3$ . Outro ponto da solução incluiu empregar *scaling factors* (fatores de escala), representados por  $\alpha$  para auxiliar essa padronização dos valores de forma a reduzir o esforço necessário para a produção de bons resultados pela NN.

#### 3.2.3.4 Falhas e Desafios

Dentre os desafios ao se trabalhar com PINNs, destaca-se a necessidade de estruturar adequadamente a função *loss*, bem como atentar-se a possíveis dificuldades de otimização derivadas da aplicação de *soft constraints*. Um estudo apontou ainda dificuldades em gerar modelos com previsões de baixo erro para sistemas relativamente simples – o uso de parâmetros de difusividade de valores muito distintos foi o suficiente para obter erros relativos de mais de 100% (KRISHNAPRIYAN et al., 2021). No mesmo trabalho foi demonstrado que a NN empregada possuía capacidade suficiente para realizar as previsões necessárias, mas que a aplicação dos PINNs se provou difícil e necessitou de um ajuste fino dos hiperparâmetros, tornando-a dispendiosa.

Com base nos estudos mencionados e no que já foi discutido anteriormente, é possível ver que aplicações que envolvam PINNs se deparam com frequência com alguns desafios e problemas, sobretudo relacionados a:

- **Grande empirismo:** as decisões com relação ao número de camadas e neurônios por camada, bem como LRs são um tanto arbitrárias (embora seja possível realizar testes variando os parâmetros e determinar os mais adequados);
- **Necessidade de escolha adequada dos dados de alimentação:** o que implica em necessidade de escolha de uma boa região para treino da rede pelo sistema de equações empregado, para garantir que seja fornecida informação representativa do comportamento físico do sistema;

- **Estagnação de aprendizado**, bem como produção de valores lineares ou constantes por não conseguir lidar com instabilidades no treinamento. Mais visível em casos que a rede não possui complexidade o suficiente para representar o problema desejado;
- **Exigência de grande poderio computacional para a etapa de treino** ou tempos de otimização elevados. Questões como a relevância da adimensionalização de variáveis de entrada e de saída e seu impacto na capacidade de produção de modelos de baixo erro e com menor custo computacional ainda permanecem em aberto.

### 3.3 DeepXDE

Dentre as alternativas de bibliotecas computacionais voltadas a PINNs, destacam-se a SimNet, PyDEns, GpyTorch e NeuroDiffEq. Praticamente todas as bibliotecas de maior notoriedade são escritas em Python, com algumas poucas em Julia (KARNIADAKIS et al., 2021). Isso se deve em parte ao amplo ecossistema de Machine Learning em Python, sobretudo pela disponibilidade de frameworks de alto nível e com grande disponibilidade de conteúdo educativo e ampla documentação, como Tensorflow e Pytorch.

A DeepXDE é uma biblioteca em linguagem de programação Python e que emprega outras bibliotecas de Machine Learning e Inteligência Artificial (como Tensorflow 1 e 2 e Pytorch) para a solução de sistemas de equações diferenciais através de PINNs. A DeepXDE é usada ao longo de todo este trabalho para a construção e avaliação dos modelos PINN descritos. A construção da DeepXDE é detalhada no artigo original (LU et al., 2021). A ferramenta atribui *soft constraints* ou condições *soft* à rede, assim como os PINNs descritos na seção 3.2 (KARNIADAKIS et al., 2021). A escolha da DeepXDE se baseou na disponibilidade de documentação e exemplos, testes preliminares e relevância na literatura técnica, onde é citada com certa frequência em trabalhos que revisam a aplicação e desenvolvimento de PINNs.

### **3.4 Engenharia de Processos**

A Engenharia de Processos surge da necessidade da Engenharia Química de melhor integrar, otimizar e modelar processos. É uma área interdisciplinar por natureza, abordando conceitos de Ciências básicas (como Biologia e Química), Fundamentos (compreensão de fenômenos e modelagem matemática) e Engenharia de Equipamentos (projeto e modelagem de equipamentos como reatores, trocadores de calor e colunas de adsorção) (PERLINGEIRO, 2018). É a Engenharia de Processos que permite o projeto de processos integrados, conectando blocos de processos e equipamentos que outrora seriam modelados e avaliados separadamente. Da mesma forma que representa um grande potencial por possibilitar a geração sistemática de sistemas baseados em variáveis de interesse, a área também apresenta muitos desafios, pois se faz necessário conhecimento em áreas distintas, integração dos mesmos, boa compreensão de conceitos e fundamentos básicos e boa organização das ideias e dos algoritmos empregados.

#### **3.4.1 Processos estacionários e transientes**

A depender da natureza, os processos podem ser classificados em estacionários ou transientes. São considerados estacionários aqueles processos ou equipamentos cujas propriedades (como concentração, pressão, temperatura ou volume) não variam ao longo do tempo. Já o estado transiente representa aqueles processos onde há variação de qualquer propriedade ao longo do tempo (DORAN, 2013).

#### **3.4.2 Dimensionamento e Simulação de Equipamentos**

O dimensionamento de um equipamento consiste na determinação das propriedades e variáveis que, em conjunto, permitirão a construção e/ou simulação dos equipamentos. Para tanto, se fazem necessários 2 subconjuntos de variáveis: as condições conhecidas (como temperatura na superfície externa do reator, se o equipamento é isolado ou não do meio em que está inserido) e as metas de projeto, que determina quais variáveis devem ser priorizadas durante a proposta da solução (PERLINGEIRO, 2018). No projeto de um reator contínuo, a minimização do volume a fim de reduzir gastos é uma meta de projeto, e a produção fixada de produto em kg/h é uma condição conhecida. A simulação consiste em empregar as variáveis vindas das correntes de entrada (sejam elas de matéria ou energia) e as dimensões do equipamento para simular seu comportamento.

As operações de dimensionamento e simulação, aplicadas à otimização, podem ser empregadas para melhorar o design, segurança e lucratividade de processos químicos, mas requerem uma boa bagagem teórica dos fundamentos e fenômenos envolvidos no processo de interesse. A simulação pode, ainda, ser empregada como uma ferramenta auxiliar para diagnosticar problemas em uma indústria já existente e propor soluções (JANA, 2011).

### **3.5 Ácido Lático**

O Ácido Lático (LA) é um nome alternativo para o ácido 2-hidroxiopropanóico (DE OLIVEIRA et al., 2021), uma molécula de grande importância econômica. Empregado nas indústrias alimentícia, farmacêutica, cosmética e de síntese, tem aplicações ainda em impressão 3D – mais especificamente na fabricação do polímero PLA (Poli-Ácido Lático), polímero estudado há anos (DATTA et al., 1995; LEE et al., 1998) e que pode substituir o PET (Poli etileno) em algumas aplicações. Foi descoberto em 1780 pelo químico Scheele. Em 1857, Pasteur determinou que não era uma substância presente naturalmente no leite, mas fruto do metabolismo de microrganismos.

O LA possuir dois isômeros: L(+) e D(-) Ácido Lático, e até os dias atuais a rota de produção fermentativa é amplamente estudada (KOMESU; MACIEL; FILHO, 2017). A síntese química gera uma mistura racêmica, e a fermentativa pode

favorecer expressivamente um dos isômeros a depender das condições e do microrganismo empregado. O emprego de açúcares refinados como fonte de carbono é relativamente caro, então outras alternativas estão sendo estudadas, bem como fontes de nitrogênio menos financeiramente dispendiosas (ALTAF; NAVEENA; REDDY, 2007).

Estima-se que a produção mundial de LA alcançará 1960 mil toneladas em 2025 (LÓPEZ-GÓMEZ et al., 2019), com a indústria alcançando um valor de 8,7 bilhões de dólares americanos (DIN et al., 2021) com cerca de 90% sendo obtido por via fermentativa. A produção de LA costumeiramente ocorre em modo batelada, embora existam modelos contínuos e batelada-alimentada em operação. Alguns processos empregam ainda um reciclo de células para maximizar a conversão do substrato. O processo em batelada costuma ter maior percentual de conversão do substrato (ou seja, o substrato é melhor aproveitado) mas apresenta menor produtividade. Assim, como muitos substratos apresentam valor econômico considerável, a escolha do processo em batelada em detrimento do contínuo, muitas vezes, se dá pela otimização dos custos através do maior aproveitamento possível da matéria-prima (KOMESU; MACIEL; FILHO, 2017). Fica claro, portanto, que o custo da matéria-prima é um ponto importante na determinação do preço de venda e da margem de lucro no mercado de LA, e que são necessários esforços para a descoberta de substratos de custos reduzidos, com menor impacto ambiental e com melhor apelo de marketing.

Muitos dos esforços para redução de custos na indústria do LA envolvem a busca por novos microrganismos, engenharia genética e novas fontes de matérias-primas para serem usadas como substratos (como rejeitos ou subprodutos industriais). O Quadro 1 exhibe algumas matérias-primas empregadas na produção de LA em diversos estudos. Na Linha Método, o termo “B” indica o emprego de organismos microbiológicos, e o termo “Q” o emprego de síntese de natureza química.

Quadro 1 - Substratos empregados na produção de LA

<b>Substrato</b>	<b>Método de Produção</b>	<b>Microrganismo ou tecnologia química empregado</b>	<b>Referência</b>
------------------	---------------------------	--	-------------------



Cassava Flour (Farinha de Mandioca)	B	<i>Lactobacillus brevis</i>	(QUINTERO et al., 2012)
Glycerol (glicerina)	Q	Conversão hidrotérmica	(ARCANJO; FERNANDES; SILVA, 2015)
Potato Starch (Amido de Batata)	B	<i>Thermotoga neapolitana</i>	(PRADHAN et al., 2021)
Whey (Soro de leite)	B	<i>Lactobacillus casei</i>	(ALTIOK; TOKATLI; HARSA, 2006)
Xarope de açúcares (subproduto da produção de cenouras)	B	<i>Rhizopus oryzae</i> e <i>Rhizopus arrhizus</i>	(SALVAÑAL et al., 2021)
Sugar cane juice (Suco da cana-de-açúcar)	B	<i>Lactobacillus delbrueckii</i>	(DEY; PAL, 2013)
Glicose	B	<i>Rhizopus oryzae</i>	(HAMAMCI; RYU, 1994)
Amido	B	<i>Lactobacillus amylophilus</i>	(ALTAF; NAVEENA; REDDY, 2007)

Fonte: Autoria Própria (2022)

Os substratos tradicionais, como açúcares refinados, competem diretamente com a indústria alimentícia e são denominados como 1G (primeira geração). Os substratos advindos de rejeitos ou subprodutos de outras indústrias, como o soro de leite (ALTIOK; TOKATLI; HARSA, 2006) ou um xarope derivado de cenouras descartadas (SALVAÑAL et al., 2021) oferecem custos reduzidos e maior apelo ambiental, e são determinados 2G (segunda geração). Embora em um primeiro momento a produção a partir de matéria-prima 2G pareça necessariamente mais vantajosa, muitas vezes não é o que é observado. Como essas matérias-primas são mais complexas e quase sempre não são refinadas, é necessário lidar com as variações de safra ocasionadas por diversos fatores, muitos fora do controle do homem – como o clima no local da produção. Além disso, incluem contaminantes como vanilina e furfural, que precisam ser separadas posteriormente e podem elevar o custo do processo de purificação (DIN et al., 2021).

A maioria dos microrganismos empregados na produção de Ácido Lático são bactérias, embora também existam processos que empreguem fungos. As LAB (Bactérias Produtoras de Ácido Lático, do inglês *Lactic Acid Bacteria*) são em sua grande maioria *cocci*, e os pontos ótimos de operação de reatores que as empregam variam de 25 a 45°C (temperatura) e 5 a 7 (pH) (KOMESU; MACIEL; FILHO, 2017).

Em geral, as LAB também são microrganismos anaeróbios facultativos e toleram pHs relativamente ácidos. Contudo, em pHs muito ácidos, a produção de LA, que é o grande objetivo do processo, é fortemente prejudicada, então algumas estratégias devem ser adotadas para manter o pH numa faixa apropriada. A adição de bases ou álcalis e/ou o emprego de soluções tampão no meio são descritas na literatura como procedimentos apropriados pela literatura. Esses procedimentos são essenciais porque, tendo em vista que o LA é um ácido orgânico, sua conversão de substrato em produto naturalmente reduz o pH do meio, o que por sua vez pode reduzir a produtividade.

### 3.5.1 Cinética de produção

A cinética de produção de LA a partir de LABs normalmente inclui termos de inibição por substrato, biomassa ou produto, indicando uma importante limitação do processo e uma justificativa da importância técnica da simulação e otimização dos processos de produção do mesmo (ALTIOK; TOKATLI; HARSA, 2006; NANCIB et al., 2015; THAKUR; PANESAR; SAINI, 2019). Ao longo de toda esta seção, os termos X, P e S representam respectivamente a concentração de biomassa, a concentração de produto e a concentração de substrato em cada um dos estudos mencionados. Os valores acrescidos do subscrito “0”,  $X_0$ ,  $P_0$  e  $S_0$ , por sua vez, representam as concentrações no momento inicial ( $t = 0$ ).

O processo de produção de L(+)-LA a partir de lactose do soro do leite por *Lactobacillus casei* foi estudado por Altiook e colaboradores (ALTIOK; TOKATLI; HARSA, 2006) e é descrito nas Equações 62 a 64. O trabalho consistiu em determinar os parâmetros das funções de cinética para um reator com 3 L de volume útil, operando a 37 °C e pH 5.5 em regime batelada, em diferentes valores de X, P e S no momento  $t = 0$ . O valor de maior produtividade,  $2.5 \text{ g.L}^{-1}.\text{h}^{-1}$ , foi alcançado com  $X_0 = 35.5 \text{ g.L}^{-1}$ .

$$\frac{dX}{dt} = \mu_{max} \left(1 - \frac{X}{X_m}\right)^f \left(1 - \frac{P}{P_m}\right)^h \quad (62)$$

Em que  $\mu_{\max}$  é a velocidade máxima de crescimento,  $S$  é a concentração de substrato,  $K_S$  é a constante de Monod,  $X_m$  é a concentração máxima de biomassa,  $P_m$  é a concentração máxima de produto e  $f$  e  $h$  são parâmetros que permitem os ajustes dos fatores de inibição por biomassa e por produto, respectivamente.

$$\frac{dP}{dt} = \alpha \frac{dX}{dt} + \beta X \quad (63)$$

Em que  $\alpha$  é o coeficiente de geração de produto associada ao crescimento celular (relaciona-se à variação da concentração de biomassa), e  $\beta$  é o coeficiente de geração de produto não associado ao crescimento celular (relaciona-se apenas à concentração de biomassa).

$$\frac{dS}{dt} = \frac{-1}{Y_{PS}} \frac{dP}{dt} - m_s X \quad (64)$$

Em que  $Y_{PS}$  é o coeficiente de rendimento do produto e  $m_s$  é o coeficiente de manutenção celular.

A Equação 62 representa a cinética de crescimento celular através do clássico modelo de Monod, modificado para representar as limitações de crescimento celular através do termo  $(1 - X/X_m)^f$  e a inibição por produto através do termo  $(1 - P/P_m)^h$ . A Equação 63 descreve a produção de LA associando a variação da concentração de biomassa ( $dX/dt$ ) e o próprio valor de  $X$  através das constantes  $\alpha$  e  $\beta$ . Equação 64 descreve a formação de produto através do modelo de *Luedeking-Pirret*. Enquanto os termos  $(1 - X/X_m)$  e  $(1 - P/P_m)$  puramente representam as respectivas inibições por biomassa e por produto, os expoentes  $f$  e  $h$  são empregados como parâmetros para permitir a otimização do sistema a cada um dos casos estudados. O termo de manutenção celular,  $m_s$ , foi considerado irrelevante para a representação adequada do modelo.

Um outro trabalho estudou a produção de LA por *L. casei* MTCC 1423 tendo melaço de cana-de-açúcar como substrato (THAKUR; PANESAR; SAINI, 2019). A variação de biomassa foi satisfatoriamente representada pela Equação 65, que considera o fator de inibição populacional ou de biomassa como proporcional a  $X^2$

(equação de Riccati). A variação da concentração de produto é dada pela Equação 63 e a de substrato pela Equação 67.

$$\frac{dX}{dt} = \mu X (1 - \delta X) \quad (65)$$

Em que  $\mu$  é a taxa de crescimento celular (Equação 66) e  $\delta$  é o inverso da concentração máxima de X, ou  $1/X_M$ .

$$\frac{1}{\mu} = \frac{1}{\mu_{max}} + \frac{K_S}{\mu_{max}} \left( \frac{1}{S} + \frac{S}{S_M^2} \right) \quad (66)$$

Em que  $\mu$  é a taxa específica de crescimento,  $K_S$  é a constante de inibição por substrato e  $S_M$  é a concentração máxima de substrato.

$$\frac{dS}{dt} = \frac{-1}{Y_{XS}} \frac{dX}{dt} - \frac{1}{Y_{PS}} \frac{dP}{dt} - m_s X \quad (67)$$

Em que  $Y_{XS}$  é o rendimento de biomassa com base no substrato, e é representado pela Equação 68.

$$Y_{XS} = \frac{\Delta X}{\Delta S} \quad (68)$$

Em que  $\Delta X$  e  $\Delta S$  são respectivamente a variação de ácido láctico e de substrato no meio durante a fase de crescimento celular.

O pH foi variado em vários testes, e o valor ótimo encontrado, 6,75, foi consideravelmente menos ácido do que outros estudos. Isso se deu, em parte, pelas altas concentrações de LA em alguns casos (até 120 g/L), o que contribui para a inibição por produto em pHs mais ácidos. O modelo de consumo de substrato (Equação 67) também foi considerado em estudo anterior (ALTIOK; TOKATLI; HARSA, 2006), mas nele foi incapaz de produzir bons resultados e, por isso, descartada.

Um estudo mais recente investigou a produção de ácido láctico a partir da fermentação de dois carboidratos, lactose e glucose, simultaneamente, por *Lactiplantibacillus plantarum* (VERA-PEÑA; HERNÁNDEZ-GARCÍA; VALENCIA-GARCÍA, 2022). O reator empregado possuía 4 L de volume útil (5 L volume total), agitado a 100 rpm, e operou em batelada, à temperatura de 32 °C e o pH 5,5. O modelo foi representado adequadamente pelas Equações 63 e 69 a 72. Os

subscritos “glu” e “lac” indicam que o substrato referido pela letra S, em cada uma das variáveis ou parâmetros, corresponde à glucose e à lactose, respectivamente.

$$\frac{dX}{dt} = (\mu - k_d) X \quad (69)$$

Em que  $k_d$  é a taxa específica de morte celular e  $\mu$  é dado pela equação 70

$$\mu = \mu_{max} \left( \frac{k_{Sglu} S_{glu}}{k_{Sglu} + S_{glu}} + \frac{k_{Slac} S_{lac}}{k_{Slac} + S_{lac}} \right) \left( \frac{1}{k_{Sglu} + k_{Slac}} \right) \quad (70)$$

Em que  $k_{Sglu}$  e  $k_{Slac}$  são as constantes de Monod para a glucose e a lactose, respectivamente.

$$\frac{dS_{glu}}{dt} = \left( \frac{\mu}{Y_{XSglu}} + \frac{\alpha\mu + \beta}{Y_{PSglu}} + m_{Sglu} \right) X \quad (71)$$

$$\frac{dS_{lac}}{dt} = \left( \frac{\mu}{Y_{XSlac}} + \frac{\alpha\mu + \beta}{Y_{PSlac}} + m_{Slac} \right) X \quad (72)$$

A cinética de crescimento celular é dada pelas Equações 69 e 70. O conceito de taxa de morte celular, ausente nos demais modelos apresentados nesta seção, foi introduzido através do termo  $k_d$ . A taxa de crescimento celular foi adaptada da cinética dupla de Monod (*double Monod kinetics*) para representar as duas fontes de carbono (glucose e lactose). A cinética de produção de LA também pode ser representada pela Equação 63. Como o modelo permite concentrações de substrato abaixo de zero, um condicional foi criado de tal forma que se  $S < 0$ ,  $S = 0$ , para contornar essa representação fisicamente inapropriada. Um modelo do código é fornecido como a Equação 73.

$$\begin{aligned} & \text{if}(S_{glu} < 0): S_{glu} \leftarrow 0 \\ & \text{if}(S_{lac} < 0): S_{lac} \leftarrow 0 \end{aligned} \quad (73)$$

Em que *if* indica uma função condicional, onde o código após “:” é executado caso o parâmetro avaliado dentro dos parâmetros de *if* seja verdadeiro.

### 3.6 Modelagem matemática

#### 3.6.1 Tanque

A modelagem de um tanque é um simples balanço de matéria, que emprega que a variação interna da propriedade  $m$  é igual à entrada menos a saída de tal propriedade, conforme indicado na Equação 74:

$$[acúmulo] = [entrada] - [saída] + [geração] \quad (74)$$

Caso o termo de geração/consumo seja considerado nulo (implicando que no tanque não ocorrem reações químicas, interações ou variações de temperatura, pressão e volume significativas) a equação pode ser expressa como:

$$\frac{dm}{dt} = \frac{dm_{in}}{dt} - \frac{dm_{out}}{dt} \quad (75)$$

Em que  $m$  é a propriedade,  $m_{in}$  é a propriedade na corrente de entrada do tanque e  $m_{out}$  é a propriedade na corrente de saída do tanque. A equação pode ser expandida, com  $m$  se aplicando a cada propriedade de cada componente das correntes de entrada e saída.

#### 3.6.2 Reatores CSTR e Batelada

A modelagem de um volume de controle qualquer, que servirá de base para a representação de diferentes reatores, é relativamente parecida com a apresentada no tópico 3.6.1, sendo a principal diferença a existência de um termo de geração ou consumo de matéria, o que acaba por implicar também na possibilidade de termos

para o consumo ou geração de energia em virtude das reações químicas que ocorrem. O balanço de matéria é dado pela Equação 76:

$$[\text{corrente entrada}] - [\text{corrente saída}] + [\text{variação por reações}] = [\text{acúmulo}] \quad (76)$$

O que por sua vez pode ser transformado na Equação 77, que representa o balanço de matéria de uma dada substância em um reator:

$$F_{j0} - F_j + \int_V r_j dV = \frac{dN_j}{dt} \quad (77)$$

Em que  $F$  representa a vazão molar da substância “j”,  $N$  representa o número de mols,  $V$  o volume,  $r_j$  a taxa de reação de j,  $t$  o tempo. O subscrito “0” em  $F_{j0}$  indica se tratar da corrente localizada na entrada do reator, ao passo que  $F_j$  representa a corrente de saída. A equação 77 é generalista, e pode ser empregada tanto para a modelagem de reatores batelada quanto contínuos.

Um CSTR (Reator perfeitamente agitado, do inglês *Continuous stirred-tank reactor*) é um tipo de reator muito empregado no meio industrial. Dentre suas vantagens, destacam-se a simplicidade de projeto e modelagem, uma vez que as propriedades são consideradas como uniformes em todo o interior do reator (FOGLER, 2018). Enquanto reatores CSTR podem operar em estacionário ou transiente, reatores em batelada são sistemas essencialmente transientes. Mesmo que não seja observada variação de massa durante determinado período de operação, outras propriedades como a concentração e número de moles ou massa de cada substância ou microrganismo presente pode mostrar dependência temporal, além de variações de energia, temperatura, volume e pressão (DORAN, 2013).

### 3.6.3 Variáveis de Processo

Comumente, um processo químico pode ser descrito empregando três tipos variáveis: *input* (entrada), *output* (saída) e variáveis de estado (JANA, 2011). As variáveis de entrada e de saída são, frequentemente, os pontos de “comunicação” entre um equipamento ou processo e os demais. Já uma variável de estado é aquela que é capaz de descrever o *estado* do sistema em determinado ponto do espaço e

do tempo, e aparecem com frequência nos termos de acúmulo de equações de balanço. Por exemplo, a composição química é uma variável de estado capaz de descrever o balanço de massa do sistema em um determinado ponto.

#### **3.6.4 Modelo**

O modelo de um processo é uma abstração matemática capaz de descrevê-lo. O conjunto de equações usados não representa o modelo físico no qual ele é baseado de forma completa, mas apenas o suficientemente adequada para manter a fidelidade da representação (JANA, 2011). O fenômeno é mais comum em reações químicas com resistência à transferência de massa: por vezes o processo de determinação dos coeficientes e do equacionamento do fenômeno difusivo e convectivo é tão complexo que torna impeditiva uma modelagem tão detalhista. Contudo, uma modelagem que ignora esses efeitos e os inclui dentro das taxas de reação costuma ser uma aproximação boa o suficiente para representar o sistema de forma apropriada. Em alguns casos – que na prática são encontrados com frequência - é possível desprezar efeitos difusivos com base nas condições do sistema, velocidade de reação, sistema de agitação, dentre outras possíveis considerações.