INSTITUTO LATINO AMERICANO DE CIÊNCIAS DA VIDA E DA NATUREZA (ILACVN)

ENGENHARIA FÍSICA

# DESENVOLVIMENTO DE UM ALGORITMO PILOTO PARA A PREVISÃO DA TEMPERATURA DE SAÍDA DO HIDROGÊNIO NA PLANTA DE PRODUÇÃO SITUADA NO ITAIPU PARQUETEC.

GUSTAVO SOARES PINTO

Foz do Iguaçu
2025

**INSTITUTO LATINO-AMERICANO DE CIÊNCIAS DA VIDA E DA NATUREZA (ILACVN)**
**ENGENHARIA FÍSICA**

# DESENVOLVIMENTO DE UM ALGORITMO PILOTO PARA A PREVISÃO DA TEMPERATURA DE SAÍDA DO HIDROGÊNIO NA PLANTA DE PRODUÇÃO SITUADA NO ITAIPU PARQUETEC.

Trabalho de Conclusão de Curso apresentado ao Instituto Latino-Americano de Ciências da Vida e da Natureza da Universidade Federal da Integração Latino-Americana, como requisito parcial à obtenção do título de Bacharel em Engenharia Física.

Orientador: Prof. Dr. Raphael Fortes Infante Gomes
Coorientador: Angel Ambrocio Quispe

Foz do Iguaçu
2025

**GUSTAVO SOARES PINTO**

# DESENVOLVIMENTO DE UM ALGORITMO PILOTO PARA A PREVISÃO DA TEMPERATURA DE SAÍDA DO HIDROGÊNIO NA PLANTA DE PRODUÇÃO SITUADA NO ITAIPU PARQUETEC.

Trabalho de Conclusão de Curso apresentado ao Instituto Latino-Americano de Ciências da Vida e da Natureza da Universidade Federal da Integração Latino-Americana, como requisito parcial à obtenção do título de Bacharel em Engenharia Física.

**BANCA EXAMINADORA**

---

Orientador: Prof. Dr. Raphael Fortes Infante Gomes
UNILA

---

Prof. Dr. Rodrigo Santos da Lapa
(UNILA)

---

Prof. Dr. Joylan Nunes Maciel
(UNILA)

Foz do Iguaçu, _____ de _____ de _____.

# AGRADECIMENTOS

Gostaria de expressar minha gratidão a todos que me apoiaram ao longo desta trajetória. Em primeiro lugar, aos meus pais, cujo apoio tornou possível a realização deste curso. Aos professores, cujo conhecimento e dedicação foram fundamentais para meu sucesso. Aos técnicos, sempre solícitos e dispostos a ajudar no que fosse necessário. Aos amigos, que tornaram essa jornada mais leve e divertida – embora nem sempre mais fácil.

Meu agradecimento também ao Itaipu Parquetec e a seus funcionários, que viabilizaram este trabalho, assim como aos meus orientadores, Raphael Fortes Infante Gomes e Angel Ambrocio Quispe, pelo suporte.

Por fim, um agradecimento especial às duas pessoas que me aturaram diariamente, tanto em casa quanto na faculdade, mesmo diante das minhas inúmeras trapalhadas: Esdras Rebecchi de Almeida e Guilherme Soares Pinto.

*"O líder é aquele que bate o pênalti no último minuto quando o time precisa. E é aquele que deixa o outro bater quando o jogo está ganho."— Sócrates Brasileiro Sampaio de Souza Vieira de Oliveira*

# Resumo

Este trabalho propõe o desenvolvimento e teste de um algoritmo para prever a temperatura de saída do hidrogênio ($H_2$) em uma planta industrial, utilizando aprendizado de máquina. Plantas de produção de hidrogênio são instalações responsáveis pela geração, purificação e armazenamento desse elemento, que desempenha um papel fundamental como vetor de energia limpa. O hidrogênio é amplamente utilizado para o armazenamento de energia gerada a partir de fontes renováveis, permitindo a redução da dependência de combustíveis fósseis. Foram explorados modelos Long Short-Term Memory (LSTM) e sua versão quântica, Quantum Long Short-Term Memory (QLSTM), ambos avaliados com e sem Deep Reinforcement Learning (DRL). A previsão precisa dessa variável é essencial para otimizar processos, reduzir custos e garantir a segurança do sistema. Além disso, este estudo está entre os primeiros a investigar o uso combinado de DRL com QLSTM, bem como a aplicação de DRL com LSTM, ampliando o campo de pesquisa na área. A metodologia incluiu pré-processamento de dados, regressões lineares e polinomiais, além da otimização de hiperparâmetros com Optuna. Os modelos foram avaliados por métricas como Erro Quadrático Médio (RMSE), onde o LSTM obteve 0,0658 no treinamento e 0,0826 na validação. O QLSTM apresentou dificuldades em sua execução devido ao alto custo computacional, e o DRL demonstrou potencial na seleção de variáveis. Este estudo contribui para o uso da inteligência artificial na indústria do hidrogênio, ressaltando desafios e oportunidades da computação quântica aplicada ao aprendizado de máquina, além de reforçar a importância da pesquisa contínua para aprimoramento de modelos. Ao aprimorar a previsibilidade e a estabilidade dos processos em plantas de hidrogênio, este trabalho apoia o avanço de tecnologias voltadas para a transição energética e a sustentabilidade.

**Palavras-chave**: Produção de Hidrôgenio, Machine Learning, Quantum Machine Learning.

# Resumen

Este trabajo propone el desarrollo y prueba de un algoritmo para predecir la temperatura de salida del hidrógeno ($H_2$) en una planta industrial utilizando aprendizaje automático. Las plantas de producción de hidrógeno son instalaciones responsables de la generación, purificación y almacenamiento de este elemento, que desempeña un papel fundamental como vector de energía limpia. El hidrógeno se utiliza ampliamente para el almacenamiento de energía generada a partir de fuentes renovables, contribuyendo a la reducción de la dependencia de los combustibles fósiles. Se exploraron modelos Long Short-Term Memory (LSTM) y su versión cuántica, Quantum Long Short-Term Memory (QLSTM), ambos evaluados con y sin Deep Reinforcement Learning (DRL). La predicción precisa de esta variable es esencial para optimizar procesos, reducir costos y garantizar la seguridad del sistema. Además, este estudio se encuentra entre los primeros en investigar el uso combinado de DRL con QLSTM, así como la aplicación de DRL con LSTM, ampliando el campo de investigación en esta área. La metodología incluyó preprocesamiento de datos, regresiones lineales y polinomiales, además de la optimización de hiperparámetros con Optuna. Los modelos fueron evaluados mediante métricas como el Error Cuadrático Medio (RMSE), donde LSTM obtuvo 0,0658 en el entrenamiento y 0,0826 en la validación. QLSTM presentó dificultades en su ejecución debido a su alto costo computacional, mientras que DRL demostró potencial en la selección de variables. Este estudio contribuye al uso de la inteligencia artificial en la industria del hidrógeno, resaltando los desafíos y oportunidades de la computación cuántica aplicada al aprendizaje automático, además de reforzar la importancia de la investigación continua para la mejora de modelos. Al mejorar la previsibilidad y estabilidad de los procesos en plantas de hidrógeno, este trabajo apoya el avance de tecnologías orientadas a la transición energética y la sostenibilidad.

**Palabras clave**: Producción de hidrógeno, aprendizaje automático, aprendizaje automático cuántico.

# Abstract

This work proposes the development and testing of an algorithm to predict the hydrogen ($H_2$) outlet temperature in an industrial plant using machine learning. Hydrogen production plants are facilities responsible for generating, purifying, and storing this element, which plays a fundamental role as a clean energy carrier. Hydrogen is widely used for storing energy generated from renewable sources, helping to reduce dependence on fossil fuels. Long Short-Term Memory (LSTM) models and their quantum version, Quantum Long Short-Term Memory (QLSTM), were explored, both evaluated with and without Deep Reinforcement Learning (DRL). Accurate prediction of this variable is essential to optimize processes, reduce costs, and ensure system safety. Furthermore, this study is among the first to investigate the combined use of DRL with QLSTM, as well as the application of DRL with LSTM, expanding research in this field. The methodology included data preprocessing, linear and polynomial regressions, as well as hyperparameter optimization using Optuna. The models were evaluated using metrics such as Root Mean Squared Error (RMSE), where LSTM achieved 0.0658 in training and 0.0826 in validation. QLSTM faced execution difficulties due to its high computational cost, while DRL showed potential in variable selection. This study contributes to the application of artificial intelligence in the hydrogen industry, highlighting challenges and opportunities in quantum computing applied to machine learning, while also emphasizing the importance of continuous research for model improvement. By enhancing the predictability and stability of processes in hydrogen plants, this work supports the advancement of technologies aimed at energy transition and sustainability.

**Keywords**: Hydrogen Production, Machine Learning, Quantum Machine Learning.

# Sumário

# Lista de ilustrações

# Lista de tabelas

# Glossário de Siglas

**AI** Inteligência Artificial

**CPU** Unidade Central de Processamento (componente principal de computadores que executa instruções)

**DRL** Deep Reinforcement Learning (Aprendizado por Reforço Profundo)

**GPU** Unidade de Processamento Gráfico (componente especializado em processamento paralelo)

**H_2** Hidrogênio (elemento químico gasoso)

**H2O** Água (molécula composta por hidrogênio e oxigênio)

**IBM** Empresa multinacional de tecnologia pioneira em computação e inteligência artificial

**LSTM** Long Short-Term Memory (Memória de Longo Curto Prazo)

**MAE** Erro Absoluto Médio (métrica de avaliação de modelos)

**MAPE** Erro Percentual Absoluto Médio (métrica de precisão percentual)

**ML** Machine Learning (Aprendizado de Máquina)

**MSELoss** Função de perda usando Erro Quadrático Médio

**O2** Oxigênio (elemento químico essencial para respiração)

**QLSTM** Quantum Long Short-Term Memory (Memória Quântica de Longo Curto Prazo)

**RNAs** Redes Neurais Artificiais

**RNNs** Redes Neurais Recorrentes

**RMSE** Erro Quadrático Médio (Raiz do Erro Quadrático Médio)

# 1 INTRODUÇÃO

A necessidade de novas fontes de energia limpa vem crescendo cada vez mais, e o hidrogênio ($H_2$) surge como uma alternativa promissora. Visto que é amplamente reconhecido por sua capacidade de atuar como uma alternativa viável aos combustíveis fósseis, especialmente devido à sua elevada densidade energética e emissões nulas de carbono quando utilizado em células a combustível.

Para melhorar os processos de produção e armazenamento de $H_2$, é essencial entender e prever o comportamento de variáveis-chave; uma das propostas na literatura foi prever a temperatura de saída do hidrogênio. Além disso, com o avanço da indústria 4.0 e a manutenção preditiva, em parceria com o Itaipu Parquetec, foi vista a necessidade de iniciar trabalhos nessa área. As motivações principais são regidas por um desenvolvimento de manutenção mais eficiente além de otimizar os processos de produção de $H_2$.

Com isso, este trabalho tem como objetivo desenvolver e testar um algoritmo piloto para prever essa temperatura, utilizando abordagens de aprendizado de máquina, incluindo modelos clássicos como LSTM (Long Short-Term Memory) e sua versão quântica QLSTM, com e sem Deep Reinforcement Learning (DRL).

Para isso, inicialmente, realiza-se uma análise detalhada dos conceitos fundamentais relacionados ao aprendizado de máquina, manutenção preditiva e computação quântica, contextualizando a importância da previsão precisa da temperatura de saída do hidrogênio para a eficiência operacional da planta.

Em seguida, são descritas as metodologias adotadas, que incluíram etapas de pré-processamento de dados (limpeza, análise de correlação e normalização), treinamento de regressões lineares e polinomiais para compreensão preliminar dos dados e implementação de modelos LSTM e QLSTM.

O estudo compara os diferentes algoritmos, avaliando o desempenho dos modelos clássicos e quânticos por meio de métricas como Erro Quadrático Médio (RMSE), Erro Absoluto Médio (MAE) e Erro Percentual Absoluto Médio (MAPE), além da análise de generalização para evitar overfitting.

Os resultados demonstraram que o modelo LSTM clássico apresentou maior robustez, com RMSE de 0,0658 no treino e 0,0826 na validação, enquanto o QLSTM enfrentou limitações práticas devido ao alto custo computacional. O uso de Optuna para otimização de hiperparâmetros reduziu significativamente o tempo de treinamento, porém houve um decaimento nas métricas e um possível *overfitting*. O DRL, por sua vez, mostrou potencial para simplificar a seleção de variáveis.

Este trabalho contribui para o avanço da produção sustentável de hidrogênio ao demonstrar a viabilidade de modelos clássicos de IA em ambientes industriais, enquanto sinaliza

caminhos para futuras pesquisas em computação quântica. Os desafios identificados – como a dependência de hardware avançado e a necessidade de datasets mais robustos – reforçam a importância de investimentos em infraestrutura e coleta de dados. A integração de técnicas como DRL e otimização automatizada abre perspectivas para sistemas autônomos e adaptativos, capazes de operar em tempo real em plantas industriais, alinhando-se aos objetivos da Indústria 4.0 e à transição global para energias limpas.

# 2 FUNDAMENTAÇÃO TÉORICA

## 2.1 PRODUÇÃO DE HIDROGÊNIO

Com o aumento da produção mundial é necessário encontrar novas soluções sustentáveis, que ainda assim consigam atender às necessidades da humanidade. Tendo isso em vista, o hidrogênio se destaca como um transportador de energia limpa (Stargate Hydrogen, ). Principalmente pelo fato de ele poder ser produzido a partir de fontes renováveis, pela sua versatilidade como vetor energético e estar em acordo com o alinhamento com politicas globais de mudança climática (KUMAR; LIM, 2022).

Existem diversas maneiras de se produzir hidrogênio, cada uma com suas peculiaridades e impactos ambientais, sendo eles:

- **Eletrólise da água:** este método utiliza eletricidade para dividir a água em hidrogênio e oxigênio. Quando a eletricidade é proveniente de fontes renováveis, como solar ou eólica (KUMAR; LIM, 2022).
- **Reformagem de gás natural:** Este é o método mais comum atualmente, onde o gás natural é convertido em hidrogênio através de um processo de reforma a vapor. Embora seja uma forma eficiente de produção, o processo CO2 como subproduto (KUMAR; LIM, 2022).
- **Gasificação de biomassa ou carvão:** neste método, a biomassa ou o carvão são convertidos em gás de síntese (que contém hidrogênio, entre outros gases) através de reações químicas em altas temperaturas com uma quantidade limitada de oxigênio ou vapor d'água. O hidrogênio produzido a partir da biomassa pode ser considerado mais sustentável (KUMAR; LIM, 2022).

Na eletrolise existem três métodos mais utilizados, sendo eles: a alcalina, de óxido sólido e de membrana de troca de prótons. Entretanto todos usam a mesma reação química:

$$2\,H_2O(l) \rightarrow 2\,H_2(g) + O_2(g),$$

### 2.1.1 Eletrólise Alcalina

É um dos métodos mais utilizados na produção de hidrogênio, cujo processo envolve a quebra da molécula de água utilizando uma célula eletrolítica alcalina, que possui uma solução eletrolítica alcalina tipicamente hidróxido de potássio (KOH) ou hidróxido de sódio (NaOH), que serve como transportador de íons. A célula é dividida em um ânodo e um cátodo, ambos submersos na solução de eletrólito alcalino (Stargate Hydrogen, ). No cátodo (eletrodo

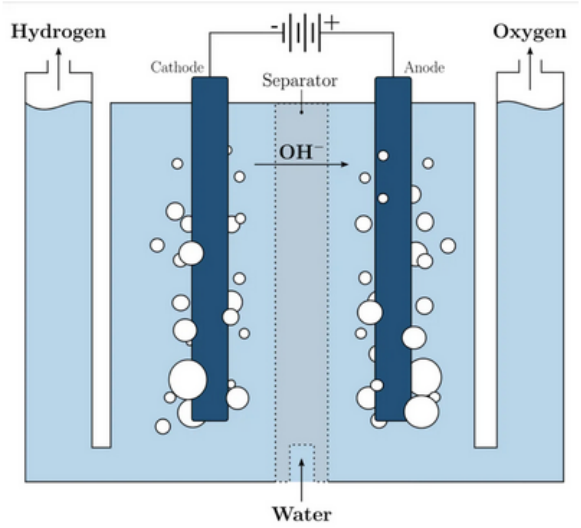



Figura 1 – Esquema de um eletrolisador alcalino Fonte: HYDROGEN, S. The Basics of Hydrogen Electrolysis

(Stargate Hydrogen, )

negativo), a água é reduzida para gerar hidrogênio e íons hidróxido $(OH^-)$. Já no ânodo (eletrodo positivo), os íons hidróxido se oxidam para formar água e oxigênio.

Além disso, ainda possuem diafragmas/separadores que evitam a mistura dos gases produzidos (hidrogênio e oxigênio). Tradicionalmente, materiais como amianto ou ZrO foram utilizados (KUMAR; LIM, 2022), e as células eletrolíticas, compostas por várias unidades de pilha, são empregadas para aumentar a produção de hidrogênio, como podem ser vistos na Fig. 1

A eletrólise funciona a partir da corrente elétrica que passa pela solução alcalina e segue as seguintes reações:

No Ânodo (oxidação):

$$4OH^- -> O2(g) + H2O(aq) + 4e^-.$$

No Cátodo (redução):

$$2H2O(l) + 2e^- -> H2(g) + 2OH^-(aq).$$

Equação Global Balanceada:

$$2H2O(l) -> 2H2(g) + O2(g).$$

A eletrólise alcalina trabalha em uma faixa de operação entre 30 e 80°C. Esta reação possui uma densidade de corrente entre 0.1 a 0.5 A/cm$^2$ e um custo de investimento que varia entre USD 500 e 1000 por kW, com uma vida útil estimada de até 90.000 horas (KUMAR; LIM, 2022).

As principais vantagens do uso da eletrólise alcalina são o fato de ser uma tecnologia bem estabelecida, com vários anos de experiência em aplicações industriais, e seu custo relativamente

baixo, uma vez que utiliza eletrodos que não requerem metais nobres, o que contribui para a redução dos custos. Além disso, apresenta estabilidade a longo prazo, demonstrando boa durabilidade e confiabilidade ao longo do tempo. (KUMAR; LIM, 2022).

## 2.2 MANUTENÇÃO PREDITIVA

Graças à aceleração do desenvolvimento tecnológico nas últimas décadas, foi alcançado o estágio correspondente à chamada "Quarta Revolução Industrial", onde se conseguiu associar diversos sistemas físicos a dispositivos digitais e, com isso, facilitar e aumentar a coleta de dados. A entrada no contexto da indústria 4.0 e o aumento da quantidade de dados disponíveis aumentaram a pressão para maximizar a produção e lucros, em detrimento da redução dos custos de operação nos respectivos processos(RODRIGUES et al., 2023).

Para aumentar a produtividade, é necessário que a confiabilidade e a disponibilidade dos equipamentos sejam otimizadas para reduzir gastos e perdas com manutenção, mantendo a operação das máquinas continuamente, acelerando seu retorno para a produção ou até mesmo otimizando sua produção (RODRIGUES et al., 2023).

As manutenções podem ser definidas segundo o momento em que elas são executadas e pelos seus objetivos prévios, como definido em (SELCUK, 2017):

1. **Manutenção corretiva (ou não planejada)**: por ser executada somente após o momento da falha do sistema, para somente aí restaurá-lo. Todavia, este procedimento só pode ser executado se as consequências das falhas não forem significativas.

2. **Manutenção preventiva**: também conhecida como manutenção baseada em tempo ou manutenção de rotina, é fundamentada em estatísticas fornecidas pelo fabricante ou por experiências passadas. Neste caso, as operações são planejadas com antecedência para que não ocorram falhas e o equipamento permaneça ocioso pelo menor tempo possível.

3. **Manutenção preditiva**: Como o sugere a ideia é prever, a partir de informações e dados de deterioração, quando haverá falha e realizar a manutenção somente quando realmente for necessário. Este modelo necessita de um monitoramento contante, ou seja, através de análises periódicas ou por sensores.

A implementação da manutenção preditiva nas indústrias oferece uma série de benefícios, que incluem o aumento da confiabilidade (uma vez que o problema será identificado e resolvido antes de resultar em falhas) redução de custo (já que a operação somente será pausada quando realmente for necessário) além de diminuir o gasto com peças de reposição e mão-de-obra. E, por fim, este processo permite aumentar a eficiência energética, pois existe a possibilidade de identificar problemas que levam a um consumo excessivo de energia, permitindo assim uma redução dos custos(SELCUK, 2017).

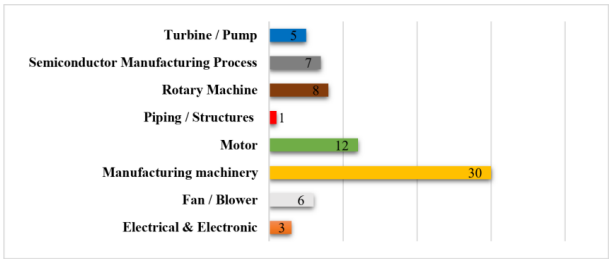



Figura 2 – Estudo de casos sobre aprendizado de máquina bem sucedidas em equipamentos adaptado (NACCHIA et al., 2021).

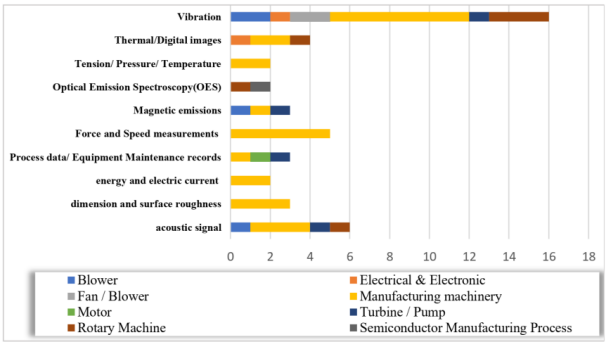



Figura 3 – Comparação entre equipamentos e tipos de monitoramento adaptado (NACCHIA et al., 2021).

Entretanto, esse modo de operação demanda um aumento no valor empregado, principalmente pelo uso de técnicas de *machine learning*. Com isso em mente, M. Nacchia e outros pesquisadores tiveram a ideia de procurar as respostas para algumas questões. A primeira questão relevante para este trabalho é: "**quais classes de equipamentos são candidatas a serem mapeadas para a manufatura inteligente? Além disso, qual a relação entre esses equipamentos e os tipos de dados sob investigação?**"(NACCHIA et al., 2021). No estudo foram realizadas duas consultas com o intuito de responder: o primeiro deles sobre os tipos de equipamentos e processos onde aprendizado de máquina foi bem-sucedida: já o segundo, focado na degradação dos equipamentos, como podem ser vistos nas Figs. 2 e em 3.

A Fig.2 relata os tipos de equipamentos e processos de aprendizado bem-sucedidos que foram registrados nos trabalhos consultados. A fabricação de máquinas foi a área mais proeminente de fabricação, seguida por motores e máquinas rotativas. Por sua vez, setor mais relevante que fez uso de ML foi a fabricação de semicondutores. Já a Fig.3 mostra quais foram os tipos de dados mais utilizados para prever as degradações em equipamentos, sendo a vibração o mais utilizado, entre os 42 artigos apresentados, e os dados térmicos, ou imagens digitais e emissões magnéticas, os que menos foram aplicados.

A outra questão que nos interessa é respondida pelo mesmo artigo sobre quais são as técnicas mais frequentes de *Machine Learning* utilizadas: **qual estratégia de aprendizado de máquina é mais regularmente aplicada para manutenção preditiva e para quais tarefas ela é definida?** A resposta pode ser obtida através da análise das Figs. 4 e 5.

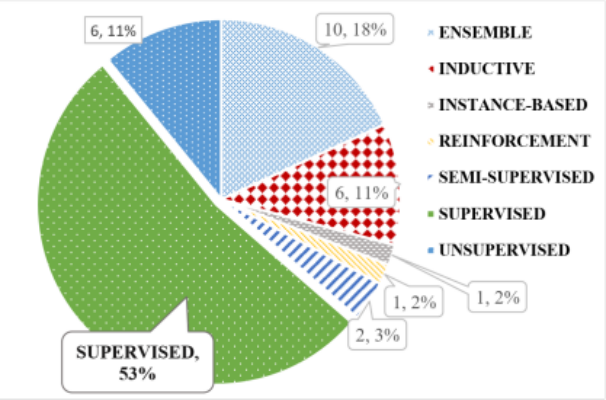



Figura 4 – Popularidade das técnicas de *Machine Learning* adaptado de (NACCHIA et al., 2021).

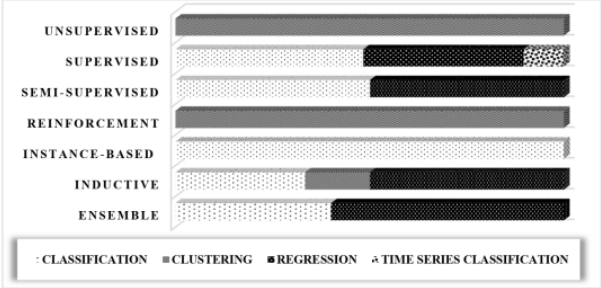



Figura 5 – Comparação entre as diferentes técnicas de Machine Learning adaptado de (NACCHIA et al., 2021).

A Fig.4 mostra que o grupo de técnicas mais utilizado é o de aprendizado supervisionado com 53% seguido de aprendizado não supervisionado, enquanto a Fig.5 mostra a análise entre as técnicas e tarefas associadas a elas. Por sua vez, a supervisionada foi bem-sucedida nas tarefas de classificação de dados brutos, classificação de séries temporais e regressão (NACCHIA et al., 2021).

## 2.3 APRENDIZADO DE MÁQUINA

O termo *machine learning* ou aprendizado de máquina já foi mencionado algumas vezes ao longo deste trabalho. Podemos caracterizá-lo como um subgrupo da ciência da computação que se concentra em criar algoritmos que, com a finalidade de serem úteis, necessitam de um conjunto de exemplos de algum fenômeno, onde esses exemplos podem ser gerados por outros algoritmos ou processados à mão por seres humanos (BURKOV, 2020). Também pode ser definido como o processo de resolver um problema prático, realizando:

- Coleta de dados;
- Realizar algoritmicamente o treinamento de um modelo estatístico com base num conjunto de dados.

Vale ressaltar que quanto maior a amostra de dados, mais fácil será a tarefa, que também depende da qualidade dos rótulos (anotações que identificam e categorizam dados brutos) atribuídos aos documentos na amostra. Logo, o sucesso de um algoritmo está intrinsecamente ligado aos dados utilizados, à subsequente análise dos mesmos e à sua respectiva distribuição estatística (MOHRI; ROSTAMIZADEH; TALWALKAR, 2012).

### 2.3.1 Classificações de ML

Já foram citadas algumas utilidades para a indústria (especialmente para a categoria 4.0), onde podemos realizar a manutenção preditiva em sistemas com valores agregados grande. Entretanto, vale ressaltar alguns outros usos, que incluem (MOHRI; ROSTAMIZADEH; TALWALKAR, 2012):

- Classificação de textos documentos (como detecção de spam em e-mails), ou classificando tópicos a um texto ou documento.
- Processamento de linguagem natural, que tem como intuito realizar a análise de classes gramaticais, reconhecimento de entidades nomeadas, que são utilizadas nos softwares de tradução, assistentes virtuais e mecanismos de busca.
- Aplicações de processamento de fala, isso inclui reconhecimento e síntese de fala.
- Aplicações de visão computacional, ou seja, reconhecimento de objetos, identificação de objetos e detecção de rosto.
- Biologia computacional, que tem como objetivo prever funções de proteínas e analisar genes e redes proteicas.

Todavia, não são os mesmos algoritmos que solucionam esses problemas, principalmente pelo fato de estarem alocados em tipos de adversidade diferentes, que podem ser agrupados no seguinte formato (MOHRI; ROSTAMIZADEH; TALWALKAR, 2012):

- Classificação: problemas que precisam atribuir uma categoria a cada item, podendo ser distribuídos em subgrupos como classificar imagens, categorizar o que é cada objeto (exemplos, carros, trem avião, cachorro ou gato), ou classificação de documentos, para por exemplo separar matérias pelos assuntos, como politica, esportes ou clima.
- Regressão: são problemas de previsão de valores reais, os exemplos clássicos são a previsão de variações na economia e valores de ações.
- *Clustering:* tem como foco particionar o grupo de dados em subconjuntos homogêneos. Um exemplo de uso consiste em encontrar nas redes sociais comunidades naturais dentro de grandes grupos de pessoas.

Além de classificarmos os modelos de aprendizado de máquina pelos problemas que eles resolvem, podemos alocá-los pelas formas que eles são ensinados e classificá-los de quatro maneiras (BURKOV, 2020):

- Supervisionado;
- Semi supervisionado;
- Não-supervisionado;
- Por reforço.

Embora outros autores considerem um número maior de tipos de aprendizado, este trabalho focará apenas naqueles que serão abordados ao longo do texto ou que apresentam um contraste direto com os utilizados.

### 2.3.2 Aprendizado Supervisionado

Segundo a IBM, o aprendizado supervisionado é a técnica que utiliza um conjunto de dados rotulados para treinar modelos de algoritmos de inteligência artificial (IBM, 2025), sendo essa sua característica mais evidente do aprendizado supervisionado são os dados serem rotulados, $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$ ou seja, cada valor de uma variável possui uma outra variável que o complementa. Esse objeto chamaremos de vetor de características (ou do inglês, *feature vector*) é uma representação numérica estruturada de um exemplo, onde cada posição $j$ no vetor ($x_i^{(j)}$) armazena uma característica específica, como:

- $x_1^{(1)}$: massa (em kg),
- $x_1^{(2)}$: força (em Newtons),
- $x_1^{(3)}$: idade (em anos).

(BURKOV, 2020).

Uma determinada posição no vetor deve representar a mesma característica para todos os exemplos (ex.: $x_k^{(1)}$ sempre corresponde à massa, para todo $k$).

Como as características podem ser números, ou qualquer tipo de elementos, com classes, é possível separar os problemas de previsão que serão resolvidos. Se ele prediz um número, ela será uma regressão, enquanto se for uma classe o problema é chamado de classificação. Um exemplo de regressão é a predição dos valores de casas com relação ao número de quartos: neste caso, nosso alvo será o preço. As Figs. 6 e 7 temos os exemplos para os resultados de cada uma delas.

Na regressão temos uma linha ou hiperplano que tenta se adequar o melhor possível aos dados indicados, visando minimizar os erros da predição. Em contrapartida, a classificação

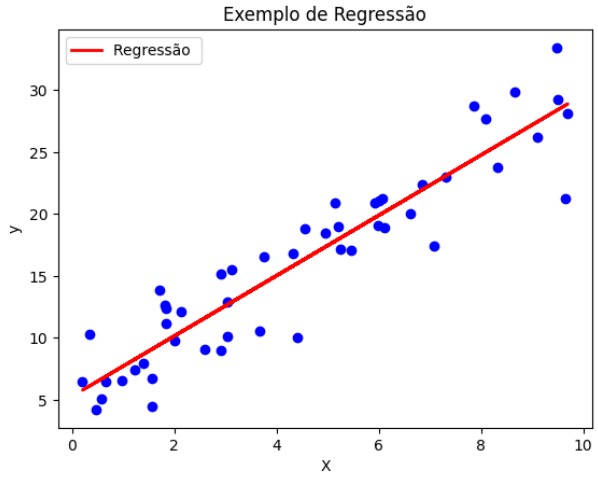



Figura 6 – Exemplo de gráfico de regressão adaptado de (BURKOV, 2020). Fonte: Autor.

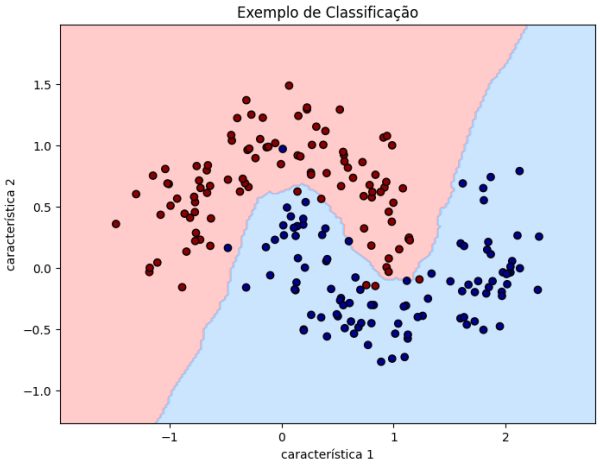



Figura 7 – Exemplo de gráfico de classificação baseada em (BURKOV, 2020). Fonte: Autor.

tem como objetivo encontrar a linha ou hiperplano que melhor divide os dados, classificando-os em grupos.

Consequentemente, o objetivo desse modo de aprendizado é criar um modelo a partir de um conjunto de dados rotulados, possibilitando que ele faça a união de um vetor de características (de entrada $x_1$) a um rótulo. Em outras palavras, treinar o modelo para reconhecer padrões e efetuar previsões com base em exemplos anteriores (BURKOV, 2020).

Ao construir um modelo de aprendizado supervisionado, surgem questões importantes: qual é o melhor modelo para o meu problema? Como saber a qual classe um novo exemplo pertence? Para resolver esses desafios, alguns algoritmos podem ser utilizados, cada um com uma abordagem única. Podemos citar:

- **Naive Bayes:** é um modelo de classificação que adota o princípio de independência condicional de classe do teorema de Bayes. Normalmente utilizada em classificação de texto e identificação de spam (IBM, 2025).

- **Regressão Linear:** empregado para encontrar uma relação linear entre as variáveis independente e dependente para predições. Se houver apenas uma variável independente, a regressão é denominada simples; caso haja mais de uma, é chamada de múltipla. (IBM, 2025).
- **Regressão Polinomial:** assim como a linear tem como objetivo encontrar as relações entre as variáveis, mas em vez de ajustar por uma função linear se ajusta por uma polinomial (IBM, 2025).
- **Máquina de Vetores de Suporte:** usada tanto para classificações quanto para regressões, com um foco maior para a primeira, e tem como objetivo visualizar o hiperplano que maximiza a distância entre os grupos de pontos de dados (IBM, 2025).
- ***K-Nearest Neighbor:*** é um algoritmo não paramétrico, ou seja, não faz suposições sobre a forma da função que está sendo aprendida, sendo mais flexível e podendo se ajustar melhor aos dados, sem possuir um número fixo de parâmetros. Ele se baseia em sua proximidade e associação a outros dados disponíveis (IBM, 2025).
- ***Random forest:*** também usado tanto para classificação quanto para regressão, o *forest* se deve ao fato de usar diversas arvores de decisão para fazer a melhor escolha possível na situação (IBM, 2025).

### 2.3.3 Aprendizado Não Supervisionado

Existem desafios que envolvem dados que não possuem rótulos, e para solucioná-los utilizando aprendizado de máquina temos que adotar outra abordagem, e é aí que surge o aprendizado não supervisionado. Esses algoritmos possuem a função de encontrar padrões e agrupar dados, sem que o ser humano diga o que é cada dado (IBM, 2025).

Seguindo a lógica dos vetores de características novamente, podemos dizer que o objetivo desses modelos é buscar um vetor $x$ como entrada e transformá-lo em outro vetor ou em algum tipo de informação que pode servir para resolver um problema prático.

### 2.3.4 Aprendizado Semi Supervisionado

Existe ainda um aprendizado que vai trabalhar tanto com dados rotulados quanto com não rotulados, o objetivo principal é similar ao aprendizado supervisionado. A proporção de elementos não rotulados, normalmente, acaba sendo maior. O aprendizado semi supervisionado necessita fazer suposições para os dados não rotulados e como os pontos de dados de diferentes classes se relacionam entre si(BERGMANN, 2023).

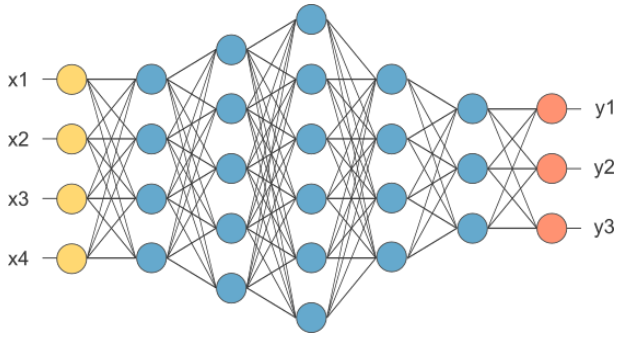



Figura 8 – Exemplo de uma rede neural artificial. Fonte: Adaptado de UFOP, 2025.

### 2.3.5 Aprendizado Por Reforço

No aprendizado por reforço, a máquina ou agente é criada dentro de um ambiente no qual ela consegue perceber os estados desse espaço como se fossem vetores de características e, a partir disso, pode tomar decisões em estados que não sejam terminais. Ele possui como objetivo principal escolher uma política ótima. Essa função recebe o vetor de características e gera uma ação, buscando sempre maximizar sua eficiência (BURKOV, 2020).

Diferentemente do treinamento supervisionado, este método não possui uma resposta fornecida diretamente para a máquina referente à assertiva correta. Por meio de tentativa e erro, deve aprender quais escolhas são as melhores, recebendo recompensas ao longo das atividades realizadas. Se a resposta estiver correta, recebe uma recompensa positiva; se não, recebe uma negativa, e a partir disso vai aprendendo quais são as melhores escolhas a serem tomadas, tentando sempre maximizar a recompensa (SUTTON; BARTO, 2018).

Aqui surge um problema de *trade-off*, ou seja, qual ação ele deve escolher ou priorizar entre *exploration* e *exploitation*, onde, o primeiro tem como papel criar novos conhecimentos quando prevê soluções novas, enquanto o segundo executa suas ações com base na combinação de soluções passadas. Para otimizar suas recompensas, o algoritmo deveria testar combinações que já encontraram boas respostas no passado, mas para determiná-las é necessário testar novas escolhas e aprender com elas, e assim, progredir gradativamente na busca da respectiva.

## 2.4 REDES NEURAIS ARTIFICIAIS

O conceito de redes neurais artificiais consiste em uma técnica de aprendizado de máquina que possui como objetivo simular uma rede neural biológica, onde neurônios interconectados processam informações de forma distribuída a partir das respectivas conexões existentes entre cada um. Essas células são representadas por unidades computacionais que se conectam entre si por pesos e limiares, os quais possuem como função simular as forças de conexão das sinapses biológicas. Como ilustrado na Fig. 8 (AGGARWAL, 2018).

As RNAs são compostas por nós (ou neurônios artificiais) e divididas em camadas, sendo elas (SILVA; SPATTI; FLAUZINO, 2010):

- Camada de entrada: é responsável por receber os dados, de uma forma geral já normalizados, advindos do meio externo;
- Camadas ocultas: são as camadas que são compostas por neurônios, são nelas que acontecem as operações e processamento interno;
- Camada de saída: também são compostas de neurônios, mas dessa vez produzem e apresentam os dados finais.

A passagem dos dados entre as camadas ocorre com base em um limiar predefinido. Se a saída do neurônio ultrapassar esse limiar, o sinal é transmitido para a próxima camada. Se não for ativado, o dado não percorre o caminho, assim como acontece com as nossas sinapses (IBM, 2025). Para aprender, elas necessitam de treinamento, assim como todos os outros modelos de aprendizado de máquina a partir de dados.

O funcionamento de cada nó possui um modelo de regressão linear próprio que irá carregar os pesos e os respectivos limiares, e responderá com uma saída que depende da entrada fornecida, dos pesos, do viés (ou limiar) e da própria saída, sendo representada matematicamente por:

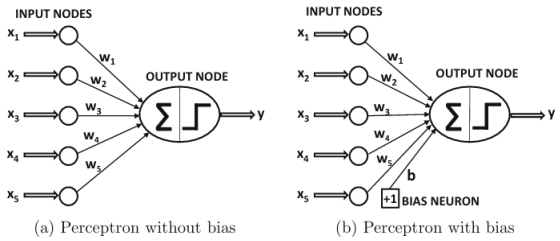
$$\sum w_1 x_1 + b = w_1 x_1 + w_2 x_2 + w_3 x_3 + b$$

. Onde os w são os pesos, b o viés e os x são as entradas, quanto maiores forem os pesos, mais importantes serão as informações das entradas, e o viés é responsável por ajustar o comportamento da função. Mas ela não depende somente disso para funcionar, ainda precisamos de uma função de ativação que fará com que o neurônio dispare para a próxima camada. Um exemplo seria a função degrau descrita a seguir (IBM, 2025):

$$f(x) = \begin{cases} 1, & \text{se } \sum w_i x_i + \text{bias} \geq 0. \\ 0, & \text{se } \sum w_i x_i + \text{bias} < 0. \end{cases}$$

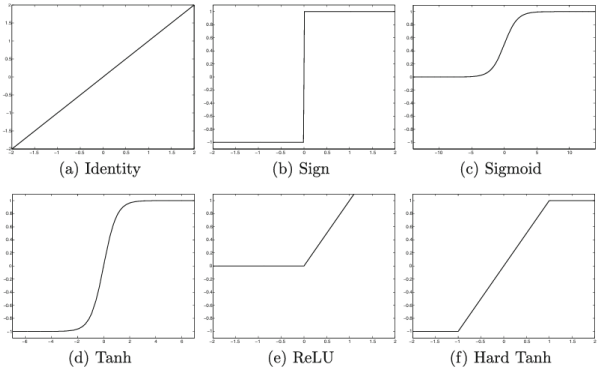
Caso a soma seja maior ou igual a zero, a função será ativada e o neurônio irá disparar. Se ela for menor do que zero, o neurônio não a será ativado, como mostrado na Fig. 9. Em redes modernas utilizamos funções não lineares como **ReLU** ou **Sigmoid** como mostrado na Fig.10 com o propósito de introduzir um padrão não-linear. Sem isso, a rede seria apenas uma combinação linear de entradas e seria incapaz de aprender padrões complexos (AGGARWAL, 2018).

Outro desafio era a dificuldade em resolver problemas não separáveis linearmente. Para contornar essa limitação, Rumelhart, Hinton e Williams desenvolveram o algoritmo de treinamento backpropagation que permitiu ajustar os pesos com base nos erros calculados para cada amostra de treinamento. O processo ocorre em duas fases principais (RUMELHART; HINTON; WILLIAMS, 1986):




(a) Perceptron without bias (b) Perceptron with bias

Figura 9 – Exemplo de um funcionamento de um neurônio artificial ou perceptron. Fonte: AGGARWAL, C. C. Neural Networks and Deep Learning.



(a) Identity (b) Sign (c) Sigmoid

(d) Tanh (e) ReLU (f) Hard Tanh

Figura 10 – Exemplo de funções de ativação modernas. Fonte: AGGARWAL, C. C. Neural Networks and Deep Learning.

- **Propagação direta *(Forward Pass)***: Os dados de entrada passam pela rede, camada por camada, até a saída. Nesse momento, a predição do modelo é comparada ao valor real por meio de uma função de perda.
- **Propagação reversa *(Backward Pass)***: Neste caso, o erro na saída é retropropagado para ajustar os pesos da rede, utilizando o algoritmos de otimização.

A retropropagação funciona aplicando a Regra da Cadeia,a qual é derivada para calcular como cada peso contribui para o erro final. Outro componente essencial para as redes neurais são as funções de perda (*loss function*, em inglês), que são responsáveis por medir o erro entre as previsões do modelo e os valores reais, permitindo que o modelo aprenda ajustando seus parâmetros para minimizar o referido erro. Existem diversos tipos de funções de perda com funções específicas para cada tipo de problema (AGGARWAL, 2018).

Para otimizar a *loss function* podemos utilizar um algoritmo de otimização, sendo o mais conhecido o Gradiente Descendente. Ele possui o objetivo de ajustar os pesos da rede neural para minimizar a função de perda. Ele realiza esse processo calculando o gradiente da função de perda em relação aos pesos do modelo e, em seguida, atualizando esses pesos na direção oposta. (RUMELHART; HINTON; WILLIAMS, 1986). Mas já existem derivações desse algoritmo, sendo um exemplo o otimizador Adam (Adaptive Moment Estimation), que é uma versão melhorada do Gradiente Descendente, combinando (KINGMA; BA, 2014):

- Momentum → ajuda a suavizar as atualizações dos pesos.
- RMSProp → ajusta a taxa de aprendizado com base na variância dos gradientes.

Esse funcionamento das redes é conhecido como redes *feedforward*, além delas existem outras que trabalham de formas um pouco diferentes com o intuito de melhorar em pontos específicos que *feedforward* não consegue realizar com maestria.

### 2.4.1 Redes Neurais Recorrentes (RNNs)

Redes Neurais Recorrentes representam uma classe de redes neurais que possuem o objetivo de processar dados sequenciais. São exemplos texto, áudio e séries temporais que, diferentes das redes neurais citadas anteriormente, possuem uma memória, ou seja, elas podem "lembrar"de passos anteriores, com isso, elas conseguem captar padrões e modelar dependências temporais (GOODFELLOW; BENGIO; COURVILLE, 2016).

Outro aspecto importante é o fato de as camadas terem "acesso"aos parâmetros das outras, onde o treinamento é feito com todas as camadas tendo os mesmos pesos, que são ajustados pelos processos de retropropagação e gradiente descendente, utilizando a propagação direta e retropropagação através do tempo. Ela faz isso pelo fato de lidar com sequências que podem acabar saindo das posições corretas ou não serem vistas, e dessa forma, ela consegue tratar séries variáveis e aprender padrões independentemente da posição em que aparecem (IBM, 2024).

Assim como em todas as redes neurais, as recorrentes possuem diversas variantes; dentre as quais podemos citar:

- RNNs padrão;
- Redes neurais recorrentes bidirecionais;
- Memória de curto longo prazo (LSTM).

### 2.4.2 Long Short-Term Memory (LSTM)

Um dos problemas que ocorre em redes neurais que trabalham com sequências temporais é o fenômeno conhecido como *vanishing gradient*, ou gradiente que desaparece. Ele ocorre quando os sinais de erro que são retropropagados durante o treinamento se tornam extremamente pequenos, o que dificulta atualização dos pesos da rede de forma correta. Esse erro complica o aprendizado das dependências de longo prazo (HOCHREITER; SCHMIDHUBER, 1997).

Essa complicação fez com que Sepp Hochreiter e Jürgen Schmidhuber desenvolvessem um algoritmo para resolver esse conflito, e com isso chegaram ao Long Short-Term Memory. Para isso eles implementaram uma série de soluções, sendo elas:

- **Células de Memória:** armazenam informações ao longo de longos períodos de tempo;

- **Fluxo de Erro Constante:** o algoritmo projeta um fluxo de erro constante através de conexões específicas (chamadas de carrosséis de erro constantes);
- **Portões Multiplicativos:** Esses portões controlam o que é adicionado à célula de memória, o que é mantido e o que é utilizado na saída.
- **Truncamento da Retropropagação:** O algoritmo aplica um método de retropropagação truncada, que corta o fluxo do gradiente em determinados pontos da rede, que ajuda a impedir que ocorram os efeitos de *vanishing gradient*.

Os portões, que constituem grande parte da estrutura do algoritmo, podem ser classificados em três principais categorias:

- **Portão de Esquecimento:** decide quais informações devem ser descartadas da célula de memória. Recebendo a entrada atual $x_t$ e o estado oculto anterior $h_{t-1}$, e passando essas informações por uma ativação sigmoide:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f).$$

- **Portão de Entrada:** controla quais novas informações devem ser armazenadas na célula de memória, utilizando novamente uma ativação sigmoide:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i).$$

Além disso, há um novo vetor candidato de memória $\tilde{C}_t$ calculado usando uma ativação tangente hiperbólica:

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C).$$

O estado da célula, responsável por manter as informações a longo prazo, é atualizado por meio de:

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t.$$

- **Portão de saída:** define a parte da célula de memória que será usada para calcular o estado oculto atual $h_t$. Esse gate também passa por uma ativação sigmoide:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o).$$

O estado oculto é então atualizado com:

$$h_t = o_t \cdot \tanh(C_t).$$

Todos esses portões podem ser vistos em ordem no modelo da Fig. 11

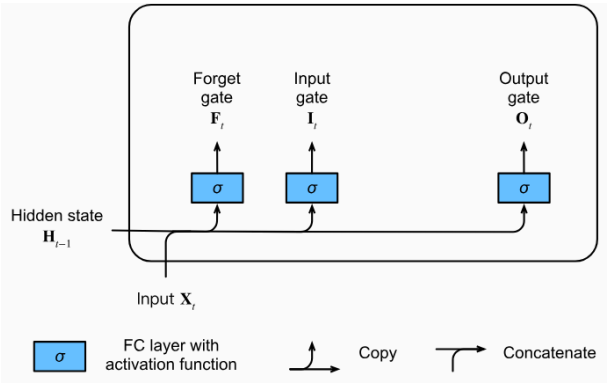



Figura 11 – Esquema dos portões de uma rede neural do tipo LSTM. Fonte: (LEARNING, 2025).

### 2.4.3 Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) é uma técnica que combina os princípios do aprendizado por reforço (RL) com as capacidades de modelagem de redes neurais profundas (deep learning). Em essência, o DRL permite que um agente aprenda a tomar decisões em um ambiente dinâmico, utilizando redes neurais para estimar funções de valor ou políticas que maximizam recompensas ao longo do tempo (NAYERI; GHAFARIAN; JAVADI, 2021).

DRL é um método que envolve um agente que interage com um ambiente ao realizar ações e recebendo recompensas, utilizando redes neurais profundas para aproximar a função de valor ou política. O objetivo é maximizar a recompensa acumulada ao longo de uma sequência de decisões, sendo capaz de aprender com experiências passadas de forma adaptativa e em tempo real, mesmo em espaços de estado e ação complexos (NAYERI; GHAFARIAN; JAVADI, 2021).

## 2.5 INTRODUÇÃO A COMPUTAÇÃO QUÂNTICA

A grande maioria das redes neurais, dependendo do número de dados que devem ser processados, há uma demanda de recursos computacionais consideravelmente elevada (tanto para a CPU como para a GPU) para que seja possível realizar essas tarefas em escalas de tempo viáveis. Porém, problemas complexos interagem de uma maneira complicada e muitos computadores não conseguem solucioná-los.

A computação quântica surge como uma alternativa promissora para resolver problemas que os computadores clássicos não conseguem solucionar de maneira eficiente. Uma forma de entender isso é olhar primeiramente para a primeira grande diferença entre a computação clássica e a quântica: a unidade básica de processamento de informação. Na computação convencional trabalhamos com bits que podem variar entre 0 e 1, enquanto que, para a computação quântica, temos os bit quânticos (denominados qubits) (NIELSEN; CHUANG, 2010).

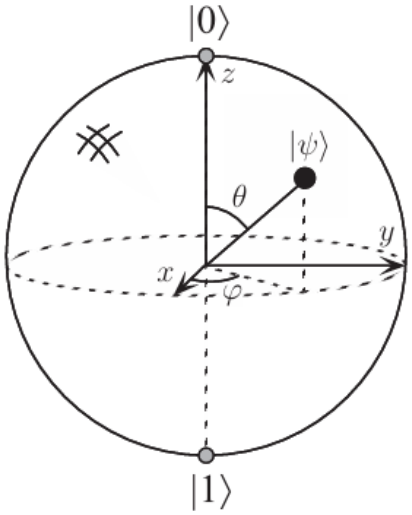



Figura 12 – Representação de um qubit na esfera de Bloch Fonte: NIELSEN; CHUANG, 2010

Assim como na computação clássica, representamos os estados por $0$ e $1$. No entanto, adotamos a notação $|0\rangle$ e $|1\rangle$, respectivamente. A grande diferença está no fato de que um sistema quântico também pode estar em um estado de superposição, ou seja, em uma combinação linear dos estados $|0\rangle$ e $|1\rangle$, que pode ser escrita como:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle,$$

onde $\alpha$ e $\beta$ são coeficientes complexos que obedecem à condição de normalização dada por $|\alpha|^2 + |\beta|^2 = 1$.

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle,$$

onde $\alpha$ e $\beta$ são números complexos (MCMAHON, 2007).

Uma das maneiras intuitivas de entender os qubits é na forma geométrica dada pela esfera de Bloch, mostrada na Fig.12, onde o polo norte é dado pelo $|0\rangle$ e o polo sul é representado por $|1\rangle$, enquanto qualquer outra posição na esfera será uma combinação linear entre esses dois vetores.

Além dos qubits, os computadores quânticos usam outras propriedades, sendo elas:

- **Superposição:** fenômeno que permite que um sistema quântico esteja em mais de um estado ao mesmo tempo. (desde que não haja interação com um ambiente externo ou que o mesmo não seja observado).
- **Emaranhamento:** é o fenômeno que ocorre quando partículas interagem de modo que as propriedades de cada uma dependem das configurações assumida pelas demais.
- **Interferência:** as amplitudes de probabilidades associadas a um determinado estado quântico podem ser modificadas através de combinações lineares envolvendo outras funções de onda.

## 2.6 PORTAS LÓGICAS QUÂNTICAS

As operações que atuam nos qubits inseridos nos circuitos quânticos são realizadas através das portas lógicas quânticas, que são representadas por matrizes unitárias. Diferentemente das portas clássicas, as portas quânticas precisam ser reversíveis, ou seja, suas operações podem ser desfeitas através da aplicação das respectivas portas lógicas inversas (MERMIN, 2007).

- **Portas de Pauli (X, Y, Z):** A porta Pauli-X equivale à operação NOT clássica:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix},$$

  enquanto as portas Pauli-Y e Pauli-Z realizam rotações quânticas.

- **Porta Hadamard (H):** Responsável por criar superposição ao transformar $|0\rangle$ e $|1\rangle$ em combinações lineares:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

  Aplicando $H$ ao estado $|0\rangle$, temos:

$$H|0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}}.$$

- **Porta CNOT (Controlled-NOT):** Opera em dois qubits e realiza uma inversão condicional:

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix},$$

- **Portas de Fase (S, T):** Introduzem mudanças de fase no estado quântico, alterando suas propriedades probabilísticas.

- **Portas de Rotação ($R_x, R_y, R_z$)**: responsáveis por realizar rotações quânticas arbitrárias na esfera de Bloch ao redor dos eixos $X$, $Y$ e $Z$, definidas por (BENENTI; CASATI; STRINI, 2004):

$$R_x(\theta) = \begin{bmatrix} \cos(\theta/2) & -i\sin(\theta/2) \\ -i\sin(\theta/2) & \cos(\theta/2) \end{bmatrix}.$$

$$R_y(\theta) = \begin{bmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{bmatrix}.$$

$$R_z(\theta) = \begin{bmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2}. \end{bmatrix}$$

Essas operações permitem controlar a evolução do estado quântico ao longo dos eixos da esfera de Bloch (ALVES; GOMES; CURY, 2025).

- **Porta de Rotação Arbitrária ($U_3$)**: combina rotações nos três eixos para permitir qualquer transformação de um único qubit (BENENTI; CASATI; STRINI, 2004):

$$U_3(\theta, \phi, \lambda) = \begin{bmatrix} \cos(\theta/2) & e^{i\lambda}\sin(\theta/2) \\ e^{i\phi}\sin(\theta/2) & e^{i(\phi+\lambda)}\cos(\theta/2) \end{bmatrix}.$$

Isso garante a universalidade das operações de um qubit, permitindo a implementação de qualquer porta unitária (ALVES; GOMES; CURY, 2025).

## 2.7 QUANTUM MACHINE LEARNING (QML)

O aprendizado de máquina quântico é uma área de estudo que une o aprendizado de máquina clássico (com todas as características já citadas nesse trabalho) e outras, com a computação quântica, com o intuito de descobrir se é possível aumentar a velocidade de aprendizado ou a precisão dos modelos, e se são viáveis para obter previsões (PENNYLANE, 2023).

No entanto, os qubits são instáveis, pois seu tempo de coerência (o período em que mantêm seu estado ) é curto, o que representa um grande desafio. Por isso, utilizamos combinações com otimizadores clássicos para resolver problemas. Uma dessas soluções são os circuitos variacionais quânticos (em inglês, VQCs). Eles são chamados de "variacionais"porque dependem de um conjunto de parâmetros ajustáveis que são otimizados iterativamente para minimizar ou maximizar uma função (GRIOL-BARRES et al., 2021).

Seu funcionamento, de modo geral, segue os seguintes passos:

1. Os dados são codificados em qubits aplicando portas quânticas a um conjunto para representar a informação que queremos processar;
2. Um circuito parametrizado manipula esses qubits com operações quânticas ajustáveis, esse circuito consiste em uma sequência de operações quânticas (portas quânticas) que dependem de parâmetros ajustáveis;
3. Segue para a medição, colapsando em valores clássicos;
4. Logo após um algoritmo clássico analisa os resultados e ajusta os parâmetros do circuito para melhorar a performance.

Os VQCs são úteis para aprendizado de máquina e otimização, aproveitando a superposição e o emaranhamento quântico para explorar um grande espaço de soluções.

## 2.8 QUANTUM LSTM (QLSTM)

O Quantum LSTM é o algoritmo análogo quântico do Long Short-Term Memory que também possui o intuito de aprender dados sequenciais, mas dessa vez com aprendizado de máquina quântico. Tendo em vista a ideia de executar um QLSTM foi realizado a substituição das redes neurais clássicas nas células LSTM por VQCs, que seriam responsáveis por toda a extração de dados e o aprendizado. (CHEN; YOO; FANG, 2020).

Com isso, teríamos três camadas:

- **Camada de Codificação de Dados:** é responsável por codificar os dados clássicos em estados quânticos. Os dados de entrada são convertidos em superposições quânticas, garantindo que as informações sejam carregadas e representadas no espaço quântico.
- **Camada Variacional:** os estados quânticos passam por uma série de operações unitárias. Essas operações incluem portas de rotação em um único qubit e portas CNOT para gerar emaranhamento entre qubits.
- **Camada de Medição Quântica:** no final de cada bloco de circuito variacional, a camada de medição realiza medições dos qubits no estado computacional. Os valores de expectativa dos qubits são coletados, e esses resultados são processados em um computador clássico para obter previsões

# 3 Objetivos

## 3.1 Objetivos gerais

Desenvolver e comparar modelos de previsão de temperatura de saída dos gás hidrogênio ($H_2$) em uma planta de hidrogênio, utilizando abordagens clássicas (LSTM) e quânticas (QLSTM), com e sem Deep Reinforcement Learning (DRL), para identificar a melhor arquitetura em termos de precisão e eficiência.

## 3.2 Objetivos específicos

- Treinar modelos LSTM (sem DRL e com DRL) para prever a temperatura de saída de $H_2$.
- Avaliar o desempenho dos modelos usando métricas como RMSE, MAE e MAPE.
- Treinar modelos QLSTM (sem DRL e com DRL) para prever a temperatura de saída de $H_2$.
- Comparar o desempenho com os modelos LSTM em termos de precisão e tempo de treinamento.
- Comparar os resultados dos modelos LSTM e QLSTM, com e sem DRL, para identificar qual abordagem oferece o melhor desempenho.
- Analisar a generalização dos modelos (diferença entre métricas de treino e validação) para afastar a possibilidade de ter ocorrido *overfitting* ou *underfitting*.

# 4 DESENVOLVIMENTO

## 4.1 Metodologia

Todas as atividades foram realizadas em um computador de uso pessoal, cujo dispositivo corresponde a um Acer Nitro 5 com as seguintes especificações: Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz, 2.81 GHz, 16,0 GB de RAM e NVIDIA GeForce GTX 1050 Ti, e todos os algoritmos realizados podem ser encontrados nos apêndices do trabalho.

Os dados foram fornecidos pela equipe do Centro de Tecnologias de Hidrogênio do Itaipu Parquetec e correspondem aos dados de operação entre janeiro de 2023 e novembro de 2024. Sendo eles:

- Temperatura_in_A_[ºC];
- Temperatura_out_H2_A_[ºC];
- Corrente_A_[A];
- Tensao_A_[V];
- Potencia_A_[kW];
- PowerSet_[%];
- Pressão_Buffer_Int_[bar];

Entretanto como são dados sensíveis não é possível especificar mais sobre eles nem dar maiores detalhes de como se comportaram e qual a quantidade de dados que foram disponibilizados.

O atual trabalho foi baseado no artigo *"A Novel Deep Reinforcement Learning (DRL) Algorithm to Apply Artificial Intelligence-Based Maintenance in Electrolysers"*, fazendo a mudança da linguagem do algoritmo de MatLab para Python, além da mudança do tipo do eletrolisador, do tipo PEM para Alcalino (ABIOLA; MANZANO; ANDúJAR, 2023). Com um acréscimo da utilização do QLSTM para comparar como técnicas de QML poderiam beneficiar na precisão ou velocidade de previsão.

### 4.1.1 Pré-Processamento

A primeira etapa do trabalho consistiu no tratamento dos dados e na análise de seu comportamento. Para isso, foram realizadas as seguintes tarefas:

- Limpeza dos dados;

- Análise de correlação;
- Treinamentos de Regressões Lineares;
- Treinamento de Regressões Polinomiais;
- Análise dos dados em relação ao tempo;
- Análise dos dados por *Power Set*;
- Normalização dos dados.

#### 4.1.1.1 Limpeza dos dados

Como os dados foram previamente tratados por outra equipe, antes de serem utilizados no trabalho, foram removidos os dias em que os sensores da planta registraram informações, mas nenhuma operação concreta ocorreu. Além disso, foi realizada uma segunda checagem para garantir a consistência dos dados.

Com as bibliotecas do *Pandas* foi possível exercer a importação dos dados (que foram disponibilizados em formato .csv). Com isso, separamos somente as colunas do Buffer A da planta, para que trabalhássemos com menos variáveis. Além disso, utilizamos as funções *"dropna"e "dropna duplicate"* para excluir os valores nulos e duplicados de todas as colunas e linhas. (TEAM, 2024)

#### 4.1.1.2 Análise de correlação

A análise de correlação foi implementada de duas maneiras distintas, a primeira foi executada usando dois parâmetros para a análise, um deles sendo a métrica $R^2$ e o outro Coeficiente de Pearson, enquanto a segunda abordagem foi utilizando o mapa de calor.

Para o primeiro passo de análise do comportamento dos dados, foi realizada uma avaliação de linearidade entre todas as variáveis uma a uma, sem repetir os grupos (x,y) e (y,x) já que a relação tem que ser a mesma, o que nos resultou em 23 gráficos distintos.

Como o Coeficiente de Pearson foi realizado junto, também temos quais eram os dados com maior influência uns aos outros. Com os valores de correlação linear, foram separados em dois grupos de análise. O primeiro foi selecionado com base em valores de coeficiente linear $R^2$ superiores a 0,5, o que mostraria uma certa condição linear.

Em seguida, o segundo grupo foi separado para os valores abaixo de 0,5, já que possuiriam uma menor condição linear. Para o mapa de calor foi realizado a partir da função *df.corr()* e plotando usando o *sns.heatmap*, com isso a visualização da correlação se tornou bem mais simples.

#### 4.1.1.3 Treinamento de Regressão Linear

A partir desse ponto, iniciaram-se os treinamentos de modelos de Machine Learning, ainda com o objetivo de compreender o comportamento dos dados. Os treinamentos foram realizados com 80% do total de dados disponibilizados, sendo esse grupo utilizado para o treinamento de cada um dos modelos subsequentes. Já para o teste, foi utilizado 20% dos dados para testar os modelos e validá-los. Com o primeiro grupo, foi realizado o treinamento de regressão linear para se obter a função que regia o comportamento dessas variáveis (PEDREGOSA et al., 2011). Para essa finalidade foi realizada a importação dos dados e logo após executada a separação dos dados para teste e treino utilizando a função *train_test_split()* e logo após definido o modelo de regressão utilizando a *linear_model.LinearRegression()*, além de calcular a predição utilizando *regr.predict()* para fazer a predição e depois calcular o erro médio absoluto (PEDREGOSA et al., 2011).

Nesse grupo, executamos ao final 7 gráficos entre os seguintes grupos de variáveis:

1. Temperatura_out_O2_A_[ºC] e Temperatura_in_A_[ºC]
2. Temperatura_in_A_[ºC] vs Temperatura_out_H2_A_[ºC]
3. emperatura_out_H2_A_[ºC] vs Temperatura_out_O2_A_[ºC]
4. Corrente_A_[A] e Tensao_A_[V]
5. Corrente_A_[A] e Potencia_A_[kW]
6. Corrente_A_[A] e PowerSet_[
7. Corrente_A_[A] e Pressão_Buffer_Int_[bar]

#### 4.1.1.4 Treinamento de Regressão Polinomial

O segundo grupo foi treinado com uma regressão polinomial. Para isso, primeiramente foi feita a escolha do melhor polinômio para cada caso. Para esse objetivo, foi executada uma função *for* junto da *PolynomialFeatures* e *polinomial.fit_transform(x)* que criam os graus polinomiais e transformam os dados; logo após, é realizada uma regressão linear com *LinearRegression()* (PEDREGOSA et al., 2011). Isso é feito para os primeiros graus; com os valores de $R^2$, foram escolhidas as melhores métricas.

Para completar, foi realizado o treinamento com os melhores graus polinomiais para cada caso, utilizando novamente a função *regr.predict()* utilizando novamente a métrica de erro absoluto médio.

### 4.1.2 Análise dos dados por Power Set

Depois de analisar as regressões polinomiais, fomos entender o comportamento dos dados em relação ao *Power Set* estabelecido para cada etapa da produção de hidrogênio. Para

isso, foi feito um filtro para separar todos os dados em grupos correspondentes aos respectivos valores de *power set*, que correspodem respectivamente a 30, 50, 60 e 70. E, por fim, plotados os gráficos de cada grupo.

### 4.1.3 Análise dos dados em relação ao tempo

Além da análise de *power set*, a regressão polinomial também deu *insights* de como deveriam se comportar as variáveis em relação ao tempo; para essa finalidade, foi feito um plot simples dos dados em relação aos passos, ou coletas de dados (eram coletados dados a cada 15 segundos). Para que ficasse mais simples, foi separada a coluna das datas e horas e padronizado o formato dela para dia/mês/anos e horas/minutos/segundos utilizando a função *pd.todatetime*.

#### 4.1.3.1 Normalização dos dados

Para evitar enviesamento nos dados, causado por características com amplitudes significativamente maiores do que outras, foi realizada a padronização dos dados por meio do *StandardScaler()*. Esse método aplica a padronização conforme a equação do Z-score:

$$z = \frac{x - \mu}{\sigma}.$$

Onde:

- x:: Valor original da variável.
- $\mu$: Média da variável (calculada a partir dos dados).
- $\sigma$ : Desvio padrão da variável (também calculado a partir dos dados).

A padronização transforma os dados para terem média 0 e desvio padrão 1. Depois, foi realizada a plotagem do gráfico com as variáveis padronizadas para poder ver se estavam da maneira esperada. Existem outro métodos de realizar normnalizações, mas foi escolhido utilizar a padronização para seguir os mesmos passos do trabalho Abiola, A. et al.

### 4.1.4 Treinamento LSTM

Foram selecionadas quatro variáveis de entrada: Temperatura_in_A_[$\check{z}C$], Temperatura_out_O2_A_[ºC], PowerSet e Pressão_Buffer_Int_[bar]. Já a variável alvo foi definida como Temperatura_out_H2_A_[ºC]. A escolha das características de entrada no treinamento inicial foi baseada na correlação direta com a temperatura de saída do hidrogênio, sendo selecionadas aquelas com valores acima de $0,7$.

A estrutura dos dados foi reorganizada para formar um dataset de séries temporais, no qual cada amostra correspondeu a um conjunto de 100 pontos temporais (coletados a cada 15

segundos) utilizados para prever o valor alvo subsequente. Uma classe personalizada chamada *TimeSeriesDataset* foi criada para organizar os dados, ela tem como objetivos:

- Define o comprimento das sequências temporais (100 pontos de entrada para cada predição);
- Retorna um tensor contendo os valores normalizados das variáveis de entrada e a saída correspondente.

Os dados foram divididos em conjuntos de treinamento (80%) e validação (20%), garantindo que o modelo fosse treinado com a maior parte dos dados e testado em um conjunto separado. Logo após, é desenvolvido um modelo baseado na arquitetura de Redes Neurais Recorrentes do tipo LSTM (Long Short-Term Memory), com as seguintes características, escolhidas de maneira arbitrária:

- **Entrada:** Vetor de 4 variáveis normalizadas.
- Camadas LSTM: 3 camadas com 256 núcleos ocultos cada e dropout de 20% para evitar overfitting.
- Camada totalmente conectada (fully connected): Uma camada densa para transformar as saídas da LSTM em um único valor de previsão.

O modelo foi implementado utilizando PyTorch e colocada a opção de treinar em uma GPU, se disponível, o que não foi o caso. Além de ser treinado utilizando a função de perda MSELoss (Erro Quadrático Médio), que mede a diferença entre os valores previstos e reais. O otimizador Adam foi empregado com uma taxa de aprendizado de 0.0001 para um ajuste eficiente dos pesos da rede, realizado ao longo de 40 épocas, utilizando um tamanho de lote (batch size) de 32 amostras. Durante o treinamento. Foram registradas métricas como RMSE (Raiz do Erro Quadrático Médio), MAE (Erro Absoluto Médio) e MAPE (Erro Percentual Absoluto Médio) para monitorar a qualidade das previsões tanto no conjunto de treinamento quanto no conjunto de validação.

Ao final do treinamento, gráficos foram gerados para ilustrar a evolução das métricas ao longo das épocas, permitindo uma análise visual da convergência do modelo. Três gráficos foram plotados:

- Evolução do RMSE (Raiz do Erro Quadrático Médio).
- Evolução do MAE (Erro Absoluto Médio).
- Evolução do MAPE (Erro Percentual Absoluto Médio).

### 4.1.5 Treinamento QLSTM

A construção desse treinamento foi baseada no trabalho já citado anteriormente, adaptando o que era necessário para que pudesse ser executado. A seleção das características foi realizada de forma análoga ao treinamento de LSTM, onde ele se diferencia na construção do modelo que é criado usando a biblioteca da *pennylane*, com uma arquitetura de rotações de Pauli-Y e Pauli-Z. Cada entrada do modelo é codificada em estados quânticos por meio de portas de rotação RY e RZ e a medição do circuito retorna a expectativa do operador Pauli-Z.

A rede neural foi construída com base em 3 camadas:

- Uma camada quântica que processa as entradas e retorna características extraídas.
- Uma camada LSTM clássica para modelar padrões temporais.
- Uma camada linear de saída para gerar a previsão.

E as portas de ativação utilizadas incluem sigmoid para as portas de esquecimento e entrada, e tanh para a célula de estado. Novamente foi usada para a função perda o erro quadrático médio, o otimizador foi o Adam com um *learning rate* de 0,001 ao longo de 10 épocas com batch_size de 32, input_size=4, hidden_size=256, n_qubits=4, n_qlayers=2, e dropout=0,2.

### 4.1.6 Treinamentos utilizando Optuna

Nos primeiros treinamentos, as escolhas dos parâmetros foram realizadas de forma arbitrária, e por conta disso, foi utilizada a biblioteca *optuna*, utilizando a otimização bayesiana, para realizar a escolha de hiperparâmetros melhores. Já para o treinamento do LSTM, foram realizados 5 testes, enquanto que para o QLSTM, foi realizado somente 3 por conta da maior complexidade e demora do algoritmo. A saída do LSTM foi:

| **Parâmetro** | **Valor** |
|---|---|
| hidden_size | 187 |
| num_layers | 2 |
| dropout | 0.26034976401761745 |
| lr | 0.002885890167256685 |

Tabela 1 – Parâmetros do modelo

e do QLSTM

Após as trocas dos parâmetros, foi novamente realizado o treinamento.

### 4.1.7 Treinamento DRL

Por último, foi realizada a construção de um algoritmo de aprendizado por reforço para escolher quais seriam as melhores características para o problema. Para isso, foi criado um

| Parâmetro | Valor |
|---|---|
| hidden_size | 163 |
| n_qubits | 2 |
| n_qlayers | 1 |
| dropout | 0.10545446560473426 |
| lr | 1.946109066698077e-05 |

Tabela 2 – Parâmetros do modelo

ambiente no OpenAI Gym chamado *FeatureSelectionEnv*. Esse ambiente tem como objetivo permitir que um agente de aprendizado por reforço selecione subconjuntos de características que melhor predizem a Temperatura_out_H2_A_[ºC]. Foram definidos no ambiente:

- **Espaço de Ação:** Definido como um espaço binário, onde cada ação corresponde à seleção, ou não seleção, de uma característica.
- **Espaço de Observação:** Representado por um vetor contínuo de tamanho igual ao número de características disponíveis.
- **Função de Recompensa:** Baseada no erro quadrático médio (RMSE) entre os valores reais e previstos pelo modelo LSTM.
- **Critério de Parada:** O episódio termina quando o agente seleciona pelo menos 4 características.

A predição foi utilizando um modelo LSTM com a taxa de aprendizado de 0,1 e o treinamento do agente foi utilizando PPO (*Proximal Policy Optimization*) da biblioteca *stable-baselines3* (RAFFIN et al., 2020).

Após o treinamento, a escolha se deu pela seleção que melhor se adequou, mas que também fosse menos complexa (menos características possíveis); por isso, se deu a escolha pelas variáveis: Temperatura_in_A_[ºC], Temperatura_out_H2_A_[ºC], Temperatura_out_O2_A_[ºC], PowerSet_[%], Pressão_Buffer_Int_[bar].

-

### 4.1.8 Métricas

A comparação entre treinamentos foi realizada utilizando as seguintes métricas: Raiz do Erro Quadrático Médio (do inglês, RMSE), Erro Médio Absoluto (do inglês, MAE) e Erro Percentual Absoluto Médio (do inglês, MAPE). As comparações se dão pelos gráficos de cada métrica em relação às épocas (número de vezes que o algoritmo recebeu os dados). O artigo referencia utiliza somente RMSE, mas para ter uma melhor visualização do comportamento dos dados foram adicionadas as outras duas métricas, sendo o MAE para visualizar dados menores (ele é sensível a erros grandes) e o MAPE para visualizar erros maiores (já que ele possui o problema de não lidar bem com os dados menores)

## 4.2 Resultados e Discussões

### 4.2.1 Análise correlação

Conforme mencionado na metodologia, selecionamos duas abordagens distintas para analisar o comportamento das variáveis e, com isso, geramos dois tipos de gráficos diferentes: um com representações visuais (para comparações diretas) e outro com mapas de calor.

Nos gráficos podemos dividir os resultados em dois grupos diferentes, o primeiro sendo composto pelas duplas de variáveis que foram consideradas para o treinamento de regressão linear, um exemplo está na Fig 13:

1. Temperatura_in_A_[ºC] vs Temperatura_out_H2_A_[ºC] com $R^2$: 0.8867
2. Temperatura_in_A_[ºC] vs Temperatura_out_O2_A_[ºC] com $R^2$: 0.8469
3. Temperatura_out_H2_A_[ºC] vs Temperatura_out_O2_A_[ºC] com $R^2$: 0.9075
4. Corrente_A_[A] vs Tensao_A_[V] com $R^2$: 0.7595
5. Corrente_A_[A] vs Potencia_A_[kW] com $R^2$: 0.9995
6. Corrente_A_[A] vs Pressão_Buffer_Int_[bar] com $R^2$: 0.7583
7. Tensao_A_[V] vs Potencia_A_[kW] com $R^2$: 0.7483

No entanto, embora todos tenham apresentado um resultado satisfatório na métrica, alguns exibiram um comportamento distinto do esperado para uma regressão linear (Fig. 14). Em particular, esses casos apresentaram um padrão discreto, com uma alta concentração de dados em pontos específicos. Já as demais duplas foram agrupadas no segundo conjunto, devido aos baixos valores de $R^2$.

Já para o mapa de calor, obtivemos correlações entre dois grandes grupos: as características que possuem origem elétrica tiveram correlações altas, assim como todas as temperaturas também tiveram boas correlações, enquanto as piores correlações foram entre tensão e todas as temperaturas, como mostrado na Fig. 15.

### 4.2.2 Treinamento de Regressão Linear

O treinamento de Regressão Linear obteve resultados satisfatórios, com a maioria dos valores dos erros médios quadráticos e $R^2$ estando dentro do esperado, com exceção de duas duplas, sendo elas as duplas 3 e 7 da tabela 3.

O resultado da dupla 2 foi satisfatório devido ao fato de a potência ter sido calculada através de uma multiplicação direta entre os valores de tensão e corrente, visto que o equipamento não possuía um sensor específico para medir a primeira variável, enquanto que o resultado da métrica $R^2$ ter sido baixo, em relação ao que se obtive na fase de análise de

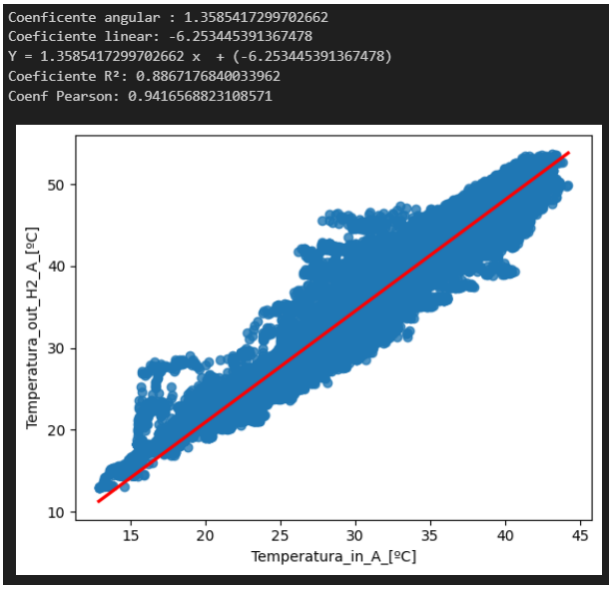



Figura 13 – Gráfico de análise de correlação e $R^2$ da temperatura de entrada em relação a temperatura de saída de Hidrogênio. Fonte: Autor.

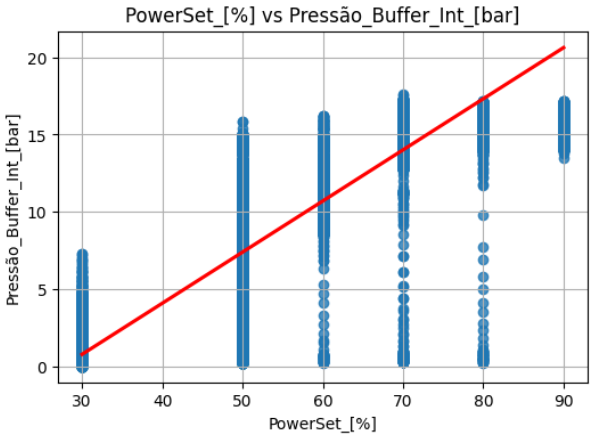



Figura 14 – Gráfico de análise de correlação e $R^2$ do *power set* em relação a pressão do *buffer*. Fonte: Autor.

correlação, pode ter sido causado por conta de terem sido utilizadas bibliotecas diferentes entre elas.

### 4.2.3 Treinamento de Regressão Polinomial

Os resultados das regressões polinomiais foram importantes para entender o comportamento dos dados, pois ficou claro que existem pelo menos dois tipos de relação. A primeira consistia na verificação de platôs nos gráficos, mostrando que existia uma dependência causada por alguma das variáveis que limitava a ação das outras, como visto na Fig. 16. A variável que faz essa limitação é o *Power Set* que faz com que a operação seja mais "forte"ou mais "fraca".

Por sua vez, a segunda ocorre de forma similar à do comportamento de um transistor,

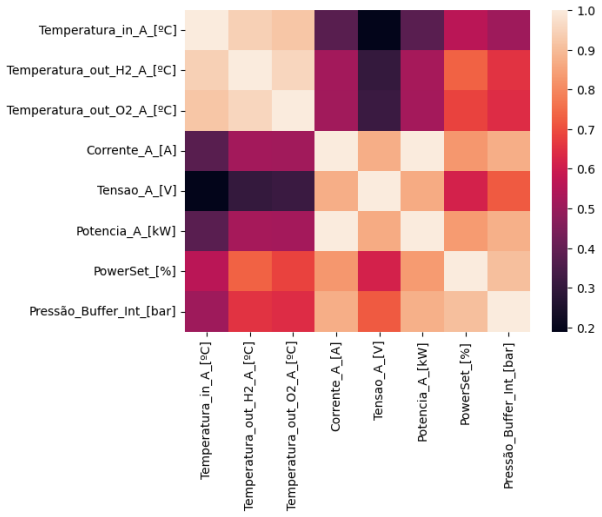



Figura 15 – Mapa de calor das características. Fonte: Autor

| Modelo | Variáveis | MAE | $R^2$ |
|---|---|---|---|
| 1 | Temperatura de saída de O2 vs Temperatura de entrada | 0.05 | 0.85 |
| 2 | Corrente vs Tensão | 0.17 | 0.74 |
| 3 | Corrente vs Potência | 0.00 | 1.00 |
| 4 | Corrente vs PowerSet | 0.14 | 0.67 |
| 5 | Corrente vs Pressão | 0.14 | 0.75 |
| 6 | Temperatura de saída de H2 vs Temperatura de entrada | 0.04 | 0.89 |
| 7 | Temperatura de saída de H2 vs Temperatura de saída O2 | 0.03 | 0.31 |

Tabela 3 – Métricas dos Modelos de Regressão

uma vez que este componente necessita de um valor mínimo de corrente para iniciar sua operação. De fato, note que a magnitude destas variáveis cresce gradativamente a partir de um valor mínimo, conforme o padrão exibido na Fig.17.

### 4.2.4 Normalização dos dados

A normalização ocorreu de forma correta, como podemos ver na Fig. 18.

### 4.2.5 Treinamento LSTM

O resultado do treinamento LSTM pode ser considerado satisfatório já que não houve *overfitting*, pois os erros no treino e na validação são próximos (sendo o principal RMSE). A única situação um pouco mais preocupante seriam os valores do MAPE,que podem ser considerados adequados pelo fato de não ter sido feito ajuste dos hiperparâmetros nesse modelo. Portanto, foi um treinamento satisfatório.

Os valores das métricas podem ser vistos na tabela 4, enquanto que o progresso do treinamento e da validação, para cada métrica, por épocas podem ser vistos nas Figs. 19, 20 e 21.

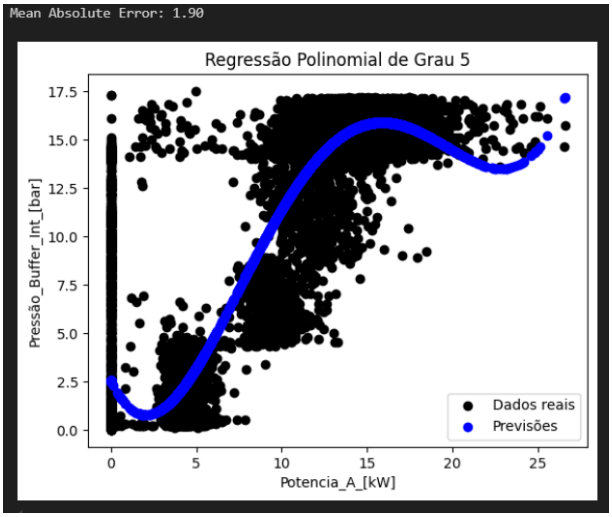



Figura 16 – Treinamento da Regressão Polinomial entre potência e pressão. Fonte: Autor

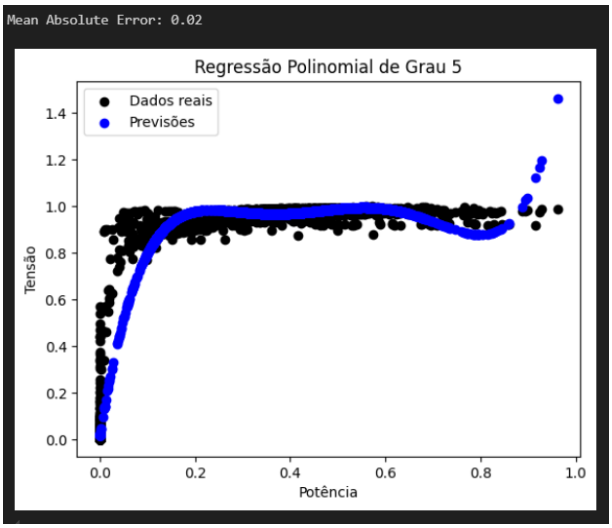



Figura 17 – Treinamento da Regressão Polinomial entre potência e tensão. Fonte: Autor.

Uma das desvantagens do treinamento reside no respectivo tempo de operação (próximo de 11 horas), causado principalmente pela falta de otimização de parâmetros e por ter sido realizada por uma CPU.

Tabela 4 – Métricas do Treinamento LSTM

| Métrica | Treino | Validação |
|---|---|---|
| RMSE | 0.0709 | 0.0732 |
| MAE | 0.0427 | 0.0421 |
| MAPE | 23.25% | 22.27% |

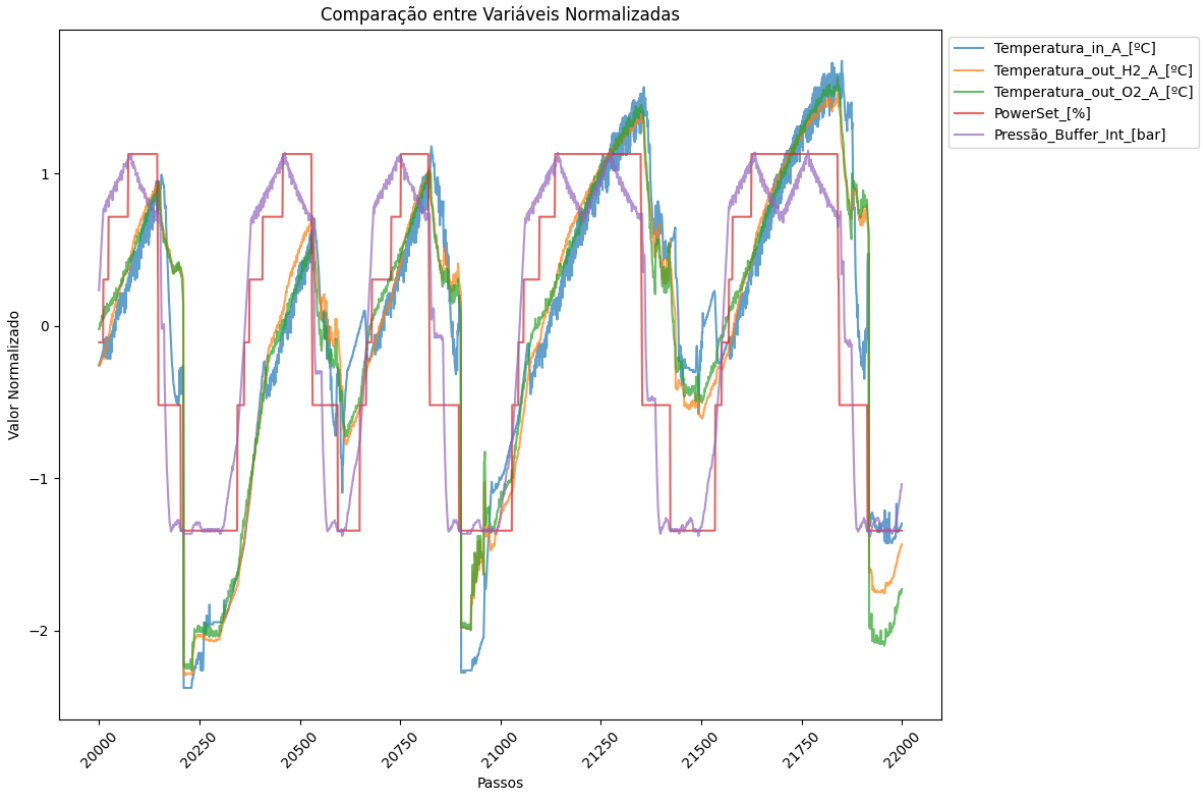



Figura 18 – Normalização de todas as características. Fonte: Autor.

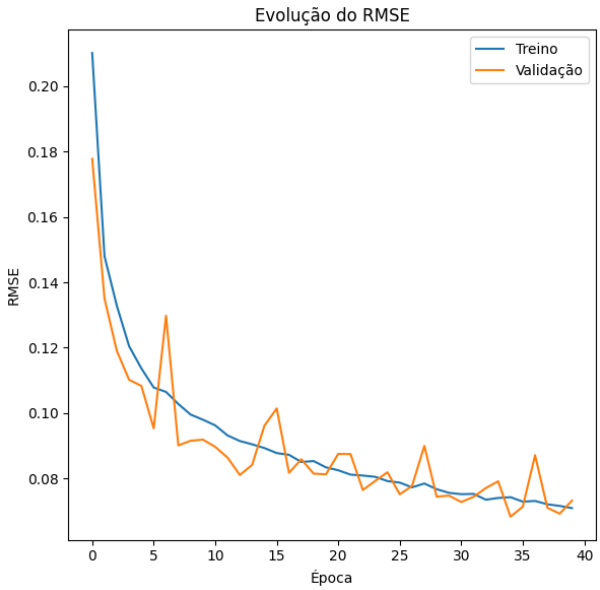



Figura 19 – Evolução da métrica RMSE em relação as épocas no treinamento LSTM. Fonte: Autor.

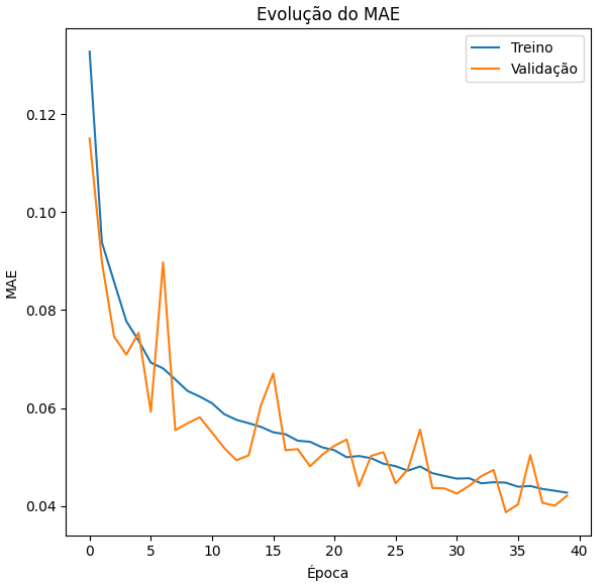



Figura 20 – Evolução da métrica MAE em relação as épocas no treinamento LSTM. Fonte: Autor.

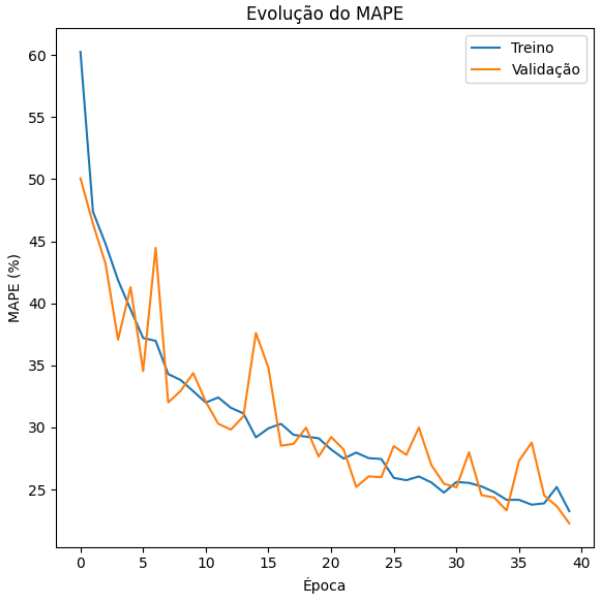



Figura 21 – Evolução da métrica MAPE em relação as épocas no treinamento LSTM. Fonte: Autor.

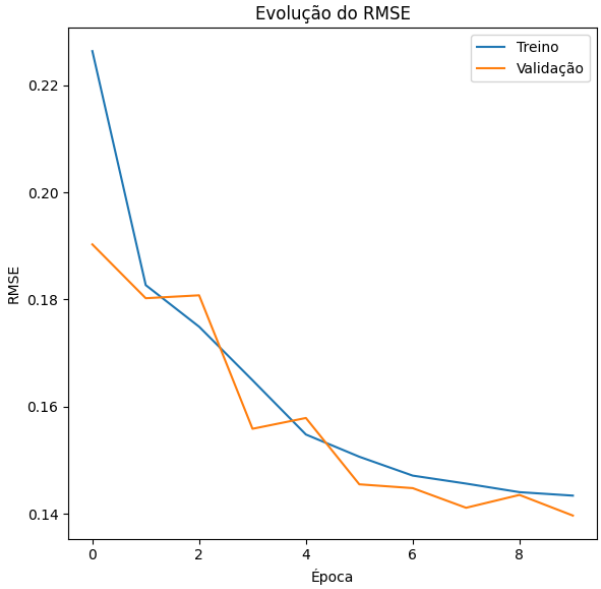



Figura 22 – Evolução da métrica RMSE em relação as épocas no treinamento QLSTM. Fonte: Autor.

### 4.2.6 Treinamento QLSTM

Já o treinamento do QLSTM não foi bem executado, principalmente pelo baixo número de épocas realizadas. Se compararmos este padrão com o modelo anterior, na mesma época exibiram acurácia próximas à metade do valor das vistas na tabela 5. Portanto, uma das possíveis soluções seria realizar o treinamento com mais épocas, que para o caso desse trabalho foi inviável por conta do tempo que demoraria, houve configurações de parâmetros que realizariam este processo em 144 horas. A falta de uma GPU para executar os códigos inviabilizou um treinamento maior, principalmente pelo fato de ter sido realizado um modelo mais simples do que o modelo esquematizado na referência da metodologia. Além disso, o uso de parâmetros que reduzem o tempo de processamento, mas comprometem a qualidade do treinamento, o processo ainda levou cerca de 10 horas.

A evolução das métricas do treinamento em relação às épocas pode ser vista nas Fig. 22, 23 e 24.

Tabela 5 – Métricas do Treinamento QLSTM.

| **Métrica** | **Treino** | **Validação** |
|---|---|---|
| RMSE | 0.1434 | 0.1397 |
| MAE | 0.0943 | 0.0902 |
| MAPE | 52.11% | 49.80% |

### 4.2.7 Treinamentos utilizando Optuna

Os treinamentos utilizando o Optuna para escolher hiperparâmetros mostraram-se muito eficientes, principalmente porque diminuiu o tempo de treinamento do LSTM para 3 horas.

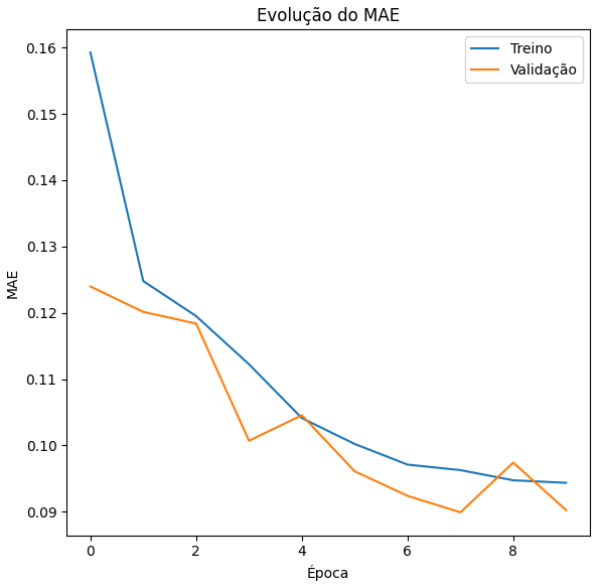



Figura 23 – Evolução da métrica MAE em relação as épocas no treinamento QLSTM. Fonte: Autor.

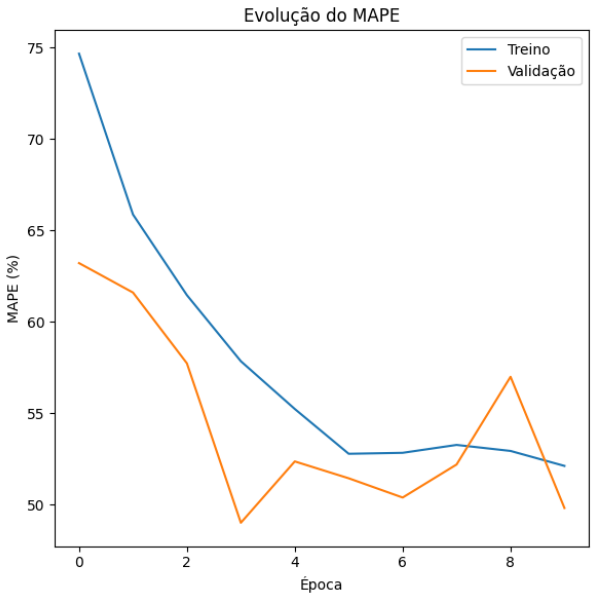



Figura 24 – Evolução da métrica MAPE em relação as épocas no treinamento QLSTM. Fonte: Autor.

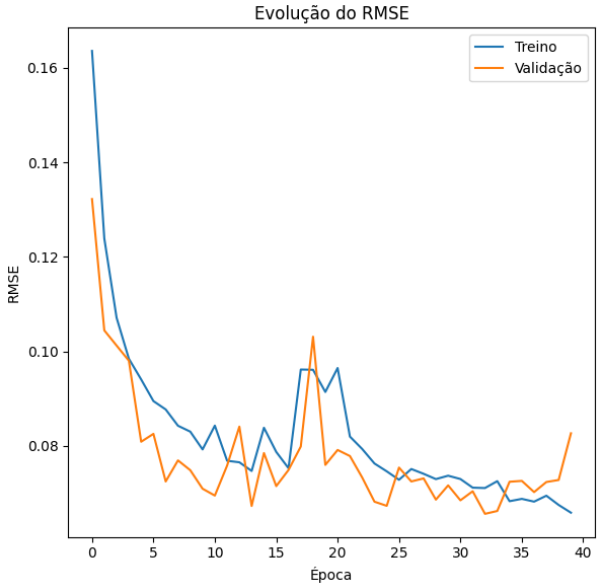



Figura 25 – Evolução da métrica RMSE em relação as épocas no treinamento LSTM com optuna. Fonte: Autor

Além de ter diminuído um pouco os valores das métricas, como pode ser visto na tabela 6, mesmo sendo relativamente pouca a diferença. Entretanto, houve uma diferença grande entre validação e treino, o que demonstra que o modelo generalizou pouco, o que acabou em um provável *overfittng*. As evoluções por épocas são vistas nas Fig. 25, 26 e 27

| **Epoch** | **RMSE** | **MAE** | **MAPE** |
|---|---|---|---|
| Treino | 0.0658 | 0.0398 | 21.60% |
| Val | 0.0826 | 0.0476 | 33.61% |

Tabela 6 – Resultados de Treinamento e Validação usando Optuna.

Houve uma grande vantagem de utilização do Optuna na parte do tempo gasto para poder treinar o modelo QLSTM também, diminuindo para 3 horas; porém, novamente, por conta das poucas épocas, não foi possível ver bons resultados. Ademais, ocorreu um aumento das métricas, como mostrado na tabela 7, provavelmente causado pela diminuição *qubits* e *QLayers*. Lembrando que Optuna não escolhe o melhor modelo para métricas necessariamente, mas sim o melhor custo-benefício, contando principalmente com o tempo para processar todas as camadas.

Entretanto, olhando para os gráficos que mostram os comportamentos das métricas, em todos eles, os pontos finais ainda estavam em declive,como mostrado nas Fig.28, 29 e 30 o que pode indicar que ainda haveria uma queda das métricas com o passar das épocas, mostrando que pode ser uma opção viável se utilizada com as ferramentas corretas.

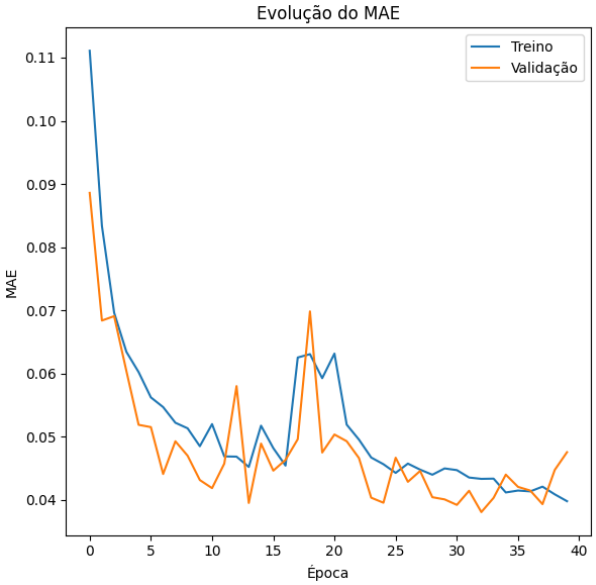



Figura 26 – Evolução da métrica MAE em relação as épocas no treinamento LSTM com optuna. Fonte: Autor

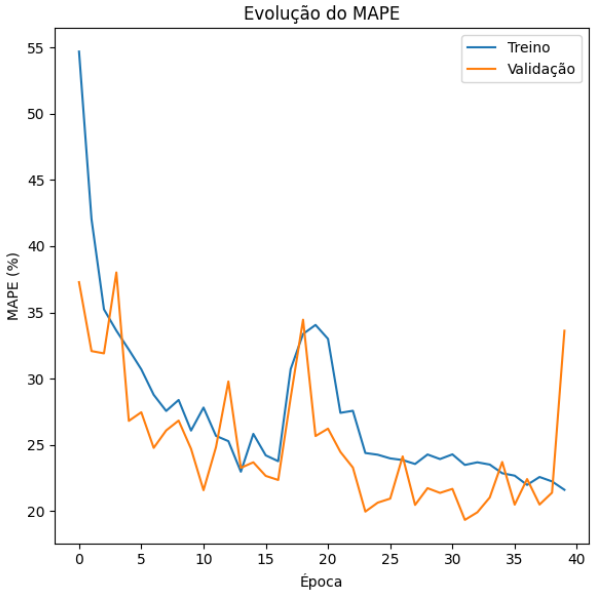



Figura 27 – Evolução da métrica MAPE em relação as épocas no treinamento LSTM com optuna. Fonte: Autor

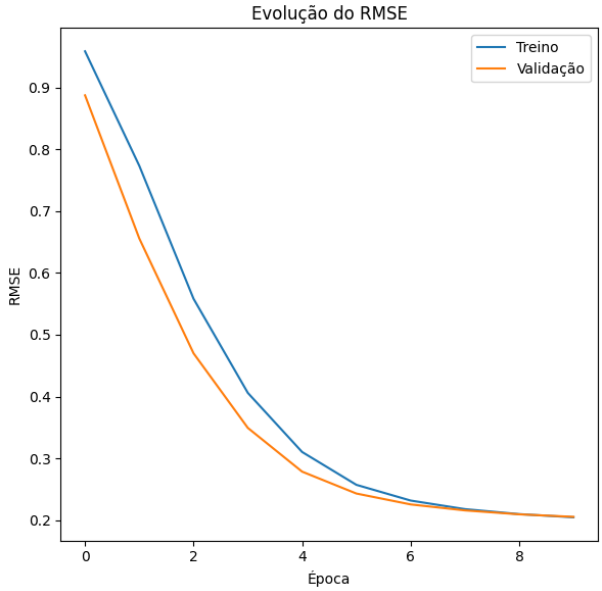



Figura 28 – Evolução da métrica RMSE em relação as épocas no treinamento QLSTM com optuna. Fonte: Autor.

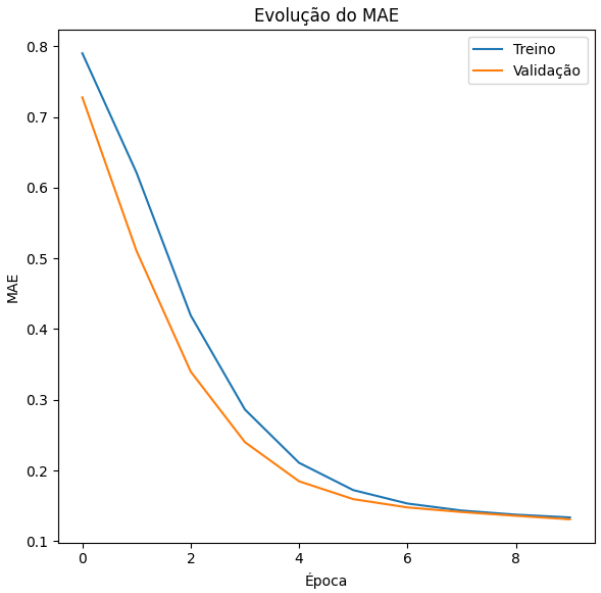



Figura 29 – Evolução da métrica MAE em relação as épocas no treinamento QLSTM com optuna. Fonte: Autor.

| Tipo | RMSE | MAE | MAPE |
|---|---|---|---|
| Treino | 0.2045 | 0.1337 | 66.43% |
| Validação | 0.2052 | 0.1311 | 58.68% |

Tabela 7 – Resultados de Treinamento e Validação do QLSTM usando optuna.

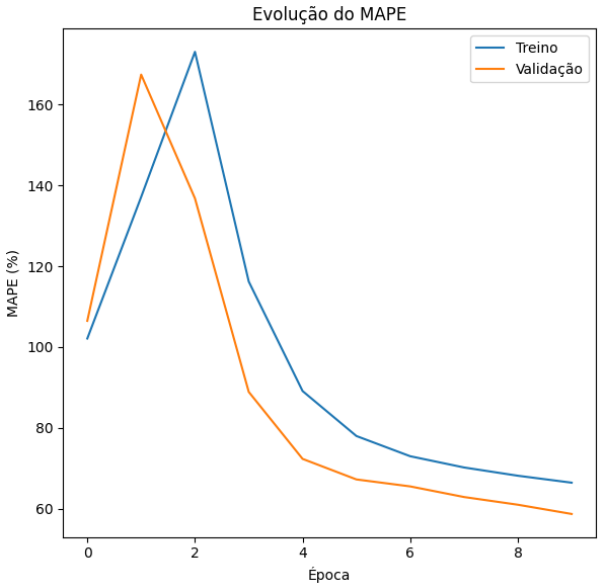



Figura 30 – Evolução da métrica MAPE em relação as épocas no treinamento QLSTM com optuna. Fonte: Autor.

### 4.2.8 Treinamento DRL

Por último, o modelo que utilizou o DRL para fazer as melhores escolhas se mostra com os menores valores de MAPE, porém valores mais altos de RMSE que os outros modelos. Isso pode ter ocorrido porque o modelo cometeu poucos erros, mas esses erros podem ter sido em valores elevados de temperatura. Com isso, os valores absolutos tendem a aumentar, enquanto isso, percentualmente, se eu erro 5 em 500, o valor do erro não é alto.

Em relação ao tempo, seu desempenho está situado entre os dos modelos anteriores, já que o procedimento durou cerca de cinco horas. O maior problema aqui foi não ter realizado uma nova passagem do Optuna para melhorar os hiperparâmetros; com isso, o modelo tende a ser pior do que os outros.

As métricas podem ser vistas na tabela 8 e as progressões das métricas nas Fig.31, 32 e 33.

Tabela 8 – Métricas do Treinamento DRL com LSTM.

| Métrica | Treino | Validação |
|---|---|---|
| RMSE | 0.5520 | 0.4674 |
| MAE | 0.3528 | 0.2775 |
| MAPE | 1.00% | 0.81% |

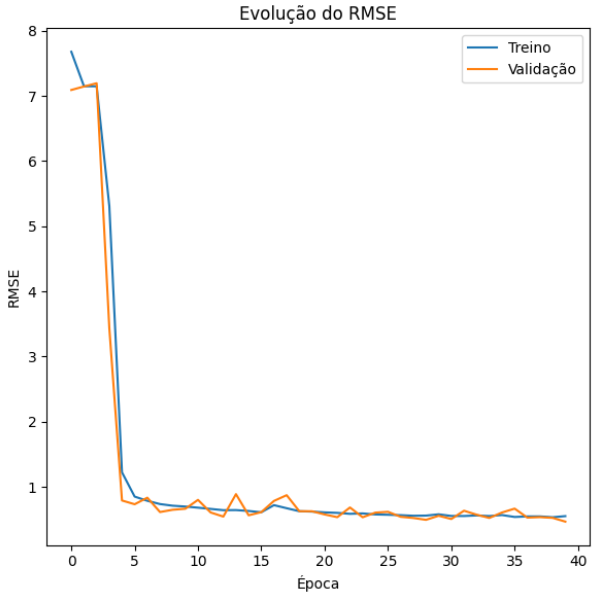



Figura 31 – Evolução da métrica RMSE em relação as épocas no treinamento DRL/LSTM. Fonte: Autor.

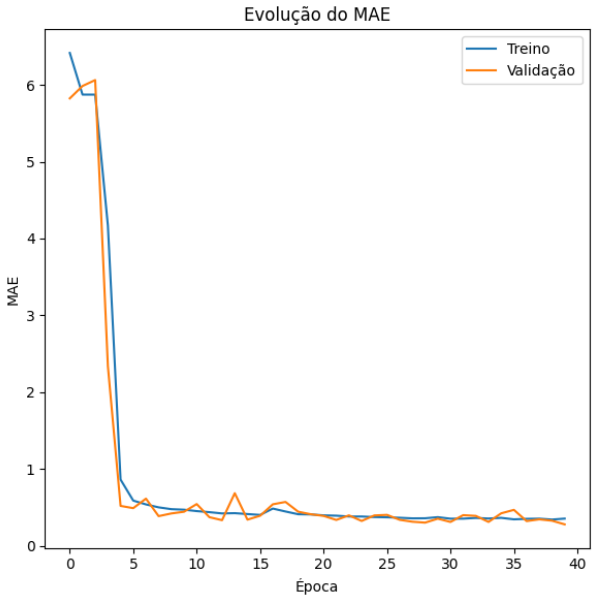



Figura 32 – Evolução da métrica MAE em relação as épocas no treinamento DRL/LSTM. Fonte: Autor.

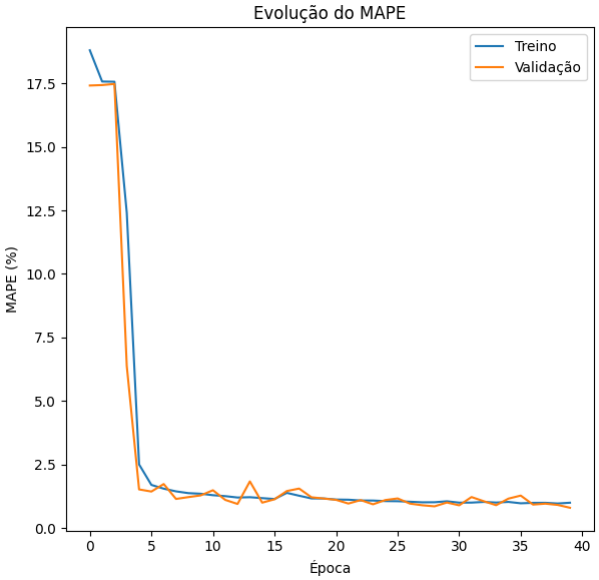



Figura 33 – Evolução da métrica MAPE em relação as épocas no treinamento DRL/LSTM. Fonte: Autor.

Vale apontar que o treinamento de QLSTM com DRL teve valores que mostravam que claramente ele não estava aprendendo; logo, as métricas e previsões ficaram muito ruins.

# 5 CONSIDERAÇÕES FINAIS

## 5.1 Conclusões

O presente trabalho explorou a aplicação de modelos de aprendizado de máquina para previsão da temperatura de saída do hidrogênio em uma planta de produção, utilizando abordagens clássicas e quânticas. Foram desenvolvidos e comparados modelos baseados em LSTM, QLSTM e Deep Reinforcement Learning (DRL), com o objetivo de avaliar sua precisão e eficiência computacional.

Os resultados obtidos demonstram que o modelo LSTM apresentou um desempenho mais robusto e consistente em termos das métricas MAE e RMSE, indicando sua viabilidade para a previsão da temperatura do sistema. O modelo QLSTM, por sua vez, apesar de seu potencial teórico, apresentou limitações devido ao tempo de treinamento prolongado, e com isso, a utilização de poucas épocas, especialmente considerando as limitações da máquina utilizada.

O uso de DRL para seleção de features mostrou-se uma abordagem promissora, reduzindo a complexidade do modelo, diminuindo principalmente o tempo de processamento do treinamento, sem comprometer significativamente a precisão dos resultados. Além de ter melhorado as métricas, principalmente a MAPE. O que indica que o algoritmo conseguiu prever bem os maiores valores.

Dessa forma, pode-se afirmar que a combinação de DRL com LSTM apresenta um potencial promissor para a previsão das temperaturas de saída, sendo possível sua aplicação em outras plantas de hidrogênio, desde que disponham de dados com características semelhantes. Além disso, essa abordagem também pode ser empregada na previsão de outras variáveis, como a resistividade, a qual pode fornecer informações relevantes sobre a degradação do sistema.

Essa capacidade implica que o algoritmo pode ser utilizado em estratégias de manutenção preditiva, permitindo a identificação antecipada de comportamentos anômalos. Para isso, é fundamental a definição de um limiar que estabeleça até que ponto uma variação nos parâmetros pode ser considerada aceitável, evitando diagnósticos falsos de falha.

Um dos desafios identificados foi a necessidade de utilizar GPUs ou clusters para a execução de modelos quânticos de forma eficiente. Além disso, a calibração adequada dos hiperparâmetros mostrou-se essencial para o aprimoramento do desempenho dos modelos, conforme evidenciado pela otimização realizada com a biblioteca Optuna.

## 5.2 Trabalhos futuros

Com base nas análises realizadas, diversas direções futuras podem ser exploradas para aprimorar os resultados obtidos:

- Investir em infraestrutura com GPUs/TPUs para reduzir o tempo de treinamento e explorar circuitos quânticos mais complexos, bem como verificar a disponibilidade de utilizar o Google Colab ou afins, ou alguma outra máquina virtual;
- Coletar mais dados operacionais da planta (incluindo variáveis como umidade e pressão externa) para melhorar a robustez dos modelos;
- Implementar os modelos em sistemas embarcados para monitoramento contínuo e integração com sistemas de controle da planta.

# Referências


ABIOLA, A.; MANZANO, F. S.; ANDúJAR, J. M. A novel deep reinforcement learning (drl) algorithm to apply artificial intelligence-based maintenance in electrolysers. *Algorithms*, v. 16, n. 12, 2023. ISSN 1999-4893. Disponível em: <https://www.mdpi.com/1999-4893/16/12/541>.

AGGARWAL, C. C. *Neural Networks and Deep Learning*: A textbook. Cham: Springer, 2018. 497 p. ISBN 978-3-319-94462-3.

ALVES, V. H. M.; GOMES, R. F. I.; CURY, A. New perspectives on structural health monitoring using unsupervised quantum machine learning. *Mechanical Systems and Signal Processing*, v. 229, p. 112489, 2025. ISSN 0888-3270. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0888327025001906>.

BENENTI, G.; CASATI, G.; STRINI, G. *Principles of quantum computation and information: Basic tools and special topics*. [S.l.]: World Scientific, 2004. v. 2.

BERGMANN, D. *O que é aprendizado semissupervisionado?* 2023. Acessado em 18 de março de 2025. Disponível em: <https://www.ibm.com/br-pt/think/topics/semi-supervised-learning>.

BURKOV, A. *Machine Learning Engineering*. True Positive Incorporated, 2020. ISBN 9781999579579. Disponível em: <https://books.google.com.br/books?id=HeXizQEACAAJ>.

CHEN, S. Y.-C.; YOO, S.; FANG, Y.-L. L. *Quantum Long Short-Term Memory*. 2020. Disponível em: <https://arxiv.org/abs/2009.01783>.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. MIT Press, 2016. Book in preparation for MIT Press. Disponível em: <http://www.deeplearningbook.org>.

GRIOL-BARRES, I. et al. Variational quantum circuits for machine learning. an application for the detection of weak signals. *Applied Sciences*, v. 11, n. 14, p. 6427, 2021. Disponível em: <https://www.mdpi.com/2076-3417/11/14/6427>.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. *Neural computation*, MIT Press, v. 9, n. 8, p. 1735–1780, 1997.

IBM. *Recurrent Neural Networks*. 2024. <https://www.ibm.com/br-pt/think/topics/recurrent-neural-networks>. Acessado em: 22 Fev. 2024.

IBM. *Aprendizado Não Supervisionado*. 2025. Acesso em: 21 fev. 2025. Disponível em: <https://www.ibm.com/br-pt/topics/unsupervised-learning>.

IBM. *Redes Neurais*. 2025. Acessado em: 22 fev. 2025. Disponível em: <https://www.ibm.com/br-pt/topics/neural-networks>.

IBM. *Supervised Learning*. 2025. Acesso em: 21 fev. 2025. Disponível em: <https://www.ibm.com/think/topics/supervised-learning>.

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. Disponível em: <https://arxiv.org/abs/1412.6980>.

KUMAR, S. S.; LIM, H. An overview of water electrolysis technologies for green hydrogen production. *Energy Reports*, 2022. Disponível em: <https://api.semanticscholar.org/CorpusID:253141292>.

LEARNING, D. into D. *Long Short-Term Memory (LSTM)*. 2025. Acessado em 18 de março de 2025. Disponível em: <https://pt.d2l.ai/chapter_recurrent-modern/lstm.html>.

MCMAHON, D. *Quantum Computing Explained*. [S.l.]: Wiley, 2007. ISBN 9780470096994.

MERMIN, N. D. *Quantum Computer Science: An Introduction*. [S.l.]: Cambridge University Press, 2007. ISBN 9780521876582.

MOHRI, M.; ROSTAMIZADEH, A.; TALWALKAR, A. *Foundations of Machine Learning*. MIT Press, 2012. (Adaptive Computation and Machine Learning series). ISBN 9780262018258. Disponível em: <https://books.google.com.br/books?id=maz6AQAAQBAJ>.

NACCHIA, M. et al. A systematic mapping of the advancing use of machine learning techniques for predictive maintenance in the manufacturing sector. *Applied Sciences*, v. 11, n. 6, 2021. ISSN 2076-3417. Disponível em: <https://www.mdpi.com/2076-3417/11/6/2546>.

NAYERI, Z. M.; GHAFARIAN, T.; JAVADI, B. Application placement in fog computing with ai approach: Taxonomy and a state of the art survey. *Journal of Network and Computer Applications*, v. 185, p. 103078, 2021. ISSN 1084-8045. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1084804521000989>.

NIELSEN, M. A.; CHUANG, I. L. *Quantum Computation and Quantum Information*. 10th anniversary edition. ed. [S.l.]: Cambridge University Press, 2010. ISBN 9781107002173.

PEDREGOSA, F. et al. *Scikit-learn: Machine Learning in Python*. 2011. 2825–2830 p.

PENNYLANE. *Pennylane Documentation: Quantum Machine Learning*. 2023. Disponível em: <https://pennylane.ai/qml/whatisqml>.

RAFFIN, A. et al. *Stable-Baselines3: Reliable Reinforcement Learning Implementations*. 2020. Available at `https://github.com/DLR-RM/stable-baselines3`. Disponível em: <https://github.com/DLR-RM/stable-baselines3>.

RODRIGUES, A. L. V. et al. Estudo de viabilidade para implementação de manutenção preditiva no contexto da indústria 4.0. *Brazilian Journal of Development*, v. 9, n. 8, p. 23133–23154, Aug. 2023. Disponível em: <https://ojs.brazilianjournals.com.br/ojs/index.php/BRJD/article/view/61839>.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. *nature*, Nature Publishing Group UK London, v. 323, n. 6088, p. 533–536, 1986.

SELCUK, S. Predictive maintenance, its implementation and latest trends. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, v. 231, n. 9, p. 1670–1679, 2017. Disponível em: <https://doi.org/10.1177/0954405415601640>.

SILVA, I. N. d.; SPATTI, D. H.; FLAUZINO, R. A. *Redes neurais artificiais para engenharia e ciências aplicadas*. [S.l.]: Artliber Editora, 2010.

Stargate Hydrogen. *The Basics of Hydrogen Electrolysis*. <https://stargatehydrogen.com/blog/basics-of-hydrogen-electrolysis/>. Disponível em: <https://stargatehydrogen.com/blog/basics-of-hydrogen-electrolysis/>. Acesso em: 01 out. 2023.

SUTTON, R. S.; BARTO, A. G. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018. Disponível em: <http://incompleteideas.net/book/the-book-2nd.html>.

TEAM, T. pandas development. *pandas-dev/pandas: Pandas*. Zenodo, 2024. Disponível em: <https://doi.org/10.5281/zenodo.10957263>.

# APÊNDICE A – Algoritmos

## A.1 Correlação

```

#bibliotecas
import pandas as pd
import numpy as np
import seaborn as sns
from scipy import stats
import matplotlib as plt

df = pd.read_csv('dados_marcelo_limpos.csv') #importa os dados do excel/csv
print (df)
slope, intercept, r_value, p_value, std_err=stats.linregress(df['Temperatura_in_A_[ C ]'], df['Temperatura_out_O2_A_[ C ]']) #calcula todas as m tricas citadas usando o stats do scipy
sns.regplot(y="Temperatura_in_A_[ C ]", x = "Temperatura_out_O2_A_[ C ]", data=df, line_kws={"color":"red"}) #cria o gr fico com regress o simples
pearson_coef,p_value = stats.pearsonr (df['Temperatura_in_A_[ C ]'], df['Temperatura_out_O2_A_[ C ]']) # calcula o valor do Coeficiente Pearson e do valor P para saber a correla  o das variaveis
# print dos coeficientes e da equa  o da reta junto ao coeficiente R
print(f"Coenficente angular : {slope}")
print(f"Coeficiente linear: {intercept}")
print(f"Y = {slope} x " f" + ({intercept})")
print(f"Coeficiente R  : {r_value**2}")
print (f"Coenf Pearson: {pearson_coef}")
```

## A.2 Treinamento Linear

```

# Importa  o de bibliotecas
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score

# Carregamento dos dados limpos, removendo os dias de teste
```

```
df = pd.read_csv('dados_marcelo_normalizados.csv')

# Defini  o das vari veis independentes (X) e dependentes (Y)
x1 = df["Temperatura_out_O2_A_[ C ]"].values.reshape(-1, 1)
y1 = df["Temperatura_in_A_[ C ]"].values.reshape(-1, 1)

# Divis o dos dados em conjunto de treinamento (80%) e teste (20%)
x1_train, x1_test, y1_train, y1_test = train_test_split(x1, y1,
    test_size=0.2, random_state=42)

# Cria  o e treinamento do modelo de regress o linear
modelo_temp = linear_model.LinearRegression()
modelo_temp.fit(x1_train, y1_train)

# Previs es do modelo
predicoes_temp = modelo_temp.predict(x1_test)

# Gr fico de dispers o: valores reais vs previs es
plt.scatter(x1_test, y1_test, color='black', label='Dados reais')
plt.scatter(x1_test, predicoes_temp, color='blue', label='Previs es')
plt.xlabel('Temperatura de Sa da O2 ( C )')
plt.ylabel('Temperatura de Entrada ( C )')
plt.title('Regress o Linear: Temperatura de Entrada vs. Sa da')
plt.legend()
plt.show()

# Defini  o de outras vari veis para regress o
x2 = df["Corrente_A_[A]"].values.reshape(-1, 1)
y2 = df["Tensao_A_[V]"].values.reshape(-1, 1)

x3 = df["Corrente_A_[A]"].values.reshape(-1, 1)
y3 = df["Potencia_A_[kW]"].values.reshape(-1, 1)

x4 = df["Corrente_A_[A]"].values.reshape(-1, 1)
y4 = df["PowerSet_[%]"].values.reshape(-1, 1)

x5 = df["Corrente_A_[A]"].values.reshape(-1, 1)
y5 = df["Press o_Buffer_Int_[bar]"].values.reshape(-1, 1)

# Modelo 1: Regress o Linear para Temperatura de Entrada vs. Sa da
x1_train, x1_test, y1_train, y1_test = train_test_split(x1, y1,
    test_size=0.2, random_state=42)

modelo_temp_ajustado = linear_model.LinearRegression()
modelo_temp_ajustado.fit(x1_train, y1_train)
predicoes_temp_ajustado = modelo_temp_ajustado.predict(x1_test)

```

```
# C lculo do erro m dio absoluto (MAE) e coeficiente de determina  o (R )
mae_temp = np.mean(np.absolute(predicoes_temp_ajustado - y1_test))
r2_temp = r2_score(y1_test, predicoes_temp_ajustado)

# Exibi  o dos resultados
print("Modelo 1: Temperatura de Sa da O2 vs Temperatura de Entrada")
print(f"Mean Absolute Error (MAE): {mae_temp:.2f}")
print(f"R  : {r2_temp:.2f}\n")

# Exibi  o do gr fico do modelo ajustado
plt.scatter(x1_test, y1_test, color='black', label='Dados reais')
plt.scatter(x1_test, predicoes_temp_ajustado, color='blue', label='Previs es')
plt.xlabel('Temperatura de Sa da O2 ( C )')
plt.ylabel('Temperatura de Entrada ( C )')
plt.title('Regress o Linear: Temperatura de Entrada vs. Sa da')
plt.legend()
plt.show()
```

## A.3 Treinamento Polinomial

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Carregar os dados
df = pd.read_csv('dados_marcelo_limpos.csv')

x1 = df["Potencia_A_[kW]"].values.reshape(-1, 1)
y1 = df["Press o_Buffer_Int_[bar]"].values.reshape(-1, 1)

# Separando os dados em conjunto de treinamento e teste
x1_train, x1_test, y1_train, y1_test = train_test_split(x1, y1, test_size=0.2, random_state=42)

# Escolher o grau do polinomio e cria a Feature dele
grau = 5  # Escolha o grau desejado aqui
poly = PolynomialFeatures(degree=grau)

# Transformar os dados de entrada no novo grau
```

```
x1_train_poly = poly.fit_transform(x1_train)
x1_test_poly = poly.transform(x1_test)

# Instanciar e ajustar o modelo de regress o linear aos dados
    transformados
regr = LinearRegression()
regr.fit(x1_train_poly, y1_train)

# Previsoes
predictions = regr.predict(x1_test_poly)

# Calcular o erro absoluto m dio
print("Mean Absolute Error: %.2f" % np.mean(np.absolute(predictions -
    y1_test)))

# Plotar resultados
plt.scatter(x1_test, y1_test, color='black', label='Dados reais')
plt.scatter(x1_test, predictions, color='blue', label='Previs es')
plt.xlabel('Potencia_A_[kW]')
plt.ylabel('Press o_Buffer_Int_[bar]')
plt.title('Regress o Polinomial de Grau %d' % grau)
plt.legend()
plt.show()
```

## A.4 Optuna

Os algoritmos Optuna estão neste modelos a seguir:

```

import numpy as np
import pandas as pd
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Dataset
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
import optuna
import time

#        1. Carregar e Normalizar os Dados
df = pd.read_csv("dados_marcelo_limpos.csv")
df["Data / Hora"] = pd.to_datetime(df["Data / Hora"], format="%d/%m/%y %
    H:%M:%S")

variaveis_para_normalizar = [
```

```
    "Temperatura_in_A_[ C ]", "Temperatura_out_H2_A_[ C ]", "
    Temperatura_out_O2_A_[ C ]", "PowerSet_[%]", "Press o_Buffer_Int_[
    bar]"
]
scaler = StandardScaler()
df[variaveis_para_normalizar] = scaler.fit_transform(df[
    variaveis_para_normalizar])

target_col = "Temperatura_out_H2_A_[ C ]"
inputs = df[["Temperatura_in_A_[ C ]", "Temperatura_out_O2_A_[ C ]", "
    PowerSet_[%]", "Press o_Buffer_Int_[bar]"]].values
target = df[target_col].values
data = np.hstack([inputs, target.reshape(-1, 1)])

#        2. Criar o Dataset de s ries temporais
class TimeSeriesDataset(Dataset):
    def __init__(self, data, seq_length, target_idx):
        self.data = data
        self.seq_length = seq_length
        self.target_idx = target_idx

    def __len__(self):
        return len(self.data) - self.seq_length

    def __getitem__(self, idx):
        x = self.data[idx:idx + self.seq_length, :-1]
        y = self.data[idx + self.seq_length, self.target_idx]
        return torch.tensor(x, dtype=torch.float32), torch.tensor(y,
    dtype=torch.float32)

#        3. Dividir os Dados em Treino, Valida  o e Teste
seq_length = 50  #        Reduzido para economizar mem ria
target_idx = inputs.shape[1]
dataset = TimeSeriesDataset(data, seq_length, target_idx)

train_size = int(len(dataset) * 0.8)
val_size = int(len(dataset) * 0.1)
test_size = len(dataset) - train_size - val_size

train_dataset, val_dataset, test_dataset = torch.utils.data.random_split
    (dataset, [train_size, val_size, test_size])

batch_size = 16  #        Reduzido para evitar estouro de mem ria
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=
    True)
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=
    False)
```

```
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

#        4. Criar o Modelo LSTM com Hiperpar metros Otimiz veis
class LSTMModel(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers, dropout):
        super(LSTMModel, self).__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True, dropout=dropout)
        self.fc = nn.Linear(hidden_size, 1)

    def forward(self, x):
        out, _ = self.lstm(x)
        out = self.fc(out[:, -1, :])
        return out.squeeze()

# Fun   o para calcular m tricas
def calculate_metrics(y_pred, y_true):
    y_pred = y_pred.to(y_true.device)
    mse = nn.MSELoss()(y_pred, y_true)
    rmse = torch.sqrt(mse)
    mae = nn.L1Loss()(y_pred, y_true)
    epsilon = 1e-8
    mape = torch.mean(torch.abs((y_true - y_pred) / (torch.abs(y_true) + epsilon))) * 100
    return rmse.item(), mae.item(), mape.item()

#        5. Otimiza   o Bayesiana com Optuna
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

def objective(trial):
    hidden_size = trial.suggest_int("hidden_size", 64, 256)  # Reduzido para economizar mem ria
    num_layers = trial.suggest_int("num_layers", 1, 3)  # Reduzido para evitar estouro
    dropout = trial.suggest_float("dropout", 0.1, 0.4)
    lr = trial.suggest_float("lr", 1e-5, 1e-2, log=True)  # Corrigido para ‘suggest_float(..., log=True)‘

    model = LSTMModel(input_size=inputs.shape[1], hidden_size=hidden_size, num_layers=num_layers, dropout=dropout).to(device)
    optimizer = optim.Adam(model.parameters(), lr=lr)
    criterion = nn.MSELoss()

    model.train()
    num_epochs = 5  #        Reduzido para acelerar a busca
    for epoch in range(num_epochs):
```

```
97          for batch_inputs, batch_targets in train_loader:
98              batch_inputs, batch_targets = batch_inputs.to(device),
    batch_targets.to(device)
99              optimizer.zero_grad()
100             outputs = model(batch_inputs)
101             loss = criterion(outputs, batch_targets)
102             loss.backward()
103             optimizer.step()
104
105     model.eval()
106     val_rmse = 0
107     with torch.no_grad():
108         for batch_inputs, batch_targets in val_loader:
109             batch_inputs, batch_targets = batch_inputs.to(device),
    batch_targets.to(device)
110             outputs = model(batch_inputs)
111             rmse, _, _ = calculate_metrics(outputs, batch_targets)
112             val_rmse += rmse
113
114     return val_rmse / len(val_loader)
115
116 #         6. Executar a Otimiza  o
117 study = optuna.create_study(direction="minimize")
118 study.optimize(objective, n_trials=5)
119
120 # Melhor configura  o encontrada
121 best_params = study.best_params
122 print("\nMelhores Hiperpar metros:", best_params)
123
124 #         7. Treinar o Modelo Final
125 model = LSTModel(input_size=inputs.shape[1], **best_params).to(device)
126 optimizer = optim.Adam(model.parameters(), lr=best_params["lr"])
127 criterion = nn.MSELoss()
128
129 num_epochs = 20
130 start_time = time.time()
131
132 for epoch in range(num_epochs):
133     model.train()
134     for batch_inputs, batch_targets in train_loader:
135         batch_inputs, batch_targets = batch_inputs.to(device),
    batch_targets.to(device)
136         optimizer.zero_grad()
137         outputs = model(batch_inputs)
138         loss = criterion(outputs, batch_targets)
139         loss.backward()
140         optimizer.step()
```

```
141
142 print("\nTreinamento finalizado!")
143
144 #         8. Avalia  o no Conjunto de Teste
145 model.eval()
146 test_rmse, test_mae, test_mape = 0, 0, 0
147
148 with torch.no_grad():
149     for batch_inputs, batch_targets in test_loader:
150         batch_inputs, batch_targets = batch_inputs.to(device),
    batch_targets.to(device)
151         outputs = model(batch_inputs)
152         rmse, mae, mape = calculate_metrics(outputs, batch_targets)
153         test_rmse += rmse
154         test_mae += mae
155         test_mape += mape
156
157 print(f"\nPerformance no Teste -> RMSE: {test_rmse / len(test_loader):.4
    f}, MAE: {test_mae / len(test_loader):.4f}, MAPE: {test_mape / len(
    test_loader):.2f}%")
```