FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Técnicas de previsão de vida útil restante utilizando algoritmos de inteligência artificial

André Escobar Moutinho

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: João Pedro Carvalho Leal Mendes Moreira

27 de outubro de 2023

Resumo

Esta dissertação de mestrado procura oferecer uma resposta ao problema de predição de tempo útil restante de funcionamento (RUL - do inglês, Remaining Usefull Life Time) usando técnicas de aprendizagem profunda. Aliado a um ecossistema de sensores e micro controladores que permitem a monitorização do estado de degradação e a previsão de falha em tempo real de componentes de difícil acesso. Os métodos utilizados foram redes neuronais recursivas (RNN), em específico a rede de memória de curto prazo é uma rede neural recorrente (LSTM), que implementa mecanismos de memória seletivos. Estes métodos foram testados em dois algoritmos de otimização ADAM e SGD. A experiência foi conduzida usando um conjunto de dados gerado por uma ferramenta de simulação. As metodologias usadas para a classificação do RUL consiste numa função de classificação que favorece previsões precoces em ralação a previsões tardias.

Entende-se por *Prognosis and health management (PHM)* a tarefa de monitorizar em tempo real os agentes e a evolução da degradação de sistemas no seu período de funcionamento. Avaliando o estado de funcionamento, *state of healf (SoH)* permite determinar o *remaining useful life time (RUL)*, que é o período entre o presente e o final de vida útil de um dado sistema.

Prever a degradação e a falha de componentes é essencial, tanto para assegurar o correto funcionamento de um sistema, como a segurança daqueles que o operam. Este requisito torna-se ainda mais importante no que diz respeito a sistemas que operam em condições adversas (ex: temperatura elevada).

Portanto, as técnicas de inteligência artificial são uma resposta inquestionável para este problema, pois utilizam dados históricos de degradação para replicar múltiplos estados de saúde (SoHs) sob várias condições de funcionamento

A capacidade de auto extração de características e adaptabilidade de algoritmos de aprendizagem profunda e a capacidade das RNN extraírem a informação intrínseca nas séries temporais de dados de degradação, tornam este tipo de modelos uma boa solução de algoritmo de previsão para um sistema de manutenção preventiva (PHM) para rolamentos. Assim, neste trabalho de dissertação proponho uma solução de manutenção preventiva baseada em dados, usando modelo de redes neuronais profundas recorrentes para previsão do tempo de vida restante (RUL) de rolamentos.

Diferentes conjuntos de dados foram usados e analisados. Todos eles são constituídos por séries temporais de dados de sensores que refletem as condições de funcionamento de um dado sistema.

Os resultados obtidos mostram que a eficiência e credibilidade do modelo preditivo depende fortemente dos hiperparâmetro escolhidos, nomeadamente o algoritmo de otimização. ADAM sendo um algoritmo que usa uma média dos N últimos gradientes de um batch é um algoritmo muito eficiente no treino de redes RNN que lidam com series temporais, não sendo assim facilmente afetado por valores atípicos, garantindo uma convergência estável do modelo na fase de

treino.

Keywords: Machine Learning, Deep Learning, Data Sience, Remaining Usefull Life Time, ADAM

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

CM	Condition Monitoring
CB	Condition Based
PM	Predictive Maintenence
PHM	Predictive Health Maintenence
AE	Auto Encoder
MCU	Microcontroller
RPM	Rotações Por Minuto
RUL	Remaining Usefull Life Time
AP	Aprendizagem Computacional
ML	Machine Learning
ANN	Artificial Neural Network
DNN	Deep Neural Networks
RNN	Recurrent Neural Network
FNN	Feedforward Neural Network
LSTM	Long Short Term Memory
ADAM	Adaptive Moment Estimation
SGD	Stochastic Gradient Descent

Capítulo 1

Introdução

O presente documento é o resultado do trabalho realizado no contexto do projeto final de mestrado. Neste capitulo introdutório, podemos encontrar uma contextualização, os principais objetivos, assim como as questões a que se pretende responder e metodologia tomada. Por fim, é apresentada uma descrição da estrutura global do documento.

1.1 Contexto

Os rolamentos são componentes críticos em máquinas rotativas, e a sua falha pode conduzir a custosas paragens e riscos de segurança. Como tal, ser capaz de prever com precisão a vida útil remanescente (RUL) dos rolamentos é essencial para a manutenção otimizada e segurança operacional de sistemas industriais. No entanto, prever o RUL é um desafio devido à natureza complexa e incremental dos processos de degradação dos rolamentos.

Este trabalho pretende dar resposta ao problema de monitorização e determinação do momento de falha de rolamentos de motores de arranque no setor da aviação. Um estudo à cerca de sistemas de PHM - manutenção preventiva, dando ênfase aos algoritmos de inteligência artificial para previsão de falhas, foi realizado e exposto neste documento.

Este trabalho foi inspirado no projeto iBearing - Instrumented bearing for oil cooled starter / generator - desenvolvido pela Active Space. O projeto iBearing consiste na produção, teste e validação de sensores inteligentes para monitorizar a degradação estrutural de rolamentos no decorrer da sua vida útil. Envolve o desenvolvimento de eletrónica analógica, digital e software. O projeto procura desenvolver metodologias que prevejam a falha de rolamentos com, pelo menos, 100 horas de antecedência, com um nível de significância de 0.997 (3 sigma).

Os principais objetivos são:

- Monitorização *in-situ* do tempo real dos rolamentos;
- Utilização de metodologias e técnicas *State of the Art* e validadas para monitorização de rolamentos num ambiente hostil (altas temperaturas e vibrações).

- Estudo de possíveis novas soluções que permitam uma previsão do tempo de fim de vida do componente.

Uma contextualização detalhada do projeto iBearings é feita neste capitulo. Componentes digital e de hardware do projeto são explicados e algumas soluções alternativas são sugeridas. Este trabalho foi desenvolvido para dar resposta a estas questões.

Nesta tese, são propostos métodos baseados em dados usando aprendizagem profunda para avaliar a saúde de rolamentos e estimar o RUL diretamente a partir de dados brutos de sensores. Em comparação com abordagens tradicionais baseadas em modelos, as técnicas baseadas em dados não requerem extenso conhecimento de domínio para derivar modelos analíticos dos mecanismos de falha. Com dados de treino suficientes, os modelos de aprendizagem automática podem aprender automaticamente padrões complexos de saúde e trajetórias de degradação.

Especificamente, esta tese explora o uso de redes neuronais recorrentes (RNN) incluindo modelos de memória de curto e longo prazo (LSTM) para previsão de RUL. As arquiteturas RNN são adequadas para dados sequenciais como sinais de sensores em séries temporais. A memória de curto prazo das RNN é aumentada com unidades LSTM para capturar dependências temporais de longo prazo nas tendências de degradação. Uma combinação de extração de características e modelação de sequência permite que os modelos RNN estimem de forma eficaz o RUL a partir de medições brutas de vibração e temperatura.

Os métodos propostos são avaliados conjunto de dados de rolamentos run-to-failure sob várias condições operacionais. As técnicas desenvolvidas poderiam ajudar a possibilitar monitorização baseada na condição e manutenção preditiva de componentes rotativos críticos de segurança.

Esta tese foca-se na componente de algoritmos de Aprendizagem Computacional, tendo em mente o tipo de dados a serem utilizados e a sua finalidade. Ela vai dar uma resposta adaptada a séries temporais usando um tipo de redes neuronais recursivas otimizadas com um algoritmo que acompanha a tendência de uma certa janela em vez de se basear em casos isolados. Assim as principais contribuições foram:

- Uma proposta de uma arquitetura de modelo baseada em LSTM que combina aprendizagem automática de características a partir de dados brutos do sensor de vibração com modelagem de sequência de tendências de degradação.

- Foi demonstrado que o modelo superou abordagens de aprendizado de máquina superficial (shallow ML) utilizadas como referência.

- Um estudo e discussão sobre metodologias AI foram discutidas e articuladas. Estratégias de implementação foram discutidas e apresentadas para dois tipos de conjuntos de dados diferentes, um primeiro simulado e completo, outro real e acelerado.

- Uma fórmula de avaliação de performance dos modelos que favorece previsões precoces em relação a previsões tardias.

Para resolver este problema de manutenção preventiva, será estudada a literatura e uma solução abstrata será proposta e fundamentada por pequenos ensaios, com recursos encontrados online ou feitos por mim. Foi usado um conjunto de dados públicos que servem de Benchmark com características específicas que requerem certos algoritmos e técnicas de PHM e pré-processamento. Um estudo destas nuances é feito ao longo da dissertação. Nenhum modelo de simulação ou pressupostos físicos dos rolamentos serão tidos em conta no modelo de previsão. Assim, uma abordagem indutiva dos dados será utilizada. Requisitos de desempenho do modelo serão tidos em conta como objetivo secundário.

A aprendizagem profunda revela-se uma solução lógica para este problema de previsão de tempo de vida restante, não necessitando nem de especialistas capazes de formular um modelo matemático do comportamento físico, nem de engenharia de indicadores de saúde. Com mais otimizações, a abordagem RNN baseada em dados pode ser implementada em dispositivos embebidos de baixa potência para previsões em tempo real.

Assim, o objetivo e motivação desta dissertação é o estudo de estratégias de inteligência artificial, em específico, aprendizagem profunda, que prevejam RUL diretamente de dados crus de sensores.

Primeiramente, foi estudado um conjunto de trabalhos visando *"fornecer orientação"*. Nomeadamente em [5], é fornecido um fluxograma que guia a escolha do modelo de inteligência artificial, assim como, uma lista dos modelos mais comuns para previsões de RUL, com os respetivos artigos que os suportam. Por fim, uma discussão das vantagens e desvantagens de cada modelo e as metodologias de treino dos modelos são esquematizados detalhadamente.

Segundo [5], são necessários três passos para a construção de um modelo de previsão de RUL:

- **Seleção do Modelo** - Seleção de modelos de acordo com as características do conjunto de dados;
- **Reconstrução do Modelo** - Um conjunto de iterações de treino bem definidas, obtendo uma boa generalização tanto no conjunto de dados de treino e também nas amostras nunca vistas.
- **Previsão de RUL** - Estimação de RUL em tempo real de novas amostras.

"(...) by expanding the lifetime of vital systems, the aviation industry can safely expand the operational time of aircraft, thus reducing materials disposal and recycling needs."

António Santos of Active Space Technologies

1.2 Objetivos

Os objetivos principais desta dissertação são:

- O estudo dos requisitos de um sistema PHM, usando como referencia o projeto iBearings.
- Fazer um estudo bibliográfico do state of art de algoritmos de inteligência artificial, desenhados para processamento de dados sequenciais, experimentação dos seus hiperparâmetros e comparação dos resultados.
- Utilização de métricos objetivas para classificar o modelo preditivo numa perspetiva de modelo PHM.

Em termos de objetivos de uma perspetiva PHM, pode ser descrito da seguinte forma:

A monitorização das condições dos rolamentos é vital quando se trata da aplicação de máquinas rotativas. De facto, as consequências de falha dos rolamentos em aplicações críticas em aviões podem ser catastróficas.

No entanto, é difícil prever o momento de falha devido ao seu tamanho reduzido e à sua inacessibilidade uma vez instalados no motor.

O objetivo final é monitorizar o rolamento em tempo real, este sujeito a condições adversas, caracterizado por altas temperaturas. O sistema proposto aplica um algoritmo avançado capaz de integrar dados sensoriais de acelerações de baixa frequência, a fim de calcular a previsão do tempo até à falha, sem intervenção de qualquer operador de teste. Um sistema avançado de monitorização constituído por sensores que avalia com precisão a saúde de rolamentos criticamente importantes que operam em ambientes adversos de alta temperatura permitirá que os geradores de arranque de motores de avião funcionem com segurança em velocidades mais altas, reduzindo assim o consumo de combustível e contribuindo para os objetivos da ACARE.

ACARE Sigla para Advisory Council for Aviation Research and Innovation in Europe (Conselho Consultivo para Pesquisa e Inovação em Aviação na Europa), é uma organização dedicada a impulsionar a pesquisa e a inovação no setor da aviação na Europa. Foi estabelecida em 2001 como resultado da colaboração entre a Comissão Europeia, empresas, universidades e organizações governamentais e não governamentais. A ACARE visa promover o desenvolvimento sustentável da aviação por meio de uma abordagem holística, integrada e colaborativa. Tem os seguintes objetivos [6]:

- Até 2050, alcançar a neutralidade climática na aviação com base em ferramentas e modelos validados e globalmente aceites, no contexto completo da sustentabilidade (ambiental, económica e social), alinhado com objetivos de Desenvolvimento Sustentável (ODS) das Nações Unidas, integrando o conceito de economia circular para ser um contribuinte equitativo, com outros modos de transporte, para uma mobilidade plenamente neutra em termos climáticos

- Garantir a sustentabilidade e outras necessidades dos cidadãos não apenas como viajantes e clientes, mas também como destinatários dos impactos externos da aviação, como ruído e outras perturbações. Este objetivo também inclui atender às necessidades do sistema educacional e da força de trabalho qualificada.

- Projetar e aplicar os instrumentos necessários para manter a liderança global e a competitividade da indústria da aviação europeia em toda a cadeia de suprimentos, incluindo investigadores, fabricantes, infraestrutura e operadores de aeronaves, e prestadores de serviços, fornecendo os produtos e serviços mais inovadores, eficientes em custo e de maior qualidade, além de desenvolver e sustentar o capital humano, conhecimento e habilidades necessários.

Desta forma, a ACARE pretende fortalecer a indústria da aviação europeia, impulsionando a inovação, a sustentabilidade e o crescimento económico, ao mesmo tempo, em que promove a segurança e a eficiência operacional. A organização desempenha um papel fundamental na definição de uma visão estratégica de longo prazo para o setor, incentivando a colaboração e a cooperação entre diferentes partes interessadas e impulsionando o avanço tecnológico na aviação na Europa.

No que diz respeito ao projeto PHM, existem três componentes principais:

- **Componente de Hardware:** Consiste nos sensores para auxiliar na monitorização da degradação estrutural dos rolamentos ao longo de sua vida útil;

- **Componente Digital:** Seleção do MCU (Microcontrolador) apropriado e desenvolvimento da comunicação com o hardware;

- **Componente de Software:** Desenvolvimento e avaliação da confiabilidade de um algoritmo preditivo capaz de funcionar em recursos limitados.

O projeto ambiciona obter uma maior precisão da janela ideal para o serviço de substituição dos rolamentos do gerador de arranque do motor, e está alinhado com a mudança na estratégia de manutenção na indústria aeroespacial. Esta estratégia inclui o planeamento e manutenção de forma proativa, realizando as intervenções necessárias antes que ocorra uma falha, diminuindo a dependência de intervalos fixos de manutenção. Para isto, uma transição para soluções baseadas em sensores, em que a condição e o desgaste dos componentes tornam-se os fatores determinantes para revisão ou substituição, é necessária. [7]

O objetivo da PHM consiste em utilizar técnicas de previsão para estimar o tempo restante de vida útil de um sistema e os seus componentes, com base em informações histórias, condições operacionais atual e futuras. Tal permite a implementação de medidas preventivas e a mitigação de riscos, evitando assim catástrofes e garantindo a operação segura e confiável dos sistemas.[8]

A informação histórica inclui dados de falhas histórica de sistemas semelhantes e registos de eventos. A condição de operação atual pode ser na forma de uma característica derivada de medições de sensores que indica o estado de saúde atual do sistema. A condição operacional futura refere-se a fatores operacionais e ambientais que podem afetar o estado futuro do sistema. Essas

informações podem ser obtidas com base em opinião de especialistas ou por meio de análise dos planos de produção ou por algoritmos de previsão. [9] [10] A PHM oferece diversos benefícios, incluindo:

- Evitar falhas catastróficas, manutenção não programada e perda de produção.
- Reduzir os custos de manutenção ao minimizar o número de intervenções desnecessárias e revisões de máquinas.
- Aumentar a vida útil dos componentes fornecendo informações avançadas sobre a gravidade da falha a ser corrigida.
- Reduzir o consumo de combustível ao aumentar com segurança as RPM (rotações por minuto).

Tendo em consideração estes aspetos, prever a falha dos rolamentos em geradores elétricos de aviação reduzirá custos de manutenção e aumentará a segurança, simultaneamente, reduzirá a pegada de carbono.

Além dos benefícios de manutenção e segurança, a previsão antecipada da falha dos rolamentos também tem um impacto positivo no meio ambiente. Ao evitar falhas catastróficas, reduz-se o risco de derramamentos de combustível, danos ambientais e emissões adicionais de carbono resultantes de reparos de emergência. Além disso, uma manutenção mais eficiente e otimizada reduz o tempo de inatividade e, consequentemente, os custos de operação.

Portanto, prever a falha dos rolamentos em geradores elétricos de aviação com um período significativo de antecedência, com alta confiabilidade estatística, traz benefícios em termos de manutenção, segurança, sustentabilidade ambiental e económicos.

Assim, segundo os objetivos de modelos de inteligência artificial procedeu-se à seleção de modelos de aprendizagem computacional adequados, recolha e analise e pre-processamento de conjuntos de dados, (*data-sets*).

1.2.1 Seleção de modelos

Segundo a esquematização proposta em [5] da seleção de modelos de previsão de RUL, o processo envolve considerar quatro critérios principais. Em particular, conjuntos de dados disponíveis, a sua complexidade, a sua variabilidade e a complexidade dos modelos. Esta sub-secção é dedicada a explicar esses quatro critérios.

Integridade do conjunto de dados Este é um requisito para desenvolver um modelo *data driven* supervisionado para previsão de RUL. Com esta afirmação entende-se a existência de conjuntos de dados com amostras de treino e respetivas classificações completos. Se um conjunto de dados estiver completo, constituído por amostras reais em tempo real do sistema, uma direta entre os dados e a previsão do tempo útil de vida restante pode ser feito com uma simples regressão. No caso de testes acelerados, modelos generativo (*GMs - generative models*) devem ser escolhidos

para completar o decorrer da degradação. Métodos de criação de indicadores de saúde são úteis para devir o conjunto de dados em diferentes fases de degradação.

Complexidade dos dados é classificada usando *data 3V* que representa - *Volume; Velocidade; Variedade* - Volume define-se pela quantidade de dados disponíveis; Velocidade - Como os dados são adquiridos, frequência e período de amostragem; Variedade - A diversidade de sensores e media pela qual o conjunto de dados é constituída. Entende-se por variedade do formato, como estrutura interna, linearidade e natureza sequencial ou pontual. No caso de os dados apresentarem grande complexidade, nomeadamente na sua dinâmica interna, como acontece em séries temporais, deve-se utilizar um modelo que possa aprender a partir dessa estrutura interno, como é o caso de aprendizagem profunda (*DL - deep learning*).

Variabilidade dos dados refere-se ao dinamismo e não linearidade que um conjunto de dados pode apresentar, nomeadamente em séries temporais. Ignorar esta característica irá provocar grandes défices na performance de um modelo. Assim, modelos com capacidades adaptativas em tempo real durante o treino, por exemplo mecanismos de esquecimento permitem controlar a generalização e divergência do processo de treino de um modelo de AI.

Complexidade do modelo está geralmente relacionada com a arquitetura do modelo, que depende especificamente nos parâmetros de aprendizagem (pesos, *bias* e hiperparâmetros) que formam a função de aproximação. Quanto mais parâmetros um modelo tiver mais difícil será a sua otimização. Esta pode ser feita automaticamente através de técnicas como *grid search*, com um elevado custo computacional ou então por intervenção humana. Assim, a solução mais simples, que solucione o problema que estamos a abordar.

Com esta análise de [5], concluímos que a seleção do modelo está fortemente relacionada com o tipo de dados e recursos disponíveis.

Os dados C-MAPSS são o resultado de um modelo de simulação de um tipo específico de motor turbofan. Neste processo de simulação baseado em modelo, medições de funcionamento até a falha são registadas sob diferentes condições. Durante a simulação, também são considerados ruído de diferentes fontes para imitar cenários reais. Compreendemos que os dados são massivos, enquanto o ruído complexifica esses dados. Além disso, a mudança contínua nas condições de funcionamento leva a uma maior dinâmica nos dados que são registados em séries temporais num processo de sequência online. Nesse contexto, uma vez que os dados estão disponíveis, complexos, dinâmicos e orientados online, seguindo o fluxograma proposto em [5], a metodologia de aprendizagem mais apropriada deve ser um modelo de aprendizado profundo adaptativo online capaz de ser atualizado dinamicamente para lidar com a análise de séries temporais.

1.3 Estrutura da Dissertação

Para além da introdução, esta dissertação contém mais cinco capítulos. No capítulo 2, é feito um enquadramento mais detalhado do projeto iBearings e a descrição do trabalho realizado pela Active Space e empresas parceiras. Para além disso, são apresentados os principais componentes de um sistema de manutenção preventiva, assim como fluxogramas e uma análise de nível alto.

No capítulo 3, são introduzidos os conceitos de manutenção preventiva, os seus objetivos, enquadramento do tema, e as vantagens apresenta como alternativa à manutenção periódica.

No capítulo 4, desenvolvem-se os conceitos base de aprendizagem computacional até chegar a redes neuronais artificiais profundas e LSTM.

No capítulo 5, é apresentado o conjunto de dados utilizados, a sua análise e pré-processamento. Assim como a descrição do *set-up* experimental que os originou. Usando estes fundamentos, é proposta uma solução que responde ás necessidades do problema segundo a literatura estudada.

No capítulo final 6, é feito um apanhado de toda a dissertação e discutido o objetivos cumpridos e possíveis melhoramentos e trabalhos futuros.

Capítulo 2

iBearings

2.1 Componentes de Hardware e Digital

Com o iBearing, os dados coletados por sensores embutidos no motor durante o voo permitem que os operadores verifiquem temperatura, pressão, vibração, aceleração e ruído a qualquer momento, tornando viável detetar a degradação dos rolamentos, desgaste mecânico e o início de micro-fissuras. Utilizar dados em tempo real para monitorizar o estado dos rolamentos e dessa forma possibilita a adoção de medidas preventivas adequadas de manutenção. [7]

Ao monitorizar continuamente os dados obtidos pelos sensores, é possível identificar alterações nos padrões de temperatura, vibração e outros parâmetros relevantes que possam indicar problemas iminentes nos rolamentos. Essas mudanças podem ser sinais de degradação ou desgaste excessivo, permitindo a intervenção preventiva antes que ocorra uma falha catastrófica.

Além disso, ao acompanhar a ocorrência de micro-fissuras, é possível detetar problemas nas fases iniciais, antes que se tornem críticos. Isso permite a realização de reparos ou substituição dos rolamentos no momento adequado, evitando danos adicionais e aumentando a vida útil do equipamento.

A utilização de dados em tempo real para monitorizar o estado dos rolamentos também permite uma abordagem mais pro-ativa na manutenção. Com base nos dados coletados, é possível tomar medidas preventivas, como lubrificação adequada, ajustes de componentes e programação de intervenções de manutenção programada. Isso ajuda a evitar falhas inesperadas, reduzindo o tempo de inatividade e os custos associados à manutenção corretiva.

Em resumo, o uso de dados coletados por sensores embutidos no motor durante o voo, por meio do iBearing, possibilita a monitorização contínua do estado dos rolamentos, permitindo a deteção precoce de problemas e a adoção de medidas preventivas adequadas de manutenção. Isso contribui para a segurança operacional, aumenta a vida útil dos rolamentos e reduz o tempo de inatividade e os custos de manutenção.

A equipa do iBearing obteve sucesso no desenvolvimento de um dispositivo protótipo para monitorização in situ de rolamentos, capaz de suportar todo o espectro de valores de velocidade de rotação ao longo das diferentes fases do voo, geralmente de 10.000 a 30.000 RPM. O desempenho

do protótipo foi testado em temperaturas variando entre 150°C e 200°C e demonstrou resultados encorajadores.[7]

O tamanho compacto, a operação independente e a resistência ao ambiente da turbina aeronáutica foram essenciais para o projeto de um sistema de monitorização de condição capaz de medir a vida útil segura do rolamento.[7]

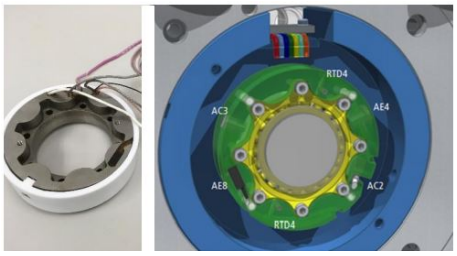


Figura 2.1: Design do iBearing e posicionamento dos sensores.

Figura 2.2: Dois módulos analógicos para sensores de emissão acústica e acelerômetros.

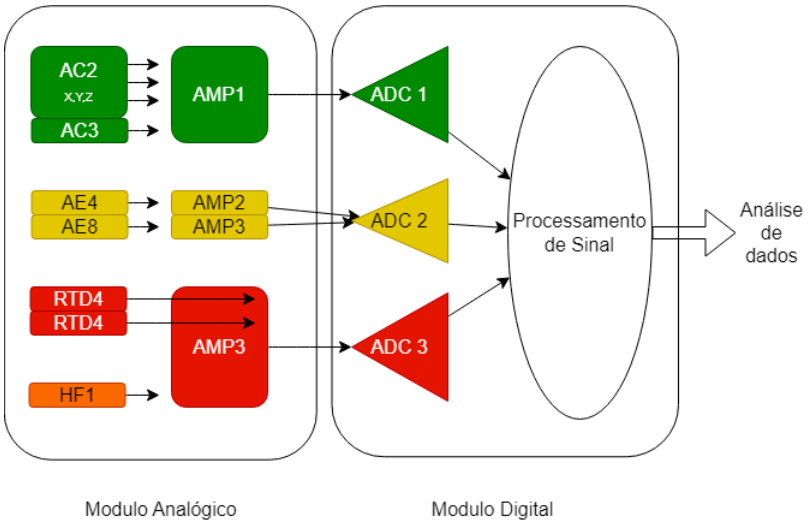


Figura 2.3: Sistema analógico de prova de conceito.
AC - sensor de aceleração; AE - sensor de ruído; RTD - sensor de temperatura
ADC - conversor de sinal analógico para digital; AMP - amplificador de sinal

Sensor	**Taxa de Amostragem**
AC2 - Acelerômetro triaxial (125°C)	200 kS/s
AC3 - Acelerômetro uniaxial (250°C)	200 kS/s
AE4 - Sensor de Emissão Acústica (117°C)	2 MS/s
AE8 - Sensor de Emissão Acústica (180°C)	2 MS/s
RTD4 - Detector de Temperatura por Resistência (250°C)	100 S/s

Tabela 2.1: Sensores usados no projeto.

Este protótipo validou a prova de conceito. Foi crucial escolher componentes que pudessem suportar o ambiente rigoroso que esta aplicação requer. Assim o design do iBearing incorpora uma estrutura compacta e robusta para garantir sua adequação ao exigente ambiente dos motores aeronáuticos. O dispositivo é projetado para suportar altas velocidades de rotação, temperaturas extremas e outras condições desfavoráveis.

Quanto ao posicionamento dos sensores, o sistema iBearing é colocado estrategicamente dentro do motor para monitorizar efetivamente os rolamentos. Os sensores são instalados próximos dos rolamentos para capturar dados relevantes sobre a sua condição. Isso permite o monitorizar e análise em tempo real os parâmetros cruciais, como temperatura, vibração e ruído, permitindo assim inferir informações valiosas sobre a saúde e a vida útil restante dos rolamentos, permitindo manutenção pro-ativa e minimizando o risco de falhas inesperadas.

Também em termos de processamento de sinal, foi importante filtrar e amplificar a saída dos sensores, pois o ruído induzido pela vibração e temperatura poderia deteriorar a utilidade dos dados de saída. Além disso, o processamento de dados foi realizado para alimentar o algoritmo de aprendizagem computacional, pois os dados brutos são inúteis, sendo necessária a extração de características.

Neste trabalho de dissertação foi proposta uma solução utilizando aprendizagem profunda, assim é possível utilizar os dados brutos. Mais detalhes à cerca de software e PHM são desenvolvidos em detalhe nos próximos capítulos.

Em resumo, o design do iBearing e o posicionamento dos sensores são projetados para fornecer informações precisas, oportunas e acionáveis sobre a condição dos rolamentos, contribuindo para uma maior segurança, eficiência operacional e economia de custos de operação em aplicações de motores aeronáuticos.

No que diz respeito à componente digital, existe uma ampla seleção de MCUs ou microcontroladores e sistemas embarcados disponíveis no mercado. Muitos deles possuem tamanho reduzido e grande poder de processamento, portas e protocolos de comunicação, além de memória ampla. No entanto, essa seleção torna-se mais restrita quando buscamos uma solução capaz de funcionar em condições adversas, como este projeto exige.

A configuração atual utilizada pela Active Space é uma combinação do DSC SM320F28335 e da memória flash SM28VLT32-HT (32-Mbit High-Temp Flash Memory com Serial Peripheral Interface (SPI) Bus). Essa combinação proporciona as seguintes características:

- High-Performance Static CMOS Technology
  - Up to 150 MHz for TC = –55°C to 125°C and Up to 100 MHZ for TC = 210°C
  - 1.9-V Core, 3.3-V I/O Design
- High-Performance 32-Bit CPU
- On-Chip Memory
  - 256K × 16 Flash, 34K × 16 SARAM
  - 1K × 16 OTP ROM
- Plus 32-Megabit Flash for High-Temperature SUPPORTS EXTREME TEMPERATURE Applications (125°C), Available in Extreme (–55°C to 210°C) Temperature Range
- Serial Port Peripherals
  - Up to 2 CAN Modules
  - Up to 3 SCI (UART) Modules
  - Up to 2 McBSP Modules (Configurable as SPI)
  - One SPI Module
  - One Inter-Integrated-Circuit (I2C) Bus

Capítulo 3

Manutenção Preditiva e Gestão de Saúde

A manutenção preditiva e gestão de saúde, geralmente abreviado como PHM (do inglês, *Prognostics and Health Management*), é uma área de conhecimento que se concentra na aplicação de tecnologias avançadas para prever, diagnosticar e gerir falhas e problemas em sistemas complexos. Esta envolve o uso de tecnologias como sensores, algoritmos de análise de dados e inteligência artificial para monitorizar continuamente o estado de sistemas complexos, com potencial aplicabilidade em múltiplas disciplinas e indústrias.

Tirando partido de dados em tempo real e passados de subsistemas, permite inferir o estado de desgaste de componentes e prever falhas do sistema. A informação que os utilizadores podem usar para detetar e identificar potenciais problemas antes que ocorram, para poderem ser corrigidos antes que causem danos ou interrupções. É uma abordagem proativa para a gestão de sistemas que ajuda a reduzir custos, aumentar a segurança e a fiabilidade, e melhorar o desempenho de sistemas complexos. Devido à sua importância é uma área de estudos com muito interesse e estudada nos últimos anos.

Este capitulo servirá como contextualização e introdução do tema de manutenção preventiva e enquadramento no problema de previsão de tempo restante de vida de rolamentos.

3.1 Contextualização

A manutenção preditiva e a gestão de saúde é uma disciplina da engenharia cujo objetivo é minimizar custos de manutenção pela avaliação, prognóstico, diagnóstico e gestão da saúde de sistemas. Independentemente da aplicação, um objetivo comum de PHM é a capacidade de traduzir dados crus de sensores em informação num formato que facilite a tomada de decisões de manutenção. Tem-se visto um aumento da sua popularidade em indústrias como aeroespacial, fabricas inteligentes, transportes e centrais elétricas, graças a avanços em poder de computação e tecnologia de sensores [11].

Apesar dos esforços para evitar falhas, reduzir o tempo de paragem dos equipamentos e agendamento de manutenções, os custos associados a falhas são ainda elevados. Usarei o exemplo da aviação.

Segundo o relatório anual da Royal Dutch Airlines (KLM), os custos de manutenção desde 2013 e 2017 são 669, 665, 934, 1009, e 994 milhões de euros respetivamente. Estes valores correspondem a cerca de 11% a 18% dos custos totais de operação. Em setembro de 2003, a Comissão Europeia reportou que manutenção e reparação correspondem a 40% do custo total de um veículo [12].

Os custos consequentes à falha de equipamentos são altos. DHL estimou que custo de um *AOG (Aircraft On Ground)* devido a falhas técnicas de um A380 Airbus ronda 925.000 euros por dia. Se um acidente grave ocorrer devido a estas falhas, os custos são incalculáveis.

Um sistema PHM robusto deve conseguir detetar falhas incipientes em componentes ou sistemas, realizar diagnósticos de falhas, prognósticos de falhas e gestão de saúde. Os prognósticos de falhas são o cerne do PHM. Referem-se especificamente à fase envolvida na previsão do comportamento futuro e à vida útil restante do sistema em termos de estado operacional atual e ao agendamento das ações de manutenção necessárias para manter a saúde do sistema. A vida útil restante em inglês de "Remaining Useful Life (RUL)"é, tipicamente, uma variável aleatória e desconhecida, e como tal, deve ser estimada a partir de fontes de informação disponíveis, como a informação obtida na monitorização de condição e saúde, em inglês, Condition Monitoring (CM).

Geralmente um projeto de PHM envolve os seguintes passos [1]:

- Definir componentes críticos;
- Seleção de conjunto de sensores apropriados para a monitorização de condição e saúde;
- Prognóstico da evolução do conjunto de características extraídas dos dados observados;
- Metodologia de prognóstico, ferramentas e diretrizes de avaliação.

Baseado nos requisitos do sistema ou produto, PHM pode ser classificadas em duas principais categorias, PHM 'On-line' (em tempo real) ou PHM off-line. A maioria das aplicações onde a segurança é um requesito principal requerem PHM 'on-line', o qual é o caso do problema desta dissertação, havendo capacidade de monitorização dos dados em tempo real, usando um ecossistema de sensores.

Baseado nas abordagens e metodologias de PHM, podem-se geralmente classificar em quatro categorias: baseada em confiabilidade, baseada em modelo, baseada em dados e híbrida [13]. Cada abordagem tem as suas próprias vantagens e desvantagens.

Existem três questões principais a serem consideradas ao construir um PHM robusto: uma estimativa do estado de saúde atual, previsão de um estado futuro com o tempo até a falha e determinação do impacto de uma falha no desempenho de um sistema. Para os profissionais, selecionar e implementar a tecnologia PHM baseia-se na sua capacidade e conhecimento sobre abordagens,

ferramentas, etc., de PHM. Modelos que envolvem interpretações matemáticas, suposições e aproximações tornam o PHM difícil de entender e aplicar. O conhecimento prévio para implementar o PHM em sistemas complexos é crucial para construir sistemas altamente confiáveis. [1]

Neste capitulo será discutido cada um dos tópicos a traz mencionados. A principal referência foram, *A Review: Prognostics and Health Management in Automotive and Aerospace* [14], *Prognostics and health management for maintenance practitioners - Review, implementation and tools evaluation.* [1] *Prognostics: a literature review.* [13].

3.1.1 Abordagens a PHM e metodologias

Como anteriormente referido, as abordagens de prognósticos são classificadas em quatro tipos, nomeadamente i) abordagens baseadas em confiabilidade, ii) abordagens baseadas em modelo, iii) abordagens baseadas em dados e iv) abordagens híbridas. Cada abordagem tem as suas vantagens e limitações. De forma geral, a complexidade, custo e precisão das técnicas de prognósticos são inversamente proporcionais à sua aplicabilidade. Aumentar a precisão do algoritmo de prognóstico com baixo custo e complexidade é um grande desafio.

Os arquitetos de sistemas de prognóstico podem servir-se desta classificação como auxílio na escolha da abordagem mais conveniente. Dependendo do conhecimento perito sobre o funcionamento do sistema ou do conjunto de dados disponível. O principal objetivo desta classificação é criar um procedimento padrão para a criação de sistemas de prognóstico. A seguir explicarei cada uma, qual foi a escolhida para este projeto e qual a razão para que abordagens baseadas em dados seja a mais indicada.

Abordagens baseadas em confiabilidade Também conhecido por abordagens de métodos de confiabilidade estatísticos, este tipo de abordagens são a forma mais simples de prognóstico de falhas, pois. exigem menos informações detalhadas do que outras abordagens prognosticas. Elas são baseadas na distribuição de registo de eventos de uma população de itens idênticos.

Muitas abordagens de confiabilidade tradicionais, como distribuições exponenciais, Weibull e log-normal, são usadas para modelar a confiabilidade do sistema. Estas abordagens não conseguem prever o RUL do componente ou sistema e geralmente não são utilizados em sistemas críticos, pois geralmente não funcionam em tempo real. Estes modelos são produzidos em massa e têm uma aplicabilidade muito versátil.

Abordagens baseadas em modelo Esta é a abordagem mais importante em PHM devido à sua exatidão e desempenho em tempo real. Trata-se de um método determinístico que permite a estimativa e a previsão dos estados dinâmicos.

Nesta abordagem, é utilizado um modelo físico /matemático do sistema monitorizado e um conjunto de características residuais. Estas características são extraídas entre a comparação das medições em tempo real do sistema monitorizado e das previsões obtidas pelos modelos matemáticos.

Para estabelecer este modelo, é necessária uma compreensão aprofundada da física do sistema/componente e por isso a sua confiança geralmente diminui à medida que aumenta a complexidade do sistema. No entanto, estes métodos não necessitam de conjuntos de dados grandes nem de dados de falha do sistema, sendo estes últimos muitas vezes difíceis de obter.

As abordagens comuns baseadas em modelos são os filtros de Kalman (KF), filtros de Kalman estendidos (EKF), filtros de Kalman não lineares (UKF) e filtros de partículas (PF). Os filtros de Kalman são um conjunto de algoritmos matemáticos usados para estimar o estado de um sistema baseado em medições incompletas e imprecisas. O algoritmo funciona em dois passos: primeiramente uma previsão de uma certa variável (estado do sistema) é inferida a partir da última amostra e a sua distribuição probabilística, segundo um modelo estipulado (por exemplo, equações de movimento); seguidamente, quando uma nova amostra está disponível, a previsão é atualizada. Assim, funciona recursivamente, necessitando apenas da presente observação, o estado previamente calculado e a matriz de incerteza para calcular a variável (estado do sistema), tendo a vantagem de funcionar em tempo real.

Os filtro de partículas é uma técnica baseada em simulação que utiliza inúmeras partículas para representar o estado atual do sistema. Ele é adequado para modelos não lineares e pode lidar com erros não gaussianos. Enquanto os filtros de Kalman usam uma equação de atualização recursiva, os filtro de partículas atualizam as partículas segundo a sua probabilidade. O filtro de partículas é mais flexível e pode lidar com situações em que a distribuição de probabilidade é altamente não linear ou multimodal. No entanto, o filtro de Kalman e as suas variantes são computacionalmente mais eficiente e é mais adequado para sistemas com modelos lineares ou ligeiramente não lineares.

Abordagens baseadas em dados Os Modelos baseados em dados são ferramentas de análise que utilizam dados coletados dos sistemas e máquinas para compreender o seu comportamento, desempenho e saúde. Esses modelos são construídos a partir de dados de sensores e outras fontes de informação, técnicas de inteligência artificial (AI), que dispõe de múltiplas ferramentas que podem ser usadas para monitorizar a condição de uma ampla variedade de sistemas e equipamentos, necessitando de pouca adaptação. Eles podem aprender a partir de dados históricos e em tempo real, identificando padrões e tendências que podem indicar problemas iminentes antes mesmo que eles ocorram.

As principais vantagens desta abordagem reside na facilidade de aplicação prática e diversa, principalmente quando é difícil obter um modelo físico do sistema. O baixo custo de desenvolvimento destes algoritmos e o facto da sua confiança e eficácia computacional não diminuírem com o aumento da complexidade do sistema são as principais vantagens desta abordagem.

Por outro lado, as principais desvantagens que apresenta relativamente às abordagens baseadas em modelos é a necessidade de conjuntos de dados grandes e diversificados, incluindo dados históricos para o treino dos modelos e dados em tempo real que monitorizem o sistema / componente. Em princípio, quantos mais exemplos de falha o conjunto de dados de treino tiver, melhor será a precisão do modelo. No entanto, estes dados são de difícil acesso e caros, pois estes eventos são raros na indústria.

As abordagens PHM baseadas em dados são classificadas em duas categorias: i) abordagem estatística e ii) aprendizagem computacional.

A primeira tira partido de alguns parâmetros estatísticos como média, variância, mediana, etc. Para fazer decisões baseando-se em distribuições probabilísticas estudadas à 'priori' do conjunto de dados. Esta categoria de abordagem é utilizada principalmente para deteção de anomalias, alguns exemplos são: testes de hipóteses, análise de variância (ANOVA), máxima verosimilhança (ML), modelo de misturas de gaussianas (GMM), modelo oculto de Markov (HMM), Análise de Componentes Principais (PCA).

A segunda faz previsões usando dados adquiridos (dados de bom funcionamento ou falha), convertendo o conjunto de dados num modelo que em conjunto com os dados adquiridos em tempo real permitem fazer previsões da evolução do estado do sistema.

Esta última foi a abordagem escolhida como solução para o problema desta dissertação. O cálculo de RUL de rolamentos é um problema que beneficia da capacidade de análise de séries temporais de alguns modelos de inteligência artificial, como as RNN.

Aprendizagem computacional e inteligência artificial é um ramo de engenharia com múltiplas aplicações, incluindo PHM. Uma contextualização sobre o tema e explicação dos modelos utilizados serão expostos no próximo capítulo.

Abordagens híbridas Como anteriormente mencionado, cada abordagem apresenta as suas vantagens e desvantagens. A abordagem híbrida pretende fundi-las minimizando as limitações de cada e tirando partido das vantagens. Obtendo melhores estimativas do estado de desgaste e previsões de RUL, combina as vantagens de cada uma das outras abordagens. Complementa a falta de conhecimento especializado sobre o sistema da abordagem baseada em modelo e a eventual escassez de dados da abordagem baseada em dados.

3.1.2 Mecanismos e deteção de falha

Existem muitos fatores que causam a degradação dos componentes do sistema ao longo do tempo, resultando na perda do seu desempenho inicial, e que, portanto, precisam ser considerados na criação de um PHM. A deteção do estado de saúde é o processo de detetar e reconhecer falhas incipientes e/ou anomalias a partir dos dados CM (*Condition Monitoring*).

Falha nem sempre significa falha física, esta é definida quando um sistema não consegue realizar uma certa função. Assim, esta depende dos requisitos de operação do sistema.

A deteção de falhas é tipicamente baseada na quantificação das inconsistências entre o comportamento real e o esperado do sistema em condições nominais. A Figura 3.1 ilustra a propagação da falha de um componente com base nos dados CM. [1]

Na figura 3.1, o CM do componente aumenta com o tempo à medida que o componente se degrada. Conforme ilustrado, a evolução do estado de saúde do componente pode ser dividida em 3 fases: fase 1 (T0 < T1), onde o componente está no estado saudável; fase 2 (T1 < T2), onde o componente está no estado de degradação; e Fase 3 (T2 <), onde o componente está no estado de falha. Pré-definir os limiares (T1 e T2) é um desafio difícil, que requer experiência significativa por

parte dos profissionais. Dados históricos de monitorização de condição também podem ser usados para comparar e definir novos limiares para o mesmo tipo de componentes, assim como recorrer a ferramentas de inteligência artificial, como será discutido nas próximas secções. A estimativa do tempo até à falha (RUL) é realizada na Fase 2 após o processo de deteção, onde as atividades de manutenção são planeadas com base no tempo estimado até à falha. Portanto, a deteção precoce da falha do componente é importante. [1]

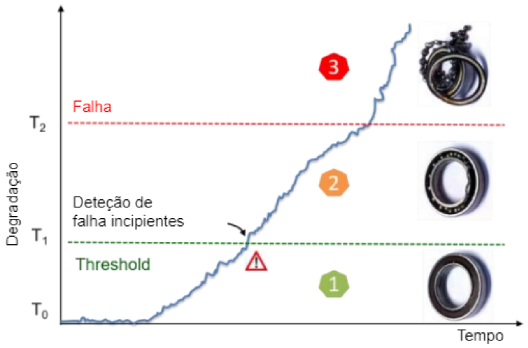


Figura 3.1: Divisão da série temporal dos dados de vibração de um rolamento nas suas diferentes fases de degradação. [1]

Os mecanismos de falha podem ser processos físicos, químicos, termodinâmicos ou outros resultando em falha. Na PHM, conhecer os mecanismos de falha é fundamental para identificar e selecionar características nos métodos baseados em dados. Os mecanismos de falha são categorizados como mecanismos de sobrecarga ou mecanismos de desgaste. A falha por sobrecarga ocorre devido a uma condição de carga única que excede uma propriedade fundamental de resistência. A falha por desgaste ocorre como resultado de danos cumulativos relacionados com as cargas aplicadas ao longo do tempo [15].

O conhecimento desses mecanismos, e especialmente o efeito das cargas dominantes na falha, é essencial para entender por que, como e quando os componentes falham e como isso pode ser prevenido. Eles são de extrema importância na PHM. O leitor interessado pode encontrar mais detalhes sobre os mecanismos de falha em *Principles of Loads and Failure Mechanisms* [16].

3.1.3 Métricas de desempenho

A existência de métricas permite estabelecer requisitos de desenho, especificações, diretrizes ou características que podem ser usadas consistentemente para garantir que os métodos sejam adequados para seus propósitos. As métricas permitem medir e avaliar a qualidade e eficácia dos métodos, bem como comparar diferentes abordagens e técnicas, auxiliando e guiando o processo de escolha e desenvolvimento dos métodos de PHM.

Este ponto é particularmente importante no campo da PHM, onde a precisão e a confiabilidade dos métodos utilizados podem ter impactos significativos na segurança, disponibilidade e custos operacionais dos sistemas monitorados. As métricas comuns usadas em PHM incluem medidas de confiabilidade, como taxa de falsos positivos, taxa de falsos negativos e probabilidade de deteção, bem como métricas relacionadas à precisão de previsões, como erro médio absoluto e raiz quadrada do erro médio ao quadrado. Baseado nos requisitos de sistema, podemos agrupar estas métricas em três categorias principais: desempenho de algoritmo, computacional e métricas de custo — beneficio. [17]

As métricas de desempenho de algoritmo focam-se em avaliar a precisão e exatidão das previsões.

As métricas de desempenho computacional focam-se em avaliar a desempenho computacional dos algoritmos em termos de requisitos de processamento e memória. São importantes em aplicações de previsão 'online', em que segurança é um requisito principal e em aplicações integradas.

As métricas de custo - beneficio como custo de ciclo de vida e ROI (Retorno sobre o investimento, *Return on investment*) pretendem avaliar os benefícios de usarem estes métodos de prognóstico. Este último está fora dos objetivos para esta dissertação.

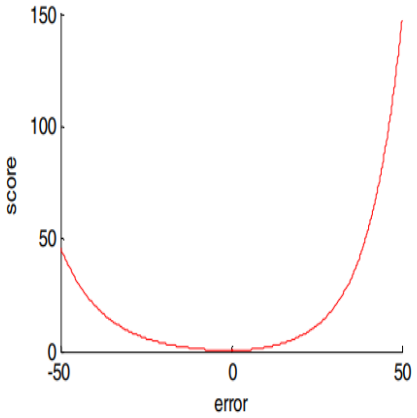
Ao estabelecer métricas para PHM, os arquitetos e utilizadores desses métodos podem ter uma linguagem e estrutura comuns para avaliar e melhorar o desempenho desses sistemas. No entanto, não existe um protocolo estandardizado na comunidade, tornando-se difícil a comparação entre as diferentes abordagens. Com a escassez de conjuntos de dados de referência torna a escolha da abordagem com melhor aplicabilidade no mundo real uma missão difícil. No entanto, existem algumas sugestões na literatura: A Review: Prognostics and Health Management in Automotive and Aerospace [14], Prognostics and health management for maintenance practitioners - Review, implementation and tools evaluation [1], Damage propagation modeling for aircraft engine run-to-failure simulation [18].

No último é sugerido que no contexto de PHM (Prognóstico e Monitorização de Saúde), como o aspeto chave é evitar falhas, geralmente é desejável prever cedo em comparação com prever tarde. Então é proposto um algoritmo de pontuação assimétrico onde as previsões tardias eram mais penalizadas do que previsões antecipadas. Em ambos os casos, a penalidade aumenta exponencialmente com o aumento do erro. A preferência assimétrica é controlada pelos parâmetros a1 e a2 na função de pontuação fornecida abaixo 3.2a, e a Figura 3.2b mostra a pontuação como uma função do erro.

As funções de pontuação assimétricas capturam bem a preferência por previsões antecipadas

$$s = \begin{cases} \sum_{i=1}^{n} e^{-\left(\frac{d}{a_1}\right)} - 1 \text{ for } d < 0 \\ \sum_{i=1}^{n} e^{\left(\frac{d}{a_2}\right)} - 1 \text{ for } d \geq 0, \end{cases}$$

(a) Expressão de pontuação onde: S é a pontuação, d é a diferença entre RUL estimado e real, a1 e a2 são parâmetros que controlam a penalidade para previsões antecipadas.



(b) Gráfico da pontuação em função do erro.

Figura 3.2: Algoritmo de pontuação.

e podem ser ajustadas adequadamente para quantificar o grau dessa preferência. Essas funções permitem atribuir um peso maior às previsões feitas mais cedo, refletindo a importância de evitar falhas e tomar medidas preventivas o mais cedo possível.

Capítulo 4

Inteligência artificial

Os avanços de desempenho em hardware, a disponibilidade de conjuntos de dados, e o desenvolvimento de bibliotecas de código aberto, permitiram avanços no que diz respeito a redes neuronais e aprendizagem computacional [19].

Como referido anteriormente, o método para previsão do RUL escolhido foi aprendizagem profunda, em específico LSTM´s.

Nesta secção será exposta informação introdutória do tema, esquematização e explicação dos algoritmos usados na criação dos modelos preditivos, nomenclatura e medidas de avaliação de desempenho.

4.1 DL - Aprendizagem profunda - Uma introdução

Neste segmento apresentarei uma breve introdução ao tema de aprendizagem profunda. A principal referência foi [20, Goodfellow et al. Chapter 1,6 e 10], com o intuito de contextualizar LSTMs na área de inteligência artificial e explicar o seu funcionamento a partir dos seus conceitos mais simples que a constituem.

Inicialmente criados para reconhecimento de fala e rostos em imagens, a aprendizagem profunda é utilizada para a resolução de problemas onde é difícil descrever formalmente uma solução, sendo esta intuitiva e fácil para seres Humanos.

A aprendizagem profunda permite aos computadores aprenderem por meio de experiência e entender o mundo em termos de uma hierarquia de conceitos, onde cada conceito é definido pela sua relação a conceitos mais simples. Esta abordagem evita a necessidade da descrição formal e manual de operadores Humanos do conhecimento que o computador necessita. Esta hierarquia de conceitos permite que o computador resolva problemas complexos, através das soluções de problemas mais simples. Se desenharmos um grafo onde é representado as relações entre estes conceitos, é visível que estes são construídos em cima uns dos outros, o grafo será profundo, com várias camadas. Por isso esta abordagem a inteligência artificial é denominada aprendizagem profunda. [20].

Numa perspetiva taxonómica, aprendizagem profunda (DL - "deep learning") classifica-se como uma subcategoria de aprendizagem computacional (ML- "machine learning") que por sua vez é uma subcategoria de inteligência artificial (AI - "artificial inteligence").

Várias categorias de AI apresentam vantagens e desvantagens nos seus métodos. Perceber e encontrar aquele que melhor se adequa a um dado problema, é o primeiro passo a tomar.

Muitos problemas de inteligência artificial podem ser resolvidos usando aprendizagem computacional na condição de que um conjunto de atributos sejam fornecidos ao algoritmo de aprendizagem. Ao contrário da programação convencional que necessita que um algoritmo seja desenhado para ter em conta todas as nuances de um sistema, o principal objetivo de aprendizagem computacional é que a maquina aprenda a partir do conjunto de atributos fornecidos na fase de treino. Sendo assim possível captar padrões intrínsecos do problema, que manualmente seriam dificilmente captados e codificados.

O conjunto de atributos é extraído de um conjunto de dados. Este processo é feito usando competências de ciências dos dados, analisando o conjunto de dados, para encontrar o grupo de características que melhor definam a evolução de um dado sistema. Por exemplo, no problema de classificação do estado de desgaste de um rolamento, um conjunto de informações fornecidas à cerca do rolamento irá permitir que modelos de aprendizagem computacional adquiram conhecimento extraindo padrões nos dados. Assim sendo, a representação dos dados e a extração de características do conjunto de dados é importante para a desempenho destes algoritmos.

No entanto, em muitos casos é difícil encontrar características relevantes, uma solução é, portanto utilizar aprendizagem computacional para não só solucionar o mapeamento das características entre entrada e saída, mas também as próprias características. Esta abordagem denomina-se por *representation learning*, extração automática de características. Estes algoritmos são treinados para preservar o máximo de informação possível, mas também que as novas representações obtidas apresentem boas propriedades para serem usados como entrada nos algoritmos de aprendizagem computacional.

No que diz respeito à engenharia de características, métodos de DL apresentam uma grande vantagem, já que este processo é feito pela própria rede, sendo possível usar os dados crus ou com pouco pré-processamento, os algoritmos descobrem não só o mapeamento das características entre entrada e saída, mas também as próprias características.

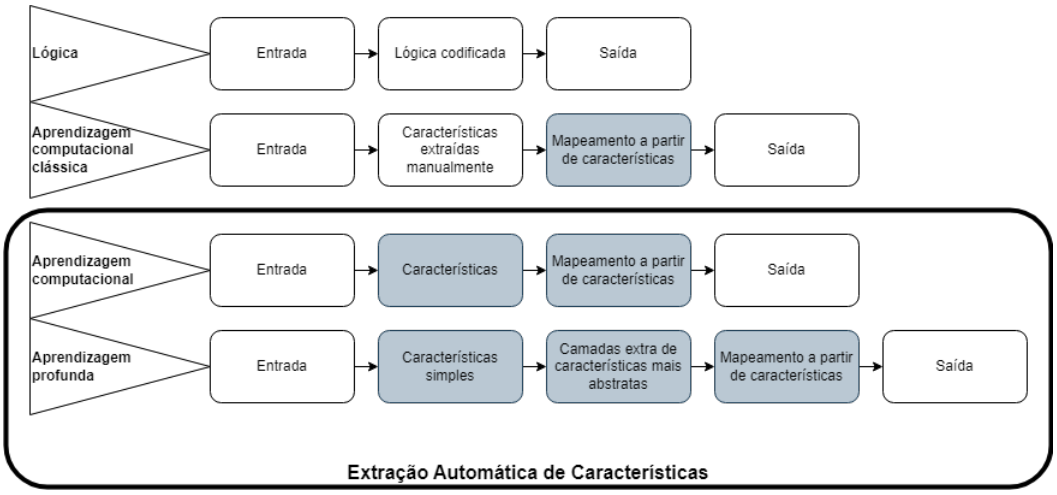


Figura 4.1: Fluxograma mostrando as diferentes partes de sistemas de inteligência artificial relativamente aos diferentes métodos. Componentes em cinzento conseguem aprender a partir de conjuntos de dados.

Nos próximos parágrafos apresentarei conceitos-base e transversais na área de otimização e inteligência artificial:

Aprendizagem supervisionada VS Aprendizagem não supervisionada Aprendizagem não supervisionada consiste num algoritmo que encontrar padrões e estruturas em dados não rotulados, ou seja, não há informações prévias sobre as classes ou categorias dos dados. Neste tipo de aprendizagem, o modelo deve identificar as relações entre os elementos e agrupá-los significativamente. São exemplos de modelos de aprendizagem não supervisionada: *K-Means:* Um dos algoritmos mais populares de 'clusters' (agrupamento), o K-Means visa dividir os dados em K grupos, minimizando a distância entre os pontos dentro de cada grupo e maximizando a distância entre os grupos. *Autoencoders:* São redes neuronais que buscam aprender a codificar e descodificar os dados de entrada. Eles podem ser usados para redução de dimensões, deteção de anomalias e até mesmo geração de dados similares aos originais [21].

Em aprendizagem supervisionada, são usados dados de treino rotulados, partindo do princípio que existe uma relação entre a entrada (dados de treino) e a saída (rótulos). Um algoritmo de aprendizagem supervisionado analisa os dados de treino e produz uma função inferida, que pode ser usada para mapear novos exemplos. Um cenário ideal permitirá que o algoritmo determine corretamente os rótulos para exemplos não vistos.

Problemas de aprendizagem supervisionada podem dividir-se em duas categorias, regressão e classificação. Em problemas de regressão, tentamos prever resultados num espectro contínuo de valores, querendo isto dizer que tentamos mapear valores de entrada, usando uma função continua. Por outro lado, problemas de classificação, tentamos prever resultados num conjunto discreto de valores, por outras palavras, mapeamos (classificar) valores de entrada usando categorias discretas.

Sucintamente, se uma dada variável que queremos prever é contínua, chamamos de um problema de regressão. Se pelo contrário, tentamos prever uma variável que apenas pode tomar valores num conjunto discreto de valores, então trata-se de um problema de classificação.

Usando dados da série temporal de valores de sensores instalados em rolamentos, tem-se por objetivo prever o seu tempo de vida restante (RUL). O RUL em função do 'output' dos sensores, resulta em valores de saída contínuos, logo trata-se de um problema de regressão.

Definição de nomenclatura De forma estabelecer notação futura; $x^{(i)}$ representa as variáveis de entrada, $y^{(i)}$ representa as variáveis de saída, isto é, os valores que tentamos prever. Um par $(x^{(i)}, y^{(i)})$ é denominado como exemplo de treino, uma lista de m exemplos de treino, $(x^{(i)}, y^{(i)}); i = 1, ..., m$ é o conjunto de dados de treino, *training set*, que usaremos para treinar um modelo.

O expoente "(i)", representa o índex de um dado exemplo de treino dentro do*training set*. Usaremos X para denotar o espaço de valores de entrada e Y para denotar o espaço de valores de saída.

De uma forma mais sucinta, o nosso objetivo é, dado um *training set*, aprender uma função h : X $\rightarrow$ Y para que h(X) seja uma "boa"previsão quando comparado ao valor de Y. A função h é chamada de hipótese.

Esquematicamente:

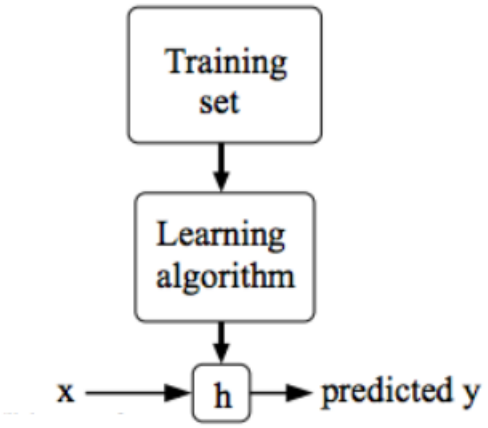


Figura 4.2: Modelo de aprendizagem supervisionada

Otimização e aproximação de funções: Estes dois conceitos estão na base de aprendizagem computacional e podem ser descritos da seguinte forma: (i) *Otimização* é o processo de encontrar o conjunto de parâmetros de uma função que maximiza ou minimiza essa mesma função. (ii) *Aproximação* é o processo de encontrar o conjunto de parâmetros de uma função que a vão moldar segundo um conjunto de pontos, entradas e saídas que a função deve mapear. Essa função é

geralmente determinada por meio de técnicas de ajuste de curvas, regressão ou algoritmos de aprendizado de máquina.

Assim um modelo preditivo pode ser definido como a função h : X $\rightarrow$ Y para que $h_\theta(x) = \Theta * X$. O processo de treino de um modelo, consiste em encontrar o conjunto Θ que melhor aproxima $h_\theta(x)$ a Y. Sendo este o nosso objetivo principal, podemos criar uma função de erro que caracterize esta relação. Existem múltiplas medidas de erro e custo. Por exemplo $MSE(\Theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - h_\theta(x_i))^2$, esta medida carateriza a diferença entre o par (X, Y) do conjunto de dados e o par (X, $h_\theta(X)$) que consiste nas entradas do conjunto de dados e nas previsões do modelo. Assim, podemos usar processos de otimização para minimizar função de erro ou custo e encontrar o conjunto Θ que melhor aproxima o modelo ao conjunto de dados.

Usando uma *cost function* conseguimos quantificar e avaliar a precisão da nossa modelo preditivo. Podemos interpretar isto como o "preço"que o nosso algoritmo deve "pagar"quando prevê o valor - h(x) - quando o rotulo era Y, sendo (X, Y) o *data set*. Esta função deve ser convexa para garantir a melhor desempenho no processo de aproximação e otimização.

$$J(\theta_0, (...), \theta_n) = \frac{1}{m}\sum_{i=1}^{m}\left(cost(h_\theta(X^{(i)}), y^{(i)}\right)$$

Vários exemplos de *cost function* são encontrados na literatura, um exemplo é a MSE - *mean square error* já anteriormente apresentado. Esta função usa a média das diferenças quadráticas das hipóteses obtidas pelo modelo preditivo e os rótulos do conjunto de dados.

$$cost(h(X^{(i)}, y^{(i)}) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$$

$(\theta_0, (...), \theta_n)$ são os n parâmetros usados.

$h_\theta(x_i) - y_i$ é a diferença entre o valor da hipótese e o rotulo verdadeiro.

Para ilustrar como este tipo processo acontece usarei um exemplo simples: 'Gradient descent' para solucionar um problema de regressão linear do tipo $h_\theta(x) = \theta_1 * x + \theta_0$. Aproximaremos esta função a um pequeno conjunto de dados e ilustrar o algoritmo de otimização iterativamente.

A figura 4.3 é um exemplo de como podemos encontrar os parâmetros *Theta* (otimizar a função de previsão) usando derivadas parciais da função de custo.

O algoritmo requer o cálculo do gradiente da função custo. Este algoritmo chamasse método do gradiente (Gradient descent) e serve de base para muitos algoritmos mais avançados usados em AI. No caso do gradiente apontar para cima, o negativo do gradiente de cada variável de entrada é seguido para baixo, resultando em novos valores para cada variável que resultará num custo menor. O tamanho de passo é utilizado para dimensionar o gradiente e controlar o quanto cada variável de entrada deve ser alterada relativamente ao gradiente. E assim, iterativamente, é se encontrado os valores de Θ para os quais a função de custo é menor.

Este exemplo é ilustrativo da necessidade de escolher uma função de custo que seja convexa e diferencial. Convexa para evitar que o algoritmo fique preso num mínimo local e não chegar a encontrar o mínimo global, a melhor solução para o problema. É importante cuidado na escolha de α, o *learning rate*. Este fator, que é multiplicado às derivadas parciais, deve ser pequeno o

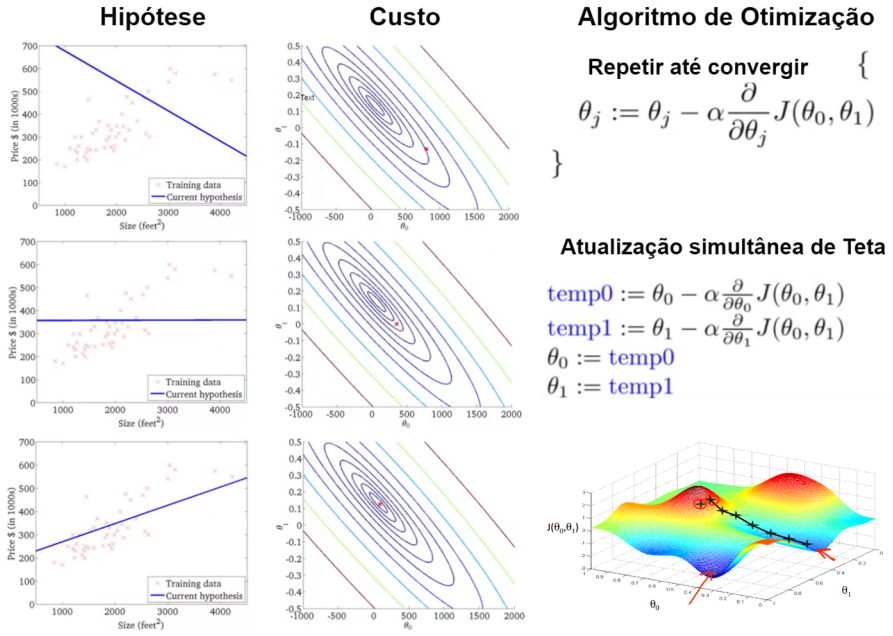


Figura 4.3: Exemplo de aproximação e otimização de uma função linear de primeiro grau.

suficiente para o algoritmo convergir e não pequeno para não ser necessário um número excessivo de iterações.

Este é um exemplo muito simples, onde o modelo preditivo é definido com uma função de primeiro grau, mas é muito elucidativo do funcionamento de algoritmos baseados em derivadas da função de custo, utilizadas em redes neuronais artificiais e aprendizagem profunda.

Assim, algoritmo preditivo usa formas da função de mapeamento diferentes, influenciando o tipo de problema de otimização a ser resolvido. O algoritmo de otimização padrão usado para cada algoritmo de aprendizado de máquina não é arbitrário; ele representa o algoritmo mais eficiente para resolver o problema de otimização específico proposto pelo algoritmo, por exemplo, método do gradiente estocástico (Stochastic gradient descent) para redes neuronais.

SGD é uma extensão do algoritmo de descida do gradiente para minimizar uma função de custo aproximando o modelo a um conjunto de dados de treino. É um exemplo de um algoritmo de busca global iterativo. É geralmente usado quando otimizamos um modelo de rede neuronal dado que o espaço de busca é multimodal e altamente não linear. As entradas para a função de mapeamento são os pesos da rede. [22]

Outro exemplo usado em DL é Adam [23], um método para otimização estocástica eficiente que requer apenas gradientes de primeira ordem com baixa exigência de memória. O método é computacionalmente eficiente, requer pouca memória, adequado para problemas que envolvem abundância de dados. [24]

Tanto ADAM como SGD serão aprofundados mais adiante quando forem introduzidas as redes neuronais profundas. Estes algoritmos necessitam de alguns melhoramentos quando são usados em redes profundas, pois o valor de saída das camadas intermédias é desconhecido.

No processo de otimização, não só os parâmetros θ são ajustados, um conjunto de hiperparâmetro podem ser ajustados e moldados, incluindo inicialização aleatória de θ com diferentes distribuições, taxa de aprendizagem, funções de ativação, etc, visando obter a melhor desempenho do modelo.

A aprendizagem de máquina consiste no desenvolvimento de modelos preditivos. Para determinar se um modelo é melhor do que outro. Temos algumas métricas de avaliação para medir o desempenho de um modelo num determinado conjunto de dados. Nesse sentido, se considerarmos os parâmetros que criaram o modelo como a entrada, o algoritmo interno do modelo e o conjunto de dados em questão como constantes, e a métrica avaliada a partir do modelo como a saída, então temos uma função construída. O que queremos fazer a seguir é manipular a entrada e verificar a saída até encontrarmos a melhor saída possível. [25]

Assim, escolha do algoritmo de otimização para o modelo de aprendizagem profundo pode fazer a diferença entre obter bons resultados em minutos, horas ou dias. É crucial escolher cuidadosamente o algoritmo de otimização adequado para o modelo de aprendizagem profunda, considerando o equilíbrio entre tempo, recursos e desempenho desejado.

Em suma, aprendizagem profunda é um tipo particular de aprendizagem computacional com muito potencial e flexibilidade na representação do mundo. Modela abstrações de dados usando um grafo profundo com várias camadas de processamento, compostas de várias transformações lineares e não lineares. Assim resolve gradualmente conceitos complexos a partir de conceitos mais simples, representações mais abstratas computadas a partir de representações menos abstratas, camada após camada, fazendo a correspondência entre os dados de entrada e o sinal de saída. Durante o processo de treino, os dados de um conjunto de dados são passados várias vezes através da rede neuronal e os parâmetros da rede são ajustados consoante o sinal de saída desejado. Assim que os critérios de fidelidade são cumpridos, consideramos que o modelo aprendeu a solução do problema.

As principais desvantagens de DL são a necessidade de conjunto de dados (data - sets) grandes e bem balanceados e a dificuldade em avaliar o correto funcionamento em camadas intermédias da rede, sendo por isso chamados modelos "caixa preta".

Na próxima secção explorarei os conceitos de redes neuronais, a sua estrutura e terminologia. O que faz uma rede neuronal "profunda". O que acontece no processo de treino, métodos de otimização e critérios de fidelidade.

4.2 ANN - Rede artificial Neuronal

ANN (artificial neural network) São modelos de processamento e geração de dados criados para replicar o sistema neuronal para revelar padrões não lineares presentes em conjuntos de dados, e uma forma de simular o conhecimento empírico humano, daí o seu nome.

Descobrimentos em neurociência, como em [26], sugerem que cérebros de mamíferos usam um único algoritmo para resolver a maioria das funções cerebrais. Numa experiência onde se redirecionou os sinais visuais de furões, para zonas dos seus cérebros encarregue posteriormente

por funções de audição, verificou-se que estas zonas conseguiram aprender a ver. Assim, também um algoritmo de aprendizagem profunda poderá resolver diferentes problemas. Com esta hipótese, várias comunidades de aprendizagem computacional convergiram na metodologia aprendizagem profunda.

Hoje, neurociência é considerada uma importante fonte de inspiração para desenvolvimentos em aprendizagem profunda, mas já não é um guia predominante na área. Na verdade, sabemos que os neurónios computam funções diferentes das usadas em redes neuronais de animais, mas a ideia base de ter várias unidades computacionais que formam inteligência a partir da sua interação mútua é inspirada pelo cérebro.

O neurónio, a unidade computacional mais básica, opera da seguinte forma: recebe sinal de entrada de outros nós ou fontes externas (sensores, data-sets) e aplica uma função de ativação à soma ponderada de cada entrada. Algumas funções comuns são: Sigmoid, tanh e ReLU.

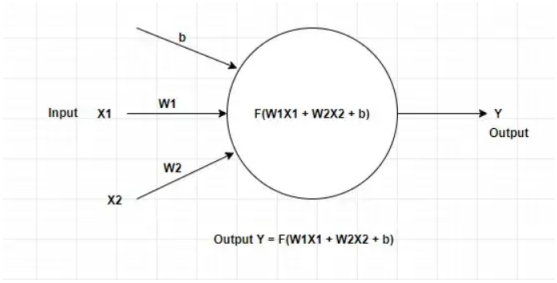


Figura 4.4: Modelo de uma unidade computacional - neurónio

Na figura 4.4, X representa o sinal de entrada, W os pesos que cada entrada terá no somatório e b é o termo de bias e $F()$ a função de ativação, sendo o seu resultado o sinal de saída Y. Assim, o principal objetivo é associar um conjunto de valores de entrada $x^{(1)},...,x^{(n)}$ a um valor de saída Y. Para tal os modelos vão aprender um conjunto de pesos $w^{(1)},...,w^{(n)}$ e computar o valor de saída tal que $Y(X,W) = \Phi(x^{(1)} * w^{(1)} + ... + x^{(n)} * w^{(n)})$.

Para que a saída do modelo corresponda à saída pretendida, o conjunto de pesos devem ser definidos corretamente. Este processo ocorre no momento de treino da rede neuronal.

Função de ativação permite que pequenas mudanças nos pesos e bias causem apenas uma pequena alteração no 'output'. Esse é o fato crucial que permitirá que uma rede de neurónios artificiais aprenda.

Assim é um importante componente de uma rede neural artificial que introduz não linearidade na saída de um neurónio ou camada. Elas ajudam a modelar relações complexas e permitem que as redes neuronais aprendam e aproximem uma ampla variedade de funções. [27]

As principais características gerais das funções de ativação em redes neuronais são as seguintes:

- *Não linearidade:* As funções de ativação introduzem não linearidades na rede, permitindo que ela aprenda e represente padrões e relações complexas nos dados. Sem funções de

ativação, uma rede neural ficaria limitada a representar apenas relações lineares entre a entrada e a saída. Assim permite que as redes neuronais aprendam mapeamentos complexos entre a entrada e a saída. Ao aplicar transformações não lineares aos dados de entrada, as funções de ativação permitem que a rede aprenda e aproxime funções altamente não lineares.

- *Introdução de Saídas Não Nulas:* As funções de ativação garantem que os neurónios produzam saídas não nulas, mesmo que a entrada seja zero. Isso ajuda a quebrar a simetria entre os neurónios e permite que cada neurónio aprenda diferentes características e padrões nos dados.

- *Propagação de Gradientes:* As funções de ativação afetam o fluxo de gradientes durante a retropropagação, essencial para o treino da rede neural. Diferentes funções de ativação possuem propriedades de gradiente distintas e impactam a estabilidade e velocidade do aprendizado.

Uma função de ativação executa o seguinte algoritmo:

$$\Phi = Activation(\sum_{i=1}^{m}((\Theta * x_i) + bias))$$

As funções de ativação permitem a propagação posterior desde que os gradientes sejam fornecidos com o erro para atualizar os pesos e bias. Sem a função não linear diferenciáveis, isso não seria possível. Alguns exemplos de funções de ativação usadas em redes neuronais incluem e as suas funções especificas:

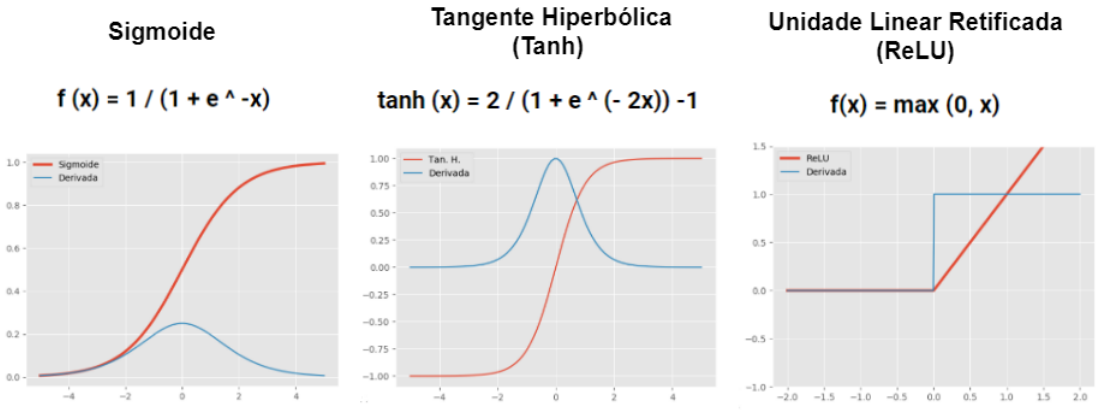


Figura 4.5: Funções de ativação, expressões matemáticas e respetivos gráficos com derivadas

- *Função Sigmoide:* A função sigmoide mapeia a entrada para um valor entre 0 e 1, fornecendo uma ativação suave e limitada. É uma função não linear. A função essencialmente tenta empurrar os valores de Y para os extremos. Nas extremidades da função sigmoide, os valores de Y tendem a responder muito menos às mudanças em X. O que significa que

o gradiente nessa região será pequeno. Isso dá origem a um problema de *Vanishing Gradient*. Se as ativações alcançarem a parte "quase horizontal"da curva em ambos os lados, o gradiente torna-se pequeno ou desaparece (não pode causar mudanças significativas devido ao valor extremamente pequeno). A rede não consegue aprender mais ou fá-lo de forma extremamente lenta.

- *Função Tangente Hiperbólica (Tanh):* A função tangente hiperbólica (tanh) mapeia a entrada para um valor entre -1 e 1, fornecendo uma versão deslocada e dimensionada da função sigmoide. Basicamente, soluciona o nosso problema dos valores nulos, e do mesmo sinal. Todas as outras propriedades são as mesmas da função sigmoide. É contínuo e diferenciáveis em todos os pontos. Funções Sigmoide e Tanh às vezes são evitadas devido ao problema de *Vanishing Gradient*.

- *Unidade Linear Retificada (ReLU):* A ReLU define os valores de entrada negativos como zero e mantém os valores positivos inalterados. Tornou-se popular devido à sua simplicidade e eficácia no treino de redes neuronais profundas. A principal vantagem de usar a função ReLU sobre outras funções de ativação é que ela não ativa todos os neurónios ao mesmo tempo, tornando a rede esparsa e eficiente e fácil para a computação, a sua principal desvantagem é que pode ter problemas com os gradientes que se deslocam em direção a zero. ReLU deve ser usada apenas nas camadas ocultas. A função ReLU não é limitada. A faixa de valores da função ReLU é [0, infinito). Isso significa que ela pode amplificar a ativação, levando a valores muito altos.

A escolha da função de ativação depende do domínio do problema, da arquitetura da rede e dos requisitos específicos da tarefa em questão. A escolha da função de ativação está intimamente ligada à questão de otimização e aproximação de um modelo ao conjunto de dados. Seguem alguns exemplos.

Para redes neuronais profundas (reconhecimento de imagem usando deep residual learning da Microsoft com 150 camadas) [28], se fosse usada uma função de ativação linear, a saída em cada camada cresceria exponencialmente no caso de $\theta > 0$ e decresceria exponencialmente no caso de $\theta < 0$, assim como os seus gradientes, tornando o processo de treino impossível, ou extremamente lento. A este problema chamasse *Exploding Gradient* e *Vanishing Gradient* respetivamente. Outra forma de combater este problema seria inicializar os parâmetros $\theta_1(...)\theta_n$ com variância $\frac{a}{n}$, sendo n o número de parâmetros e a uma constante que poderia ser ajustada como um hiperparâmetro no processo de otimização.

A seguinte secção descreve a tecnologia central de aproximação de funções paramétricas que está por trás de quase todas as aplicações práticas modernas de aprendizagem profunda. Começo por descrever o modelo de alimentação direta usado para representar essas funções. Em seguida, apresento técnicas avançadas de regularização e otimização de tais modelos. Escalar esses modelos para sequências temporais longas, requer especialização, sendo para tal, introduzida a rede neural recorrente para processar sequências temporais.

4.3 FNN - Rede neural direta

O algoritmo quinta-essência de aprendizagem profunda é a rede neural de alimentação direta (FFN - "feedforward deepnetwork") também conhecido por perceptron multicamadas (MLP - "multilayer perceptron"). Um perceptron multicamadas é essencialmente uma função que mapeia valores de entrada a valores de saída. A função é formada por um conjunto de várias funções mais simples. Podemos pensar na aplicação de cada função simples como uma transformação que nos dá uma nova representação dos dados de entrada. Podemos olhar para aprendizagem profunda como o processo de descobrir a representação dos dados que nos permite conferir a solução de um problema. Outra é que a profundidade do grafo permite ao computador aprender um programa de múltiplos passos que se sucedem uns aos outros. Cada camada da representação pode ser pensada como o estado da memória do computador após executar um conjunto de instruções em paralelo. Nem todas as saídas de todos perceptrons / neurónios codifica fatores de variância que explicam os dados de entrada, alguns apenas auxiliam o modelo a organizar o seu processamento, funcionando analogamente a um contador ou apontador.

Estas redes podem ser vistas como grafos computacionais, combinando vários elementos básicos (neurónios artificiais). Cada um destes neurónios, também conhecidos por nós são organizados por camadas. Existem múltiplas configurações possíveis de nós e camadas, nesta secção explicarei a teoria matemática por de trás das configurações utilizadas neste trabalho.

Redes neuronais diretas são a mais simples configuração de redes neuronais artificiais, a informação é propagada da camada de entrada até à camada de saída, passando pelas camadas intermediárias/escondidas num processo denominado *forward pass*. Existem três tipos diferentes de camadas numa rede neuronal:

Camada de entrada: A camada de entrada é a primeira camada de uma rede neuronal. Esta é classificada como camada visível, pois contem variáveis observáveis. Esta camada recebe os dados de entrada e passa-os para a próximas camadas.

Camadas intermédias: Também chamadas camadas escondidas, são uma série de camadas, entre as camadas de entrada e saída, que aplicam a função de ativação e passam o seu resultado às camadas seguintes entre as camadas de entrada e saída. A sua função é processar sinais de entrada das camadas anteriores, extraindo assim, características cada vez mais abstratas. Os seus valores não são observáveis no conjunto de dados, cabe ao modelo decidir o conceito que melhor explica a relação entre os dados observados.

Camada de saída: A camada de saída é responsável por computação dos valores de saída, usando uma função de ativação e de gerar e transferir informação da rede para o mundo real.

Feed forward Usando a figura 4.6 como referência, todo o processo iniciasse na camada de entrada. Assim, z_1 e z_2 são obtidos por combinação linear da entrada x com w_1 e b_1, e w_2 e b_2,

respetivamente. A seguir, a_1 e a_2 são as saídas resultantes da aplicação da função de ativação, que neste exemplo é ReLU, resultando em z_1 e z_2, respetivamente, estas são a saídas dos neurónios da primeira camada oculta e vão servir como entrada para a camada adjacente. Este processo continua até à camada de saída. Na prática, estas operações acontecem com cálculo matricial, da seguinte forma:

$$\begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} w_1 & b_1 \\ w_2 & b_2 \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} z_3 \\ z_4 \end{bmatrix} = \begin{bmatrix} w_3 & w_5 & b_3 \\ w_4 & w_6 & b_4 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} z_5 \end{bmatrix} = \begin{bmatrix} w_7 & w_8 & b_5 \end{bmatrix} \begin{bmatrix} a_3 \\ a_4 \\ 1 \end{bmatrix}$$

O último passo neste processo é o cálculo do erro de treino que, no fundo, é o que caracteriza a diferença entre o valor de saída esperado para uma dada entrada (par do conjunto de dados, no caso de aprendizagem supervisionado), e a saída da rede neuronal.

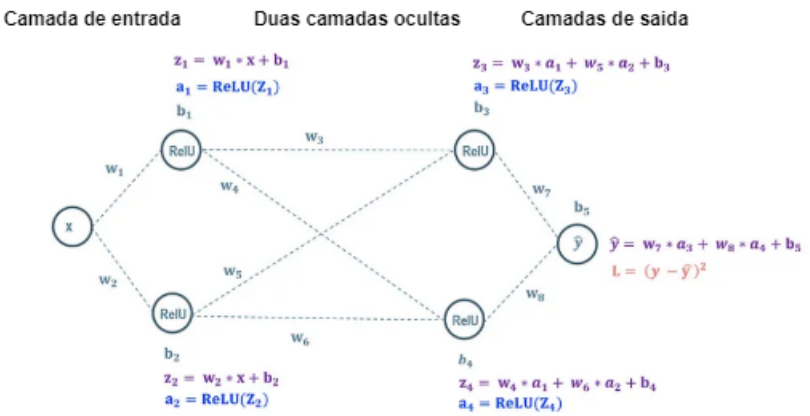


Figura 4.6: ANN e cálculos de forward pass [2]

Backpropagation Assim como feed fowrd é como se calcula a saída da rede neuronal com respeito às entradas, existe uma forma eficiente de calcular as derivadas da função de custo a cada entrada e assim poder ajustar os pesos de cada ligação, a isto se chama backpropagation (retropropagação).

O algoritmo foi desenvolvido explicitamente para calcular os gradientes das variáveis em estruturas de grafo. Permite que a informação do custo flua para trás através da rede, de modo a calcular o gradiente. Calculando as derivadas de funções formadas pela composição de outras

funções cujas derivadas são conhecidas. O Backpropagation é um algoritmo que computa a regra da cadeia, com uma ordem específica de operações que é altamente eficiente. [20].

A regra da cadeia permite o cálculo de derivadas de funções compostas, isto é a entrada de uma dada função $h()$ é a saída de uma função $f()$ cuja entrada é a saída de outra função $g()$, isto é $Y = h(f(g(x)))$ ou então $Y = (h \circ (f \circ g))(x)$. Computar as saídas da função derivada de Y segundo x pode ser calculada da seguinte forma:

$$Y' = (h \circ (f \circ g))'(x) \equiv \frac{\partial Y}{\partial x} = \left(\frac{\partial h}{\partial f} * \frac{\partial f}{\partial g} * \frac{\partial g}{\partial x} \right) \tag{4.1}$$

Usando como exemplo a rede neuronal na figura 4.6 devemos começar por calcular a derivada da função de custo em função a Y, $(\frac{dL}{d\hat{Y}}) = -2 * (y - \hat{y})$ e partindo dai podemos calcular o gradiente relativamente a cada peso w_n aplicando a regra da cadeia, por exemplo, relativamente ao peso que liga o primeiro e o terceiro neurónio, w_3:

$$\begin{aligned} \frac{dL}{dw_3} &= \frac{dL}{da_3} * \frac{da_3}{dz_3} * \frac{dz_3}{dw_3} \\ &= \frac{dL}{d\hat{Y}} * \frac{d\hat{Y}}{da_3} * \frac{da_3}{dz_3} * \frac{dz_3}{dw_3} \\ &= L_y * w_7 * dRL(z_3) * a_1, \\ onde : dLR(z_3) &= \frac{dReIU(z_3)}{dz_3} \end{aligned} \tag{4.2}$$

Assim os passos para otimização dos parâmetros e treino de um modelo de DL são os seguintes:

- *Feed forward:* Os dados de entrada são propagados através da rede e as ativações são calculadas em cada camada até chegarmos à camada de saída. Durante essa passagem, as ativações são armazenadas para uso posterior no cálculo dos gradientes.

- *Cost function:* Após a passagem de feed forward, a função de custo é calculada. Esta função mede o quão distante as previsões do modelo estão dos valores reais nos dados de treino. O objetivo do treino é minimizar essa função, tornando as previsões do modelo mais precisas.

- *Back propagation:* A etapa de backpropagation começa calculando os gradientes dos parâmetros Θ relativamente à função de perda. Esse cálculo ocorre camada por camada, a partir da camada de saída até a camada de entrada. O gradiente é calculado usando a regra da cadeia. Para cada unidade em cada camada, calculamos o gradiente local, sendo o gradiente da função de ativação da unidade em relação à sua entrada ponderada. Esse gradiente indica como uma pequena mudança na entrada da unidade afetará a saída da unidade. Usando os gradientes locais, podemos calcular os gradientes dos parâmetros relativamente à função de perda. Isso é feito multiplicando o gradiente local de cada unidade pelo valor da ativação da unidade anterior (na passagem de feed forward). Essa multiplicação indica como uma pequena mudança nos parâmetros afetará o erro total do modelo.

- *Atualização dos Parâmetros:* Após calcular os gradientes dos parâmetros, o algoritmo de otimização é usado para ajustar os parâmetros do modelo na direção que reduz o erro. Os parâmetros são atualizados proporcionalmente ao gradiente e a uma taxa de aprendizado (learning rate), controlando o tamanho dos passos de atualização.

Este processo repetem-se múltiplas vezes, cada ciclo é um epoch até que o modelo alcance uma convergência desejada e o erro seja minimizado.

Este tipo de rede funciona quando as amostras de dados são independentes umas das outras. Quando existe informação intrínseca à evolução temporal, o perceptron multicamadas demonstra-se primitivo. No caso de resolver um problema que envolva uma sequência temporal, por exemplo, um RUL com dez momentos de amostra, seria necessária uma camada de entrada com dez neurónios, para cem amostras, cem neurónios de entrada, para se poder captar estas relações temporais. Isto é uma limitação evidente das redes neuronais clássicas devia à sua 'interface' é muito restrita. Estas aceitam um vetor de tamanho fixo como entrada (por exemplo, uma imagem) e produzem um vetor de tamanho fixo como saída (por exemplo, probabilidades de diferentes classes). Além disso, esses modelos realizam esse mapeamento usando uma quantidade fixa de etapas computacionais (por exemplo, o número de camadas no modelo). As redes recorrentes abordam o problema de uma forma mais adaptada a séries temporais, tendo 'interface' na entrada e na saída uma sequência de vetores.

Redes neuronais recorrentes são adaptações de redes neuronais diretas que usam realimentação. Nas redes com realimentação ou recorrentes (recorrente), a saída de alguns neurónios alimentam neurónios da mesma camada (inclusive o próprio) ou de camadas anteriores. São utilizadas para dados numa sequência temporal.

Numa série temporal, as amostras de dados são apresentados numa sequência. A sequência apresenta implicitamente o momento no tempo em que os dados são amostrados. Enquanto uma pessoa lê um texto, o seu entendimento depende das palavras anteriores e do contexto. Redes neuronais tradicionais não têm o poder de replicar esta capacidade de reter contexto.

Assim surgem (RNN - "recurrent neural network"). Esta família de redes neuronais apresentam ciclos de realimentação, o que lhes permite reter informação. No seguinte diagrama podemos ver a estrutura base destas redes. [29]

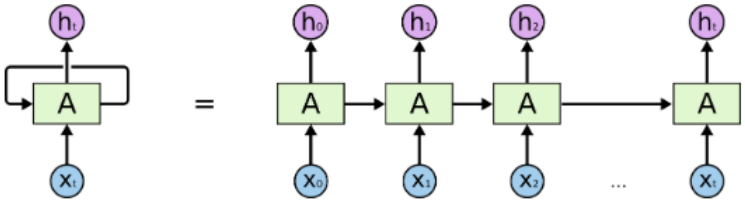


Figura 4.7: RNN

Podemos pensar nestas redes como várias FNN que passam a informação de umas para as outras, cada uma resolvendo uma entrada da sequência temporal. Uma das grandes dificuldades

das implementações de RNNs é quando o intervalo de relevância dos dados para previsão aumenta. *Dependências de longo prazo*, é um problema que ocorre quando é necessário guardar a informação de contexto durante várias iterações temporais da rede.

A questão da dependência a longo prazo pode originar dois problemas: (i) Exploding gradients e (ii) Vanishing gradients. Estes problemas surgem durante o treino de RNNs quando os gradientes se propagam numa série temporal longo, até a camada inicial. Os gradientes provenientes das camadas mais profundas passam por multiplicações contínuas de matrizes segundo a regra da cadeia. À medida que se aproximam das camadas anteriores, se possuírem valores pequenos (<1), eles diminuem exponencialmente até desaparecerem, impossibilitando que o modelo aprenda (ii), vanishing gradient. Por outro lado, se possuírem valores grandes (>1), os gradientes sobem exponencialmente (i) exploding gradients.

Quando os gradientes explodem, eles podem se tornar NaN (Not a Number) devido ao 'overflow' numérico, ou podemos observar oscilações irregulares no custo de treino ao traçar a curva de aprendizado. Uma solução para corrigir esse problema é aplicar o "gradient clipping", que impõe um limite pré-definido nos gradientes para evitar que se tornem muito grandes. Dessa forma, a direção dos gradientes não é alterada, apenas o seu comprimento.

Uma solução simples para lidar com o problema dos gradientes que desaparecem é a arquitetura de RNN identitária; onde os pesos da rede são inicializados com a matriz identidade e as funções de ativação são todas definidas como ReLU. Isso funciona bem porque, quando as derivadas do erro são propagadas para trás no tempo, elas permanecem constantes, sendo 0 ou 1.

Uma solução ainda mais popular e amplamente utilizada é a arquitetura *LSTMs (Long Short Term Memory networks)*, são um tipo especial de RNN ar que foi projetada para facilitar a captura de dependências de longo prazo em dados sequenciais, apresentando características que serão expostas na próxima secção.

A seguir apresentamos técnicas avançadas de otimização e aproximação para modelos de redes neuronais profundas, em específico aquelas usadas na implementação desta dissertação. Usar este tipo de modelos para entradas grandes como sequências temporais requer especialização e cuidados especiais.

4.3.1 LSTM

Uma LSTM (Long Short Term Memory) faz parte da família das RNNs, no sentido em que estas são realimentadas, permitindo guardar valores de saída mais antigos da série temporal, dando assim contexto aos novos valores de entrada que chegam à rede neuronal.

Onde as LSTM divergem das RNNs é que não sofrem do problema de *long-term dependency*. Elas são desenhadas com o propósito de reter informação durante várias iterações.

As RNN apresentam um encadeamento de múltiplos módulos de redes neuronais, tantas quanto o tamanho da série temporal. Em RNNs mais simples, a estrutura de cada módulo é muito simples, podendo se tratar simplesmente de uma função de ativação, como *tanh*.

LSTMs apresentam também uma estrutura em cadeia, mas cada módulo é constituído por uma estrutura diferente, havendo quatro redes neuronais que interagem entre si de uma forma muito especial.

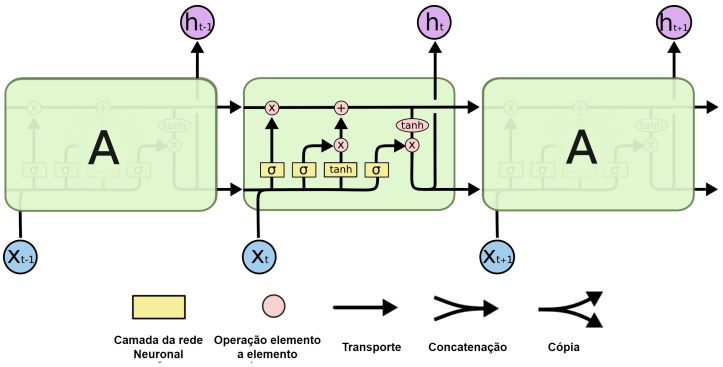


Figura 4.8: Os quatro módulos de uma LSTM

No diagrama, cada linha transporta um vetor da saída de um nó para a entrada de outros. Os círculos cor-de-rosa representam operações entre vetores de elemento a elemento, como adições. Por fim, os quadrados amarelos são redes neuronais treinadas. Linhas que se juntam denominam concatenações entre vetores enquanto as que divergem, denominam cópia de vetores.

A seguir isolo cada bloco, fazendo uma descrição do seu funcionamento para assim dar a entender o funcionamento geral deste tipo de redes neuronais.

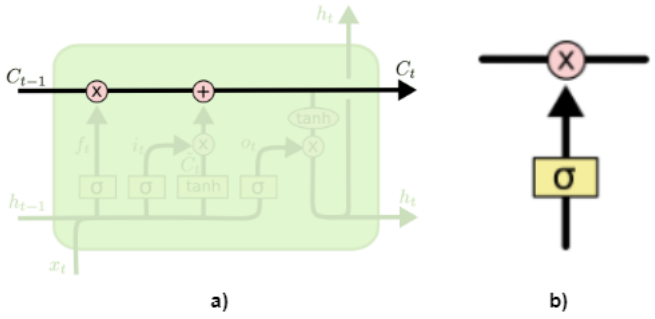


Figura 4.9: a) Estado da célula. b) Operação controlada por porta.

A chave para o funcionamento de LSTMs é o estado da célula, a linha horizontal que atravessa o topo do diagrama (a)). O estado da célula percorre toda a cadeia, e transporta a informação no momento "t". Esta pode sofrer alterações lineares (b)). Estas operações são controladas por funções que funcionam como portas.

As portas são uma forma de selecionar informação a ser adicionada ou retirada ao estado da célula, são compostas por uma rede neuronal com função de ativação *sigmoid*, com uma multiplicação entre vetores, ponto a ponto.

A saída da *sigmoid* é entre zero e um. O valor de zero, não deixará passar nada, enquanto o valor de um deixará passar tudo. LSTMs usando três destas portas, protegem e controlam o estado da célula.

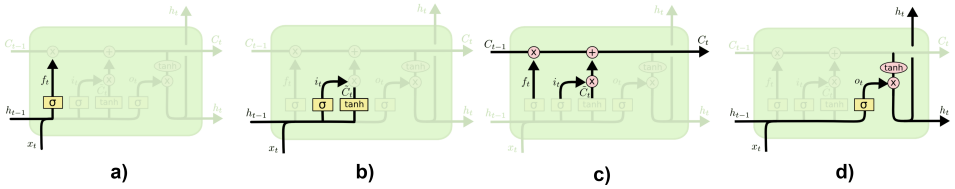


Figura 4.10: Os quatro módulos de uma LSTM

***Forget gate layer* a)** O primeiro passo numa célula LSTM é decidir que informação deve ser esquecida. Para tal, esta rede neuronal recebe como entrada $[h_{t-1}, x_t]$ (saída da LSTM no instante anterior e a entrada da série temporal atual, respetivamente). A sua função de ativação é a *sigmoid*, assim esta rede gera uma saída compreendida entre [0, 1], para cada valor no vetor estado da célula do instante anterior (C_{t-1}).

$$f_t = \sigma.(W_f.[h_{t-1}, x_t] + b_f) \tag{4.3}$$

***Input gate layer* b)** A seguir escolhemos que nova informação devemos guardar no estado da célula. Este passo tem duas etapas: Em 4.4, uma rede com função de ativação *sigmoid* serve de porta (i_t), que irá selecionar que informação será atualizada no estado da célula; em 4.4, uma rede com função de ativação *tanh* cria um vetor para novos candidatos a estado da célula ($\tilde{C}_t$), que serão adicionados ao estado de célula anterior, condicionados pelo 'input gate'. Estas redes usam como entrada a concatenação dos novos valores da série temporal e a saída da LSTM no instante anterior ($[h_{t-1}, x_t]$).

$$i_t = \sigma.(W_i.[h_{t-1}, x_t] + b_i) \tag{4.4}$$

$$\tilde{C}_t = tanh.(W_C.[h_{t-1}, x_t] + b_C) \tag{4.5}$$

Atualizar estado de célula c) Neste ponto todas as decisões foram já feitas, resta executá-las. Assim, em 4.6: *Forget gate* (f_t) é multiplicado ao estado de célula do instante anterior (C_{t-1}), eliminando ou mantendo informação antiga; E *input gate* (i_t) é multiplicado ao novo candidato a estado de célula $\tilde{C}_t$, selecionando que nova informação deve ser guardada. Os dois são adicionados, formando o novo estado de célula (C_t).

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{4.6}$$

***Output gate* d)** Por fim resta escolher que informação será a saída da LSTM. Esta decisão é baseada no estado da célula em t (C_t). Em 4.7 temos uma porta *sigmoid* (o_t) que escolhe, em 4.8 a percentagem de ($tanh(C_t)$), estado da célula normalizado entre [-1 , 1], fazendo a multiplicação entre os dois.

$$o_t = \sigma.(W_o.[h_{t-1}, x_t] + b_o) \tag{4.7}$$

$$h_t = o_t * tanh(C_t) \tag{4.8}$$

4.4 Otimização em RNNs

Neste capitulo esmiuçarei o processo de treino de redes neuronais profundas, os métodos mais usados e os desafios encontrados no treino de LSTMs, alguns dos quais mencionados anteriormente.

O processo de obter o melhor modelo de previsão é um processo iterativo que passa pelo ciclo de gerar uma ideia, codificar e testar. Vários hiperparâmetros podem ser testados como, número de camadas, unidades por camada, taxa de aprendizagem, funções de ativação, etc. Para auxiliar este processo, o conjunto de dados é dividido em três conjuntos diferentes: treino, validação e teste. Porque se trata de séries temporais, é importante que a divisão seja feita mantendo a integridade de cada série temporal. Sendo o conjunto de dados constituído por N séries temporais de dados de sensores do início ao fim de vida de rolamentos, é considerado boa prática que por exemplo 60% seja guardado para treino, 20% para validação e 20% para teste. O conjunto de treino servirá para treinar o modelo escolhido, o conjunto de validação servirá para o algoritmo de otimização encontrar os hiperparâmetros ideais para essa configuração e por fim o conjunto de teste para fazer a avaliação imparcial do modelo. Estas percentagens podem diferir para conjuntos de dados muito grandes, por exemplo, 98% seja guardado para treino, 1% para validação e 1% para teste para conjuntos de dados > 1 milhão de exemplos. É importante mencionar que o conjunto de teste e validação devem ter a mesma distribuição probabilística [4].

Assim, durante o treino de um modelo podemos contar com três tipos de indicadores de erro: erro de treino, erro de validação, erro de teste. Como já mencionado, guardando uma porção do conjunto de dados para o teste final do modelo obtido, o erro de teste permite avaliar a desempenho em condições reais do modelo. Os outros dois são indicadores para a otimização e perceber o que acontece com o modelo.

O conjunto de teste é usado para ajustar os parâmetros da rede neural durante o processo de aprendizagem. A rede é exposta a esses dados para que ela possa atualizar os seus pesos e ajustar-se aos padrões presentes nos dados. O conjunto de validação é usado para ajustar os hiperparâmetros do modelo e fazer escolhas relacionadas à arquitetura da rede, como o número de camadas, tamanho das camadas, learning rate, entre outros.

Isto é, se obtivermos um baixo erro de teste, e um alto erro de validação, diz-se que o modelo sobre ajustou (overfitting) o conjunto de dados, ou que tem alta variância, isto é um indicador de que o modelo não vai generalizar bem para dados nunca vistos, neste caso estratégias como regularização, obter mais dados ou outra arquitetura pode resultar. Se, por outro lado obtivermos um alto erro de teste e um erro de validação comparável ao de teste, diz-se que o modele sob ajustou (underfitting) o conjunto de dados, ou que tem alto bias, isto significa que a rede não está aprendendo as nuances do conjunto de dados, neste caso, aumentar a complexidade e tempo de treino poderá funcionar ou experimentar outra arquitetura de rede neuronal.

Com treino de uma rede entende-se encontrar o Θ que reduza significativamente a função de custo $J(\Theta)$, ao mesmo tempo, avaliando a desempenho de todo o conjunto de dados de treino assim como termos de regularização. O processo de treino de um neurónio já foi exposto. A seguir explicarei como o valor de saída é propagado pela rede neuronal e como esse valor é usado para calcular a função de custo com o fim de atualizar os parâmetros θ de cada neurónio da rede.

Backpropagation through time: Em redes neuronais recorrentes é usado retropropagação no tempo [30]. Esse procedimento requer que expandamos (ou desenrolemos) o grafo computacional de uma RNN passo a passo. A RNN desenrolada é essencialmente uma rede neural feed forward com a propriedade especial de que os parâmetros são partilhados pela rede, aparecendo em cada passo de tempo. Em seguida, assim como em qualquer rede neural feed 'forward', podemos aplicar a regra da cadeia, retropropagando os gradientes através da rede desenrolada. E aí é usado um método de otimização que utilize da função de custo.

Gradient descent Na secção anterior foi introduzido o Método do gradiente e as suas variantes SGD e ADAM. Quando otimizamos uma função segundo um conjunto de dados grande na ordem de milhares de amostras, calcular o custo computando todo o conjunto de dados para poder fazer alterações em Θ é altamente ineficiente. Assim surgem algoritmos que funcionam com janelas (Batches). As janelas resultam da divisão do conjunto de dados em vários "batches"de um certo tamanho (geralmente potências de 2) e computam a função de custo de um batch sequencialmente, atualizando Θ no fim de cada batch.

A evolução da função de custo pode não ser linear, pois o algoritmo só tem em conta uma quantidade limitada do conjunto de dados que pode estar melhor ou pior aproximada do que o conjunto seguinte. Assim, a escolha do tamanho desta janela terá um grande impacto na forma como a função de custo se vai comportar: se o tamanho de batch for muito grande, o custo de computação será muito elevado. Se for muito baixo, a função de custo irá oscilar muito, podendo não convergir, além disso, perde-se a vantagem da computação matricial.

Na figura 4.11 podemos ver três implementações de gradient descent usando três tamanhos de janela diferentes. Primeiro Batch gradient descente que usa o conjunto de dados inteiro (whole batch), stochastic gradient descent onde se usa um exemplo do conjunto de dados de cada vez, e por fim mini-batch onde o tamanho da janela é um intermédio dos dois últimos e que pode ser

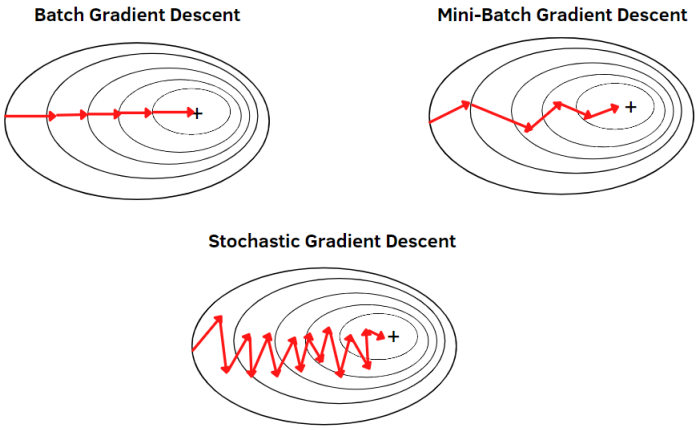


Figura 4.11: Gradient descent usando janelas diferentes [3]

testado e tratado como um hiperparametro. É importante que o tamanho do mini-batch caiba na memória do CPU ou GPU que esteja a ser usado.

ADAM Como mencionado nos últimos parágrafos e ilustrado na figura 4.11 diminuir o tamanho de cada batch tem um impacto significativo na velocidade de processamento, mas com um custo significativo na convergência e estabilidade da função de custo. ADAM é um algoritmo de otimização baseado em gradiente que resolve este problema usando três conceitos, *momentum*, *RMSprop* e *Bias Correction of Exponentially Weighted Averages*

Média móvel (Moving average) é um método de suavizar um conjunto de amostras, ou seja, um estimador calculado a partir de amostras sequenciais da população.

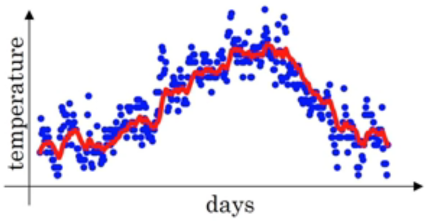


Figura 4.12: Exemplo de aplicação de média móvel [4]

Assim o estimador é calculado com a seguinte fórmula:

$$V_t = \beta V_{t-1} + (1-\beta) * J_t \tag{4.9}$$

onde, V_t é a média das últimas $\frac{1}{1-\beta}$ amostras, que neste caso seriam o resultado da função de custo do batches. Quanto maior for, β menos ruído da aproximação obtemos. Esta implementação é elegante e eficiente em termos de memória, já que não é necessário guardar todos os valores dos quais queremos calcular a média.

O algoritmo corre da seguinte forma:

$V_j = 0$

Repetir

receber amostra J_t

$V_j := \beta * V_j + (1-\beta) * J_t$

Como inicializamos, $V_j = 0$ as primeiras amostras podem induzir um erro grande, por isso usamos *Bias Correction of Exponentially Weighted Averages*. Assim, para uma dada estimativa V_j, calculamos $\frac{V_j}{(1-\beta^t)}$ o termo do denominador vai-se aproximando a 1 com o passar das iterações.

Quando usamos este algoritmo para calcular o mini-batch gradiente descent chamamos gradient descent com momentum. Desta forma os parâmetros θ e b são atualizados com a estimativa da média móvel das derivadas até à iteração t e não com as derivadas obtidas na iteração t, dando assim passos menos oscilatórios.

Momentum gradient descent: com α e β hiperparâmetro,

Na iteração t:

computar derivadas $d\theta$ e no mini-batch atual

$$V_{d\theta} = \beta * V_{d\theta} + (1-\beta) * d\theta$$

$$\Theta := \Theta - \alpha * V_{d\theta}$$

RMSprop funciona de uma forma semelhante, mas desta vez, manteremos a média do quadrado das derivadas, ou seja:

$$S_{d\theta} = \beta * S_{d\theta} + (1-\beta) * d\theta^2$$

$$\Theta := \Theta - \alpha * \frac{d\Theta}{\sqrt{S_{d\theta}}}$$

Na técnica de otimização RMSprop, atualizamos os nossos parâmetros de tal forma que o movimento na direção, com maior oscilação, é penalizado. A ideia é que nas dimensões onde estas oscilações ocorrem, é obtida uma soma maior que nas outras dimensões. Assim, no passo de atualização, esta dimensão será dividida por um valor maior, contribuindo para uma maior suavização, permitindo a escolha de uma taxa de aprendizagem maior.

Por fim, ADAM une estes três conceitos num só algoritmo de otimização:

$$V_{d\theta}^{correct} = (\beta_1 * V_{d\theta} + (1-\beta) * d\theta) * \frac{1}{1-\beta_1^t}$$

$$S_{d\theta}^{correct} = (\beta_2 * S_{d\theta} + (1-\beta) * d\theta^2) * \frac{1}{1-\beta_2}$$

$$\Theta := \Theta - \alpha * \frac{V_{d\theta}^{correct}}{\sqrt{S_{d\theta}^{correct}} + \varepsilon} * d\Theta$$

Onde, $V_{d\theta}$ e $S_{d\theta}$ são as contribuições do algoritmo de momentum e RMSprop respetivamente. A variável ε assegura que a não se divida por um número perto de zero, assegurando estabilidade

numérica. Este algoritmo combina as vantagens dos dois algoritmos e implementa a correção de bias, para evitar inícios lentos do estimador. Os hiperparâmetro são [31]:

- learning rate α, deve ser testado e encontrado o melhor.
- β_1: Na literatura é 0.9 e é o hiperparâmetro associado ao momentum.
- β_2: Na literatura é 0. 999 e é o hiperparâmetro associado ao RMSprop.
- ε na ordem do 10^-8.

Assim, a proposta para solucionar o problema de previsão de RUL para rolamento a partir de série temporal é usar uma combinação de LSTM com otimização ADAM.

Capítulo 5

Conjunto de dados e Set-up Experimental

O objetivo deste projeto é estimar o tempo útil de vida restante de rolamentos. Este é um tema comum em manutenção preventiva (PHM), já que a falha destes componentes é uma das principais causas de falha em máquinas rotativas. Múltiplos conjuntos de dados foram encontrados com o propósito de treinar modelos de previsão baseados em dados.

Assim neste capitulo descreverei cada conjunto de dados, como foram gerados e como fiz a preparação dos dados para serem entregues aos modelos de treino. Usando a análise proposta em capítulos anteriores, procurarei justificar cada escolha e interpretar os resultados.

O conjunto de dados utilizado nesta experiência foram:***C-MAPSS****: Prognostics CoE at NASA Ames - Engine degradation simulation* [18].

Começarei este capitulo por explicar como foi originado este conjunto de dados, e basear a escolha da abordagem escolhida na natureza do mesmo.

5.1 C-MAPSS data

C-MAPSS é uma ferramenta de simulação de um motor turbofan comercial de grande porte. O 'software' é codificado no ambiente MATLAB® e Simulink® e inclui uma série de parâmetros de entrada editáveis que permitem ao utilizador inserir valores específicos relativamente ao perfil operacional, controladores em malha fechada, condições ambientais, etc. O C-MAPSS simula um modelo de motor da classe de impulso de 90.000 lb e a extensão inclui um modelo atmosférico capaz de simular operações em (i) altitudes que variam do nível do mar a 40.000 pés (12,19 km), (ii) números de Mach de 0 a 0,90 e (iii) temperaturas ao nível do mar de -60 a 103 °F (ca. 39 °C). A extensão também inclui um sistema de simulação de potência que permite a operação do motor numa ampla faixa de níveis de impulso em todas as condições de voo.

O diagrama do motor na figura 5.1 mostra os principais elementos do modelo do motor e o fluxograma na figura 5.2 mostra como várias sub-rotinas são montadas na simulação. No total o modelo de simulação disponibiliza 14 variáveis de entrada que incluem o fluxo de combustível e

um conjunto de 13 entradas de parâmetros de saúde que permitem ao utilizador simular os efeitos de falhas e deterioração em qualquer um dos cinco componentes rotativos do motor (Fan, LPC, HPC, HPT e LPT). As saídas incluem várias 'interfaces' de resposta dos sensores e margens de operação. Um total de 21 variáveis, de um total de 58 saídas diferentes disponíveis no modelo, foram utilizadas na criação do conjunto de dados. [18] A simulação feita em malha fechada.

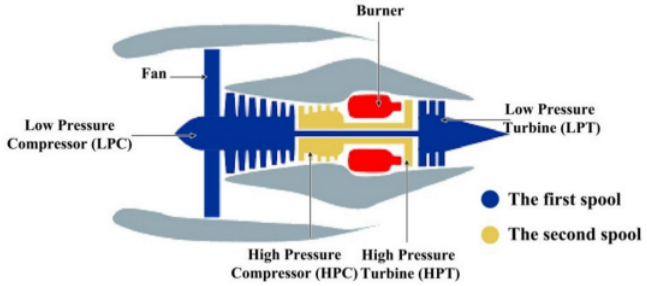


Figura 5.1: Motor turbofan

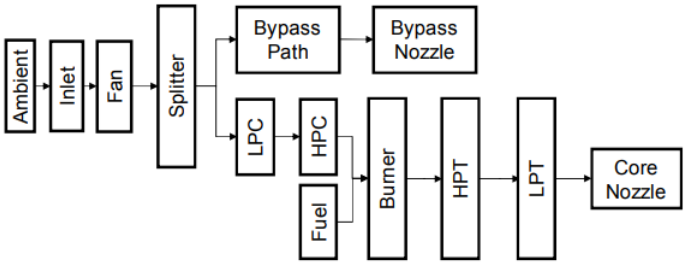


Figura 5.2: Fluxograma do simulador

Assim, os dados do C-MAPSS são o resultado de um modelo de simulação de um tipo específico de motor turbofan, figura 5.1, [18]. Nesse processo de simulação baseado em modelo, medições de funcionamento até a falha são registadas em diferentes condições. Durante a simulação, os dados também são contaminados com ruído de diferentes fontes para imitar cenários reais. Como resultado, os dados massivos são divididos em 4 subconjuntos de deteção de falhas (FD001, FD002, FD003 e FD004), onde cada subconjunto contém um número considerado de ciclos de vida do motor em condições de operação e anomalias diferentes. A série temporal (ciclos) que foram produzidos ao modificar o fluxo e as eficiências do módulo HPC a partir das configurações iniciais (indicando deterioração normal) para valores correspondentes ao limiar de falha. A degradação de outros módulos não foi intencionalmente incluída no conjunto de dados.

As anomalias seguiam modelos de deterioração exponencial. Assim, esta evolução exponencial denota a progressão de uma anomalia na turbina e apesar de fator exponencial ser limitado superiormente, foi escolhido aleatoriamente em cada simulação. A série temporal foi gerada até o critério de falha do sistema ter sido cumprido. Esta simulação gerou assim um conjunto de séries temporais de dados sensoriais geralmente utilizados nestas aplicações. Não só a progressão de

uma anomalia, que podemos encontrar no mundo real, mas dados até à falha de um sistema. [18]

O objetivo era gerar conjuntos de dados de treino, teste e validação para o desenvolvimento de prognósticos baseados em dados. Para isso, um número grande de simulações foram criadas com as seguintes propriedades:

- Simulação de degradação no módulo HPC sob 6 combinações diferentes de altitude, TRA (Thrust Reversal Actuator), e número Mach em condições operacionais. Margens sensoriais (fan, HPC, LPC e EGT) foram usadas para calcular um índice de saúde e determinar o critério de paragem da simulação.
- Séries temporais de amostras, incluindo variáveis operacionais que variam de uma condição inicial não definida para um limiar de falha.
- Divisão dos dados em conjunto de treino, conjunto de teste e conjunto de validação. O conjunto de treino continha trajetórias que terminaram no limiar de falha, enquanto os conjuntos de teste e validação foram ajustados para parar algum tempo antes do limiar de falha.
- A variação do RUL (Remaining Useful Life) foi expandida para o conjunto de validação para testar a robustez dos algoritmos treinados no conjunto de teste. Os RULs do conjunto de teste variaram entre 10 e 150 ciclos, enquanto os RULs de validação variaram entre 6 e 190 ciclos. No entanto, todas as outras características, como variação no desgaste inicial, níveis de ruído e parâmetros de degradação, permaneceram inalteradas.

Entende-se que os dados são massivos, enquanto o ruído acrescenta complexidade nos dados. Além disso, a mudança contínua nas condições de funcionamento leva a uma maior dinâmica nos dados. Os dados são registados em séries temporais num processo sequencial 'online'. Nesse contexto, uma vez que os dados estão disponíveis, complexos, dinâmicos e orientados 'online', a metodologia de aprendizado mais apropriada, segundo o esquema sugerido no capítulo 3 deve ser, um modelo de DL adaptativo 'online' capaz de ser atualizado dinamicamente para lidar com análise de séries temporais.

Nesse contexto, exemplos encontrados na literatura usam modelos de aprendizagem DL adaptativos. Por exemplo, os trabalhos propostos em [32], [33], [34], [35] e [36] geralmente utilizam variantes de RNN, como long short-term memory (LSTM), gated recurrent unit (GRU) e adaptive denoising 'online' sequential extreme learning machine (OSELM). Enquanto apenas alguns estudos não consideram a aprendizagem adaptativa, como em [37] e [38], onde CNN é o principal algoritmo de aprendizagem.

5.2 Experiência

Para fundamentar os argumentos anteriormente expostos, foram conduzidas experiências utilizando código livre encontrado no gitHub, com pequenos ajustes quando necessário. O código foi escrito em python e os respetivos jupyter notebooks podem ser encontrados aqui: [39]. Neste repositório podemos encontrar várias implementações de soluções baseadas em dados que procuram

solucionar o problema de cálculo de RUL para o conjunto de dados C-MAPSS. Essencialmente existem três etapas nesta experiência: primeiramente, uma descrição do conjunto de dados e tratamento dos mesmos; depois o treino de uma versão de uma RNN, a LSTM a partir de uma porção de treino do conjunto de dados, aqui foram usados dois tipos de algoritmos de otimização, primeiramente SGD e depois ADAM para se entender qual o impacto de se usar um algoritmo apropriado. Uma implementação usando svr (support vector regression) é apresentada para servir de comparação entre modelos ML clássicos e DL.

Descrição do conjunto de dados O conjunto de dados é composto por múltiplas simulações *run to failure* em quatro configurações de operação e tipos de falha:

Data Set id	Conjunto de treino	Conjunto de Teste	Condições	Tipo de falha.
FD001	100	100	Nível do mar	HPC
FD002	260	259	Atmosfera	HPC
FD003	100	100	Nível do mar	HPC e Fan
FD004	248	249	Atmosfera	HPC e Fan

O conjunto de dados de treino contém vários motores sendo executados simultaneamente. Por exemplo, o conjunto de treino do FD001 contém dados para 100 motores até à falha. No entanto, os conjuntos de teste são ligeiramente diferentes. Embora as simulações tenham sido conduzidos até à falha, a série temporal não é completamente disponibilizada. Em vez disso, são fornecidos dados para um número arbitrário de ciclos para cada motor do conjunto de teste. A tarefa consiste, então, em prever após quantos ciclos cada motor irá falhar. Esse valor previsto é a nossa Vida Útil Restante (RUL - Remaining Useful Life). O objetivo é prever a RUL com a maior precisão possível a partir de dados operacionais. É por isso que chamamos previsão de RUL baseada em dados.

Por exemplo, no Data set de teste FD01, encontramos dados correspondentes a 31 ciclos de vida do motor, com esses dados o nosso modelo de previsão deverá prever que o motor ainda irá funcionar 112 ciclos, isto é, o seu RUL é 112. Esta informação está no Data set RUL FD01 na posição correspondente ao motor 1. Assim, sabemos que para o motor 1 foram simulados 143 ciclos até ser considerado que o motor falhou.

Para analisar o conjunto de dados, este é importado para um Panda dataframe. Nos próximos parágrafos faço uma análise mais profunda da forma e conteúdo dos dados, usando métodos estatísticos e apresentando funções auxiliares encontradas em [39], estas funções resolvem três principais desafios na preparação dos dados: (i) Definição do modelo de degradação; (ii) Definição da janela de dados; (iii) Normalização dos dados.

Conjunto de dados de treino: Após importado o ficheiro *trainXXX.txt*, é gerada uma matriz onde a primeira coluna identifica o motor, a segunda identifica o ciclo de vida, da terceira à quinta-coluna são identificadas e caracterizadas as definições de operação e por fim as restantes contêm os dados crus dos sensores.

A informação de RUL para cada momentos pode ser inferida, sabendo que o motor 1 funcionou 192 ciclos, os dados do motor 1 no seu primeiro ciclo de vida tem um RUL de 191 ciclos.

Conjunto de dados de teste: Estruturalmente os dados de teste são similares, obtendo-se a mesma matriz que o exemplo anterior, mas desta vez os dados são truncados num número arbitrário de ciclos. Havendo assim a necessidade de uma matriz de 1 coluna que contém os correspondentes RUL para cada motor do conjunto de teste.

Deve-se observar que os dados de teste nunca devem ser usados durante o treino de um algoritmo.

Os dados de teste não são usados para o treino dos modelos. Durante o processo iterativo de treino de um modelo é importante verificar se o erro de treino desce pelo sucesso do algoritmo ou se simplesmente o algoritmo decorou os dados de teste. No último caso, mesmo que se obtenha um bom erro de treino, o modelo irá ter uma má desempenho para dados nunca vistos (data set de teste, ou aplicação real). Assim, é extraída uma parte dos dados de treino como conjunto de validação e usamos o conjunto de validação para verificar o desempenho do modelo treinado a cada iteração. Uma vez que estejamos satisfeitos com o modelo final, podemos usar os dados de teste para avaliar o seu desempenho.

Modelo de degradação: Este tipo de previsão, previsão de RUL é por definição uma variável que evolui linearmente. Neste projeto são apresentados dois tipos. No primeiro, o RUL começa no número máximo de ciclos para um dado motor e decresce até zero linearmente. No segundo é escolhido um valor para o qual, inicialmente (por um número de ciclos), o RUL é fixo. Esse número fixo é chamado de RUL inicial. Quando o valor do RUL atinge o RUL inicial, a partir desse ponto o RUL segue um modelo de degradação linear. As duas figuras 5.3 representam os dois modelos. Neste caso o valor para o RUL fixo inicial foi 125.

A função

```
process_targets(data_length, early_rul):
```

encontra e retorna como um vetor de uma dimensão o RUL do conjunto de dados. Recebe como parâmetros o tamanho da série temporal e o valor inicial do RUL.

Janela de dados: Para gerar qualquer tipo de estrutura e divisão de matrizes a partir dos dados é utilizada a seguinte função:

```
output_data, output_targets = process_input_data_with_targets(input_data,
                                target_data, window_length, shift)
```

Dependendo do valor dos parâmetros `window_length` e `shift`, esta função gera lotes de dados e alvos a partir de `input_data` e `target_data`, sendo o último opcional. A função retorna `output_data` e, opcionalmente, `output_targets` (apenas se `target_data` for fornecido).

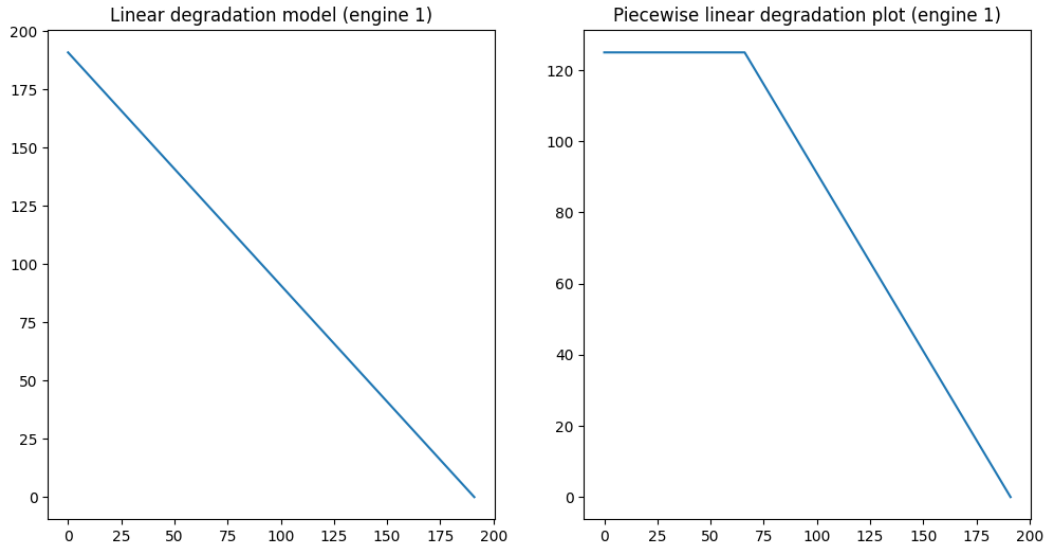


Figura 5.3: Modelo de degradação linear e modelo de degradação linear segmentado (piecewise linear degradation model)

Se nenhum `target_data` for fornecido (ou seja, `target_data = None`), nenhum `output_targets` será gerado. Este caso deve ser aplicado no conjunto de dados destinado a teste.

Se `target_data` for fornecido, a função gera `output_targets`, que é uma matriz 1D contendo os alvos correspondentes aos lotes de dados. O valor do alvo para cada lote é determinado pelo último valor do intervalo de deslocamento `(shift*batch + (window_length- 1))` nos dados de destino.

A matriz `output_targets` é inicializada com valores NaN e tem a forma (`num_batches`, `window_length`, `num_features`), Onde `num_batches` é o número de lotes, `window_length` é o comprimento da janela e `num_features` é o número de recursos dos dados de entrada (número de colunas).

O argumento `shift` está intimamente relacionado ao número de linhas de sobreposição. Por exemplo, se o comprimento da janela for 30 e o shift for 2, isso significa que há uma sobreposição de 28 linhas entre lotes sucessivos de dados. A seguinte fórmula é usada para determinar o número de lotes:

```
Number of batches =
    int(np.floor((len(input_data) - window_length)/shift)) + 1
```

A figura 5.4 a seguir ilustra isso para um conjunto de dados fictício com 6 linhas e 4 colunas.

Assim, esta função permite processar os dados de entrada em lotes segmentados usando janelas deslizantes, facilitando a preparação dos dados para modelos de aprendizagem computacional. É importante garantir que os dados de entrada e alvo estejam no formato correto antes de usar essa função para evitar erros ou resultados indesejados.

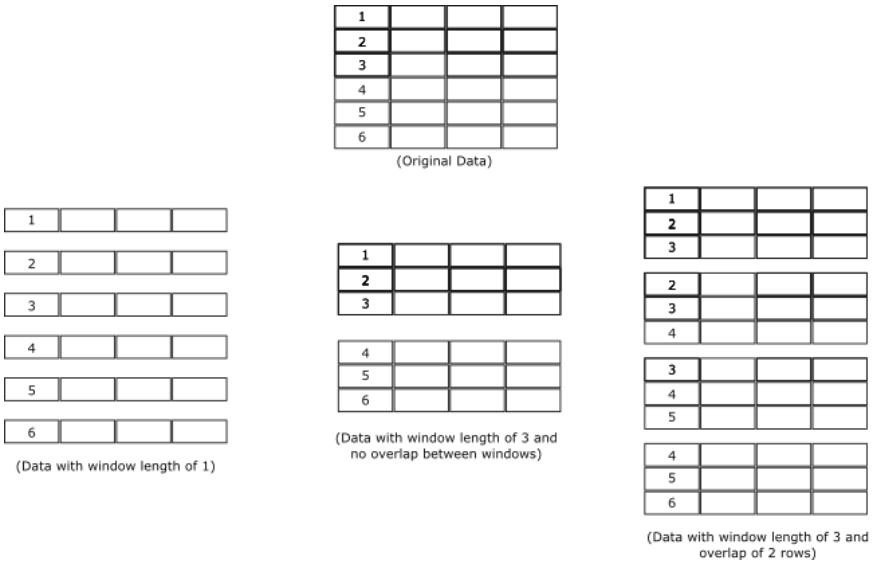


Figura 5.4: Exemplos de janelas de sobreposição possíveis

Para processar o conjunto de dados de teste podemos utilizar uma outra função auxiliar que, para além de dividir os dados em lotes de um específico tamanho, permite escolher quantos (do fim) desses lotes serão utilizados para testar o algoritmo de previsão:

```
(batched_test_data_for_an_engine, extracted_num_test_windows)=
          process_test_data(test_data_for_an_engine,
                            window_length, shift,
                            num_test_windows)
```

Por exemplo, suponhamos que, para um determinado motor, os dados do conjunto de testes tenham uma forma de (45×14). Para `window_length` = 30 e `shift` = 1, podemos extrair 16 conjuntos de dados segmentados de tamanho (30×14). Se utilizar apenas a última janela para fazer a previsão poderia obter uma previsão errada no caso de haver alguns valores discrepantes (outliers). Uma abordagem para contornar esse problema é considerar, por exemplo, os últimos 5 exemplos em vez de apenas o último exemplo para cada motor. Assim, obteremos 5 valores de previsão do RUL. Podemos fazer a média desses valores para chegar a nossa estimativa final do RUL. A nossa estimativa do RUL será um pouco conservadora, mas robusta em relação a outliers no último exemplo de teste.

Esta função recebe como argumentos a série temporal de teste para um dado motor. (`test_data_for_an_engine`), o número de lotes a serem extraídos (`num_test_windows`) por padrão, ela extrai apenas o último exemplo.

A função retorna os últimos exemplos e o número de últimos exemplos (um valor escalar) como saída. Precisaremos do segundo valor de saída posteriormente. Se estivermos a extrair mais de um exemplo, teremos que fazer a média dos resultados de previsão.

Normalização dos dados: É por boa prática normalizar o data set, especialmente quando se usa tipos de sensores que produzem valores de diferentes escalas. Foi então aplicada normalização de escala a cada motor individualmente e normalização do conjunto de dados inteiro.

Para normalização individual de cada motor é usada a função `MinMaxScaler()` da biblioteca *sklearn* com parâmetro `feature_range = (-1, 1)` . Este processo de normalização do conjunto de treino e teste é feito em três linhas de código:

```
scaler = MinMaxScaler(feature_range = (-1, 1))
temp_train_data = scaler.fit_transform(temp_train_data)
temp_test_data = scaler.transform(temp_test_data)
```

Esta função executa o seguinte algoritmo:

$$X^i_{std} = \frac{X^i - X_{min}}{X_{max} - X_{min}}$$

$$X^i_{normalizado} = X^i_{std} * (max - min) + min$$

Onde X^i é o i-ésimo elemento da coluna, X_{max} e X_{min} são respetivamente o valor máximo e mínimo da coluna. X^i_{std} será o desvio padrão do i-ésimo elemento e $X^i_{normalizado}$ é esse mesmo elemento normalizado. Por fim, min e max são os parâmetros passados na função, respetivamente (-1, 1).

Para normalização do conjunto de dados inteiro é usada a função `StandardScaling` com atributos padrão. Esta função normaliza atributos subtraindo a média (do atributo na série temporal) e normaliza o desvio padrão para unitário. A função aplica o algoritmo a cada atributo separadamente.

Análise estatística: Algumas colunas não são consideradas para treino e teste do modelo; O autor desta experiência escolheu utilizar apenas os dados dos sensores. Além disso, após uma análise da densidade dos valores de série temporal que constitui cada coluna, foram encontrados alguns valores constantes, e por boa prática, não foram considerados. Após remover as primeiras 5 colunas ('id' de motores e condições de operação) e as 7 colunas que contém valores constantes, os nossos dados têm uma forma de (20631×14), isto é uma matriz (m × n) em que m são momentos t da série temporal de 100 motores e n o número de atributos.

Um algoritmo generaliza bem se os dados não vistos forem semelhantes em distribuição aos dados de treino. Os dados não vistos não precisam de ser exatamente da mesma distribuição dos dados de treino, mas não devem ser muito diferentes dos dados de treino. Na figura 5.5 verificaremos a distribuição dos dados de treino e teste.

Embora não sejam exatamente iguais, os dados de teste têm uma distribuição semelhante aos dados de treino. Portanto, se projetado cuidadosamente, o nosso algoritmo generalizará bem.

Assim, reunimos as condições para aplicar o modelo de treino escolhido, fazendo as transformações no conjunto de dados mais indicadas. Como já foi referido neste capitulo, o modelo escolhido foi uma rede neuronal recursiva com memória LSTM.

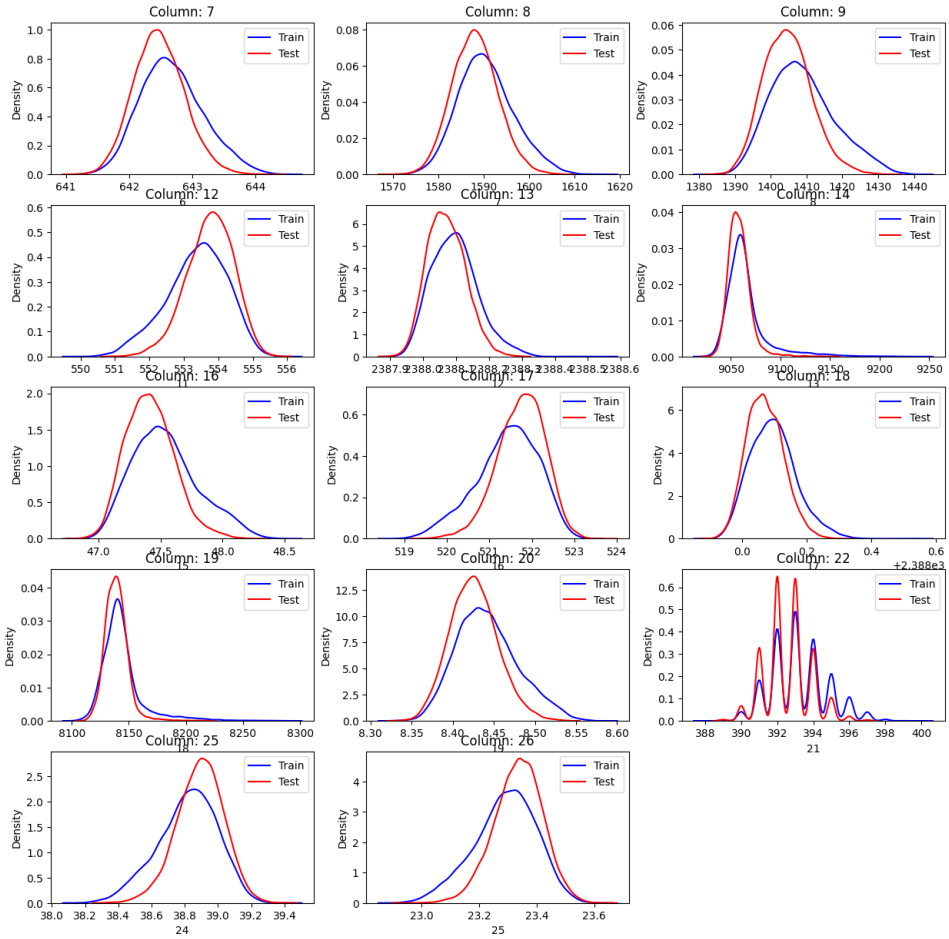


Figura 5.5: Distribuição de dados de sensores

Definição de modelo: Começamos por definir a função que irá compilar o nosso modelo de DL. `def create_compiled_model()`: dentro desta função a arquitetura do nosso modelo deve ser definida: consiste em três camadas LSTMs e 3 camadas de NNs (`Dense`).

```
model = keras.Sequential([
        layers.LSTM(128, input_shape=(window_length, 14),
    return_sequences=True, activation="tanh"),
        layers.LSTM(64, activation="tanh", return_sequences=True),
        layers.LSTM(32, activation="tanh"),
        layers.Dense(96, activation="relu"),
        layers.Dense(128, activation="relu"),
        layers.Dense(1)
    ])
```

O modelo é definido como sequencial. Num modelo sequencial, cada camada está ligada à

camada seguinte sequencialmente. É o tipo mais simples e comum de modelo utilizado no Keras para construir modelos de aprendizagem profunda. Os modelos sequenciais são adequados para construir modelos nos quais a saída de uma camada serve como entrada para a próxima camada.

Camadas NN são usadas após as camadas LSTM para processamento adicional e extração de características a partir da saída das camadas LSTM.

Camadas LSTM (Long Short-Term Memory) são um tipo de camada de rede neural recorrente (RNN) usadas para processamento de dados sequenciais, como séries temporais. As camadas LSTM são eficazes na captura de dependências e padrões de longo prazo em dados sequenciais.

No entanto, a saída das camadas LSTM pode conter padrões complexos e representações que precisam de processamento adicional e transformação antes de fazer previsões ou extrair informações significativas. Camadas densas, também conhecidas como camadas totalmente conectadas, são usadas para esse propósito.

As camadas densas conseguem aprender mapeamentos não lineares complexos entre a entrada e a saída. Elas consistem em múltiplos neurónios, onde cada neurónio está conectado a todos os neurónios da camada anterior. Ao adicionar camadas densas após as camadas LSTM, o modelo pode aprender características e representações de alto nível adicionais a partir da saída das camadas LSTM.

Assim, camadas densas pretendem refinar ainda mais as representações aprendidas e transformá-las de uma forma adequada.

Definir `return_sequences= True` na camada LSTM garante que a camada retorne a sequência de saída para cada etapa de tempo. Assim, as camadas adjacentes `LSTM` e `Dense` têm acesso às saídas intermediárias.

Função de ativação: Foram usados dois tipos de função de ativação: (i) Tanh, (ii) ReLU. Este tema foi explorado no capítulo 4, na figura 4.5 podemos observar o gráfico de cada figura e a sua derivada no seu domínio. São funções convergentes, limitadas e as suas saídas normalizadas. As suas derivadas têm valores altos longe do mínimo global e valor baixo perto do mínimo global, sendo zero no seu mínimo. Fazendo-as funções indicadas quando se usa algoritmos de otimização baseados em gradientes, como é este caso.

Função de otimização: Como função de otimização foi usada a função ADAM e SGD, também profundamente explorada em 4.

A experiência foi conduzida da seguinte forma, foi treinador um modelo utilizando as especificações anteriores e uma classificação objetiva foi feita como descrito no capítulo 3, usando uma função de pontuação que favorece previsões antecipadas relativamente a previsões tardias.

Foram conduzidas duas experiências, uma usando otimização adam (batch size 128) e outra com stochastic gradient descent (batch size = 1), para verificar o efeito de cada um na evolução da função de custo a cada época. E uma usando SVR para servir de representação de métodos de ML clássicos Os seguintes resultados foram obtidos:

SVR É uma técnica de aprendizado de máquina utilizada para realizar regressão, ou seja, para prever valores numéricos a partir de dados de entrada. Foi usado grid search para encontrar o melhor conjunto de hiperparametros, entre eles quatro parâmetros de regularização e quatro epsilons. Obteve uma pontuação S: 1004.75. Tempo total de processamento: 45 min.

Na figura 5.6 é possível observar uma comparação entre o RUL esperado e obtido em que cada vértice do gráfico representa um motor diferente, num total de 100. Os hiperparâmetro encontrados pelo grid search ideias foram: SVR(C=10, epsilon=10).

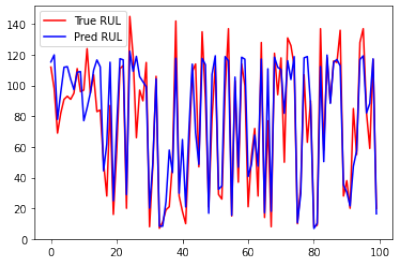


Figura 5.6: Previsões de RUL de SVR usando o conjunto de dados completo

Tanto o SGD como o ADAM foram testados usando LSTM como modelo preditivo. O intuito desta experiência era comparar ADAM, algoritmo de otimização mais avançado, com o modelo mais básico de otimização baseado em gradiente, o SGD.

SGD É um algoritmo de otimização que usa amostras aleatórias de dados de batch size: 1, em cada etapa para ajustar iterativamente os parâmetros de um modelo de DL. Obteve-se assim uma evolução oscilatória e convergente, obtendo-se um RMSE de teste de 42.57 e uma pontuação de S = 27629.224. Tempo processamento: 208s/epoch - 13ms/step. Tempo total: $\tilde{3}$h

Na figura 5.7 é possível observar à esquerda a evolução de RMSE de validação em cada época, no total de 50. E à direita uma comparação entre o RUL esperado e obtido em que cada vértice do gráfico representa um motor diferente, num total de 100.

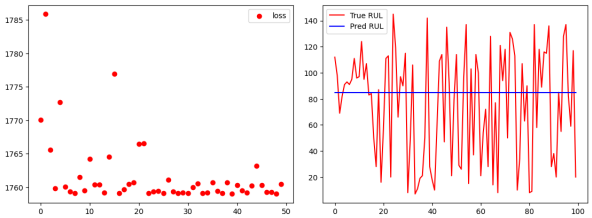


Figura 5.7: Evolução de erro RMSE de validação e previsões de RUL usando o conjunto de dados completo

Neste caso, apesar de certas oscilações, o erro de validação convergiu, mas o modelo não conseguiu captar a informação temporal dos dados, isto porque o algoritmo usa batch size: 1. O algoritmo aprendeu um valor "médio"para as previsões de RUL.

ADAM Este algoritmo adapta a taxa de aprendizagem (learning rate) para cada parâmetro da rede neural com base nas estimativas dos momentos do gradiente. Obteve-se assim uma evolução estável e convergente, obtendo-se um RMSE de teste de 16.647 e uma pontuação de S = 551.765. Tempo processamento: 22s/epoch - 201ms/step. Tempo total: $\tilde{1}$4min.

Na figura 5.8 é possível observar à esquerda a evolução de RMSE de validação em cada época, no total de 50. E à direita uma comparação entre o RUL esperado e obtido em que cada vértice do gráfico representa um motor diferente, num total de 100.

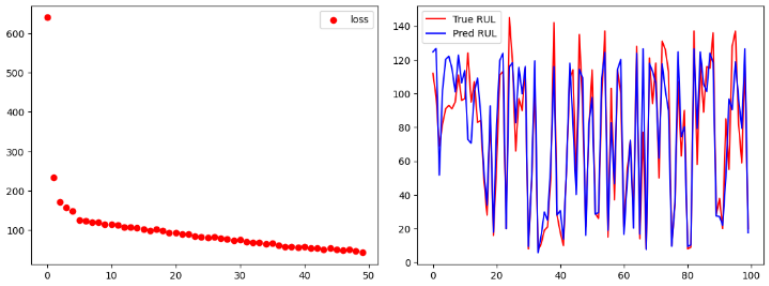


Figura 5.8: Evolução de erro RMSE de validação e previsões de RUL usando o conjunto de dados completo

	SVR	LSTM + SGD	LSTM + ADAM
RMSE de teste	-------	42.57	16.647
Pontuação S	1004.75	27629.224	551.765
Tempo /epoch	-----	208s	22s
Tempo Total	45min	3h	14 min

Figura 5.9: Síntese dos resultados

Conclusão: Estes resultados são elucidativos das vantagens da combinação de modelos de inteligência artificial especializados em processamento de séries temporais em combinação com algoritmos de otimização avançados que permitem uma convergência mais estável. Verificamos que LSTM com otimização ADAM é não só mais eficiente na pontuação proposta, como com o melhor erro de validação como também em termos de tempo de processamento.

Capítulo 6

Conclusões e Trabalho Futuro

Nesta tese de mestrado foram explorados métodos de manutenção preventiva baseados em dados, usando redes neuronais profundas recorrentes para previsão do tempo de vida restante (RUL) de rolamentos. As principais contribuições podem ser sumariadas da seguinte forma:

- Foi proposta uma arquitetura de modelo baseada em LSTM que combina aprendizagem automática de características a partir de dados brutos do sensor de vibração com modelagem de sequência de tendências de degradação.
- Foi demonstrado que o modelo superou abordagens de aprendizado de máquina superficial (shallow ML) utilizadas como referência.
- Um estudo e discussão sobre metodologias AI foram discutidas e articuladas. Estratégias de implementação foram discutidas e apresentadas para dois tipos de algoritmos de otimização diferentes.

Os resultados validam as redes recorrentes profundas como uma ferramenta eficaz para diagnósticos e prognósticos baseados em dados de equipamentos complexos. Com otimizações adicionais, as técnicas mostram potencial para implementação em tempo real em dispositivos incorporados para monitorização de condições.

Existem oportunidades para trabalhos futuros. Conjuntos de dados de treino maiores, com mais variabilidade, podem ajudar a melhorar a generalização do modelo. Métodos baseados em modelos físicos e modelagem híbrida poderiam potencialmente aprimorar ainda mais a precisão. Métodos de ajuste automático de hiperparâmetros podem ajudar a agilizar o treino.

Para um estudo mais aprofundado da viabilidade de um sistema de PHM para condições adversas, uma simulação do 'hardware' poderia ser feita, com o fim de testar as limitações do sistema e verificar se o modelo funcionaria em condições reais de utilização. Com mais otimizações, a abordagem RNN baseada em dados pode ser implementada em dispositivos embebidos de baixa potência para prognósticos em tempo real.

No geral, esta tese sumariza os principais componentes de um sistema de manutenção preventiva (PHM), faz um apanhado de estado de arte dos subsistemas e apresenta algumas simples implementações ajustadas a dois tipos de conjuntos de dados diferentes.

Referências

[1] Vepa Atamuradov, Kamal Medjaher, Pierre Dersin, Benjamin Lamoureux, e Noureddine Zerhouni. Prognostics and health management for maintenance practitioners - Review, implementation and tools evaluation. *International Journal of Prognostics and Health Management*, 8(3):1–31, Dezembro 2017. URL: https://hal.science/hal-03512803, doi:10.36001/ijphm.2017.v8i3.2667.

[2] Ritwick Roy. Neural networks: Forward pass and backpropagation. URL: https://towardsdatascience.com/neural-networks-forward-pass-and-backpropagation-be3b75a1cfcc.

[3] Akash Wagh. Gradient descent and its types. URL: https://www.analyticsvidhya.com/blog/2022/07/gradient-descent-and-its-types/.

[4] Andrew Ng. Deep learning specialization. URL: https://www.coursera.org/specializations/deep-learning?utm_source=deeplearningai&utm_medium=institutions&utm_campaign=SocialYoutubeDLSC2W1L01.

[5] Tarek Berghout e Mohamed Benbouzid. A systematic guide for predicting remaining useful life with machine learning. *Electronics*, 11(7), 2022. URL: https://www.mdpi.com/2079-9292/11/7/1125, doi:10.3390/electronics11071125.

[6] ACARE. Structure - acare. URL: https://www.acare4europe.org/structure/.

[7] Head of Communications Maria-Fernanda Fau. Powering partnerships towards innovation architecture, 2020.

[8] B. Saha, K. Goebel, e J. Poll, S;Christophersen. Prognostics methods for battery health monitoring using a bayesian framework. *Instrum. Meas.s*, páginas 58, 291–296, 2009.

[9] K.B. Misra. Maintenance enginerring and maintenability: An introduction. Relatório técnico, Springer: London, UK, 2008.

[10] A Tondon, N.; Choudhury. A review of vibration and acoustics measurement methods for the detection of defects in rolling element bearing. páginas 32, 469–480, 1999.

[11] Van Nguyen, Marios Kefalas, Kaifeng Yang, Asteris Apostolidis, Markus Olhofer, Steffen Limmer, e Thomas Bäck. *Probabilistic Prognostics and Health Management of Energy Systems*. Springer, 2017. http://hdl.handle.net/11449/170039.

[12] Mostafa Anwar Taie, Mohammed Diab, e Mohamed ElHelw. Remote prognosis, diagnosis and maintenance for automotive architecture based on least squares support vector machine and multiple classifiers. Em *2012 IV International Congress on Ultra Modern Telecommunications and Control Systems*, páginas 128–134, 2012. doi:10.1109/ICUMT.2012.6459652.

[13] Riad A.M. Elattar H.M., Elminir H.K. Prognostics: a literature review. *Complex and Intelligent Systems*, 06 2016. doi:https://doi.org/10.1007/s40747-016-0019-3.

[14] Van Nguyen, Marios Kefalas, Kaifeng Yang, Asteris Apostolidis, Markus Olhofer, Steffen Limmer, e Thomas Bäck. A review: Prognostics and health management in automotive and aerospace. *International Journal of Prognostics and Health Management*, 10, 11 2019. doi:10.36001/ijphm.2019.v10i2.2730.

[15] Jie Gu. Pecht M. Physics-of-failure-based prognostics for electronic products. *Transactions of the Institute of Measurement and Control.*, 2009. doi:doi:10.1177/0142331208092031.

[16] T. Tinga. *Principles of Loads and Failure Mechanisms*. Springer, 2013. ISBN : 978-1-4471-4916-3.

[17] Abhinav Saxena, Jose Celaya, Edward Balaban, Kai Goebel, Bhaskar Saha, Sankalita Saha, e Mark Schwabacher. Metrics for evaluating performance of prognostic techniques. *Prognostics and Health Management.*, 2008.

[18] A. Saxena, K. Goebel, D. Simon, e N. Eklund. Damage propagation modeling for aircraft engine run-to-failure simulation. *International Conference on Prognostics and Health ManagemenT*, 2008.

[19] Olga Fink, Quin Wang, Markus Svensén, Pierre Dersion, Wan-Jui, e Melanie Ducoffe. Engineering applications of artificial intelligence. *Elsevier*, páginas 103–114, Abril 2020.

[20] Ian Goodfellow, Yoshua Bengio, e Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[21] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[22] PhD Jason Brownlee. Why optimization is important in machine learning. *Machine Learning Mastery*, 2021. URL: `https://machinelearningmastery.com/why-optimization-is-important-in-machine-learning/`.

[23] Diederik P. Kingma e Jimmy Ba. Adam: A method for stochastic optimization, 2017. arXiv:1412.6980.

[24] Diederik P. Kingma e Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv e-prints*, página arXiv:1412.6980, Dezembro 2014. arXiv:1412.6980, doi:10.48550/arXiv.1412.6980.

[25] PhD Adrian Tam. Optimization for machine learning crash course. *Machine Learning Mastery*, 2021. URL: `https://machinelearningmastery.com/optimization-for-machine-learning-crash-course/`.

[26] von Melchner L. Pallas S. e Sur M. Visual behaviour mediated by retinal projections directed to the auditory pathway. *Nature*, Abril 2000. `https://doi.org/10.1038/35009102`.

[27] *Deep Learning Book*. Data Science Academy. https://www.deeplearningbook.com.br/.

[28] MIke James. Microsoft wins imagenet using extremely deep neural networks. URL: `https://www.i-programmer.info/news/105-artificial-intelligence/9266-microsoft-wins-imagenet-using-extremely-deep-neural-networks.html`.

[29] Oinkina. Understanding lstm networks, Agosto 2015. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ [último acesso em 2023-01-03].

[30] P.J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990. doi:10.1109/5.58337.

[31] KERAS. Adam. URL: https://keras.io/api/optimizers/adam/.

[32] Shuai Zheng, Kosta Ristovski, Ahmed Farahat, e Chetan Gupta. Long short-term memory network for remaining useful life estimation. Em *2017 IEEE International Conference on Prognostics and Health Management (ICPHM)*, páginas 88–95, 2017. doi:10.1109/ICPHM.2017.7998311.

[33] Yuting Wu, Mei Yuan, Shaopeng Dong, Li Lin, e Yingqi Liu. Remaining useful life estimation of engineered systems using vanilla lstm neural networks. *Neurocomputing*, 275:167–179, 2018. URL: https://www.sciencedirect.com/science/article/pii/S0925231217309505, doi:https://doi.org/10.1016/j.neucom.2017.05.063.

[34] Zhenyu Wu, Shuyang Yu, Xinning Zhu, Yang Ji, e Michael Pecht. A weighted deep domain adaptation method for industrial fault prognostics according to prior distribution of complex working conditions. *IEEE Access*, 7:139802–139814, 2019. doi:10.1109/ACCESS.2019.2943076.

[35] Huihui Miao, Bing Li, Chuang Sun, e Jie Liu. Joint learning of degradation assessment and rul prediction for aeroengines via dual-task deep lstm networks. *IEEE Transactions on Industrial Informatics*, 15(9):5023–5032, 2019. doi:10.1109/TII.2019.2900295.

[36] Sheng Xiang, Yi Qin, Jun Luo, Huayan Pu, e Baoping Tang. Multicellular lstm-based deep learning model for aero-engine remaining useful life prediction. *Reliability Engineering and System Safety*, 216:107927, 2021. URL: https://www.sciencedirect.com/science/article/pii/S0951832021004439, doi:https://doi.org/10.1016/j.ress.2021.107927.

[37] Xiang Li, Qian Ding, e Jian-Qiao Sun. Remaining useful life estimation in prognostics using deep convolution neural networks. *Reliability Engineering and System Safety*, 172:1–11, 2018. URL: https://www.sciencedirect.com/science/article/pii/S0951832017307779, doi:https://doi.org/10.1016/j.ress.2017.11.021.

[38] Han Li, Wei Zhao, Yuxi Zhang, e Enrico Zio. Remaining useful life prediction using multi-scale deep convolutional neural network. *Applied Soft Computing*, 89:106113, 2020. URL: https://www.sciencedirect.com/science/article/pii/S1568494620300533, doi:https://doi.org/10.1016/j.asoc.2020.106113.

[39] Biswajit Sahoo. Data-driven remaining useful life (rul) prediction, 9 2020. URL: https://biswajitsahoo1111.github.io/rul_codes_open/, doi:10.5281/zenodo.5890595.