



UNIVERSIDADE FEDERAL DA BAHIA
ESCOLA POLITÉCNICA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

BASSEM YOUSSEF MAKHOUL JUNIOR

Dissertação de Mestrado

**SIMULAÇÃO DO ENRUGAMENTO CINÉTICO NO
CRESCIMENTO DE INTERFACES EM MATERIAIS
MULTICAMADAS UTILIZANDO APRENDIZADO DE
MÁQUINA**

Salvador

8 de fevereiro de 2023

Universidade Federal da Bahia
Programa de Pós-Graduação em Engenharia Elétrica

Bassem Youssef Makhoul Junior

SIMULAÇÃO DO ENRUGAMENTO CINÉTICO NO CRESCIMENTO
DE INTERFACES EM MATERIAIS MULTICAMADAS UTILIZANDO
APRENDIZADO DE MÁQUINA

Dissertação de Mestrado apresentada ao
Programa de Pós-Graduação em Engenharia
Elétrica, PPGEE, da Universidade Federal da
Bahia, como parte dos requisitos necessários
para obtenção do grau de Mestre em Engenharia
Elétrica.

Orientadores:

Prof. Dr. Eduardo Furtado de Simas Filho.

Prof. Dr. Thiago Albuquerque de Assis.

Salvador
8 de Fevereiro de 2023

M235 Makhoul Júnior, Bassem Youssef
Simulação do enrugamento cinético no crescimento de
interfaces em materiais multicamadas utilizando aprendizado de
máquinas /Bassem Youssef Makhoul Júnior. -- Salvador, 2023.
83 f.: il. color.

Orientadora: Prof. Dr. Eduardo Furtado de Simas Filho.
Orientador: Prof. Dr. Thiago Albuquerque de Assis.

Dissertação (Mestrado) - Programa de Pós-Graduação em
Engenharia Elétrica - Universidade Federal da Bahia,
Universidade Federal da Bahia - Escola Politécnica, 2023.

1. Disposição Balística. 2. Processos de Deposição. 3.
Redes Neurais. I. Simas Filho, Eduardo F. de. II. Assis, Thiago
Albuquerque. III. Universidade Federal da Bahia. IV. Título.

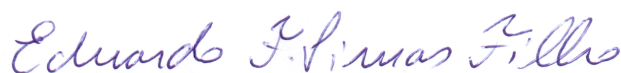
CDD: 621.3

SIMULAÇÃO DO ENRUGAMENTO CINÉTICO NO CRESCIMENTO DE
INTERFACES EM MATERIAIS MULTICAMADAS UTILIZANDO APRENDIZADO
DE MÁQUINA

Bassem Youssef Makhoul Junior

DISSERTAÇÃO SUBMETIDA AO PROGRAMA DE PÓS-
GRADUAÇÃO EM ENGENHARIA ELÉTRICA (PPGEE) DA
UNIVERSIDADE FEDERAL DA BAHIA COMO PARTE DOS
REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE
MESTRE EM CIÊNCIAS EM ENGENHARIA ELÉTRICA

Aprovada por:



Prof. Dr. Eduardo Furtado de Simas Filho
Orientador/UFBA



Prof. Dr. Thiago Albuquerque de Assis
Coorientador/UFBA



Prof. Dr. Antônio Carlos Lopes Fernandes Jr.
Avaliador/UFBA



Prof. Dr. Fernando Albuquerque de Oliveira
Avaliador/UnB/UFBA



Prof. Dr. Eduardo Pestana de Aguiar
Avaliador/UFJF

SALVADOR, BA - BRASIL
8 DE FEVEREIRO DE 2023

Agradecimentos

Agradecer primeiramente a minha mãe pelo constante apoio durante esta jornada, sendo que sem ela não conseguiria chegar até este ponto.

Ao meu pai e aos meus familiares por todo o apoio.

Aos meus amigos, em especial à Igor Caetano, Luiz Carlos, Rafael Almeida, João Vitor, Kaio Freitas e William Max.

Aos meu Orientadores, Prof. Dr. Eduardo Furtado de Simas Filho e Prof. Dr. Thiago Albuquerque de Assis por todo o aprendizado nestes dois anos de mestrado, pelo suporte durante toda a pesquisa e pela oportunidade de poder realizar este trabalho. À todos os professores e funcionários do PPGEE/UFBA.

Aos grupos de pesquisa em Física Estatística e Sistemas Complexos (FESC) e de Superfícies e Materiais (GSUMA) que me permitiram acessar o Cluster PERAU para realizar as simulações deste trabalho.

E, por fim, agradecer a CNPq pelo apoio financeiro.

Resumo

Por conta do aumento da quantidade de dados disponíveis em modo global e da constante evolução dos modelos e algoritmos de treinamento, técnicas de aprendizado de máquina vêm se tornando cada vez mais populares, isto, seja na indústria como na academia. Na área de ciência dos materiais, modelos que envolvem simulação computacional do crescimento de superfícies em situações fora do equilíbrio termodinâmico ajudam a entender e melhorar a produção de novos materiais. A partir destas premissas, este trabalho tem como principal objetivo empregar modelos de aprendizado de máquina para realizar uma previsão da evolução temporal da rugosidade global, considerando o modelo de deposição balística. O objetivo é realizar uma previsão de sistemas com tamanhos laterais que inviabilizam a obtenção do regime estacionário para a rugosidade global via simulação computacional. Utilizamos como parâmetro de validação a raiz do erro quadrático médio, de forma a avaliar o quão bem sucedido o aprendizado foi realizado pela máquina. Podemos destacar como resultados o acesso a evolução temporal da rugosidade global, bem como ao seu estado estacionário, para tamanhos laterais que ainda não foram observados na literatura, e a partir destes, exploramos situações para o crescimento interfacial considerando substratos unidimensionais e bidimensionais, com resultados consistentes para os expoente de rugosidade previstos no limite termodinâmico.

Abstract

Due to the increase in the amount of data available globally and the constant evolution of training models and algorithms, machine learning techniques are becoming increasingly popular, whether in industry or academia. In the area of materials science, models involving computational simulation of surface growth in situations outside of thermodynamic equilibrium help to understand and improve the production of new materials. Based on these assumptions, this work has as main objective to use machine learning models to predict the temporal evolution of global roughness, considering the ballistic deposition model. The objective is to perform a prediction of systems with lateral sizes that make it impossible to obtain the steady state for the global roughness via computational simulation. We used the root mean squared error as a validation parameter, in order to assess how successful the learning was performed by the machine. We can highlight as results the access to the temporal evolution of global roughness, as well as its steady state, for lateral sizes that have not yet been observed in the literature, and from these, we explore situations for interfacial growth considering one-dimensional and two-dimensional substrates, with results consistent for the predicted roughness exponents in the thermodynamic limit.

Lista de ilustrações

Figura 1 – Regra de deposição de partículas para o modelo de deposição aleatório. (Imagem retirada de [26])	6
Figura 2 – Ilustração de um substrato de tamanho $L = 100$ após cinco unidades de tempo (cada cor representa uma unidade de tempo).	6
Figura 3 – Gráfico de rugosidade para o modelo não correlacionado.	7
Figura 4 – Crescimento da rugosidade no modelo correlacionado.	8
Figura 5 – Procedimento para normalizar as curvas de rugosidade.(Imagem retirada de [26])	9
Figura 6 – Regra de deposição para o modelo RDSR.(Imagem retirada de [26]) . .	10
Figura 7 – Gráfico de rugosidade para deposição aleatória com relaxação de superfície.	10
Figura 8 – Ilustração do substrato após 8000 deposições para o modelo de deposição aleatória com relaxação de superfície (cada cor representa uma unidade de tempo).	11
Figura 9 – Regra de Deposição para o modelo de deposição balística.(Imagem retirada de [26])	11
Figura 10 – Crescimento do substrato na deposição balística.	12
Figura 11 – Gráfico de rugosidade para a deposição balística unidimensional para "pequenos" comprimentos.	12
Figura 12 – Gráfico de rugosidade normalizado para a deposição balística unidimensional para "pequenos" comprimentos.	13
Figura 13 – Gráfico de rugosidade para a deposição balística unidimensional para "grandes" comprimentos.	13
Figura 14 – Gráfico de rugosidade normalizado para a deposição balística unidimensional para "grandes" comprimentos.	14
Figura 15 – Gráfico de rugosidade para deposição balística bidimensional.	14
Figura 16 – Ilustração da evolução da curva a cada N iterações.(Imagem retirada de [33])	18
Figura 17 – Pontos gerado a partir de uma função linear com ruído.	18
Figura 18 – Ilustração da evolução do modelo a cada N iterações.	19
Figura 19 – Curva gerada usando o método dos mínimos quadrados.	19
Figura 20 – Curvas geradas usando a máquina de vetores de suporte.(Imagem retirada de [31])	21
Figura 21 – Exemplo de árvore de decisão para o conjunto de dados Iris.	22
Figura 22 – Árvore de decisões para os pontos gerados anteriormente.	23
Figura 23 – Ilustração da curva gerada pela árvore de decisão.	23
Figura 24 – Ilustração simplificada do neurônio biológico.(Imagem Retirada de [33])	24

Figura 25 – Ilustração Matemática do neurônio artificial.(Imagem Retirada de [33])	25
Figura 26 – Ilustração do Perceptron Multicamadas.(Imagem Retirada de [33]) . . .	27
Figura 27 – Ilustração de um exemplo de arquitetura do Perceptron Multicamadas.	27
Figura 28 – Exemplo de rede recorrente formada com simples células recorrentes. .	30
Figura 29 – Ilustração de diversas arquiteturas de redes recorrentes (imagem retirada de [38]).	30
Figura 30 – Ilustração de simples célula recorrente.	30
Figura 31 – Ilustração de célula LSTM.	32
Figura 32 – Gráfico de rugosidade para deposição balística normalizado.	35
Figura 33 – Diagrama de fluxo para o tratamento e seleção dos dados.	36
Figura 34 – 2ª parte dos dados de rugosidade normalizados para deposição balística.	36
Figura 35 – Diagrama do fluxo para a separação dos dados que irão para o modelo.	37
Figura 36 – Dados que serão usados para treinar e validar os modelos de aprendizado de máquina.	37
Figura 37 – Diagrama do fluxo para a interpolação da última parte dos dados. . . .	37
Figura 38 – Curvas geradas pela metodologia.	38
Figura 39 – Gráfico de rugosidade onde cada cor representa cada parte da metodologia.	38
Figura 40 – Diagrama com toda a metodologia para regressão da curva de rugosidade.	39
Figura 41 – Diagrama de fluxo para a metodologia 2.	41
Figura 42 – Curvas aplicando o pré-processamento usando o logaritmo e o negativo do logaritmo.	42
Figura 43 – Curvas simuladas computacionalmente em comparação com as curvas médias geradas via rede neural.	43
Figura 44 – Metodologia para o pré-processamento dos dados de rugosidade.	44
Figura 45 – Diagrama da metodologia para o caso bidimensional.	44
Figura 46 – Regressão para curvas conhecidas usando máquinas de vetores de suporte.	47
Figura 47 – Regressão para curvas conhecidas usando florestas aleatórias.	47
Figura 48 – Regressão para curvas conhecidas usando perceptron multicamadas-1 camada escondida.	48
Figura 49 – Regressão para curvas conhecidas usando perceptron multicamadas-2 camadas escondida.	48
Figura 50 – Extrapolações Usando máquinas de vetores de suporte.	49
Figura 51 – Extrapolações Usando florestas aleatórias.	49
Figura 52 – Extrapolações Com perceptron multicamadas-1 camada escondida. . . .	50
Figura 53 – Extrapolações Com perceptron multicamadas-2 camadas escondidas. . .	50
Figura 54 – Correções de α_L segundo o modelo proposto pelo autor [16].	51
Figura 55 – Curvas geradas pela segunda metodologia comparado com a segunda metodologia.	52
Figura 56 – Correções de α_L para a segunda metodologia em comparação o modelo proposto pelo autor [15].	52

Figura 57 – Previsões da rede neural sem <i>fine tuning</i> para comprimentos até $2^{13} - 2^{19}$.	53
Figura 58 – Previsões da rede neural sem <i>fine tuning</i> para comprimentos entre $2^{20} - 2^{25}$.	53
Figura 59 – Previsões da rede neural com <i>fine tuning</i> 1 para comprimentos até $2^{13} - 2^{19}$.	54
Figura 60 – Previsões da rede neural com <i>fine tuning</i> para comprimentos até $2^{20} - 2^{25}$.	54
Figura 61 – Correções de α para a rede com <i>fine tuning</i> em relação ao modelo proposto pelo autor [16].	55
Figura 62 – Previsões da segunda rede neural para comprimentos entre $2^{13} - 2^{19}$.	55
Figura 63 – Previsões da segunda rede neural para comprimentos entre $2^{20} - 2^{25}$.	56
Figura 64 – Correções de α para o segundo modelo de rede em comparação ao do autor [16].	57
Figura 65 – Previsões da metodologia 2 comparada com a metodologia 1 para os comprimentos $2^{15} - 2^{20}$	59
Figura 66 – Correções de α_L para a primeira e segunda metodologia em comparação o modelo proposto pelo autor [15].	60
Figura 67 – Previsão das curvas de comprimento lateral $2^{13} - 2^{19}$ para os diferentes treinos das redes neurais.	61

Lista de tabelas

Tabela 1 – Tabela dos respectivos modelos	9
Tabela 2 – Tabela com as Características das simulações do modelo de deposição balística unidimensional.	34
Tabela 3 – Tabela com as Características das simulações do modelo de deposição balística bidimensional.	34
Tabela 4 – Hiperparâmetros Máquinas de Vetores de Suporte	40
Tabela 5 – Hiperparâmetros Florestas Aleatórias	40
Tabela 6 – Hiperparâmetros Perceptron Multicamadas 1	40
Tabela 7 – Hiperparâmetros Perceptron Multicamadas 2	40
Tabela 8 – Hiperparâmetros da Rede Neural	41
Tabela 9 – Hiperparâmetros da rede neural para o caso bidimensional	45
Tabela 10 – Tabela das RMSEPE	48
Tabela 11 – Tabela dos valores de alpha	50
Tabela 12 – Tabela com as rugosidades de saturação para cada modelo.	51
Tabela 13 – RMSEPE das curvas de crescimento para ambas metodologias.	51
Tabela 14 – Comparação da rugosidade de saturação em ambas metodologias	52
Tabela 15 – Comparação da rugosidade de saturação para todos os modelos para comprimentos entre $2^{13} - 2^{19}$	56
Tabela 16 – Comparação da rugosidade de saturação para todos os modelos para comprimentos entre $2^{20} - 2^{25}$	56
Tabela 17 – Comparação entre o tempo para a simulação computacional e a metodologia 1.	57
Tabela 18 – Média dos erros de todos os modelos da metodologia 1.	58
Tabela 19 – RMSEPE das curvas de crescimento para ambas metodologias.	59

Lista de Símbolos

Variável	Descrição
w	Rugosidade
t	Tempo
L	Largura do sistema
h	Altura de um determinado sítio
d	Dimensão do sistema
β	Expoente de crescimento
α	Expoente de rugosidade
z	Expoente dinâmico
t_x	Tempo De <i>Crossover</i>
w_{sat}	Rugosidade de saturação
x	Dado de entrada
y	Dado de saída
\hat{y}	Dado de saída do modelo
w	Pesos do modelo
\hat{Y}	Matriz de saída do modelo
Y	Matriz de saída
X	Matriz de entradas
W	Matriz de pesos do modelo
γ	Taxa de aprendizado
∇	Operador gradiente
E	Métrica de erro
G	Gini
θ	Viés
$g(\cdot)$	Função de ativação
σ	Função de ativação <i>Sigmoid</i>
h_t	Estado escondido

Lista de Abreviaturas

RNA	Rede Neural Artificial
BEC	Condensado de Bose-Einstein
IA	Inteligência Artificial
PVD	Deposição a Vapor Física
CVD	Deposição a Vapor Química
KPZ	Kardar-Parisi-Zhang
RD	Deposição Aleatória
RDSR	Deposição Aleatória com Relaxação de Superfície
BD	Deposição Balística
MSE	Erro Médio Quadrático
MAE	Erro Médio Absoluto
RMSE	Raiz do Erro Quadrático Médio
RMSEPE	Raiz do Erro Quadrático Médio Percentual
SMV	Máquinas de Vetores de Suporte
RF	Florestas Aleatórias
ReLU	<i>Rectified Linear Unit</i>
SELU	<i>Scaled Exponential Linear Unit</i>
PMC	Perceptron Multicamadas
RNN	Rede Neural Recorrente
LSTM	<i>Long Short Term Memory</i>
JIT	<i>Just in Time</i>
RI	Realização Independente

Sumário

1	INTRODUÇÃO	1
1.1	Contextualização	1
1.2	Justificativa	2
1.3	Objetivos	3
1.4	Conteúdo do Trabalho	4
2	REVISÃO BIBLIOGRÁFICA	5
2.1	Processos de Deposição	5
2.1.1	Modelo Não Correlacionado	5
2.1.2	Modelo Correlacionado	7
2.1.2.1	Modelo de Deposição Aleatória com Relaxação de Superfície	10
2.1.2.2	Modelo de Deposição Balística	11
2.2	Aprendizado de Máquina	15
2.2.1	Regressão Linear	16
2.2.1.1	Gradiente Descendente	17
2.2.1.2	Método dos Mínimos Quadrados	19
2.2.2	Máquinas de Vetores de Suporte	20
2.2.3	Florestas Aleatórias	21
2.3	Redes Neurais	23
2.3.1	Perceptron	24
2.3.2	Perceptron Multicamadas	26
2.3.3	Redes Neurais Recorrentes	29
2.3.4	Long Short Term Memory	31
3	METODOLOGIA	33
3.1	Simulações dos modelos de deposição	33
3.2	Modelo Balístico Unidimensional	35
3.2.1	Metodologia 1	35
3.2.2	Metodologia 2	40
3.3	Modelo Balístico Bidimensional	43
4	RESULTADOS	47
4.1	Balístico Unidimensional	47
4.2	Balístico Bidimensional	53
4.3	Comentários e Discussão	57
5	CONCLUSÕES	62

6	APÊNDICE	64
6.1	Apêndice A	64
6.2	Apêndice B	65
	REFERÊNCIAS	66

1 Introdução

1.1 Contextualização

Na última década a inteligência artificial revolucionou a forma com a qual lidamos com problemas computacionais. Problemas relacionados a classificação de imagens, a visão computacional, ao reconhecimento de fala, a regressão, a detecção de anomalias, vêm sendo solucionados com sucesso utilizando aprendizado de máquina [1].

Entre os diversos modelos de aprendizado de máquina existentes na literatura, as redes neurais artificiais (RNA) têm sido aplicadas extensivamente a problemas de engenharia e ciências naturais. Na física, por exemplo, as RNAs são aplicadas em diversos campos como: física de partículas, cosmologia, física estatística, computação quântica, física de muitos corpos e ciência dos materiais [2, 3]. Podemos também mencionar o trabalho de P. B. Wigley et al. [4], que reproduziu o experimento que levou ao prêmio nobel de física de 2001. Usando inteligência artificial (IA) e visão computacional, o próprio instrumento de medição conseguiu otimizar o seu arranjo experimental, de forma a reproduzir o experimento do Condensado de Bose-Einstein (BEC) [4]. Além disto, mais recentemente, uma nova classe de *Deep Learning* conhecido como *physics-informed neural networks* vem ganhando destaque, por ser um tipo de rede neural que consegue aproximar soluções de equações diferenciais ordinárias, parciais, lineares ou não lineares sem necessidade de uma solução analítica. Neste modelo, a equação diferencial se torna um agente regularizador, que irá limitar o domínio no qual a rede neural pode extrapolar [5]. Isto é justificável pelo fato de que redes neurais podem ser consideradas como aproximadores universais, uma vez que com apenas uma camada oculta e um número suficiente de neurônios, este pode aproximar qualquer tipo de função [6].

Já é reconhecido que aprendizado de máquina deu grande avanço na química computacional [7]. Os principais processos que foram otimizados com IA, são: o entendimento e o controle de sistemas químicos; o cálculo, a otimização e a predição de propriedades estruturais; a teoria do funcional da densidade e os potenciais interatômicos; a triagem, a síntese e a caracterização de componentes e materiais [7]. Podemos também citar o trabalho realizado por J. Hermann et al. [8], onde utilizou-se uma rede neural profunda para resolver a equação de Schrödinger para uma molécula com trinta elétrons, algo computacionalmente custoso a partir dos métodos tradicionais [8].

O crescimento de superfícies e interfaces é necessário em diversas aplicações nas áreas de ciência e tecnologia, incluindo física, engenharia, química, biologia e ciência de materiais [9]. Diversas propriedades mecânicas, ópticas e eletrônicas de materiais dependem das imperfeições em suas superfícies, sendo estas originadas por *bulk valencies*, deslocamentos e defeitos

topológicos [10, 11]. Aplicações incluem a fabricação de nanomateriais, a manufatura de revestimentos ópticos, dispositivos eletrônicos, e revestimentos de instrumentos [9, 12]. O processo de deposição pode ser classificado em dois tipos distintos, a deposição a vapor física (PVD) e a deposição a vapor química (CVD) [12]. Técnicas de PVD são comumente usadas em laboratórios, estas proveem a possibilidade de preparar uma variedade de filmes finos com diversos materiais, que por consequência, implicarão em diversos parâmetros de deposição, permitindo a síntese de substratos para diversas aplicações [13].

Atualmente existem diversos trabalhos relacionados ao modelo de deposição balística, motivados pelas incertezas na estimação de seus parâmetros, e pela ampla aplicação do modelo ao longo dos anos [14]. O aumento da capacidade dos processadores permitiu com que diversos trabalhos surgissem para calcular computacionalmente os expoentes ligados ao modelo [15, 16, 17], acompanhado de análises teóricas/numéricas dos mesmos para dimensões superiores [18, 19, 20, 11, 21]. Atualmente, ainda há diversos trabalhos que vêm surgindo em torno de processos de deposição balística, isto, pelo custo computacional de simulações em mais de uma dimensão. Além destes, já existe uma variedade de trabalhos que estudam modelos de deposição balística com competição, nos quais, a partícula se agarrará ao seu vizinho com uma probabilidade p [11, 22, 23, 24, 25].

1.2 Justificativa

Essencialmente, quando realizamos as simulações de processos de deposição de partículas, há um total de três estruturas de repetições a saber: a primeira consiste no número de partículas depositadas por unidade de tempo, sendo que, geralmente, esta é proporcional ao tamanho do sistema L ; a segunda que contabiliza cada unidade de tempo de deposição, posto que, quanto maior o comprimento do substrato, maior é o tempo para que este atinja o valor de rugosidade estacionária, e por último, o número de iterações independentes, este é justificado pelo fato de estarmos tratando de processos aleatórios, por conta disto, realizamos diversas médias configuracionais, com o intuito de mitigar o ruído do processo. Assim, para sistemas de interesse, na ordem de $\sim 10^6$ unidades de comprimento, os quais chamamos de limite hidrodinâmico, o número de iterações cresce exponencialmente, tornando a simulação inviável.

Por conta do tempo computacional envolvido para calcular a rugosidade global para tempos de interesse e para larguras de superfícies que vão ao limite termodinâmico, seria possível aplicar métodos de aprendizado de máquina para otimizar este problema? Ou seja, é possível gerar um aprendizado de máquina significativo, de forma que, o algoritmo consiga aprender o padrão da rugosidade para tamanhos de substrato pequenos, e extrapolar estes para tamanhos maiores?

Atualmente, não há simulações computacionais de modelos de deposição balística para comprimento de substrato de tamanhos que tendem ao limite hidrodinâmico, isto, pelo

fato destas terem tempo de execução crescente com o tamanho do substrato e com o tempo. FARNUDI [15] é o autor encontrado até o momento que fez simulações computacionais para o modelo de interesse com maior valor de comprimento lateral, este realiza uma simulação com $L = 2^{16}$. Portanto, o propósito deste aprendizado poderá não somente gerar estes padrões de rugosidade, como também, nos dar uma intuição de como o modelo teórico se comporta quando tratamos esta escala de tamanhos. Além disto, como veremos nos próximos capítulos, ainda existe algumas incertezas em relação aos parâmetros do modelo de deposição balística. Não sabemos como estes se comportam para grandes comprimentos de substrato, portanto, um modelo de aprendizado de máquina pode nos trazer informações importantes a esse respeito.

Este trabalho propõe uma forma alternativa de calcular a rugosidade global para o modelo balístico $d+1$ dimensões, com $d = 1$ e $d = 2$. Atualmente, não há trabalhos registrados que realizem simulações computacionais com comprimentos de substrato de $L = 2^{30}$, isto pelo fato do modelo balístico ser um modelo de difícil convergência, ou seja, é um modelo que requer um tempo computacional elevado. Consequentemente, não se pode afirmar com certeza a classe de universalidade a qual este pertence. Assim, os resultados deste trabalho podem servir para indicar se o modelo balístico pertence de fato a classe de universalidade Kardar–Parisi–Zhang (KPZ). Além disto, não há trabalhos até o momento na literatura que abordem sobre previsões da rugosidade global de modelos de deposição a partir de métodos de aprendizado de máquina.

1.3 Objetivos

Podemos destacar como principal objetivo deste trabalho a aplicação de métodos de aprendizado de máquina para realizar previsões da evolução temporal da rugosidade global para o modelo de deposição balística para tamanhos de substrato que são computacionalmente inviáveis. Para este fim, utilizaremos diversos modelos de aprendizado de máquina, com o intuito de selecionar os melhores modelos de regressão para o problema em questão.

Para atingir o objetivo geral, são necessários passos intermediários, entre os quais podemos citar:

- Compreender o comportamento da rugosidade nos processos de deposição balística em $d+1$ dimensões, com $d = 1$ e $d = 2$;
- Realizar simulações computacionais de modelos de deposição de forma a obter dados para efetuar o aprendizado de máquina;
- Realizar aprendizado supervisionado, de modo que o modelo de aprendizado de máquina consiga aprender o padrão subjacente do processo de crescimento de rugosidade;

- Treinar diferentes modelos de aprendizado de máquina, tais como, as redes neurais, as máquinas de vetores de suporte e as florestas aleatórias para o problema em análise;
- Realizar análise comparativa entre os modelos utilizados de modo a apontar as vantagens e desvantagens de cada um;
- Extrair, a partir dos resultados, os expoentes dinâmicos relacionados ao modelo de deposição balística, e os comparar com os teóricos.

1.4 Conteúdo do Trabalho

O trabalho é organizado na seguinte estrutura: no segundo capítulo, é realizado uma revisão teórica sobre os principais modelos de interesse de deposição de partículas e alguns modelos de aprendizagem de máquina, especialmente aqueles que poderão se tornar possíveis candidatos para a resolução do problema de interesse. No terceiro capítulo, explicamos toda a metodologia empregada no trabalho, evidenciando, os modelos de aprendizado de máquina utilizados, a arquitetura dos modelos para as soluções propostas, o pré-processamento dos dados, e a forma com que validamos os dados extrapolados. O quarto capítulo expõe todos os resultados adquiridos, este contém gráficos de crescimento de rugosidade, tabela de erros entre modelos e valores de parâmetros que desejamos encontrar, no final deste é realizado uma discussão mais profunda acima dos resultados encontrados. Por fim, no capítulo cinco concluímos comentando sobre os resultados, a viabilidade dos dados previstos, e sobre futuros trabalhos a serem desenvolvidos no tema.

2 Revisão Bibliográfica

Este capítulo expõe de forma resumida os conceitos fundamentais sobre processos de deposição e aprendizado de máquina. É apresentado a teoria dos modelos de deposição de partículas e suas principais quantidades de interesse, para em seguida, apresentar os modelos de aprendizado de máquina e seus métodos de treinamento. O principal intuito deste capítulo é realizar uma contextualização sobre como será tratada a informação dos processos de deposição, e quais ferramentas de aprendizado de máquina podem ser empregadas para processar os dados adquiridos.

2.1 Processos de Deposição

Em processos de deposição a principal quantidade de interesse é a rugosidade [16]. Essencialmente, o sistema é composto por um hipercubo d dimensional de largura L . Neste trabalho, estamos interessados em analisar sistemas em $d = 1$ e $d = 2$, ou seja, o substrato é uma reta para o caso unidimensional e um quadrado para o bidimensional.

A cada unidade de tempo serão depositados aleatoriamente uma quantidade de partículas, onde esta quantidade é proporcional ao tamanho lateral do sistema, L , e isto fará com que o substrato cresça verticalmente, assim, a partir da altura de cada sítio podemos calcular a rugosidade $w(L, t)$ usando:

$$w(L, t) = \left\langle \left[\frac{1}{L^d} \sum_i (h_i - \bar{h})^2 \right] \right\rangle^{1/2}, \quad (2.1)$$

onde h_i é a altura da coluna na posição i , \bar{h} é a altura média do substrato e d a dimensão do sistema. Como estamos trabalhando com processos aleatórios, o $\langle \cdot \rangle$ simboliza uma média configuracional sobre as realizações independentes do sistema, ou seja, realizamos o experimento n vezes, de forma a tomar uma média sobre estas n realizações.

2.1.1 Modelo Não Correlacionado

O modelo de deposição aleatório, *Random Deposition* (RD), é o modelo mais simples de deposição de partículas, neste, partículas cairão verticalmente e aleatoriamente na superfície do substrato, tal regra de deposição é evidenciado na figura 1.

O sistema de interesse é definido inicialmente como um substrato “sem partículas”, ou seja, as alturas nos sítios são nulas. A cada unidade de tempo (uma unidade de tempo é o número de partículas depositadas em relação ao comprimento, $t = N/L$), serão depositados um total de L partículas no substrato, fazendo com que a altura nos sítios cresça verticalmente. Como exemplo prático, vamos supor um substrato de tamanho igual

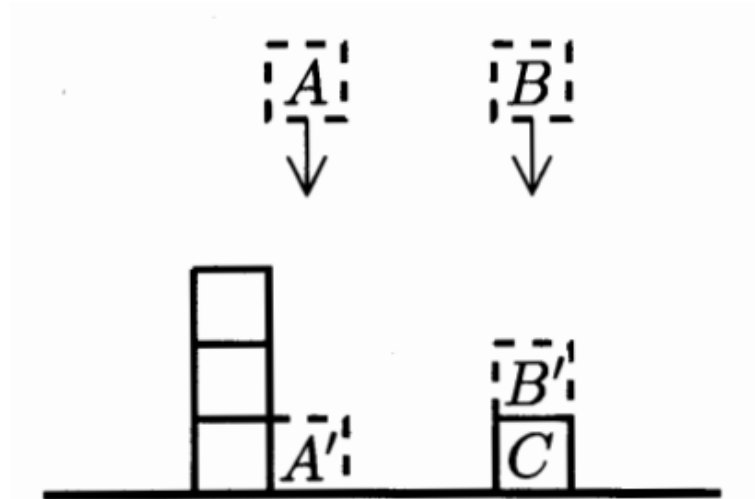


Figura 1 – Regra de deposição de partículas para o modelo de deposição aleatório. (Imagem retirada de [26])

à $L = 100$ e a cada unidade de tempo serão depositados 2000 partículas neste, de modo a gerar o padrão da figura 2, onde cada cor representa uma unidade de tempo. Neste, foi empregado mais partículas de modo a evidenciar o padrão do modelo aleatório.

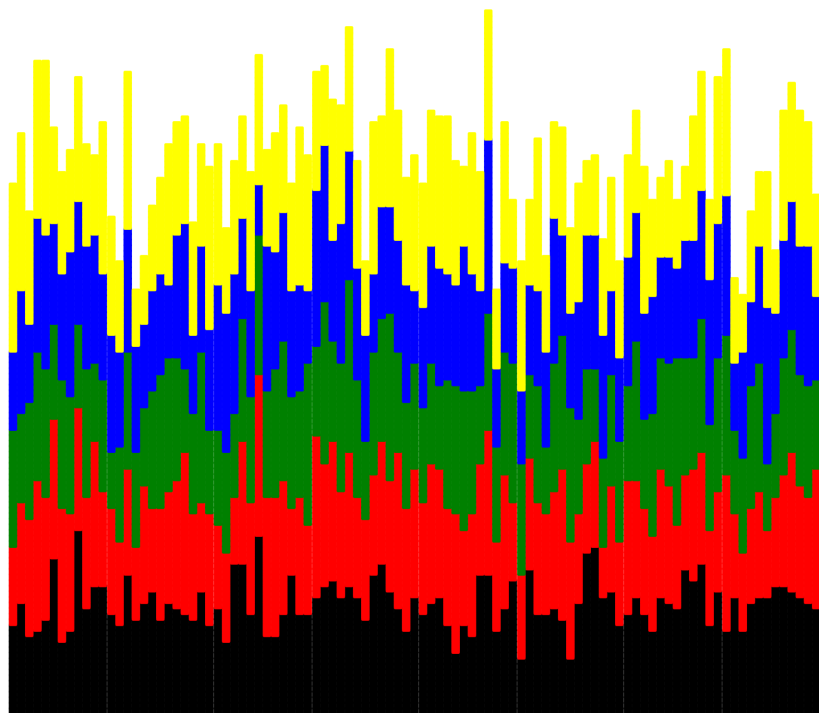


Figura 2 – Ilustração de um substrato de tamanho $L = 100$ após cinco unidades de tempo (cada cor representa uma unidade de tempo).

Já que não há correlação entre as colunas, podemos inferir que a probabilidade de uma partícula ser depositada em uma determinada coluna é $p = 1/L$. Com isto, a probabilidade de uma coluna ter uma altura h após N deposições pode ser descrita a partir de uma distribuição binomial de probabilidade.

$$P(h, N) = \binom{N}{h} p^h (1-p)^{N-h}. \quad (2.2)$$

Calculando o primeiro momento da altura do sistema,

$$\langle h \rangle = \sum_{h=1}^N h P(h, N) = Np = \frac{N}{L} = t, \quad (2.3)$$

o segundo momento,

$$\langle h^2 \rangle = \sum_{h=1}^N h^2 P(h, N) = Np(1-p) + N^2 p^2. \quad (2.4)$$

Além da forma 2.1, podemos calcular a rugosidade a partir do primeiro e segundo momento,

$$w^2(t) = \langle (h - \langle h \rangle)^2 \rangle = \langle h^2 \rangle - \langle h \rangle^2 = Np(1-p) = \frac{N}{L} \left(1 - \frac{1}{L}\right). \quad (2.5)$$

A rugosidade em função do tempo para este modelo tem a forma segundo a figura 3,

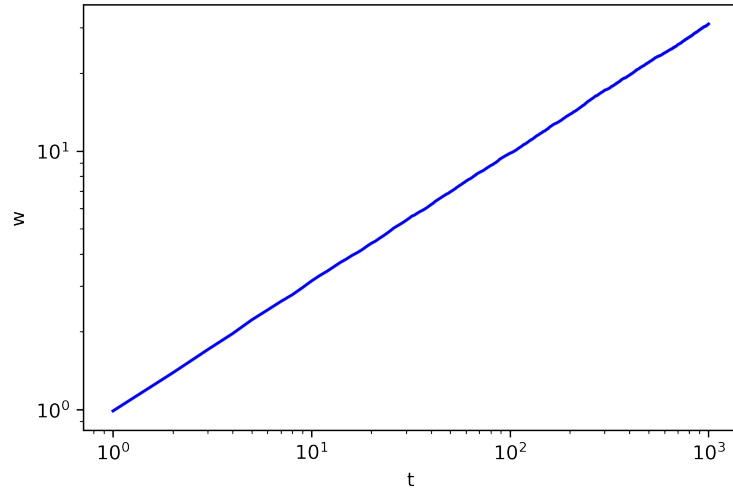


Figura 3 – Gráfico de rugosidade para o modelo não correlacionado.

sendo este padrão esperado, pois cada coluna cresce de forma independente, logo, é previsto que o sistema cresça indefinidamente. Além disto, se olharmos para a equação 2.5, o segundo termo entre parênteses vai à zero para grandes comprimentos, com isto, este crescerá apenas no tempo.

2.1.2 Modelo Correlacionado

No modelo correlacionado assumimos que existe uma correlação entre as colunas, de modo que uma influencie no crescimento da outra. Fundamentalmente, após T unidades de

tempo, a correlação irá afetar o sistema, de modo que a rugosidade permaneça constante no tempo. A evolução temporal da rugosidade global de um modelo correlacionado pode ser visto na figura 4.

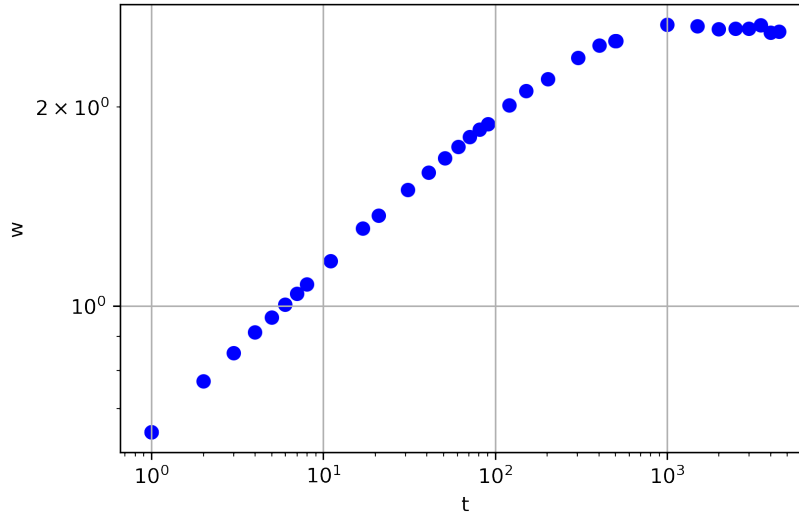


Figura 4 – Crescimento da rugosidade no modelo correlacionado.

A partir da figura 4 podemos definir duas regiões,

- a primeira, que consiste de um crescimento inicial até $t \sim 1000$, em que este cresce de acordo com a lei de potência,

$$w(L, t) \sim t^\beta, \quad (2.6)$$

onde β é definido como expoente de crescimento, e este irá caracterizar a dinâmica de dependência temporal do processo de rugosidade [26]. A rugosidade evoluirá segundo 2.6 até um tempo dito de tempo de *crossover* t_\times ,

$$t_\times \sim L^z, \quad (2.7)$$

em que z é o expoente dinâmico.

- Após o tempo de *crossover*, por conta das correlações entre as colunas, a rugosidade irá saturar segundo,

$$w_{sat}(L) \sim L^\alpha, \quad (2.8)$$

sendo α o expoente de rugosidade.

A princípio, o que foi realizado até o momento, foram simulações computacionais de cada um dos modelos propostos, porém, cada um destes é descrito por uma classe de universalidade, ou seja, cada modelo tem uma equação que descreve a sua dinâmica. A tabela 1 mostra o valor teórico dos expoentes, juntos com suas respectivas classes.

Os expoentes β, α e z são de extrema importância, já que são estes que descrevem a dinâmica de crescimento de cada modelo [26]. Podemos observar que estes variam de

Tabela 1 – Tabela dos respectivos modelos

Nome	equação	Expoente de Escala
Deposição Aleatória	$\frac{\partial h}{\partial t} = \eta(x, t)$	$\beta = 0.5, \alpha$ não definido
Edwards-Wilkinson Equation	$\frac{\partial h}{\partial t} = v\nabla^2 h + \eta(x, t)$	$\alpha = \frac{2-d}{2}, z = 2$
KPZ equation	$\frac{\partial h}{\partial t} = v\nabla^2 h + \frac{\lambda}{2}(\nabla h)^2 + \eta(x, t)$	$d = 1: \alpha = 0.5, \beta = 1/3, z = 1.5$

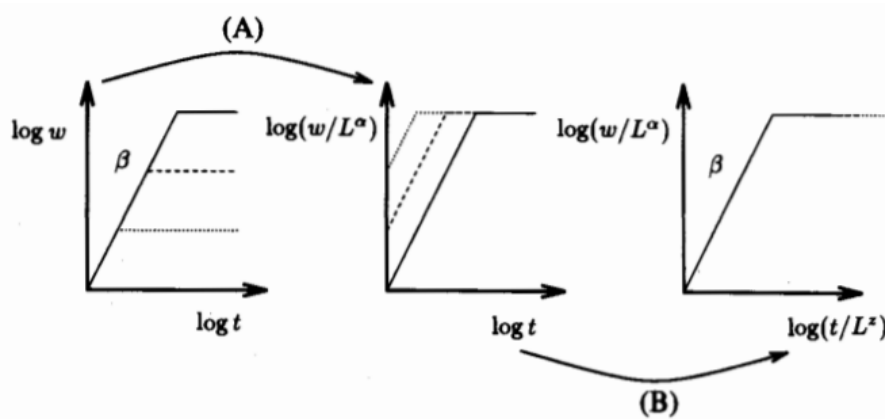
acordo com a classe de universalidade, sendo os resultados da tabela 1, resultados analíticos, soluções numéricas serão discutidas mais a frente. Além disto, esses valores são únicos para cada modelo, e a partir destes, pode-se clamar sua devida classe de universalidade. Em geral, os expoentes dinâmicos não são independentes, e seguem a seguinte relação,

$$z = \frac{\alpha}{\beta}, \quad (2.9)$$

para a classe de universalidade KPZ ainda temos,

$$z = 2 - \alpha. \quad (2.10)$$

A partir das equações 2.6 à 2.10, é possível perceber que diferentes tamanhos de substrato resultarão em curvas de evolução temporal da rugosidade global distintas. FAMILY [27] propõe uma forma de normalizar estas curvas de rugosidade para diferentes comprimentos laterais de um determinado modelo, de modo que estas colapsem em apenas uma, a figura 5 expõe este procedimento.

**Figura 5** – Procedimento para normalizar as curvas de rugosidade. (Imagem retirada de [26])

O passo A consiste em dividir a rugosidade pelo comprimento do sistema elevado ao α , e aplicar uma função logarítmica. O passo B é a divisão do tempo pelo comprimento elevado a z , e aplicar uma função logarítmica. Os expoentes α e z são constantes com valores teóricos e computacionais já bem definidos para o caso unidimensional [16, 15].

Para o modelo balístico, que supostamente é regido pela equação KPZ, os valores teóricos de α , β e z são respectivamente 0,5, 0,333 e 1,5.

2.1.2.1 Modelo de Deposição Aleatória com Relaxação de Superfície

O segundo modelo de deposição estudado foi o modelo de deposição aleatória com relaxação de superfície, *Random Deposition with Surface Relaxation* (RDSR). Este obedece as mesmas regras que o modelo de deposição aleatório, com o complemento de que se um dos vizinho laterais possuir uma altura menor que a do sítio no qual a partícula esta caindo, este irá se descolar para um destes vizinhos, tal regra está exposta na figura 6. Simulando este sistema com auxilio da equação 2.1, ficaremos com o gráfico de rugosidade global exposto na figura 7.

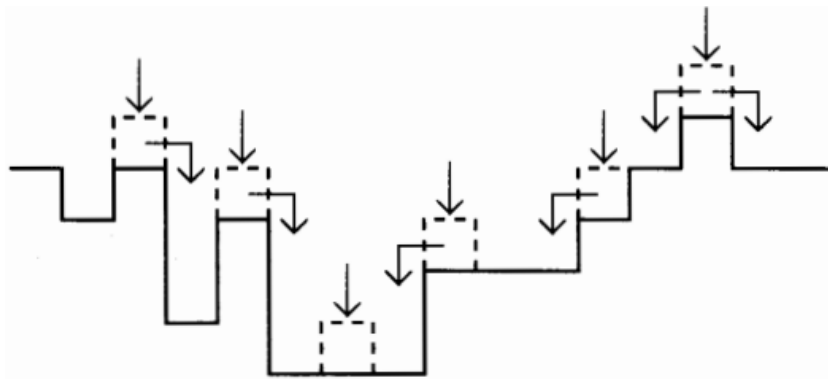


Figura 6 – Regra de deposição para o modelo RDSR.(Imagem retirada de [26])

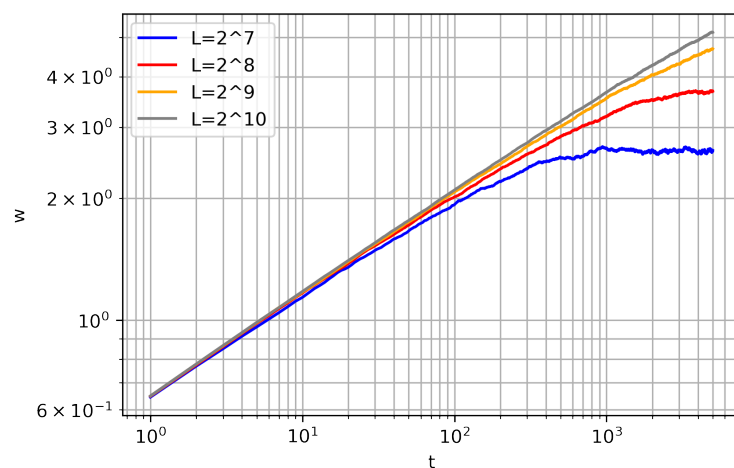


Figura 7 – Gráfico de rugosidade para deposição aleatória com relaxação de superfície.

Por conta desta relaxação, as partículas, a medida que chegam ao substrato, compararam a altura dos vizinhos laterais antes de grudar em algum destes, fazendo com que a superfície se torne mais suave quando comparado com o modelo aleatório. O padrão após 10 unidades de tempo, com $N = 800$ e $L = 100$ pode ser observado na figura 8, onde cada cor representa uma unidade de tempo.

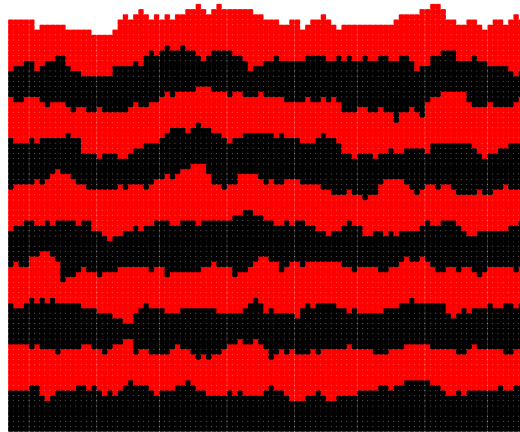


Figura 8 – Ilustração do substrato após 8000 deposições para o modelo de deposição aleatória com relaxação de superfície (cada cor representa uma unidade de tempo).

Essa correlação que existe entre as colunas levará à uma saturação da interface, ou seja, a variação será constante com o tempo, algo que pode ser visto na figura 7 para o tamanho lateral de 2^7 , após $t > 1000$.

2.1.2.2 Modelo de Deposição Balística

A deposição balística segue a mesma ideia que a aleatória, com o adicional, de que as partículas irão se grudar no vizinho lateral de maior altura, tal processo está exposto na figura 9.

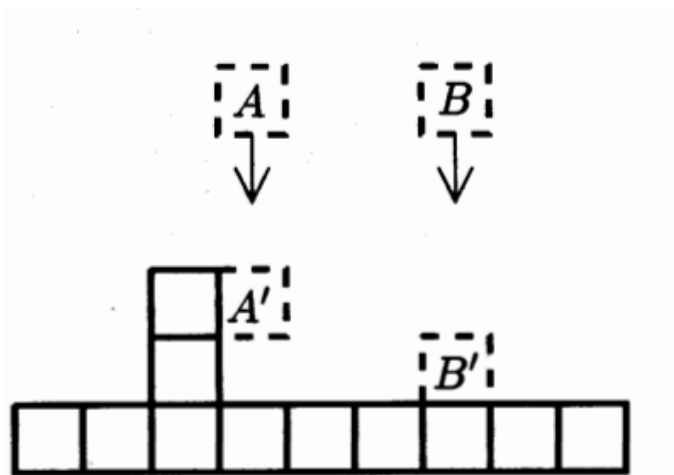


Figura 9 – Regra de Deposição para o modelo de deposição balística.(Imagem retirada de [26])

A deposição Balística foi introduzida como um modelo de agregados coloidais, e estudos iniciais se concentram nas propriedades de agregados porosos produzido pelo modelo [26]. O padrão de rugosidade após $t = 10$, $L = 100$ e $N = 800$ pode ser visto na figura 10. Um gráfico da evolução temporal da rugosidade global por unidade de tempo é ilustrado na figura 11.

É possível perceber que existe um padrão no crescimento da rugosidade global, todas as curvas crescem inicialmente de forma similar, isto, pelo fato das partículas estarem caindo

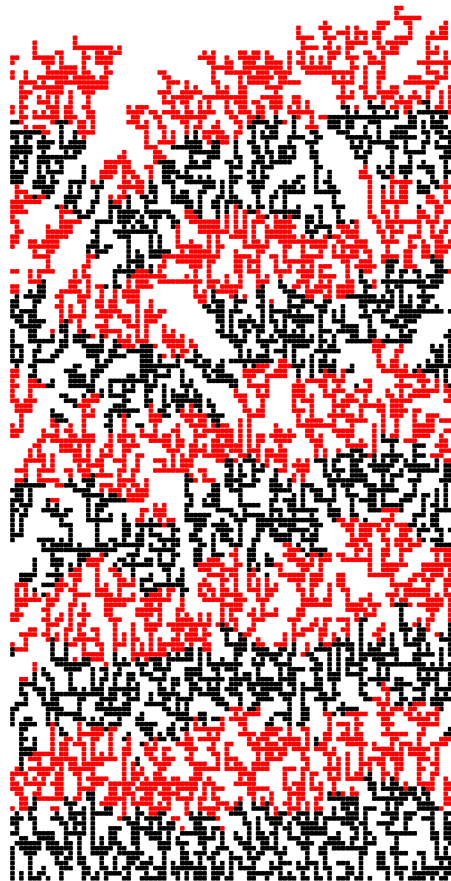


Figura 10 – Crescimento do substrato na deposição balística.

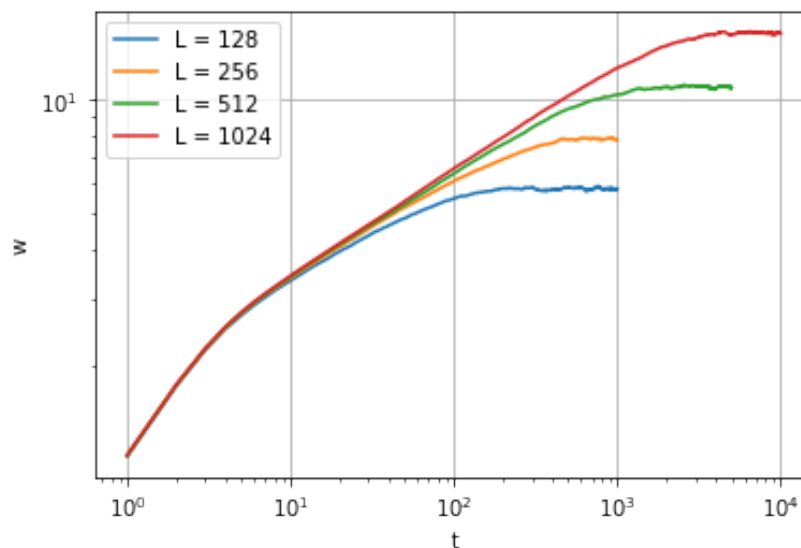


Figura 11 – Gráfico de rugosidade para a deposição balística unidimensional para “pequenos” comprimentos.

verticalmente e preenchendo as colunas vazias, como se o modelo estivesse seguindo uma deposição aleatória. Após tal, a rugosidade começa a crescer de acordo com a equação 2.6 até chegar a um ponto onde as colunas começaram a "se enxergar", saturando a rugosidade.

Por conta de efeitos de pequenos comprimentos, as curvas da figura 11 não convergem

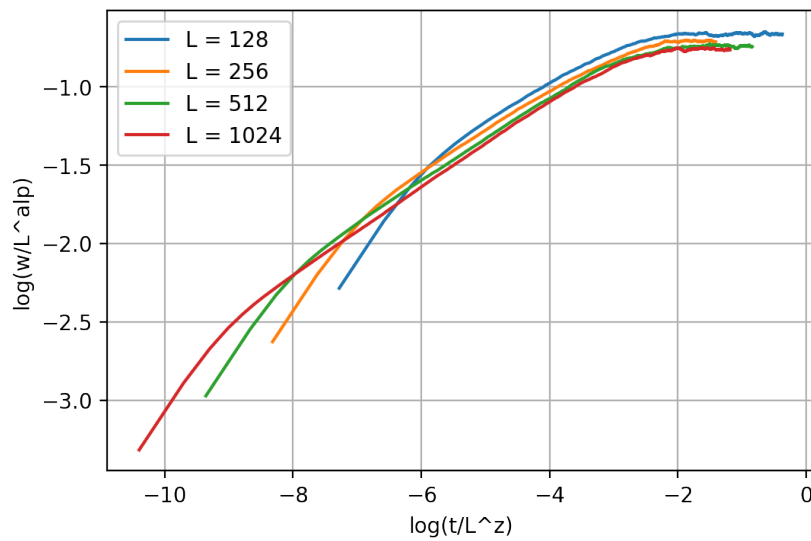


Figura 12 – Gráfico de rugosidade normalizada para a deposição balística unidimensional para "pequenos" comprimentos.

quando aplicado o *Scaling* de Family-Viksek com os expoentes dinâmicos teóricos do modelo KPZ, tal pode ser observado na figura 12. Por conta disto, foram realizadas simulações para comprimentos laterais maiores, tais estão exposto na figura 13.

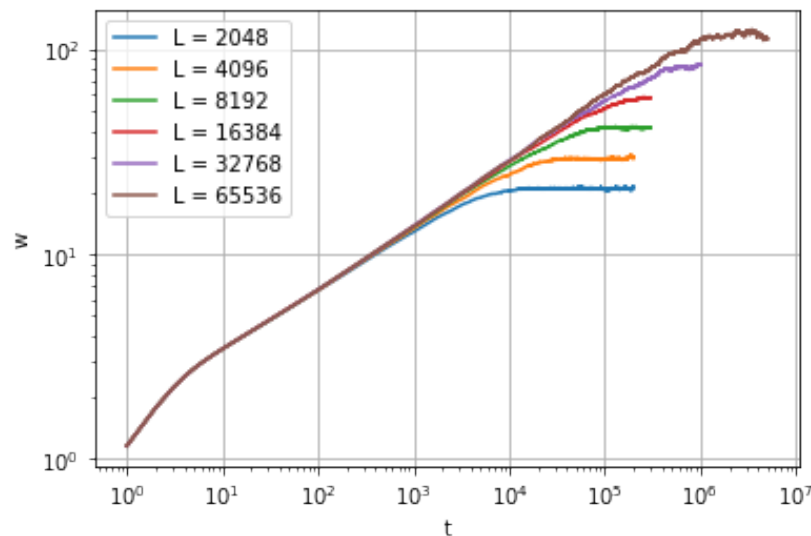


Figura 13 – Gráfico de rugosidade para a deposição balística unidimensional para "grandes" comprimentos.

Tomando o *Scaling* de Family-Vicsek com os expoentes dinâmicos da classe de universalidade KPZ nas curvas da figura 13, ficaremos com as curvas colapsadas da figura 14.

O caso bidimensional se torna mais complexo computacionalmente, já que, o custo computacional de cada curva cresce exponencialmente, isto se dá ao fato de estarmos trabalhando com duas dimensões, logo, a partir do algoritmo situado no anexo 6.2 é possível notar o último laço de repetição vai até L^2 , diferente do caso unidimensional no anexo 6.1.

Por conta desta complexidade, não foi realizado muitas realizações independentes nas

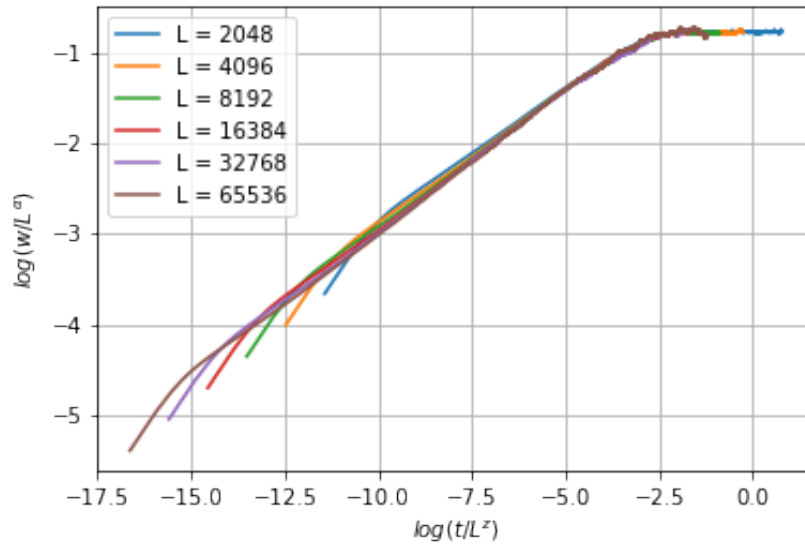


Figura 14 – Gráfico de rugosidade normalizado para a deposição balística unidimensional para "grandes" comprimentos.

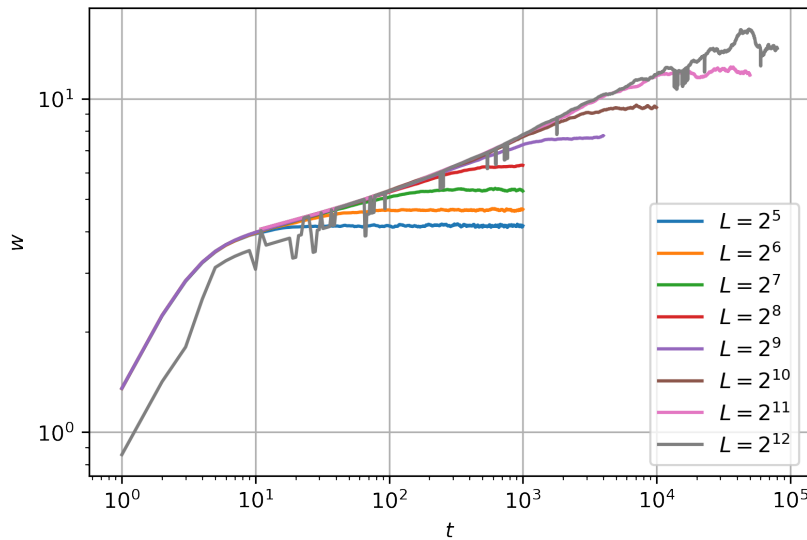


Figura 15 – Gráfico de rugosidade para deposição balística bidimensional.

simulações, gerando curvas com uma grande quantidade de ruído, tais estão expostas na figura 15.

Quando tentamos normalizar as curvas do caso bidimensional, as mesmas não colapsam totalmente quando aplicado o *Scaling* de Family-Vicsek com os expoentes teóricos do modelo KPZ, similar as curvas do modelo balístico (1+1)D para pequenos comprimentos, figura 12. Por conta disto, REIS [16], propõe um fator de correção para o valor de α que assume a seguinte forma,

$$\alpha_L = \alpha + \frac{B}{L\Delta}, \quad (2.11)$$

sendo

$$\alpha_L = \frac{\ln(w'_L/w'_{L/2})}{\ln 2} \quad (2.12)$$

com w'_L sendo a rugosidade de saturação para o comprimento L , B uma constante e Δ “*correction-scaling exponent*”. Assim, ao obtermos curvas com rugosidade estacionária, podemos comparar os valores de α_L obtidos pelas curvas extrapoladas e comparar por aqueles propostos pelo autor.

2.2 Aprendizado de Máquina

Geralmente quando falamos sobre aprendizado de máquina, as pessoas tendem a ter duas linhas de pensamento sobre o assunto, uma consiste em um robô mordomo, sendo este inteligente e confiável, como o Jarvis, e a segunda, um robô que possa eventualmente entrar em conflitos com os humanos [28].

Na realidade, aprendizado de máquina não é apenas uma fantasia futurista, inicialmente foi introduzido como um modelo de reconhecimento Ótico de Caracteres em algumas publicações, contudo, o primeiro software que o alavancou foi o filtro de spam [28].

Assim, podemos definir aprendizado de máquina como a área da computação na qual os computadores aprendem com os dados [28].

Arthur Samuel, 1959, define como: "aprendizado de máquina é o campo de estudo que dá aos computadores a habilidade de aprender sem ser explicitamente programado." [28]

Tom Mitchell, 1997, traz uma definição mais voltada aos engenheiros: "Diz-se que um programa de computador aprende pela experiência e em relação a algum tipo de tarefa T e alguma medida de desempenho P se seu desempenho em T , conforme medido por P , melhora com a experiência E ." [28]

Formalmente, aprendizado de máquina é um subcampo de inteligência artificial e ciência cognitiva, que ainda pode ser dividido em três ramos: aprendizado supervisionado, aprendizado não-supervisionado e aprendizado por reforço [29].

O aprendizado supervisionado é aquele no qual é fornecido tanto os dados de entrada, como os de saída para comparação, sendo estes últimos, chamados de rótulos [28]. Técnicas inclusas neste são: a regressão linear, as máquinas de vetores de suporte, as florestas aleatórias, o perceptron multicamadas [28].

No aprendizado não supervisionado, a única coisa que temos disponível são os dados de entrada, ou seja, não há rótulos nos dados, e o intuito é retirar informações em cima destes. Aplicações incluem: agrupamento, visualização e diminuição de dimensionalidade, aprendizado da regra da associação [28].

Por último, temos o aprendizado por reforço, neste, teremos um agente que será susceptível ao ambiente, e em cima deste, irá efetuar ações e obter recompensas em troca -

ou penalidades na forma de recompensas negativas [28].

2.2.1 Regressão Linear

A regressão linear é o modelo de regressão mais simples que existe. Ele consiste em realizar uma soma ponderada em cima dos dados de entrada, e mais uma constante chamada de termo de polarização (ou mais conhecido como coeficiente linear) [28]. Sua expressão matemática para uma única entrada é mostrada abaixo,

$$\hat{y} = w_1x_1 + w_0, \quad (2.13)$$

- \hat{y} é o valor previsto;
- x_1 é o valor de entrada;
- w_0 e w_1 são os pesos do modelo,

este modelo pode ser expandindo para N entradas,

$$\hat{y} = w_Nx_N + \dots + w_2x_2 + w_1x_1 + w_0, \quad (2.14)$$

$$\hat{y} = \sum_{i=1}^N w_i x_i + w_0. \quad (2.15)$$

Uma forma alternativa de representar as entradas, as saídas e os pesos é na forma matricial, em que teremos,

$$\hat{Y} = X \cdot W^T, \quad (2.16)$$

nesta abordagem o primeiro elemento do vetor de entradas $X[0] = x_0 = 1$.

Assim, em problemas de regressão, geralmente, temos uma quantidade de valores de entrada x , e de saída y , contudo, não possuímos os pesos w do modelo desejado, sendo estes necessário para modelar as entradas o mais próximo possível das saídas.

Podemos inferir que um problema de regressão é um problema de encontrar os melhores valores de w , de modo que o valor de saída prevista do modelo \hat{y} seja a mais próxima possível saída real y . Para este fim, existe duas formas distintas de se resolver este problema de encontrar o melhor conjunto de pesos para a regressão linear, sendo estes:

- Gradiente Descendente;
- Método dos Mínimos quadrados.

2.2.1.1 Gradiente Descendente

Gradiente Descendente é um algoritmo de otimização iterativo que tem como objetivo encontrar o máximo e mínimo de funções usando suas derivadas. A ideia é empregar uma função custo (*Loss Function*) $E(y, \hat{y})$, sendo esta, uma métrica para inferir se o modelo está próximo do desejado, e a partir destes, encontrar os melhores valores de w de forma a minimizar esta função. A função custo pode assumir diversas formas, isto irá depender principalmente do problema. Por exemplo, podemos ter as seguintes funções custo:

1. Erro Quadrático Médio/*Mean Square Error* (MSE)

$$MSE(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2, \quad (2.17)$$

2. Erro Quadrático Absoluto/*Absolute Square Error* (ASE)

$$ASE(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^N |y_i - \hat{y}_i|, \quad (2.18)$$

3. Raiz do Erro Quadrático Médio/*Root Mean Square error* (RMSE)

$$RMSE(y, \hat{y}) = \sqrt{\frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2}. \quad (2.19)$$

Para problemas de regressão, a função custo geralmente empregada para treinar modelos de aprendizado de máquina é a MSE. Para medir a performance do algoritmo pós-treino é usado o RMSE ou RMSEPE, onde este último é o RMSE Percentual.

$$RMSEPE(y, \hat{y}_i) = \sqrt{\frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2} \times 100\%. \quad (2.20)$$

Definido a métrica, podemos definir o funcionamento do algoritmo. Temos a função de custo, e queremos minimizar esta para ter o modelo com um valor de erro tolerável, a ideia é usar o gradiente da função em relação aos seus parâmetros, e a partir deste, altera os pesos de forma a descer a função até o ponto que o gradiente seja zero, este ponto será um possível ponto de mínimo, a ideia deste algoritmo pode ser observado na figura 16.

O tamanho dos passos, ou taxa de aprendizado ε , é um fator importante, pois este fará com que o algoritmo chegue mais rapidamente no seu mínimo. Passos muito grandes podem fazer com que o algoritmo oscile ao redor de um ponto, ou divirja, e passos muito pequenos podem fazer com que o algoritmo leve um tempo proibido para chegar em tal. Não existe uma fórmula propriamente dita para saber qual o valor de ε usar, este é determinado empiricamente.

A partir disto, a atualização da matriz de pesos se dará da seguinte forma:

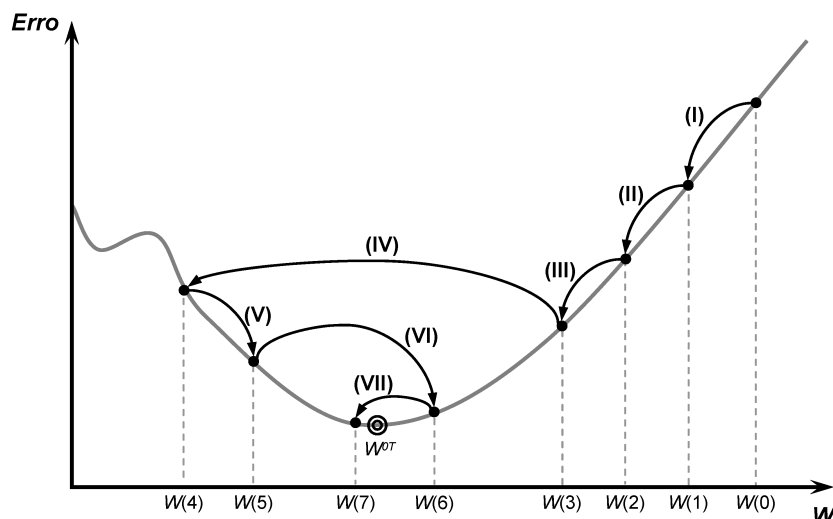


Figura 16 – Ilustração da evolução da curva a cada N iterações.(Imagem retirada de [33])

$$W = W - \varepsilon \nabla_W E(y, \hat{y}), \tag{2.21}$$

sendo ∇E o gradiente da função de custo. O gradiente é empregado de forma a atualizar os pesos do modelo até o ponto no qual o mesmo seja nulo, isto implicará que o modelo atingiu um possível valor de mínimo. Existem diversos métodos para otimizar a convergência do gradiente descendente, tais como, o gradiente descendente em Lotes, o Gradiente descendente Estocástico, momentum, Adagrad, Adam [28].

Para ilustrar o conceito acima, vamos supor os pontos da figura 17. Aplicando o método do gradiente descendente nestes dados, ficaremos com a figura 18, onde este apresenta algumas curvas de aprendizado a partir do número de iterações.

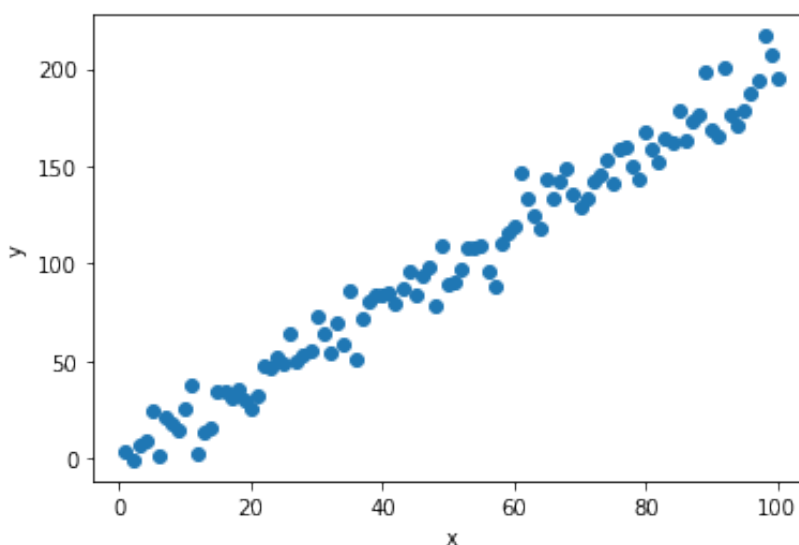


Figura 17 – Pontos gerado a partir de uma função linear com ruído.

A partir da figura 18 pode-se observar que quanto maior N (número de iterações), menor a soma das distâncias entre os pontos e a reta. Assim, a cada ciclo o algoritmo irá

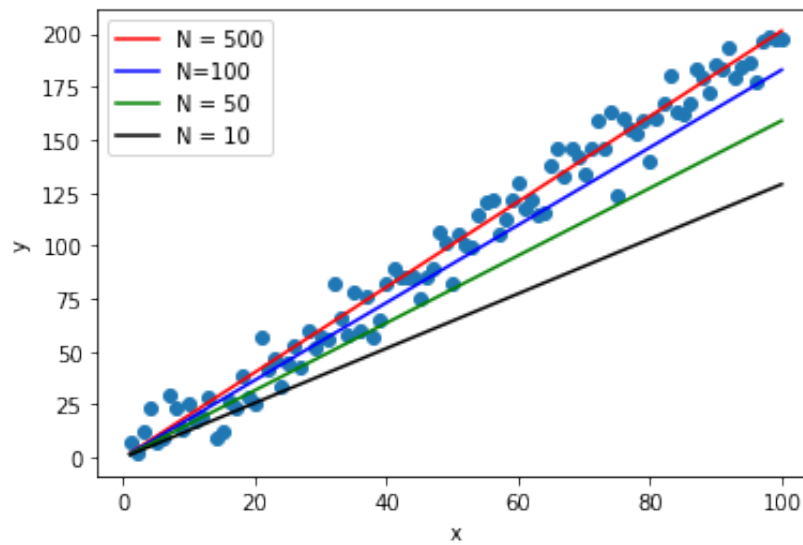


Figura 18 – Ilustração da evolução do modelo a cada N iterações.

atualizar o conjunto de pesos de forma a minimizar o valor da função custo.

2.2.1.2 Método dos Mínimos Quadrados

O método dos mínimos quadrados, consiste em uma equação direta de “forma fechada” que calcula diretamente os pesos do modelo que melhor se encaixam no conjunto de treino [28] (ou seja, estes valores de peso após o uso do método, serão aqueles que minimizam a função de custo).

A forma matemática do método pode ser escrita como:

$$W = (X^T X)^{-1} X^T Y, \quad (2.22)$$

Aplicando o método dos mínimos quadrados nos pontos da figura 17, ficaremos com a curva da figura 19.

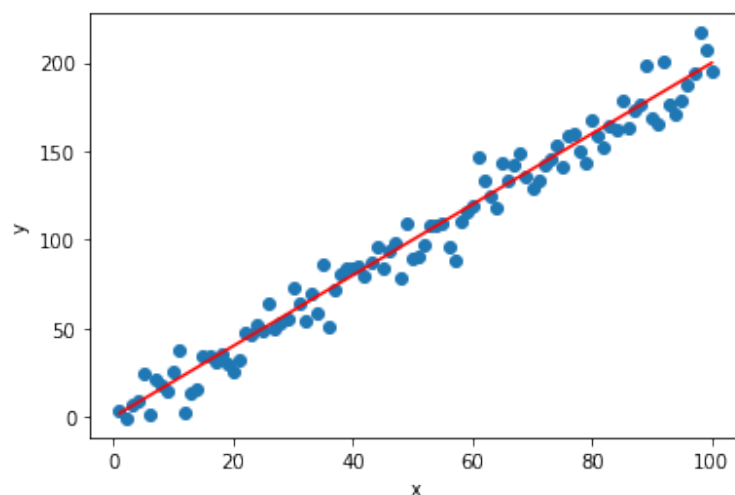


Figura 19 – Curva gerada usando o método dos mínimos quadrados.

A vantagem deste método, é que ele é direto, com apenas uma iteração é possível encontrar o melhor conjunto de pesos de modo a minimizar a função de custo.

Vimos dois tipos de métodos para chegar na reta que melhor engloba os dados de treino, porém fica o questionamento, qual dos dois métodos utilizar? Isto irá depender principalmente da quantidade de dados que temos a disposição. Caso tenhamos muitos dados, é recomendado utilizar o método do Gradiente, pois calcular o inverso da matriz no método dos mínimos quadrados pode ser computacionalmente custoso, enquanto que no Gradiente irá essencialmente depender da forma com o qual os dados estão disposto e da taxa de aprendizado.

2.2.2 Máquinas de Vetores de Suporte

Máquinas de vetores de suporte, *Support Vector Machines* (SVM), é um método de Aprendizado supervisionado que foi introduzido em 1990 por Vpnik e colegas. Este modelo de aprendizado é focado em classificação binária, porém, diversos problemas práticos podem ser modelados como problemas de classificação binária [30]. Inicialmente, o SVM foi introduzido como um modelo de classificação, porém com o tempo, foi notado que este era capaz de resolver problemas de regressão linear e polinomial [31].

A depender do problema de classificação, o SVM pode obter uma acurácia tão boa quanto as redes neurais [32]. Como o problema deste trabalho é um problema de regressão, iremos analisar a ideia básica de SVM para classificação, para em seguida partir para o SVM para regressão.

A ideia principal do SVM é gerar um hiperplano que consiga distinguir duas classes distintas, e além disto, gerar mais dois hiperplanos de suporte, de modo que estes últimos consigam gerar uma margem de distância entre as classes e a curva principal (este é chamado de classificação de margens largas), nos fornecendo um limite de decisão [28].

A figura 20 ilustra um exemplo das SVM. As curvas tracejadas são as curvas de suporte e a curva contínua é a de classificação.

A SVM é baseada na função:

$$f(x) = \text{sign}(w \cdot x + b), \quad (2.23)$$

de forma que se tomarmos dois pontos na fronteira de decisão, x^+ e x^- , teremos,

$$w \cdot x^+ + b = +1, \quad (2.24)$$

$$w \cdot x^- + b = -1, \quad (2.25)$$

e a margem pode ser escrita como

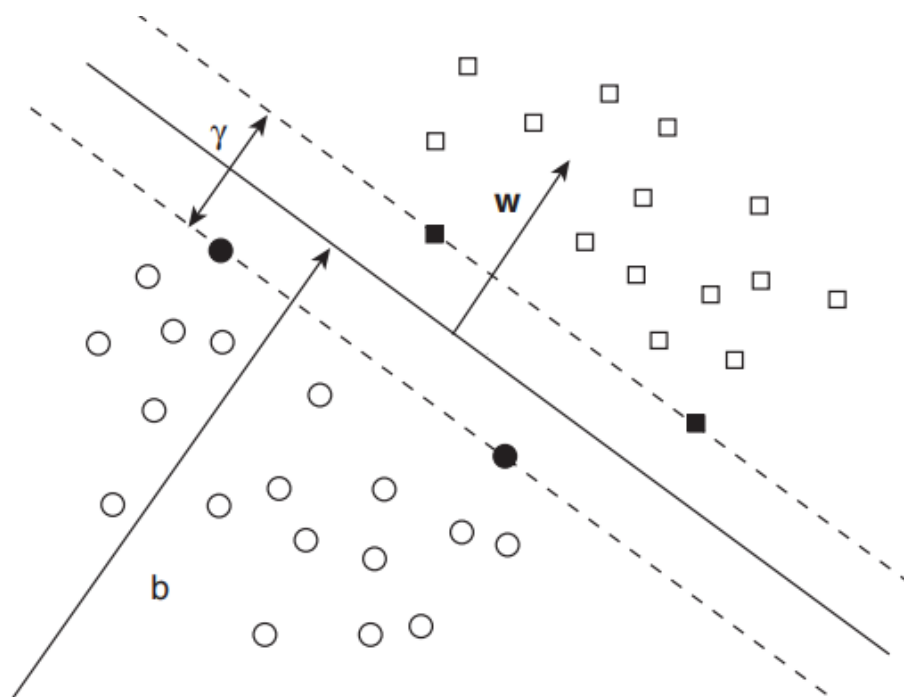


Figura 20 – Curvas geradas usando a máquina de vetores de suporte. (Imagem retirada de [31])

$$\gamma = \frac{2}{\|w\|} \cdot (x^+ - x^-) = \frac{2}{\|w\|}. \quad (2.26)$$

Um aspecto importante da SVM é que esta é sensível a escala das entradas, ou seja, se tivermos pontos com diferentes valores de escala, a SVM terá dificuldades em classificar corretamente [28]. Além disso, existe um problema na classificação de margem larga, que é susceptibilidade a ruído nos dados.

É preferível utilizar um modelo mais flexível, a fim de evitar este problema. A ideia é ter um equilíbrio entre o tamanho da via (valor de γ), de forma a limitar o máximo possível as violações de margem (as instâncias que acabam no meio da via, ruído dos dados). Isto é chamado de classificação de margem suave [28].

A regressão SVM tem uma ideia inversa ao de classificação. O truque consiste em reverter o objetivo, ao invés de tentar gerar a maior via possível entre as duas classes, ou seja, uma distância máxima entre as curvas de suporte e a curva principal, a regressão SVM tenta preencher o maior número de instâncias possíveis na via, a partir disto, gerar curvas de suporte que consigam englobar todos os dados na via entre as curvas, limitando as violações de margem [28].

2.2.3 Florestas Aleatórias

A Árvore de decisão é um componente fundamental para as Florestas Aleatórias, sendo esta, uma das mais poderosas técnicas de aprendizado de máquina disponíveis [28].

A ideia da árvore de decisão é similar a da árvore binária, começamos com o topo da

árvore que chamamos de nó raiz, e vamos descendo segundo as condições de cada nó filho. Assim, a cada exemplo de treino, a árvore vai atualizando seus parâmetros para cada novo dado de entrada. Uma das principais vantagens da árvore de decisões, é que esta não precisa necessariamente de um pré-processamento de dados [28].

Vamos considerar a figura 21, esta expõe uma árvore de decisões para o conjunto de dados Iris (exemplo tirado de [28]).

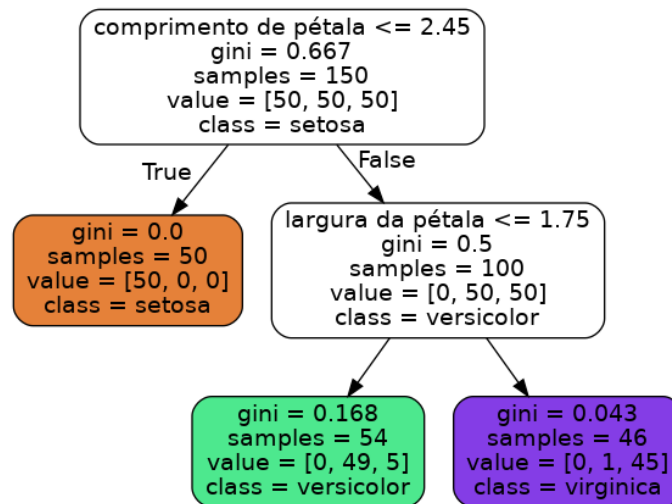


Figura 21 – Exemplo de árvore de decisão para o conjunto de dados Iris.

Primeiro, iremos desmembrar as informações de cada nó. **Samples** é o número de instâncias classificadas pelo modelo, no exemplo acima, temos 100 instâncias classificadas como versicolor e 50 como setosa. **Value** é a quantidade de instâncias de treinamento que se aplicam naquele nó, o primeiro da esquerda, foi classificado 50 Iris-setosa, 0 Iris-versicolor e 0 Iris-virgínica. Por fim, o atributo **gini** de um nó mede sua impureza (um nó é puro, **gini** = 0, se todas as instâncias de treino pertencem a sua respectiva classe).

A equação que descreve o atributo gini é:

$$G = 1 - \sum_{k=1}^n p_{i,k}^2, \quad (2.27)$$

sendo $p_{i,k}$ a média das instâncias da classe k entre as instâncias de treino no nó i.

As árvores de decisões também podem ser usadas para resolver problemas de regressão. Vamos supor que seja desejado realizar uma regressão em cima dos dados utilizados na regressão linear da sub secção anterior. A árvore de decisão neste caso está exposta na figura 22.

Podemos perceber que a árvore da figura 22 é similar ao de classificação, com a diferença que, em cada nó, ao invés de prever uma classe, prevê um valor. Neste, temos uma árvore de profundidade igual a 2, assim, gerando o gráfico da figura 23.

É notável que este modelo não tem precisão para prever novos valores, já que este mal consegue prever os dados de treino, portanto, para melhorar este, podemos aumentar a

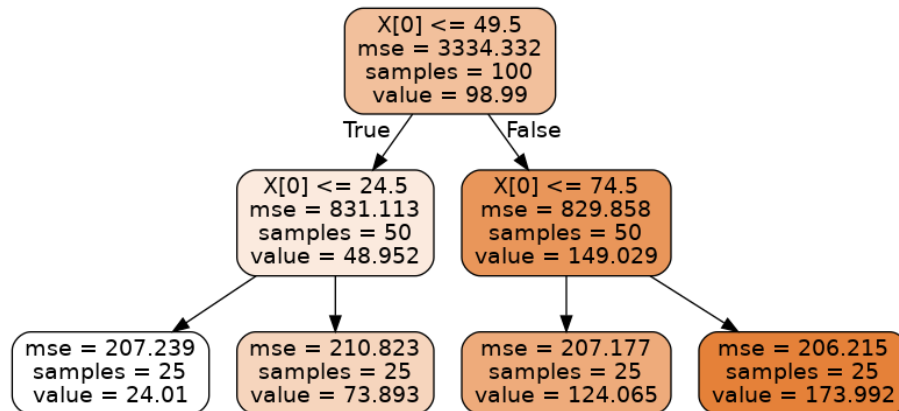


Figura 22 – Árvore de decisões para os pontos gerados anteriormente.

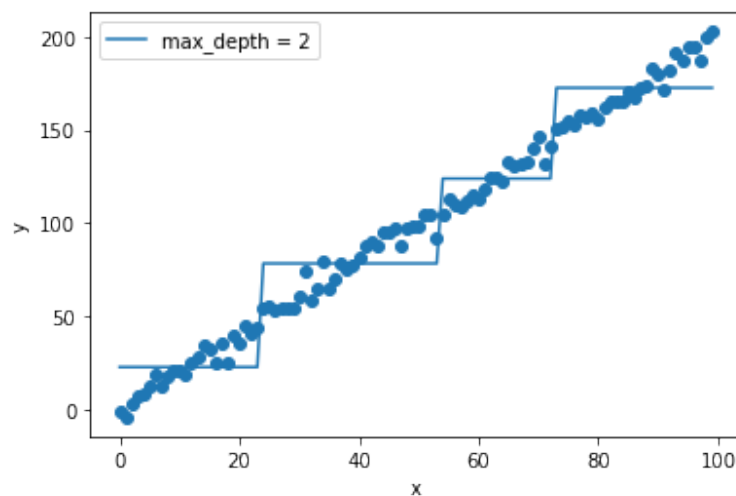


Figura 23 – Ilustração da curva gerada pela árvore de decisão.

profundidade da árvore, melhorando seu desempenho até um certo ponto.

Definido uma árvore de decisão, uma floresta aleatória nada mais é do que um ensemble de árvores de decisão, ou seja, iremos escolher uma quantidade de árvore de decisões com diferentes hiperparâmetros (profundidade da árvore, número mínimo de amostras), e treinar estes com o conjunto de dados de treino misturados a partir de uma dada técnica de amostragem. Após o treino, podemos definir algum critério de decisão final, como, tomar a média de todas as arvores, ou o valor com maior frequência.

2.3 Redes Neurais

As redes neurais artificiais(RNA) foram introduzidas em 1943 pelo neurofisiologista Warren McCulloch e pelo matemático Walter Pitts. Em seu artigo "Logical Calculus of Ideas Immanent in Nervous Activity", eles apresentam um modelo computacional simplificado utilizando lógica proposicional de como os neurônios biológicos podem trabalhar juntos em cérebro de animais na realização de cálculos complexos [28]. Esta foi a primeira arquitetura de rede neural artificial criada. Durante um tempo esta área de pesquisa despopularizou

por conta do custo computacional, além de que diferentes técnicas começaram a surgir apresentando resultados satisfatórios e com melhores custo computacionais, tais como Máquinas de vetores de suporte e a lógica nebulosa [28].

Redes neurais são modelos computacionais inspirados no sistema nervoso dos seres vivos [33]. O sistema nervoso humano é composto por diversas células, sendo estas referidas como neurônios. Os neurônios são conectados entre si a partir dos axônios e dendritos. Nestes, pulsos elétricos são passados pelos axônios até o corpo celular do neurônio onde estes serão ativados, se somente se, as somas destes pulso cheguem a valor de pico, com isto, o neurônio irá disparar um sinal que irá percorrer outros milhares de neurônios [33].

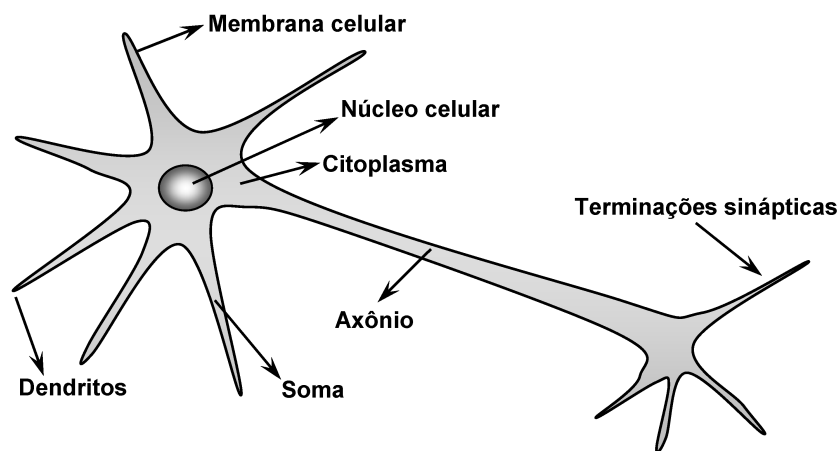


Figura 24 – Ilustração simplificada do neurônio biológico. (Imagem Retirada de [33])

Ao longo dos anos diversas aplicações surgiram do uso de redes neurais, tais como:

- Avaliação de Imagens;
- Classificações de padrões de escrita e fala;
- Reconhecimento de faces em visão computacional;
- Controle de trens de grande velocidade;
- Previsão de ações no mercado financeiro;
- Identificação de anomalias em imagens médicas;
- Identificação automática de perfis de crédito para clientes de instituições financeiras;
- Controle de aparelhos eletrônicos e eletrodomésticos.

2.3.1 Perceptron

O perceptron, que foi idealizado em 1958 por Rosenblatt, é a forma mais simples de configuração de rede neural artificial. O seu principal objetivo consistia em identificação de padrões geométricos, sendo inspirado na retina humana [33].

Pelo fato do perceptron ter somente uma camada neural e um único neurônio, este é limitado à problemas de classificação/regressão lineares. A célula principal do perceptron é o neurônio, o qual podemos representar matematicamente, da seguinte forma:

$$u(x) = \sum_i^n w_i x_i - \theta, \quad (2.28)$$

$$\hat{y} = g(u(x)) \quad (2.29)$$

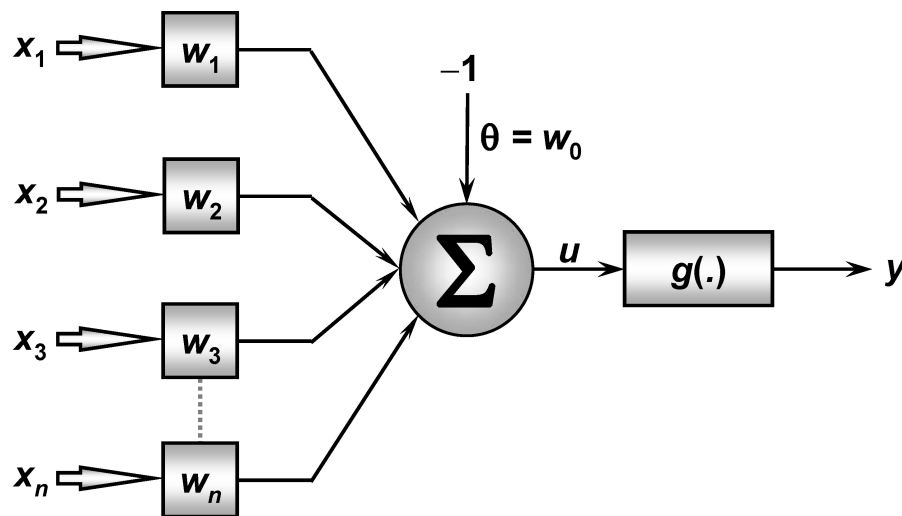


Figura 25 – Ilustração Matemática do neurônio artificial.(Imagem Retirada de [33])

assim, de forma similar com qual o neurônio humano recebe impulsos elétricos e os ativa caso a soma deste passe um certo limiar, o neurônio artificial realiza uma soma ponderada em cima das entradas x_1, x_2, \dots, x_n , e após este, aplica uma função na saída $g(\cdot)$ para ativá-lo.

A Função de ativação é a essência da rede, pois, com ela, o modelo consegue gerar não linearidades nos dados, e reproduzir funções não lineares na saída. Existem diversas funções ativas para processar a saída do neurônio, algumas são:

- Linear

$$g(x) = x, \quad (2.30)$$

- ReLU

$$g(x) = \begin{cases} x, & \text{if } x > 0 \\ 0 & \text{otherwise,} \end{cases} \quad (2.31)$$

- Sigmoid

$$\sigma(x) = \frac{1}{1 + \exp(-x)}, \quad (2.32)$$

- Tangente Hiperbólica

$$\tanh(x) = \frac{\exp x - \exp(-x)}{\exp x + \exp(-x)}. \quad (2.33)$$

- SELU (Scaled Exponential Linear Unit)

$$g(x) = \begin{cases} \lambda x, & \text{if } x > 0 \\ \lambda \alpha (e^x - 1) & \text{otherwise,} \end{cases} \quad (2.34)$$

A função de ativação SELU (Scaled Exponential Linear Unit) é relativamente nova comparada com as demais, proposta em 2017 [34], a ideia é promover propriedades de auto normalização, além disto, ela resolve alguns problemas relacionados a função de ativação ReLU, tais como, *vanishing gradient*.

Existe diversos métodos para se treinar o perceptron, dado que, no treinamento, nosso objetivo é encontrar um conjunto de pesos que minimizem a função custo. Pelo fato de haver somente uma camada de neurônios, o treinamento deste se torna simples, e para realizar este, é aplicado o algoritmo gradiente descendente visto na seção 2.2.1.1.

2.3.2 Perceptron Multicamadas

Perceptron Multicamadas (PMC) consiste de diversos neurônios empilhados em camadas, onde estes são interligados com os neurônios da próxima camada, havendo uma propagação da informação da primeira camada de entrada até a camada de saída. Podemos ilustrar esta arquitetura das redes neurais nas formas expostas pelas figuras 26 e 27.

Como estamos tratando de um problema de regressão unidimensional, podemos assumir que a arquitetura de interesse pode ser representada pela figura 27. Este tipo de arquitetura é a mais simples quando tratamos do PMC, além disto, segundo [6], com apenas uma quantidade certa de neurônios, podemos representar qualquer distribuição de dados.

Cada neurônio irá atuar similarmente como o perceptron, segundo a equação 2.28, e se propagar em seguida para o próximo neurônio, este processo é chamado de alimentação avante (*feedforward*). O grande desafio do PMC consiste em treinar este tipo de arquitetura usando o algoritmo do gradiente descendente. Neste problema, a função de erro não é explicitamente uma função dos pesos das camadas intermediárias, portanto, a sua derivação se torna um processo complexo. Para realizar o ajuste dos pesos das camadas intermediárias, devemos propagar a derivada da função erro, ou seja aplicar a regra da cadeia, até chegar a um ponto no qual faça sentido derivar a função sobre o peso de interesse. Além disto, esta atualização é feita de forma reversa, logo, a atualização começa

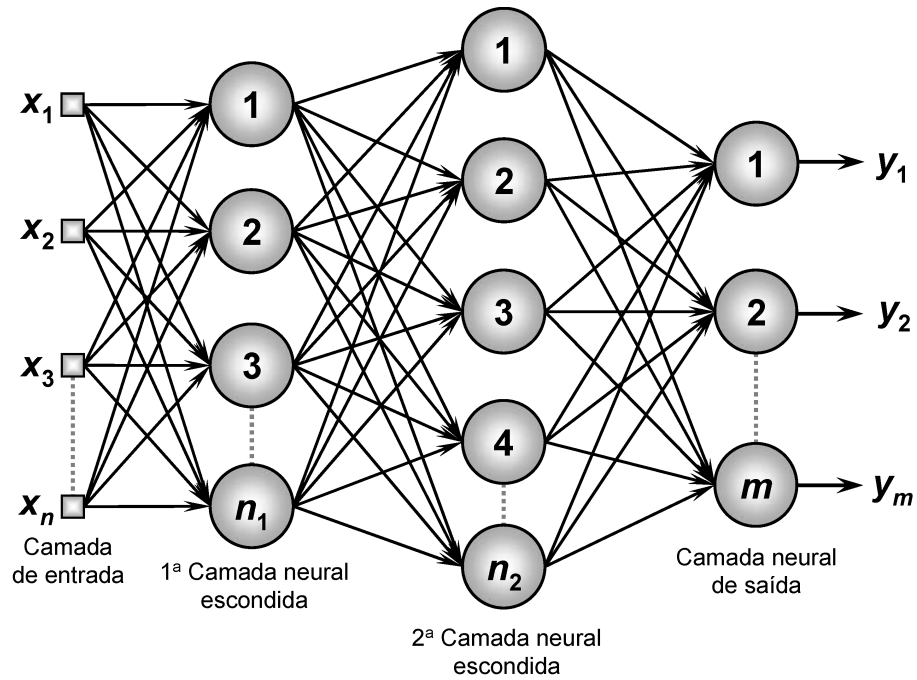


Figura 26 – Ilustração do Perceptron Multicamadas.(Imagem Retirada de [33])

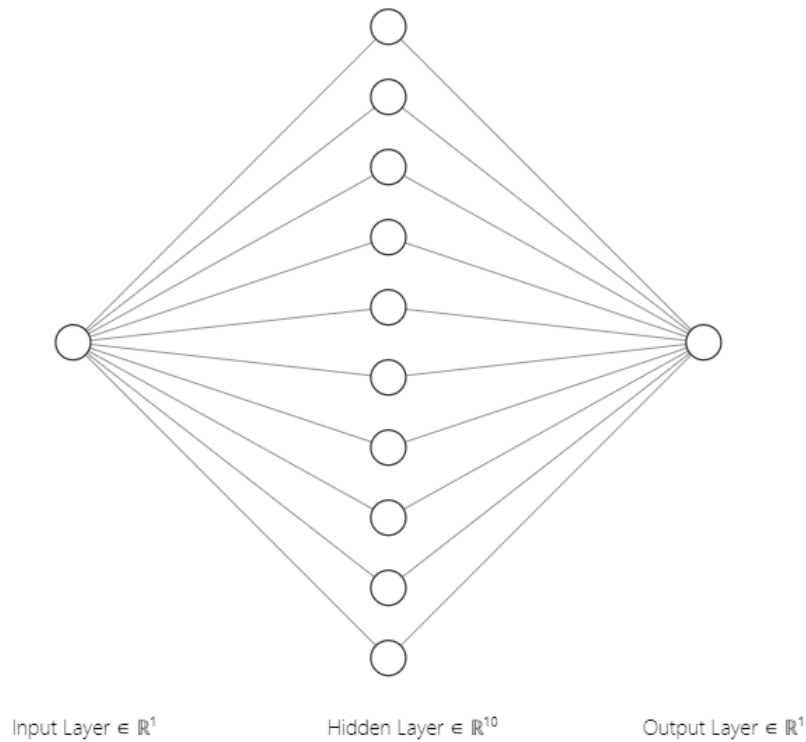


Figura 27 – Ilustração de um exemplo de arquitetura do Perceptron Multicamadas.

com os pesos das camadas finais, em direção as camadas iniciais, chamamos este processo de retropropagação (*backpropagation*).

De forma geral, a matemática empregada para as redes neurais é a álgebra linear. Ao invés de realizarmos a atualização de um único neurônio por vez, atualizamos todos de

uma determinada camada, que é representado por uma matriz de tamanho $n \times m$, onde estes representam o número de neurônios entre cada camada.

Supondo uma rede neural com a arquitetura da figura 27, as funções de ativação da rede são funções sigmoid, o número de neurônios são $n_1 = 1$, $n_2 = 10$ e a saída $n_3 = 1$, e por fim, que a função erro é a MSE.

Realizando a Alimentação Avante, ficaremos com:

$$\begin{aligned} U_1 &= X \cdot W_1^T, \\ \hat{Y}_1 &= \sigma(U_1), \end{aligned} \quad (2.35)$$

$$\begin{aligned} U_2 &= \hat{Y}_1 \cdot W_2^T, \\ \hat{Y}_2 &= \sigma(U_2), \end{aligned} \quad (2.36)$$

sendo W_1 a matriz de pesos da camada de entrada para a 1ª camada escondida e W_2 a matriz de pesos da 1ª camada escondida para a saída. Iremos calcular o erro associado a este conjunto de pesos, para em seguida atualizar a última camada de pesos que liga os neurônios de saída aos da camada escondida, W_2 , com isto, teremos o seguinte gradiente:

$$\frac{\partial E}{\partial W_2} = \frac{\partial}{\partial W_2} \frac{1}{N} \sum_i^N (Y_i - \hat{Y}_{2i})^2. \quad (2.37)$$

Calculando cada derivada da regra da cadeia, teremos:

- primeiro temos a derivada da função erro em função da saída \hat{Y}_2 .

$$\frac{\partial E}{\partial \hat{Y}_2} = -2 \times (Y - \hat{Y}_2), \quad (2.38)$$

- após isto, temos a derivada da saída \hat{Y}_2 em função da soma ponderada Z_2 , sendo este a derivada da função de ativação sigmoid.

$$\frac{\partial \hat{Y}_2}{\partial U_2} = \hat{Y}_2 \times (1 - \hat{Y}_2), \quad (2.39)$$

- por fim, como a soma ponderada é função do peso de interesse, podemos realizar a derivada do mesmo em relação ao peso.

$$\frac{\partial U_2}{\partial W_2} = \hat{Y}_1, \quad (2.40)$$

O gradiente da função erro será uma combinação de operadores Hadamard e produtos internos, neste trabalho representaremos o operador Hadamard com o símbolo \odot . Portanto, o gradiente será,

$$\frac{\partial E}{\partial W_2} = \frac{1}{N} \frac{\partial E}{\partial \hat{Y}_2} \odot \frac{\partial \hat{Y}_2}{\partial U_2} \cdot \frac{\partial U_2}{\partial W_2}, \quad (2.41)$$

$$\frac{\partial E}{\partial W_2} = \frac{1}{N} (-2 \times (Y - \hat{Y}_2)) \odot (\hat{Y}_2 \times (1 - \hat{Y}_2)) \cdot (\hat{Y}_1), \quad (2.42)$$

$$W_2 i = W_2 - \alpha \nabla E(\hat{Y}_2), \quad (2.43)$$

isto é para a última camada de neurônios, W_2 , para a primeira camada, W_1 , teremos o seguinte gradiente,

$$\frac{\partial E}{\partial W_1} = \frac{1}{N} \frac{\partial E}{\partial \hat{Y}_2} \odot \frac{\partial \hat{Y}_2}{\partial U_2} \cdot \frac{\partial U_2}{\partial Y_1} \odot \frac{\partial \hat{Y}_1}{\partial U_1} \cdot \frac{\partial U_1}{\partial W_1}. \quad (2.44)$$

Assim, o processo de treinamento de uma rede PMC consiste em realizar a regra da cadeia na função erro de modo que seja possível realizar a sua derivada. Após encontrar a derivada, basta usar a equação 2.21 para atualizar o peso. Percebemos ainda, que a complexidade do problema aumenta cada vez mais ao adicionamos mais neurônios na camada intermediária e mais camadas ocultas. De forma geral, existem métodos mais robustos para calcular esta derivada numericamente, tais como a diferenciação automática, onde neste, é usada a ideia de grafos para facilitar o cálculo das derivadas [35].

2.3.3 Redes Neurais Recorrentes

Uma das principais características das perceptron multicamadas visto anteriormente é que este não possui memória, ou seja, sua estrutura é projetada de forma a não salvar um determinado estado anterior [36]. Portanto, sua estrutura é principalmente usada em dados multidimensionais e independentes, de modo que os atributos sejam independentes uns dos outros. Contudo, diversos problemas possuem dependência sequencial, tais como, séries temporais, textos e dados biológicos [37]. Inteligência biológica processa a informação de forma incremental enquanto mantém um modelo interno no que está sendo processado, sendo este construindo de informações passadas e atualizadas de acordo com que novas informações vão chegando [36].

A rede neural recorrente (RNN) adota um princípio similar, embora de uma forma muito mais simplificada: ela processa a sequência interagindo sob os elementos da mesma e retendo um estado que contém informação relativo ao que já foi visto até o momento [36].

As redes neurais recorrentes possuem diversas arquiteturas, a forma com a qual é formada irá depender principalmente da aplicação desejada [38], a figura 29 expõe diversas arquiteturas de redes recorrentes.

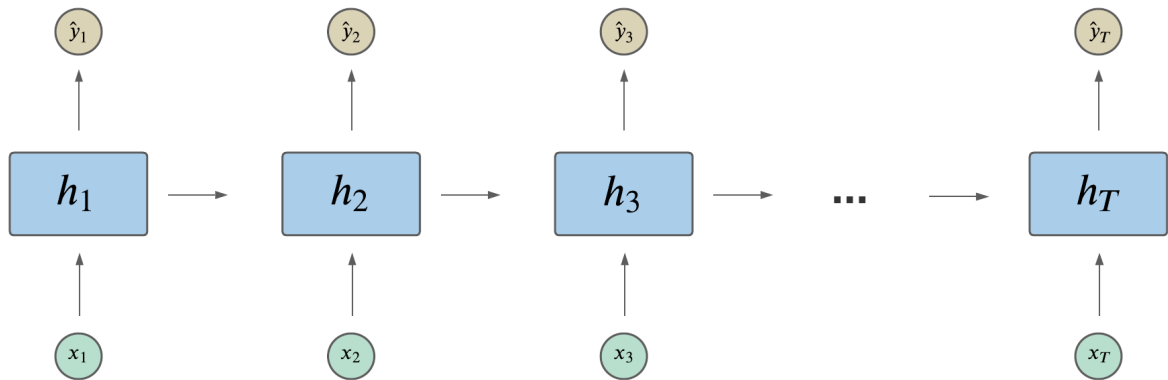


Figura 28 – Exemplo de rede recorrente formada com simples células recorrentes.

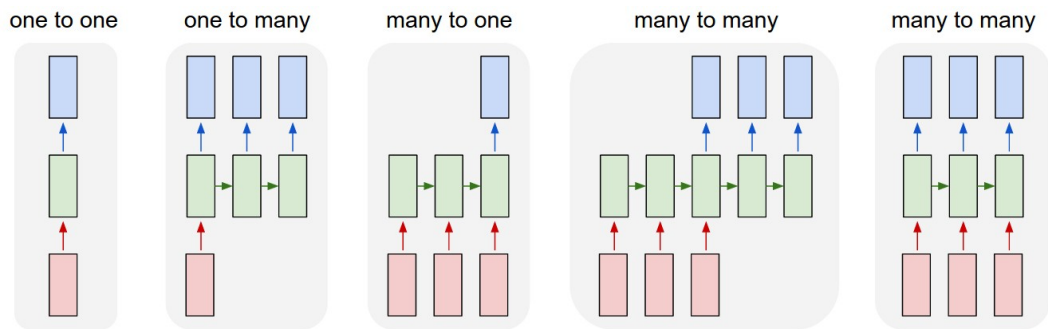


Figura 29 – Ilustração de diversas arquiteturas de redes recorrentes (imagem retirada de [38]).

Olhando com um enfoque maior em uma das células recorrentes, figura 28, podemos notar que ela possui três matrizes de pesos que serão similares para cada célula, a razão para tal é evitar que o modelo precise aprender uma grande quantidade de pesos.

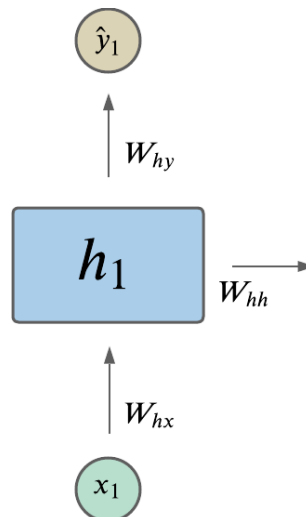


Figura 30 – Ilustração de simples célula recorrente.

De uma forma geral, as equações que descrevem a célula recorrente são:

$$h_t = \tanh(W_{xh}x_t + W_{hh}h_{t-1}), \tag{2.45}$$

$$\hat{y}_t = W_{hy}h_t, \quad (2.46)$$

sendo W_{hx}, W_{hh}, W_{hy} os pesos da rede e h_t o estado da célula (*hidden state*). A ideia da rede recorrente, é passar adiante a informação do estado atual para o próximo. Como esta estrutura obedece uma sequência temporal, os dados de entrada possuem uma estrutura distinta de uma PMC, em geral, os dados são tensores modelados de forma que sua estrutura seja composto da seguinte forma, (número de instâncias, passos no tempo, dimensões)[36].

Para realizar o treino da rede é usado a retropropagação, contudo, como temos contribuições de diferentes janelas de tempo, a derivada da função de custo se torna,

$$\frac{\partial E}{\partial W_{xh}} = \sum_{t=1}^T \frac{\partial E}{\partial W_{xh}^{(t)}}, \quad (2.47)$$

$$\frac{\partial E}{\partial W_{hh}} = \sum_{t=1}^T \frac{\partial E}{\partial W_{hh}^{(t)}}, \quad (2.48)$$

$$\frac{\partial E}{\partial W_{yh}} = \sum_{t=1}^T \frac{\partial E}{\partial W_{yh}^{(t)}}, \quad (2.49)$$

sendo que estamos somando o erro associado a cada janela de tempo, chamamos tal abordagem de retropropagação ao longo do tempo (*backpropagation through time*).

Na prática as RNNs apresentam diversos problemas de instabilidade durante o treinamento. Um problema comum está na atualização dos pesos durante o treino, como há diversas contribuições no erro, está pode divergir facilmente, chamamos tal efeito de *exploding gradient*, e há casos no qual as derivadas possuem pequenos valores, de modo a zerar o gradiente, resultando com que os pesos parem de ser atualizados, isto é dito como *vanishing gradient*. Diversas soluções já foram propostas para resolver os problemas acima, contudo, há uma célula recorrente específica que resolve parcialmente estes problemas, chamada *Long Short Term Memory*.

2.3.4 Long Short Term Memory

Uma forma de visualizar a instabilidade comentada na seção anterior, está no fato de que o parâmetro de atualização da redes neurais, de uma forma geral, é adquirido usando apenas multiplicações, isto faz com que redes com poucas camadas ocultas ou de pequenas sequências sejam treinadas com maior estabilidade do que redes profundas ou com longas sequências [39]. Portanto, para resolver tal problema, uma solução seria mudar o estado escondido da célula recorrente de modo a que ela obtenha uma memória de longo prazo.

A figura 31 [40], expõe o interior de uma célula LSTM. Nesta, há uma quantidade maior de informação entrando e saindo. As equações que regem a célula LSTM são:

$$\begin{aligned}
 f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f), \\
 i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i), \\
 o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o), \\
 c_t &= f_t \circ c_{t-1} + i_t \circ \tanh(W_c x_t + U_c h_{t-1} + b_c), \\
 h_t &= o_t \circ \tanh(c_t).
 \end{aligned}
 \tag{2.50}$$

Comentando sobre cada equação da 2.50,

- f_t é dito *forget gate*, este controla o que a memória guarda e o que ela apaga;
- i_t é dito *input gate*, este controla quais partes do conteúdo será repassado para as novas células;
- o_t é dito o *output gate*, este controla quais partes contribuirão para a saída;
- c_t é dito o *cell state*, este apaga uma quantidade de conteúdo do estado anterior e escreve novas informações.

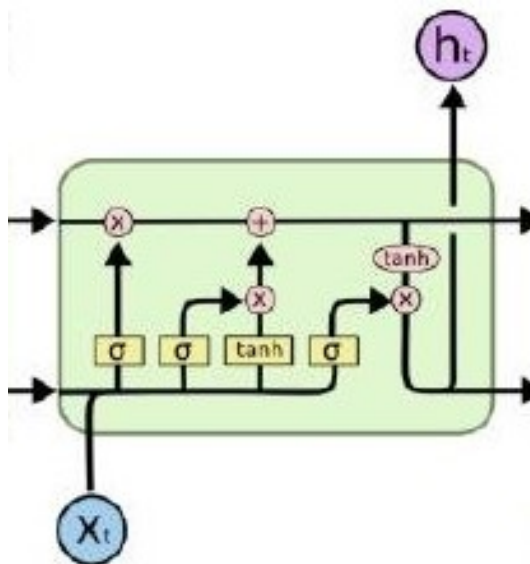


Figura 31 – Ilustração de célula LSTM.

Em termos gerais, as células LSTMs apresentam melhor desempenho do que uma célula RNN [41]. Além de ter um treino mais estável, a mesma consegue absorver dependências temporais mais longas, por conta disto, esta célula é a principal componente para aplicações envolvendo sequencia de dados.

3 Metodologia

Nesta secção veremos de quais formas foram empregados métodos de aprendizado de máquina para realizar a previsão da rugosidade global, ademais, será exposto o tratamento dos dados e as considerações feitas para realizar a extrapolação. Iremos dividir a metodologia em três partes, a primeira será focada em comentar sobre as simulações computacionais dos processos de deposição, a segunda apresentará a metodologia para o caso unidimensional e a terceira bidimensional.

3.1 Simulações dos modelos de deposição

Por se tratar de um problema de regressão, o modelo de aprendizado proposto neste trabalho, é um supervisionado. A partir disto, o primeiro passo consistiu em coletar dados de rugosidade que seriam postos para treino, validação e teste do algoritmo, ou seja, foram realizadas simulações computacionais para a obtenção de curvas de crescimento da rugosidade para os modelos de deposição de interesse, sendo este último, gerado a partir dos algoritmos nos anexos 6.1 e 6.2.

O padrão com o qual os dados simulados se comportam já foi exposto na figura 13 para o caso unidimensional e figura 15 para o bidimensional. Foram simulados padrões de rugosidade para os modelo de deposição aleatório, aleatório com relaxação de superfície e o Balístico, sendo o último de maior interesse.

Para o modelo Balístico unidimensional foram efetuadas simulações para comprimentos (L) entre $L = 2^7$ e $L = 2^{16}$, com mil realizações independentes (RI) para todos os comprimentos, com a exceção dos dois últimos, e entre unidades de tempo que vão de $T = 1 \times 10^3$ à $T = 3 \times 10^6$, posto que este último varia de acordo com o tamanho do substrato, para esta dimensão a tabela 2 expõe as principais características de cada comprimento lateral. para o caso bidimensional, a dificuldade aumenta exponencialmente, por conta disto, foram realizadas simulações computacionais para comprimentos entre $L = 2^6$ e $L = 2^{12}$, o número de realizações independentes diminuiu dramaticamente nas últimas três potências, a tabela 3 expõe as principais características de cada comprimento lateral para esta dimensão. Para realizar a simulação computacional de 2^{12} para 150000 unidades de tempo, foi empregado o cluster PERAU para realizar esta simulação e foram necessários em torno de 40 dias para executa-la.

Algo que vale ser mencionado, é que estas simulações são agudamente sensíveis, no sentindo que caso não forem definidas corretamente a regra de deposição, condições de contorno e gerador de números pseudo aleatórios, as curvas de rugosidade não irão convergir corretamente. Neste trabalho, foi utilizado o algoritmo *Mersenne Twister*, sendo este um

Tabela 2 – Tabela com as Características das simulações do modelo de deposição balística unidimensional.

L	RI	w_{sat}	unidades de tempo	tempo de simulação
2^7	10^3	5,8557	10^4	< 1 horas
2^8	10^3	7,8481	10^4	< 1 horas
2^9	10^3	10,7759	10^4	< 1 hora
2^{10}	10^3	14,9587	10^4	< 1 hora
2^{11}	10^3	21,1704	2×10^5	~ 1 horas
2^{12}	10^3	29,4149	2×10^5	~ 2 horas
2^{13}	10^3	41,2503	3×10^5	~ 5 horas
2^{14}	10^3	57,9659	3×10^5	~ 10 horas
2^{15}	200	83,7410	10^6	~ 24 horas
2^{16}	50	112,7918	5×10^6	~ 24 horas

Tabela 3 – Tabela com as Características das simulações do modelo de deposição balística bidimensional.

L	RI	w_{sat}	unidades de tempo	tempo de simulação
2^5	500	4,1581	10^3	< 1 horas
2^6	500	4,6428	10^3	< 1 horas
2^7	500	5,3233	10^3	< 1 horas
2^8	500	6,3038	10^3	< 1 horas
2^9	500	7,7348	4×10^3	< 1 hora
2^{10}	100	9,6321	2×10^4	~ 24 hora
2^{11}	50	12,1333	2×10^5	~ 48 horas
2^{12}	10	15,3068	8×10^5	~ 96 horas

dos melhores geradores de números pseudo aleatórios atualmente. Estamos mencionando este, pois, inicialmente, houve diversos conflitos nos gráficos de rugosidade do modelo de deposição balística quando comparados com os simulados por [16], isto foi corrigido posteriormente quando mudamos o gerador de números pseudo aleatórios.

Podemos também mencionar um problema subjacente da equação 2.6, por esta tomar o quadrado da altura, e o fato de que o modelo de deposição balística cresce mais rapidamente que o modelo aleatório, quando alcançado unidades de tempo em torno de $\sim 10^5$, a altura começou a atingir grandes escalas de valores, de modo que seu quadrado gerasse erro de ponto flutuante, fazendo com que as curvas começassem a crescer de forma equivocada. Para corrigir tal erro, tivemos que subtrair a altura mínima do substrato a cada unidade de tempo, ou tomar a equação 2.1. Por conta deste, tivemos que descartar a curva gerada para o comprimento de $L = 2^{12}$ para o caso bidimensional e refazer a simulação com uma quantidade menor de realizações independentes.

Para as simulações dos modelos de deposição, foi utilizada a linguagem Python. Contudo, esta linguagem é baseada em interpretador, portanto, é mais lento do que linguagens como C/C++ ou Java que é compilada, no qual o código-fonte completo é convertido em

linguagem de máquina. Assim, um dos maiores desafios do trabalho, foi a simulação dos modelos de deposição balística com $d = 1$ e $d = 2$ para tamanhos maiores que 2^{10} . Para contornar este problema do Python, foi-se utilizado a biblioteca Numba, sendo esta, um compilador open source JIT que traduz um sub conjunto do código em Python e Numpy em código de máquina de forma mais eficiente [42].

3.2 Modelo Balístico Unidimensional

Para esta dimensão foram propostas duas metodologia distintas que dividiremos em duas subsecções.

3.2.1 Metodologia 1

Para esta metodologia empregamos três modelos de aprendizado de máquina, As Florestas Aleatórias, as Máquinas de Vetores de Suporte e as Redes Neurais. Nesta secção serão expostos os hiperparâmetros de cada modelo, junto a metodologia para treinamento e previsões dos padrões de rugosidade.

Todos os modelos tiveram os mesmos dados de treino, sendo estes, os dados normalizados segundo o *Scaling Family-Vicsek* para o comprimento lateral de $L = 2^{11}$. O principal motivo de usar este comprimento, é que a partir deste, é possível usar os demais dados simulados $L = 2^{12}, 2^{13}, 2^{14}, 2^{15}, 2^{16}$ para validar e testar o treinamento dos modelos.

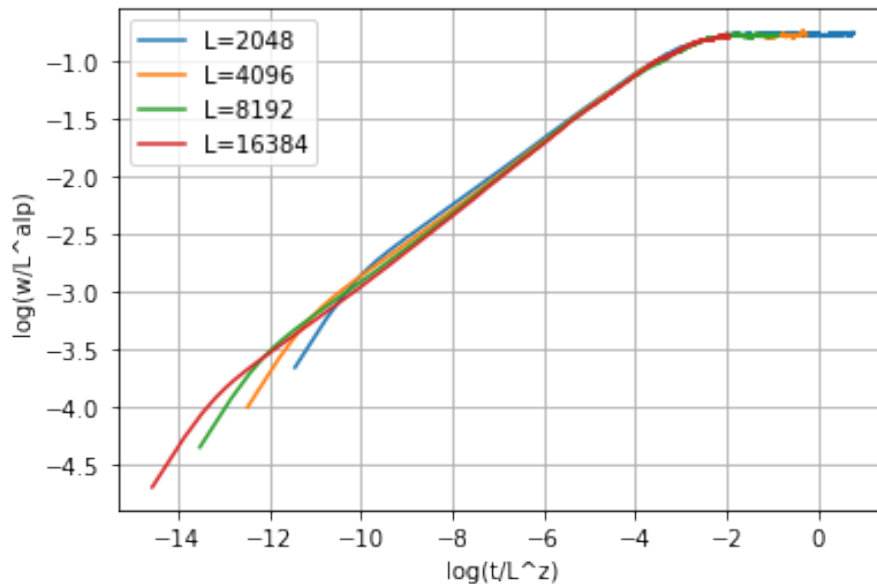


Figura 32 – Gráfico de rugosidade para deposição balística normalizado.

Uma parte importante antes de começar a realizar o aprendizado foi inferir quais valores de α e z seriam usados para normalizar os dados simulados. Ao invés de usarmos valores teóricos e computacionais que já estimam o possível valor de α e z [16], decidimos aplicar um algoritmo de busca para encontrar o melhor valor de α que minimizasse a distância entre as curvas, $L = 2^{13}$ e $L = 2^{14}$. Para este fim, geramos um algoritmo de busca por

força bruta para buscar o melhor valor de α dentro de seu domínio. Encontramos $\alpha = 0,5$, portanto, adotaremos $\alpha = 0,5$ e segundo a equação 2.10, podemos usar $z = 1,5$ para normalizar os dados.

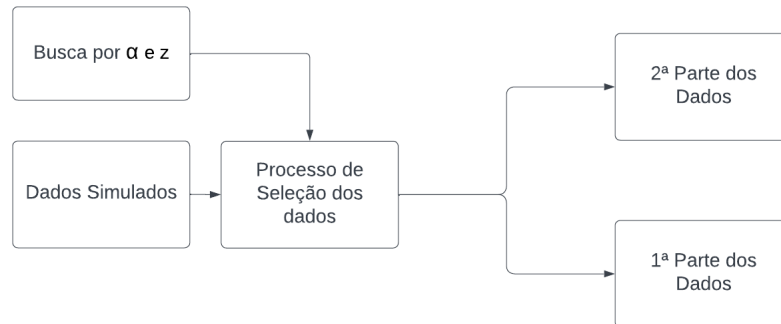


Figura 33 – Diagrama de fluxo para o tratamento e seleção dos dados.

Observando as curvas normalizadas, figura 32, é possível notar que estas ainda não colapsam no início da curva, então, podemos dividir os dados de treino em dois subconjuntos. O primeiro conjunto de dados consiste nos sete primeiros pontos simulados, e a segunda parte, os demais pontos. Devemos guardar esta primeira parte, pois este será empregada mais a frente. A figura 33 expõe o diagrama desta primeira parte da metodologia.

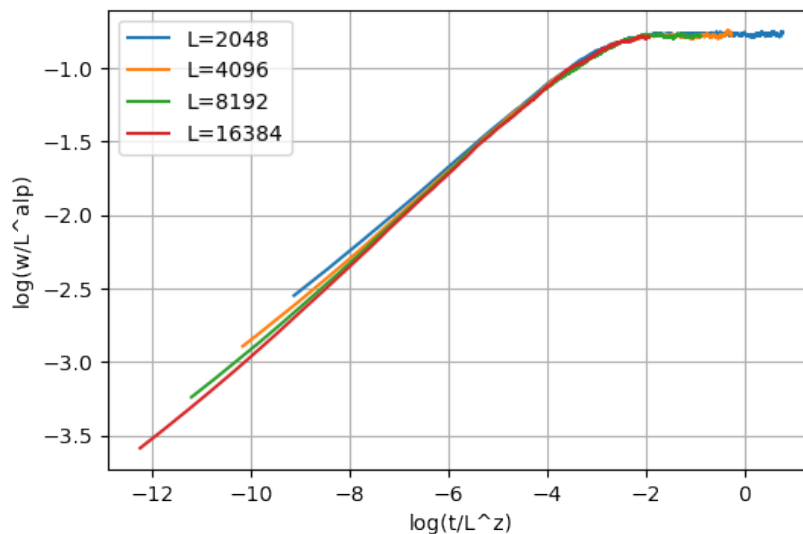


Figura 34 – 2ª parte dos dados de rugosidade normalizados para deposição balística.

Porém, ainda há um problema na segunda parte dos dados. Podemos notar que estes ainda não convergem totalmente, isto, pelo fato de haver efeitos de pequenos comprimentos, portanto, é removido os dados que não convergem inicialmente. Tal processo é descrito pelo diagrama 35 e exposto na figura 36.

A ideia subjacente das transformações é realizar um aprendizado em cima dos dados normalizados de tempo para o tamanho de $L = 2^{11}$, a fim de aprender o padrão dos dados de rugosidade normalizado, de forma que quando inserirmos dados de tempo normalizados

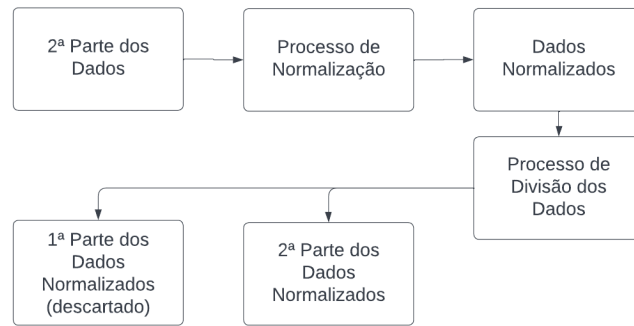


Figura 35 – Diagrama do fluxo para a separação dos dados que irão para o modelo.

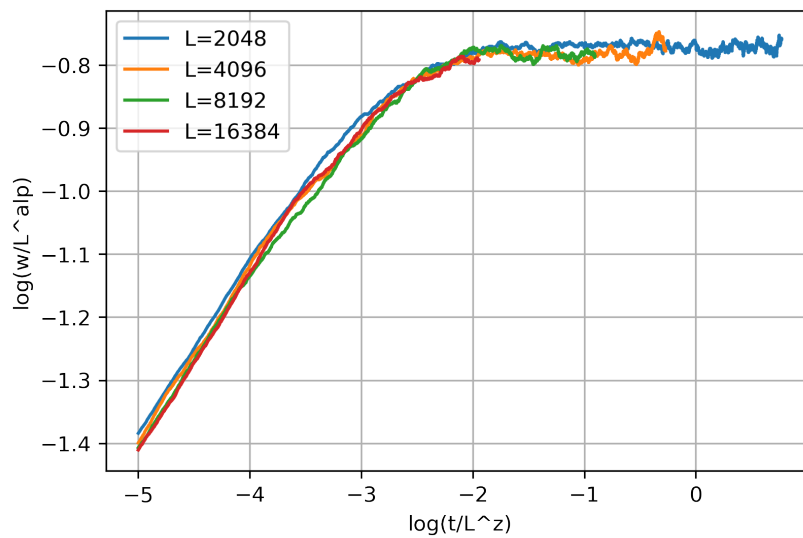


Figura 36 – Dados que serão usados para treinar e validar os modelos de aprendizado de máquina.

para maiores comprimentos de substratos, obteremos dados de rugosidade normalizados para este mesmo comprimento, onde finalmente será realizado a transformação reversa para obter a curva desejada de rugosidade.

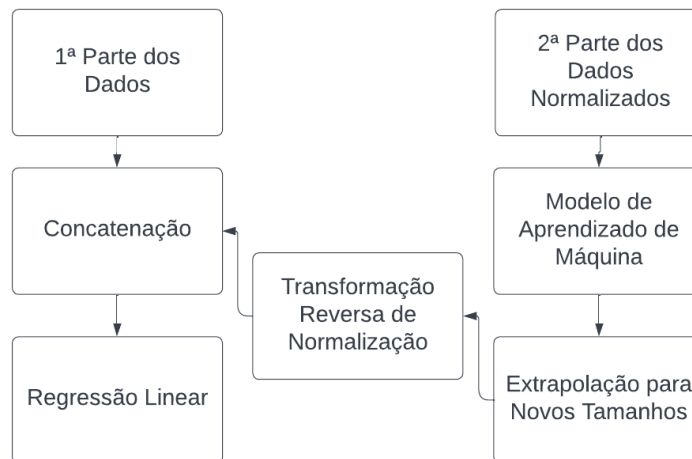


Figura 37 – Diagrama do fluxo para a interpolação da última parte dos dados.

Até momento, realizamos um aprendizado em cima de uma parcela da curva de crescimento, e não a curva como um todo. Analisando o problema, sabemos que após $t = 8$ a curva crescerá segundo a equação 2.6 até o tempo de *crossover*. Contudo, com o aprendizado de máquina, somos capazes de apenas gerar uma parte desse crescimento exposto pela equação 2.6, passando pelo tempo de *crossover* e indo até a saturação da rugosidade. Portanto, falta apenas interpolar este crescimento intermediário que cresce "linearmente". Para isto, podemos realizar uma regressão linear entre os pontos finais da primeira parte dos dados separados no início da metodologia, e os pontos iniciais da parte aprendida pelo modelo de aprendizagem de máquina, tal processo é descrito no diagrama 37 e exposto na figura 38.

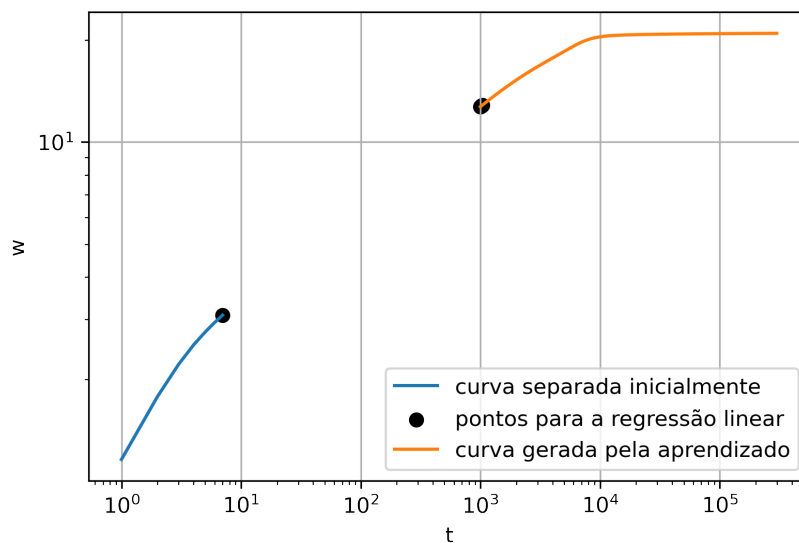


Figura 38 – Curvas geradas pela metodologia.

A figura 39 expõe todas as três partes da nova curva que descreve todo o comportamento da rugosidade e o diagrama 40 expõe toda a metodologia do processo.

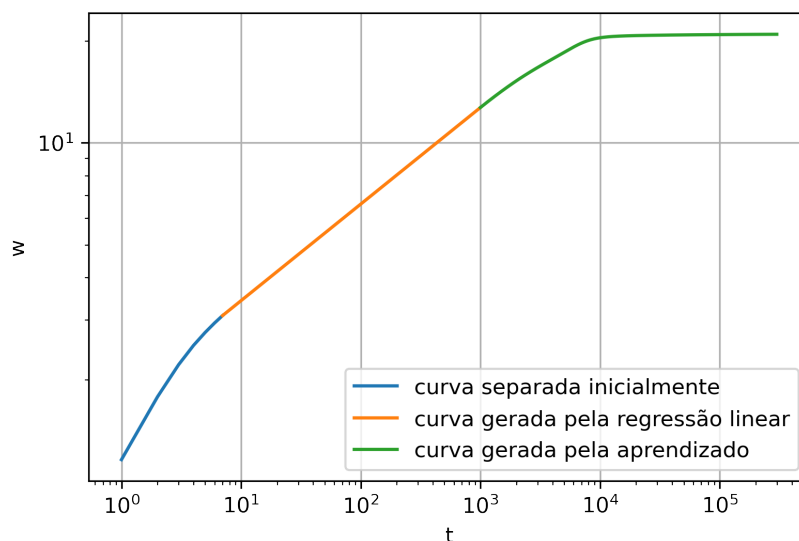


Figura 39 – Gráfico de rugosidade onde cada cor representa cada parte da metodologia.

Um dos objetivos do trabalho consiste em calcular o valor de α para comprimentos que são inviáveis de simular computacionalmente. Assim, após realizar a extrapolação a partir da metodologia 1, foi calculada a rugosidade de saturação tomando a média em cima da região de saturação, e com o uso da equação 3.1, podemos encontrar o α quando temos curvas de crescimento que vão para o limite hidrodinâmico.

$$\log(w_{sat}) = \alpha \log(L) + C. \quad (3.1)$$

Assim, a medida que serão obtidos mais pontos a partir do aprendizado, será possível realizar a regressão para maiores valores de rugosidade saturação, de modo a obter α para o limite hidrodinâmico, onde este deve se aproximar do valor teórico de 0,5.

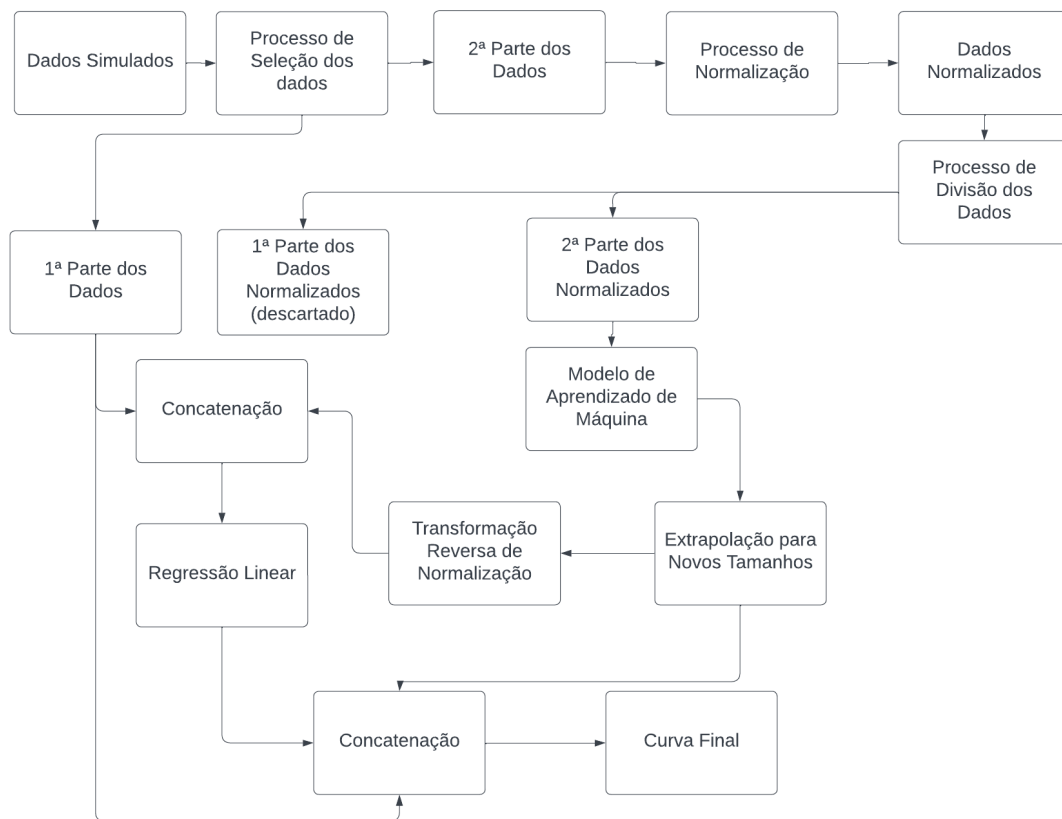


Figura 40 – Diagrama com toda a metodologia para regressão da curva de rugosidade.

Com os resultados encontrados podemos calcular os valores de α_L e verificar se as correções do expoente seguem o modelo proposto pelo autor [16].

A linguagem de programação utilizada para todas as simulações e cálculos foi o Python. Atualmente, Python é uma das principais linguagens de programação para se realizar computação científica. Sua simplicidade, suporte da comunidade, número de bibliotecas para cálculos numéricos e para inteligência artificial, fez com que este se tornasse uma das principais linguagens atualmente. As bibliotecas usadas para realizar o aprendizado de máquina foram: Scikit e Tensorflow. Com este é possível criar diversos modelos de

aprendizado de máquina com apenas algumas linhas de código. Para cada modelo treinado, foram utilizados os seguintes parâmetros:

Tabela 4 – Hiperparâmetros Máquinas de Vetores de Suporte

Kernel	Degree	γ	C	ϵ	Número de Iterações
rbf	3	scale	1	0.01	30000

Tabela 5 – Hiperparâmetros Florestas Aleatórias

Profundidade da Árvore	Número de Estimadores	Erro
10	50	Erro Quadrático

Tabela 6 – Hiperparâmetros Perceptron Multicamadas 1

Batch Size	Loss	Otimizador	1º camada oculta
3000	MSE	Adam	17

Tabela 7 – Hiperparâmetros Perceptron Multicamadas 2

Batch Size	Loss	Otimizador	1º camada oculta	2º camada oculta
3000	MSE	Adam	10	3

Scikit é uma biblioteca voltada para ciência de dados, nela, já há métodos prontos para se realizar tarefas de regressão e classificação, tais como, máquinas de Vetores de Suporte, Arvore de Decisões e Regressão Linear. Enquanto que a Tensorflow é uma biblioteca voltada para o desenvolvimento e treinamento de redes neurais. Todos os hiperparâmetros deste trabalho foram otimizados eurísticamente. Para esta metodologia, foi empregado apenas a curva de rugosidade de $L = 2^{11}$ para treino, como consequência, o tempo computacional para treinar estas curvas foi consideravelmente baixo quando comparado com as próximas metodologias. As redes neurais foram aqueles com menor tempo de treino, aproximadamente, dez minutos, enquanto que as SVM e RF tiveram em torno de 20 minutos de treino. Vale ressaltar que todos estes cálculos foram realizados utilizando a plataforma Google Colab disponibilizado pelo Google.

3.2.2 Metodologia 2

Como redes neurais são considerados aproximadores universais, a ideia na metodologia 2 é utilizar apenas os dados de tempo e comprimento, de forma que a rede consiga aproximar uma função que descreva o crescimento da rugosidade $w(L, t)$. Para tal, foram empregados

os dados de $L = [2^{11}, 2^{12}, 2^{13}, 2^{14}]$ como dados de treino, 2^{15} validação e 2^{16} teste. Usamos estes dados pelo fato de que a partir do comprimento de $L = 2^{11}$, é possível observar que o crescimento da rugosidade começa a apresentar um padrão, e como é desejado realizar uma previsão para comprimentos superiores, foi empregado as curvas de maiores comprimentos para validar e testar a rede.

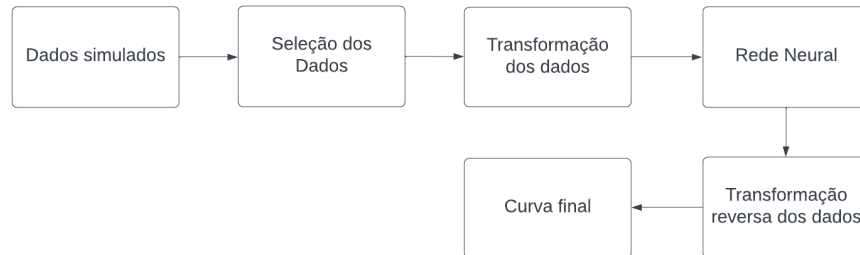


Figura 41 – Diagrama de fluxo para a metodologia 2.

Como a escala de tempo varia bruscamente foi utilizado a função logarítmica para deixar os dados em uma escala mais próxima, este processo foi realizado para todas as variáveis, tempo, comprimento e rugosidade. Dividimos todas as variáveis por uma constante, de modo que os dados estivessem contidos entre 0 e 1, isto é justificado pelo fato de que as redes neurais aprendem de forma mais estável com valores nesta região. Além disto, aumentamos o alcance dos dados adicionando ruído em torno da rugosidade de saturação para todos os comprimentos de treino. A figura 41 expõe o diagrama para esta metodologia.

Contudo, mesmo após este pré-processamento, o aprendizado do modelo não convergia para um modelo que conseguisse gerar as curvas de acordo com a teoria, repetidamente, as curvas geradas tinham uma forma que desviava das simuladas. Para resolver isto, aplicamos o logaritmo do inverso da rugosidade, onde este resulta em um sinal invertido no logaritmo. A figura 42 expõe a diferença entre as curvas.

As matrizes de entradas e saídas da rede podem ser observadas nas equações 3.2 e 3.3, nesta representação o índice superior indica a posição da variável no tempo, enquanto a inferior indica o comprimento da curva de rugosidade.

A rede neural utilizada foi uma Perceptron Multicamadas, com duas camadas intermediárias e uma saída, onde todas possuem como função de ativação SELU. Os demais hiperparâmetros estão exposto na tabela 8.

Tabela 8 – Hiperparâmetros da Rede Neural

Batch Size	Loss	Optimizer	Early Stop	Neurônios 1º camada	Neurônios 2º camada
64	MSE	Adam	200	80	80

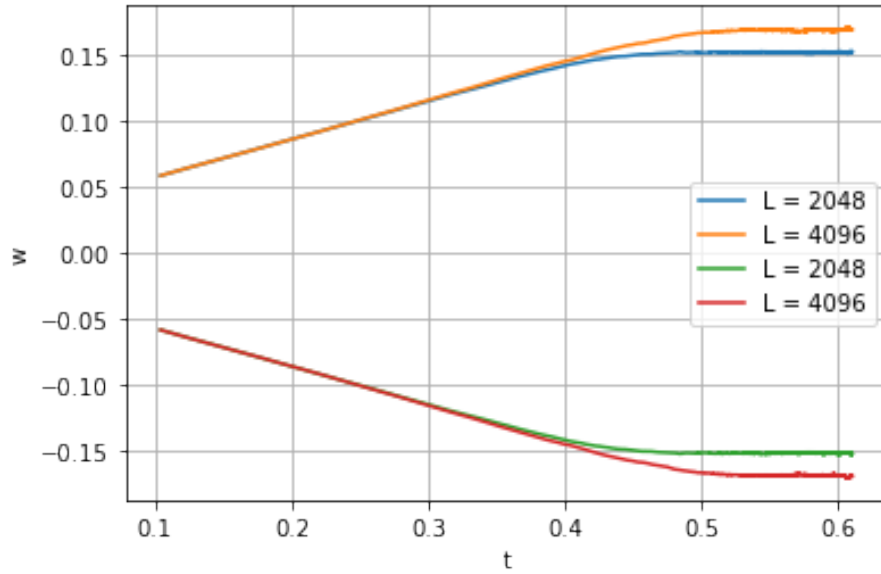


Figura 42 – Curvas aplicando o pré-processamento usando o logaritmo e o negativo do logaritmo.

$$X = \begin{bmatrix} \log(t_{2^{11}}^1)/20 & \log(2^{11})/20 \\ \log(t_{2^{11}}^2)/20 & \log(2^{11})/20 \\ \dots & \dots \\ \log(t_{2^{12}}^1)/20 & \log(2^{12})/20 \\ \log(t_{2^{12}}^2)/20 & \log(2^{12})/20 \\ \dots & \dots \\ \log(t_{2^{13}}^1)/20 & \log(2^{13})/20 \\ \log(t_{2^{13}}^2)/20 & \log(2^{13})/20 \\ \dots & \dots \\ \log(t_{2^{14}}^1)/20 & \log(2^{14})/20 \\ \log(t_{2^{14}}^2)/20 & \log(2^{14})/20 \\ \dots & \dots \\ \log(t_{2^{14}}^N)/20 & \log(2^{14})/20 \end{bmatrix}. \quad (3.2)$$

$$y = \begin{bmatrix} -\log(w_{2^{11}}^1)/20 \\ \dots \\ -\log(w_{2^{14}}^N)/20 \end{bmatrix}. \quad (3.3)$$

Esta metodologia tem como objetivo ser uma forma alternativa de calcular a rugosidade de saturação. Além de servir de referência para a primeira metodologia, ela também nos indica que há a possibilidade de um modelo de rede neural ser capaz aprender a distribuição da função $w(L, t)$ para qualquer modelo correlacionado. O tempo computacional para treinar este modelo de rede neural foi superior aos modelos empregados na metodologia 1, isto pelo fato da quantidade de dados ser superior e pela função final que se deseja gerar

ser mais complexa. Um treinamento usando uma parada antecipada (*earlystopping*) levou cerca de três horas para ser concluído usando a plataforma Google Colab.

3.3 Modelo Balístico Bidimensional

Pela complexidade computacional, o balístico bidimensional foi o modelo mais desafiador em termos de aprendizado, pois neste, a quantidade de dados era limitado até 2^{12} . A partir da figura 15, podemos perceber que o último tamanho é aquele com mais ruído, isto pelo fato de termos realizado apenas 10 realizações independentes. A justificativa por tal, está pelo problema citado no início do capítulo quando abordamos sobre o cálculo da altura na simulação computacional, conseqüentemente, foi necessário reiniciar a simulação computacional para tal tamanho. Contudo, para não ser necessário realizar diversas realizações, e ter que esperar pela curva, um método para mitigar o ruído pós simulação foi utilizar uma rede neural para gerar uma curva média em cima da curva com ruído, a nova curva em comparação da primeira pode ser observada na figura 43.

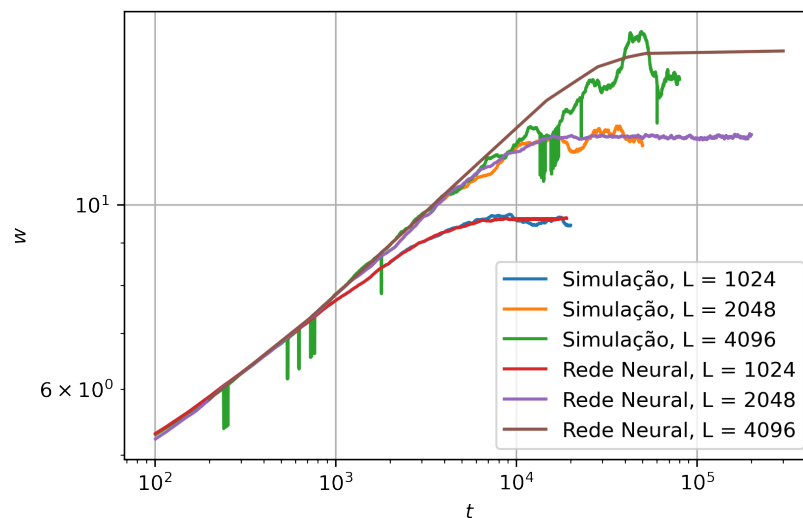


Figura 43 – Curvas simuladas computacionalmente em comparação com as curvas médias geradas via rede neural.

A ideia por trás dessa mitigação é usar as curvas do modelo unidimensional de tamanhos equivalentes, juntamente com o tempo, como entrada da rede neural, e usar como saída a curva ruidosa, de forma que uma rede bem regularizada conseguirá encaixar ambas as curvas gerando uma curva final menos ruidosa, o diagrama 44 expõe a forma com a qual os dados foram processados.

Para treinamento foi utilizado os comprimentos de $L = [2^9, 2^{10}, 2^{11}]$, e como validação usamos a curva de 2^{12} , como uma simulação computacional de 2^{13} demandariam um tempo muito longo com a estrutura computacional disponível para o uso do grupo de pesquisa, usamos apenas uma estimativa bruta de qual possível resultado que este poderia obter, posteriormente nesta seção serão apresentados mais detalhes a esse respeito.

Nesta dimensão não fomos capazes de usar a primeira metodologia do unidimensional,

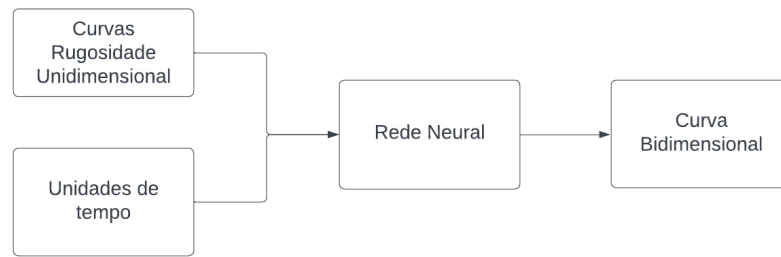


Figura 44 – Metodologia para o pré-processamento dos dados de rugosidade.

pelo fato destas curvas não convergirem para valores de α próximos ao teórico, além que os mesmo apresentam valores que variam de acordo com o comprimento. Então, decidimos adotar uma rede neural mais complexa para lidar com este problema. Para gerar as novas curvas, usamos uma rede LSTM com entradas tempo, comprimento, e rugosidade de comprimentos anteriores, o diagrama 45 expõe a metodologia para esta dimensão. Foram realizadas previsões com um, dois e três comprimentos anteriores como entrada da rede, contudo, a configuração com a informação dos três comprimentos anteriores foi a que apresentou os melhores resultados.

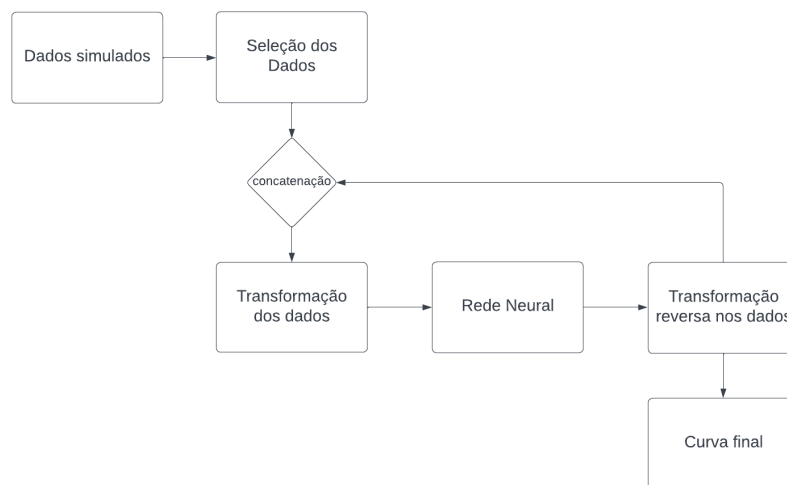


Figura 45 – Diagrama da metodologia para o caso bidimensional.

Foi Aplicado um pré-processamento similar ao da metodologia 2 vista anteriormente para o caso unidimensional, a única diferença neste é que todas as variáveis tiveram sinais alterados para negativo, a matriz de entradas e saídas podem ser observadas nos tensores 3.4 e 3.5.

$$X = \begin{bmatrix} [-\log(w_{L/8}^1)/20 & -\log(L/8)/20 & -\log(t^1)/20] \\ [-\log(w_{L/4}^1)/20 & -\log(L/4)/20 & -\log(t^1)/20] \\ [-\log(w_{L/2}^1)/20 & -\log(L/2)/20 & -\log(t^1)/20] \\ \dots & \dots & \dots \\ [-\log(w_{L/8}^N)/20 & -\log(L/8)/20 & -\log(t^N)/20] \\ [-\log(w_{L/4}^N)/20 & -\log(L/4)/20 & -\log(t^N)/20] \\ [-\log(w_{L/2}^N)/20 & -\log(L/2)/20 & -\log(t^N)/20] \end{bmatrix} \quad (3.4)$$

$$y = \begin{bmatrix} -\log(w_L^1)/20 \\ \dots \\ -\log(w_L^N)/20 \end{bmatrix} \quad (3.5)$$

A partir das equações 3.4 e 3.5 é possível notar que foi necessário realizar um alinhamento das rugosidades anteriores com a rugosidade que desejamos prever, sendo que estas precisam ter a mesma quantidade de pontos no tempo, para tal, foi adicionado ruído aleatório em torno da rugosidade de saturação de forma a aumentar o alcance dos dados.

Tabela 9 – Hiperparâmetros da rede neural para o caso bidimensional

Batch Size	Loss	Otimizador	LSTM 1 ^o camada	LSTM 2 ^o camada	PMC 3 ^o camada
64	MSE	Adam	30	30	80

Após treinamento, para gerar curvas que ainda não haviam sido simuladas, foi empregado as novas curvas geradas pela rede neural como entrada da rede, ou seja, para realizar a previsão de $L = 2^{14}$ foi necessário $w_{2^{11}}, w_{2^{12}}, w_{2^{13}}$, em que a curva de comprimento de $L = 2^{13}$ foi adquirido a partir da rede neural.

Foram realizados treinamentos iniciais que consistiram em empregar o dados de $L = [2^9, 2^{10}, 2^{11}]$ como treino e 2^{12} , como validação. Todas as curvas iniciais geradas pelas redes convergiam para valores condizentes com os autor [16], contudo, para comprimentos superiores à $L = 2^{16}$, as curvas começavam a divergir do esperado. Uma possível explicação para tal comportamento é pelo número de comprimentos empregados para realizar o treinamento. Então, como os modelos conseguiram gerar a curva de 2^{13} com um ponto de saturação próximo as estimativas do autor [16], realizamos um ajuste fino, *fine tuning*, da rede alterando os valores de *batch size*, taxa de aprendizado e usando os comprimentos de $L = [2^9, 2^{10}, 2^{11}, 2^{12}]$ como treino, e 2^{13} como validação. Além disto, realizamos um segundo treinamento da rede neural usando os comprimentos de $L = [2^9, 2^{10}, 2^{11}, 2^{12}]$ como treino, e 2^{13} como validação, sem empregar pesos pré-treinados. Como nesta metodologia a quantidade de dados é inferior em comparação da metodologia 2 do caso unidimensional, o tempo computacional para treinar o modelo de rede para o caso bidimensional foi em torno de duas horas.

Como não há na literatura simulações para comprimentos acima de 2^{10} , a principal forma de validar os dados foi aplicar as estimativas do trabalho [16] como uma forma de validar o crescimento do modelo da rede neural.

4 Resultados

Neste capítulo são apresentados os resultados obtidos a partir das metodologias vistas no capítulo anterior. Este é dividido em três seções, a primeira e segunda expõem os resultados obtidos para a deposição balística unidimensional e bidimensional, apresentando as previsões de curvas de rugosidade global e os expoente de interesse, para em seguida realizar uma discussão mais profunda sobre os resultados encontrados.

4.1 Balístico Unidimensional

Primeiro, iremos comparar como a primeira metodologia reproduz os resultados simulados, para em seguida realizar extrapolações para comprimentos maiores. A figura 46 expõe as curvas geradas pelo modelo da máquinas de vetores de suporte, em que estas crescem de acordo com as curvas geradas computacionalmente, contudo, na região de saturação elas começam a desviar levemente da média.

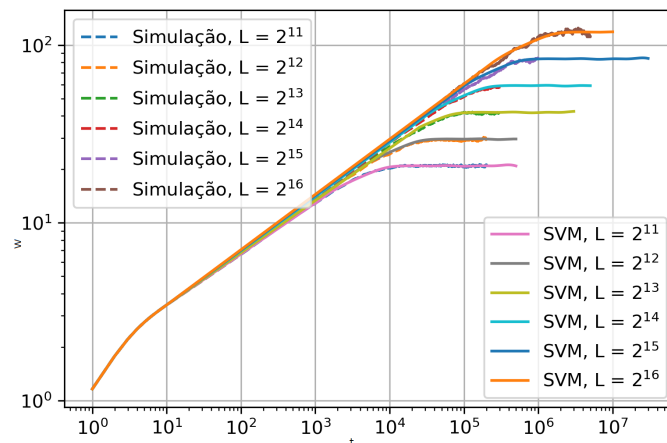


Figura 46 – Regressão para curvas conhecidas usando máquinas de vetores de suporte.

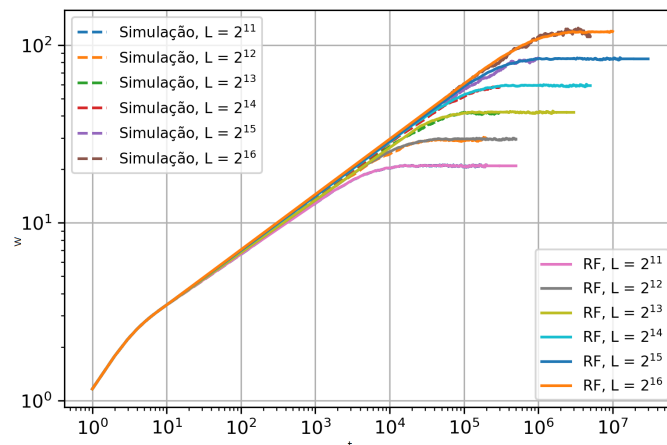


Figura 47 – Regressão para curvas conhecidas usando florestas aleatórias.

No modelo usando florestas aleatórias, figura 47, é possível notar que durante o treino o modelo aprendeu o ruído dos dados e os reproduziu novamente nas demais curvas.

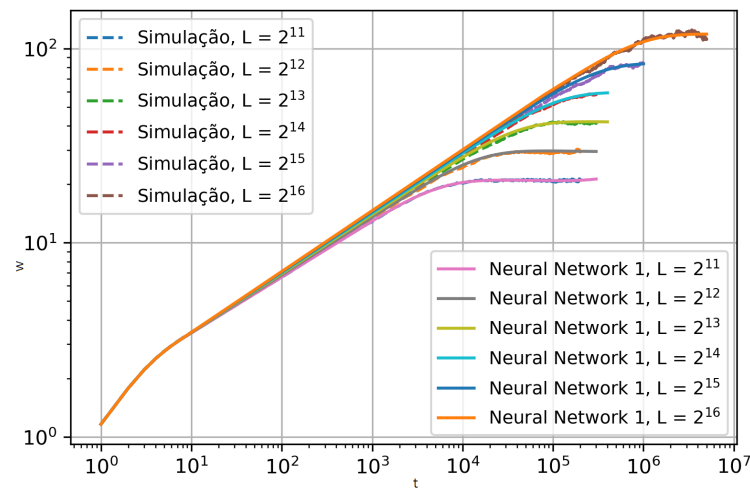


Figura 48 – Regressão para curvas conhecidas usando perceptron multicamadas-1 camada escondida.

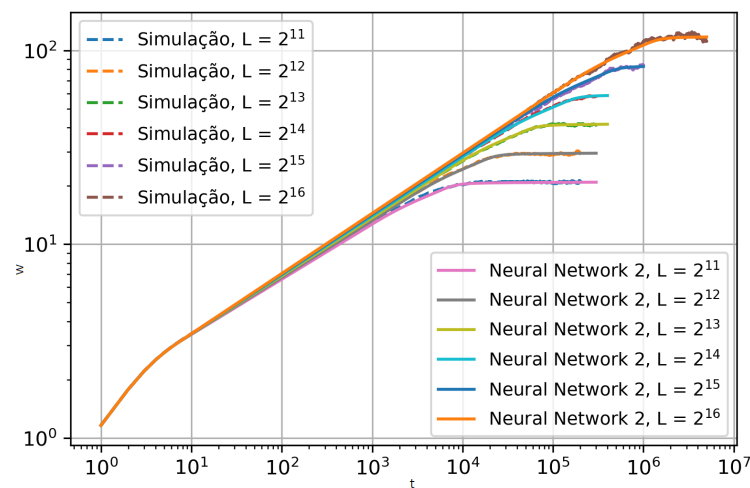


Figura 49 – Regressão para curvas conhecidas usando perceptron multicamadas-2 camadas escondida.

Por fim, as figuras 48 e 49 apresentam os resultados para as duas arquiteturas de rede neural, em que estas conseguem gerar curvas finais com menos desvios em torno da saturação.

Tabela 10 – Tabela das RMSEPE

$L/\%$	SVM	RF	RNN1	RNN2	média	desvio
2^{11}	0,6991	0,0941	0,6020	0,8803	0.5688	0.2917
2^{12}	1,5867	1,6203	1,5203	0,9755	1.4257	0.2624
2^{13}	1,5465	1,5652	1,5743	1,2383	1.4810	0.1405
2^{14}	1,7040	1,4832	1,3984	0,8795	1.3662	0.3023
2^{15}	2,9700	2,9295	2,8817	1,6718	2.6132	0.5444
2^{16}	2,6431	2,6964	2,7541	2,7763	2.7174	0.0519

A tabela 10 expõe a Raiz do Erro Quadrático Médio Percentual entre as curvas de rugosidade simulados computacionalmente e os valores gerados pelos modelos de aprendizado de máquina.

Podemos notar pela tabela 10 que cada um dos modelos consegue interpolar e extrapolar bons resultados comparado com as simulações realizadas. O Perceptron Multicamadas é aquele com melhor desempenho comparado com os demais. Como as duas últimas curvas foram geradas com uma quantidade menor de realizações independentes, é esperado que as mesmas possuam RMSEPE superior aos das curvas com mais realizações.

Como os resultados para os comprimentos conhecidos foram satisfatórios, realizaremos extrapolações para comprimentos superiores aos simulados, as figuras 50 - 53 expõem extrapolações para comprimentos entre $L = [2^{20}, 2^{25}, 2^{30}, 2^{35}]$.

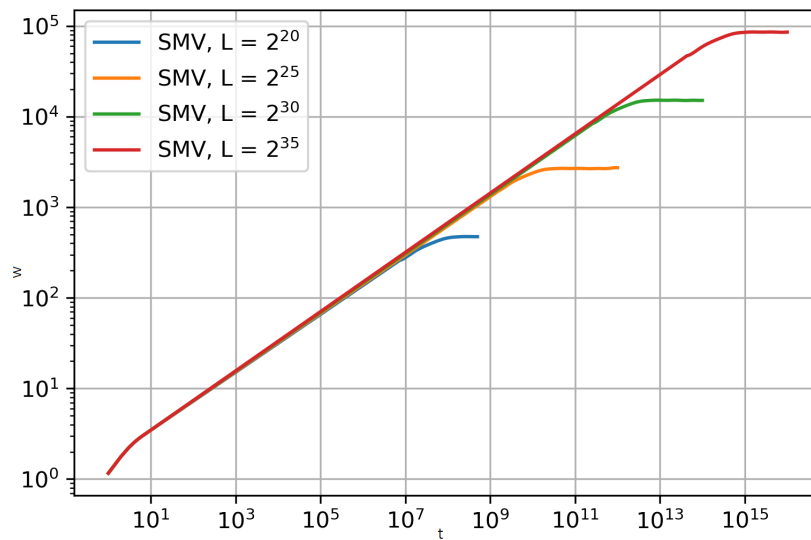


Figura 50 – Extrapolações Usando máquinas de vetores de suporte.

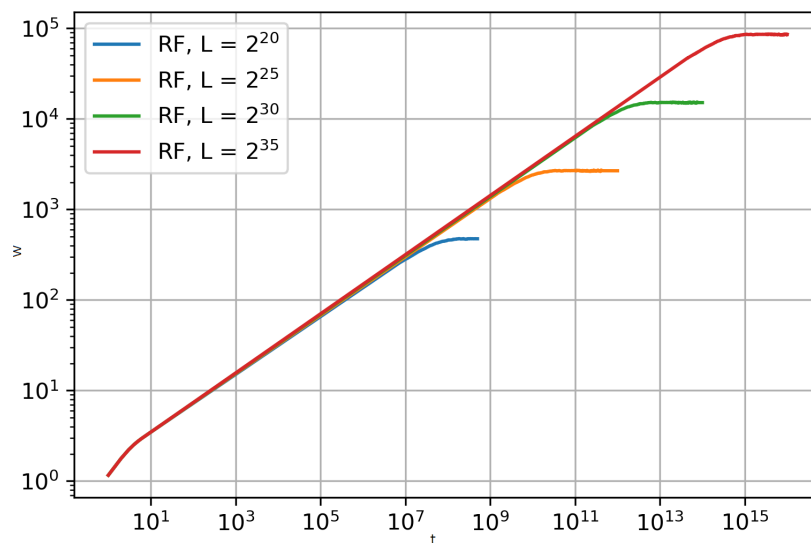


Figura 51 – Extrapolações Usando florestas aleatórias.

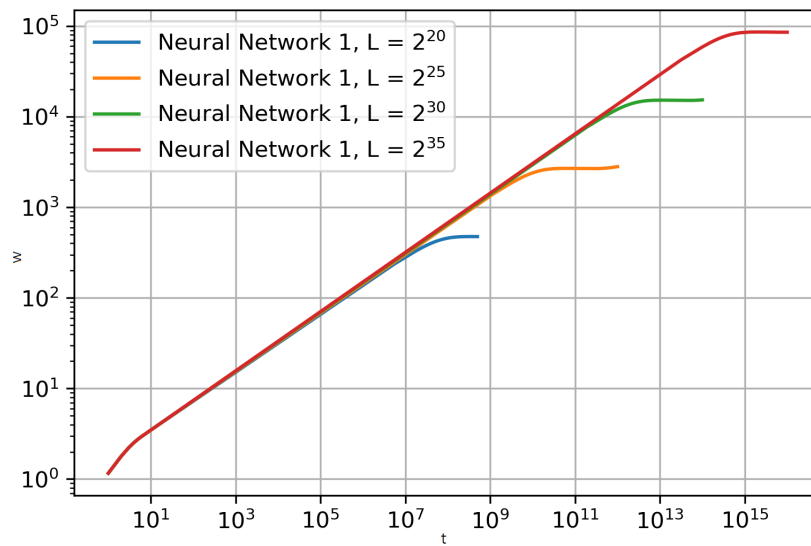


Figura 52 – Extrapolações Com perceptron multicamadas-1 camada escondida.

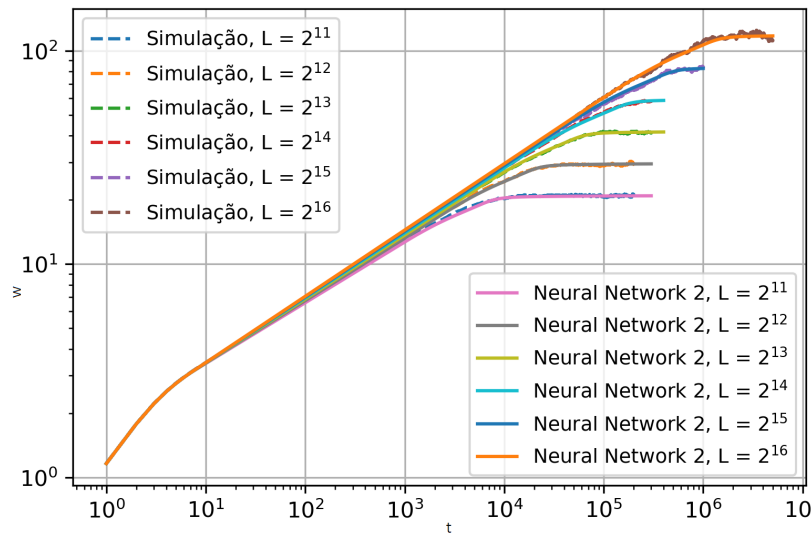


Figura 53 – Extrapolações Com perceptron multicamadas-2 camadas escondidas.

Como até o momento não há na literatura curvas para tais comprimentos, não temos uma forma concreta de confirmar a validade de tais curvas. Contudo, podemos afirmar que as mesmas crescem de acordo com a teoria proposta no capítulo 2, além de que, quando aplicado a equação 3.1 para tais comprimentos, estes convergem para valores de α que estão de acordo com a teoria. A partir disto, podemos gerar a tabela 11 com os valores de α encontrados.

Tabela 11 – Tabela dos valores de alpha

	SVM	RF	RNN1	RNN2	média
α	0,4996	0,4999	0,4991	0,5001	0,4997

Podemos notar que todos os modelos convergem para um valor de α igual à 0,5. A tabela 12 expõe a rugosidade de saturação para cada modelo, é possível notar que

Tabela 12 – Tabela com as rugosidades de saturação para cada modelo.

L/w_{sat}	SVM	RF	RNN1	RNN2	média	desvio
2^{15}	83,6876	84,3637	83,8789	83,6371	83,8918	0.2869
2^{16}	118,9727	118,6786	118,5638	117,7445	118,4899	0.4554
2^{20}	474,4225	473,1671	474,7336	470,6204	473,2359	1,6199
2^{25}	2678,0063	2702,7722	2796,4751	2676,3306	2713,3960	49,0950
2^{30}	15154,6287	15121,9478	15291,2253	15127,1727	15173,7436	68,9546
2^{35}	85925,9700	85722,0806	85493,2085	85479,9836	85655,3107	183,5289

existe uma proximidade grande entre os valores encontrados. Como as redes perceptron multicamadas 2 tiveram os melhores resultados, empregamos esta para calcular as correções de α propostas por REIS [16].

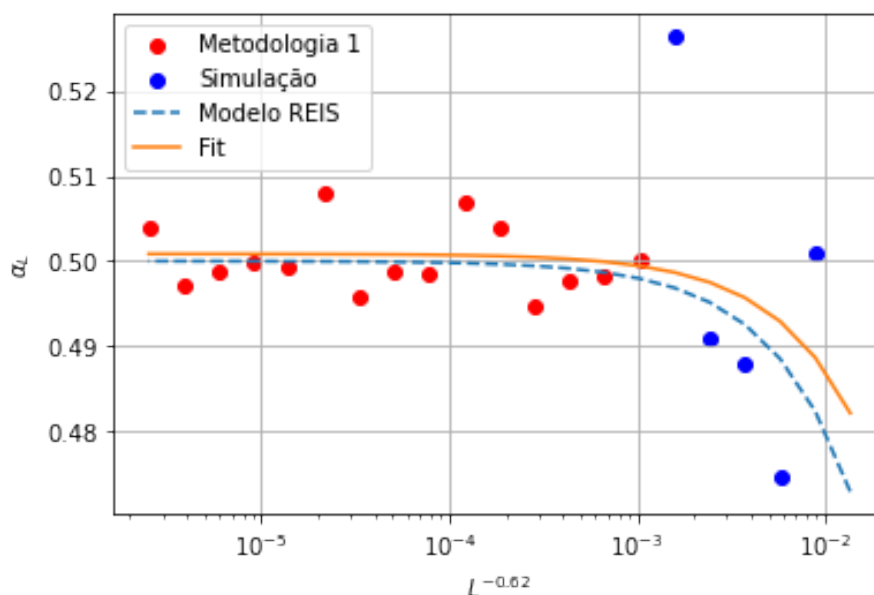


Figura 54 – Correções de α_L segundo o modelo proposto pelo autor [16].

É notável pela figura 54 que as curvas geradas pelas redes neurais incorporam bem os efeitos de correção do expoente α .

Para a segunda metodologia, a figura 55 apresenta os resultados da segunda metodologia em comparação com a primeira para os comprimentos laterais de $L = [2^{16}, 2^{20}, 2^{25}, 2^{30}]$, sendo as curvas contínuas da metodologia 2 e as curvas tracejadas da metodologia 1.

Tabela 13 – RMSEPE das curvas de crescimento para ambas metodologias.

Comprimentos	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{25}	2^{30}
Erro(%)	0,1085	0,0794	0,1075	0,1574	0,1850	0,9830	5,0211

A partir da tabela 14 é possível observar que as rugosidades de saturação geradas pela primeira e segunda metodologia estão próximas, com exceção das duas últimas, uma

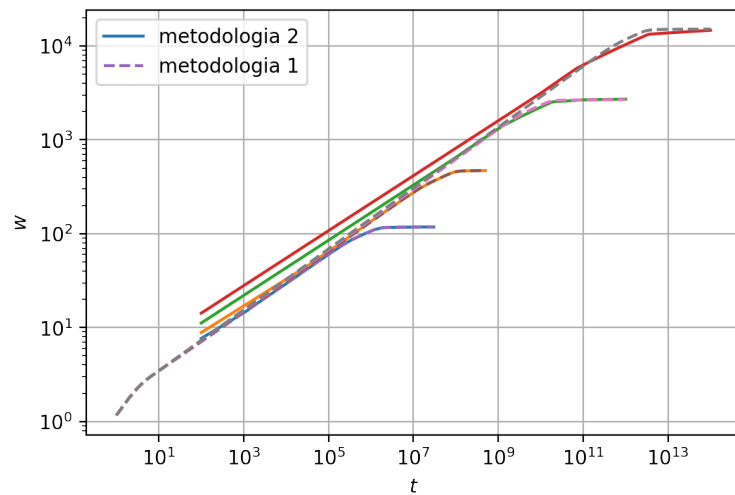


Figura 55 – Curvas geradas pela segunda metodologia comparado com a segunda metodologia.

Tabela 14 – Comparação da rugosidade de saturação em ambas metodologias

Comprimentos	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{25}	2^{30}
Metodologia 1	118,0963	166,8193	235,5291	331,8636	470,6337	2676,4833	15128,2948
Metodologia 2	117,9407	166,6785	235,2172	331,2588	470,7196	2713,0050	14672,3800

possível justificativa para tal está no fato de que existe um desbalanceamento dos dados, havendo mais dados na saturação que no crescimento, de forma que a rede aprendeu apenas a parte estacionária, além disto, as redes não foram expostas para tal alcance de tempo, fazendo com que elas comecem a perder o comportamento estacionário na saturação. Para este modelo encontramos um expoente α igual à 0.4988. Suas correções podem ser observadas na figura 56.

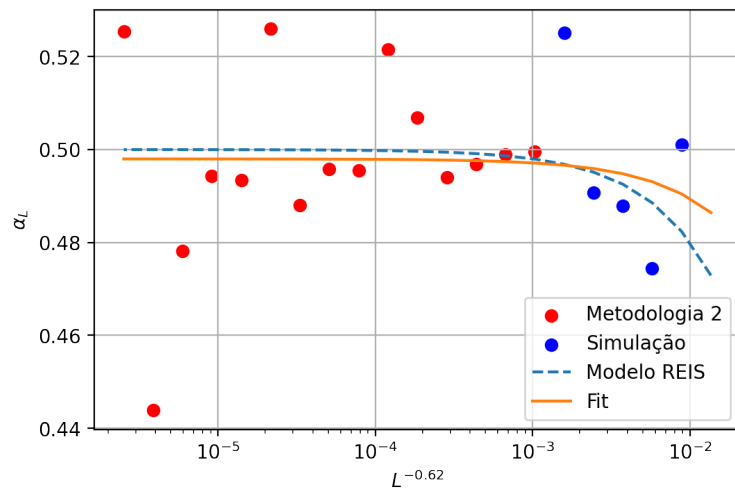


Figura 56 – Correções de α_L para a segunda metodologia em comparação o modelo proposto pelo autor [15].

Percebemos que o α_L apresenta mais ruído comparado a metodologia anterior, isto será mais explicado mais a frente nas discussões.

4.2 Balístico Bidimensional

Para o modelo de deposição balística bidimensional foi realizado inicialmente um treino a partir das curvas de crescimento com comprimentos de $L = [2^9, 2^{10}, 2^{11}]$, posteriormente, foi realizado um *fine tuning* para gerar curvas mais suaves. As curvas geradas do modelo de deposição balística bidimensional sem *fine tuning*, rede 1, podem ser vista nas figuras 57 e 58.

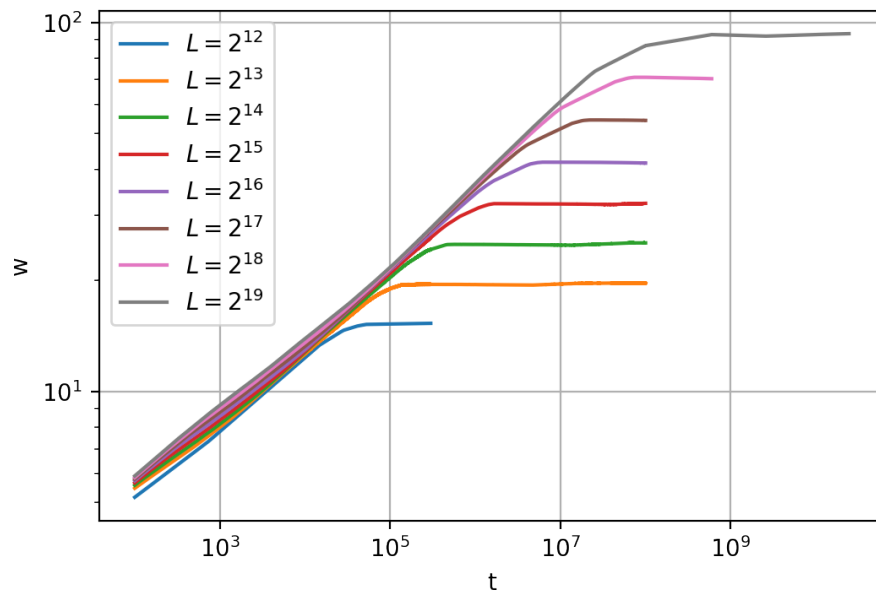


Figura 57 – Previsões da rede neural sem *fine tuning* para comprimentos até 2^{13} – 2^{19} .

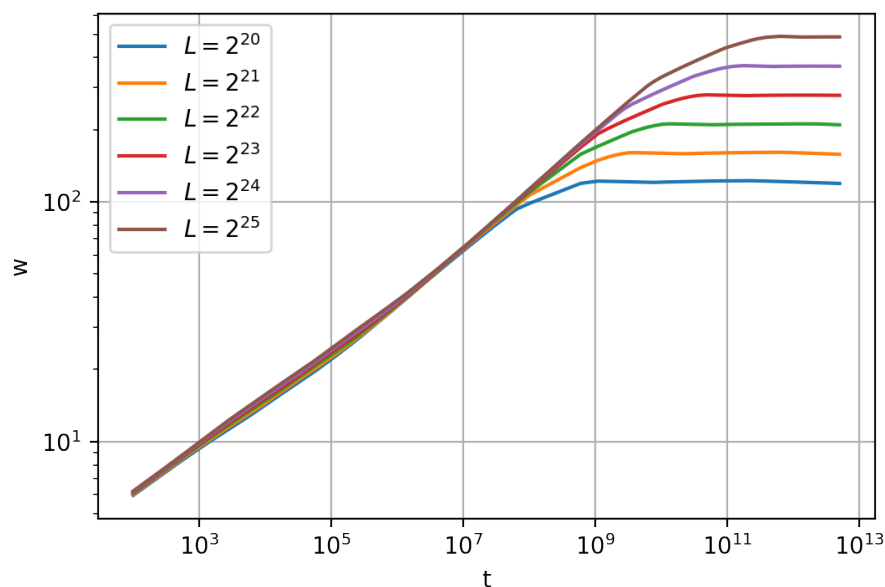


Figura 58 – Previsões da rede neural sem *fine tuning* para comprimentos entre 2^{20} – 2^{25} .

É possível notar que as curvas das figuras 57 e 58 conseguiram captar o crescimento do modelo bidimensional, contudo as mesmas não conseguiram aprender os pequenos detalhes associados ao crescimento. Isto pode ser justificado por dois fatores, o primeiro está na

quantidade de comprimentos utilizados para treino, e o segundo motivo está no fato de realizarmos um processo iterativo, propagando o erro das curvas anteriores.

Para melhorar o modelo da rede 1, realizamos um *fine tuning*, rede 2, nos hiperparâmetros *batch size*, taxa de aprendizado e acrescentamos a curva de 2^{12} no conjunto de treino, gerando as curvas 59 e 60.

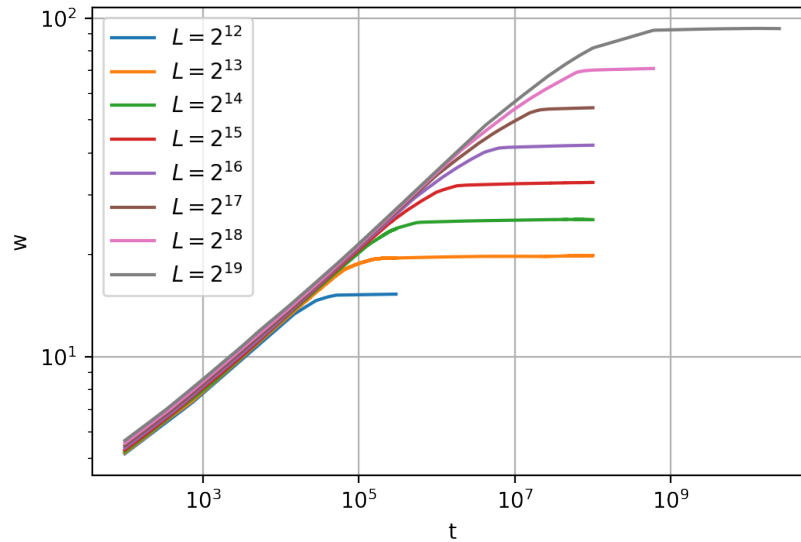


Figura 59 – Previsões da rede neural com *fine tuning* 1 para comprimentos até $2^{13} - 2^{19}$.

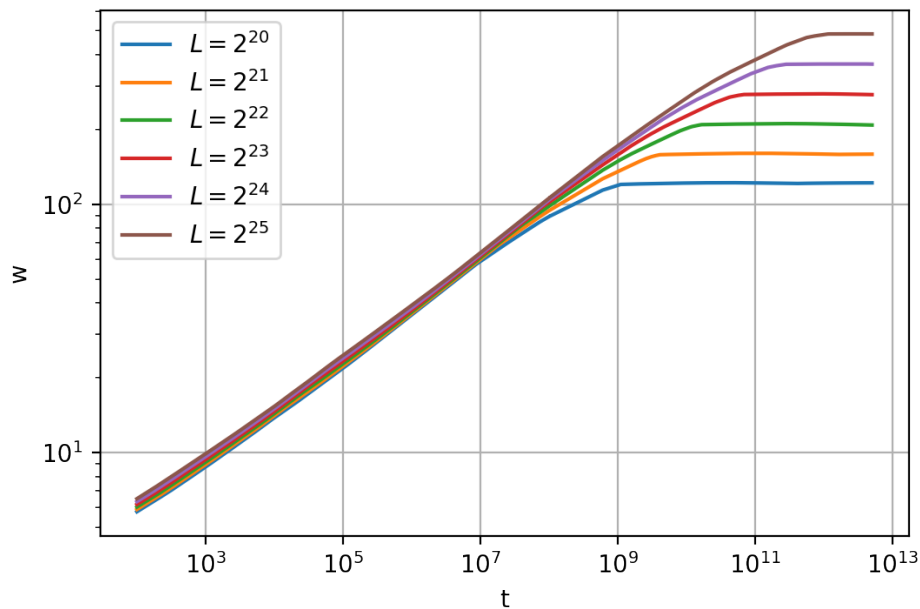


Figura 60 – Previsões da rede neural com *fine tuning* para comprimentos até $2^{20} - 2^{25}$.

Percebemos que as curvas se comportam melhor após este ajuste fino, isto tanto no crescimento como na saturação. As correções de α_L para esta rede podem ser observadas na figura 61.

O modelo com ajuste fino apresenta correções com pontos mais próximos em torno do modelo proposto pelo trabalho de referência.

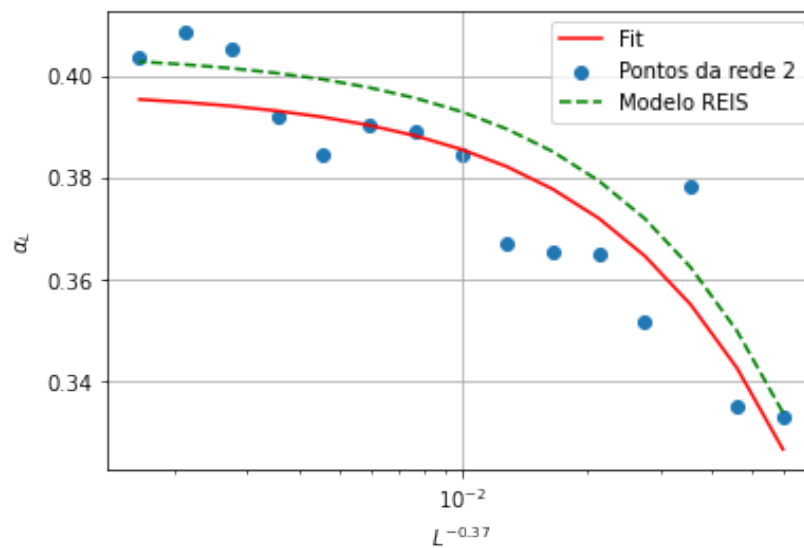


Figura 61 – Correções de α para a rede com *fine tuning* em relação ao modelo proposto pelo autor [16].

Por fim, a rede que realizou o treinamento sem pesos pré treinados e com as as curvas $L = [2^9, 2^{10}, 2^{11}, 2^{12}]$, rede 3, gerou as curvas das figuras 62 e 63.

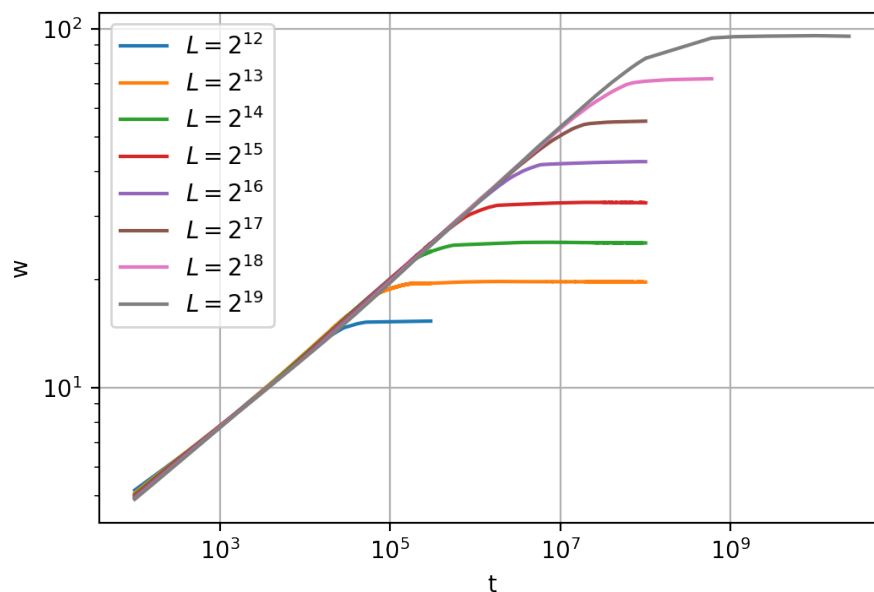


Figura 62 – Previsões da segunda rede neural para comprimentos entre $2^{13} - 2^{19}$.

É notável que as curvas para os comprimentos iniciais crescem de forma esperada quando comparada com as curvas dos modelos anteriores, contudo, para comprimentos acima de 2^{19} , as curvas começam cair levemente na região de saturação, a provável explicação para isto, é pelo fato das curvas não serem treinadas para tal alcance de tempo.

A figura 64 expõe as correções para o último modelo. A rede 2 foi treinada com *batch size* igual a 128 e taxa de aprendizado 0,008. Enquanto que a rede 3 realizou um treino com *batch size* igual a 32 e taxa de aprendizado de 0,001.

É possível notar que a rede 1 e rede 2 oscilam mais no início em relação a segunda,

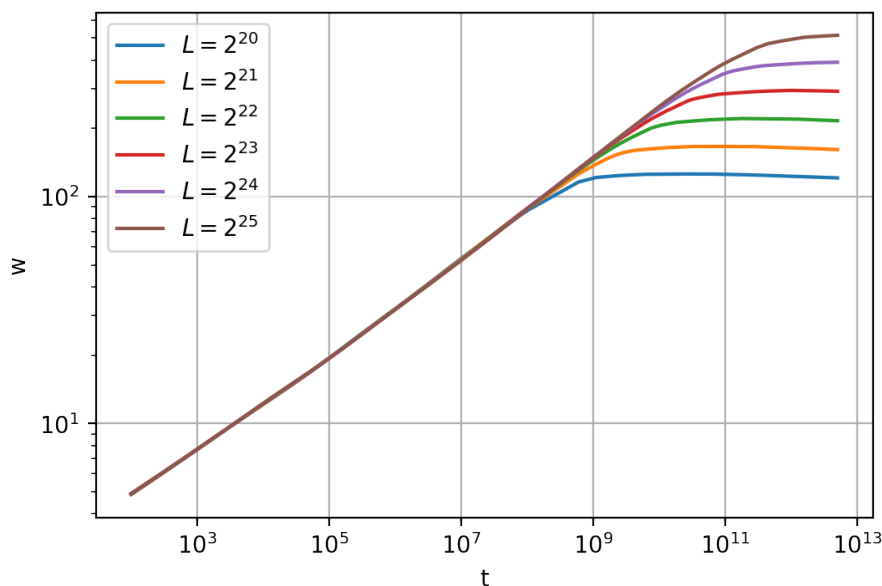


Figura 63 – Previsões da segunda rede neural para comprimentos entre $2^{20} - 2^{25}$.

Tabela 15 – Comparação da rugosidade de saturação para todos os modelos para comprimentos entre $2^{13} - 2^{19}$.

Comprimentos	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}
Modelo autor	19,8843	25,7343	33,4766	43,7212	57,2765	75,2128	98,9477
Rede 1	19,6728	25,2771	32,3612	41,6051	54,2829	70,4609	93,2196
Rede 2	19,8969	25,3933	32,7059	42,1337	54,3411	70,9400	92,8944
Rede 3	19,7032	25,3880	32,7627	42,5871	56,2064	74,2993	97,5492

Tabela 16 – Comparação da rugosidade de saturação para todos os modelos para comprimentos entre $2^{20} - 2^{25}$.

Comprimentos	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}	2^{25}
Modelo autor	130,3577	171,9276	226,9469	299,7706	396,1642	523,7605
Rede 1	122,3420	159,1248	210,0201	278,3000	367,9697	487,6380
Rede 2	121,7746	158,9627	208,5686	276,2559	366,7407	485,1406
Rede 3	131,0807	172,2267	232,3531	317,0966	425,7200	576,6094

contudo generalizam melhor a longo prazo, enquanto que a rede 3 aprendeu melhor o crescimento inicial, porém no final começou a oscilar mais no comprimentos finais. Podemos ver pelas tabelas 15 e 16 que os valores da rugosidade flutuam levemente para cada modelo de rede em relação ao modelo do autor.

Extraindo a saturação das curvas geradas, chegamos a um expoente α , 0,3958, junto com a de curva de correções 61, sendo o “*fit*” a curva gerada com as curvas até o comprimento de $L = 2^{19}$, enquanto que o “*fit geral*” é a curva com todas as rugosidades de saturação extraídas.

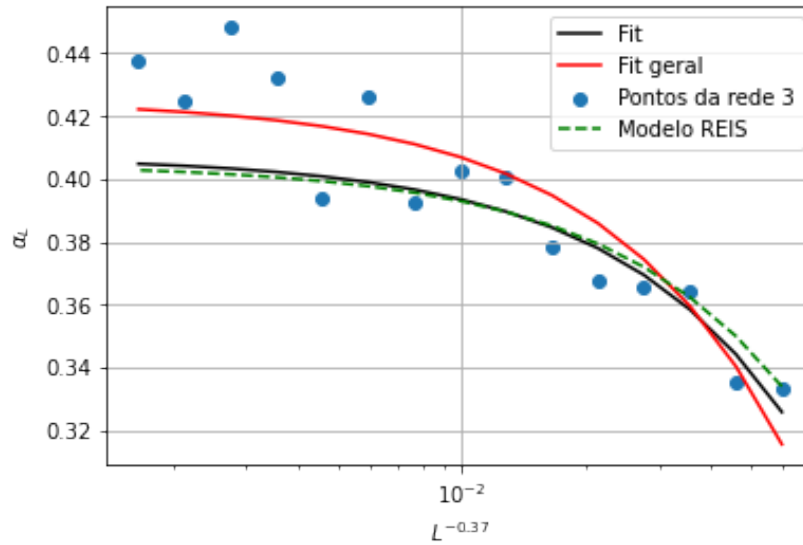


Figura 64 – Correções de α para o segundo modelo de rede em comparação ao do autor [16].

4.3 Comentários e Discussão

Dentre as metodologias propostas, a primeira metodologia para a deposição balística unidimensional foi a que obteve melhores resultados. Com apenas uma curva da rugosidade global e conhecimento do modelo, conseguimos gerar curvas de rugosidade para comprimentos laterais que ainda não foram observadas na literatura e que demandam uma grande quantidade de poder computacional para serem geradas. Podemos observar que para comprimentos laterais de interesse, $2^{20} - 2^{35}$, a quantidade de unidades de tempo para que as curvas atinjam a saturação são acima de 10^7 , chegando a 10^{15} para o maior comprimento. Analisando tal de um ponto de vista computacional, vemos que o primeiro laço de repetição terá em torno de $2^{35} \sim 3 \times 10^{10}$ iterações para a primeira unidade de tempo, e como queremos 10^{15} unidades de tempo, ficaremos com 3×10^{25} iterações para a máquina realizar, isto apenas para uma única realização independente. Enquanto que via aprendizado de máquina, o tempo computacional para gerar tal curvas dependerá majoritariamente da quantidade de dados a serem extrapolados. Para termos de comparação, para gerar a curva de $L = 2^{15}$ que possui 10^6 pontos, foi necessário aproximadamente 1 minuto para a rede neural perceptron multicamadas 2, enquanto que a simulação computacional levou 1 dia, as demais comparações podem ser visualizadas na tabela 17.

Tabela 17 – Comparação entre o tempo para a simulação computacional e a metodologia 1.

Comprimentos	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}
tempo(s) RNA	10	10	15	15	60	210
tempo(h) Sim. Comp.	1	2	5	10	24	24

A tabela 18 de erros expõe a média dos valores obtidos dos erros da raiz média quadrática percentual das curvas geradas a partir dos modelos da metodologia 1, nesta é

possível observar que todos os modelos de aprendizado de máquina propostos conseguiram obter resultados satisfatórios nas suas previsões, em que obtivemos erros menores que 2% para os comprimentos laterais entre 2^{11} , 2^{12} , 2^{13} , 2^{14} , algo esperado, já que as curvas geradas via simulação computacional apresentam ruído aleatório e as curvas geradas a partir do aprendizado de máquina geraram uma curva média. O único modelo que aprendeu o ruído dos dados foi a floresta aleatória, este claramente aprendeu o ruído dos dados durante o treino, tal comportamento pode ser observado na tabela de erro 12, sendo o valor do erro para o comprimento de 2^{11} desproporcionalmente menor que os demais comprimentos, além disto, podemos observar que as demais curvas geradas pelas florestas aleatórias apresentam ruído na saturação. Por termos realizado uma quantidade menor de médias configuracionais para os dois últimos comprimentos, 2^{15} e 2^{16} , tais curvas apresentam uma quantidade maior de ruído, que conseqüentemente gerou um maior valor de erro quando comparado com as demais curvas.

Tabela 18 – Média dos erros de todos os modelos da metodologia 1.

Comprimentos	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}
Média(%)	0.5688	1.4257	1.4747	1.366275	2.61325	2.4674

Além desta metodologia ser capaz de gerar curvas próximas ao esperado, ela também pode ser aplicada à outros modelos de deposição. Por exemplo, as curvas geradas para o modelo de deposição aleatória com relaxação de superfície começam a convergir quando aplicado a escala de Family-Vicsek para os valores de α e z teóricos, portanto, podemos usar a metodologia 1 para gerar curvas para tamanhos laterais superiores de forma similar ao modelo de deposição balística.

Contudo, nem todos os modelos de deposição convergem quando aplicados à escala de Family-Vicsek com os expoentes teóricos, como o modelo de deposição balística bidimensional, para tal caso, a metodologia 1 falha em gerar curvas que descrevem a evolução temporal da rugosidade global. Portanto, na metodologia 2 é assumido que as redes neurais são aproximadores universais, e que existe uma função da rugosidade global $w(L, t)$, tal que quando a rede é exposta a um número grande o suficientemente de dados de tempo e comprimento, esta aprenderá o crescimento da rugosidade global para qualquer comprimento lateral.

Na metodologia 2 usamos um perceptron multicamadas para realizar a previsão da evolução temporal da rugosidade global, empregamos como entradas o tempo e o comprimento, saída à rugosidade. Aplicamos transformações não linear nos dados, de forma com que sua escala se torne próxima. Um ponto importante está no sinal invertido da rugosidade, ao empregar a rugosidade desta forma seu treino começou a atingir melhores resultados e um aprendizado mais acelerado. Uma possível explicação para tal está no fato de que a nova curva processada aparenta ser uma função convexa, diferente da curva com

signal positivo.

Comparando ambas metodologias, podemos perceber pela tabela 19 que as curvas geradas pelas duas metodologias apresentam crescimentos similares, com erros menores do que 1% para curvas menores que 2^{25} . Todavia, é possível notar que para a curva de 2^{30} a rede começa a falhar tanto no início como para o fim do crescimento, a principal causa disto está no desbalanceamento dos dados e pelo alcance dos mesmo. Como os dados de saturação são em quantidade superiores aos dados de crescimento linear, a rede fica mais propícia a aprender mais de uma região que a outra, além de que, estamos ensinando à rede neural dados que vão até 10^8 , enquanto que para o comprimento lateral final, a curva cresce até 10^{13} . Acreditamos que adicionando mais dados no crescimento linear inicial e aumentando o alcance dos dados, um processo de *fine tuning* poderá melhorar o desempenho da rede.

Tabela 19 – RMSEPE das curvas de crescimento para ambas metodologias.

Comprimentos	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{30}
Erro(%)	0,1085	0,0794	0,1075	0,1574	0,1850	0,9830

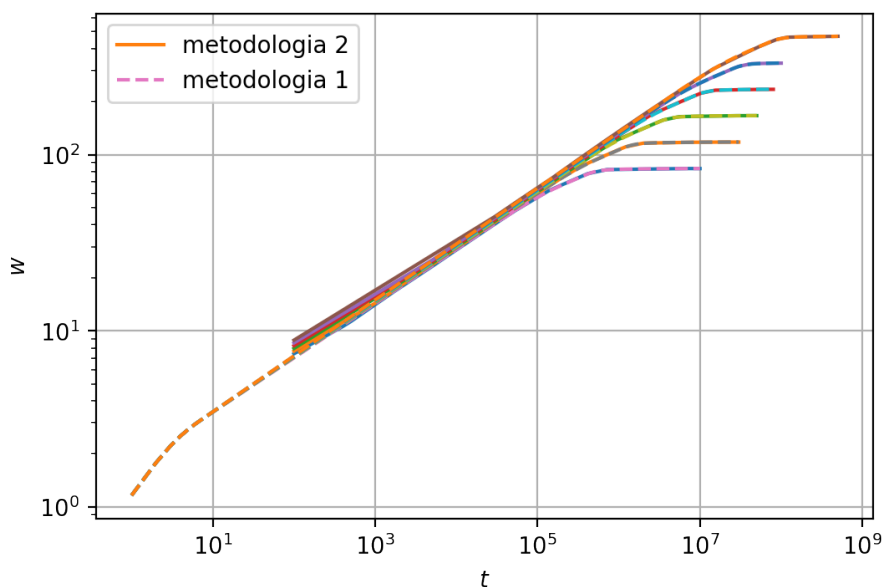


Figura 65 – Previsões da metodologia 2 comparada com a metodologia 1 para os comprimentos $2^{15} - 2^{20}$.

Outro aspecto importante do trabalho é calcular o valor do expoente α e suas correções para o limite hidrodinâmico. É possível observar que para a metodologia 1 todos os modelos de aprendizado de máquina propostos conseguiram chegar a resultados muito próximos do valor esperado teoricamente, o mesmo vale para a segunda metodologia, o que indica que o modelo balístico de fato possui um valor de expoente de rugosidade igual a 0,5 para o caso unidimensional. Além disto, nossas metodologias conseguem incorporar as correções propostas por REIS [16] de modo a evidenciar o comportamento do α ao longo dos comprimentos. A figura 66 expõe o comportamento do expoente ao longo do

comprimento para ambas metodologias, é possível notar que a metodologia 2 possui uma flutuação estatística maior, isto pelo fato de estarmos tentando aproximar uma função bidimensional com apenas quatro curvas de diferentes comprimentos. Uma quantidade maior de curvas para diferentes comprimentos geraria um aprendizado mais estável.

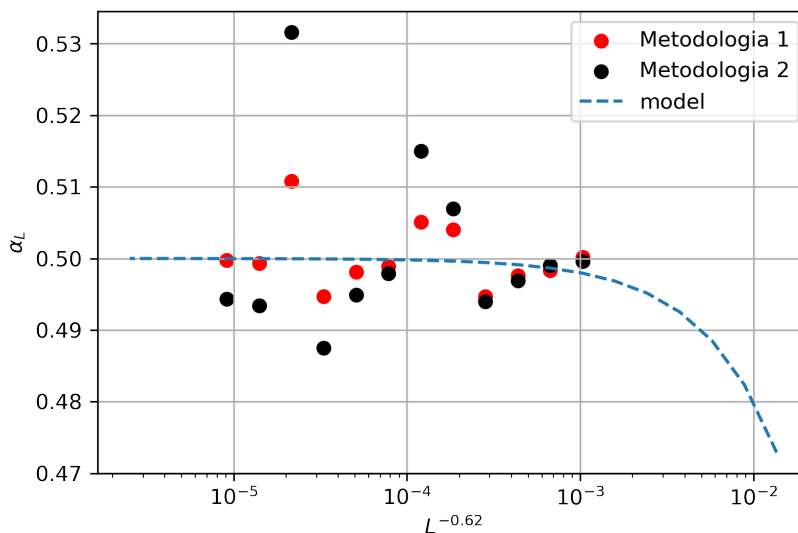


Figura 66 – Correções de α_L para a primeira e segunda metodologia em comparação o modelo proposto pelo autor [15].

A segunda parte do trabalho consistiu na regressão das curvas do modelo de deposição balística bidimensional. Neste caso, não tínhamos como realizar a metodologia 1 do caso unidimensional por conta do expoente α não possuir um valor constante para as curvas de rugosidade simuladas, e apenas os valores de tempo e comprimento não conseguiram prever o crescimento da rugosidade. Então, decidimos usar uma rede LSTM de forma a empregar informação das rugosidades anteriores para prever as próximas. Realizamos previsões utilizando um, dois e três comprimentos anteriores, contudo, com apenas três comprimentos conseguimos gerar um crescimento satisfatório. Por conta da complexidade da simulação computacional, não foi possível realizar simulações do modelo para comprimentos laterais superiores a 2^{12} , por conta disto, as curvas geradas inicialmente pela rede neural começaram a divergir do esperado como demonstrado na figura 57.

Como obtivemos bons resultados em comparação ao modelo proposto por REIS [16], e as curvas iniciais que começamos a gerar estavam de acordo com os valores propostos pelo mesmo, decidimos utilizar a curva 2^{13} para treinar a rede, sendo que realizamos um treino com pesos pré-treinados e um outro sem. Com isto conseguimos gerar curvas que crescem de forma coerente com o que já vínhamos adquirindo, e com menos desvios em seu crescimento. A partir das curvas de correção para estes modelos, podemos argumentar que a rede 2 oscila mais que a rede 3, contudo generaliza melhor o crescimento da rugosidade de saturação, enquanto que a rede 3 flutua menos ao redor das correções, porém falha na hora de generalizar para grandes comprimentos.

A figura 67 expõe as curvas da evolução temporal da rugosidade global para o caso

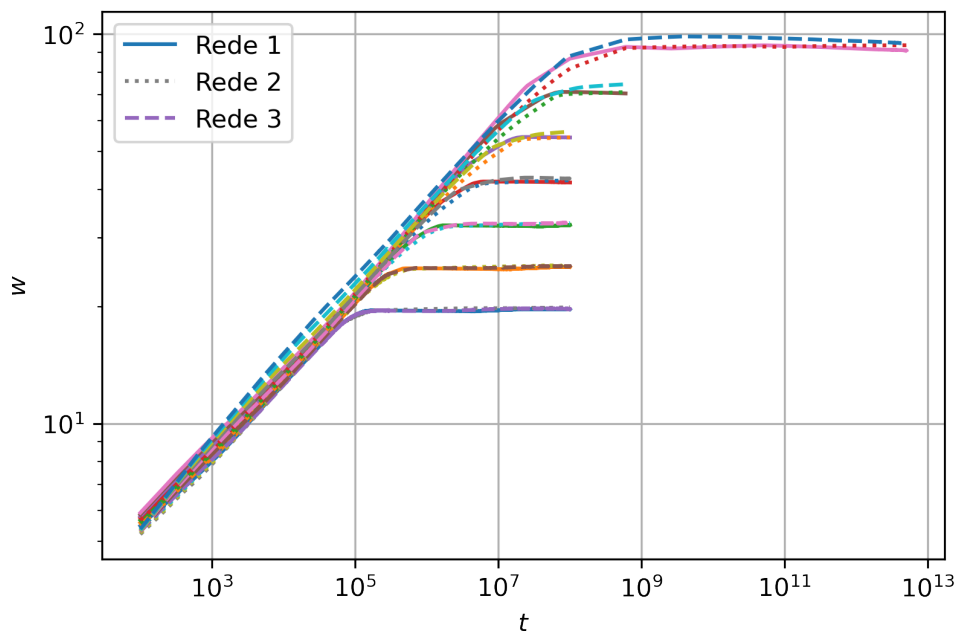


Figura 67 – Previsão das curvas de comprimento lateral $2^{13} - 2^{19}$ para os diferentes treinos das redes neurais.

bidimensional. Uma característica importante deste modelo de aprendizado, é que ele usa as rugosidades dos comprimentos anteriores de forma a extrapolar para novos comprimentos, então, a rede irá propagar os erros associados as rugosidades anteriores, este é um dos principais fatores pelos qual não realizamos previsões para comprimentos laterais superiores a $L = 2^{25}$.

A partir dos gráficos de correções 61 e 64 é possível observar que apenas a rede 3 com uma parcela dos dados consegue realizar um *fitting* próximo ao do proposto pelo autor, enquanto que os demais flutuam em torno do modelo, isto está atrelado ao fato que α_L é sensível a pequenas variações de w_{sat} , algo que o autor discute em seu trabalho. Esta sensibilidade pode ser observada quando comparamos os valores de w_{sat} das curvas geradas pela rede neural com aquelas geradas pelo modelo do autor, diferenças de casas decimais geram flutuações nos valores α_L . Ainda existe diversas incertezas em torno deste coeficiente para o caso bidimensional, contudo, os resultados encontrados via rede neural estão próximos aos de REIS [16], sendo este a principal referência deste trabalho, em que este encontra $\alpha = 0,404$. Contudo, ainda há trabalhos que propõem diferentes valores para este expoente, tais como GOMES [43] que propõe um valor de $\alpha = 0,3888$ e OLIVEIRA [44] que propõe $\alpha = 0,3819$.

5 Conclusões

Neste trabalho vimos como aprendizado de máquina pode ser empregado para realizar a previsão da curva de crescimento da rugosidade para o modelo balístico unidimensional e para o bidimensional. Em ambos os casos, conseguimos curvas que crescem segundo a teoria do modelo e apresentam resultados satisfatórios de um ponto de vista teórico. Um aspecto importante dos modelos propostos é que todos estes conseguiram resultados em pequenas quantidades de tempo, enquanto via simulação computacional levaria meses, senão anos. A simulação do modelo balístico bidimensional para o comprimento de $L = 2^{12}$ durou cerca de 48 horas para 5 realizações independentes, enquanto que a rede neural gerou a mesma quantidade de pontos com curva mais suavizada, em cerca de ~ 1 minuto. O tempo computacional da rede neural varia de acordo com a quantidade de pontos que queremos prever, isto para qualquer comprimento, diferente da simulação computacional.

Vimos que as correções de α seguem o modelo proposto por REIS [16], isto tanto para o caso unidimensional, como para o bidimensional. Conseguimos perceber que os modelos que tentam realizar uma regressão direta a partir do padrão de rugosidade apresentam um pouco de desvio quando tendemos a grandes valores de tempo, uma possível justificativa para tal está no alcance dos dados de tempo e pela pouca quantidade de dados disponíveis para treino. Todavia, é possível concluir que os modelos de aprendizagem de máquina conseguiram aprender o padrão de crescimento do modelo de deposição balística tanto para o caso unidimensional como para o bidimensional. A partir das metodologias propostas, conseguimos não somente prever a região de saturação, como também, prever o comportamento da rugosidade global e chegar a valores de α que são condizentes com os teóricos, o que sustenta a hipótese de que o modelo de deposição balística pertence a classe de universalidade KPZ.

As perspectivas deste trabalho são aprofundar cada vez mais o uso de redes neurais para a previsão do crescimento do padrão da rugosidade. Empregar técnicas como *transfer learning* de forma a transferir o aprendizado de uma rede neural com ótimos resultados para outra que tente prever outro modelo de deposição. Com esta abordagem poderemos salvar horas de simulação computacional, treino, *fine tuning*.

Além disto, ainda há diferentes arquiteturas de redes neurais que podem ser empregadas no problema, arquiteturas como as de redes neurais residuais, em que as entradas contribuem na saída da rede, se sabemos que a curva de rugosidade cresce com o tempo e o comprimento, podemos modelar a rede de forma a encontrar os parâmetros que acompanhem essas variáveis. além dos dados, os processos de deposição nos oferece informação extra sobre o sistema, sendo estas as equações 2.6 - 2.8, então, podemos utilizar uma rede similar a uma *physics informed neural networks* de forma a usufruir da informação extra

que temos sobre o sistema, e usar as equações 2.6 - 2.8 de forma a regularizar a rede neural.

Queremos estender este trabalho para modelos de deposição com possíveis aplicações, aos quais é inserido um fator de difusão na simulação computacional e fazendo com que este controle o crescimento da rugosidade em função da temperatura, podemos citar, como exemplo, Processos de Epitaxia por feixe molecular, que empregam este conceito [45].

6 Apêndice

6.1 Apêndice A

Código para a deposição unidimensional.

```
import numpy as np
import pandas as pd
from numba import njit , prange

@njit(parallel=True)
def w(T,RI,L):
    w = np.zeros((RI,T))
    for O in prange(RI):
        h = np.zeros(L,dtype=np.int64)
        for t in range(T):
            ale1 = np.random.randint(0,L,(1,L))[0]
            for j in ale1:
                h[j] = max(1+h[j],h[(j-1)%L],h[(j+1)%L])
            w[O][t] = np.mean(h**2)-np.mean(h)**2
    k = np.sqrt(np.array([np.sum(w[:,i])/RI for i in range(T)]))
    return k

w15 = w(5*10**6,50,2**16)
t = np.arange(1,5*10**6+1).reshape(-1,1)
def save_guard(t,y,pot):
    df = np.append(t,y.reshape(-1,1), axis = 1)
    df = pd.DataFrame(df, columns = ['tempo','rug'])
    df.to_csv(f'w{pot}.csv')

save_guard(t, w15, '16_50R')
```

6.2 Apêndice B

Código para a deposição bidimensional.

```
import numpy as np
from numba import njit, prange
import pandas as pd

@njit(parallel=True, fastmath=True)
def w(T, RI, L):
    n = L*L
    w_cres = np.zeros(T)
    for O in prange(RI):
        h = np.zeros((L,L), dtype=(np.int64))
        for t in np.arange(T):
            ale1 = np.random.randint(0,L, size = (1,n))[0]
            ale2 = np.random.randint(0,L, size = (1,n))[0]
            for i, j in zip(ale1, ale2):
                h[i][j] = max(1+h[i][j],
                               h[(i-1)%L][j%L],
                               ,h[(i+1)%L][j%L],
                               h[i][(j-1)%L],
                               h[(i)%L][(j+1)%L])
            media = np.mean(h)
            w_cres[t] += np.mean((h-media)**2)
            h = h - np.min(h)
    w_cres = np.sqrt(w_cres/RI)
    return w_cres

wn = w(8*10**4, 5, 2**12)

t = np.arange(1, 8*10**4+1).reshape(-1,1)

def save_guard(t, y, pot):
    df = np.append(t, y.reshape(-1,1), axis = 1)
    df = pd.DataFrame(df, columns = ['tempo', 'rug'])
    df.to_csv(f'w{pot}.csv')

save_guard(t, wn, '12')
```


Referências

- 1 HAGHIGHAT, E.; JUANES, R. Sciann: A keras/tensorflow wrapper for scientific computations and physics-informed deep learning using artificial neural networks. *Computer Methods in Applied Mechanics and Engineering*, Elsevier, v. 373, p. 113552, 2021. 1
- 2 PILANIA, G. Machine learning in materials science: From explainable predictions to autonomous design. *Computational Materials Science*, Elsevier, v. 193, p. 110360, 2021. 1
- 3 CARLEO, G. et al. Machine learning and the physical sciences. *Reviews of Modern Physics*, APS, v. 91, n. 4, p. 045002, 2019. 1
- 4 WIGLEY, P. B. et al. Fast machine-learning online optimization of ultra-cold-atom experiments. *Scientific reports*, Nature Publishing Group, v. 6, n. 1, p. 1–6, 2016. 1
- 5 RAISSI, M.; PERDIKARIS, P.; KARNIADAKIS, G. E. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, Elsevier, v. 378, p. 686–707, 2019. 1
- 6 HORNIK, K.; STINCHCOMBE, M.; WHITE, H. Multilayer feedforward networks are universal approximators. *Neural networks*, Elsevier, v. 2, n. 5, p. 359–366, 1989. 1, 26
- 7 COVA, T. F.; PAIS, A. A. Deep learning for deep chemistry: optimizing the prediction of chemical patterns. *Frontiers in chemistry*, Frontiers, v. 7, p. 809, 2019. 1
- 8 HERMANN, J.; SCHÄTZLE, Z.; NOÉ, F. Deep-neural-network solution of the electronic schrödinger equation. *Nature Chemistry*, Nature Publishing Group, v. 12, n. 10, p. 891–897, 2020. 1
- 9 RAY, S.; MAL, B.; SHAMANNA, J. Generalized ballistic deposition in 2 dimensions: scaling of surface width, porosity and conductivity. *arXiv preprint arXiv:1503.01047*, 2015. 1, 2
- 10 KRUG, J. Origins of scale invariance in growth processes. *Advances in Physics*, Taylor & Francis, v. 46, n. 2, p. 139–282, 1997. 2
- 11 MAL, B.; RAY, S.; SHAMANNA, J. Surface properties and scaling behavior of a generalized ballistic deposition model. *Physical Review E*, APS, v. 93, n. 2, p. 022121, 2016. 2
- 12 MESSIER, R. Thin film deposition processes. *MRS Bulletin*, Cambridge University Press, v. 13, n. 11, p. 18–21, 1988. 2
- 13 GRÜNER, C. et al. Avoiding anisotropies in on-lattice simulations of ballistic deposition. *physica status solidi (b)*, Wiley Online Library, v. 258, n. 3, p. 2000036, 2021. 2
- 14 BANERJEE, K.; SHAMANNA, J.; RAY, S. Surface morphology of a modified ballistic deposition model. *Physical Review E*, APS, v. 90, n. 2, p. 022111, 2014. 2

- 15 FARNUDI, B.; VVEDENSKY, D. D. Large-scale simulations of ballistic deposition: The approach to asymptotic scaling. *Physical Review E*, APS, v. 83, n. 2, p. 020103, 2011. 2, 3, 9
- 16 REIS, F. A. Universality and corrections to scaling in the ballistic deposition model. *Physical Review E*, APS, v. 63, n. 5, p. 056116, 2001. 2, 5, 9, 14, 34, 35, 39, 45, 46, 51, 59, 60, 61, 62
- 17 REIS, F. A. Roughness fluctuations, roughness exponents and the universality class of ballistic deposition. *Physica A: Statistical Mechanics and its Applications*, Elsevier, v. 364, p. 190–196, 2006. 2
- 18 MIRANDA, V. G.; REIS, F. D. A. Numerical study of the kardar-parisi-zhang equation. *Physical Review E*, APS, v. 77, n. 3, p. 031134, 2008. 2
- 19 ALVES, S. G.; FERREIRA, S. C. Scaling, cumulant ratios, and height distribution of ballistic deposition in $3+1$ and $4+1$ dimensions. *Physical Review E*, APS, v. 93, n. 5, p. 052131, 2016. 2
- 20 TALBOT, J.; RICCI, S. Analytic model for a ballistic deposition process. *Physical review letters*, APS, v. 68, n. 7, p. 958, 1992. 2
- 21 SONG, T.; XIA, H. Extensive numerical simulations of surface growth with temporally correlated noise. *Physical Review E*, APS, v. 103, n. 1, p. 012121, 2021. 2
- 22 HOROWITZ, C. M.; ALBANO, E. V. Dynamic properties in a family of competitive growing models. *Physical Review E*, APS, v. 73, n. 3, p. 031111, 2006. 2
- 23 REIS, F. A. Anomalous roughening in competitive growth models with time-decreasing rates of correlated dynamics. *Physical Review E*, APS, v. 84, n. 3, p. 031604, 2011. 2
- 24 COMETS, F.; DALMAU, J.; SAGLIETTI, S. Scaling limit of the heavy-tailed ballistic deposition model with p -sticking. *arXiv preprint arXiv:2203.06133*, 2022. 2
- 25 DAS, S.; BANERJEE, D.; ROY, J. Stochastic study of random-ballistic competitive growth model in $2+1$ dimension and related scaling exponents. *Journal of The Institution of Engineers (India): Series D*, Springer, p. 1–8, 2022. 2
- 26 BARABÁSI, A.-L.; STANLEY, H. E. et al. *Fractal concepts in surface growth*. [S.l.]: Cambridge university press, 1995. 8, 11
- 27 FAMILY, F.; VICSEK, T. Scaling of the active zone in the eden process on percolation networks and the ballistic deposition model. *Journal of Physics A: Mathematical and General*, IOP Publishing, v. 18, n. 2, p. L75, 1985. 9
- 28 GÉRON, A. Hands-on machine learning with scikit-learn and tensorflow: Concepts. *Tools, and Techniques to build intelligent systems*, 2017. 15, 16, 18, 19, 20, 21, 22, 23, 24
- 29 SKANSI, S. *Introduction to Deep Learning: from logical calculus to artificial intelligence*. [S.l.]: Springer, 2018. 15
- 30 MAMMONE, A.; TURCHI, M.; CRISTIANINI, N. Support vector machines. *Wiley Interdisciplinary Reviews: Computational Statistics*, Wiley Online Library, v. 1, n. 3, p. 283–289, 2009. 20

- 31 SMOLA, A. J.; SCHÖLKOPF, B. A tutorial on support vector regression. *Statistics and computing*, Springer, v. 14, n. 3, p. 199–222, 2004. 20
- 32 THUKARAM, D.; KHINCHA, H.; VIJAYNARASIMHA, H. Artificial neural network and support vector machine approach for locating faults in radial distribution systems. *IEEE transactions on power delivery*, IEEE, v. 20, n. 2, p. 710–721, 2005. 20
- 33 SILVA, I. d. Redes neurais artificiais para engenharia e ciências aplicadas. 24
- 34 KLAMBAUER, G. et al. Self-normalizing neural networks. *Advances in neural information processing systems*, v. 30, 2017. 26
- 35 BAYDIN, A. G. et al. Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*, Microtome Publishing, v. 18, p. 1–43, 2018. 29
- 36 CHOLLET, F. *Deep learning with Python*. [S.l.]: Simon and Schuster, 2021. 29, 31
- 37 CHARU, C. A. *Neural networks and deep learning: a textbook*. [S.l.]: Springer, 2018. 29
- 38 KARPATY, A. 2015. Url<http://karpathy.github.io/2015/05/21/rnn-effectiveness/>. 29
- 39 KOLEN, J. F.; KREMER, S. C. *A field guide to dynamical recurrent networks*. [S.l.]: John Wiley & Sons, 2001. 31
- 40 OLAH, C. *Understanding LSTM Networks*. 2015. Disponível em: <<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>>. 32
- 41 GREFF, K. et al. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, IEEE, v. 28, n. 10, p. 2222–2232, 2016. 32
- 42 LAM, S. K.; PITROU, A.; SEIBERT, S. Numba: A llvm-based python jit compiler. In: *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*. [S.l.: s.n.], 2015. p. 1–6. 35
- 43 GOMES-FILHO, M. S.; PENNA, A. L.; OLIVEIRA, F. A. The kardar-parisi-zhang exponents for the 2+ 1 dimensions. *Results in Physics*, Elsevier, v. 26, p. 104435, 2021. 61
- 44 OLIVEIRA, T. J. Kardar-parisi-zhang universality class in (d+ 1)-dimensions. *Physical Review E*, APS, v. 106, n. 6, p. L062103, 2022. 61
- 45 EVANS, J.; THIEL, P.; BARTELT, M. C. Morphological evolution during epitaxial thin film growth: Formation of 2d islands and 3d mounds. *Surface science reports*, Elsevier, v. 61, n. 1-2, p. 1–128, 2006. 63