UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

Implementação de Soluções para Confiabilidade de Dados em Redes de Sensores sem Fios

João Miguel Mendes Penim

Mestrado em Engenharia Informática

Especialização em Sistemas de Informação

Dissertação orientada por:
Prof. António Casimiro Ferreira da Costa
Prof. Pedro Miguel Frazão Fernandes Ferreira

2020

"Dependability of a computing system is the ability to deliver service that can justifiably be trusted."

- Avizienis [1]

Agradecimentos

Em primeiro lugar gostava de dizer que este percurso pelo mestrado em Engenharia Informática não foi fácil. Descobri novas coisas e posso dizer que cresci como pessoa, mas claro como todos, não estive sozinho e com este pequeno texto gostava de agradecer a todos os que me ajudaram, apoiaram e estiveram comigo neste percurso.

Quero agradecer ao Laboratório de Sistemas Informáticos de Grande Escala (LASIGE) por me ter dado a oportunidade de desenvolver a minha tese de Mestrado assim como a todos os meus colegas envolvidos no projecto AQUAMON.

Aos meus orientadores Professor António Casimiro, Professor Pedro Ferreira e um agradecimento especial ao Gonçalo Jesus do Laboratório Nacional de Engenharia Civil que mesmo não sendo meu orientador me foi ajudando sempre que precisei.

À minha família que me apoiou desde sempre nos bons e maus momentos, obrigado Mãe, Pai e Irmã.

À minha namorada Joana Requeijo que teve um papel muito importante neste percurso, muito obrigado por me aturares e ajudares.

Ao meu amigo Fábio Joel que esteve cá sempre para me apoiar, obrigado pelas gargalhadas e pelos bons momentos.

E a todos os meus amigos que, nas últimas 2 décadas partilharam bons e maus momentos comigo, bons e maus resultados, não só academicamente mas também na minha vida pessoal.

Um grande obrigado a todos.

Este trabalho foi suportado pela Fundação para a Ciência e a Tecnologia (FCT) através do projeto estratégico do LASIGE (UIDB/00408/2020 e UIDP/00408/2020) e através do projeto AQUAMON (PTDC/CCI-COM/30142/2017).

Resumo

Redes de sensores sem fios consistem num número variado de dispositivos electrónicos (nós) que realizam monitorização de factores físicos ou ambientais. Dependendo do tipo de sensor acoplado ao nó, este pode executar inúmeras medições, tal como, temperatura, pressão, movimentos, entre outros.

Estes dispositivos estarão sempre sujeitos a interferências devido a condições externas ambientais ou humanas, levando os sensores a relatar medições erradas tornando os relatórios não confiáveis. Quando estas medições inesperadas são produzidas, é necessário detectá-las e identificá-las em tempo real. Para tal serão combinadas diversas técnicas de redundância e fusão de dados que, em conjunto, permitirão melhorar a qualidade da monitorização.

O trabalho desenvolvido para esta dissertação concentra-se sobre o domínio dos sistemas ciber-físicos e está incluído no projecto AQUAMON.

Este projecto tem como objectivo principal o desenvolvimento de uma plataforma de recolha e tratamento de dados de sensores aquáticos. A monitorização contínua de ambientes aquáticos usando sensores de água é importante para diversas aplicações relacionadas com prevenção de acidentes, recursos hídricos e gestão da aquicultura e actividades recreativas. Portanto, é fundamental garantir a qualidade dos dados de monitorização para evitar falsos alarmes ou ignorar eventos relevantes.

Este trabalho irá concentrar-se na implementação de uma plataforma de acordo com uma arquitectura previamente definida, denominada de ANNODE. Esta arquitectura utiliza técnicas de Aprendizagem Automática, mais especificamente Redes Neuronais, com o intuito de criar previsões, medir a qualidade destas, detectar falhas e corrigi-las, em ambiente *off-line*. No entanto o foco deste trabalho, não é apenas a implementação desta arquitectura, mas sim a modificação desta permitindo utilizá-la num ambiente *on-line*, possibilitando o processamento de medições em tempo-real, mantendo todos os aspectos centrais anteriores. Como será possível ver nos capítulos seguintes, este trabalho obteve resultados positivos e comparando-os com a arquitectura original observaram-se melhorias.

Palavras-chave: Redes de sensores, Confiabilidade, Aprendizagem Automática, Previsão em tempo-real, Redes Neuronais

Abstract

Wireless sensor networks consist of a set of devices equipped with sensors, with processing capabilities, and connected together in a wireless network, which perform the measurement of physical or environmental factors and transmit these measurements to some central node. Depending on the type of sensor attached to the node, it can perform various measurements such as temperature, pressure, velocity, etc.

These devices will always be subject to interference due to external conditions, whether environmental or human, which can lead to poor communication between sensor nodes or can result in errors being introduced in measurements. When these unexpected measurements are produced, there is a need to detect and identify them, for which various methods can be implemented. This work will focus on the implementation of a monitoring platform in which the data processing methods for the detection and correction of these errors are based on neural networks.

This work is done in the context of the AQUAMON project, which aims to create a dependable monitoring platform for application in aquatic environments.

This work will focus on the implementation of a platform according to a previously defined architecture, called ANNODE. This architecture uses Machine Learning techniques, more specifically Neural Networks, in order to predict measurements and use this predictions to detect failures, measure the quality of measurements and correct erroneous measurements. The objective of the work was to implement a platform according to this architecture, therefore, rather than defining new methods for the detection of anomalies in sensor data, this work will exploit previous work done in AQUAMON on the definition of concrete methods for dependable monitoring, implementing them in a real-time platform, evaluating the implementation and proposing possible improvements.

Key-Words: Sensor networks, Dependability, Machine Learning, Real-time Forecasting

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Abreviações

RSSF	**R**edes de **S**Ensores **S**em **F**ios
RN	**R**edes **N**euronais
MLP	**M**ulti **L**ayer **P**erceptron
PMC	**P**erceptão **M**ulti **C**amada
MT	**M**omemtum **T**erm
LR	**L**earning **R**ate
EUA	**E**stados **U**nidos da **A**mérica
CMOP	**C**enter for **C**oastal **M**argin **O**bservation & **P**rediction
CDF	**C**umulative **D**ensity **F**unction

Capítulo 1

Introdução

Redes de Sensores Sem Fios (RSSF) refere-se a um grupo de nós inteligentes, constituídos por um micro-controlador com um propósito geral, integrado com um rádio e um ou vários sensores. Estes nós ficam espacialmente dispersos e dedicados a monitorizar e registar as condições físicas do ambiente. Uma RSSF é normalmente constituída por um nó - *sink node*, que recebe as medições dos nós sensores e envia-as para um local central. Estas medições são enviadas, muitas vezes para sistemas de suporte à decisão, onde são processados. Posteriormente podem alertar sistemas de emergência ou as autoridades relevantes. As RSSF medem condições ambientais como: temperatura, som, níveis de poluição, humidade, velocidade e direcção do vento, pressão, etc.

Perante esta informação é evidente que a qualidade e a validade das medições é crítica e deve conferir alta confiabilidade. A implementação de uma RSSF em ambientes severos, como ambientes aquáticos ou florestais onde estão sujeitos a condições incertas, deixa os nós vulneráveis a perturbações externas, podendo estas ser humanas ou ambientais. Um avanço importante para o aumento na qualidade das medições é a detecção de possíveis erros, que possam afectar os nós na rede. A caracterização destes em ambientes severos pode ser um desafio devido a mudanças inesperadas no ambiente, como por exemplo: incêndios ou inundações que criam padrões muito diferentes dos normais e não devem ser tratados como medições erradas.

O aumento da confiabilidade das aplicações de monitorização em redes de sensores sem fios é feito a partir da implementação de plataformas capazes de verificar a qualidade das medições e de corrigir medições incorrectas, utilizando informação de outros sensores e usando uma previsão calculada a partir de uma espécie de rede de sensores virtuais fornecidos por modelos de previsão dos ambientes monitorizados.

Para a concretização destes modelos de previsão será necessário perceber como implementar mecanismos de aprendizagem automática, como por exemplo redes neuronais. Este trabalho vai consistir no estudo, comparação e implementação de redes neuronais sobre medições de sistemas de monitorização com sensores, como será analisado nos próximos capítulos

1.1 Motivação

O mundo da aprendizagem automática é imenso e bastante aliciante. Mais especificamente as Redes Neuronais (RN), que já vêm a ser estudadas desde os anos 40 e são cada vez mais comuns. As RN são bastantes úteis para detecção de padrões e na resolução de problemas em diversas áreas, como: medicina, engenharia, bioinformática, entre outros. As RN são muito versáteis podendo reconhecer som, imagens, vídeo e texto. Sendo por este motivo muito apetecíveis no mundo da automação, robótica, monitorização e previsão de factores ambientais.

Como já foi referenciado, as RSSF muitas vezes são implementadas em ambientes severos onde podem ocorrer eventos e distúrbios externos imprevisíveis. Isto faz com que as medições dadas por estes sistemas de monitorização se tornem mais complexas e imprevisíveis, requerendo um esforço adicional. A tarefa de detecção de *outliers*, desvios e *offsets* por parte da RN torna-se ainda mais complicada onde certos padrões parecem ser erros mas não o são.

No desenvolvimento de uma solução para esta tarefa é possível comparar várias abordagens e estudar sobre os diversos tipos de redes Neuronais.

Será explorada uma metodologia já previamente implementada em *Matlab* (ANNODE [2]) num ambiente *off-line*. No contexto do projecto AQUAMON é necessário implementar uma plataforma que possa ser usada numa aplicação real, recolhendo e processando dados em tempo-real. Por outro lado, existe ainda a necessidade de estudar e implementar outras soluções baseadas em redes neuronais, no sentido de procurar melhorar a precisão da ferramenta no que diz respeito à detecção de erros.

1.2 Objectivos e contribuições

O objectivo principal é estimar e melhorar a qualidade das medições, sendo necessário:

- Estudar, aplicar e perceber as diferentes abordagens das redes neuronais e implementá-las de modo a obter uma métrica que demonstre a precisão de cada uma;
- Comparar cada abordagem e utilizar aquelas que tenham maior precisão;
- Integrar estas abordagens de processamento de dados numa plataforma de monitorização para demonstrar a eficácia de cada uma e daí retirar conclusões.

As principais contribuições deste trabalho consistem na proposta e avaliação de novas soluções de aprendizagem automática. As quais, como poderemos verificar no capítulo de resultados, demonstram ter melhor precisão do que a metodologia anteriormente proposta no projecto AQUAMON. Outra contribuição importante é a implementação de uma plataforma, em *Python*, capaz de receber e processar dados de sensores em tempo-real.

Neste trabalho serão usadas redes perceptrão multi-camada (*Multilayer Perceptron*) (MLP). Esta rede é semelhante à *Perceptron* [4], mas com várias camadas de "neurónios"ligados entre si por sinapses com pesos. As redes MLP têm uma metodologia *feedforward*, isto é as ligações entre neurónios não formam ciclos. O treino neste tipo de rede é geralmente feito através do algoritmo de *backpropagation* [5], mas existem outros algoritmos para este fim, como o *Rprop* [6]. Todos estes temas serão contextualizados no capítulo 2 na secção 2.2.

1.3 Metodologia

A inerente redundância fornecida pela implementação de vários sensores será explorada para aumentar a tolerância a falhas, aplicando procedimentos de fusão de medições do sensor em conjuntos de medições.

O conhecimento sobre o posicionamento temporal das medições dos sensores, permitirá que estas sejam relacionadas com precisão entre si para uma fusão precisa. Dadas as falhas detectadas e os erros estimados que afectam as medições, será possível elaborar sobre a sua qualidade.

Numa fase inicial serão estudadas os vários tipos de redes neuronais, para se perceber quais as estratégias usadas por cada abordagem e, deste modo, os conhecimentos sobre as redes neuronais serem alargados. Mesmo que não se tenha a intenção de usar todas as abordagens é importante ter este conhecimento, obtendo uma visão alargada das possibilidades.

Segue-se a fase de aplicação da rede neuronal sobre um ou mais conjuntos de dados e terá também uma fase de estudo. Serão estudadas duas ferramentas: o *TensorFlow* e o *PyTorch*, utilizadas para o teste de várias abordagens.

Numa 3ª fase, com as RN treinadas na fase anterior, serão feitas avaliações a cada uma delas. Será extraído um valor para cada abordagem, que representará a precisão. Com estes resultados será possível comparar abordagens e retirar conclusões. Será também possível escolher uma ou mais soluções com maior precisão e aplicá-las num ambiente real.

Esta solução irá ser aplicada numa rede de sensores sem fios e para produzir resultados serão usadas medições de uma rede num ambiente aquático, implementada no estuário do *Columbia River*, situado no nordeste dos Estados Unidos da América e mantida pela *Center for Coastal Margin Observation & Prediction* (CMOP) da *Science and Technology University Research Network*.

A plataforma resultante irá assim fornecer medições de sensores com qualidade melhorada e informação complementar sobre a validade das medições.

1.4 Estrutura do documento

Este documento segue a seguinte estrutura:

- **Capítulo 1** - Introdução
- **Capítulo 2** - Contexto

  Descreve o contexto e define os conceitos base para este trabalho.
- **Capítulo 3** - Desenvolvimento da plataforma

  Descreve a arquitectura da plataforma desenvolvida, o fluxo de dados e os passos dados até alcançar a solução final.
- **Capítulo 4** - Implementação

  Descreve a implementação da solução final.
- **Capítulo 5** - Resultados e avaliação
- **Capítulo 6** - Conclusão e trabalho futuro

Capítulo 2

Contexto

Neste capítulo serão abordados os temas base para este trabalho e alguns conceitos fundamentais. Na secção 2.1 será explicado o que é uma rede de sensores, o que é a monitorização usando uma rede de sensores, os tipos de aplicações possíveis e também os tipos de erros que podemos encontrar nas medições de uma rede. Na secção 2.2 será contextualizada a parte das Redes Neuronais referindo pontos importantes da sua história, o tipo de RNs que existem hoje em dia e também o treino de RNs. Na secção 2.3 são descritos alguns conceitos base sobre confiabilidade. Na secção 2.4 é descrito em que consiste o Projecto AQUAMON. Por fim na secção 2.5 serão sucintamente descritas soluções existentes para atingir confiabilidade em redes de sensores. A união dos temas abordados nas secções 2.1 e 2.2 é importante para atingir confiabilidade (secção 2.3) na monitorização com redes de sensores.

2.1 Monitorização com redes de sensores

Uma rede de sensores sem fios é composta por um conjunto de dispositivos autónomos, que integram sensores capazes de monitorizar diferentes parâmetros físicos. A estes dispositivos autónomos chamamos de nós inteligentes, que, numa arquitectura típica, se separam em nós sensores e em *sink nodes*. A Figura 2.1 ilustra a arquitectura típica, incluindo os dois tipos de nós e a ligação a uma rede externa.

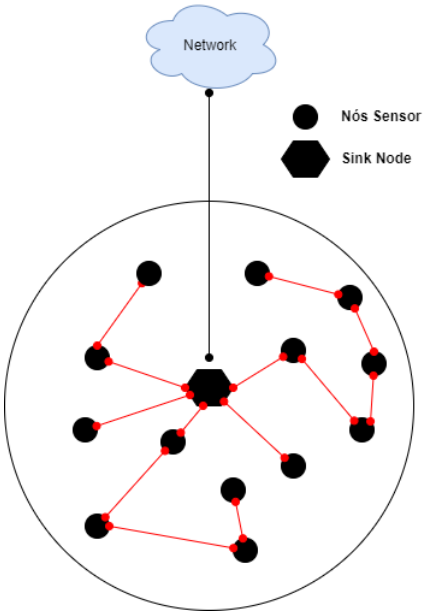

FIGURA 2.1: Representação de uma arquitectura Típica de RSSF

Os *sink nodes*, podendo haver mais que um, servem de nó central. Estes nós tratam da recepção das medições efectuadas pelos nós sensores e do envio destas para um local central onde são armazenadas e processadas.

Os nós sensores, responsáveis pela captação das medições de interesse para a RSSF, podem enviar as suas medições para o nó central através de duas vias: entre nós sensores chegando posteriormente ao nó central ou directamente ao nó central. Utilizando a primeira via, tem como vantagem aumentar a distância possível de monitorização, pois os nós sensores não necessitam de enviar directamente as suas medições para o *sink node*.

Estes nós possuem cinco componentes básicos:

- Micro-controlador - processa os dados importantes para a rede (responsável por colectar e processar os dados dos sensores, armazenar ou enviar para outros nós, executar programas necessários para a rede, entre outras funções);
- Memória - armazena programas e dados colectados. (RAM, ROM, EPPROM ou memórias flash);
- Sensores - responsáveis pela captação das medições de interesse para a RSSF;
- Rádio e Protocolo de comunicação - o rádio permite a transmissão dos dados colectados ou recebidos (também através dele) de outros nós da rede, para o *sink node* ou outros nós, sendo esta transmissão realizada por um protocolo de comunicação apropriado para o propósito da rede e da aplicação.
- Fonte de energia - em geral uma bateria. Os *sink nodes* muitas vezes possuem uma fonte de alimentação contínua pois necessitam de estar activos durante todo o funcionamento da rede.

A Monitorização com redes de sensores consiste na utilização de vários nós inteligentes dispersos numa área, que recolhem informação sobre um ou mais parâmetros físicos nessa área.

2.1.1 Áreas de aplicação

As redes de sensores sem fio podem compreender vários tipos de sensores, podendo estes fornecer medições de tipo sísmico, magnético, térmico, visual, infravermelho, radar, acústico, entre outros. Estes nós inteligentes proporcionam uma ampla gama de possíveis situações de monitorização. Os nós são usados para detecção constante de eventos e sua identificação. As aplicações de uma rede de sensores incluem, principalmente, áreas da saúde, militar, ambiental, residencial e áreas comerciais.

Algumas aplicações típicas de monitorização:

- **Monitorização Ambiental** - Neste cenário é usada uma rede de sensores para monitorizar o estado de um ecossistema. Como exemplos temos: detecção de incêndios florestais, controlo da poluição do ar, controlo de qualidade de águas, prevenção de terramotos e prevenção de cheias.
- **Monitorização Agrícola** - Nesta área pode controlar-se a produção agrícola, desde a plantação à colheita. Como exemplos nesta área temos: a monitorização de vinhas e monitorização de estufas.

Outros exemplos de aplicações usando redes de sensores são: as áreas de transporte e logística, aplicações industriais, entretenimento, segurança e vigilância, cuidados e saúde, controlo de sistemas de energia, casas inteligentes e monitorização de construção civil.

Devido ao facto de muitas destas aplicações serem críticas é necessário que o sistema de monitorização seja fiável, para evitar erros que possam contribuir para decisões erradas e alertas falsos. Mais à frente iremos verificar que nem sempre é simples aplicar este conceito nestes sistemas, considerando que muitas destas redes serão aplicadas em ambientes severos, onde o erro é menos detectável.

2.1.2 Tipos de erros e falhas

Neste trabalho a principal preocupação será a detecção de *outliers*, *drifts* e *offsets* nas medições dadas pelos sensores.

- *Outliers* - é uma medição que difere significativamente de outras medições. Um *outlier* é uma medição com erro que, se não for detetado e corrigido, pode levar à falha da aplicação ou do sistema que usa a medição.
- *Drifts* - é medida de diferença entre o valor observado de uma medição e outro valor (geralmente a média do valor dessa medição). Quando um *drift* ocorre é possível visualizar o valor da medição a subir ou a descer ao longo de um certo período de tempo. O sinal do desvio informa a sua direcção(positivo quando excede o valor observado). A magnitude do valor indica o tamanho da diferença.
- *Offsets* - são dados que estão a uma certa distancia constante do valor real. Um exemplo pode ser o processamento de dados da temperatura: a temperatura real ser 10ºC mas nos obtidos ser 15ºC, neste caso diz-se que existe um *offset* de 5ºC em relação ao valor real.
- *Noise* (Ruído) - dados com ruído ou ruidosos são dados sem sentido. O termo costuma ser usado como sinónimo para dados corrompidos. No entanto, o seu significado foi expandido para incluir dados que não podem ser entendidos e interpretados correctamente por máquinas, como texto não estruturado.

Serão criados modelos de previsão para a detecção e correcção destes erros que afectam as medições. Estas medições serão exploradas usando redes neuronais artificiais, as quais são referenciadas na próxima secção.

2.2 Redes Neuronais

Desde o século 20 que investigadores e engenheiros de diversas áreas tentam criar um sistema que replique o modo de funcionamento do cérebro humano. O surgimento das Redes Neuronais (RN), por parte de Warren McCulloch e Walter Pitts em 1943 [12], causou um grande avanço nesta área.

As RN apresentam as seguintes características:

- várias unidades de processamento,
- existem "pesos" associados à ligação entre as unidades de processamento,
- a aprendizagem é conseguida pelo ajuste dos "pesos".

Tal como no cérebro humano, o poder de processamento nas RN situa-se no neurónio. Contudo existem diferenças significativas entre ambos, como se pode ver na tabela 2.1.

Cérebro Humano	**Rede neuronal**
10^{11} neurónios com 10^{14} sinapses	Processador único com circuitos complexos
Processamento Não Linear	Processamento Linear
Processamento Distribuído	Processamento Central
Processamento Paralelo	Processamento Sequencial

Tabela 2.1: Comparação entre cérebro humano e rede neuronal [13]

O cérebro executa inúmeras tarefas com facilidade, mas no fundo são as unidades de processamento (neurónios) interligadas que conferem esta habilidade. Nas redes neuronais artificiais isto também não foi esquecido.

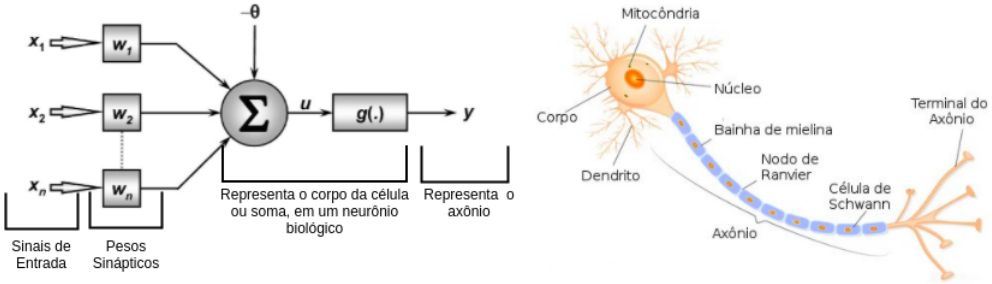

Figura 2.2: Neurónio Artificial *vs* Neurónio Biológico [11]

Num neurónio humano são recebidos estímulos provenientes do ambiente ou de órgãos sensoriais. Comportando-se de maneira semelhante, o neurónio artificial recebe dados de entrada em forma de vector. As dendrites no neurónio humano recebem os sinais de entrada e encarregam-se de levar esses sinais para o local correcto. Nos neurónios artificiais acontece um processo análogo: recebem pesos e direccionam a respectiva entrada modificando-a de forma adequada. A agregação de todos estes pesos é realizada pela função soma no neurónio artificial, de forma similar ao que acontece no corpo celular do neurónio humano. Esta comparação pode ser visualizada na figura 2.2.

2.2.1 História

Muitos cientistas, matemáticos e engenheiros tentaram aplicar o conceito de rede Neuronal no mundo da computação mas tudo acelerou com o lançamento do artigo [12] escrito pelo psiquiatra Warren McCulloch e o matemático Walter Pitts no ano 1943. Neste artigo os autores fizeram uma analogia entre células nervosas e o processo electrónico, simulando o comportamento de neurónios humanos. Nesta analogia o neurónio apenas tinha uma saída, que era a soma de todas as suas entradas. Na figura 2.3 são descritas sucintamente algumas das mais importantes conquistas na área de redes neuronais. (adaptação [15])

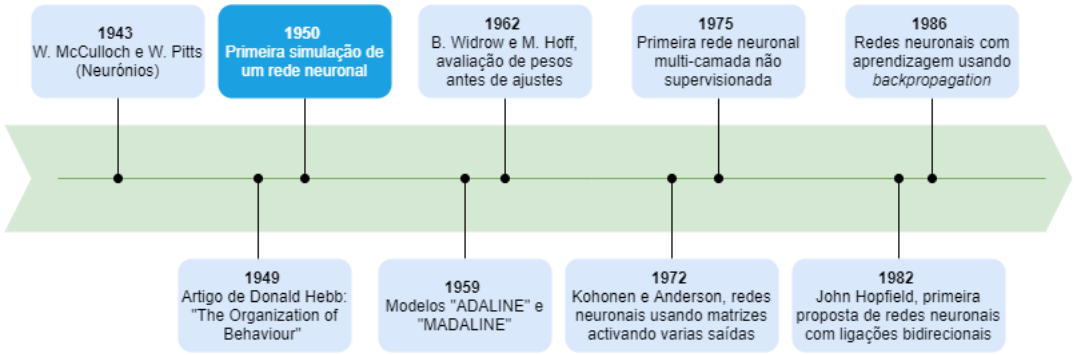

FIGURA 2.3: Conquistas mais importantes na área de redes neuronais

- 1949 - Donald Hebb escreve um artigo [14] sobre as ligações neuronais se fortalecerem sempre que são utilizadas.
- 1950 - primeira simulação de uma rede neuronal.
- 1962 - desenvolvido um processo de avaliação dos valores dos pesos antes de os ajustar.
- 1972 - Kohonen e Anderson desenvolveram duas RN similares que usavam matrizes matemáticas. Com isto, era possível activar vários outputs em vez de apenas um.
- 1975 - primeira rede neuronal multi-camada.
- 1982 - John Hopfield apresenta a sua ideia: ligações bidireccionais entre neurónios.
- 1986 - aparecimento do modo de treino *backpropagation* [5].

2.2.2 Perceptrão

O perceptrão é um tipo de neurónio criado em 1957 por Frank Rozenblatt [4], podendo afirmar-se que é o tipo de rede neuronal *feedfoward* mais simples.

O perceptrão é um classificador binário que mapeia uma entrada, vector $\boldsymbol{x}$, para uma saída $\varphi(\boldsymbol{x})$.

$$\varphi(x) = \begin{cases} 1 & \text{if } \boldsymbol{x} \cdot \boldsymbol{w} + b > 0 \\ 0 & \text{else} \end{cases}$$

Na formula acima apresentada, w é um vector de peso real, $\boldsymbol{x} \cdot \boldsymbol{w}$ é o produto escalar $\sum_{i=1}^{n} \boldsymbol{w}_i \boldsymbol{x}_i$ onde n é o número de entradas, e b é um *bias*, um valor constante que não depende dos valores de entrada e faz com que a barreira de decisão varie.

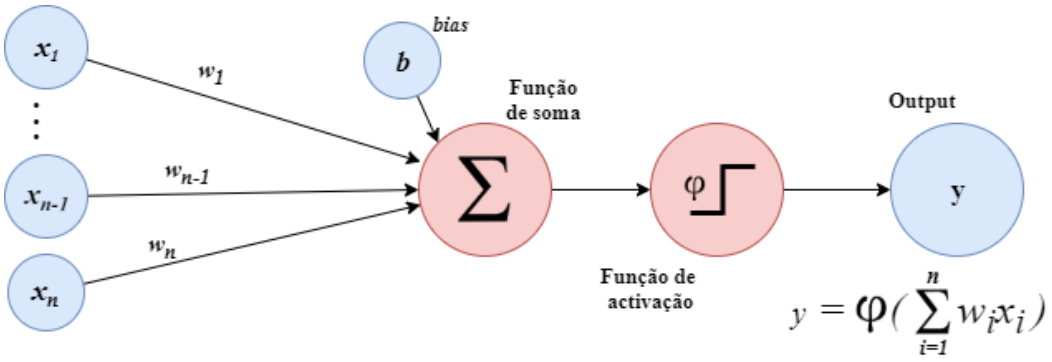

FIGURA 2.4: Perceptrão

Todas as variações de perceptrões consistem em: conjunto de entradas ($x = x_1, ..., x_n$), uma função de soma, uma função de activação e uma saída. Na figura 2.4 é representado um perceptrão e o seu fluxo.

2.2.3 Funções de activação

Funções de activação são usadas para determinar o output de um neurónio. Estas funções mapeiam os resultados, por exemplo, entre 0 e 1 ou -1 e 1, dependendo da função. Estas funções dividem-se em 2 tipos: lineares e não lineares.

Apenas serão realçadas funções não lineares pois as funções lineares não ajudam com a complexidade dos parâmetros e não linearidade dos dados que geralmente são dados às redes neuronais.

Sigmoide

Como se pode ver na figura 2.5 a função Sigmoide tem a forma de um "S"e os limites 0 e 1.

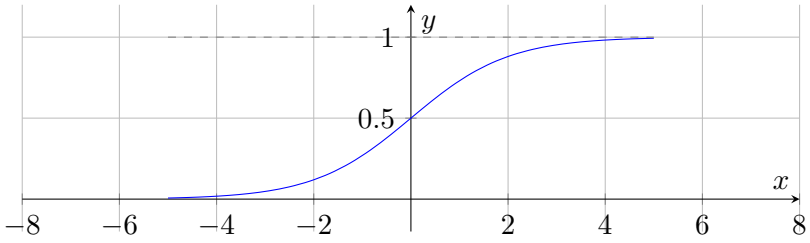

FIGURA 2.5: Função Sigmoide

Equação: $f(x) = \frac{1}{1+e^{-x}}$ **Limites:** $]0, 1[$.

A principal razão pelo qual se usa esta função, como função de activação, são os seus limites (0 e 1). É especialmente usada em modelos onde o output é uma probabilidade, pois apenas varia entre 0 e 1.

O uso desta função foi motivado desde muito cedo pela analogia de os neurónios estarem apenas desligados ou ligados (0 ou 1). Existe também um variação da sigmoig, chamada de função tangente hiperbólica (Tanh), tendo também a forma de um "S"mas os seus limites são -1 e 1. Hoje em dia, para redes de aprendizagem profunda, a função sigmoid e tangente hiperbólica não são tão utilizadas como antigamente, devido ao "aparecimento"da função *Rectified Linear Unit* (ReLU) e a sua capacidade de aumentar a velocidade de aprendizagem das redes neuronais.

ReLU

A função ReLU (*Rectified Linear Unit*) (Fig. 2.6) "surgiu"com a necessidade de encontrar uma função de activação que proporcionasse uma melhor aprendizagem.

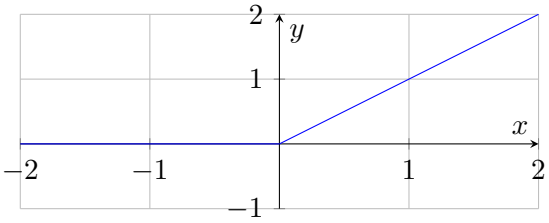

FIGURA 2.6: Função ReLU

$$f(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{else} \end{cases} \quad \textbf{Limites: } [0, +\infty[.$$

Esta é actualmente a função de activação mais utilizada no mundo, sendo usada em quase todas as *Convolutional Neural Networks* e redes de aprendizagem profunda.

2.2.4 Classificação de redes neuronais

As redes neuronais podem ser classificadas entre outros aspectos, pela forma de aprendizagem e pelo tipo de arquitectura que esta contém.

Para o modo de aprendizagem existem os seguintes tipos:

- **Aprendizagem supervisionada** - a rede tem noção dos resultados que são supostos aparecer. Caso o resultado obtido pela rede seja semelhante aos valores desejados a rede acaba o treino, caso não sejam, esta rede procede à alteração dos pesos das conexões entre neurónios com base na diferença entre os valores obtidos e os valores desejados.
- **Aprendizagem não supervisionada** - a rede adapta-se baseando-se apenas nas regras definidas para o seu comportamento. Ao contrário do método anterior as redes não conhecem o output desejado.

Em relação ao tipo de arquitectura as redes podem ser classificadas em:

- **Redes totalmente conectadas** - nesta arquitectura todos os neurónios de uma camada, independentemente da camada onde se situam, estão conectados a cada neurónio da camada seguinte.
- **Redes parcialmente conectadas** - nesta arquitectura os neurónios de uma camada, independentemente da camada onde se situam, estão parcialmente conectados a neurónios da camada seguinte, isto é, podem estar conectados a um, vários ou mesmos a todos os neurónios da camada seguinte, podendo variar de neurónio para neurónio.
- **Redes de camada única** - redes compostas apenas por uma camada de entrada e uma camada de saída.
- **Redes de multicamada** - compostas por uma camada de entrada, uma ou mais camadas ocultas e uma camada de saída.

- **Redes *feedforward*** - redes sem realimentação têm neurónios agrupados em camadas. O sinal percorre a rede em uma única direcção, da entrada para a saída. Os neurónios da mesma camada não são conectados uns aos outros.

- **Redes recorrentes** - nas redes com realimentação ou recorrentes (*recurrent*), a saída de alguns neurónios alimentam neurónios da mesma camada (inclusive o próprio) ou de camadas anteriores. O sinal percorre a rede em duas direcções, têm memória dinâmica e capacidade de representar estados em sistemas dinâmicos.

Neste trabalho serão utilizadas redes neuronais *Multilayer Perceptron* (MLP), redes multicamada, completamente ligadas, de metodologia *feedforward* e com aprendizagem supervisionada. A escolha destas para este trabalho deve-se aos estudos já realizados utilizando estas redes, no contexto de redes de sensores e do projecto AQUAMON em [2].

2.2.5 Redes MLP

As redes MLP são redes supervisionadas de multicamada, dependendo do problema pode ou não necessitar de uma quantidade significativa de treino. O *Perceptron*, conforme foi descrito na secção 2.2.2, é um algoritmo destinado a realizar uma classificação binária; isto é, prevê se a entrada pertence a uma categoria ou não. Usando outra função de activação pode também calcular um valor escalar em vez de um valor binário.

Uma MLP é uma rede neuronal composta por vários *Perceptrons*, por uma camada de entrada, uma camada de saída e entre estas um número arbitrário de camadas ocultas. Estas camadas ocultas contém o verdadeiro mecanismo computacional das MLP. Esta descrição pode ser visualizada na figura 2.7.

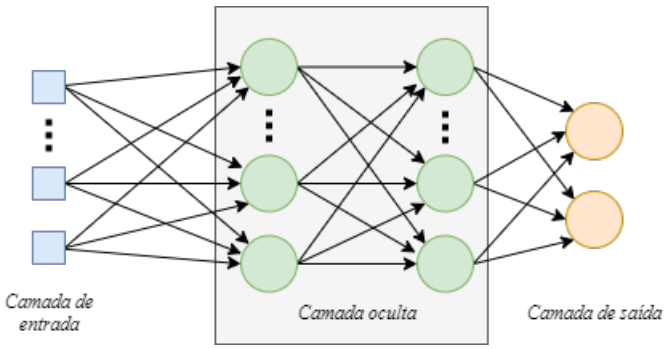

FIGURA 2.7: Representação de uma MLP

As MLP são frequentemente aplicadas a problemas com aprendizagem supervisionada, isto é, treinam sobre um conjunto de pares entrada-saídas e aprendem a modelar a relação entre entradas e saídas.

O treino nas MLP consiste no ajuste dos pesos e viés com objectivo de minimizar o erro, para isto é usado o algoritmo de *backpropagation*. Com este algoritmo é possível ajustar pesos e viés em relação a erros de classificação ou aproximação.

2.2.6 Treino (*Backpropagation*)

Redes neuronais supervisionadas requerem um grande numero de dados de treino, incluindo a saída esperada. Para cada entrada de treino é construído, pela rede, uma saída. Esta saída é então comparada com o valor esperado e é calculado a previsão

de erro ($esperado - obtido$). Para medir o quão bem a saída da rede se encaixa nos valores esperados é normalmente usado o erro quadrático médio (EQM):

$$EQM = \sum_{dados} \sum_{\text{saídas}} (esperado - calculado)^2$$

onde o erro quadrático é calculado com base em todas as saídas e todos os dados de treino. O problema consiste em construir um modelo de pesos e *biases* que irá minimizar o EQM. Desta maneira os pesos são análogos aos parâmetros de regressão. Os valores "correctos" para os pesos são desconhecidos, a tarefa é estimar estes valores. Mas devido à natureza não-linear das funções de activação (Sigmoid e ReLU) não existe nenhuma solução exacta para isto. Este processo é feito com o auxilio do método do gradiente descendente.

Ao percorrer o gráfico de um intervalo dos pesos versus o EQM (figura 2.8) e aplicando-lhe a derivada em cada ponto multiplicada por um variável chamada taxa de aprendizagem (a qual será explicada mais à frente), o método do gradiente descendente pretende encontrar o ponto do gráfico onde a derivada é zero ou próximo deste. Desta forma consegue-se encontrar o valor do peso para o qual a RN se deve deslocar para que o EQM seja mínimo, aplicando este peso na sua respectiva conexão. Este processo é feito pela rede em cada uma das suas conexões. Na figura 2.8 é apresentado um exemplo do EQM em cada peso para um dada conexão, o objectivo do treino é chegar o mais próximo possível do valor mínimo (min). Valores perto da derivada 1 terão sinal positivo e valores perto da derivada 2 terão sinal negativo, estes sinais são a direcção para a qual se deve ajustar o peso.

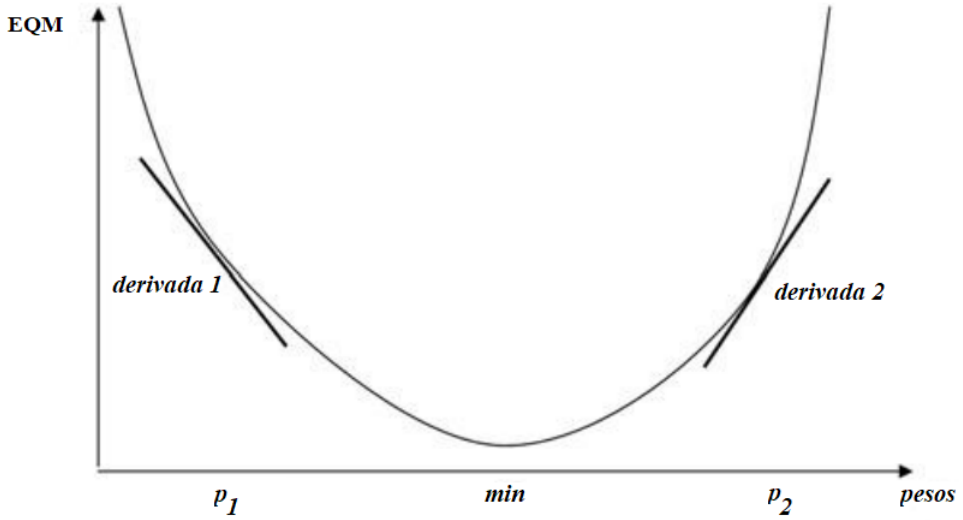

FIGURA 2.8: Exemplo de correlação do intervalo de um peso versus EQM [18]

O processo de aprendizagem por parte das redes neuronais é normalmente um processo lento, por isso é necessário definir alguns critérios de paragem:

- o número de iterações do algoritmo;
- um erro máximo que pode ser atingido;
- tempo limite de treino;
- combinação dos critérios anteriores.

Definir estes critérios nem sempre é suficiente, o programador pode ser demasiado restrito ou ambicioso e consequentemente a rede pode não aprender da maneira correcta. Pode acontecer o que é chamado de **overtraining**, isto é, a rede constrói uma maneira de reagir aos dados que lhe são apresentados como dados de treino e apenas consegue classificar com qualidade estes. Ao alimentar a rede com dados que lhe são desconhecidos, esta não tem capacidade de os classificar correctamente.

O treino de redes neuronais segue normalmente a seguinte ordem:

- Guardar uma percentagem, normalmente 30%, dos dados de treino para validação da rede - *test set*;
- O restante é usado para treinar a rede - *training set*;
- Quando treinada a rede irá ser validada com o *test set*;

O treino e validação são repetidos inúmeras vezes, atribuindo em todas as iterações do treino novos pesos à rede obtidos através do método de gradiente descendente. De seguida esta rede é comparada com a "melhor"rede até agora calculada (rede com menor EQM), guardando sempre a que contém menor EQM como a "melhor". Isto faz com que o EQM vá diminuindo, isto é, vamos encontrando melhores pesos para as conexões da rede.

Esta abordagem pode levar a que a rede pare num mínimo local e não num mínimo global. Em teoria isto não é um problema insuperável:

- Por exemplo, é possível treinar varias redes neuronais, cada uma com diferentes pesos iniciais. Por fim escolher a rede que melhor aproxima.
- Outra hipótese é o *online* ou *stochastic backpropagation*, neste método é introduzido um elemento aleatório na função de gradiente descendente, evitando ficar preso num mínimo local.
- Em alternativa existe o uso da variável de Inércia ou *Momentum Rate*, explicado na subsecção seguinte.

Taxa de aprendizagem (*Learning Rate - LR)*

A LR é um valor entre 0 e 1. É multiplicada pela correcção ao erro obtida na iteração actual e adicionada ao peso da iteração anterior, possibilitando assim localizar o mínimo global. Contudo existe um problema, se o valor de LR for muito alto, podemos "ultrapassar"o mínimo global, sem que assim seja possível encontrar a solução óptima. Se o valor for muito baixo a variação dos pesos é muito lenta e encontrar o mínimo global poderá tornar-se impossível. Uma solução é variar a LR ao longo do treino: no inicio conter um valor alto para chegar à vizinhança do mínimo global rapidamente. Depois este valor deve ser diminuído para evitar "ultrapassar"o valor óptimo.

Inércia (*Momentum Rate - MT)*

O algoritmo de *backpropagation* é melhorado usando a variável Inércia. Essencialmente a variável inércia irá influenciar o ajuste do peso actual a mover-se na mesma direcção que os últimos ajustes, ao contrário da LR que apenas afecta a modificação ao peso do momento. Tal como a LR, usando altos valores podemos "ultrapassar"o mínimo global, e baixos valores podemos nem alcançá-lo. Na figura 2.9 podemos verificar uma analogia da variável inércia para uma melhor compreensão. A bola com massa pequena, representando uma inércia pequena, não passa do mínimo local

correspondente ao ponto "A". A bola grande, que representa uma inércia grande, ganha grande velocidade e ultrapassa o ponto "B", o mínimo global.

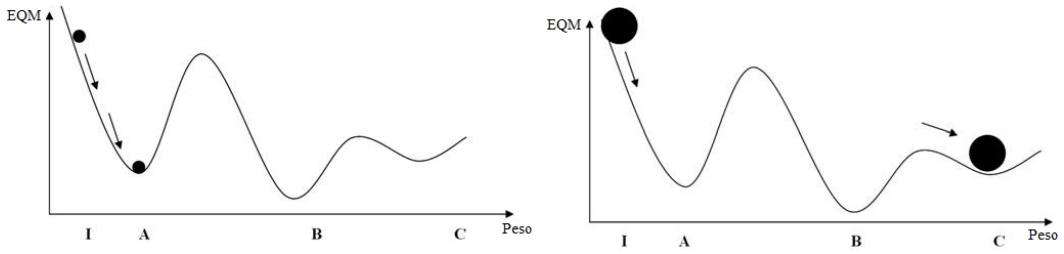

FIGURA 2.9: Representação de inércia baixa e alta [18]

Portanto é necessário considerar cuidadosamente quais os valores definir para a taxa de aprendizagem e de inércia. Experimentar vários valores entre ambos é muitas vezes necessário antes de se conseguir obter os melhores resultados. (adaptação [18])

2.2.7 Tecnologias

Tensor Flow e PyTorch

Para este trabalho irão ser exploradas duas ferramentas. O TensorFlow é uma plataforma de código aberto *end-to-end* para aprendizagem automática. Possui um ecossistema abrangente e flexível de ferramentas, bibliotecas e recursos da comunidade que permite um desenvolvimento mais moderno e faz com que os programadores construam e implantem aplicações mais facilmente com *Machine Learning*. Tal como TensorFlow, o PyTorch oferece uma grande gama de ferramentas: para a criação de redes neuronais e também para criação de modelos estatísticos.

2.3 Confiabilidade

Confiabilidade, de uma maneira geral, é uma medida da disponibilidade, confiança e manutenção de um sistema, desempenho do suporte de manutenção e, em alguns casos, outras características como durabilidade, segurança e protecção. Em engenharia de software, confiabilidade é a capacidade de fornecer serviços que podem ser defensivamente confiáveis num período de tempo [3].

Avizienis definiu 6 atributos de confiabilidade em [1], sendo quatro destes:

- **disponibilidade**: prontidão para o serviço correcto;
- **fiabilidade**: continuidade do serviço correcto;
- **integridade**: ausência de alterações não autorizadas à informação;
- **reparabilidade**: capacidade de se realizar reparações e modificações.

As duas restantes classificam-se como atributos de segurança para confiabilidade:

- **segurança crítica**: ausência de consequências catastróficas no(s) utilizador(es) e no meio ambiente;
- **confidencialidade**: ausência de acesso não autorizado à informação;

Dependendo do destino das aplicações, o ênfase em certos atributos pode diferir, quando falamos de confiabilidade em RSSF, mais especificamente neste trabalho, existe uma maior concentração na disponibilidade, confiabilidade e integridade.

Avizienis [1] fala ainda das maneiras de atender a esta necessidade de confiabilidade, utilizando meios para tratar as faltas latentes nos sistemas e que podem levar à sua falha:

- **prevenção de faltas**: como evitar a ocorrência ou introdução de faltas;
- **tolerância a faltas**: como evitar que o sistema falhe apesar de ocorrerem faltas;
- **remoção de faltas**: como reduzir o número ou a gravidade das faltas;
- **previsão de faltas**: como estimar o número actual, a incidência futura e as prováveis consequências das faltas.

Em redes de sensores sem fios é importante perceber que faltas podem ocorrer e que efeitos produzem. Em muitos casos, estas faltas geram erros nos valores, erros estes que podem ser detectados e mitigados, antes de serem usados numa determinada aplicação.

2.4 Projecto AQUAMON

No projecto AQUAMON será desenvolvida uma plataforma para monitorização confiável em ambientes aquáticos utilizando redes de sensores sem fios, respondendo a alguns dos problemas levantados pela operação nestes ambientes. Em concreto, serão endereçados os problemas de qualidade na transmissão causados por ondas e características de propagação sobre uma superfície aquática, de previsibilidade da comunicação, causados pela contenção no acesso de vários nós ao meio de transmissão, e de qualidade dos dados, causados por faltas que afectam quer os próprios sensores quer a comunicação, originando erros nos valores ou ausência de informação.

O trabalho desenvolvido nesta dissertação enquadra-se no projecto AQUAMON, na parte de processamento de dados. Neste projecto serão combinadas diversas técnicas que, em conjunto, permitirão melhorar a qualidade da monitorização. Por um lado, serão detectadas faltas com base em técnicas de análise e processamento de sinais e em técnicas de inferência baseadas em série de dados. Por outro, a redundância oferecida por múltiplos sensores permitirá aplicar técnicas de fusão de dados e caracterizar a qualidade dos dados resultantes. Finalmente, os dados serão ainda analisados à luz de modelos de previsão do comportamento específicos para os ambientes monitorizados, que funcionarão como um elemento adicional de redundância e de garantia de qualidade.

A estrutura resultante fornecerá, portanto, dados do sensor com melhor qualidade e informações complementares sobre esta validade de dados. Uma perspectiva geral do quadro previsto é fornecida na figura 2.10.

Para demonstrar a validade das soluções desenvolvidas, será feita uma aplicação ao sistema de monitorização e previsão da hidrodinâmica e da qualidade da água da baía do Seixal, no estuário do Tejo.

Esta dissertação irá fornecer, no contexto do AQUAMON, a plataforma de tratamento das medições dos sensores, focando-se essencialmente na implementação desta plataforma. Serão implementadas técnicas já desenvolvidas anteriormente, mas, também explorados outros caminhos. Serão estudados diversos tipos de implementações e feita a comparação de diversas técnicas para tentar introduzir melhorias face a técnicas já desenvolvidas.

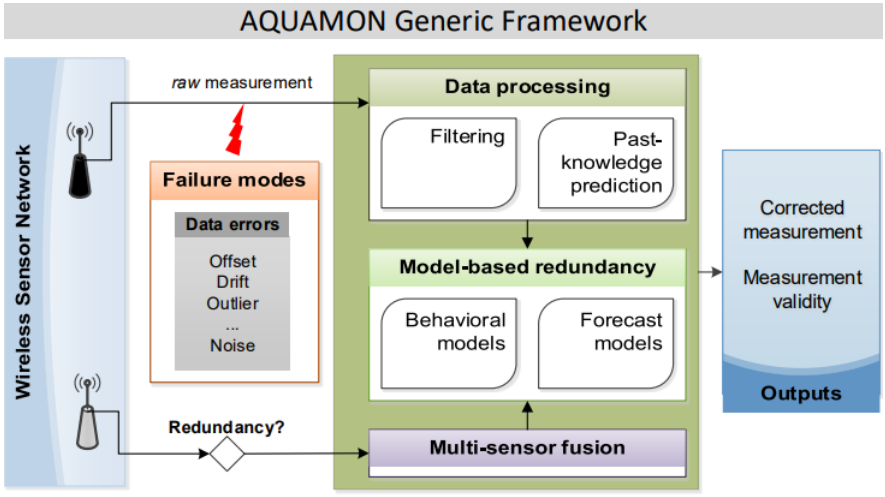

FIGURA 2.10: Arquitectura genérica AQUAMON

2.5 Soluções existentes

Na ultima década foram realizados vários estudos sobre o uso de Aprendizagem Automática e técnicas de fusão de dados para a detecção e classificação de fenómenos, como por exemplo a detecção de falhas.

Em [2] os autores propõem uma solução baseada em técnicas de fusão de medições de sensores usando Aprendizagem Automática. Cada sensor é explorado em termos espaciais e temporais, recorrendo a medições anteriores. Usam medições de sensores correlacionadas que simulam a dinâmica do sistema de monitorização, funcionando como uma rede virtual de sensores correlacionados, fornecendo medições estimadas. Para calcular estas previsões os autores usaram redes neuronais. Esta solução foi aplicada a um dataset de uma RSSF de um estuário, onde depois mostraram a eficácia e qualidade desta para detecção de *outliers*. Este dataset continha medições como: temperatura da água, salinidade, composição química da água, entre outras.

Em [16] os autores propõem uma solução usando fusão de sensores para detectar acontecimentos numa RSSF através de uma técnica de classificação implementada no *sink node*. Este classificador pode ser aplicado usando Redes Neuronais Artificiais ou um algoritmo *Naïve Bayes*. Apresentando ambas baixa complexidade computacional, permitindo assim uma detecção em tempo real dos acontecimentos numa rede de sensores sem fios. Os autores aplicam esta solução num dataset de alarme de fogo, com algum ruído e algumas falhas de sinal.

Em [17], o autor apresenta comparações de desempenho entre as mais conhecidas abordagens de detecção de outliers, comparando as forças, fraquezas e a complexidade computacional de cada abordagem. Esta comparação conclui que as abordagens que usam técnicas *Support Vector Machines* (SVM) tinham um desempenho inferior em relação ao resto. Este estudo não é totalmente aceite pois em [7] o autor compara uma técnica baseada em *Principal Component Analysis* (PCA) apelidada de *Principal Component Classifier based Anomaly Detection* (PCCAD) com outras duas abordagens SVM (H-OCSVM [8, 9, 10] e QS-OCSVM [8, 9, 10]) e conclui que a abordagem PCCAD tem um desempenho superior em relação às restantes.

2.6 Resumo e trabalho seguinte

Redes de sensores estão susceptíveis a faltas, principalmente em ambientes severos. Perante este problema é clara a necessidade da implementação de uma plataforma que processe as medições da rede de sensores e que tenha a capacidade de identificar faltas, erros e também, se possível, corrigi-los da maneira apropriada. A fase seguinte consistirá na implementação desta plataforma. Para isto haverá uma continuação do estudo das redes neuronais MLP, pois requerem um grande conhecimento na área de aprendizagem automática. Serão usadas, para o seu estudo e desenvolvimento, as ferramentas *TensorFlow* e *PyTorch*.

Nesta fase, haverão várias etapas, sendo a primeira concentrada no estudo aprofundado dos casos de uso das redes MLP, isto é, estudar qual a forma apropriada da construção da rede neuronal (pesos, *biases*, número de camadas e número de neurónios) e qual a melhor forma para o seu treino (LR, MT, dados de treino e validação), dependendo do tipo de dados que lhe serão injectados. Numa fase seguinte, serão organizados os dados recebidos das redes de sensores, de forma a que haja a transformação destes para que seja possível injectá-los na rede neuronal.

Com a colaboração de Gonçalo Jesus, no contexto do projecto AQUAMON, será implementada a plataforma de processamento de dados que será baseada na arquitectura desenvolvida anteriormente [2].

A implementação desta plataforma será concretizada utilizando a linguagem *Python*, usando ambas as ferramentas já acima mencionadas. A escolha final da ferramenta a ser utilizada irá depender das conclusões e resultados obtidos a partir destas.

Capítulo 3

Desenvolvimento da plataforma

Neste capítulo é proposta uma plataforma de tratamento de dados em tempo-real para redes de sensores, aplicando técnicas de aprendizagem automática para criar previsões, detectar falhas, medir a qualidade das medições e, quando possível, corrigi-las.

No desenvolvimento desta plataforma foi usado como base a solução ANNODE [2], e neste capítulo serão citadas algumas passagens deste trabalho. Esta solução (ANNODE) já encontra e corrige *outliers* sobre um conjunto de dados, mas em ambiente *off-line*. Os objectivos deste trabalho consistem na implementação desta lógica numa ferramenta *on-line* que consiga calcular previsões e corrigir erros em tempo real, criando um programa mais independente e ao mesmo tempo tentando melhorar o desempenho desta metodologia. Para tal foi necessário modificar esta metodologia com o propósito de criar métodos mais generalistas que funcionem em ambos os cenários, *off-line* e *on-line*.

3.1 Arquitectura genérica da plataforma

Para detectar erros nos dados e caracterizar a sua qualidade é necessário comparar cada medição recebida de um sensor com vários valores correspondentes a previsões do que deveria ser uma medição correta, produzidas por redes neuronais que usam dados temporalmente e espacialmente correlacionados com as medições em análise. Por isso é necessário ter várias redes, cada uma correlacionando dados de diferentes formas (como será explicado adiante) e treinar todas estas redes antes do sistema ser usado para analisar dados em tempo-real. Para isto, é também necessário preparar dados de treino previamente escolhidos.

Os valores previstos por estas redes, actualizados para cada nova medição que é analisada, servem também para determinar a qualidade desta medição e verificar se poderá ter sido afectada por uma falha do sensor ou ser um *outlier*.

Na utilização dos modelos em tempo-real existe outro requisito: criar previsões a partir dos modelos, sendo necessários dados de entrada específicos, a que chamamos de vectores de entrada. Assim, na plataforma *on-line*, existe também um processamento de dados anterior ao cálculo de previsões, para gerar estes vectores.

Por fim, temos o módulo a que chamamos de Servidor, que está encarregue de receber as medições dos sensores e de as passar para o módulo de análise.

Com isto faz sentido separar ambos estes processos em dois blocos gerais: **Treino de modelos** (*off-line*) e **Utilização dos modelos em tempo-real** (*on-line*), pois são executados separadamente.

Este blocos podem ser visualizados na figura 3.1 e serão explorados mais aprofundadamente nas subsecções seguintes.

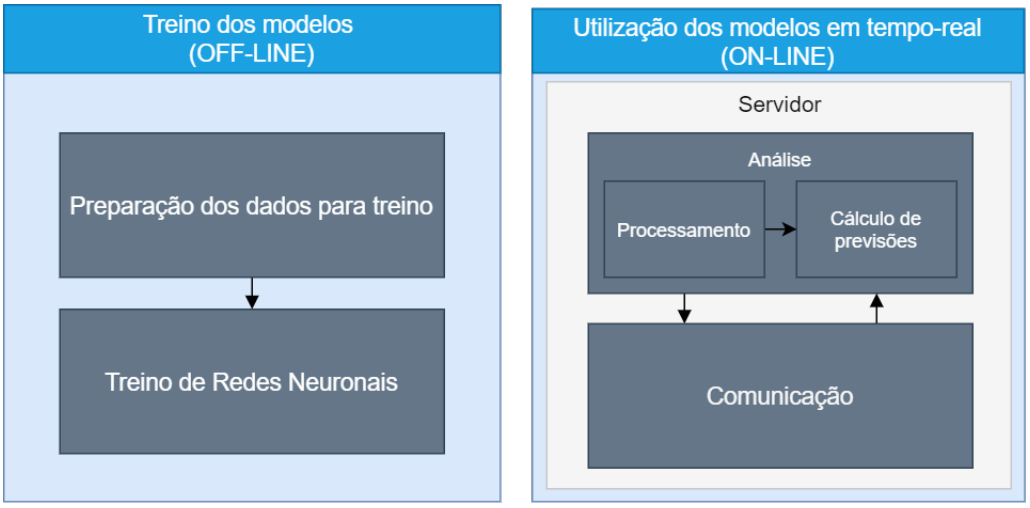

FIGURA 3.1: Blocos Gerais

3.2 Preparação de dados e treino

3.2.1 Alinhamento de dados e construção de vectores de entrada

São recebidas medições de vários sensores de forma periódica cada uma com uma estampilha temporal associada, no entanto estas não são recebidas em simultâneo. Podem existir falhas, como dados em falta, e em maior parte dos casos não estão alinhadas temporalmente.

Para lidar com este problema é construída uma estrutura, denominada de *new_times*. Foi dado este nome pois é uma aglomeração das medições de todos os sensores em relação a um sensor alvo, para o qual queremos criar vectores de entrada, usando as estampilhas **temporais** (*Timestamps*). Isto é, para cada medição do sensor alvo é ligada uma medição dos sensores vizinhos, sendo esta a mais próxima, evitando ao mesmo tempo ligações repetidas (como pode ser visualizado na figura 3.2, usando medições de temperatura). No treino dos modelos e na produção de previsões são usados vectores de entrada com medições passadas, sendo estes vectores provenientes da estrutura *new_times*.

Para cada sensor é associada uma lista de medições: estas listas apenas contêm as medições de cada sensor de forma ordenada temporalmente, sendo a construção da estrutura *new_times* feita a partir destas listas. Para o treino das redes neuronais, estas listas já estão prontas a serem processadas, mas no caso de previsões em tempo real é necessário, primeiro, acumular medições nestas listas, até terem um número mínimo necessário de medições. Estas listas estão representadas na figura 3.2.

Na figura 3.2 , ao analisar o Vizinho 2 conseguimos ver que os dados estão perfeitamente alinhados temporalmente e temos uma correlação fácil de construir onde não existem faltas de dados nem desalinhamentos temporais (Exemplo Perfeito), o que por vezes não acontece. No "Exemplo Realista"é-nos mostrado o que ocorre com alguma frequência: medições com desalinhamentos temporais e faltas. É para este último caso que a construção do *new_times* é mais importante. Ao correlacionar os dados existem duas regras: começar no mínimo comum. Para um exemplo de 4 sensores, o sensor alvo começa as medições às 09:00 e os vizinhos às 09:10, 09:15 e 09:30. A construção do *new_times* em relação ao sensor alvo só pode começar a partir do tempo 09:30, pois só a partir desse tempo é que existem medições em comum em

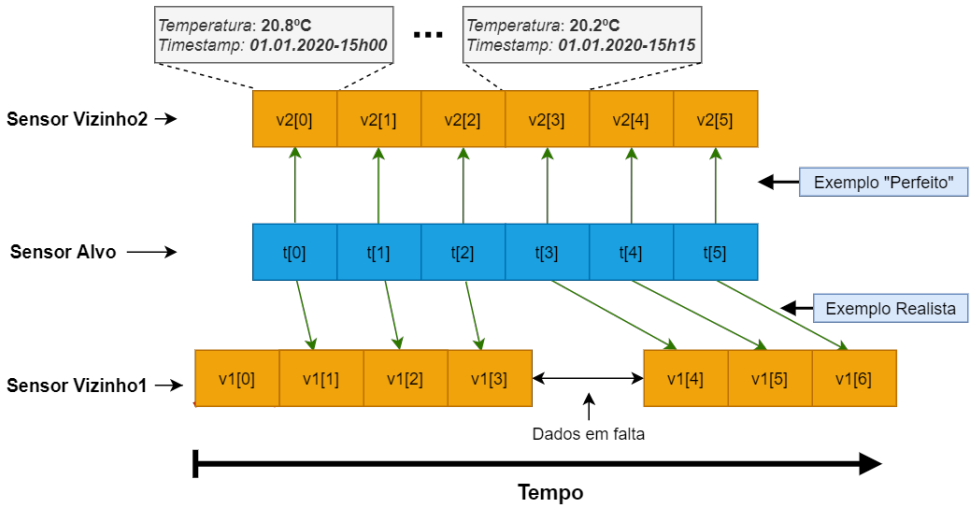

FIGURA 3.2: Construção do *new_times*.

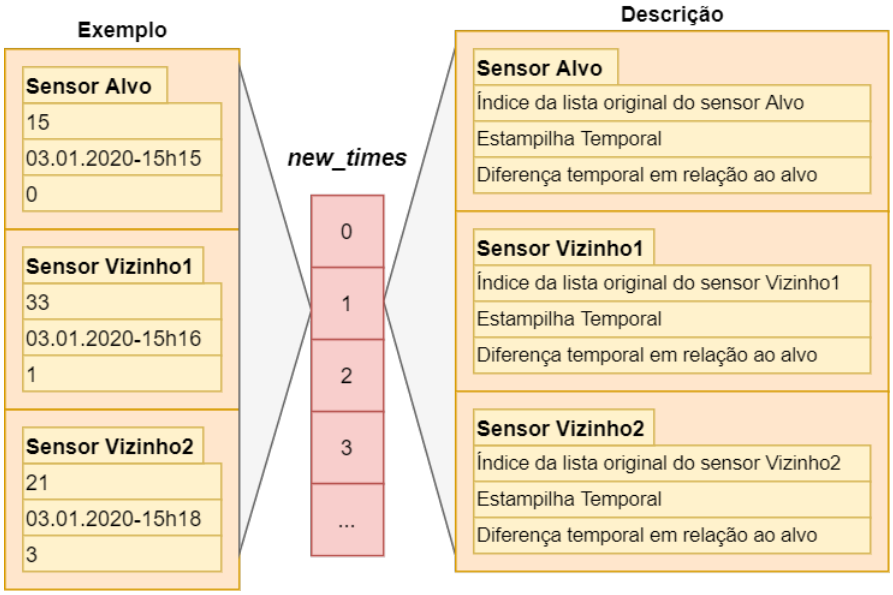

FIGURA 3.3: Estrutura final do *new_times*.

todos os sensores. A segunda regra consiste na não repetição de valores: por exemplo, se o $t0$ do sensor alvo se relaciona com $t1$ do sensor vizinho 1 mais nenhuma medição do sensor alvo se pode relacionar com o $t1$ do sensor vizinho 1. Na figura 3.3 é apresentado o aspecto final do *new_times*. Cada posição desta estrutura contém dados correlacionados com o sensor alvo. Olhando para a descrição podemos verificar que esta estrutura guarda a estampilha temporal da medição, a diferença temporal entre essa medição e a medição do sensor alvo e o índice da lista original do sensor. É possível verificar, também, que não é guardado nesta estrutura a métrica que o sensor mede, mas sim o índice da lista original. Para obter o valor lido pelo sensor é apenas necessário aceder à lista original utilizando como índice o valor guardado no *new_times*. Por outras palavras esta estrutura não guarda os dados das medições dos sensores apenas guarda uma referência para esses dados.

Com estes três dados é possível, então, a criação de vectores de entrada. A criação destes vectores é sempre feita em relação ao sensor alvo, para o qual queremos fazer uma previsão. Seguindo o trabalho desenvolvido anteriormente [2] a criação usa,

no mínimo 60 e no máximo 120 medições, isto, num universo de apenas 4 sensores. Estas primeiras 60, correspondem a medições anteriores do sensor alvo, as restantes correspondem a medições anteriores dos sensores vizinhos, tendo cada uma destas 20 medições associadas ao vector.

A figura 3.4 representa, de maneira simplificada a construção de um vector de entrada.

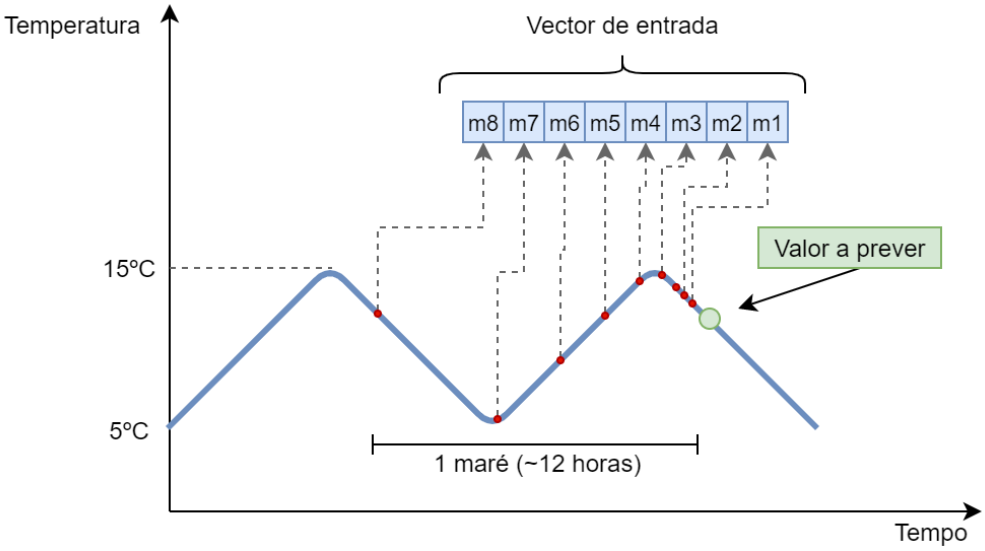

FIGURA 3.4: Exemplo de construção de um vector de entrada utilizando apenas o sensor alvo.

Estas medições são as *features* que irão determinar a saída das redes neuronais sendo utilizadas medições anteriores do sensor alvo e dos sensores vizinhos para explorar possíveis correlações entre diferentes sensores. Assim, é possível capturar mais eficazmente a informação do ambiente e distinguir erros de sensores com eventos relacionados com o local onde estão instalados.

Isto é, segundo a metodologia ANNODE existem 3 tipos de modelos de redes neuronais: um que irá modelar o comportamento de apenas o sensor alvo, outro irá modelar o comportamento de todos os sensores e por fim um que irá modelar apenas o comportamento dos sensores vizinhos.

O primeiro modelo utiliza 60 medições do sensor alvo, o segundo utiliza 60 medições do sensor alvo mais 20 de cada sensor vizinho, dando um total de 120 medições (60+(20x3)). Por fim, o último modelo utiliza medições apenas dos sensores vizinhos, 20 medições de cada um, dando um total de 60 medições (20x3). Em síntese:

- Modelo 1 - **Entrada**: Sensor alvo; **Saída**: Sensor alvo.
- Modelo 2 - **Entrada**: Sensor alvo e vizinhos; **Saída**: Sensor alvo.
- Modelo 3 - **Entrada**: Sensores vizinhos; **Saída**: Sensor alvo.

	Modelo 1		**Modelo 2**		**Modelo 3**	
Sensores	*Alvo*	*Vizinhos*	*Alvo*	*Vizinhos*	*Alvo*	*Vizinhos*
Medições	60	0	60	60 (20x3)	0	60 (20x3)
Total medições	60		120		60	

TABELA 3.1: Tipo de modelos utilizados e a composição dos respectivos vectores de entrada

O vector de entrada é construído usando medições anteriores, sendo estas escolhidas a partir de um conjunto de dados. Estes dados equivalem às ultimas 12 horas de medições recebidas. Na metodologia ANNODE foram usadas 12 horas que representam informação suficiente de uma maré completa. Neste trabalho foi determinado que este é o número ideal de horas a ter em conta para a criação de previsões.

As figuras 3.4 e 3.5, representam, em síntese, a criação de um vector de entrada. Pode-se verificar que são usadas mais medições próximas da medição a ser prevista, comparando com o número de medições mais antigas. Isto não é por acaso, pois existe maior correlação entre as medições mais próximas e a medição a ser prevista do que a correlação com as medições mais antigas.

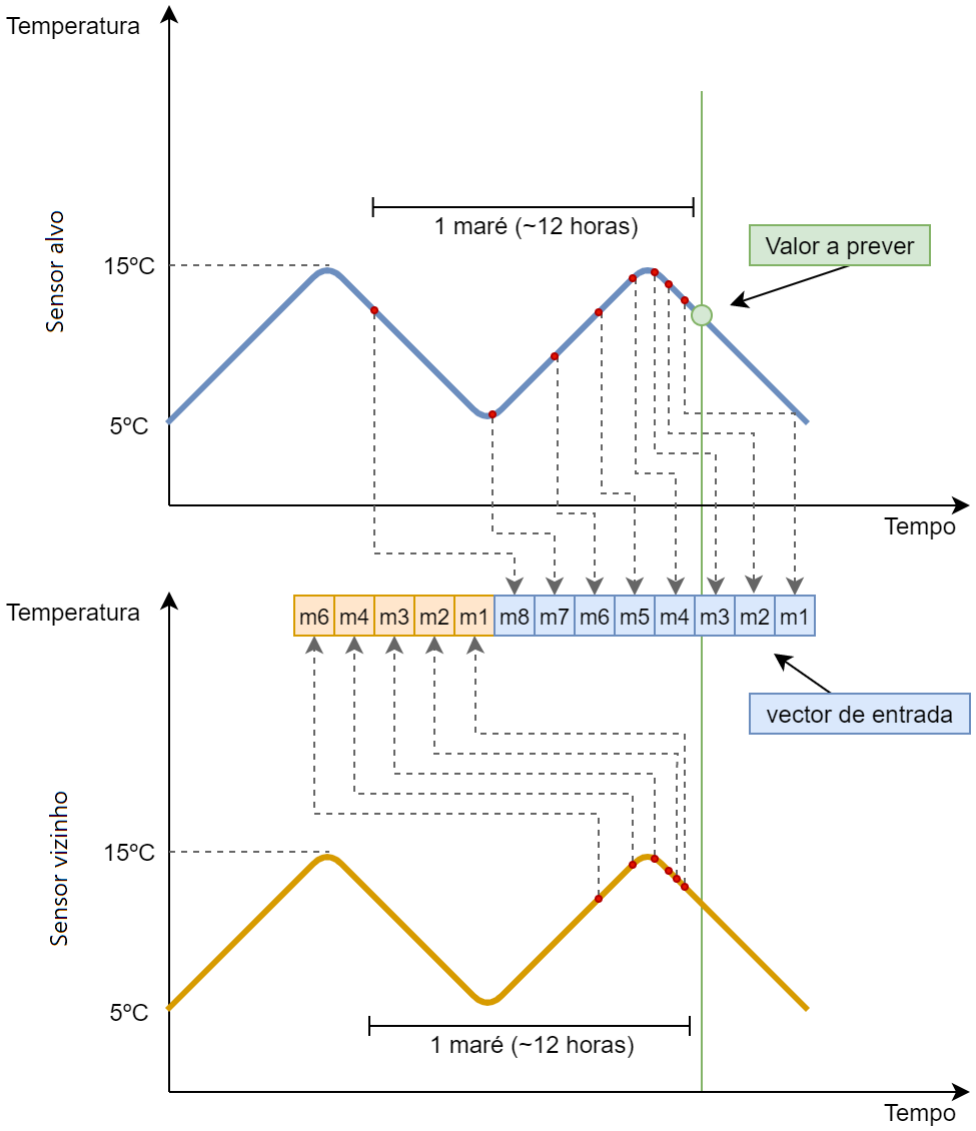

Figura 3.5: Exemplo de construção de um vector de entrada utilizando o sensor alvo e um vizinho.

Seguindo a metodologia ANNODE, é necessário definir e treinar várias redes, pelo que se torna necessário criar vários conjuntos de dados de entrada para treinar cada uma dessas redes. Uma das primeiras tarefas foi a criação de um *script* de processamento de dados e outro para o treino das redes neuronais. O primeiro *script* começa por ler um ficheiro de entrada do tipo *JSON*, verificando se todos os os

valores de entrada escritos nesse ficheiro estão correctos, isto é, verifica se os dados de entrada necessários se encontram neste ficheiro. O fluxo de dados, deste *script* pode ser visualizado na figura 3.6.

FIGURA 3.6: *Fluxo de dados no processamento dos dados*

De seguida começa a processar os dados formatando-os n vezes, sendo este n o número de sensores. Como o treino das redes neuronais de cada sensor necessita dos dados de todos os sensores, é criado um ficheiro para cada sensor que contém os dados de todos. O que difere de ficheiro para ficheiro é a formatação, cada um é formatado em relação ao sensor alvo.

3.2.2 Treino das redes neuronais

Após a conclusão do processamento dos dados e com os vectores de entrada prontos, é realizado o treino das redes neuronais. O segundo *script*, começa por ler um ficheiro que, como o *script* anterior, contém os valores de entrada necessários para o treino. No primeiro passo começa por ler os dados processados e de seguida é criada uma rede neuronal não treinada. Ao ter os dados prontos, o treino da rede neuronal não treinada é iniciado. São geradas sempre entre 5 a 10 redes com o mesmo pressuposto e destas é escolhida a rede "óptima", isto é, a com melhor desempenho. Seguindo a metodologia ANNODE, este processo é feito para cada uma das 3 redes neuronais necessárias. O tempo de treino varia entre 2-3 horas no computador de teste. Este continha um processador *AMD Ryzen 5 3600* (4.2Ghz), 16GB de memória (3733Mhz) e o sistema operativo *Windows 10 Pro* (versão 10.0.18363). O fluxo de dados está desenhado na figura 3.7.

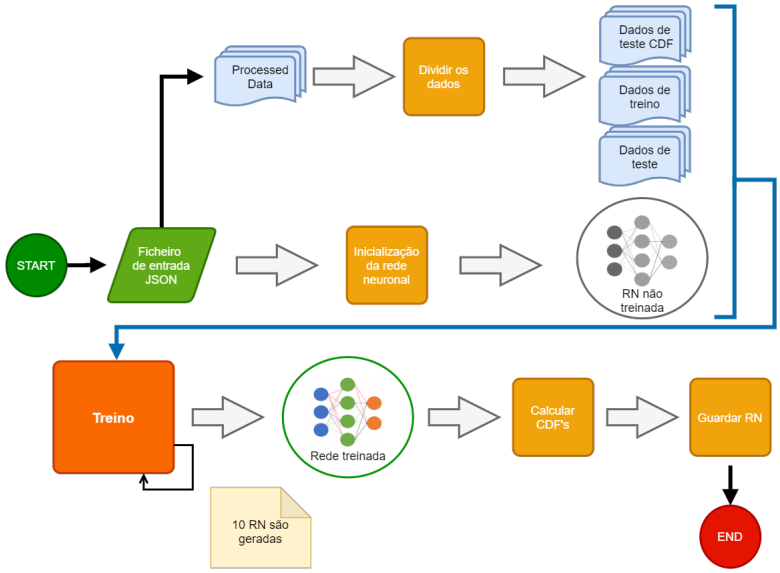

FIGURA 3.7: Fluxo de dados no treino das redes neuronais

Para além do treino este *script* é também responsável pelo cálculo da Função de Densidade Cumulativa da Probabilidade dos Erros Quadráticos (CDF), necessária para a detecção de *outliers* e o cálculo de qualidade (explorado na secção seguinte).

3.3 Execução em tempo-real

3.3.1 Processamento de medições

O modulo de **Análise** consiste na implementação da lógica ANNODE com as necessárias modificações. Esta lógica é baseada em técnicas de fusão de dados usando aprendizagem automática que modela o comportamento de cada sensor de acordo com medições passadas. Considerando que deve haver um histórico de medições anteriores à aplicação dos procedimentos, existe uma etapa preliminar que consiste na criação dos tais modelos. Para fins de detecção de falhas, cada sensor deve ser representado por, pelo menos, dois modelos explorando correlações temporais, espaciais e de valor entre medições passadas do sensor alvo ou uma combinação dos sensores existentes na rede de sensores. Para isto existirá um modelo auto-regressivo e outro(s) com entras exógenas. Além disto, a estrutura ANNODE é composta pelos seguintes quatro componentes que são executados sempre que uma nova medição de um sensor é recebido:

- *Bloco de Previsão (P)* - Quando uma medição é recebida, a sua qualidade deve ser calculada. Não sabendo se a medição é correta ou não, são usados N modelos para gerar várias estimativas que serão usadas para comparar com a medição recebida para avaliar a sua qualidade. Se aplicável, será usada uma previsão com maior qualidade para substituir esta.
- *Bloco de Detecção de Falhas (FD)* - Este bloco é usado para identificar possíveis falhas nos dados, sendo as medições classificadas como correctas ou incorrectas. Este bloco tem também a responsabilidade de distinguir erros dos sensores de possíveis eventos no ambiente, não sendo os últimos necessariamente erros.
- *Bloco de Avaliação de Qualidade (QE)* - Usando a saída dos blocos anteriores é possível calcular o coeficiente de qualidade da medição. Se uma medição for considerada incorrecta este coeficiente tem o valor de 0, se for considerada correcta este coeficiente terá um valor máximo de 1.
- *Bloco de Reavaliação (MR)* - Se uma medição é considerada incorrecta não deve ser usada para não propagar erros no futuro. Este bloco tem como objectivo diminuir a propagação de erros substituindo a medição incorrecta com uma estimativa derivada dos modelos de redes neuronais que tenha uma qualidade suficientemente boa.

Um Diagrama dos 4 blocos é apresentada na figura 3.8. Aqui é possível ver as ligações entre cada bloco, considerando os valores de entrada necessários para a criação de previsões e do coeficiente de qualidade.

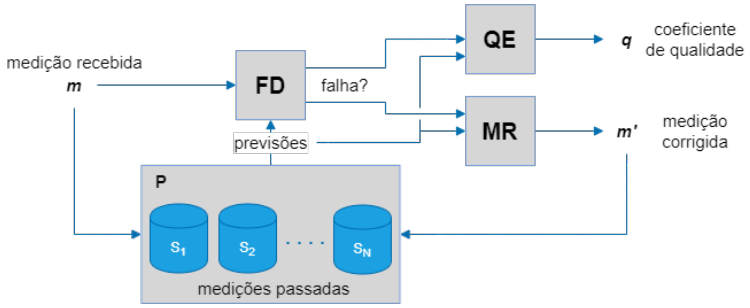

FIGURA 3.8: Diagrama de fluxo da implementação ANNODE [2]

Bloco de Previsão

Em relação ao bloco de Previsão (P) foram usadas redes neuronais *feedforward Multilayer Perceptron* (MLP), por recomendação de [2] devido à sua estabilidade e capacidade de resolver problemas não lineares. As redes utilizadas contêm 2 camadas de neurónios ocultas como é mostrado na figura 3.9: a camada de entrada contem N neurónios correspondendo ao número de valores no vector de entrada, a primeira camada oculta contém 20 neurónios e a segunda contém 15 neurónios. Por último existe apenas um neurónio na camada de saída que corresponde ao valor da previsão.

Os neurónios da camada oculta usam como função de activação a função *hyperbolic tangent sigmoid (tansig)* e o neurónio de saída implementa um combinação linear.

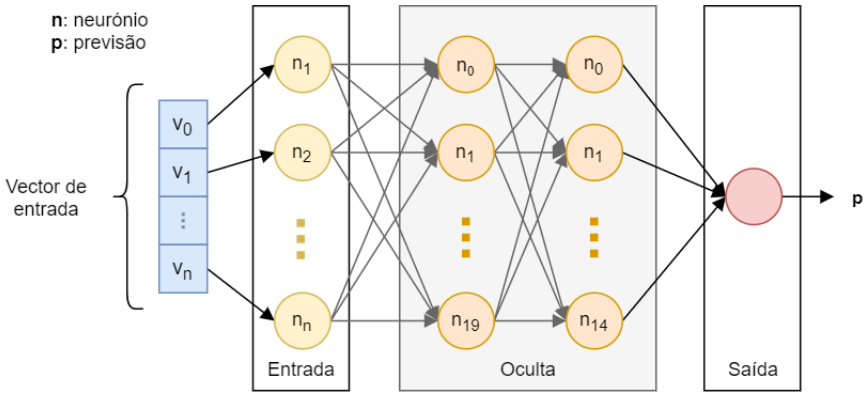

FIGURA 3.9: Rede Neuronal implementada [2]

Foram utilizados 2 algoritmos de optimização sendo ambos algoritmos de *back-propagation*, *RMSprop* e *Adam*. Este último usa uma combinação de *RMSprop* e *Stochastic Gradient Descent* (SGD). Ambos têm prós e contras. Em geral foi usado o algoritmo *Adam* mas em certas situações verifica-se que o *RMSprop* foi mais eficaz.

Adam é o algoritmo de optimização mais rápido e que obteve maior desempenho no geral. No entanto verificou-se que para as redes que continham dados de todos os sensores, alvo e vizinhos, este algoritmo muitas vezes não melhorava o seu desempenho. Nesses casos foi utilizado o algoritmo *RMSprop* verificando-se que obteve maior desempenho.

No bloco de Previsão são usadas três redes neuronais correspondentes a:

1. RN que representa o sensor alvo, com dados apenas do sensor alvo;
2. RN que representa o sensor alvo, com dados do sensor alvo e vizinhos;
3. RN que representa o sensor alvo, com dados apenas dos vizinhos.

Ao usar estas 3 redes podemos correlacionar sensores e destingir eventos ambientais de erros relacionados com o sensor. Ao usar redes com medições exógenas de sensores vizinhos podemos distinguir eventos localizados apenas no sensor alvo ou nos sensores vizinhos e daí concluir, com mais certeza, se uma medição é ou não um erro.

Bloco de Detecção de Falhas

No bloco de detecção de falhas o objectivo é verificar se uma medição difere das previsões calculadas pelas redes neuronais. Para isto adoptou-se uma técnica estatística [2].

Esta técnica aprende a distribuição estatística das diferenças entre uma medição m e cada uma das três previsões, para cada sensor seleccionado. O objectivo é calcular

as diferenças entre a medição m e cada uma das previsões das RN, com o propósito de caracterizar a distribuição dessas diferenças ao longo de um período de treino. Para isto, precisamos novamente de um conjunto de dados de treino, em que coleccionamos as diferenças entre as medições reais do sensor alvo e de cada previsão respectiva.

O cálculo das diferenças foi baseado no erro quadrático da medição m com cada previsão p das RN, sendo os erros quadráticos calculados pela seguinte equação:

$$e = (p - m)^2$$

Para o cálculo das distribuições estatísticas foi usado um dataset diferente do usado para o treino das redes neuronais, que será referenciado no capítulo de resultados e avaliação (capítulo 5). Extraindo as diferenças baseadas na equação anterior sobre os dados mencionados obtemos uma curva bastante semelhante à distribuição *Log-Logistic*, sendo esta a escolha para calcular a função de distribuição de probabilidades, como pode ser observado na figura 3.11 e 3.12.

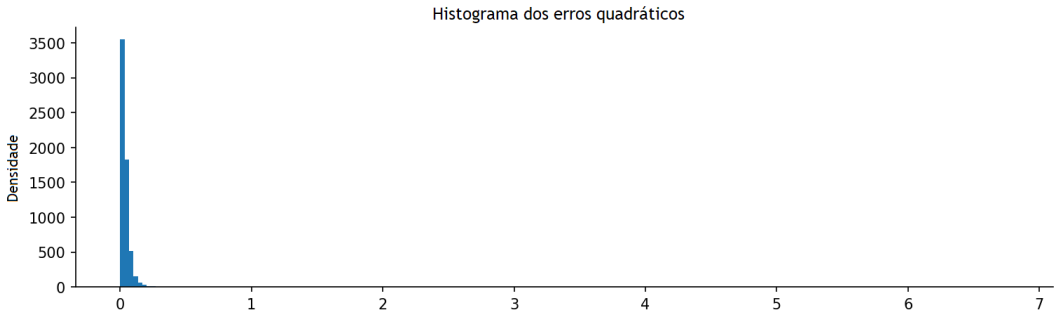

FIGURA 3.10: *Log-Logistic Fit (1)*

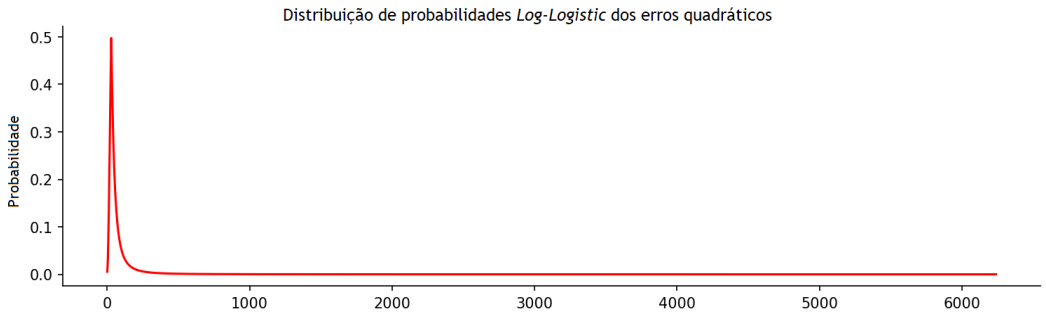

FIGURA 3.11: *Log-Logistic Fit (2)*

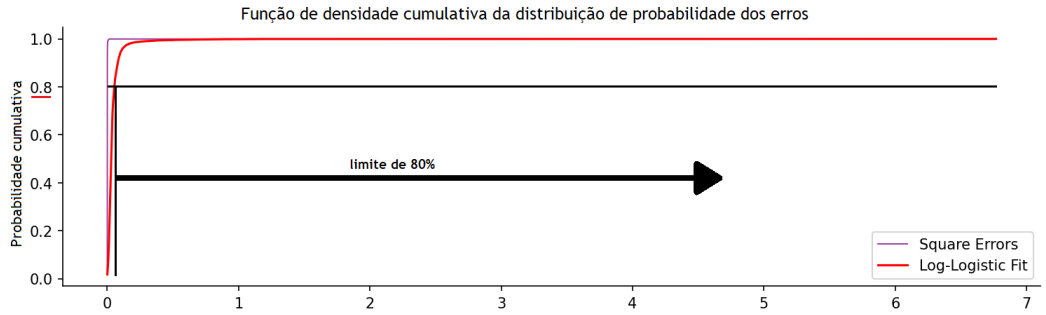

FIGURA 3.12: *Log-Logistic Fit (3)*

Ao dispor de Funções de Densidade Cumulativa (*Cumulative Density Function* - CDF) que modelam o erro esperado entre as medições fornecidas por um sensor e um modelo de previsão, é possível em tempo de execução verificar se as medições reais diferem significativamente das respectivas previsões. Ainda assim, considerando que várias comparações podem ser feitas com cada uma das previsões e que diferenças significativas podem ser devidas a eventos físicos reais e não a falhas do sensor, é necessário definir uma estratégia para decidir, com base no conjunto de comparações disponíveis, se a medição está correta ou é um erro externo. Num trabalho prévio [2], foi definido que as comparações às três RN têm de diferir todas significativamente para uma medição ser considerada um *outlier* e com isto conseguimos destingir *outliers* de eventos físicos reais. Para determinar se uma medição difere de uma previsão é usado um limiar fixo escolhido anteriormente, ao obter a probabilidade de uma medição através da CDF. Assim, podemos comparar este valor com o valor do limiar. Por exemplo: utilizando o limiar de 0.8 (80% figura 3.12), se o valor obtido for acima de 80% podemos assumir que a medição difere, se for abaixo assumimos que a medição está correcta.

Bloco de Avaliação da Qualidade e Reavaliação

O *Bloco Avaliação de Qualidade* e o *Bloco de Reavaliação* estão directamente ligados. No *bloco Avaliação de Qualidade* (QE), o objectivo é calcular o coeficiente de qualidade q que quantifica a confiança na medição m. Este valor de qualidade é calculado subtraindo a 1 o valor obtido a partir das funções de probabilidade cumulativa já estabelecidas no *Bloco de Detecção de Falhas*. Portanto, quando m não é um *outlier*, q é 1 menos a média das CDFs dos erros quadráticos. Caso contrário q é igual a 0. Este cálculo pode ser representado pela seguinte formula matemática:

$$q = 1 - \left(\frac{\sum_{i=0}^{n} CDF_{pi}(e(m, p_i))}{n} \right)$$

No *Bloco de Reavaliação* é tomada uma estratégia simples para substituir uma medição em falta ou de baixa qualidade, esta estratégia consiste em calcular a média das previsões realizadas para essa medição, como é descrito na seguinte equação:

$$m' = \frac{\sum_{i=0}^{n} p_i}{n}$$

3.3.2 Estrutura e funcionamento do servidor

Esta subsecção apresenta, em síntese, a arquitectura do servidor. A comunicação cliente-servidor foi implementada usando chamadas de procedimento remoto (*Remote Procedure Call* - RPC). Primeiramente foi utilizado uma comunicação por *sockets* implementada em *Python*, mas foram encontrados alguns problemas e foi decidido abandoná-la. Posteriormente foi criada uma comunicação usando métodos RPC. Esta escolha deveu-se a experiência anterior com esta técnica.

Modificações da arquitectura ANNODE

De seguida será representada a estrutura do servidor que se relaciona directamente com o cálculo de previsões. Tendo em conta os objectivos deste trabalho, foram feitas modificações aos blocos da estrutura ANNODE, anteriormente referidos. Estas modificações são necessárias para o bom funcionamento em tempo-real da plataforma:

- **1** - O Servidor detecta automaticamente medições em falta. Quando isto acontece não é necessário calcular a qualidade destas, pois sabe-se que o valor será automaticamente 0. Apenas são calculadas as previsões e, se aplicável, são utilizadas para substituir as medições em falta;
- **2** - Os blocos de **avaliação de qualidade** e de **reavaliação** foram unidos. Ao calcular a qualidade é automaticamente feito o cálculo necessário para detectar eventuais falhas. Caso seja detectada uma falha, e seja possível substitui-la, é feita então a substituição.

Estas modificações foram feitas de modo a optimizar o cálculo de previsões em tempo-real reduzindo o número de ciclos necessários. Na secção seguinte é possível observar o fluxo de dados desde a recepção de uma nova medição até ao cálculo das suas previsões.

Comunicação com sensores

Ao ligar o Servidor este começa por registar o objecto remoto RPC num servidor de nome RPC, previamente ligado, ficando à espera de ligações. O cliente, sabendo o nome deste objecto remoto liga-se ao servidor de nomes e procura pelo objecto RPC. Caso seja encontrado, é criado um objecto *Proxy* com ligação ao objecto remoto, e a comunicação cliente-servidor é realizada através deste *Proxy*. Esta metedologia pode ser visualizada através da figura 3.13.

O Cliente só precisa do objecto *proxy* para realizar a comunicação com o servidor. Para tal, envia uma mensagem do tipo *String* formatada com sintaxe *JSON* que posteriormente é desserializada no Servidor, obtendo a informação num Dicionário. O cliente ao enviar uma medição fica à espera de receber uma *String* que contenha o valor: “OK”. Ao verificar que o servidor recebeu a medição e está tudo em ordem, o cliente pode então desligar-se ou enviar mais medições.

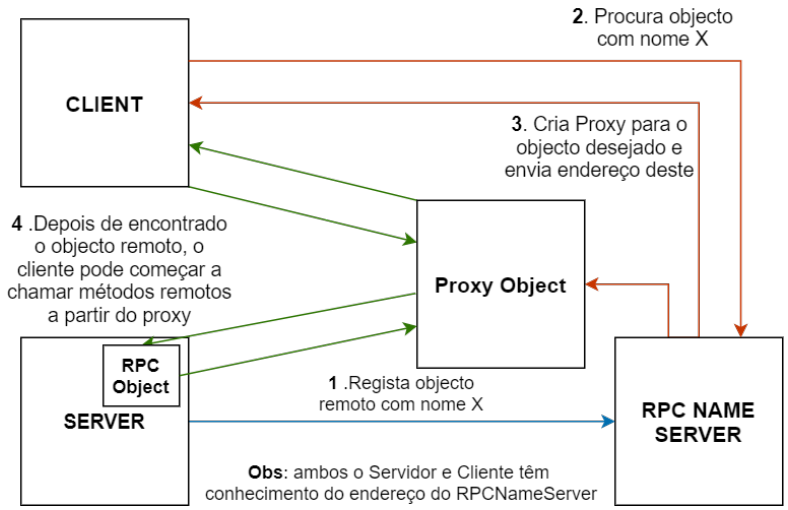

FIGURA 3.13: Chamadas de procedimento remoto

Organização interna

Na figura 3.14 é apresentada a arquitectura de software do servidor.

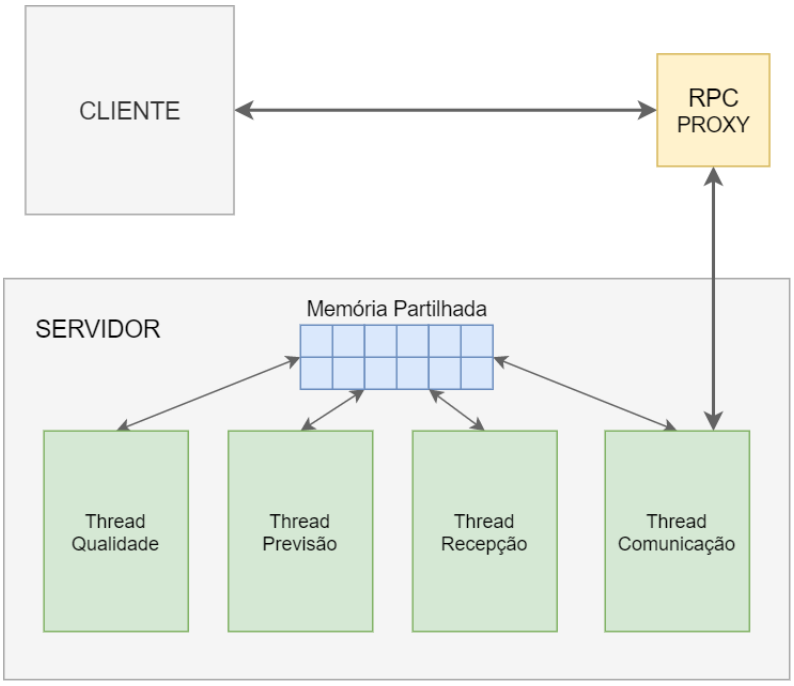

FIGURA 3.14: Arquitectura de software do servidor.

Existem 4 *threads* no servidor, uma dedicada à comunicação com os clientes (sensores), outra trata da recepção da medição, isto é, guarda-a e verifica falta de medições, outra trata do cálculo de previsões e a última trata do cálculo da qualidade das medições recebidas e também da substituição da medição por uma previsão caso seja detectado um erro. É criado então um fluxo de dados desde a chegada de uma nova medição até ao cálculo de qualidade e possível substituição desta.

Existem 4 fases de tratamento no Servidor:

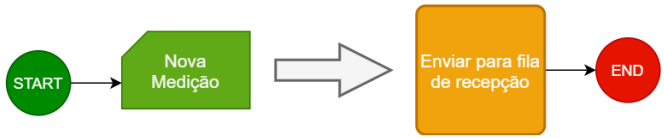

FIGURA 3.15: Fluxo de Dados na chegada de uma nova medição na *thread* de Comunicação

1. *Thread* Comunicação (figura 3.15): esta *thread* é dedicada à comunicação entre servidor e cliente, logo apenas recebe a mensagem do cliente e insere-a na fila da *thread* de tratamento.

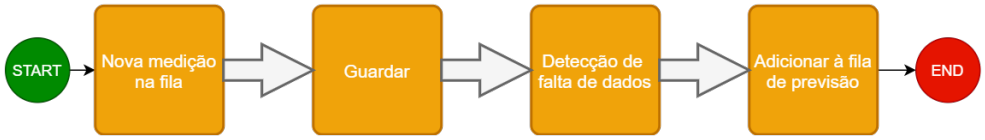

FIGURA 3.16: Fluxo de Dados na chegada de uma nova medição na *thread* de Recepção

2. *Thread* recepção (figura 3.16): é recebida uma mensagem do Cliente, contendo o valor da medição, o tipo de medição, uma estampilha temporal e o nome do sensor. Com esta informação conseguimos, guardá-la no local respectivo e detectar se entre esta medição e a última recebida existe uma falta de dados. Caso exista, será enviado para a *thread* de Previsão não só a medição recebida mas também uma representação das medições em falta para o cálculo das respectivas previsões se possível.

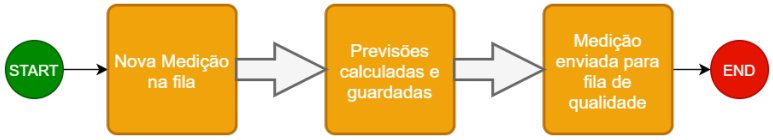

FIGURA 3.17: Fluxo de Dados na chegada de uma nova medição na *thread* de Previsão

3. *Thread* Previsão (figura 3.17): sempre que uma medição chega, esta é adicionada à fila de previsões e, ao ser adicionada à fila, esta *thread* começa o cálculo da previsão, começando pelo processamento das últimas 12 horas de dados. São construídos 3 vectores de entrada, um para cada rede neuronal. De seguida as previsões são calculadas, guardadas e enviadas para a fila da *thread* de qualidade. A única excepção será no caso de falta de medições: se as previsões foram calculadas para uma medição em falta, é calculada uma média das previsões, sendo esta média utilizada como valor de substituição directamente sem passar pelo bloco de qualidade, pois a qualidade é automaticamente 0.

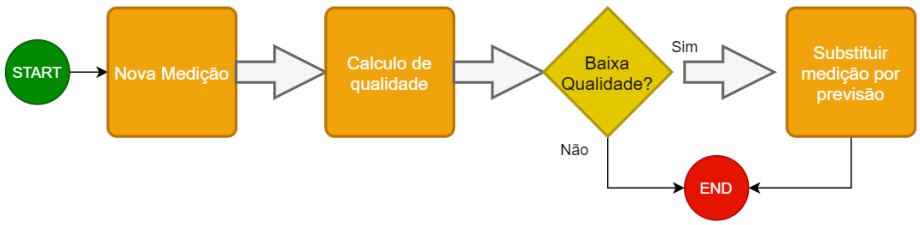

FIGURA 3.18: Fluxo de Dados na chegada de uma nova medição na *thread* de Qualidade

4. *Thread* Qualidade (figura 3.18): depois de calculadas as previsões para uma medição, estão encontradas as condições necessárias para o cálculo da qualidade. Ao obter o coeficiente de qualidade é decidido se a medição deve ser substituída ou não. Se o valor de qualidade for suficiente, o ciclo de tratamento desta medição termina, se a qualidade for abaixo do mínimo, é calculada a média das previsões e o valor da medição é substituído.

3.4 Sumário

Este capítulo está dividido em 5 partes, sendo inicialmente explicada a base desta plataforma, a solução ANNODE e os conjuntos de dados usados. De seguida, são mostrados os requisitos, explicando cada um deles. Na terceira parte é apresentada a lógica do processamento, estrutura dos dados e do treino de redes neuronais. Na quarta parte é apresentada a arquitectura do sistema em todas as suas fases, sendo descrita a sua implementação na quinta parte. No capítulo seguinte é descrita a implementação da solução final.

Capítulo 4

Implementação

Para o desenvolvimento da plataforma foi utilizada a linguagem *Python*, mais precisamente a versão 3.7. Para o desenvolvimento do servidor e do cliente foi usado *Pyro4*, uma ferramenta para criação de RPC's. Para os métodos de aprendizagem automática foi utilizado *TensorFlow*. O ambiente de desenvolvimento integrado usado foi o *PyCharm*. Inicialmente foram apenas criados os *scripts* de processamento de dados e de treino das redes neuronais, sendo o foco principal deste trabalho. Após a implementação destes *scripts* e a criação de redes neuronais utilizáveis, é então criada a comunicação servidor-cliente.

4.1 Processamento de Dados

Neste módulo foram criados vários métodos, que se separam em dois tipos: métodos para a criação de dados de treino, que processam medições gravando-as num ficheiro, e métodos para criação de vectores de entrada para o cálculo de previsões em tempo real. A figura 4.1 apresenta os métodos criados para o *script* de processamento de dados.

```
11   def build(data_cfg):...
46
47
48   def build_multiple(raw_folder, save_folder, run_periods_self, run_periods_others, tide_period, skip_period):...
100
101
102  def build_new_times(sizes, times, skip_period, tide_period):...
193
194
195  def build1_input(new_times, times, values, idx_target, tide_period, run_periods_self, run_periods_others):...
287
288
289  def generate1(target_time, sizes, times, values, tide_period, skip_period,run_periods_self,
290                run_periods_others, new_times=None):...
308
309
310  def build_inputs(sizes=None, times=None, values=None, run_periods_self=None, run_periods_others=None,
311                   tide_period=None, skip_period=None):...
425
```

FIGURA 4.1: *Script* de processamento de dados

4.1.1 Criação de dados de treino

A criação de dados de treino começa no método *build_multiple*, recebendo como valores de entrada a localização dos dados não processados e o local desejado para guardar os dados processados. De seguida é chamado o método *build_inputs*, sendo neste método calculado, em primeiro lugar, o *new_times* através do método *build_new_times* e por fim os vectores de entrada.

```
{
    "raw_path": "data/raw_main/",
    "save_path": "data/processed/",
    "metrics": ["temp"],
    "n_sensors": 4,
    "run_periods_self": 60,
    "run_periods_others": 20,
    "tide_period": 750,
    "skip_period": 0
}
```

FIGURA 4.2: *Exemplo de ficheiro de entrada para processamento de dados de treino*

Como é possível ver no ficheiro de entrada exemplo apresentado na figura 4.2 existem vários parâmetros de entrada para o processamento de dados de treino. Os dois primeiros, *raw_path* e *save_path*, indicam-nos o local onde estão guardados os dados não processados e o local onde guardar os dados processados, respectivamente.

Existem dois valores de entrada que determinam o tamanho dos vectores de entrada das redes neuronais e a importância dada a cada sensor: o valor *run_periods_self* determina o número de valores provenientes do sensor alvo e o valor *run_periods_others* que determina o número de valores provenientes de cada sensor vizinho. Neste exemplo existem 4 sensores (indicado pelo valor de entrada *n_sensors*), logo o tamanho dos vectores será 60 ou 120. Sendo 60 para redes neuronais com dados apenas do sensor alvo, ou com dados apenas dos vizinhos (20 + 20 + 20). E por último 120 para as RN com dados provenientes de todos os sensores (60 + 20 + 20 + 20).

O valor *metrics* indica-nos que iremos processar dados de temperatura. O valor *tide_period* indica-nos o coeficiente usado para calcular o tempo de uma maré.

Por fim o valor de entrada *skip_period* indica-nos a taxa de amostragem mínima. Por exemplo: num conjunto de dados em que a taxa de amostragem é de 1 minuto, mas apenas queremos usar valores de 2 em 2 minutos, devemos definir o *skip_period* para o valor de 2.

4.1.2 Criação de um vector de entrada em tempo real

A criação do vector de entrada em tempo real começa no método *generate1*. Este método recebe como input o *target_time* (valor para o qual se quer calcular o vector de entrada) e os inputs necessários para o calculo do *new_times*. Depois de obtido o *new_times* é chamado o método *build1_input*, como o próprio nome indica irá criar apenas um vector de entrada para ser usado no cálculo de previsões para a medição em causa.

A dificuldade consistiu na criação de vectores de input em tempo real, sendo como foi descrito em 3.2.1, um processo que necessita de vários dados anteriores. Já existiam métodos que criavam vários vectores de entrada para um número grande de dados, sendo estes métodos usados para criar múltiplos vectores de entrada para a criação de dados de treino. No entanto foi necessário criar um método que nos desse apenas um vector correspondente a apenas uma medição, para possibilitar o cálculo de previsões singulares em tempo-real.

Extraindo parte do código do método *build_inputs* e modificando-o foi possível elaborar um método capaz de criar apenas um input por chamada (*build1_input*).

4.2 Treino de Redes neuronais

O treino das RN é realizado em separado, com dados previamente processados.

```
19   def main():...
74
75
76   def get_cfg(ann_cfg_path):...
87
88
89   def create(ann_cfg, seed1, seed2):...
105
106
107  def create_model(ann_cfg, seed1=0, seed2=0):...
111
112
113  def calculate_cdf(model, ann_cfg, i):...
202
203
204  def train_model(model, data, ann_cfg):...
297
298
299  def save_models(ann_cfg):...
340
341
342  def unison_shuffled_copies(inputs, targets):...
346
347
348  if __name__ == "__main__":
349      main()
350
```

FIGURA 4.3: *Script* de treino das RN

O *script*, cujos métodos se apresentam na figura 4.3 começa por verificar o ficheiro de entrada, depois de verificado carrega os respectivos valores, inicializa 2 variáveis pseudo-aleatórias para serem usadas como sementes para os pesos e *bias* da rede neuronal. O método *create_model* recebe estas 2 variáveis e cria a rede neuronal com pesos e *bias* aleatórios, prontos para serem treinados. De seguida é chamado o método *train_model* recebendo o modelo previamente criado e os dados de treino. Normalmente o treino demora entre 2 a 3 horas com 2000 épocas, sendo treinadas 10 redes neuronais com valores iniciais sempre aleatórios para obter sempre redes diferentes. Quando o treino é completado é chamado o método *calculate_cdf* que irá calcular a função CDF da rede. Ao completar este passo é, finalmente, chamado o método *save_models* que irá gravar o modelo e a respectiva CDF no local desejado.

É de referir que caso algum problema aconteça e o programa pare o treino, este não recomeça do 0, o treino grava um *checkpoint* a cada 100 épocas.

4.3 Servidor

O primeiro passo no desenvolvimento do Servidor foi a criação de um sistema de classes onde se implementa toda a lógica de previsão.

A classe central no servidor é a classe *SensorHandler*, esta tem como atributos principais um objecto do tipo *SensorData*, um objecto do tipo *PredictionBlock* e um objecto *QualityBlock*. A classe *SensorData* trata do armazenamento das medições enviadas pelos clientes e é aqui que se detectam directamente as faltas de dados. A classe *PredictionBlock*, tal como o nome indica, é responsável pelo processamento das medições e o cálculo das previsões. Por fim na classe *QualityBlock*, como o próprio nome indica, é calculada a qualidade das medições. Terminado o cálculo, é também decidido se uma medição deve ou não ser substituída por uma previsão de maior qualidade. Estas classes estão representadas na figura 4.4.

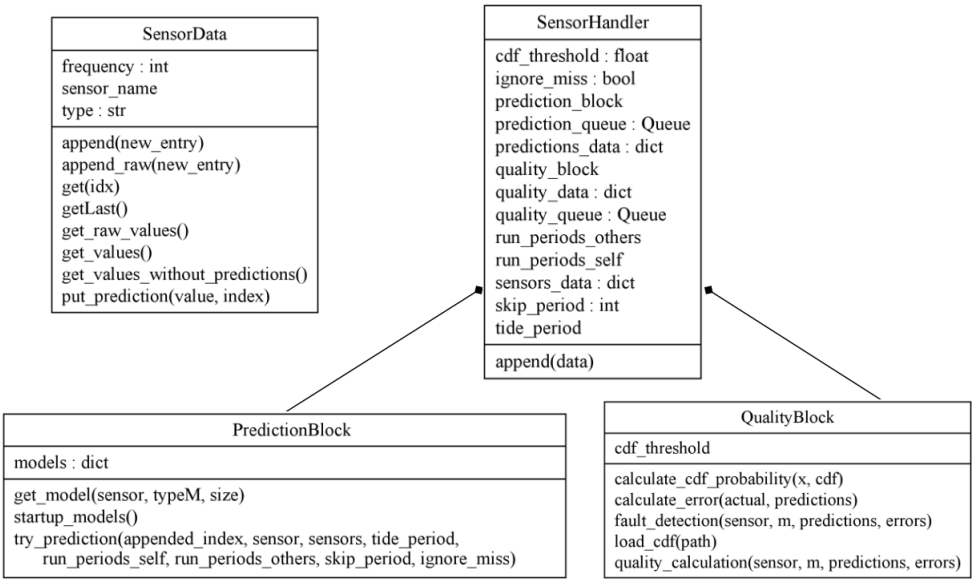

FIGURA 4.4: Diagrama de classes do Servidor

A figura 4.5 representa o diagrama de sequência desde a entrada de uma nova medição no sistema até ao cálculo da sua qualidade.

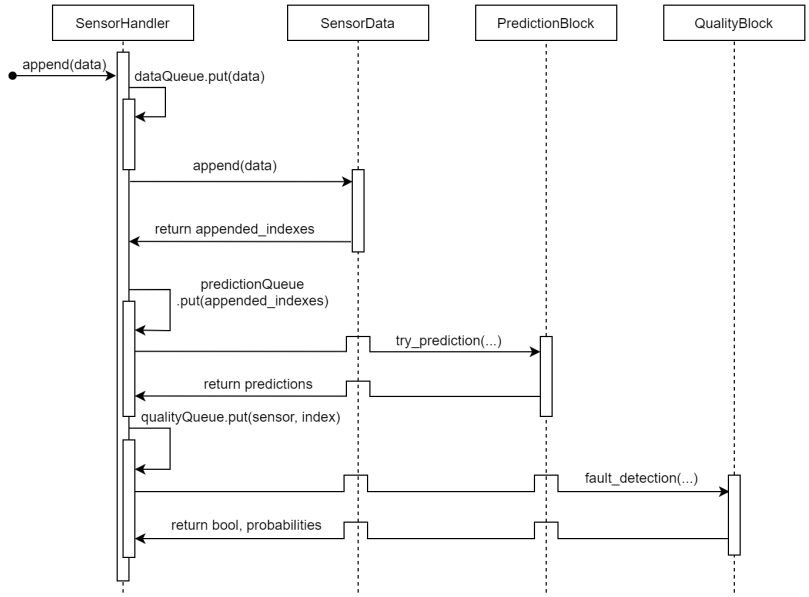

FIGURA 4.5: Diagrama de Sequência do Sistema

O segundo passo foi a construção de um protocolo sobre *sockets*. Ao procurar por soluções, foi encontrada a biblioteca *Pyro (Python Remote Objects)*, que permite construir aplicações nas quais os objectos podem comunicar entre si pela rede, usando apenas chamadas de métodos *Python*. *Pyro* é uma biblioteca construída em *Python* puro e funciona em muitas plataformas e versões diferentes.

Com isto foi construído um objecto remoto, como mostra a figura 4.6.

```
@Pyro4.expose
class Server(object):

    def connect_message(self, name):
        now = datetime.now()
        print(f'[{now:%Y-%m-%d %H:%M:%S}] {name} connected')

    def disconnect_message(self, name):
        now = datetime.now()
        print(f'[{now:%Y-%m-%d %H:%M:%S}] {name} disconnected')

    def send_message(self, message):
        dataQueue.put(message)
```

FIGURA 4.6: Objecto Remoto do Servidor

Este é o objecto remoto, a que o cliente tem acesso, permitindo-lhe chamar qualquer um destes métodos. Os 2 primeiros métodos servem apenas para descrição do processo na consola ou terminal quando um cliente se liga e desliga do servidor.

O método *send_message* é usado para enviar mensagens para o servidor. Este método recebe apenas a mensagem e insere-a numa fila para esperar o processamento, sendo este realizado pela *thread* representada na figura 4.7. Esta *thread* é criada ainda antes do registo do objecto remoto.

```
def queue_reader_thread(queue, sensorData):
    while True:
        msg = queue.get()
        data = json.loads(msg)
        sensorData.append(data)
```

FIGURA 4.7: *Thread* de processamento de mensagens

4.4 Execução da ferramenta

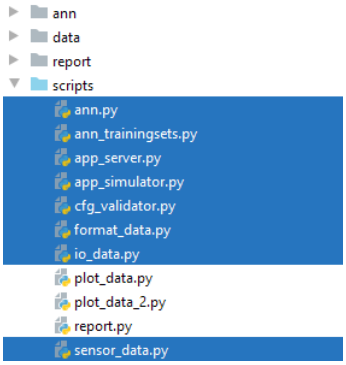

FIGURA 4.8: *Pasta e Ficheiros do programa*

Considerando a figura 4.8, que representa a pasta com os ficheiros da plataforma, podemos verificar que esta consiste em 8 ficheiros, assinalados a azul. A execução destes *scripts* e a sua função serão descritos de seguida:

1. **ann.py**: *script* de treino de redes neuronais. Recebe como parâmetro um ficheiro json com as entradas necessárias. A figura 4.9 ilustra o comando em execução bem como a saída gerada.

```
(venv) PS C:\Users\joaop\Documents\Projectos\tese> python.exe .\scripts\ann.py .\ann\ann.json
2020-09-15 19:53:15.399440: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully
2020-09-15 19:53:19.598149: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully
2020-09-15 19:53:19.624334: E tensorflow/stream_executor/cuda/cuda_driver.cc:351] failed call to cuInit:
2020-09-15 19:53:19.631629: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:169] retrieving CUDA di
2020-09-15 19:53:19.634257: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:176] hostname: JOAO-PC
2020-09-15 19:53:19.635869: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports instru
WARNING:tensorflow:`period` argument is deprecated. Please use `save_freq` to specify the frequency in nu
Train on 29235 samples
Epoch 1/2000
29235/29235 - 1s - loss: 18.5654
Epoch 2/2000
29235/29235 - 1s - loss: 12.3764
```

FIGURA 4.9: Execução do *script* treino de redes neuronais

2. **ann_trainingsets.py**: *script* de processamento para criação dos dados de treino das redes neuronais. Recebe, também, um ficheiro de entrada do tipo JSON. Este processo e a saída gerada são ilustrados na figura 4.10.

```
(venv) PS C:\Users\joaop\Documents\Projectos\tese> python.exe .\scripts\ann_trainingsets.py .\data\data.json
Configuration loaded, training sets are now being built using: data/raw_other/
| data/raw_other/01jul_2009-16jul_2009/temp/ |
-> dsdma DONE
-> jettya DONE
-> sandi DONE
-> tansy DONE
```

FIGURA 4.10: Execução do *script* de processamento de dados

3. **app_server.py**: *script* de inicialização do servidor. Este script inicializa o servidor ficando pronto para o tratamento de dados. Para tal é primeiro, necessário ligar outro servidor chamado de servidor de nomes para registar objectos remotos. A figura 4.11 demonstra a execução deste *script*. Para além desta a figura 4.12 mostra o que acontece no terminal do servidor quando este começa

a receber medições: de 10 em 10 segundos imprime o número de medições guardadas para cada sensor e ao detectar um *outlier* imprime um aviso.

```
(venv) PS C:\Users\joaop\Documents\Projectos\tese> python -m Pyro4.naming
Not starting broadcast server for localhost.
NS running on localhost:9090 (127.0.0.1)
Warning: HMAC key not set. Anyone can connect to this server!
URI = PYRO:Pyro.NameServer@localhost:9090
```

```
(venv) PS C:\Users\joaop\Documents\Projectos\tese> python .\scripts\app_server.py
2020-09-16 11:59:56.419889: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully op
ened dynamic library cudart64_101.dll
2020-09-16 11:59:57.927699: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully op
ened dynamic library nvcuda.dll
2020-09-16 11:59:57.953519: E tensorflow/stream_executor/cuda/cuda_driver.cc:351] failed call to cuInit: CU
DA_ERROR_NO_DEVICE: no CUDA-capable device is detected
2020-09-16 11:59:57.959016: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:169] retrieving CUDA diag
nostic information for host: JOAO-PC
2020-09-16 11:59:57.961330: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:176] hostname: JOAO-PC
2020-09-16 11:59:57.962969: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports instruct
ions that this TensorFlow binary was not compiled to use: AVX2
[2020-09-16  12:00:04] Server ready
```

FIGURA 4.11: Passos para a execução do servidor.

```
[2020-09-16  12:00:04] Server ready
[2020-09-16 12:05:47] Simulator connected
[2020-09-16 12:05:49] [dsdma -> M: [482]]   [tansy -> M: [482]]   [sandi -> M: [477]]   [jettya -> M: [481]]
[2020-09-16 12:05:59] [dsdma -> M: [550]]   [tansy -> M: [550]]   [sandi -> M: [546]]   [jettya -> M: [550]]
[2020-09-16 12:06:09] [dsdma -> M: [615]]   [tansy -> M: [615]]   [sandi -> M: [612]]   [jettya -> M: [618]]
[2020-09-16 12:06:19] [dsdma -> M: [681]]   [tansy -> M: [681]]   [sandi -> M: [676]]   [jettya -> M: [683]]
[2020-09-16 12:06:29] [dsdma -> M: [747]]   [tansy -> M: [747]]   [sandi -> M: [743]]   [jettya -> M: [750]]
[2020-09-16 12:06:39] [dsdma -> M: [811]]   [tansy -> M: [811]]   [sandi -> M: [806]]   [jettya -> M: [813]]
[2020-09-16 12:06:47] [dsdma at index: 517 FAULT DETECTED] AVISO
[2020-09-16 12:06:49] [dsdma -> M: [873]]   [tansy -> M: [873]]   [sandi -> M: [868]]   [jettya -> M: [875]]
```

FIGURA 4.12: *Log* do servidor.

4. **app_simulator.py**: *script* para simulação de um ou mais clientes (sensores), este programa cria uma ligação ao servidor e envia medições à velocidade escolhida pelo utilizador. Tem como entrada o caminho para os dados a serem simulados e a velocidade a que as medições devem ser enviadas, em segundos. A figura 4.13 ilustra a execução e a saída gerada.

```
(venv) PS C:\Users\joaop\Documents\Projectos\tese> python .\scripts\app_simulator.py
 data\raw_other\01out_2013-31dec_2013\temp 0.01
[2020-09-16 16:00:57] Sending data...
```

FIGURA 4.13: Execução do Simulador.

5. **cfg_validator.py**: *script* contém métodos usados para verificar se os ficheiros de entrada estão de acordo com o esperado.

6. **format_data.py**: *script* contém métodos para a formatação e processamento dos dados. Estes métodos são usados, por exemplo, para a criação de um vector de entrada para as redes neuronais, tanto para o treino das redes neuronais como para o cálculo em tempo-real de uma previsão.

7. **io_data.py**: *script* contém métodos para guardar dados em ficheiros. Estes métodos são usados na criação de dados de treino para as redes neuronais.

8. **sensor_data.py**: *script* contém as classes desenvolvidas e toda a lógica ANNODE de processamento em tempo-real (secção 4.3).

Todos os restantes *scripts*, **plot_data.py**, **plot_data_2.py** e **report.py**, encontrados nesta pasta foram usados para a criação dos gráficos apresentados neste relatório.

4.5 Sumário

Este capítulo está dividido em 4 partes. Primeiro é descrita a implementação do processamento de dados, de seguida o treino das redes neuronais. Em 3º é descrita a implementação do Servidor e por fim temos a secção onde é exibida a execução da ferramenta. No capitulo seguinte é feita avaliação e analise de resultados da solução implementada.

Capítulo 5

Resultados e avaliação

Este capítulo descreve o processo de avaliação considerando todas as estratégias de decisões anteriormente descritas para a detecção de *outliers*. Para tal, será feita a validação e avaliação da réplica implementada da metodologia ANNODE [2] comparando directamente com os resultados originais do artigo, tal como o desempenho do servidor em termos de taxa de transferência.

5.1 Aplicação sobre dados reais

Para efeitos de teste e de obtenção de resultados foi utilizado o mesmo dataset que foi usado no implementação original ANNODE [2]. Este dataset consiste em três conjuntos de medições de uma rede de monitorização ambiental, designada de *SATURN Observation Network* [19]. O primeiro conjunto é usado para treinar as redes neuronais, o segundo é usado para calcular as distribuições estatísticas dos erros das redes neuronais e por fim, o terceiro conjunto é usado para testar a plataforma desenvolvida.

O sistema de rio-estuário usado é o estuário do rio Columbia, que forma uma fronteira natural entre os estados de Washington e Oregon, nos Estados Unidos da América, ligados ao Oceano Pacífico. Esta parte do estuário é monitorizada na Universidade de Ciência e Tecnologia pela rede de monitorização (denotada de SATURN), composta por vários nós sensores inseridos ao longo do rio. Com o objectivo de avaliar o trabalho foram usadas as estações *Jetty A*, *Lower Sand Island Light*, *Desdemona Sands Light* e *Tansy Point*, como pode ser observado na figura 5.1.

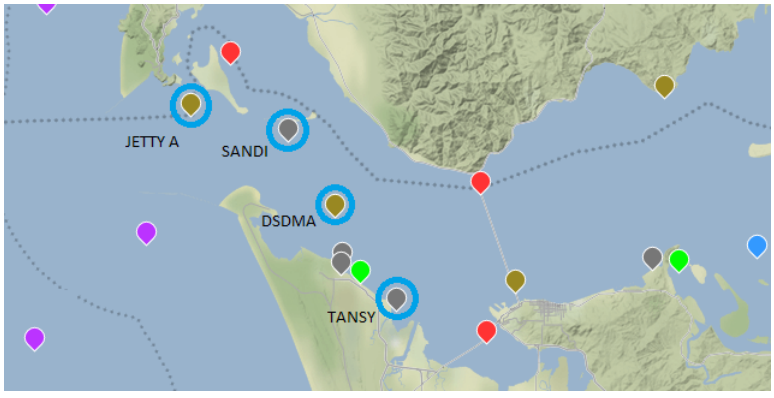

Figura 5.1: *SATURN Observation Network* [19]

Para a construção dos modelos de previsão, é necessário caracterizar os comportamentos dos parâmetros monitorizados. Estes comportamentos influenciam directamente o vector de entrada das redes neuronais (ver 3.2.1). Como se tratam de sensores aquáticos, existem dois factores evidentes que se devem ter em conta. O

primeiro factor são as marés, que influenciam bastante os sensores de temperatura. Esta influência pode ser verificada na figura 5.2.

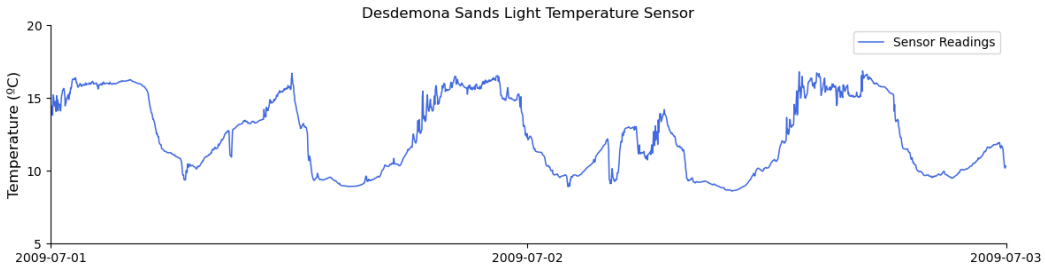

FIGURA 5.2: 2 Dias de medições de temperatura do sensor *Desdemona Sands Light*

O segundo factor são as estações do ano. Como se verifica na figura 5.3 a temperatura da água é mais alta durante o Verão e mais baixa durante o Inverno, um padrão que se repete anualmente.

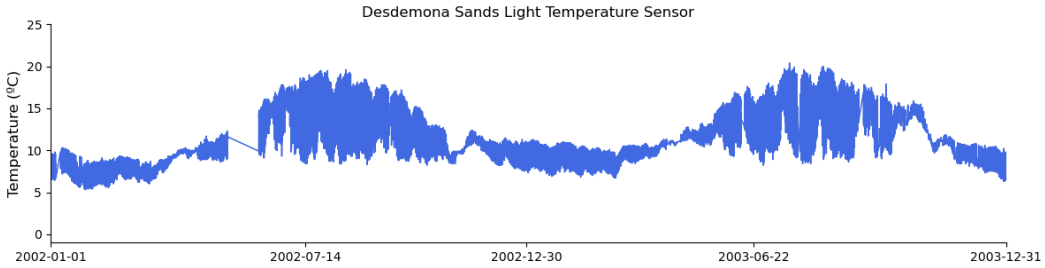

FIGURA 5.3: 2 Anos de medições de temperatura do sensor *Desdemona Sands Light*

Com esta informação é possível concluir que os dados usados para o treino das redes neuronais devem representar ambos os padrões. Os vectores de entrada devem constituir pelo menos 12 horas de dados anteriores para representarem as marés e devem ser usados, no total, aproximadamente 1 ano de dados, representando as variações do Verão e Inverno.

Seguindo os métodos de avaliação de [2], foram calculados rácios de detecção (RD) e rácios de falsos-positivos (RFP). O rácio de detecção é calculado dividindo o número de verdadeiros-positivos detectados pelo número total de verdadeiros-positivos. O rácio de falso-positivos é calculado dividindo o número de falsos-positivos pelo número total verdadeiros-falsos (medições correctas).

Com estes rácios é possível avaliar a precisão e desempenho da plataforma desenvolvida. Para os testes foi usado um *dataset* real, com número de *outliers* também fornecido. Com este número é possível tirar conclusões em relação ao desempenho.

Os testes foram realizados sobre 3 conjuntos de medições. O primeiro conjunto, usado para treinar as redes neuronais, contém medições compreendidas entre 21-07-2009 e 05-06-2010. O segundo conjunto, para calcular as distribuições estatísticas dos erros das redes neuronais, contém medições compreendidas entre 20-08-2010 e 10-10-2010. Por fim, o terceiro conjunto, usado para testar a plataforma desenvolvida, contém medições compreendidas entre 01-10-2013 e 01-01-2014, o qual irá ser mais focado daqui para a frente.

Foram sempre usados conjuntos diferentes, para eliminar qualquer tipo de influência e produzir os resultados mais fidedignos possíveis pelas redes neuronais.

Como descrito em [2], o terceiro conjunto de medições contém as seguintes falhas:

- *Desdemona* - 44 *outliers*;
- *Tansy Point* - 11 *outliers*;
- *Lower Sand Island Light* - 1 *outlier*;
- *Jetty A* - 0 *outliers*.

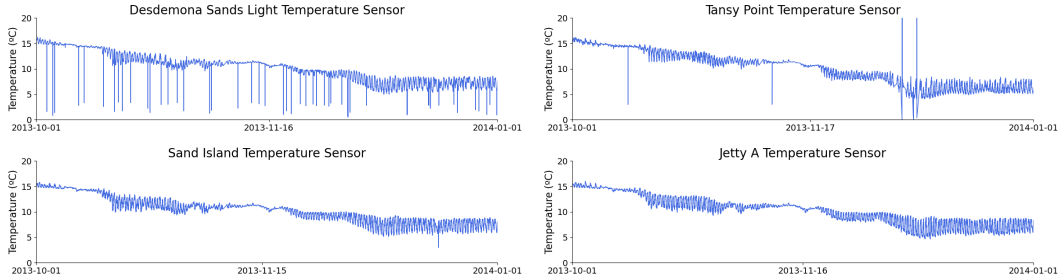

FIGURA 5.4: Dataset de teste

Estas falhas podem ser visualizadas na figura 5.4. Para cada um destes sensores foram treinadas três RN, usando valores passados de acordo com o que foi descrito na secção 5.1.

5.1.1 Simulador

Para testar a plataforma foi criado um *script* que simula um ou mais clientes. Este simulador recebe como input os dados a serem simulados e um coeficiente de velocidade correspondente à velocidade de envio de dados do sensor. A velocidade pode ser instantânea, sempre que possível, ou uma velocidade escolhida pelo utilizador. Por exemplo, de 2 em 2 segundos, ou simular o tempo real de envio dos dados. Os dados são enviados pela ordem das estampilhas temporais. Este simulador lê ficheiros do tipo *numpy* ou *matlab*.

5.2 Resultados

5.2.1 Detecção de *outliers*

Foram testados vários valores para o limiar e registada a eficácia da estratégia. Os valores para cada limiar são apresentados na tabela 5.1.

	Jetty A			**Lower Sd**			**Desdemona**			**Tansy**		
Outliers reais:	**0**			**1**			**44**			**11**		
Limiar	***Detectados***	RD	RFP	***Detectados***	RD	RFP	***Detectados***	RD	RFP	***Detectados***	RD	RFP
0.98	3	100%	0.0156%	2	100%	0.0053%	52	100%	0.0422%	9	81.81%	0%
0.995	0	100%	0%	2	100%	0.0053%	48	100%	0.0211%	9	81.81%	0%
0.996	0	100%	0%	1	100%	0%	47	100%	0.0158%	9	81.81%	0%
0.997	0	100%	0%	1	100%	0%	47	100%	0.0158%	9	81.81%	0%
0.998	0	100%	0%	1	100%	0%	45	100%	0.0053%	9	81.81%	0%
0.9985	0	100%	0%	1	100%	0%	45	100%	0.0053%	9	81.81%	0%

TABELA 5.1: Resultados para diferentes limiares

Na tabela são apresentados o número total de valores detectados considerados como *outliers* para cada limiar e ao lado o rácio de detecção (coluna RD), e o rácio de falsos-positivos (coluna RFP). Como o número de dados é bastante elevado é normal que o rácio de falsos-positivos seja baixo. Dado o elevado número de dados

no dataset de teste, o baixo valor de RFP é esperado considerando que existem muito poucos *outliers*. Da mesma forma, sempre que um *oulier* não é detectado, o RD baixa significativamente.

Analisando a tabela 5.1 podemos observar que no sensor *Tansy* não existem variações nos resultados. Posto isto, apenas os outros 3 sensores serão focados. Tendo como prioridade o rácio de detecção, mantendo o RFP o mais baixo possível, podemos verificar que os rácios de 0.998 e 0.9985 têm os melhores resultados obtendo 100% de RD em todos os 3 sensores focados. Caso a prioridade seja manter o rácio de falsos-positivos a 0%, mas mantendo o rácio de detecção o mais alto possível, os melhores valores para o limiar são também 0.998 e 0.9985. Os valores 0.995 e 0.996 apenas aumentam o RFP não dando melhorias no RD. Logo a escolha passa pela preferência: se a preferência é confiabilidade o melhor valor é 0.9985, pois dá-nos uma pequena margem (0.0005) de segurança. Se a preferência é o máximo de RD possível, a melhor escolha é 0.998.

Para os seguintes gráficos foi considerado o limiar de 0.9985 que garante detectar apenas *outliers* com o mínimo de falsos-positivos possível. Podemos observar em concreto as medições que foram consideradas *outliers* pela plataforma. Estas medições estão marcadas por pontos de cor vermelha.

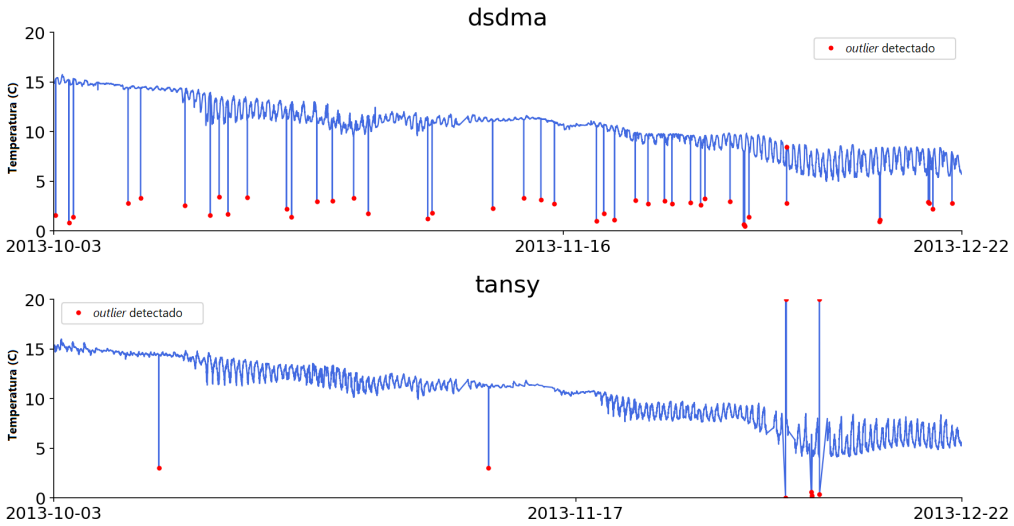

FIGURA 5.5: *Outliers* detectados

Na figura 5.5 é possível ver as medições e os *outliers* do sensor *Desdemona Sands Light* e do sensor *Tansy A*.

5.2.2 Correcção de *outliers* e coeficiente de qualidade

Na figura 5.6 é possível ver o output do Bloco de Qualidade ao longo das medições. Os *outliers* são os valores com qualidade zero.

É também importante detectar estes erros para o sistema de previsão. Caso seja usada uma destas medições com falhas nos modelos de previsão, estes erros irão propagar-se, criando previsões pouco ou não confiáveis. Para isto ao detectar uma falha nas medições, esta é substituída por uma previsão. Estas, serão usadas para o calculo de previsões futuras. Mesmo não sendo 100% correctas estas previsões terão sempre uma maior precisão do que as falhas.

Na figura 5.7 é possível ver a aplicação da metodologia, mostrando as medições que estão corrigidas em comparação com os *outliers*. Como no sensor *Desdemona Sands*

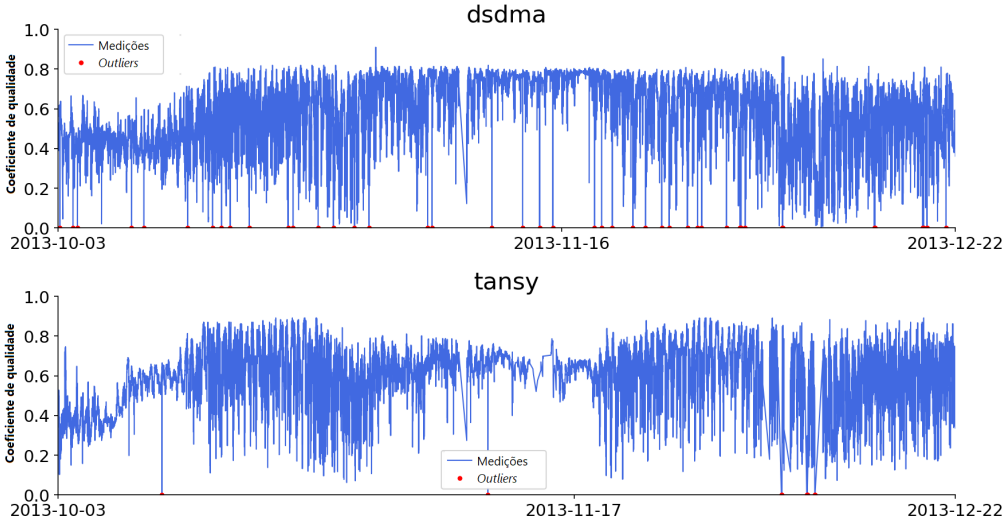

FIGURA 5.6: Coeficiente de qualidade

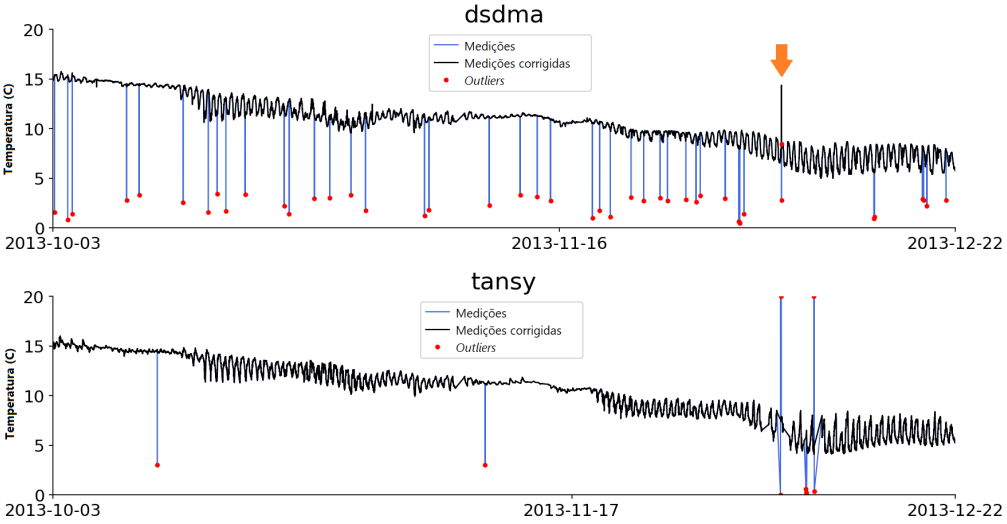

FIGURA 5.7: Medições Corrigidas

Light existe um *outlier* falso-positivo, é possível verificar que uma das previsões difere bastante do valor esperado (assinalada pela seta cor laranja).

É possível ver nas figuras 5.5, 5.6 e 5.7 que a metodologia aplicada funciona, pois são detectados os *outliers* e são produzidas previsões para substituir estes *outliers*, capazes de simular o comportamento dos sensores em tempo-real.

De seguida será comparada e avaliada esta implementação com a implementação original *ANNODE* [2].

5.2.3 Desempenho do servidor

Para além dos testes feitos sobre o dataset, foi também feito um teste à taxa de transferência do servidor, para avaliar o fluxo que consegue acompanhar. Usando o simulador anteriormente referido, foram usadas várias taxas de amostragem, e foi obtido durante este processo o tamanho da fila de previsões ao fim de 10 minutos. Tendo em conta que o computador de teste continha 16GB de Memória e um processador *AMD Ryzen* 5 3600, estes foram os resultados (Figura 5.8).

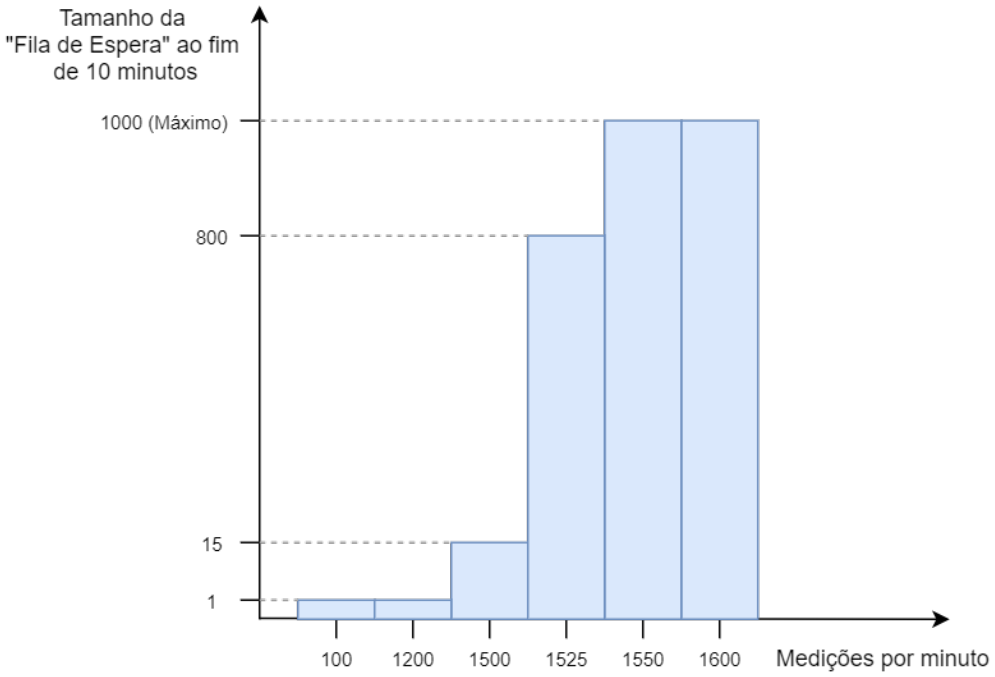

FIGURA 5.8: Medições Corrigidas

Pode-se verificar que ao enviar pouco mais de 1500 medições por minuto para o servidor chegamos ao limite deste e que a partir deste numero podemos com segurança dizer que não iremos obter mais desempenho.

5.3 Avaliação

Para validar a implementação é utilizado o mesmo *dataset* que na secção de resultados 5.2, para comparar as duas plataformas.

	Jetty A						**Lower Sd**					
Outliers reais	**0**						**1**					
	ANNODE (R)			ANNODE			ANNODE (R)			ANNODE		
	D	RD	RFP	***D***	RD	RFP	***D***	RD	RFP	***D***	RD	RFP
0.98	4	100%	0.0208%	/	/	/	2	100%	0.0053%	/	/	/
0.995	1	100%	0.0052%	/	/	/	2	100%	0.0053%	/	/	/
0.996	1	100%	0.0052%	/	/	/	1	100%	0%	/	/	/
0.997	0	100%	0%	2	100%	0.009%	1	100%	0%	7	100%	0.028%
0.998	0	100%	0%	2	100%	0.009%	1	100%	0%	7	100%	0.028%
0.9985	0	100%	0%	1	100%	0.0052%	1	100%	0%	4	100%	0.014%
0.999	0	100%	0%	1	100%	0.0052%	1	100%	0%	2	100%	0.014%

TABELA 5.2: Comparação do ANNODE original com o ANNODE replicado (ANNODE (R)) nos sensores *Jetty A* e *Lower Sd.*

Observando as tabelas 5.2 e 5.3 é possível comparar resultados entre as implementações para os mesmos limiares. A plataforma apresentada neste trabalho tem o nome de "réplica ANNODE"e é representada na tabela por "ANNODE (R)". A plataforma original é representada apenas com o seu nome original "ANNODE". Existe também uma coluna com o nome "***D***"que representa o número de outliers detectados. As barras "/"na tabela significa que não existiam dados de teste para aqueles limiares para a plataforma a original ANNODE.

	Desdemona						Tansy					
Outliers reais	**44**						**11**					
	ANNODE (R)			ANNODE			ANNODE (R)			ANNODE		
	D	RD	RFP	***D***	RD	RFP	***D***	RD	RFP	***D***	RD	RFP
0.98	52	100%	0.0422%	/	/	/	9	81.82%	0%	/	/	/
0.995	48	100%	0.0211%	/	/	/	9	81.82%	0%	/	/	/
0.996	47	100%	0.0158%	/	/	/	9	81.82%	0%	/	/	/
0.997	47	100%	0.0158%	67	100%	0.306%	9	81.82%	0%	24	100%	0.121%
0.998	45	100%	0.0053%	52	100%	0.237%	9	81.82%	0%	15	100%	0.121%
0.9985	45	100%	0.0053%	39	88.64%	0%	9	81.82%	0%	9	81.82%	0%
0.999	45	100%	0.0053%	26	59.09%	0%	9	81.82%	0%	9	81.82%	0%

Tabela 5.3: Comparação do ANNODE original com o ANNODE replicado (ANNODE (R)) nos sensores *Desdemona* e *Tansy*.

Uma das primeiras diferenças a notar é a sensibilidade aos vários valores de limiar. Na implementação original verifica-se uma grande mudança nos valores de RD e RFP ao mudar de limiar, verificando-se uma grande variação para os limiares 0.999, 0.9985, 0.998 e 0.997. Já na implementação proposta neste trabalho, a diferença é pouco significativa, podendo isto ser explicado pela diferente implementação das redes neuronais utilizadas. Como foram utilizadas ferramentas e algoritmos diferentes, as redes e o desempenho destas serão sempre diferentes da original.

Na implementação posposta neste trabalho, o melhor compromisso entre RD e RFP é no limiar 0.998 e 0.9985. Para a implementação original ANNODE o mesmo acontece, sendo o melhor valor de limiar o de 0.9985. Analisando os resultados podemos verificar que a implementação proposta, comparando com a original, está dentro dos parâmetros esperados existindo, no entanto, variações entre ambas as implementações: na implementação original existe RFP superior ao RFP deste trabalho, para o mesmo limiar (0.9985) no sensor *Jetty A*, o que à primeira vista pode indicar que a implementação proposta tem melhor desempenho. Os valores dos *outliers* detectados confirmam esta hipótese.

Contudo, ao analisar os resultados do sensor Tansy, o RD é sempre 81.82% nunca atingindo os 100%, ao contrário da implementação original. Com isto podemos deduzir que ambas são de desempenho semelhante. No entanto, têm vantagens e desvantagens dependendo do limiar escolhido: estas variações irão depender se o objectivo é encontrar o maior número de *outliers* possível, não se importando com o RFP, ou ter um rácio de falsos-positivos o mais baixo possível. Mesmo assim pode dizer-se que a réplica implementada (ANNODE (R)) apresenta melhorias em relação à original, porque a réplica, tal como se pode verificar nas tabelas 5.2 e 5.3, consegue ter um RFP de 0% nos sensores *Jetty A* e *Lower Sd* o que nunca seria possível com os limiares testados na arquitectura original.

Capítulo 6

Conclusão e trabalho futuro

6.1 Conclusão

Com o objectivo de implementar uma plataforma que conseguisse, em tempo-real, a detecção de *outliers* numa rede de sensores sem fios foi utilizada uma metodologia de Aprendizagem Automática. Com base na linguagem *Python*, foi implementada uma réplica de uma plataforma para detecção de *outliers* em dados de redes de sensores sem fios, bem como para o cálculo de um coeficiente de qualidade dos dados. Para além da capacidade de análise *off-line* da plataforma original, a implementação proposta neste trabalho permite aplicação em tempo-real.

Após a implementação da comunicação cliente-servidor, utilizando métodos RPC, o desenvolvimento do Servidor formado por 4 blocos, necessários para a detecção de erros e falhas no projecto AQUAMON e a criação e treino das Redes Neuronais para a detecção de erros dos sensores de eventos físicos, foi possível demonstrar a capacidade da plataforma desenvolvida e o processamento de previsões.

A avaliação e validação desta plataforma foi realizada comparando os resultados obtidos, com os dados da implementação original ANNODE. Para tal, foram realizadas comparações de Rácios de Detecção, Rácios de Falsos-Positivos e de precisão. Ao comparar e avaliar estas estatísticas foi possível concluir que a implementação está dentro dos parâmetros esperados e os objectivos foram alcançados, obtendo algumas melhorias quando comparada à implementação original.

6.2 Trabalho Futuro

Apesar dos objectivos terem sido todos alcançados para o desenvolvimento e implementação da proposta e esta ter gerado os resultados esperados com algumas melhorias, ainda existem aspectos que podem ser melhorados, para que a detecção de erros seja o mais confiável possível.

Para trabalho futuro, um dos focos a seguir é melhorar o Rácio de Detecção, especificamente, para o sensor *Tansy A*, pois nunca foi possível contar com uma detecção a 100%.

Outro ponto importante seria o aperfeiçoamento do servidor, pois, ao receber as medições para processamento, existe um *bottleneck* na criação de previsões. Sendo uma rede de sensores aquáticos, geralmente, a velocidade de envio das medições não é muito alta. Apesar de isto não ser um problema para um número baixo de sensores o mesmo pode não acontecer para um número maior, tornando a probabilidade da ocorrência de *bottlenecks* mais alta.

Bibliografia

[1] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, *Fundamental concepts of dependability.* University of Newcastle upon Tyne, Computing Science, 2001.

[2] Gonçalo Jesus, António Casimiro, Anabela Oliveira *Dependable Outlier Detection in Harsh Environments Monitoring Systems.* Computer Safety, Reliability, and Security. SAFECOMP 2018. Lecture Notes in Computer Science, vol 11094. Springer, Cham

[3] R. Kumar, S. A. Khan and R. A. Khan, *Revisiting Software Security: Durability Perspective.* International Journal of Hybrid Information Technology (SERSC) Vol.8, No.2 pp.311-322, 2015.

[4] F. Rozenblatt *The Perceptron: A Probabilistic Model For Information Storage and Organization in The Brain* Cornell Aeronautical Laboratory, Psychological Review. Vol. 65, No. 6, 1958.

[5] David E.Rumelhart, Geoffrey E. Hinton and Ronald J. Williams *Learning representations by back-propagating errors* Department of Computer Science, Carnegie-Mellon University Pittsburgh, Philadelphia 15213, USA, NATURE VOL 323, 1986.

[6] Martin Riedmiller and Heinrich Braun *Rprop - A Fast Adaptive Learning Algorithm. Proceedings of the International Symposium on Computer and Information Science VII*, 1992.

[7] Murad A Rassam, Mohd Aizaini Maarof, and Anazida Zainal *Adaptive and online data anomaly detection for wireless sensor systems.* Knowledge-Based Systems 60 (2014), 44–57.

[8] S. Rajasegarar, C. Leckie, J. C. Bezdek, and M. Palaniswami. *Centered Hyperspherical and Hyperellipsoidal One-Class Support Vector Machines for Anomaly Detection in Sensor Networks.* IEEE Transactions on Information Forensics and Security 5, 3 (Sept 2010), 518–533.

[9] Z. Yang, N. Meratnia, and P. Havinga. *An online outlier detection technique for wireless sensor networks using unsupervised quarter-sphere support vector machine.* In 2008 International Conference on Intelligent Sensors, Sensor Networks and Information Processing. 151–156.

[10] Y. Zhang, N. Meratnia, and P. Havinga. *Adaptive and Online One-Class Support Vector Machine-Based Outlier Detection Techniques for Wireless Sensor Networks.* In 2009 International Conference on Advanced Information Networking and Applications Workshops. 990–995.

[11] Anderson Vinicius *Redes Neuronais Artificiais*
medium.com/@avinicius.adorno/redes-neurais-artificiais-418a34ea1a39

[12] Warren S. McCullogh and Walter H. Pitts *A Logical Calculus of The Ideas Immanent in Nervous Activity* originally published in: Bulletin of Mathematical Biophysics, Vol 5., p.115-133, 1943

[13] Ana Freitas. GFBioinfo. http://web.tecnico.ulisboa.pt/ana.freitas/bioinformatics. ath.cx/bioinformatics.ath.cx/index0717.html?id=109.

[14] Donald O. Hebb *The Organization of Behaviour* A Neuropsychological Theory, McGhill University, 1949.

[15] Caroline, Dave and Jimmy. https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/index.html

[16] M. Bahrepour, N. Meratnia, and P. J. M. Havinga. 2009 *Sensor fusion-based event detection in Wireless Sensor Networks.* In 2009 6th Annual International Mobile and Ubiquitous Systems: Networking Services, MobiQuitous. 1–8. https://eudl.eu/doi/10.4108/icst.mobiquitous2009.7056

[17] A. Ayadi, O. Ghorbel, M. S. Bensaleh, A. Obeid, and M. Abid. 2017. *Performance of outlier detection techniques based classification in Wireless sensor networks.* In 2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC). 687–692. https://doi.org/10.1109/IWCMC.2017.7986368

[18] Daniel T. Larose e Chantal D. Larose *Discovering Knowledge in Data: An Introduction to Data Mining.* IEEE computer society, 2nd Edition. 2014

[19] Center for Coastal Margin Observation & Prediction (CMOP) *SATURN Observation Network* http://www.stccmop.org/datamart/observation_network