UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

ERICK HAWRYLUK DA SILVA

LARISSA PEREIRA DE LIMA

PREVISÃO DE FALHAS EM COMPONENTES AERONÁUTICOS: UM ESTUDO COMPARATIVO ENTRE REDES NEURAIS ARTIFICIAIS

PONTA GROSSA

2025

ERICK HAWRYLUK DA SILVA
LARISSA PEREIRA DE LIMA

PREVISÃO DE FALHAS EM COMPONENTES AERONÁUTICOS: UM ESTUDO COMPARATIVO ENTRE REDES NEURAIS ARTIFICIAIS

Failure prediction in aeronautical components: a comparison study among artificial neural networks

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia Elétrica do Curso de Bacharelado em Engenharia Elétrica da Universidade Tecnológica Federal do Paraná.

Orientador: Profª. Drª. Marcella Scoczynski Martins

PONTA GROSSA
2025

ERICK HAWRYLUK DA SILVA
LARISSA PEREIRA DE LIMA

PREVISAO DE FALHAS EM COMPONENTES AERONAUTICOS: UM ESTUDO COMPARATIVO ENTRE REDES NEURAIS ARTIFICIAIS

Trabalho de conclusão de curso de graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia Elétrica da Universidade Tecnológica Federal do Paraná (UTFPR).

Data de aprovação: 20/fevereiro/2025

_____
Marcella Scoczynski Ribeiro Martins
Titulação (Doutorado)
Universidade Tecnológica Federal do Paraná

_____
Cristhiane Goncalves
Titulação (Doutorado)
Universidade Tecnológica Federal do Paraná

_____
Edison Luiz Salgado Silva
Titulação (Mestrado)
Universidade Tecnológica Federal do Paraná

PONTA GROSSA
2025

RESUMO

Este estudo aborda a manutenção preditiva para prever falhas em motores de aeronaves, empregando os métodos de redes neurais como a *Recurrent Neural Network (RNN)*, *Bi-directional Recurrent Neural Networks (BRNN)*, *Long short-term memory (LSTM)*, *Extreme Gradient Boosting (XGBoost)* e *Gated Recurrent Unit (GRU)*. O trabalho é justificado pela necessidade crucial de manter motores em excelentes condições para assegurar a segurança dos passageiros e diminuir os gastos de manutenção. O objetivo fundamental é desenvolver modelos que, afundando-se no histórico de ciclos e dados sensoriais, possam prever se um motor falhará dentro de um ciclo específico, fazendo assim uma comparação entre métodos e avaliando qual é o melhor para essa aplicação. A metodologia abrange a importação de bibliotecas, o carregamento e pré-processamento do conjunto de dados, a geração da variável alvo de classificação, a normalização dos dados e análise exploratória dos dados (AED). Foram construídos e avaliados diversos modelos de RNN, incluindo RNN simples com uma e 25 características, além de um modelo bidirecional com 25 características. Posteriormente, os modelos LSTM, XGBOOST e GRU foram implementados. Os resultados mostram que os modelos de RNN mais complexos, com todas as 25 características, atingiram melhor precisão e métricas de avaliação comparados aos modelos mais simples. A conclusão ressalta a melhor precisão do modelo LSTMs para essa aplicação, sendo mais adequado na manutenção de aeronaves para evitar reparos dispendiosos ou substituições desnecessárias, contribuindo significativamente para a segurança e eficiência das operações de manutenção.

Palavras-chave: manutenção preditiva; redes neurais recorrentes; lstm; falhas em motores; comparação entre métodos.

ABSTRACT

This study addresses predictive maintenance to predict aircraft engine failures, employing neural network methods such as *Recurrent Neural Network (RNN)*, *Bidirectional Recurrent Neural Networks (BRNN)*, *Long short-term memory (LSTM)*, *Extreme Gradient Boosting (XGBoost)* and *Gated Recurrent Unit (GRU)*. The work is justified by the crucial need to keep engines running excellent conditions to ensure passenger safety and reduce maintenance expenses. The fundamental objective is to develop models that, By delving into cycle history and sensory data, they can predict whether a motor will fail within a specific cycle, thus making a comparison between methods and evaluating which is best for that application. The methodology covers importing libraries, loading and pre-processing the dataset, generating of the target variable for classification, data normalization and exploratory data analysis (EDA). Several RNN models were built and evaluated, including a simple RNN with one and 25 features, as well as a bidirectional model with 25 features. Subsequently, the LSTM, XGBOOST and GRU models were implemented. The results show that the more complex RNN models, with all 25 features, achieved better accuracy and evaluation metrics compared to the simpler models. The conclusion highlights the better accuracy of the LSTMs model for this application, being more suitable in aircraft maintenance to avoid costly repairs or unnecessary replacements, significantly contributing to the safety and efficiency of maintenance operations.

Keywords: predictive maintenance; recurrent neural networks; lstm; engine failures; comparison between methods.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTAGEM DE CÓDIGOS FONTE

LISTA DE ABREVIATURAS E SIGLAS

Abreviaturas

art.	Artigo
cap.	Capítulo
sec.	Seção

Siglas

ABNT	Associação Brasileira de Normas Técnicas
BRNN	*Bidirecional Recurrent Neural Network*
CAPES	Coordenação de Aperfeiçoamento de Pessoal de Nível Superior
CNPq	Conselho Nacional de Desenvolvimento Científico e Tecnológico
EPS	*Encapsulated PostScript*
GRU	*Gated Recurrent Unit*
HPC	Compressor de alta pressão
HPT	Turbina de alta pressão
IDs	Identificadores
LPC	Compressor de baixa pressão
LPT	Turbina de baixa pressão
LSTMs	*Long Short-Term Memory*
NASA	National Aeronautics and Space Administration
PDF	Formato de Documento Portátil, do inglês *Portable Document Format*
PS	*PostScript*
RNNs	*Recurrent Neural Network*
Sets	Conjuntos de dados utilizados no treinamento ou teste de redes neurais
SVM	Máquinas de Vetores de Suporte

UTFPR	Universidade Tecnológica Federal do Paraná
XGBoost	*Extreme Gradient Boosting*

LISTA DE SÍMBOLOS

Letras Latinas

A	Área	[m^2]
L	Comprimento	[m]
R	Raio	[m]
$T2$	Temperatura total na entrada do Fan	[F]
$T24$	Temperatura total na saída do LPC	[F]
$T30$	Temperatura total na saída do HPC	[F]
$T50$	Temperatura total na saída do LPT	[F]
$P2$	Pressão na entrada do Fan	[Psi]
$P15$	Pressão total no duto de derivação	[Psi]
$P30$	Pressão total na saída do HPC	[Psi]
Nf	Velocidade física do Fan	[RPM]
Nc	Velocidade física do núcleo	[RPM]
epr	Relação de pressão do motor (P50/P2)	[]
$Ps30$	Pressão estática na saída do HPC	[Psi]
phi	Proporção do fluxo de combustível para Ps30	[pps/Psi]
NRf	Velocidade do Fan corrigido	[RPM]
NRc	Velocidade do núcleo corrigido	[RPM]
BPR	Taxa de derivação	[]
$farB$	Relação combustível-ar do burner	[]
$htBleed$	Entalpia de Sangria	[]
Nf_dmd	Velocidade do Fan demandada	[RPM]
$PCNfR\_dmd$	Velocidade do fan demandada corrigida	[RPM]
$W31$	Sangria do líquido de resfriamento do HPT	[Lbm/s]

$W32$	Sangria do líquido de resfriamento do LPT	$[\mathrm{Lbm/s}]$
$TANGH$	Função de ativação tangente hiperbólica	[]

Letras Gregas

σ	Função sigmoide	[]

Subscritos

Z_t	Gate de atualização
r_t	Gate de redefinição
$\tilde{h_t}$	Novo estado oculto candidato
h_t	Estado oculto atual
W	Matriz de peso
x_t	Entrada atual
h_{t-1}	Estado oculto anterior

SUMÁRIO

1 INTRODUÇÃO

Para manter a segurança e eficácia das operações aéreas, a aviação comercial brasileira enfrenta desafios constantes, principalmente em relação à manutenção de aeronaves. Conforme dados do Centro de Investigação e Prevenção de Acidentes Aeronáuticos (CENIPA), o Brasil registrou um acidente aéreo a cada dois dias nos últimos dez anos, entre janeiro de 2012 e abril de 2023, foram registrados 1.878 acidentes com helicópteros, aviões, ultraleves, planadores, hidroaviões e trikes. Dessa forma, a manutenção aeronáutica foi reconhecida como um dos elementos contribuidores consideráveis.

1.1 Considerações iniciais

A aviação comercial brasileira representa um setor crucial para a conectividade nacional e internacional, desempenhando um papel fundamental no transporte de passageiros e carga. Contudo, a segurança operacional continua sendo uma prioridade constante, especialmente diante dos desafios complexos associados à manutenção de aeronaves.Isso sublinha a importância crítica de uma cultura de segurança robusta na aviação, onde práticas de manutenção preventiva desempenham um papel vital na prevenção de falhas mecânicas e incidentes como o acidente com a aeronave Cessna modelo 172 em 2016 e a aeronave modelo Bell 206B em 2019.

Dessa forma, este estudo usa os dados fornecidos pela *NASA (National Aeronautics and Space Administration) Open Data Portal* (NASA, 2025) para fazer o treinamento das Redes Neurais, se utilizando dos dados de treinamento, validação e de teste. Também é utilizado o artigo *Damage Propagation Modeling for Aircraft Engine Run-to-Failure Simulation* (SAXENA *et al.*, 2008) que descreve como a propagação de danos pode ser modelada dentro dos módulos de motores de turbina a gás de aeronaves, nas quais as superfícies de resposta de todos os sensores são geradas por meio de um modelo de simulação termodinâmica para o motor como uma função de variações de fluxo e eficiência dos módulos de interesse.

1.2 Objetivos

Este trabalho visa desenvolver e comparar modelos de redes neurais para a previsão de falhas em motores de turbina a gás de aeronaves.

1.2.1 Objetivo geral

Desenvolver modelos de redes neurais para previsão de falhas em motores de turbina a gás de aeronaves, utilizando dados disponibilizados pela *Nasa Open Portal Data*.

1.2.2 Objetivos específicos

Realizar uma revisão detalhada da literatura sobre técnicas de aprendizado de máquina, focando em redes neurais utilizadas.

Coletar e preparar dados, garantindo que os dados estejam prontos para serem analisados.

Implementar modelos de redes neurais, incluindo as *Recurrent Neural Network (RNN)* e redes de memória de longo prazo (LSTMs e GRU), utilizando a linguagem Python e bibliotecas especializadas.

Treinar e otimizar os modelos desenvolvidos utilizando dados históricos, visando maximizar a precisão na previsão de falhas e minimizar falsos positivos.

Avaliar o desempenho dos modelos através de métricas adequadas, como precisão, recall e F1-score, comparando com métodos.

1.3 Justificativa

Este trabalho se fundamenta na necessidade de melhorar a segurança e a eficiência na manutenção de aeronaves, por meio de aplicação de técnicas avançadas de aprendizado de máquina, especificamente redes neurais, para previsão e diagnóstico de falhas nos componentes das aeronaves, como os desenvolvidos por (PERINGAL; MOHIUDDIN; HASSAN, 2024) e (HASIB *et al.*, 2023).

Historicamente, a manutenção preventiva e corretiva de aeronaves é de extrema importância nas operações de transporte aéreo, afetando diretamente a segurança operacional e a confiabilidade das aeronaves. As redes neurais, especialmente as *Recurrent Neural Network (RNN)* e as *Long short-term memory (LSTM)*, trazem ferramentas promissoras para resolver problemas complexos de séries temporais e dados sequenciais, como os mostrados pelos trabalhos de (THAKKAR; CHAOUI, 2022) e (BOUJAMZA; ELHAQ, 2024), que utilizaram redes neurais para a predição de falhas em aeronaves.

Do ponto de vista teórico, este estudo pretende explorar metodologias de aprendizado de máquina, especialmente na implementação de redes neurais em Python. Python é notoriamente reconhecido como uma das linguagens mais eficientes e flexíveis para desenvolvimento de modelos de inteligência artificial, graças às suas vastas bibliotecas especializadas como TensorFlow, Keras e PyTorch, que simplificam a implementação e treinamento de redes neurais.

Este trabalho busca contribuir na implementação prática de modelos de redes neurais e comparação entre eles, para a previsão de falhas em componentes, utilizando dados simulados do motor a jato CMAPSS disponibilizados pela *Nasa Open Portal Data* (NASA, 2025).

1.4 Estrutura do trabalho

A estrutura do trabalho contém uma relação dos capítulos e uma descrição sucinta do que cada um deles contém. Esta seção fornece uma visão geral do trabalho no sentido da sua estrutura em capítulos.

1.4.1 Primeiro capítulo

No primeiro capítulo, será feita uma introdução ao tema do trabalho, trazendo uma retrospectiva histórica, a justificativa, a base teórica utilizada e os objetivos do estudo.

1.4.2 Segundo capítulo

No segundo capítulo, será apresentado o embasamento teórico do trabalho, detalhando a origem dos dados utilizados, uma introdução às redes neurais implementadas e a descrição das métricas de avaliação adotadas.

1.4.3 Terceiro capítulo

No terceiro capítulo, referente a materiais e métodos, serão descritas as ferramentas utilizadas no desenvolvimento do trabalho e a fonte dos dados de treinamento. Além disso, serão apresentados os métodos de redes neurais implementados.

1.4.4 Quarto capítulo

No quarto capítulo, serão expostos os resultados e gráficos obtidos para cada método implementado, com a devida discussão sobre o desempenho de cada um.

1.4.5 Quinto capítulo

Por fim, na conclusão, será realizada uma retrospectiva dos modelos empregados, destacando os métodos e métricas de avaliação utilizados, assim como a comparação entre o melhor e o pior desempenho. Além disso, serão sugeridas direções para estudos futuros, visando o aprimoramento e a expansão deste trabalho.

2 REFERENCIAL TEÓRICO

Nesta seção, será realizada uma revisão de literatura sobre técnicas de aprendizado de máquina aplicadas à manutenção preditiva de aeronaves, com foco específico em artigos que tragam um embasamento teórico para o desenvolvimento do trabalho.

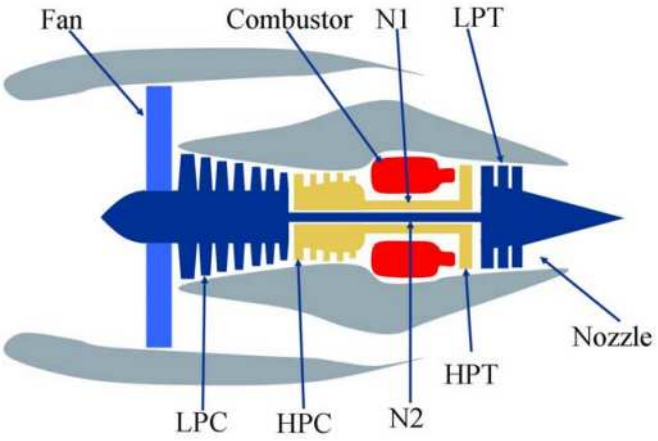
2.1 Fundamentos de Aprendizado de Máquina em Redes Neurais Aplicadas à Manutenção Preditiva

Os fundamentos teóricos do aprendizado de máquina são os conceitos essenciais, como classificação, regressão e clusterização, destacando técnicas como árvores de decisão, máquinas de vetores de suporte (SVM) e redes neurais artificiais. A ênfase será na compreensão de como essas técnicas são adaptadas para lidar com dados complexos e multidimensionais gerados por sistemas de monitoramento de aeronaves. Será concentrada a revisão nas redes neurais, que desempenham um papel importante para evolução da manutenção preditiva, sendo abordadas as principais arquiteturas, como *Recurrent Neural Network (RNNs)* e suas variantes avançadas como *Long Short-Term Memory (LSTM)*.

Para compreender como a propagação de danos pode ser modelada dentro dos módulos de motores de turbina a gás de aeronaves, foi utilizado o artigo *Damage Propagation Modeling for Aircraft Engine Run-to-Failure Simulation* (SAXENA *et al.*, 2008).

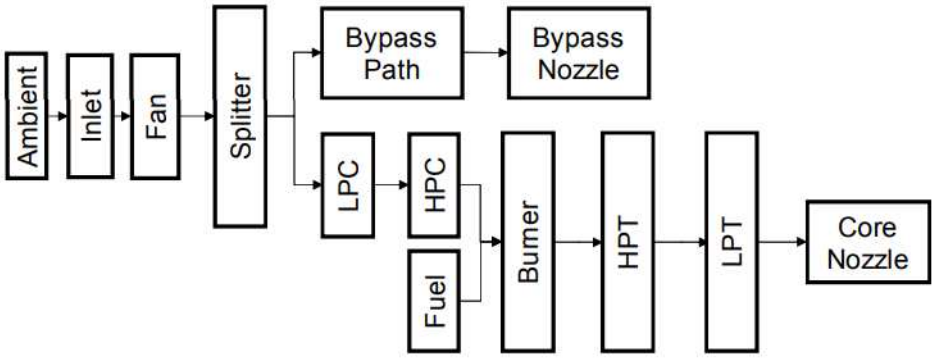
O diagrama da turbina da Figura 1 nos mostra os principais elementos do modelo da turbinas e o fluxograma da Figura 2 mostra como várias sub-rotinas são montadas na simulação.

Figura 1 – Turbina a gás



Fonte: Saxena *et al.* (2008).

Figura 2 – Layout da turbina



Fonte: Saxena *et al.* (2008).

A tabela 1 mostra as saídas que incluem várias superfícies de resposta do sensor. Tendo ao todo 21 variáveis de 58 saídas diferentes, disponíveis no modelo que foi usado no estudo de (SAXENA *et al.*, 2008).

Tabela 1 – Descrição das Variáveis e Unidades

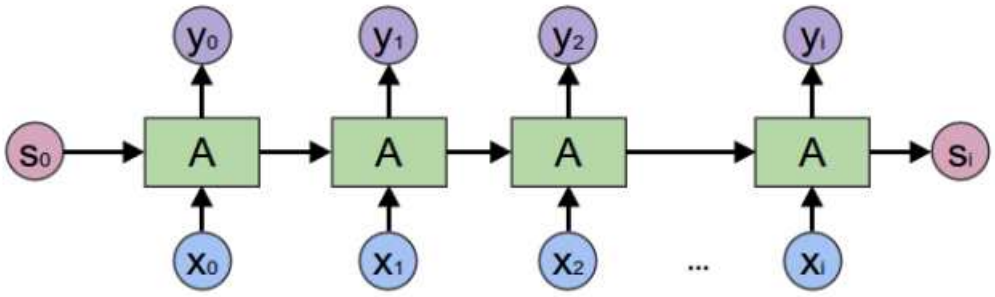
Símbolo	Descrição	Unidade
T2	Temperatura total na entrada do Fan	ºF
T24	Temperatura total na saída do LPC	ºF
T30	Temperatura total na saída do HPC	ºF
T50	Temperatura total na saída do LPT	ºF
P2	Pressão na entrada do Fan	Psi
P15	Pressão total no duto de derivação	Psi
P30	Pressão total na saída do HPC	Psi
Nf	Velocidade física do Fan	RPM
Nc	Velocidade física do núcleo	RPM
epr	Relação de pressão do motor (P50/P2)	–
Ps30	Pressão estática na saída do HPC	Psi
phi	Proporção do fluxo de combustível para Ps30	pps/Psi
NRf	Velocidade do Fan corrigido	rpm
NRc	Velocidade do núcleo corrigido	rpm
BPR	Taxa de derivação	–
farB	Relação combustível-ar do burner	–
htBleed	Entalpia de Sangria	–
Nf_dmd	Velocidade do fan demandada	rpm
PCNfR_dmd	Velocidade do fan demandada corrigida	rpm
W31	Sangria do líquido de arrefecimento do HPT	Lbm/s
W32	Sangria do líquido de arrefecimento do LPT	Lbm/s

Fonte: Adaptado de Saxena *et al.* (2008).

2.2 *Recurrent Neural Network*

Uma *Recurrent Neural Network (RNN)* é um tipo de rede neural projetada para trabalhar com dados sequenciais, como textos e séries temporais. As RNNs possuem conexões recorrentes que permitem que informações de estados anteriores influenciem o processamento atual, diferentemente das redes neurais tradicionais, que processam cada entrada de forma independente. Esse tipo de arquitetura é mais indicado para tarefas onde o contexto é essencial, como tradução automática e reconhecimento de fala. De acordo com (OLAH, 2025), o mecanismo que possibilita essa recorrência é um loop interno que alimenta a saída de um neurônio de volta como entrada para o próximo passo de tempo.

Figura 3 – Rede neural recorrente

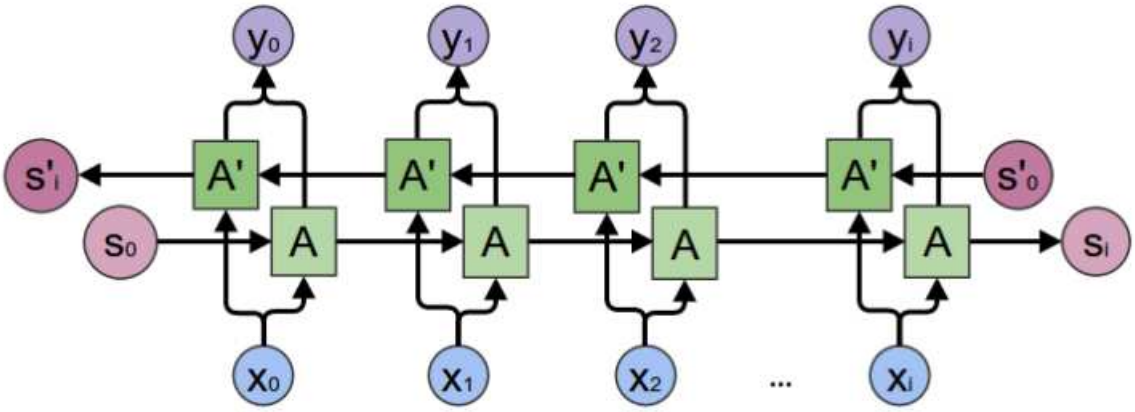


Fonte: Olah (2025).

2.3 *Bidirectional Recurrent Neural Networks*

Uma *Bidirectional Recurrent Neural Networks (BRNN)* é uma derivação das RNNs tradicionais que processa os dados em duas direções. Nas BRNN, a camada de saída recebe informações das duas camadas ocultas, permitindo acessar o contexto em ambas as direções (WöLLMER *et al.*, 2013). Isso é feito pelo uso de duas camadas recorrentes independentes, uma para cada direção, sendo combinadas na saída para produzir uma representação mais rica do contexto. Esse mecanismo é especialmente útil em tarefas onde o significado de um elemento depende tanto do que veio antes quanto do que vem depois.

Figura 4 – Rede neural recorrente bidirecional

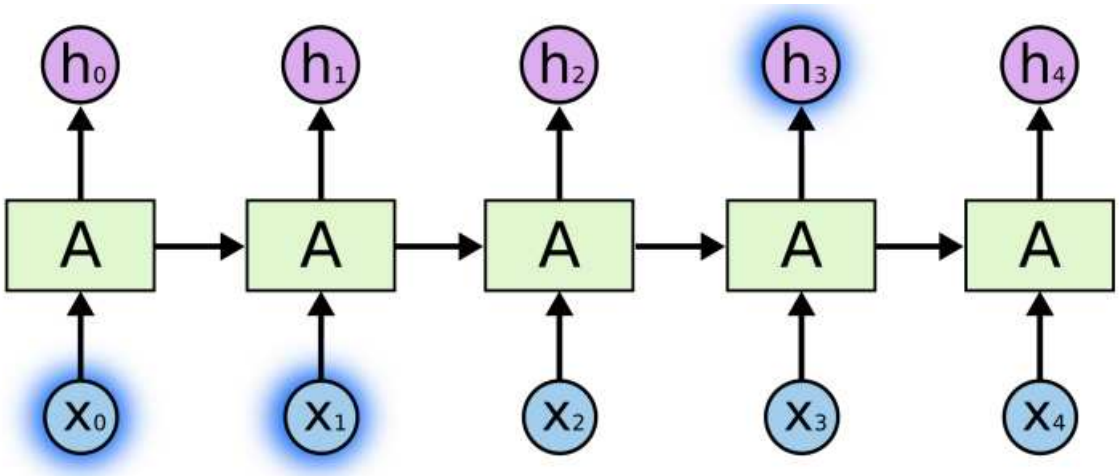


Fonte: Olah (2025).

2.4 *Long short-term memory (LSTM)*

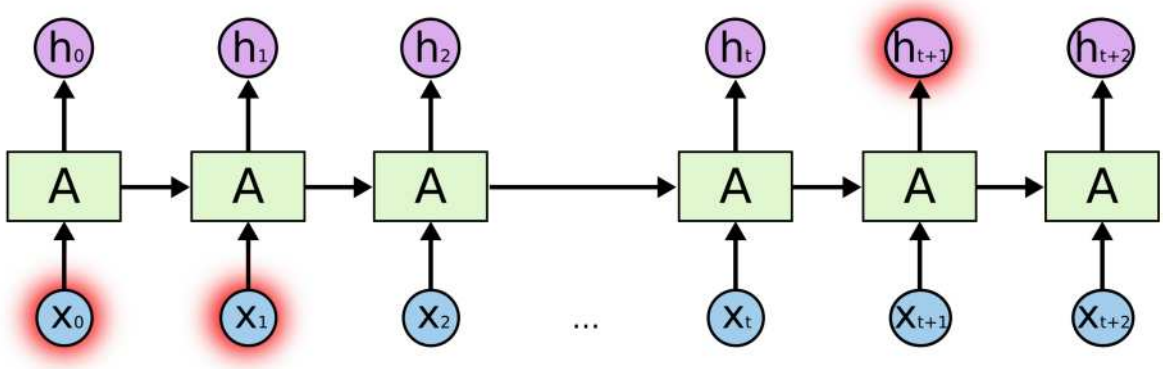
O problema do gradiente desaparecido *(vanishing gradient)* ocorre quando os componentes de longo prazo tendem rapidamente a um valor próximo de zero, tornando difícil para o modelo aprender correlações entre eventos que estão distantes no tempo (PASCANU; MIKOLOV; BENGIO, 2013). Isso faz com que as camadas iniciais da rede aprendam muito lentamente ou praticamente parem de aprender. Esse fenômeno traz ploblemas e dificulta o aprendizado de dependências de longo prazo *(long-term dependencies)*, que são relações entre elementos de uma sequência que estão distantes um do outro. Como isso as RNNs simples possuem problemas para capturar informações relevantes em séries temporais longas ou textos complexos, limitando o seu desempenho em algumas aplicações.

Figura 5 – Vanishing Gradient



Fonte: Olah (2025).

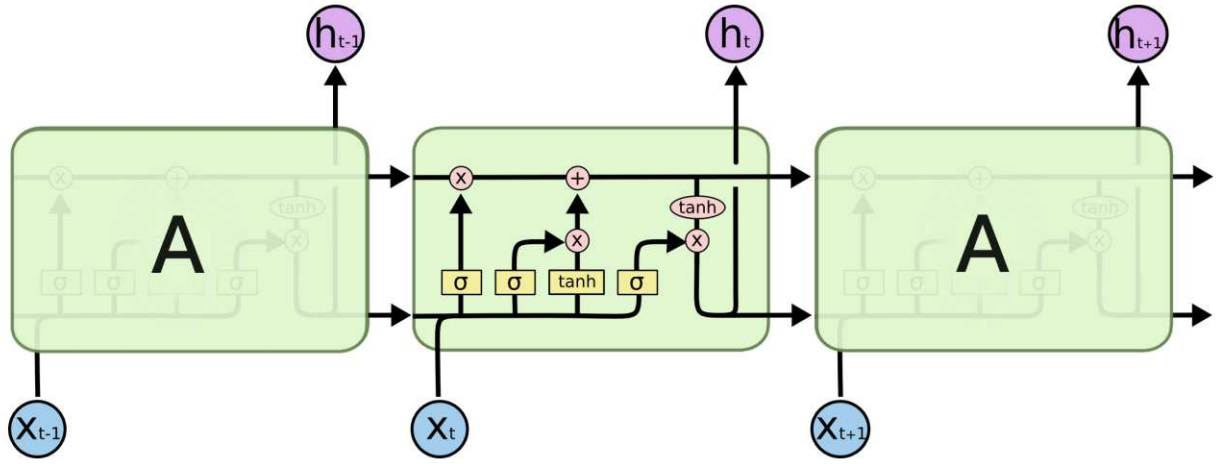
Figura 6 – Long-term dependencies



Fonte: Olah (2025).

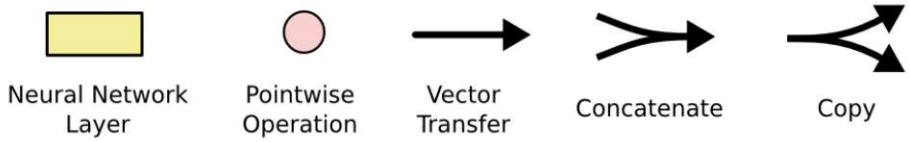
Para solucionar esse problema, foram desenvolvidas arquiteturas como as *Long Short-Term Memory (LSTM)*. As redes LSTM podem armazenar informações em células de memória por mais tempo e ajustar a quantidade de contexto necessária para melhorar o desempenho em tarefas de regressão ou classificação (WöLLMER *et al.*, 2013). Cada célula LSTM possui três portas de controle: porta de entrada, que decide quais novas informações devem ser armazenadas; porta de esquecimento, que determina quais informações anteriores devem ser descartadas; e porta de saída, que regula quais informações serão usadas para calcular a saída. Essa estrutura mantém o fluxo do gradiente estável durante o treinamento, permitindo que a rede aprenda relações de longo prazo sem sofrer com o desaparecimento do gradiente.

Figura 7 – Módulo na lstm



Fonte: Olah (2025).

Figura 8 – Operações



Fonte: Olah (2025).

2.5 *Extreme Gradient Boosting (XGBoost)*

O *Extreme Gradient Boosting* é um algoritmo baseado em árvores de decisão que se destaca por sua eficiência e alto desempenho em diversas aplicações. Diferente das RNNs e das LSTMs, que são modelos sequenciais projetados para capturar dependências temporais, o XGBoost é da classe dos modelos de *boosting*, onde múltiplas árvores de decisão são treinadas de forma sequencial para corrigir os erros das árvores anteriores.

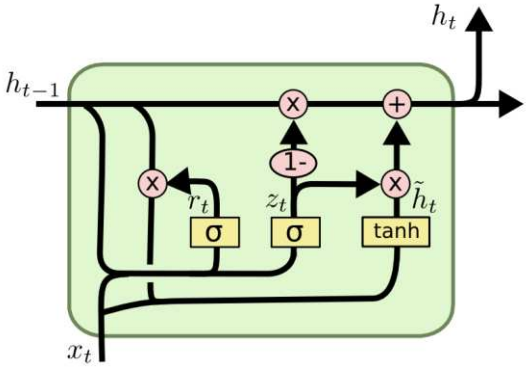
O mecanismo central do XGBoost é o *Gradient Boosting*, que constrói árvores de forma iterativa, minimizando o erro predefinido, como o erro quadrático para regressão ou a entropia para classificação. O uso de computação paralela e distribuída acelera o processo de aprendizado, permitindo uma exploração mais rápida dos modelos e melhorando seu desempenho em grandes volumes de dados (CHEN; GUESTRIN, 2016).

2.6 *Gated Recurrent Unit (GRU)*

O *Gated Recurrent Units* é uma variação das LSTMs introduzido por (CHO *et al.*, 2014) que simplifica a estrutura dos mecanismos de controle, mantendo sua capacidade de capturar dependências de longo prazo e diminuindo o problema do desaparecimento do gradiente. Diferente das LSTMs, que possuem três portas de controle (entrada, esquecimento e saída), o GRU utiliza apenas duas portas principais: a porta de atualização, que controla a quantidade de nova informação a ser armazenada, e a porta de *reset*, que decide quanto da informação anterior deve ser esquecida.

Essa simplificação diminui o número de parâmetros e, consequentemente, torna o treinamento mais rápido em comparação com as LSTMs, sem comprometer significativamente o desempenho em muitas tarefas. Além disso, como as GRUs não possuem uma célula de memória separada, a informação é armazenada diretamente no estado oculto, o que pode facilitar o aprendizado em alguns contextos onde há restrições de tempo de treinamento ou quando um modelo mais simples é desejado.

Figura 9 – Gated recurrent unit



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Fonte: Olah (2025).

2.7 Conceitos de treinamento, validação e teste

2.7.1 Treinamento

O treinamento é a fase em que a rede neural recorrente faz o ajuste de seus pesos e parâmetros para aprender padrões a partir dos dados fornecidos. Isso ocorre através da retro-propagação do erro e do otimizador. Durante essa etapa, a rede recebe entradas e ajusta seus pesos com base na minimização de uma função de perda, como o erro quadrático médio (MSE) ou a entropia cruzada.

2.7.2 Validação

A validação é usada para monitorar o desempenho do modelo durante o treinamento. Um conjunto de dados separado, chamado de conjunto de validação, é usado para avaliar como a rede generaliza para dados que não foram vistos durante o ajuste dos pesos. O objetivo é evitar problemas como o *overfitting*. Durante o treinamento, métricas como acurácia, perda e erro são avaliadas periodicamente no conjunto de validação, e ajustes como regularização e parâmetros da rede podem ser feitos para melhorar o desempenho.

2.7.3 Teste

A fase de teste ocorre após o fim do treinamento, onde a rede é avaliada com um novo conjunto de dados (conjunto de teste) que não foi utilizado nem no treinamento nem na validação. O objetivo é medir a capacidade real de generalização da rede para dados nunca antes vistos. Esse processo fornece métricas finais, como precisão, recall, F1-score.

2.8 Interpretação dos resultados

2.8.1 Precisão

Mede a proporção de predições positivas que estão corretas.
Fórmula:

$$\text{Precisão} = \frac{\text{Verdadeiros Positivos (VP)}}{\text{Verdadeiros Positivos (VP)} + \text{Falsos Positivos (FP)}} \tag{1}$$

Interpretação: Alta precisão significa que o modelo comete poucos erros ao classificar exemplos positivos.

2.8.2 *Recall*

Mede a proporção de exemplos positivos que foram identificados corretamente pelo modelo.

Fórmula:

$$\text{Recall} = \frac{\text{Verdadeiros Positivos (VP)}}{\text{Verdadeiros Positivos (VP)} + \text{Falsos Positivos (FP)}} \tag{2}$$

Interpretação: Alto *recall* significa que o modelo consegue identificar a maioria dos exemplos positivos.

2.8.3 *F1-Score*

Média harmônica entre Precisão e *Recall*, oferecendo um equilíbrio entre ambos.

Fórmula:

$$F1\text{-}Score = 2 \cdot \frac{\text{Precisao} \cdot \text{Recall}}{\text{Precisao} + \text{Recall}} \tag{3}$$

Interpretação: É útil quando há um *trade-off* entre precisão e *recall*, sendo mais relevante em cenários de classes desbalanceadas.

2.9 Capacidade de Captura de Contexto de Longo Prazo

2.9.1 RNN Simples

Tem dificuldade em capturar dependências de longo prazo devido ao problema do gradiente que desaparece ou explode, como apresentado na figura 6.

2.9.2 RNN com 25 características

Aumenta a capacidade de capturar dependências de médio prazo, pois possui mais recursos para aprender representações mais complexas.

2.9.3 BRNN

Melhora a compreensão do contexto ao processar a sequência de entrada tanto no sentido direto quanto reverso, capturando dependências em ambas as direções.

2.9.4 LSTM

É projetado especificamente para lidar com dependências de longo prazo, usando unidades de memória que podem armazenar informações ao longo de toda a sequência.

3 MATERIAIS E MÉTODOS

Para o desenvolvimento do trabalho iniciaremos com as ferramentas utilizadas, posteriormente a implementação dos métodos de redes neurais.

3.1 Materiais

Para a implementação dos métodos de redes neurais, foi utilizado um notebook em Python na plataforma Kaggle, que permitiu a execução dos métodos. O ambiente do Kaggle foi escolhido devido à sua compatibilidade com bibliotecas de aprendizado de máquina.

O presente trabalho utilizou como uma de suas bases o notebook desenvolvido disponível no repositório do GitHub, que aborda conceitos fundamentais de aprendizado profundo aplicados à manutenção preditiva (HUNT, 2023). Este estudo usa os dados fornecidos pela *NASA (National Aeronautics and Space Administration) Open Data Portal* (NASA, 2025) para fazer o treinamento das Redes Neurais, utilizando-se dos dados de treinamento, validação e de teste. Também é utilizado o artigo *Damage Propagation Modeling for Aircraft Engine Run-to-Failure Simulation* (SAXENA *et al.*, 2008) que descreve como a propagação de danos pode ser modelada dentro dos módulos de motores de turbina a gás de aeronaves.

3.2 Métodos

Para a implementação dos métodos de redes neurais recorrentes, o desenvolvimento foi separado por etapas, sendo elas: *import* das bibliotecas, carregamento dos dados, tratamento dos dados, representação dos dados, geração da sequência de *imput* e a implementação dos métodos de RNN.

3.2.1 Importar as bibliotecas

Listagem 1 – Import das bibiotecas

```
1  from tensorflow import keras
2
3  import pandas as pd
4
5  import numpy as np
6
7  import matplotlib.pyplot as plt
8
9  import os
10
11 import seaborn as sns
12
13 from sklearn import preprocessing
14
15 from sklearn.metrics import confusion_matrix, recall_score, precision_score
16
17 from tensorflow.keras.models import Sequential,load_model
18
19 from tensorflow.keras.layers import Dense, Dropout, SimpleRNN, LSTM, GRU
20
21 import xgboost as xgb
22 from sklearn.metrics import accuracy_score, precision_score, recall_score,
23 f1_score, roc_auc_score, confusion_matrix
24
25 import matplotlib.pyplot as plt
26 from sklearn.model_selection import train_test_split
27
28 np.random.seed(1234)
29
30 PYTHONHASHSEED = 0
```

Fonte: Autoria própria (2025).

3.2.2 Carregar o conjunto de dados

Listagem 2 – Carregar dados

```
# Dados de treinamento onde o ultimo ciclo e o ponto de falha dos motores
train_df = pd.read_csv('/kaggle/input/pred-maintanance-data/PM_train.txt',
sep=" ", header=None)

# Dados de teste onde o ponto de falha nao e fornecido para os motores
test_df = pd.read_csv('/kaggle/input/pred-maintanance-data/PM_test.txt',
sep=" ", header=None)

train_df.head(100)
```

Fonte: Autoria própria (2025).

Figura 10 – Dados carregados

	0	1	2	3	4	5	6	7	8	9	...	18	19	20	21	22	23	24	25	26	27
0	1	1	-0.0007	-0.0004	100.0	518.67	641.82	1589.70	1400.60	14.62	...	8138.62	8.4195	0.03	392	2388	100.0	39.06	23.4190	NaN	NaN
1	1	2	0.0019	-0.0003	100.0	518.67	642.15	1591.82	1403.14	14.62	...	8131.49	8.4318	0.03	392	2388	100.0	39.00	23.4236	NaN	NaN
2	1	3	-0.0043	0.0003	100.0	518.67	642.35	1587.99	1404.20	14.62	...	8133.23	8.4178	0.03	390	2388	100.0	38.95	23.3442	NaN	NaN
3	1	4	0.0007	0.0000	100.0	518.67	642.35	1582.79	1401.87	14.62	...	8133.83	8.3682	0.03	392	2388	100.0	38.88	23.3739	NaN	NaN
4	1	5	-0.0019	-0.0002	100.0	518.67	642.37	1582.85	1406.22	14.62	...	8133.80	8.4294	0.03	393	2388	100.0	38.90	23.4044	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
95	1	96	-0.0034	0.0001	100.0	518.67	642.19	1584.07	1395.16	14.62	...	8130.69	8.4311	0.03	392	2388	100.0	38.88	23.3255	NaN	NaN
96	1	97	0.0035	-0.0003	100.0	518.67	642.07	1595.77	1407.81	14.62	...	8128.74	8.4105	0.03	392	2388	100.0	39.01	23.2963	NaN	NaN
97	1	98	0.0006	0.0004	100.0	518.67	642.00	1591.11	1404.56	14.62	...	8127.89	8.4012	0.03	391	2388	100.0	38.96	23.2554	NaN	NaN
98	1	99	-0.0005	-0.0000	100.0	518.67	642.46	1592.73	1406.13	14.62	...	8131.77	8.4481	0.03	393	2388	100.0	38.82	23.2323	NaN	NaN
99	1	100	-0.0021	-0.0003	100.0	518.67	642.22	1589.63	1411.35	14.62	...	8132.49	8.4241	0.03	392	2388	100.0	38.93	23.4090	NaN	NaN

100 rows × 28 columns

Fonte: Autoria própria (2025).

Listagem 3 – Ajuste do dados

```
train_df.dropna(axis=1, inplace=True)

test_df.dropna(axis=1, inplace=True)

print(len(train_df))

print(len(test_df))

cols_names = ['id', 'cycle', 'setting1', 'setting2', 'setting3', 's1', 's2',
's3','s4', 's5', 's6', 's7', 's8', 's9', 's10', 's11', 's12', 's13', 's14',
's15', 's16', 's17', 's18', 's19', 's20', 's21']

train_df.columns = cols_names

test_df.columns = cols_names

train_df.head(2)
```

Fonte: Autoria própria (2025).

Figura 11 – Data frame

	id	cycle	setting1	setting2	setting3	s1	s2	s3	s4	s5	...	s12	s13	s14	s15	s16	s17	s18	s19	s20	s21
0	1	1	0.0023	0.0003	100.0	518.67	643.02	1585.29	1398.21	14.62	...	521.72	2388.03	8125.55	8.4052	0.03	392	2388	100.0	38.86	23.3735
1	1	2	-0.0027	-0.0003	100.0	518.67	641.71	1588.45	1395.42	14.62	...	522.16	2388.06	8139.62	8.3803	0.03	393	2388	100.0	39.02	23.3916

2 rows × 26 columns

Fonte: Autoria própria (2025).

Listagem 4 – Dados de referência

```
# Carrega os dados verdadeiros de teste, que indica os ciclos uteis restantes
para os motores

truth_df = pd.read_csv('/kaggle/input/pred-maintanance-data/PM_truth.txt',
sep=" ", header=None)

# Remove a coluna NAN
truth_df.dropna(axis=1, inplace=True)

truth_df.head()
```

Fonte: Autoria própria (2025).

Figura 12 – Primeiras linhas dos dados de referência

	0
0	112
1	98
2	69
3	82
4	91

Fonte: Autoria própria (2025).

3.2.3 Tratamento dos dados

A normalização dos dados é de extrema importância no tratamento de dados, especialmente em aprendizado de máquina. O objetivo da normalização é ajustar os valores das variáveis para que todos estejam dentro de um intervalo comum, geralmente entre 0 e 1. Isso é feito para que todas as variáveis contribuam de maneira equitativa para os modelos de aprendizado, evitando que variáveis com maiores magnitudes dominem o processo de otimização. A técnica *MinMaxScaler* é uma abordagem comum para a tarefa, onde os valores são escalados com base nos valores mínimos e máximos das variáveis.

A organização dos dados em matrizes é necessária para o uso em modelos de *Recurrent Neural Network (RNN)*, que são especialmente eficazes para dados sequenciais. Para isso, os dados são organizados em sequências de um número fixo de passos de tempo, onde cada sequência é uma sub-matriz que representa uma série temporal de características. A definição do comprimento da sequência é uma decisão importante e afeta significativamente o desempenho do modelo. Essa transformação permite que o modelo capture padrões temporais e relacionamentos entre os dados de diferentes momentos no tempo.

Por fim, a combinação da normalização e da organização dos dados em matrizes prepara os dados para o aprendizado de máquina. A normalização garante que as variáveis sejam processadas de forma homogênea, enquanto a organização em sequências permite que os modelos de RNN explorem as dependências temporais nos dados. Esse processo de preparação é fundamental para a precisão e a eficiência de modelos preditivos, tornando-os capazes de fazer previsões mais confiáveis e robustas em aplicações práticas.

Listagem 5 – Gerando a variável alvo

```
# Especifica quais colunas devem ser consideradas durante a classificacao
usando o parametro.
train_df.sort_values(['id','cycle'], inplace=True)

test_df.sort_values(['id','cycle'], inplace=True)

# Extrai o numero maximo de ciclos para cada ID de mecanismo.
rul = pd.DataFrame(train_df.groupby('id')['cycle'].max()).reset_index()

# Nomeia as colunas e mescla com os dados de treinamento.
rul.columns = ['id', 'max']

train_df = train_df.merge(rul, on=['id'], how='left')

# Subtraia o ciclo atual do maximo (numero maximo de ciclos) para calcular a
vida util restante.
train_df['RUL'] = train_df['max'] - train_df['cycle']

#O RUL e calculado para cada id.
train_df[['id','cycle','max','RUL']].head()

# Retirando o "max".
train_df.drop('max', axis=1, inplace=True)

# Cria uma etiqueta que indica se um motor ira falhar dentro de w1 ciclos.
w1 = 30
train_df['failure_within_w1'] = np.where(train_df['RUL'] <= w1, 1, 0 )
```

Fonte: Autoria própria (2025).

Listagem 6 – Normalizando o dataset de treinamento

```
# Cria um recurso separado para o valor normalizado da coluna do ciclo.
train_df['cycle_norm'] = train_df['cycle']

# Usando a funcao de diferenca, exclui essas colunas do processo de
normalizacao.
cols_normalize = train_df.columns.difference(['id','cycle','RUL',
'failure_within_w1'])

# MinMax normalizacao (0 ate 1) dos dados do sensor
min_max_scaler = preprocessing.MinMaxScaler()

norm_train_df = pd.DataFrame(min_max_scaler.fit_transform(
train_df[cols_normalize]),
                             columns=cols_normalize,
                             index=train_df.index)

# Junta os dados normalizados e nao normalizados.
join_df = train_df[['id','cycle','RUL','failure_within_w1']].join(
norm_train_df)

train_df = join_df.reindex(columns = train_df.columns)

train_df.head()
```

Fonte: Autoria própria (2025).

Figura 13 – Dados de treinamento normalizados

	id	cycle	setting1	setting2	setting3	s1	s2	s3	s4	s5	...	s15	s16	s17	s18	s19	s20	s21	RUL	failure_within_w1	cycle_norm
0	1	1	0.459770	0.166667	0.0	0.0	0.183735	0.406802	0.309757	0.0	...	0.363986	0.0	0.333333	0.0	0.0	0.713178	0.724662	191	0	0.00000
1	1	2	0.609195	0.250000	0.0	0.0	0.283133	0.453019	0.352633	0.0	...	0.411312	0.0	0.333333	0.0	0.0	0.666667	0.731014	190	0	0.00277
2	1	3	0.252874	0.750000	0.0	0.0	0.343373	0.369523	0.370527	0.0	...	0.357445	0.0	0.166667	0.0	0.0	0.627907	0.621375	189	0	0.00554
3	1	4	0.540230	0.500000	0.0	0.0	0.343373	0.256159	0.331195	0.0	...	0.166603	0.0	0.333333	0.0	0.0	0.573643	0.662386	188	0	0.00831
4	1	5	0.390805	0.333333	0.0	0.0	0.349398	0.257467	0.404625	0.0	...	0.402078	0.0	0.416667	0.0	0.0	0.589147	0.704502	187	0	0.01108

5 rows × 29 columns

Fonte: Autoria própria (2025).

Listagem 7 – Normalizando o dataset de teste

```
# Normalizacao MinMax (de 0 a 1)
test_df['cycle_norm'] = test_df['cycle']

# Normalizacao MinMax (0 a 1) apenas dos dados do sensor
norm_test_df = pd.DataFrame(min_max_scaler.transform(test_df[cols_normalize]),
                            columns=cols_normalize,
                            index=test_df.index)

# Junta os dados normalizados e nao normalizados. (nao contem 'RUL' e
'failure_within_w1')
test_join_df = test_df[test_df.columns.difference(cols_normalize)].join(
norm_test_df)

test_df = test_join_df.reindex(columns = test_df.columns)

test_df = test_df.reset_index(drop=True)

# Vamos calcular o RUL total somando os ciclos maximos fornecidos no conjunto
de teste e o RUL adicional da verdade basica.
rul = pd.DataFrame(test_df.groupby('id')['cycle'].max()).reset_index()

rul.columns = ['id', 'max']

truth_df.columns = ['additional_rul']

# O respectivo id pode ser obtido adicionando 1 ao indice, ja que o indice
comeca em 0.
truth_df['id'] = truth_df.index + 1

# Adiciona os ciclos maximos fornecidos no conjunto de teste e a RUL
adicional fornecida no DataFrame de verdade basica.
truth_df['max'] = rul['max'] + truth_df['additional_rul']

truth_df.drop('additional_rul', axis=1, inplace=True)

# Gera o RUL para os dados de teste.
test_df = test_df.merge(truth_df, on=['id'], how='left')

test_df['RUL'] = test_df['max'] - test_df['cycle']

test_df.drop('max', axis=1, inplace=True)

# Gera as colunas de rotulo w0 e w1 para os dados de teste.
test_df['failure_within_w1'] = np.where(test_df['RUL'] <= w1, 1, 0 )

test_df.head()
```

Fonte: Autoria própria (2025).

Figura 14 – Dados de teste normalizados

	id	cycle	setting1	setting2	setting3	s1	s2	s3	s4	s5	...	s15	s16	s17	s18	s19	s20	s21	cycle_norm	RUL	failure_within_w1
0	1	1	0.632184	0.750000	0.0	0.0	0.545181	0.310661	0.269413	0.0	...	0.308965	0.0	0.333333	0.0	0.0	0.558140	0.661834	0.00000	142	0
1	1	2	0.344828	0.250000	0.0	0.0	0.150602	0.379551	0.222316	0.0	...	0.213159	0.0	0.416667	0.0	0.0	0.682171	0.686827	0.00277	141	0
2	1	3	0.517241	0.583333	0.0	0.0	0.376506	0.346632	0.322248	0.0	...	0.458638	0.0	0.416667	0.0	0.0	0.728682	0.721348	0.00554	140	0
3	1	4	0.741379	0.500000	0.0	0.0	0.370482	0.285154	0.408001	0.0	...	0.257022	0.0	0.250000	0.0	0.0	0.666667	0.662110	0.00831	139	0
4	1	5	0.580460	0.500000	0.0	0.0	0.391566	0.352082	0.332039	0.0	...	0.300885	0.0	0.166667	0.0	0.0	0.658915	0.716377	0.01108	138	0

5 rows × 29 columns

Fonte: Autoria própria (2025).

3.2.4 Representação dos dados

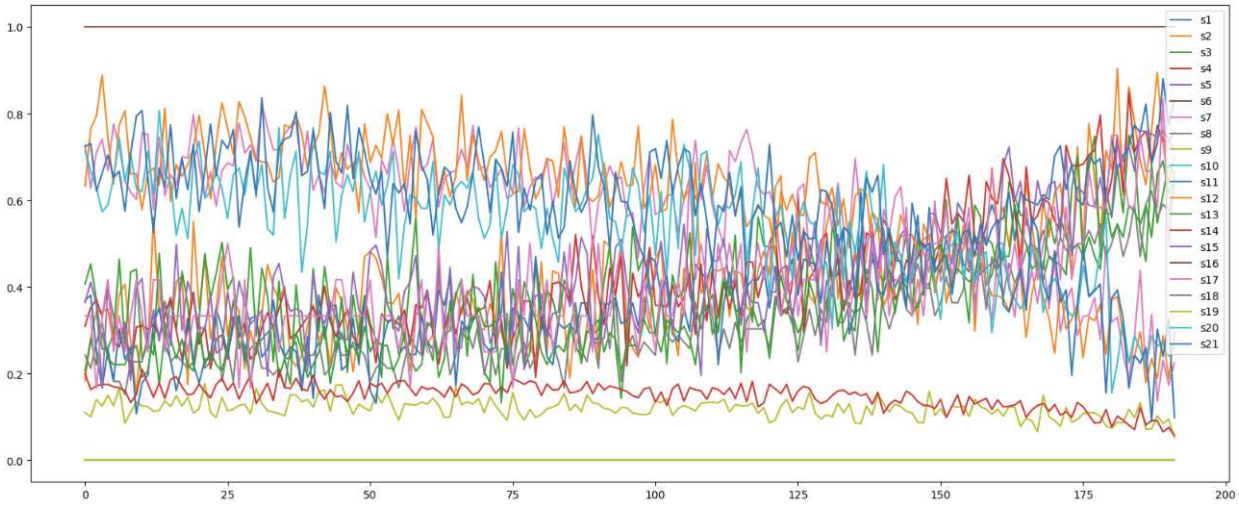
Na representação dos dados podemos ver como eles se comportam ao longo do tempo, assim, podemos ter uma melhor ideia de como implementar o métodos de redes neurais recorrentes.

Listagem 8 – Visualização para a turbina com id=1

```
1 sensor_cols = cols_names[5:]
2
3 train_df[train_df.id==1][sensor_cols].plot(figsize=(20, 8))
```

Fonte: Autoria própria (2025).

Figura 15 – Dados de teste normalizados



Fonte: Autoria própria (2025).

3.2.5 Gerando a sequência de *input*

Essa parte tem como objetivo preparar os dados para treinar os modelos de redes neurais, estruturando-os em sequências temporais para análise preditiva. Para isso, ele gera sequências de entrada e rótulos de saída a partir de um conjunto de dados contendo medições de sensores associadas a diferentes IDs de mecanismo. Esse processo é essencial para permitir que um modelo de aprendizado de máquina, como uma *Recurrent Neural Network (RNN)*,

LSTM ou GRU, possa aprender padrões temporais nos dados de sensores e prever falhas futuras nos mecanismos analisados.

Listagem 9 – Gerando a sequência de imput

```
# O comprimento da sequencia a ser usado para previsoes sequence_length = 50

# Funcao para gerar sequencias (amostras, intervalos de tempo, recursos) para um ID de mecanismo especifico

def sequence_generator(feature_df, seq_length, seq_cols):
    feature_array = feature_df[seq_cols].values
    num_elements = feature_array.shape[0]

    # Queremos gerar sequencias de 50 passos de tempo por vez.
    # Portanto, iteraremos sobre dois conjuntos de indices: (0,142),(50,192).
    # Por exemplo, id1 tem 192 linhas e seu sequence_length e igual a 50.
    # 0 50 -> Da linha 0 ate 50
    # 1 51 -> Da linha 1 ate 51
    # 2 52 -> Da linha 2 ate 52
    # ...
    # 141 191 -> Da linha 111 ate 191
    # Loop para gerar sequencias a partir da matriz de caracteristicas
    for start, stop in zip(range(0, num_elements-seq_length), range(seq_length, num_elements)):
        yield feature_array[start:stop,]

# Gera sequencias para cada ID de mecanismo.
# unique() retorna todos os IDs exclusivos em uma lista.

seq_gen = (list(sequence_generator(train_df[train_df['id']==id], sequence_length, ["s2"])) # Pegamos apenas a caracteristica (s2).
           for id in train_df['id'].unique())

# Concatena as sequencias de diferentes IDs de mecanismos em uma matriz e converte em uma matriz NumPy
# [(142, 50, 25), ...] --> [(ntotal, 50, 25)].
#print(list(seq_gen))
seq_set = np.concatenate(list(seq_gen)).astype(np.float32)

seq_set.shape
# Funcao para gerar rotulos:
def label_generator(label_df, seq_length, label):
    # Essa funcao ira retornar:
    # [[1]
    # [4]
    # [1]
    # [5]
```

Fonte: Autoria própria (2025).

Listagem 10 – Gerando a sequência de input

```
# [9]
    # ...
    # [200]]
    # Converte os rotulos em uma matriz NumPy 2D.

    label_array = label_df[label].values
    num_elements = label_array.shape[0]

    # Remova o primeiro rotulo, pois a primeira previsao sera o rotulo (
    seq_length+1).
    return label_array[seq_length:num_elements, :]

# Gera rotulos [[142,1], [121, 1], [3]] --> [1, 2, 3] (ntotal, 1).
label_gen = [label_generator(train_df[train_df['id']==id], sequence_length,
['failure_within_w1'])
             for id in train_df['id'].unique()]

label_set = np.concatenate(label_gen).astype(np.float32)

label_set.shape
```

Fonte: Autoria própria (2025).

3.2.6 Primeiro método RNN

Para a implementação do primeiro método, vamos separar o código em etapas, primeiro definindo os parâmetros de treinamento e obtendo um sumário do modelo. Após isso, iniciamos o treinamento e criamos as funções para plotar os gráficos de precisão, perda do modelo e comparação do valor previsto vs valor real. Por fim, é feito a avaliação dos *sets* de treinamento e teste, obtendo assim a performance do método aplicado.

Listagem 11 – Parâmetros de treinamento

```
out_dim = label_set.shape[1] # 1 rotulo/saida para uma sequencia.

features_dim = seq_set.shape[2] # Numero de caracteristicas (1)

print("Features dimension: ", features_dim)
print("Output dimension: ", out_dim)

RNN_fwd = Sequential()

# Adiciona a unidade RNN.
RNN_fwd.add(SimpleRNN(
          input_shape=(sequence_length, features_dim),
          units=1,
          return_sequences=False))

RNN_fwd.add(Dropout(0.2))

RNN_fwd.add(Dense(units=out_dim, activation='sigmoid'))

# Compilar o modelo.
RNN_fwd.compile(loss='binary_crossentropy', optimizer='adam', metrics=[
'accuracy'])

print(RNN_fwd.summary())

# Define o caminho para salvar o modelo.
RNN_fwd_path = '/kaggle/working/RNN_fwd.h5'
```

Fonte: Autoria própria (2025).

Figura 16 – Saída do modelo

```
Features dimension:  1
Output dimension:  1
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn (SimpleRNN)      (None, 1)                 3

 dropout (Dropout)           (None, 1)                 0

 dense (Dense)               (None, 1)                 2

=================================================================
Total params: 5
Trainable params: 5
Non-trainable params: 0
_________________________________________________________________
```

Fonte: Autoria própria (2025).

Listagem 12 – Treinamento

```
import time

epochs = 300

batch_size = 200

start = time.time()

# Ajuste da rede
RNN_fwd_history = RNN_fwd.fit(seq_set, label_set, epochs=epochs,
batch_size=batch_size, validation_split=0.05, verbose=2,
          callbacks = [keras.callbacks.EarlyStopping(monitor='val_loss',
          min_delta=0, patience=10, verbose=0, mode='min'),
                       keras.callbacks.ModelCheckpoint(RNN_fwd_path,monitor=
                       'val_loss', save_best_only=True, mode='min',
                       verbose=0)]
          )

end = time.time()

print("Total time taken for training:","{:.2f}".format((end-start))," secs")
```

Fonte: Autoria própria (2025).

Listagem 13 – Função para plotar o gráfico de precisão

```
#  Funcao para tracar a mudanca na precisao do modelo nos conjuntos de
treinamento e validacao
def plot_model_accuracy(model_name_history, width = 10, height = 10):

  fig_acc = plt.figure(figsize=(width, height))

  plt.plot(model_name_history.history['accuracy'])

  plt.plot(model_name_history.history['val_accuracy'])

  plt.title('model accuracy')

  plt.ylabel('accuracy')

  plt.xlabel('epoch')

  plt.legend(['train', 'val'], loc='upper left')

  plt.show()

# Chamada da funcao
plot_model_accuracy(RNN_fwd_history,10,5)
```

Fonte: Autoria própria (2025).

Listagem 14 – Função para plotar o gráfico de perda no treinamento

```
# Definindo uma funcao para tracar a mudanca na perda nos conjuntos de
treinamento e validacao.
def plot_training_curve(model_name_history, width = 10, height = 10):
  fig_acc = plt.figure(figsize=(width, height))

  plt.plot(model_name_history.history['loss'])

  plt.plot(model_name_history.history['val_loss'])

  plt.title('model loss')

  plt.ylabel('loss')

  plt.xlabel('epoch')

  plt.legend(['train', 'val'], loc='upper left')

  plt.show()

# Chamando a funcao
plot_training_curve(RNN_fwd_history,10,5)
```

Fonte: Autoria própria (2025).

Listagem 15 – Avaliação do set do treinamento

```
def analyze_model_on_train_set(input_sequence_set, model_name):

  # Metricas de treinamento
  model_history_scores = model_name.evaluate(input_sequence_set, label_set,
  verbose=1, batch_size=50)
  print('Train Accurracy: {}'.format(model_history_scores[1]))

  # Fazendo previsoes e calculando a matriz de confusao.
  y_pred = (model_name.predict(input_sequence_set, verbose=1, batch_size=200)
  > 0.5).astype("int32")
  y_true = label_set

  test_set = pd.DataFrame(y_pred)
  test_set.to_csv('binary_submit_train.csv', index = None)

  print('Confusion matrix\n- x-axis is true labels.\n- y-axis is predicted
  labels')
  model_cm = confusion_matrix(y_true, y_pred)
  print(model_cm)

  # Calculando a precisao e o recall.
  model_precision = precision_score(y_true, y_pred)
  model_recall = recall_score(y_true, y_pred)
  print( 'Train Precision = ', model_precision, '\n', 'Train Recall = ',
  model_recall)

# Chamando a funcao
analyze_model_on_train_set(seq_set, RNN_fwd)
```

Fonte: Autoria própria (2025).

Listagem 16 – Avaliação do set do teste

```
def analyze_model_on_test_set(input_sequence_columns, model_path, width= 10,
height=5):
  # Considerando todas as ultimas sequencias do conjunto de teste.
  last_test_seq = [test_df[test_df['id']==id][input_sequence_columns].
  values[-sequence_length:]
                       for id in test_df['id'].unique() if len(test_df
                       [test_df['id']==id]) >= sequence_length]
  last_test_seq = np.asarray(last_test_seq).astype(np.float32)

  # Obtendo os rotulos do conjunto de teste.
  y_mask = [len(test_df[test_df['id']==id]) >= sequence_length for id in
  test_df['id'].unique()]
  last_test_label = test_df.groupby('id')['failure_within_w1'].nth(-1)
  [y_mask].values
  last_test_label = last_test_label.reshape(last_test_label.shape[0],1).
  astype(np.float32)
  # Carregando os melhores pesos do modelo
  if os.path.isfile(model_path):
    print("using " + model_path)
    model_estimator = load_model(model_path)
  # Testando as metricas
  start = time.time()
  scores_test = model_estimator.evaluate(last_test_seq, last_test_label,
  verbose=2)
  end = time.time()
  print("Total time taken for inferencing: ", "{:.2f}".format((end-start)),
  " secs")
  print('Test Accurracy: {}'.format(scores_test[1]))
  # Fazendo as previsoes e calcule a matriz de confusao.
  y_model_estimator_pred_test = (model_estimator.predict(last_test_seq) >0.5)
  .astype("int32")
  y_true_test = last_test_label

  test_set = pd.DataFrame(y_model_estimator_pred_test)
  test_set.to_csv('binary_submit_test.csv', index = None)
  print('Confusion matrix\n- x-axis is true labels.\n- y-axis is predicted
  labels')
  model_estimator_conf_m = confusion_matrix(y_true_test, y_model_estimator_
  pred_test)
  print(model_estimator_conf_m)
  # Calculando os valores de precisao e recall.
  model_estimator_precision_test = precision_score(y_true_test,
  y_model_estimator_pred_test)
  model_estimator_recall_test = recall_score(y_true_test,
  y_model_estimator_pred_test)
  f1_test = 2*(model_estimator_precision_test*model_estimator_recall_test)
  /(model_estimator_precision_test + model_estimator_recall_test)
  print( 'Test Precision: ', model_estimator_precision_test, '\n',
  'Test Recall: ',model_estimator_recall_test,'\n','Test F1-score:',f1_test)
```

Fonte: Autoria própria (2025).

Listagem 17 – Plot do Gráfico Previsão vs valor real

```
# Tracando os dados previstos em azul e os dados reais em verde para
verificar visualmente a precisao do modelo
  fig_verify = plt.figure(figsize=(10, 5))
  plt.plot(y_model_estimator_pred_test, color="blue")
  plt.plot(y_true_test, color="green")
  plt.title('prediction')
  plt.ylabel('value')
  plt.xlabel('row')
  plt.legend(['predicted', 'actual data'], loc='upper left')
  plt.show()
```

Fonte: Autoria própria (2025).

3.2.7 Segundo método RNN 25 características

O segundo método envolve uma *Recurrent Neural Network (RNN)* com 25 características *(features)*. Esse modelo é utilizado para capturar informações mais ricas e detalhadas nas sequências de dados, permitindo um entendimento mais profundo e preciso do contexto e das dependências temporais presentes nos dados. Ao aumentar o número de recursos, a RNN pode representar uma variedade mais ampla de informações, como características léxicas, sintáticas e semânticas em textos, ou múltiplas variáveis em séries temporais complexas.

O funcionamento da RNN com 25 recursos segue o mesmo princípio básico das RNNs tradicionais, mas com uma maior capacidade de processamento devido ao número aumentado de entradas em cada passo de tempo. A atualização do estado oculto em cada unidade recorrente é feita pela combinação das 25 entradas atuais com o estado oculto anterior, gerando um novo estado oculto que incorpora as informações de todos os recursos. A equação de atualização pode ser representada sendo:

$$ht = \sigma(Whhht - 1 + Wxhxt + bh) \tag{4}$$

onde xt é um vetor de 25 dimensões representando os recursos no tempo t.

A inclusão de múltiplos recursos também aumenta a complexidade do treinamento do modelo, exigindo técnicas de regularização e ajuste de hiperparâmetros para evitar o *overfitting* e garantir a generalização adequada do modelo. Na implementação deste método, foi usado o mesmo padrão anterior.

Listagem 18 – Treinamento do modelo

```
# Selecionando as colunas de caracteristicas.
sensor_cols = ['s' + str(i) for i in range(1,22)]
sequence_cols_25 = ['setting1', 'setting2', 'setting3', 'cycle_norm']
sequence_cols_25.extend(sensor_cols) # Adicionando os elementos de
sensor_cols em sequence_cols.
# Gera sequencias para cada ID de motor.
# unique() retorna todos os IDs exclusivos em uma lista.
seq_gen = (list(sequence_generator(train_df[train_df['id']==id],
sequence_length, sequence_cols_25)) # Vamos aproveitar todos as
caracteristicas (25).
           for id in train_df['id'].unique())
# Concatenando as sequencias dos diferentes IDs dos motores em um e
convertendo em uma matriz NumPy [(142, 50, 25), ...] --> [(ntotal, 50, 25)].
seq_set_f25 = np.concatenate(list(seq_gen)).astype(np.float32)
# A forma denota (numero de amostras, numero de passos de tempo, numero de
caracteristicas).
seq_set_f25.shape
features_dim = seq_set_f25.shape[2] #numero de caracteristicas (25).
out_dim = label_set.shape[1] # Um rotulo (failure_within_w1).
print("Features dimension: ", features_dim)
print("Output dimension: ", out_dim)
RNN_fwd_2 = Sequential()
# Compreendendo return_sequences e a conexao entre camadas RNN:
# Temos que retornar as sequencias da primeira camada para que a proxima
camada possa obter a sequencia.
RNN_fwd_2.add(SimpleRNN(input_shape=(sequence_length, features_dim), units=5,
return_sequences=True))
RNN_fwd_2.add(Dropout(0.2))
RNN_fwd_2.add(SimpleRNN(units=3,return_sequences=False))
RNN_fwd_2.add(Dropout(0.2))
RNN_fwd_2.add(Dense(units=out_dim, activation='sigmoid'))
# Compilando o modelo.
RNN_fwd_2.compile(loss='binary_crossentropy', optimizer='adam', metrics=
['accuracy'])
print(RNN_fwd_2.summary())
# Definindo o caminho para salvar o modelo
RNN_fwd_2_path = '/kaggle/working/RNN_fwd_2.h5'
import time
epochs = 200
batch_size = 200
start = time.time()
# Ajuste da rede
RNN_fwd_2_history = RNN_fwd_2.fit(seq_set_f25, label_set, epochs=epochs,
batch_size=batch_size, validation_split=0.05, verbose=2,
    callbacks = [keras.callbacks.EarlyStopping(monitor='val_loss',
    min_delta=0, patience=10, verbose=0, mode='min'),
            keras.callbacks.ModelCheckpoint(RNN_fwd_2_path,
            monitor='val_loss',save_best_only=True,mode='min',verbose=0)]
          )
end = time.time()
print("Total time taken for training:","{:.2f}".format((end-start)),"secs")
```

Fonte: Autoria própria (2025).

Figura 17 – Sumário do modelo

```
Features dimension:  25
Output dimension:  1
Model: "sequential_13"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn_11 (SimpleRNN)   (None, 50, 5)             155

 dropout_23 (Dropout)        (None, 50, 5)             0

 simple_rnn_12 (SimpleRNN)   (None, 3)                 27

 dropout_24 (Dropout)        (None, 3)                 0

 dense_15 (Dense)            (None, 1)                 4

=================================================================
Total params: 186
Trainable params: 186
Non-trainable params: 0
```

Fonte: Autoria própria (2025).

Listagem 19 – Plot dos gráficos e análise do modelo

```
1 plot_model_accuracy(RNN_fwd_2_history, 10, 5)
2
3 plot_training_curve(RNN_fwd_2_history,10,5)
4
5 analyze_model_on_train_set(seq_set_f25, RNN_fwd_2)
6
7 analyze_model_on_test_set(sequence_cols_25, RNN_fwd_2_path, 10, 5)
```

Fonte: Autoria própria (2025).

3.2.8 Método RNN 25 características bidirecional

O terceiro método mencionado é a *Bidirectional Recurrent Neural Networks (BRNN)*. Ao contrário das RNNs unidirecionais que processam os dados sequencialmente em uma única direção (do passado para o futuro), as BRNNs processam a sequência em ambas as direções simultaneamente. Isso significa que cada unidade recorrente em uma BRNN tem acesso tanto às informações passadas quanto às futuras em relação à posição atual na sequência. Essa capacidade permite que o modelo capture contextos mais abrangentes e melhore a compreensão das dependências temporais nos dados. Sendo usado o mesmo procedimento dos métodos anteriores para sua implementação.

Listagem 20 – Treinamento do modelo bidirecional

```
#Importando a  biblioteca bidirecional
from tensorflow.keras.layers import Bidirectional
features_dim = seq_set_f25.shape[2] # Numero de caracteristicas (25)
out_dim = label_set.shape[1] # Uma saida para cada sequencia (
failure_within_w1)
print("Features dimension: ", features_dim)
print("Output dimension: ", out_dim)
RNN_bi = Sequential()
# Compreendendo return_sequences e a conexao entre camadas RNN
# temos que retornar a sequencia da primeira camada para que a proxima camada
possa obter a sequencia.
RNN_bi.add(Bidirectional( # Precisamos passar a unidade RNN como um argumento
da funcao Bidirectional().
          SimpleRNN(
          input_shape=(sequence_length, features_dim),
          units=6,
          return_sequences=True)))
RNN_bi.add(Dropout(0.2))
RNN_bi.add(SimpleRNN(
           units=3,
           return_sequences=False))
RNN_bi.add(Dropout(0.2))
RNN_bi.add(Dense(units=out_dim, activation='sigmoid'))
# Compilando o modelo
RNN_bi.compile(loss='binary_crossentropy', optimizer='adam', metrics=
['accuracy'])
# Definindo o caminho para salvar o modelo
RNN_bi_path = '/kaggle/working/RNN_bi.h5'
import time
epochs = 200
batch_size = 200
start = time.time()
# Ajustando a rede
RNN_bi_history = RNN_bi.fit(seq_set_f25, label_set, epochs=epochs,
batch_size=batch_size, validation_split=0.05, verbose=2,
          callbacks = [keras.callbacks.EarlyStopping(monitor='val_loss',
          min_delta=0, patience=10, verbose=0, mode='min'),
                       keras.callbacks.ModelCheckpoint(RNN_bi_path,
                       monitor='val_loss', save_best_only=True, mode='min',
                       verbose=0)]
          )
end = time.time()
print("Total time taken for training:","{:.2f}".format((end-start)),"secs")
print(RNN_bi.summary())
```

Fonte: Autoria própria (2025).

Figura 18 – Sumário do modelo bidirecional

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 bidirectional_2 (Bidirectio  (None, 50, 12)           384
 nal)

 dropout_25 (Dropout)        (None, 50, 12)            0

 simple_rnn_14 (SimpleRNN)   (None, 3)                 48

 dropout_26 (Dropout)        (None, 3)                 0

 dense_16 (Dense)            (None, 1)                 4

=================================================================
Total params: 436
Trainable params: 436
Non-trainable params: 0
_________________________________________________________________
```

Fonte: Autoria própria (2025).

Listagem 21 – Plot dos gráficos e análise do modelo bidirecional

```
1 plot_model_accuracy(RNN_bi_history,10,5)
2
3 plot_training_curve(RNN_bi_history,10,5)
4
5 analyze_model_on_train_set(seq_set_f25, RNN_bi)
6
7 analyze_model_on_test_set(sequence_cols_25, RNN_bi_path,10,5)
```

Fonte: Autoria própria (2025).

3.2.9 Método LSTM

O quarto método é a *Long short-term memory (LSTM)*. As LSTMs são uma evolução das RNNs tradicionais, projetadas para lidar com o problema do gradiente que desaparece/explode durante o treinamento de redes profundas. Elas são capazes de aprender dependências temporais de longo prazo em sequências de dados, o que as torna especialmente eficazes em tarefas onde é crucial lembrar informações de estados anteriores por um período prolongado. Para a implementação da LSTM foram seguidos os mesmos passos dos métodos anteriores.

Listagem 22 – Treinamento do método lstm

```
# Modelo – (100 unidades LSTM com 0,2 de dropout) + (50 unidades LSTM com 0,2
de dropout) + (camada densa com sigmoid activation)

features_dim = seq_set_f25.shape[2] # numero de caracteristicas
out_dim = label_set.shape[1] # 1 Label (Target variable is failure_within_w1)

print("Features dimension: ", features_dim)
print("Output dimension: ", out_dim)

model = Sequential()

# Compreensao de return_sequences e conexao entre camadas LSTM
model.add(LSTM(
          input_shape=(sequence_length, features_dim),
          units=100,
          return_sequences=True))
model.add(Dropout(0.2))

model.add(LSTM(
           units=50,
           return_sequences=False))
model.add(Dropout(0.2))

model.add(Dense(units=out_dim, activation='sigmoid'))

# Compilando o modelo
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=
['accuracy'])
print(model.summary())

# Definindo o caminho para salvar o modelo
model_path = '/kaggle/working/binary_model.h5'

import time
epochs = 200
batch_size = 200
start = time.time()

# Ajuste da rede
history = model.fit(seq_set_f25, label_set, epochs=epochs, batch_size=
batch_size, validation_split=0.05, verbose=2,
          callbacks = [keras.callbacks.EarlyStopping(monitor='val_loss',
          min_delta=0, patience=10, verbose=0, mode='min'),
                    keras.callbacks.ModelCheckpoint(model_path,monitor=
                    'val_loss', save_best_only=True, mode='min', verbose=0)]
          )
end = time.time()
print("Total time taken for training:","{:.2f}".format((end-start)),"secs")
```

Fonte: Autoria própria (2025).

Figura 19 – Sumário do modelo lstm

```
Features dimension:  25
Output dimension:  1
Model: "sequential_15"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_4 (LSTM)               (None, 50, 100)           50400

 dropout_27 (Dropout)        (None, 50, 100)           0

 lstm_5 (LSTM)               (None, 50)                30200

 dropout_28 (Dropout)        (None, 50)                0

 dense_17 (Dense)            (None, 1)                 51

=================================================================
Total params: 80,651
Trainable params: 80,651
Non-trainable params: 0
_________________________________________________________________
```

Fonte: Autoria própria (2025).

Listagem 23 – Plot dos gráficos e análise do modelo lstm

```
# Tracando a precisao do modelo em conjuntos de treinamento e validacao
fig_acc = plt.figure(figsize=(10, 10))
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()

# Perda do modelo de plotagem para conjuntos de treinamento e teste
fig_acc = plt.figure(figsize=(10, 10))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()

# Metricas de treinamento
scores = model.evaluate(seq_set_f25, label_set, verbose=1, batch_size=50)
print('Train Accurracy: {}'.format(scores[1]))

# Fazendo previsoes e calculando a matriz de confusao
y_pred = (model.predict(seq_set_f25, verbose=1, batch_size=200) > 0.5).astype
("int32")
y_true = label_set
test_set = pd.DataFrame(y_pred)
test_set.to_csv('binary_submit_train.csv', index = None)
print('Confusion matrix\n- x-axis is true labels.\n- y-axis is predicted
labels')
cm = confusion_matrix(y_true, y_pred)
print(cm)

# Calculando a precisao e recall
precision = precision_score(y_true, y_pred)
recall = recall_score(y_true, y_pred)
print( 'Train Precision = ', precision, '\n', 'Train Recall = ', recall)
```

Fonte: Autoria própria (2025).

Listagem 24 – Avaliação do teste set

```
# Considerando todas as ultimas sequencias do conjunto de teste
last_test_seq = [test_df[test_df['id']==id][sequence_cols_25].values
[-sequence_length:]
                        for id in test_df['id'].unique() if len(test_df
                        [test_df['id']==id]) >= sequence_length]
last_test_seq = np.asarray(last_test_seq).astype(np.float32)
# Obtendo os rotulos do conjunto de teste
y_mask = [len(test_df[test_df['id']==id]) >= sequence_length for id in
test_df['id'].unique()]
last_test_label = test_df.groupby('id')['failure_within_w1'].nth(-1)[y_mask].
values
last_test_label = last_test_label.reshape(last_test_label.shape[0],1).
astype(np.float32)
# Se os melhores pesos do modelo foram salvos, carregue-os
if os.path.isfile(model_path):
    estimator = load_model(model_path)
# Metricas de teste
start = time.time()
scores_test = estimator.evaluate(last_test_seq, last_test_label, verbose=2)
end = time.time()
print("Total time taken for inferencing:","{:.2f}".format((end-start)),"secs")
print('Test Accurracy: {}'.format(scores_test[1]))
# Fazendo previsoes e calculando a matriz de confusao
y_pred_test = (estimator.predict(last_test_seq) > 0.5).astype("int32")
y_true_test = last_test_label
test_set = pd.DataFrame(y_pred_test)
test_set.to_csv('binary_submit_test.csv', index = None)
print('Confusion matrix\n- x-axis is true labels.\n- y-axis is predicted
labels')
conf_m = confusion_matrix(y_true_test, y_pred_test)
print(conf_m)
# Calculando a precisao e recall
precision_test = precision_score(y_true_test, y_pred_test)
recall_test = recall_score(y_true_test, y_pred_test)
f1_test = 2 * (precision_test * recall_test) / (precision_test + recall_test)
print( 'Test Precision: ', precision_test, '\n', 'Test Recall:', recall_test,
'\n', 'Test F1-score:', f1_test )
# Tracando em azul os dados previstos e em verde os dados reais para
verificar visualmente a precisao do modelo.
fig_verify = plt.figure(figsize=(10, 5))
plt.plot(y_pred_test, color="blue")
plt.plot(y_true_test, color="green")
plt.title('prediction')
plt.ylabel('value')
plt.xlabel('row')
plt.legend(['predicted', 'actual data'], loc='upper left')
plt.show()
```

Fonte: Autoria própria (2025).

3.2.10 Método XGBOOST

XGBoost (*Extreme Gradient Boosting*) é uma implementação otimizada do algoritmo de *boosting* de gradiente, amplamente utilizada para resolver problemas de classificação e regressão em diversos domínios. O método combina vários modelos de árvore de decisão para formar um modelo forte, que é ajustado iterativamente para corrigir os erros dos modelos anteriores. Para sua implementação se seguiu a mesma metodologia.

Listagem 25 – Treinamento e avaliação do método xgboost

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score, precision_score,
recall_score, f1_score, confusion_matrix, ConfusionMatrixDisplay
import xgboost as xgb
import matplotlib.pyplot as plt
import seaborn as sns

# Divisão em features (X) e target (y)
feature_cols = train_df.columns.difference(['id', 'cycle', 'RUL',
'failure_within_w1'])
X = train_df[feature_cols]
y = train_df['RUL']

# Divisão em conjunto de treinamento e validação
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
random_state=42)

# Ajustando pesos para classes desbalanceadas
scale_pos_weight = sum(y_train == 0) / sum(y_train == 1)  # Razão de classes

# Configuração e treinamento do modelo XGBoost com par metros otimizados

xgb_model = xgb.XGBRegressor(
    n_estimators=200,
    learning_rate=0.05,
    max_depth=3,
    subsample=0.8,
    colsample_bytree=0.8,
    alpha=10,
    lambda_=10,
    scale_pos_weight=scale_pos_weight,
    random_state=42
)
# Incluindo o conjunto de treinamento e validação no eval_set
xgb_model.fit(X_train, y_train, eval_set=[(X_train, y_train), (X_val, y_val)],
verbose=True)

# Previsões no conjunto de validação
y_val_pred = xgb_model.predict(X_val)

# Avaliação no conjunto de validação
val_mse = mean_squared_error(y_val, y_val_pred)
val_r2 = r2_score(y_val, y_val_pred)
print(f"Validation MSE: {val_mse:.2f}, R2: {val_r2:.2f}")
```

Fonte: Autoria própria (2025).

Listagem 26 – Treinamento e avaliação do método xgboost

```
# Encontrar o melhor threshold para maximizar o F1-Score
from sklearn.metrics import f1_score

thresholds = np.linspace(min(y_val_pred), max(y_val_pred), 100)
best_f1 = 0
best_threshold = 0

for threshold in thresholds:
    y_val_pred_binary = (y_val_pred > threshold).astype(int)
    current_f1 = f1_score((y_val > y_val.mean()).astype(int),
    y_val_pred_binary)
    if current_f1 > best_f1:
        best_f1 = current_f1
        best_threshold = threshold

print(f"Best Threshold: {best_threshold}, Best F1-Score: {best_f1:.2f}")

# Aplicar o melhor threshold nas previsões binárias
y_val_pred_binary = (y_val_pred > best_threshold).astype(int)

# Atualizar métricas com o melhor threshold
val_precision = precision_score((y_val > y_val.mean()).astype(int),
y_val_pred_binary)
val_recall = recall_score((y_val > y_val.mean()).astype(int),
y_val_pred_binary)
val_f1 = f1_score((y_val > y_val.mean()).astype(int), y_val_pred_binary)

print(f"Updated Validation Precision: {val_precision:.2f}, Recall:
{val_recall:.2f}, F1-Score: {val_f1:.2f}")

# Previsões no conjunto de treinamento
y_train_pred = xgb_model.predict(X_train)

# Encontrar o melhor threshold para maximizar o F1-Score no conjunto de
treinamento
thresholds = np.linspace(min(y_train_pred), max(y_train_pred), 100)
best_f1_train = 0
best_threshold_train = 0

for threshold in thresholds:
    y_train_pred_binary = (y_train_pred > threshold).astype(int)
    current_f1_train = f1_score((y_train > y_train.mean()).astype(int),
    y_train_pred_binary)
    if current_f1_train > best_f1_train:
        best_f1_train = current_f1_train
        best_threshold_train = threshold

print(f"Best Threshold (Training): {best_threshold_train}, Best F1-Score
(Training): {best_f1_train:.2f}")
```

Fonte: Autoria própria (2025).

Listagem 27 – Treinamento e avaliação do método xgboost

```
# Aplicar o melhor threshold no conjunto de treinamento
y_train_pred_binary = (y_train_pred > best_threshold_train).astype(int)
# Atualizar métricas com o melhor threshold no treinamento
train_precision = precision_score((y_train > y_train.mean()).astype(int),
y_train_pred_binary)
train_recall = recall_score((y_train>y_train.mean()).astype(int),
y_train_pred_binary)
train_f1 =f1_score((y_train>y_train.mean()).astype(int),y_train_pred_binary)

print(f"Precisão do treinamento: {train_precision:.2f}, Recall:
{train_recall:.2f}, F1-Score: {train_f1:.2f}")

# Gráfico: Acurácia no conjunto de validação (Predicted vs Actual)
plt.figure(figsize=(10, 6))
plt.plot(range(100), y_val[:100], label="Actual Data", color="green")
plt.plot(range(100), y_val_pred[:100], label="Predicted Data", color="blue",
alpha=0.7)
plt.xlabel("Linha")
plt.ylabel("Valor")
plt.title("Set de Validação: Previsão vs Valor Real")
plt.legend()
plt.grid()
plt.show()

# Gráfico: Perda (Loss) do modelo
results = xgb_model.evals_result()
training_loss = np.array(results['validation_0']['rmse'])

# Se 'validation_1' não existir, usar os valores de 'validation_0' como
placeholder.
if 'validation_1' in results:
    validation_loss = np.array(results['validation_1']['rmse'])
else:
    print("Warning: 'validation_1' not found. Using 'validation_0' as a
    placeholder for validation loss.")
    validation_loss = training_loss.copy()

# Normalizando perdas para porcentagem
total_loss = max(max(training_loss), max(validation_loss))
training_loss_percent = (training_loss / total_loss) * 100
validation_loss_percent = (validation_loss / total_loss) * 100
plt.figure(figsize=(8, 6))
plt.plot(training_loss_percent, label="Training Loss (%)")
plt.plot(validation_loss_percent, label="Validation Loss (%)")
plt.xlabel("Epocas")
plt.ylabel("Perda (%)")
plt.title("Perda do modelo ao longo das épocas (%)")
plt.legend()
plt.grid()
plt.show()
```

Fonte: Autoria própria (2025).

Listagem 28 – Treinamento e avaliação do método xgboost

```
# Gráfico: Acurácia por época
training_accuracy = 1 - (training_loss / total_loss)
validation_accuracy = 1 - (validation_loss / total_loss)

training_accuracy_percent = training_accuracy * 100
validation_accuracy_percent = validation_accuracy * 100

plt.figure(figsize=(8, 6))
plt.plot(training_accuracy_percent, label="Training Accuracy (%)",
color="blue")
plt.plot(validation_accuracy_percent, label="Validation Accuracy (%)",
color="green")
plt.xlabel("Epocas")
plt.ylabel("precisão (%)")
plt.title("Precisão do Modelo ao longo das épocas (%)")
plt.legend()
plt.grid()
plt.show()
```

Fonte: Autoria própria (2025).

Listagem 29 – Avaliação do conjunto de teste xgboost

```
# Avaliação no conjunto de teste
X_test = test_df[feature_cols]
y_test = test_df['RUL']
y_test_pred = xgb_model.predict(X_test)

test_mse = mean_squared_error(y_test, y_test_pred)
test_r2 = r2_score(y_test, y_test_pred)
print(f"Teste MSE: {test_mse:.2f}, R2: {test_r2:.2f}")

# Conversão para classificação binária no conjunto de teste
y_test_binary = (y_test > y_test.mean()).astype(int)
y_test_pred_binary = (y_test_pred > best_threshold).astype(int)
test_precision = precision_score(y_test_binary, y_test_pred_binary)
test_recall = recall_score(y_test_binary, y_test_pred_binary)
test_f1 = f1_score(y_test_binary, y_test_pred_binary)
print(f"Precisão do teste: {test_precision:.2f}, Recall: {test_recall:.2f},
F1-Score: {test_f1:.2f}")

# Gráfico: Previsões vs Reais no conjunto de teste
#plt.figure(figsize=(10, 6))
#plt.plot(range(200), y_test[:200], label="Actual Data", color="green")
#plt.plot(range(200), y_test_pred[:200], label="Predicted Data", color="blue",
alpha=0.7)
#plt.xlabel("Row")
#plt.ylabel("Value")
#plt.title("Test Set: Predicted vs Actual Data")
#plt.legend()
#plt.grid()
#plt.show()

# Gráfico: Previsões vs Reais no conjunto de teste (binários)
plt.figure(figsize=(10, 5))
plt.plot(range(100),y_test_pred_binary[:100], color="blue", label="Valor
Previsto")
plt.plot(range(100),y_test_binary[:100], color="green", label="Valor Real")
plt.title('Set de Teste: Previsão vs Valor Real')
plt.xlabel('Linha')
plt.ylabel('Valor')
plt.legend(loc='upper left')
plt.grid(True)
plt.show()

# Matriz de confusão no conjunto de validação
conf_matrix_val = confusion_matrix((y_val > y_val.mean()).astype(int),
y_val_pred_binary)
ConfusionMatrixDisplay(conf_matrix_val).plot(cmap='Blues')
plt.title("Matrix de Confusão - Set de Validação")
plt.show
```

Fonte: Autoria própria (2025).

3.2.11 Método GRU

O *Gated Recurrent Units* (GRU) é uma variação das LSTMs que simplifica a estrutura dos mecanismos de controle. O GRU utiliza apenas duas portas principais: a porta de atualização, que controla a quantidade de nova informação a ser armazenada, e a porta de *reset*, que decide quanto da informação anterior deve ser esquecida. A sua implementação seguiu os modelos anteriores.

Listagem 30 – Treinamento e avaliação do método gru

```
# Importações
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score, precision_score,
recall_score, f1_score, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import GRU, Dense, Dropout

# Dados (treinamento e teste)
# Assuma que train_df e test_df já estão carregados
# Seleção das features (incluindo 'cycle') e target
feature_cols = train_df.columns.difference(['id', 'RUL','failure_within_w1'])
# Inclui 'cycle'
X = train_df[feature_cols]
y = train_df['RUL']

# Divisão em conjunto de treinamento e validação
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
random_state=42)

# Normalização das features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)

# Remodelagem para GRU
timesteps = 1  # Usando 1 timestep
X_train_reshaped = np.expand_dims(X_train_scaled, axis=1)
X_val_reshaped = np.expand_dims(X_val_scaled, axis=1)

# Construção do modelo GRU
gru_model = Sequential([
    GRU(64, activation='tanh', return_sequences=True, input_shape=(timesteps,
     X_train_reshaped.shape[2])),
     Dropout(0.2),
    GRU(32, activation='tanh'),
     Dropout(0.2),
     Dense(1, activation='linear')  # Saída contínua para regressão
])
# Compilação do modelo
gru_model.compile(optimizer='adam', loss='mse')
```

Fonte: Autoria própria (2025).

Listagem 31 – Treinamento e avaliação do método gru

```
# Treinamento
history = gru_model.fit(
    X_train_reshaped, y_train,
    validation_data=(X_val_reshaped, y_val),
    epochs=200, batch_size=200, verbose=1
)

# Avaliação do modelo no conjunto de validação
y_val_pred = gru_model.predict(X_val_reshaped).flatten()

# Normalização para avaliação
min_pred, max_pred = y_val.min(), y_val.max()
y_val_normalized = (y_val - min_pred) / (max_pred - min_pred)
y_val_pred_normalized = (y_val_pred - min_pred) / (max_pred - min_pred)

# Cálculo de métricas no conjunto de validação
val_mse = mean_squared_error(y_val_normalized, y_val_pred_normalized)
val_r2 = r2_score(y_val_normalized, y_val_pred_normalized)
print(f"Validation MSE (Normalized): {val_mse:.2f}, R^2 (Normalized):
{val_r2:.2f}")

# Melhor threshold para classificação
thresholds = np.linspace(min(y_val_pred), max(y_val_pred), 100)
best_f1 = 0
best_threshold = 0

for threshold in thresholds:
    y_val_binary = (y_val > y_val.mean()).astype(int)
    y_val_pred_binary = (y_val_pred > threshold).astype(int)
    current_f1 = f1_score(y_val_binary, y_val_pred_binary)
    if current_f1 > best_f1:
        best_f1 = current_f1
        best_threshold = threshold
print(f"Best Threshold: {best_threshold:.2f}, Best F1-Score: {best_f1:.2f}")

# Métricas classificatórias no conjunto de validação
y_val_pred_binary = (y_val_pred > best_threshold).astype(int)
val_precision = precision_score(y_val_binary, y_val_pred_binary)
val_recall = recall_score(y_val_binary, y_val_pred_binary)
val_f1 = f1_score(y_val_binary, y_val_pred_binary)
print(f"Validation Precision: {val_precision:.2f}, Recall: {val_recall:.2f},
F1-Score: {val_f1:.2f}")
# Previsões no conjunto de treinamento
y_train_pred = gru_model.predict(X_train_reshaped).flatten()
# Encontrar o melhor threshold para maximizar o F1-Score no conjunto de
treinamento
thresholds = np.linspace(min(y_train_pred), max(y_train_pred), 100)
best_f1_train = 0
best_threshold_train = 0
```

Fonte: Autoria própria (2025).

Listagem 32 – Treinamento e avaliação do método gru

```
for threshold in thresholds:
    y_train_pred_binary = (y_train_pred > threshold).astype(int)
    current_f1_train = f1_score((y_train > y_train.mean()).astype(int),
    y_train_pred_binary)
    if current_f1_train > best_f1_train:
        best_f1_train = current_f1_train
        best_threshold_train = threshold

print(f"Best Threshold (Training): {best_threshold_train:.2f}, Best F1-Score
(Training): {best_f1_train:.2f}")

# Aplicar o melhor threshold no conjunto de treinamento
y_train_pred_binary = (y_train_pred > best_threshold_train).astype(int)

# Atualizar métricas com o melhor threshold no treinamento
train_binary = (y_train > y_train.mean()).astype(int)
train_precision = precision_score(train_binary, y_train_pred_binary)
train_recall = recall_score(train_binary, y_train_pred_binary)
train_f1 = f1_score(train_binary, y_train_pred_binary)

print(f"Training Precision: {train_precision:.2f},Recall:{train_recall:.2f},
F1-Score: {train_f1:.2f}")

# Visualização das métricas no conjunto de treinamento
metrics_train = ['Precisão', 'Recall', 'F1-Score']
values_train = [train_precision, train_recall, train_f1]

plt.figure(figsize=(8, 5))
plt.bar(metrics_train, values_train, color='lightcoral', edgecolor='black')
plt.ylim(0, 1)
for i, v in enumerate(values_train):
    plt.text(i, v + 0.02, f"{v:.2f}", ha='center', fontsize=12)

plt.title("Métricas de Treinamento")
plt.ylabel("Valor")
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()

# Perda do Modelo (sem normalização)
loss = np.array(history.history['loss'])
val_loss = np.array(history.history['val_loss'])
epochs = range(len(loss))

# Se quiser calcular a perda como uma porcentagem relativa
max_loss = np.max(loss)  # Maior valor de perda no treinamento
min_loss = np.min(loss)  # Menor valor de perda no treinamento

# Visualizando a perda normalizada (como uma porcentagem)
normalized_loss = (loss - min_loss) / (max_loss - min_loss)  # Normalização
para intervalo de 0 a 1
normalized_val_loss = (val_loss - min_loss) / (max_loss - min_loss)
```

Fonte: Autoria própria (2025).

Listagem 33 – Treinamento e avaliação do método gru

```
plt.figure(figsize=(10, 6))
plt.plot(epochs, normalized_loss * 100, label='Training Loss')
plt.plot(epochs,normalized_val_loss*100,label='Validation Loss')
plt.xlabel('Epocas')
plt.ylabel('Perda (%)')
plt.title('Perda do Modelo ao Longo das Epocas)')
plt.legend()
plt.grid()
plt.show()

# Acurácia do Modelo
train_accuracy = 1 - (loss / np.max(loss))  # Representando a acurácia com
base na perda
val_accuracy = 1 - (val_loss / np.max(val_loss))  # Representando a acurácia
de validação

plt.figure(figsize=(10, 6))
plt.plot(epochs, train_accuracy, label='Training Accuracy', color='blue')
plt.plot(epochs, val_accuracy, label='Validation Accuracy', color='green')
plt.xlabel('Epocas')
plt.ylabel('Precisao')
plt.title('Precisao do Modelo ao Longo das Epocas')
plt.legend()
plt.grid()
plt.show()

# Gráfico: Predições vs Reais (Normalizados)
plt.figure(figsize=(10, 6))
plt.plot(range(100), y_val_normalized[:100],label="Actual Data (Normalized)",
color="green")
plt.plot(range(100), y_val_pred_normalized[:100], label="Predicted Data
(Normalized)", color="blue", alpha=0.7)
plt.xlabel("Row")
plt.ylabel("Normalized Value")
plt.title("Validation Set: Predicted vs Actual Data (Normalized)")
plt.legend()
plt.grid()
plt.show()
```

Fonte: Autoria própria (2025).

Listagem 34 – Avaliação do conjunto de teste gru

```
# Previsões no conjunto de teste
# Normalizar o conjunto de teste
X_test_scaled = scaler.transform(test_df[feature_cols])
# Remodelar para GRU
X_test_reshaped = np.expand_dims(X_test_scaled, axis=1)

y_test = test_df['RUL']  # Valores reais do conjunto de teste
# Predições do modelo no conjunto de teste
y_test_pred = gru_model.predict(X_test_reshaped).flatten()

# Gráfico: Predições vs Reais no Test Set (Estilo Semelhante)
plt.figure(figsize=(10, 6))

# Plotar os valores reais do conjunto de teste
plt.plot(range(200), y_test[:200], label="Actual Data", color="green")

# Plotar os valores previstos do conjunto de teste
plt.plot(range(200), y_test_pred[:200], label="Predicted Data", color="blue",
alpha=0.7)

# Personalização do gráfico
plt.xlabel("Linha")
plt.ylabel("Valor")
plt.title("Set de Teste: Previsão vs Valor Real")
plt.legend()
plt.grid()
plt.show()

# Previsões no conjunto de teste
y_test_pred = gru_model.predict(X_test_reshaped).flatten()

# Encontrar o melhor threshold para maximizar o F1-Score no conjunto de teste
thresholds = np.linspace(min(y_test_pred), max(y_test_pred), 100)
best_f1_test = 0
best_threshold_test = 0

for threshold in thresholds:
    y_test_pred_binary = (y_test_pred > threshold).astype(int)
    current_f1_test = f1_score((y_test > y_test.mean()).astype(int),
    y_test_pred_binary)
    if current_f1_test > best_f1_test:
        best_f1_test = current_f1_test
        best_threshold_test = threshold

print(f"Best Threshold (Test): {best_threshold_test:.2f}, Best F1-Score
(Test): {best_f1_test:.2f}")
```

Fonte: Autoria própria (2025).

Listagem 35 – Avaliação do conjunto de teste gru

```
# Aplicar o melhor threshold no conjunto de teste
y_test_pred_binary = (y_test_pred > best_threshold_test).astype(int)

# Atualizar métricas com o melhor threshold no teste
test_binary = (y_test > y_test.mean()).astype(int)
test_precision = precision_score(test_binary, y_test_pred_binary)
test_recall = recall_score(test_binary, y_test_pred_binary)
test_f1 = f1_score(test_binary, y_test_pred_binary)

print(f"Test Precision: {test_precision:.2f}, Recall: {test_recall:.2f},
F1-Score: {test_f1:.2f}")

# Visualização das métricas no conjunto de teste
metrics_test = ['Precisão', 'Recall', 'F1-Score']
values_test = [test_precision, test_recall, test_f1]

plt.figure(figsize=(8, 5))
plt.bar(metrics_test, values_test, color='skyblue', edgecolor='black')
plt.ylim(0, 1)
for i, v in enumerate(values_test):
    plt.text(i, v + 0.02, f"{v:.2f}", ha='center', fontsize=12)

plt.title("Métricas de Teste")
plt.ylabel("Valor")
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()

# Matriz de Confusão
conf_matrix = confusion_matrix(y_val_binary, y_val_pred_binary)
ConfusionMatrixDisplay(conf_matrix).plot(cmap='Blues')
plt.title("Matrix de Confusão - Set de Validação")
plt.show()
```

Fonte: Autoria própria (2025).

4 RESULTADOS

Os resultados dos modelos seguem o mesmo padrão de avaliação, tendo os gráficos e avaliações separados para cada método. Primeiramente é feito a apresentação dos dados, onde são mostradas as saídas, posteriormente é feito a comparação dos resultados obtidos e as avaliações de desempenho.

4.1 Apresentação dos resultados

Para comparar os diferentes métodos de *Recurrent Neural Network (RNN)* discutidos anteriormente - RNN simples, RNN com 25 recursos, BRNN, LSTM e XGBOOST, é importante considerar algumas métricas chave que destacam suas capacidades individuais em lidar com dados sequenciais:

4.1.1 Apresentação dos dados

Tabela 2 – Comparação da Precisão, *Recall*, and *F1-Score* entre os *sets* de Treinamento e Teste para os diferentes métodos.

Método	**Treinamento**			**Teste**		
	Precisão	Recall	F1-Score	Precisão	Recall	F1-Score
RNN (1 Característica)	0.85	0.76	0.74	0.94	0.68	0.79
RNN (25 Características)	0.96	0.84	0.82	0.91	0.84	0.87
BRNN	0.90	0.86	0.84	0.95	0.72	0.81
LSTM	0.97	0.94	0.86	0.96	0.92	0.94
XGBoost	0.81	0.91	0.86	0.65	0.93	0.76
GRU	0.83	0.92	0.87	0.69	0.86	0.77

Fonte: Autoria própria (2025).

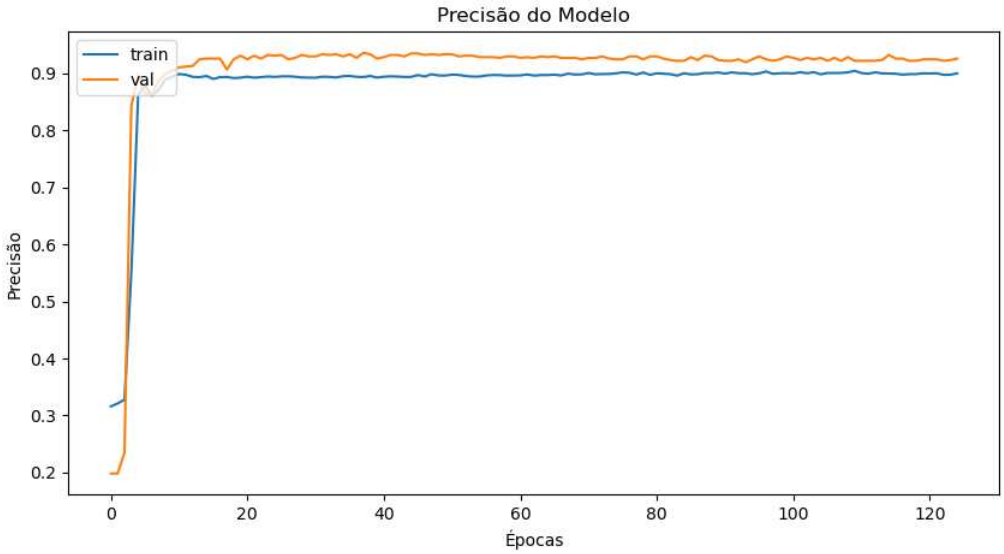
4.1.2 Avaliação de desempenho

A avaliação de desempenho dos modelos é realizada a partir dos gráficos de precisão e perda do modelo, sendo *train* os valores de treinamento e val os valores de validação. Também é feita a avaliação dos *sets* de treinamento e teste, e do gráfico de comparação dos valores previstos vs valores reais.

4.1.3 RNN de 1 característica

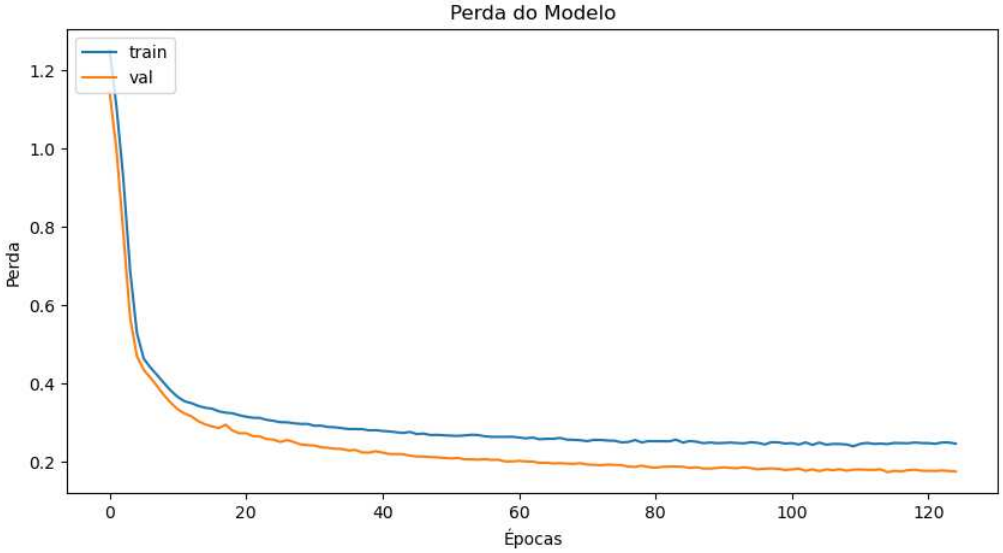
Conforme a figura 20, o modelo apresentou uma precisão no treinamento de 86% e uma perda indicada na figura 21 de aproximadamente 30%. Sendo valores razoáveis para o treinamento.

Figura 20 – Precisão do modelo rnn



Fonte: Autoria própria (2025).

Figura 21 – Perda do modelo rnn



Fonte: Autoria própria (2025).

O valor de *recall* mais baixo da figura 22 indica que ele deixa de identificar muitos positivos reais no conjunto de treinamento.

Figura 22 – Avaliação do set de treinamento

```
313/313 [==============================] - 1s 5ms/step - loss: 0.1898 - accuracy: 0.9266
Train Accurracy: 0.9265562295913696
79/79 [==============================] - 0s 4ms/step
Confusion matrix
- x-axis is true labels.
- y-axis is predicted labels
[[12117   414]
 [  734  2366]]
Precisão do Treinamento =  0.8510791366906475
 Recall do treinamento =  0.7632258064516129
```

Fonte: Autoria própria (2025).

Para o set de teste, o modelo conforme a figura 23 teve os valores de Precisão = 0.94, *Recall* = 0.68, *F1-Score* = 0.79. A precisão aumentou no teste, mas o *recall* caiu, sugerindo que o modelo é conservador ao evitar falsos positivos, mas ainda deixa passar positivos reais.

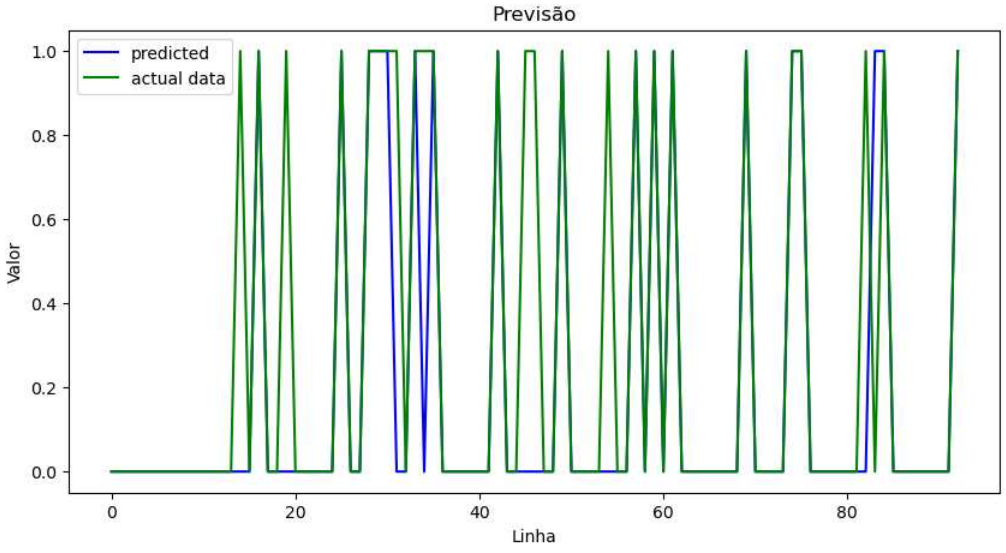
Figura 23 – Avaliação do set de teste

```
3/3 - 0s - loss: 0.2712 - accuracy: 0.9032 - 225ms/epoch - 75ms/step
Total time taken for inferencing:  0.26  secs
Test Accurracy: 0.9032257795333862
3/3 [==============================] - 0s 6ms/step
Confusion matrix
- x-axis is true labels.
- y-axis is predicted labels
[[67  1]
 [ 8 17]]
Precisão do teste:  0.9444444444444444
 Recall do teste:  0.68
 F1-score do teste: 0.7906976744186047
```

Fonte: Autoria própria (2025).

Na figura 24, observa-se alguns pontos onde o valor previsto diverge do valor real.

Figura 24 – Gráfico previsão vs valor real

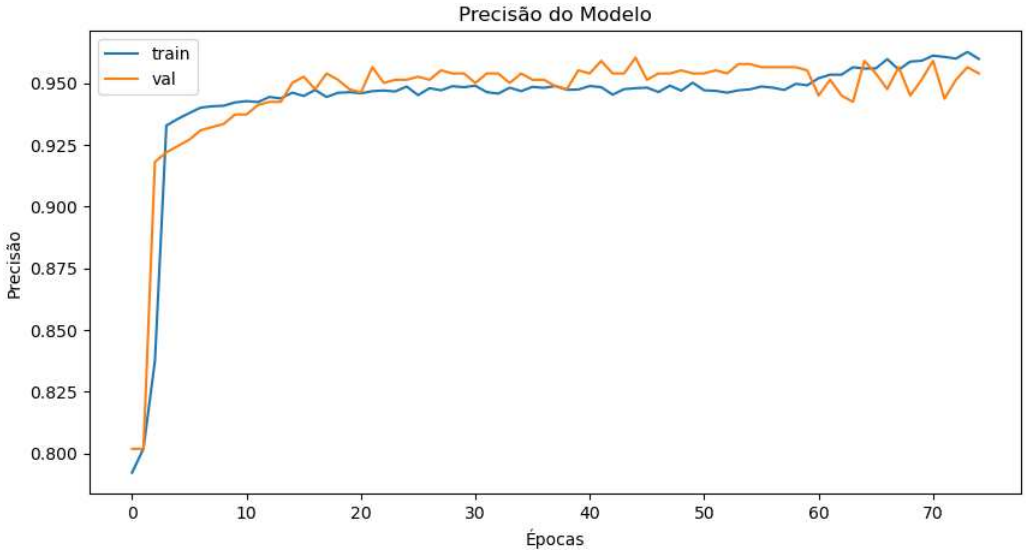


Fonte: Autoria própria (2025).

4.1.4 RNN de 25 características

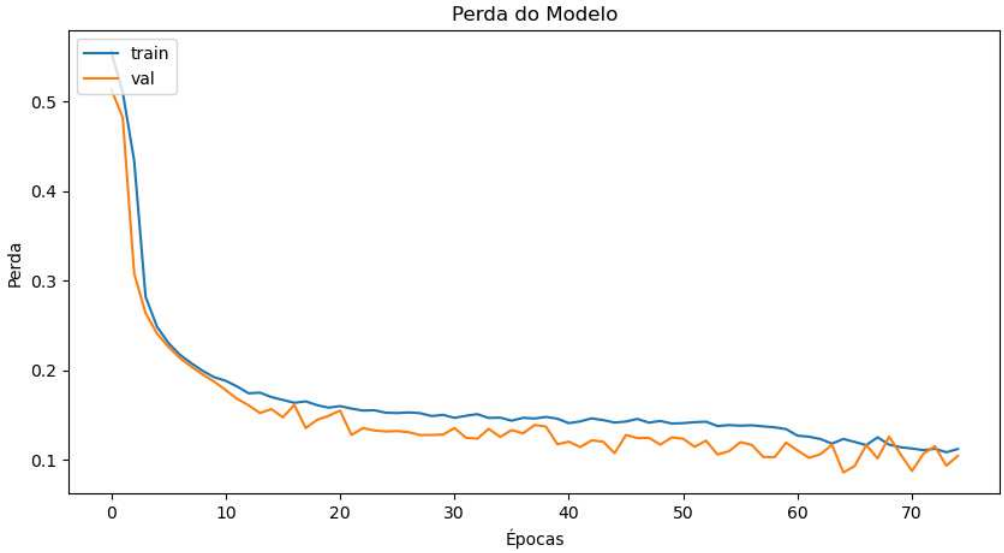
Para o segundo método, conforme a figura 25, o modelo apresentou uma precisão no treinamento de 96% e uma perda indicada na figura 26 de aproximadamente 15%. Obtendo valores melhores que o primeiro método.

Figura 25 – Precisão do modelo rnn 25 características



Fonte: Autoria própria (2025).

Figura 26 – Perda do modelo rnn 25 características



Fonte: Autoria própria (2025).

Para o set de treinamento, conforme a figura 27, o modelo teve os valores de Precisão = 0.96, *Recall* = 0.84, *F1-Score* = 0.82. Apresentando um melhor equilíbrio entre precisão e *recall* em relação à RNN com 1 característica, mostrando que incluir mais características melhora o aprendizado.

Figura 27 – Avaliação do set de treinamento 25 características

```
313/313 [==============================] - 3s 9ms/step - loss: 0.0900 - accuracy: 0.9642
Train Accurracy: 0.9642377495765686
79/79 [==============================] - 1s 8ms/step
Confusion matrix
- x-axis is true labels.
- y-axis is predicted labels
[[12448    83]
 [  476  2624]]
Precisão do Treinamento =  0.9693387513852973
 Recall do treinamento =  0.8464516129032258
```

Fonte: Autoria própria (2025).

Para o set de teste, o modelo conforme a figura 28 teve os valores de Precisão = 0.91, *Recall* = 0.84, *F1-Score* = 0.87. Tendo uma pequena redução em precisão e *recall* em relação ao treinamento.

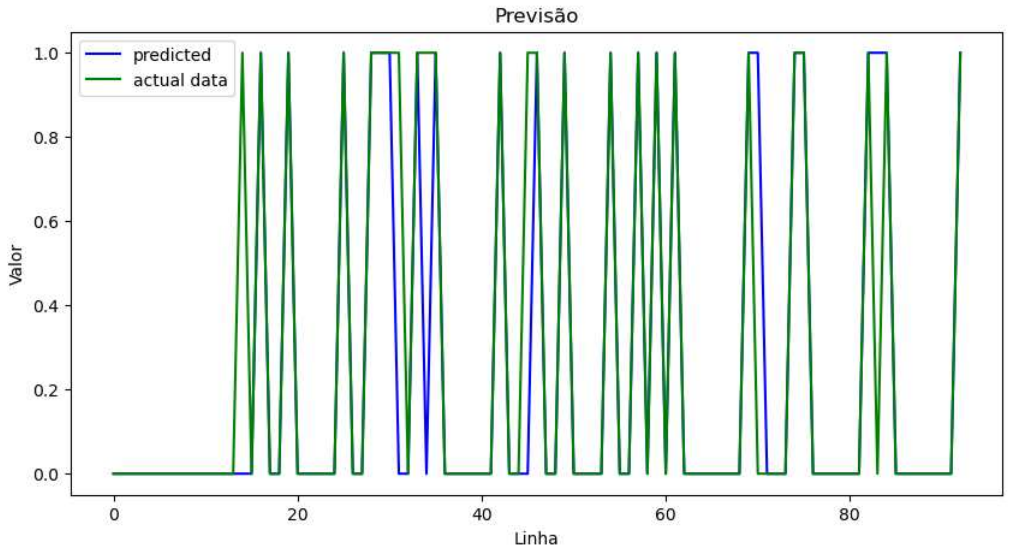
Figura 28 – Avaliação do set de teste 25 características

```
3/3 - 0s - loss: 0.1645 - accuracy: 0.9355 - 321ms/epoch - 107ms/step
Total time taken for inferencing:  0.36  secs
Test Accurracy: 0.9354838728904724
3/3 [==============================] - 0s 9ms/step
Confusion matrix
- x-axis is true labels.
- y-axis is predicted labels
[[66  2]
 [ 4 21]]
Precisão do teste:  0.9130434782608695
 Recall do teste:  0.84
 F1-score do teste: 0.8749999999999999
```

Fonte: Autoria própria (2025).

Na figura 29, nota-se que os valores previstos são mais próximos dos valores reais em comparação a RNN de 1 características.

Figura 29 – Gráfico previsão vs valor real 25 características

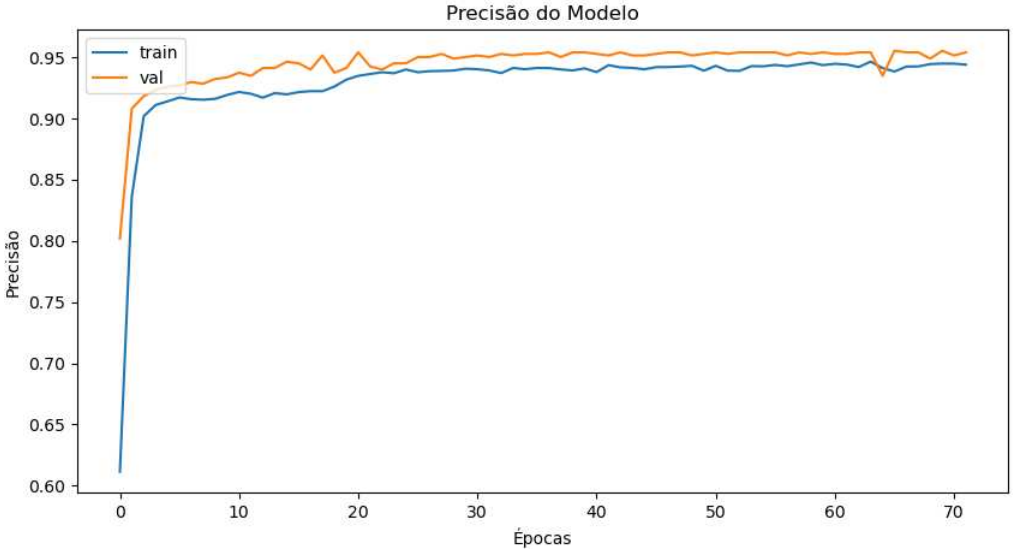


Fonte: Autoria própria (2025).

4.1.5 BRNN

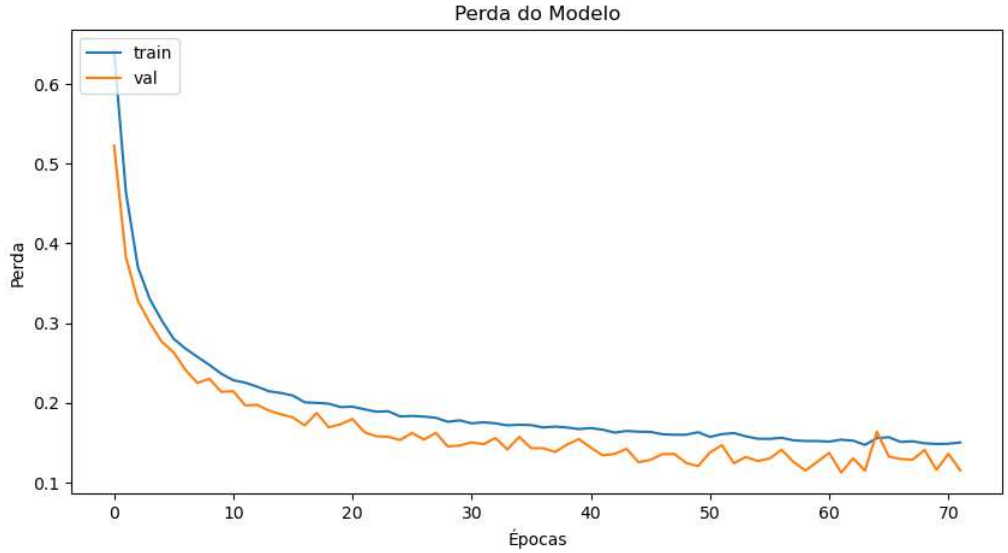
De acordo com a figura 30, o modelo BRNN teve uma precisão no treinamento de 90% e uma perda indicada na figura 31 de aproximadamente 20%. Sendo valores aproximados para treinamento do método anterior.

Figura 30 – Precisão do modelo rnn bidirecional



Fonte: Autoria própria (2025).

Figura 31 – Perda do modelo rnn bidirecional



Fonte: Autoria própria (2025).

Para o set de treinamento, conforme a figura 32, o modelo teve os valores de Precisão = 0.90, *Recall* = 0.86, *F1-Score* = 0.84. Tendo um desempenho semelhante ao da RNN com 25 características.

Figura 32 – Avaliação do set de treinamento bidirecional

```
313/313 [==============================] - 3s 11ms/step - loss: 0.1153 - accuracy: 0.9543
Train Accurracy: 0.954321563243866
79/79 [==============================] - 1s 11ms/step
Confusion matrix
- x-axis is true labels.
- y-axis is predicted labels
[[12234   297]
 [  417  2683]]
Precisão do Treinamento =  0.9003355704697986
 Recall do treinamento =  0.8654838709677419
```

Fonte: Autoria própria (2025).

Para o set de teste, conforme a figura 33, o modelo obteve os valores de Precisão = 0.95, *Recall* = 0.72, *F1-Score* = 0.81. Tendo alta precisão no teste, mas o *recall* menor indica que o modelo tende a ignorar alguns positivos reais

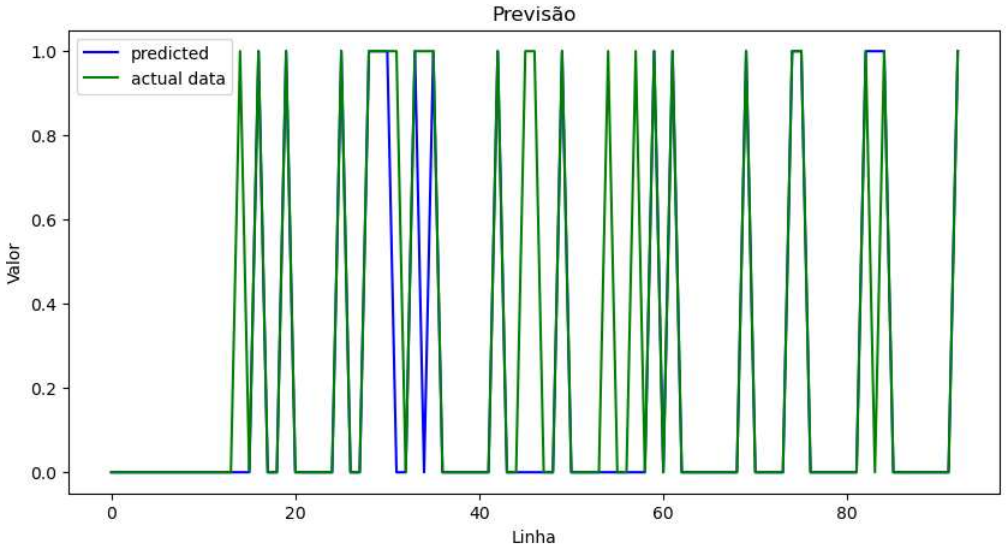
Figura 33 – Avaliação do set de teste bidirecional

```
3/3 - 0s - loss: 0.1976 - accuracy: 0.9140 - 399ms/epoch - 133ms/step
Total time taken for inferencing:  0.43  secs
Test Accurracy: 0.9139785170555115
3/3 [==============================] - 0s 9ms/step
Confusion matrix
- x-axis is true labels.
- y-axis is predicted labels
[[67  1]
 [ 7 18]]
Precisão do teste:  0.9473684210526315
 Recall do teste:  0.72
 F1-score do teste: 0.8181818181818181
```

Fonte: Autoria própria (2025).

Na figura 34, o resultado se assemelha muito a RNN de 25 características.

Figura 34 – Gráfico previsão vs valor real bidirecional

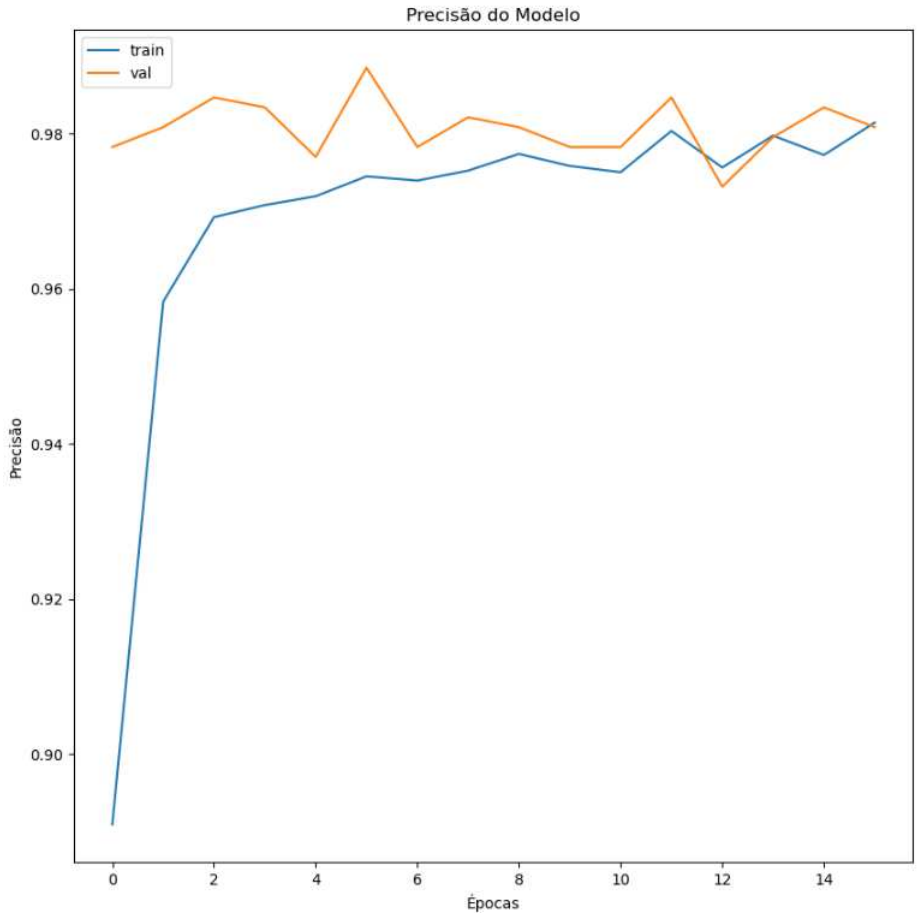


Fonte: Autoria própria (2025).

4.1.6 LSTM

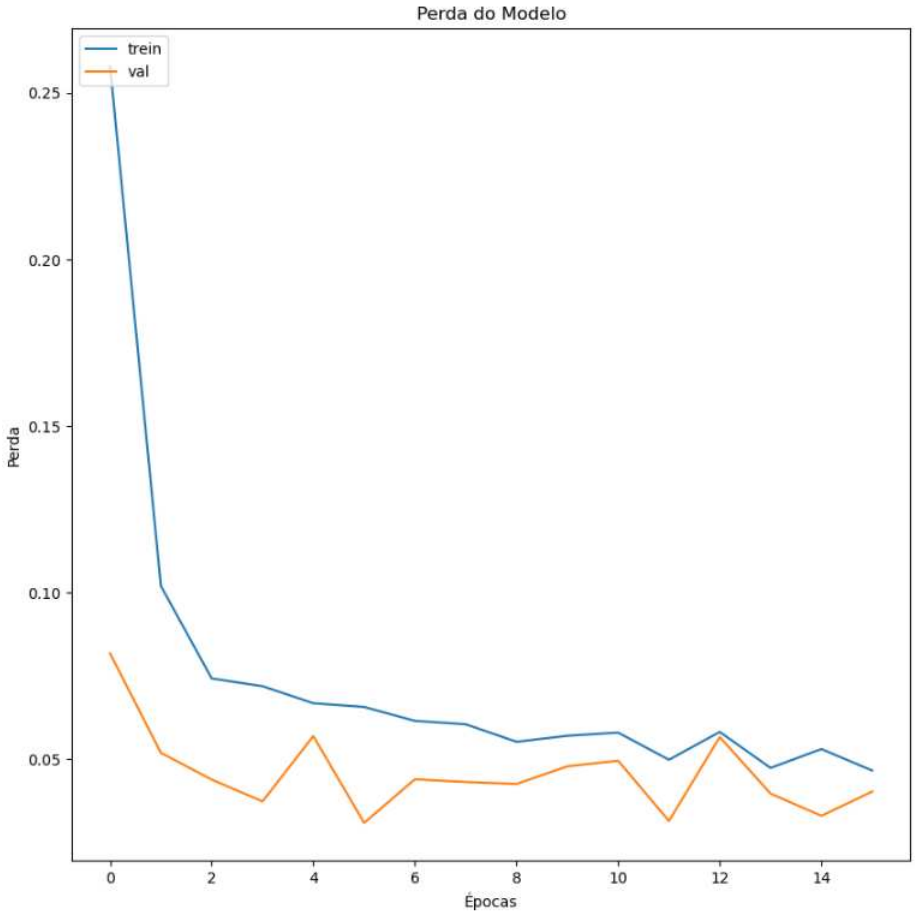
Para o modelo LSTM, de acordo com as figuras 35 e 36, o modelo LSTM teve uma precisão no treinamento de 97% e uma perda de aproximadamente 7%. Sendo um excelente desempenho no treinamento, mostrando capacidade de capturar os padrões nos dados.

Figura 35 – Precisão do modelo lstm



Fonte: Autoria própria (2025).

Figura 36 – Perda do modelo lstm



Fonte: Autoria própria (2025).

Para o set de treinamento do LSTM, conforme a figura 37, o modelo teve os valores de Precisão = 0.97, *Recall* = 0.94, *F1-Score* = 0.86. Se mostrando o melhor desempenho no treinamento entre os métodos.

Figura 37 – Avaliação do set de treinamento lstm

```
313/313 [==============================] - 9s 30ms/step - loss: 0.0409 - accuracy: 0.9830
Train Accurracy: 0.9829825162887573
79/79 [==============================] - 6s 67ms/step
Confusion matrix
- x-axis is true labels.
- y-axis is predicted labels
[[12446    85]
 [  181  2919]]
Precisão do Treinamento =  0.9717043941411452
 Recall do treinamento =  0.9416129032258065
```

Fonte: Autoria própria (2025).

No set de teste, conforme a figura 38, o modelo obteve os valores de Precisão = 0.96, *Recall* = 0.92, *F1-Score* = 0.94. Sendo o melhor modelo no teste conseguindo identificar os positivos reais e, ao mesmo tempo, mantendo a precisão alta.

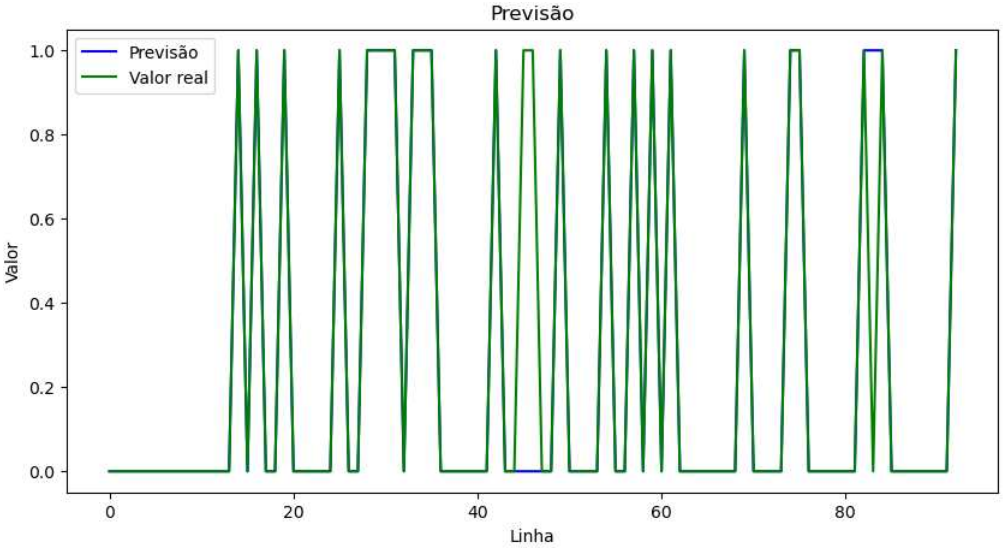
Figura 38 – Avaliação do set de teste lstm

```
3/3 - 1s - loss: 0.0693 - accuracy: 0.9677 - 933ms/epoch - 311ms/step
Total time taken for inferencing:  0.97  secs
Test Accurracy: 0.9677419066429138
3/3 [==============================] - 1s 22ms/step
Confusion matrix
- x-axis is true labels.
- y-axis is predicted labels
[[67  1]
 [ 2 23]]
Precisão do teste:  0.9583333333333334
 Recall do teste:  0.92
 F1-score do teste: 0.9387755102040817
```

Fonte: Autoria própria (2025).

Na figura 39, é obtido o melhor resultado entre os métodos aplicados.

Figura 39 – Gráfico previsão vs valor real lstm

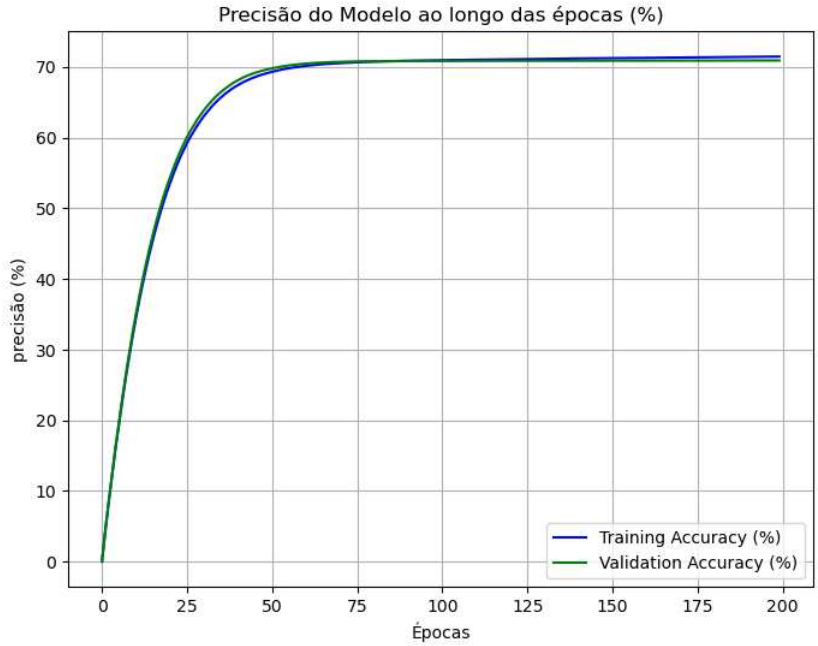


Fonte: Autoria própria (2025).

4.1.7 XGBOOST

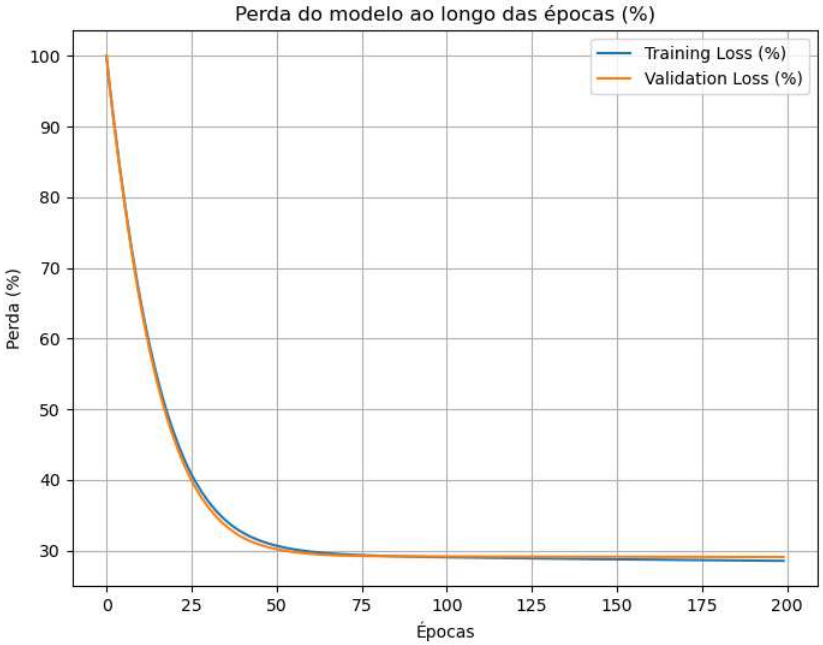
Para o método XGBOOST, a precisão no treinamento foi de 81% e uma perda de aproximadamente 30% de acordo com as figuras 40 e 41. Tendo um resultado inferior ao método LSTM.

Figura 40 – Precisão do modelo xgboost



Fonte: Autoria própria (2025).

Figura 41 – Perda do modelo xgboost



Fonte: Autoria própria (2025).

Para o set de treinamento do XGBOOST, conforme a figura 42, o modelo obteve os valores de Precisão = 0.81, *Recall* = 0.91, *F1-Score* = 0.86. Tendo resultados equilibrados, porém, inferiores ao LSTM.

Figura 42 – Avaliação do set de treinamento xgboost

```
Best Threshold: 114.30550588802859, Best F1-Score: 0.85
Updated Validation Precision: 0.84, Recall: 0.87, F1-Score: 0.85
Best Threshold (Training): 105.07271067662673, Best F1-Score (Training): 0.86
Precisão do treinamento: 0.81, Recall: 0.91, F1-Score: 0.86
```

Fonte: Autoria própria (2025).

Para o set de teste, de acordo com a figura 43, o modelo obteve os valores de Precisão = 0.65, *Recall* = 0.93, *F1-Score* = 0.76. O *recall* alto no teste mostra que o modelo identifica quase todos os positivos reais, mas a baixa precisão indica um número significativo de falsos positivos.

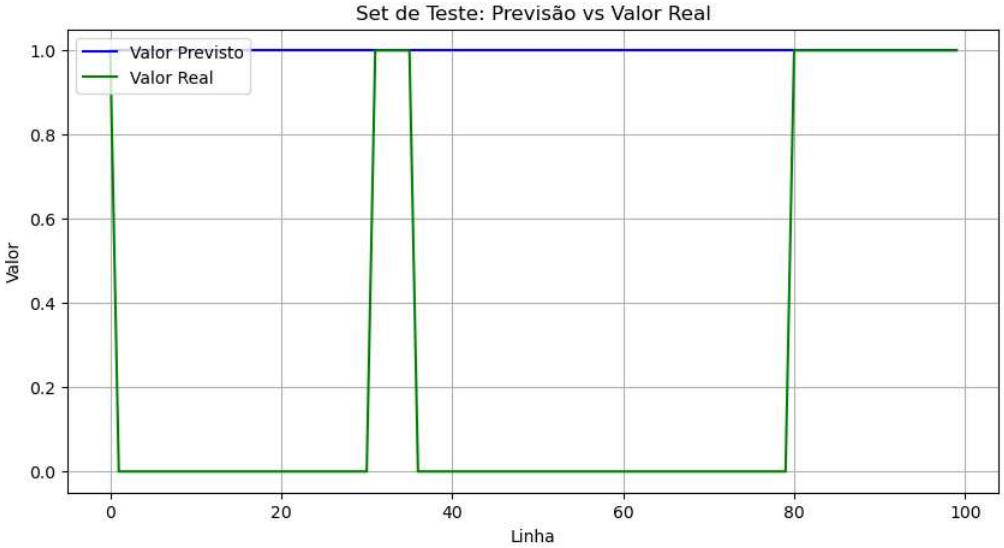
Figura 43 – Avaliação do set de teste xgboost

```
Teste MSE: 1662.05, R2: 0.52
Precisão do teste: 0.65, Recall: 0.93, F1-Score: 0.76
```

Fonte: Autoria própria (2025).

Na figura 44, se nota a baixa precisão, tendo valores previstos e reais bem diferentes.

Figura 44 – Gráfico previsão vs valor real xgboost

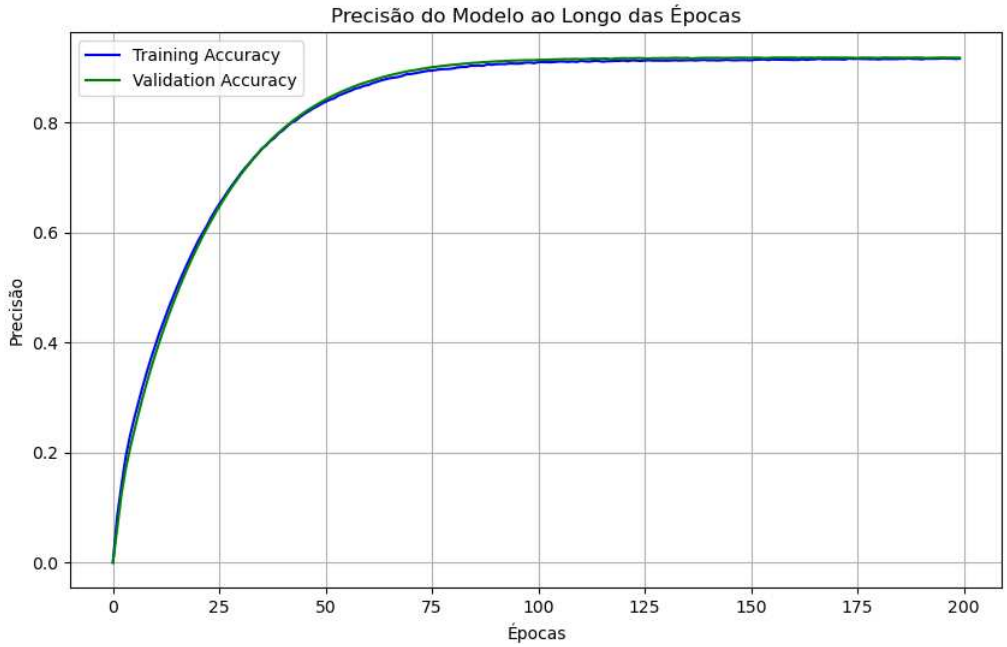


Fonte: Autoria própria (2025).

4.1.8 GRU

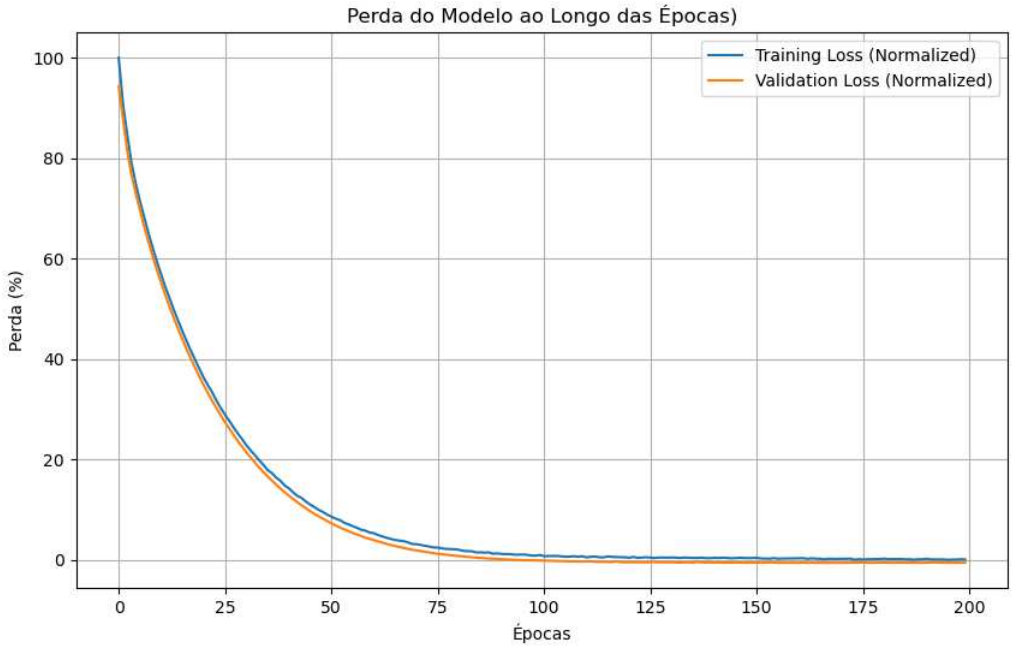
Para o último método, a precisão no treinamento foi de 83% e uma perda inferior à 5% de acordo com as figuras 45 e 46.

Figura 45 – Precisão do modelo gru



Fonte: Autoria própria (2025).

Figura 46 – Perda do modelo gru



Fonte: Autoria própria (2025).

No set de treinamento do GRU, o modelo obteve os valores de Precisão = 0.83, *Recall* = 0.92, *F1-Score* = 0.87,conforme a figura 42. Tendo um resultado equilibrado no treinamento, com prioridade ao *recall*.

Figura 47 – Avaliação do set de treinamento gru

```
Best Threshold: 105.63, Best F1-Score: 0.86
Validation Precision: 0.81, Recall: 0.92, F1-Score: 0.86
516/516 [==============================] - 1s 2ms/step
Best Threshold (Training): 108.93, Best F1-Score (Training): 0.87
Training Precision: 0.83, Recall: 0.92, F1-Score: 0.87
```

Fonte: Autoria própria (2025).

E para o set de teste, o modelo obteve os valores de Precisão = 0.69, *Recall* = 0.86, *F1-Score* = 0.77, de acordo com a figura 48. Similar ao XGBoost, o GRU apresenta *recall* alto, mas sofre com baixa precisão, indicando muitos falsos positivos no conjunto de teste.

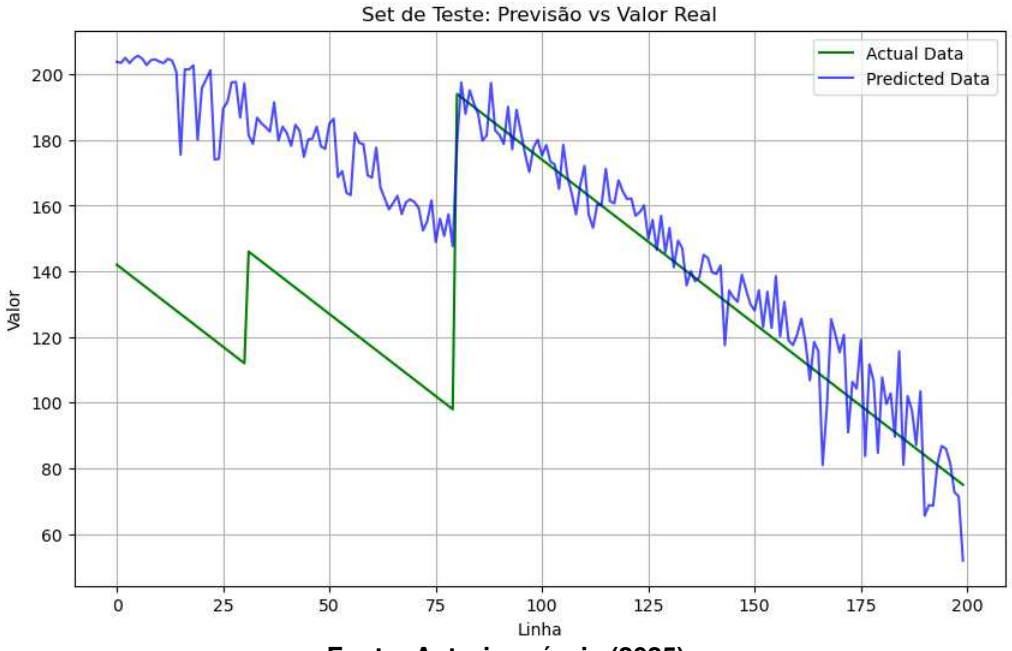
Figura 48 – Avaliação do set de teste gru

```
Best Threshold (Test): 127.90, Best F1-Score (Test): 0.77
Test Precision: 0.69, Recall: 0.86, F1-Score: 0.77
```

Fonte: Autoria própria (2025).

Na figura 49, observa-se também a baixa precisão e valores previstos e reais bem diferentes, principalmente na linhas inicias.

Figura 49 – Gráfico previsão vs valor real gru



Fonte: Autoria própria (2025).

5 CONCLUSÃO

Este trabalho desenvolveu modelos de redes neurais para previsão de falhas em motores de turbina a gás de aeronaves, comparando métodos e técnicas em dados disponibilizados pela *Nasa Open Portal Data* (NASA, 2025). Dentre os métodos utilizados estavam a *Recurrent Neural Network (RNN)* de uma característica, *Recurrent Neural Network (RNN)* de 25 características, *Bidirectional Recurrent Neural Networks (BRNN)*, *Long short-term memory (LSTM)*, *Extreme Gradient Boosting (XGBoost)* e *Gated Recurrent Unit (GRU)*. A análise de desempenho das redes neurais aplicadas se deu pela aplicação das métricas avaliação dos *sets* de treinamento e teste, utilizando a precisão, *recall* e *F1-Score* como métricas de avaliação. Também foram explorados gráficos de precisão, perda e valores previstos vs. valores reais para cada modelo, obtendo assim uma análise visual dos resultados. Os resultados demonstraram que a LSTM é mais eficaz na previsão de falhas para essa modelagem, sendo o melhor modelo em termos de precisão no teste, com uma taxa de acerto de 96%, e o mais confiável para evitar falsos positivos. No quesito *recall*, a LSTM identifica a maioria dos positivos reais, superando os demais modelos. Além disso, no teste de F1-Score, a LSTM também lidera, alcançando um F1-Score de $0{,}93$, o que demonstra o melhor equilíbrio entre precisão e recall. Os piores modelos para essa modelagem foram o XGBoost e a GRU, que apresentaram baixa precisão no teste, indicando problemas com falsos positivos.

Através dos resultados obtidos percebe-se que há oportunidades para aprimoramento e expansão deste estudo. Algumas sugestões para trabalhos futuros como a implementação de modelos híbridos de redes neurais e outras técnicas de aprendizado profundo, e a utilização de fontes de dados adicionais ou estratégias de *data augmentation* para melhorar a robustez e generalização dos modelos. Essas direções podem contribuir significativamente para o aprimoramento da previsão de falhas em turbinas, aumentando a confiabilidade e segurança dos sistemas aeronáuticos.

REFERÊNCIAS

BOUJAMZA, A.; ELHAQ, S. L. Optimizing remaining useful life predictions for aircraft engines: A dilated recurrent neural network approach. **IFAC-PapersOnLine**, v. 58, n. 13, p. 811–816, 2024. ISSN 2405-8963. 12th IFAC Symposium on Control of Power and Energy Systems - CPES 2024. Disponível em: https://www.sciencedirect.com/science/article/pii/S2405896324006815.

CHEN, T.; GUESTRIN, C. Xgboost: A scalable tree boosting system. *In*: **Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. ACM, 2016. (KDD '16), p. 785–794. Disponível em: http://dx.doi.org/10.1145/2939672.2939785.

CHO, K. *et al.* **Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation**. 2014. Disponível em: https://arxiv.org/abs/1406.1078.

HASIB, A. A. *et al.* **An Interpretable Systematic Review of Machine Learning Models for Predictive Maintenance of Aircraft Engine**. 2023. Disponível em: https://arxiv.org/abs/2309.13310.

HUNT, D. **Deep Learning Basics for Predictive Maintenance**. 2023. Acesso em: 9 fev. 2025. Disponível em: https://github.com/Azure/lstms_for_predictive_maintenance/blob/master/Deep%20Learning%20Basics%20for%20Predictive%20Maintenance.ipynb.

NASA. **C-MAPSS Jet Engine Simulated Data**. 2025. Acessado em: 8 jan. 2025. Disponível em: https://data.nasa.gov/Aerospace/CMAPSS-Jet-Engine-Simulated-Data/ff5v-kuh6/about_data.

OLAH, C. **Colah's Blog**. 2025. 8 jan. 2025. Disponível em: https://colah.github.io.

PASCANU, R.; MIKOLOV, T.; BENGIO, Y. On the difficulty of training recurrent neural networks. *In*: DASGUPTA, S.; MCALLESTER, D. (Ed.). **Proceedings of the 30th International Conference on Machine Learning**. Atlanta, Georgia, USA: PMLR, 2013. (Proceedings of Machine Learning Research, 3), p. 1310–1318. Disponível em: https://proceedings.mlr.press/v28/pascanu13.html.

PERINGAL, A.; MOHIUDDIN, M. B.; HASSAN, A. **Remaining Useful Life Prediction for Aircraft Engines using LSTM**. 2024. Disponível em: https://arxiv.org/abs/2401.07590.

SAXENA, A. *et al.* Damage Propagation Modeling for Aircraft Engine Run-to-Failure Simulation. **IEEE Transactions on Education**, p. 1–9, 2008. Disponível em: https://ieeexplore.ieee.org/document/4711414. Acesso em: 21 dez. 2024.

THAKKAR, U.; CHAOUI, H. Remaining useful life prediction of an aircraft turbofan engine using deep layer recurrent neural networks. **Actuators**, v. 11, n. 3, 2022. ISSN 2076-0825. Disponível em: https://www.mdpi.com/2076-0825/11/3/67.

WöLLMER, M. *et al.* Feature enhancement by bidirectional lstm networks for conversational speech recognition in highly non-stationary noise. *In*: **2013 IEEE International Conference on Acoustics, Speech and Signal Processing**. [*S.l.*]: IEEE, 2013. p. 6822–6826.