PROJETO DE CONTROLADORES USANDO REDES NEURONAIS

Luís Miguel Mota Ferreira

Departamento de Engenharia Eletrotécnica

Mestrado em Engenharia Eletrotécnica e de Computadores

Área de Especialização em Automação e Sistemas

2015

Relatório elaborado para satisfação parcial dos requisitos da Unidade Curricular de Tese/Dissertação do Mestrado em Engenharia Eletrotécnica e de Computadores

Candidato: Luís Miguel Mota Ferreira, Nº 1080500, 1080500@isep.ipp.pt
Orientação científica: Ramiro De Sousa Barbosa, rsb@isep.ipp.pt

Departamento de Engenharia Eletrotécnica

Mestrado em Engenharia Eletrotécnica e de Computadores

Área de Especialização em Automação e Sistemas

2015

Resumo

Neste documento descreve-se o projeto desenvolvido na unidade curricular de Tese e Dissertação durante o 2º ano do Mestrado de Engenharia Eletrotécnica e de Computadores no ramo de Automação e Sistemas, no Departamento de Engenharia Eletrotécnica (DEE) do Instituto Superior de Engenharia do Porto (ISEP).

O projeto escolhido teve como base o uso da tecnologia das redes neuronais para implementação em sistemas de controlo. Foi necessário primeiro realizar um estudo desta tecnologia, perceber como esta surgiu e como é estruturada. Por último, abordar alguns casos de estudo onde as redes neuronais foram aplicadas com sucesso.

Relativamente à implementação, foram consideradas diferentes estruturas de controlo, e entre estas escolhidas a do sistema de controlo estabilizador e sistema de referência adaptativo. No entanto, como o objetivo deste trabalho é o estudo de desempenho quando aplicadas as redes neuronais, não se utilizam apenas estas como controlador. A análise exposta neste trabalho trata de perceber em que medida é que a introdução das redes neuronais melhora o controlo de um processo. Assim sendo, os sistemas de controlo utilizados devem conter pelo menos uma rede neuronal e um controlador PID.

Os testes de desempenho são aplicados no controlo de um motor DC, sendo realizados através do recurso ao *software* MATLAB. As simulações efetuadas têm diferentes configurações de modo a tirar conclusões o mais gerais possível. Assim, os sistemas de controlo são simulados para dois tipos de entrada diferentes, e com ou sem a adição de ruído no sensor. Por fim, é efetuada uma análise das respostas de cada sistema implementado e calculados os índices de desempenho das mesmas.

Palavras-Chave

Rede Neuronal, Controlo, Retropropagação, Mapeamento, PID, Controlador Estabilizador, Controlador de Referência Adaptativo, Índices de Desempenho.

Abstract

This paper reports the project developed in the Thesis/Dissertation course during the 2nd year of the Master of Electrical and Computer Engineering in the field of Automation and Systems, Department of Electrical Engineering (DEE) of the Instituto Superior de Engenharia do Porto (ISEP).

The chosen project was based on the use of technology of neural networks for implementation in control systems. Therefore, it was first necessary to conduct a study of this technology, understand how it was created and how it is structured. Finally, it is presented some case studies in which neural networks have been successfully applied.

In what concerns the implementation, it was considered different control structures, and among these chosen the stabilizer and the adaptive reference control systems. However, this study does not only employs neural networks controllers. The analysis tries to understand the extent to which the introduction of neural networks improves a process control. Therefore, the control systems used contain at least one neural network and a PID controller.

The performance tests are conducted through the use of MATLAB software and in the control of a DC motor. The simulations have different settings in order to get the most general conclusions. Thus, the control systems are simulated for two different input types and with or without added noise in the sensor. Finally, an analysis of the responses of each implemented system is performed.

Keywords

Neural Network, Control, Back-propagation, Mapping, PID, Stabilizing Controller, Reference Adaptive Controller, Performance Indexes.

Índice

Índice de Figuras

Índice de Tabelas

Acrónimos

ADALINE	–	*Adaptive Linear Neuron*
ART	–	*Adaptative Ressonance Theory*
CERN	–	*Conseil Européen pour la Recherche Nucléaire*
CSOM	–	Mapas auto-organizativos concorrentes
EWMA	–	*Exponentially Weighted Moving Average*
GFF	–	*Generalized Feedfoward*
IAE	–	*Integral Absolute Error*
ISE	–	*Istanbul Stock Exchange*
ISE	–	*Integral Square Error*
JavaNNS	–	*Java Neural Network Simulator*
MAE	–	*Mean Absolute Error*
MIMO	–	*Multiple Input Multiple Output*
MIT	–	*Massachusetts Institute of Technology*
MLP	–	*Multi Layer Perceptron*
MSE	–	*Mean Square Error*
PSD	–	*Power Spectral Density*
RBF	–	Função de base radial
RNA	–	Redes Neuronais Artificiais

SISO – *Single Input Single Output*

SNNS – *Stuttgart Neural Network Simulator*

SOM – Mapas auto-organizativos supervisionados

VLSI – *Very Large Scale Integration*

WSI – *Wilhelm-Schickard Institute*

XGUI – X *Graphical User Interface*

1. INTRODUÇÃO

Neste capítulo é feita uma introdução a este trabalho de dissertação. Aqui é tratada a contextualização do trabalho a desenvolver, bem como os seus objetivos e calendarização. Por fim, é apresentada a organização deste relatório e os temas abordados em cada capítulo.

1.1. CONTEXTUALIZAÇÃO

Um dos maiores problemas enfrentados pelos projetistas de sistemas em diferentes áreas, é encontrar meios que controlem e ajustem o sistema sob consideração dado a incerteza sobre a natureza do processo subjacente. Os processos do controlo adaptativo, ou seja, processos que apreendem e adaptam-se ao longo do tempo, têm sido projetados visando este problema para certos tipos de modelos de sistemas, onde apenas os parâmetros do modelo são desconhecidos. Estas técnicas baseiam-se no uso da forma do modelo conhecido para construir a regra de controlo, e usando a informação do sistema para estimar estes parâmetros. Embora os processos de controlo adaptativo tenham sido aplicados numa variedade de áreas (manipulação de braços robóticos, controlo de tráfego aéreo, entre outras) estes são limitados pela necessidade de se assumir que as formas das equações do sistema são conhecidas. Contudo, para um processo complexo, as formas do sistema de equações podem ser desconhecidas, tornando impossível determinar a regra de controlo necessária para ser usada nos processos de controlo adaptativo. Isto fornece a motivação para considerar o uso de redes neuronais no controlo adaptativo.

As Redes Neuronais Artificiais (RNA) têm então atraído muita atenção para o potencial de endereçar um número de difíceis problemas na modelação. Uma das áreas que está a receber especial atenção é o uso destas redes para o controlo e ajuste de sistemas dinâmicos não-lineares. Tradicionalmente, o desenvolvimento de controladores para sistemas não-lineares é extremamente difícil, mesmo com definições determinísticas onde as equações que regem a dinâmica do sistema são totalmente conhecidas [1]. Porém, este tipo de redes oferecem a potencialidade de endereçar problemas de controlo mais amplos. Desta forma, estas são utilizadas na indústria aeroespacial, automóvel, bancária, de defesa, eletrônica, de entretenimento, financeira, de seguros, de fabrico, petróleo e gás, robótica, telecomunicações, e de transporte.

1.2. OBJETIVOS

O objetivo deste trabalho de dissertação é usar a tecnologia das redes neuronais para a implementação em sistemas de controlo. Estes sistemas deverão modelar processos não-lineares e serem capazes de os controlar de forma eficaz face a uma certa entrada.

O estudo realizado tem em vista uma comparação de desempenhos entre diferentes topologias de controlo, utilizando, de alguma forma, tanto as redes neuronais como o controlador PID. Porém, os desempenhos obtidos não tem em vista apenas a comparação entre diferentes sistemas de controlo. Estes devem também ser comparados com um sistema do controlador PID em realimentação. No final, deve ser percetível se a introdução de redes neuronais como complemento a este controlador, melhora ou não a resposta dada pelo sistema.

1.3. CALENDARIZAÇÃO

Na Figura 1 pode ser encontrado um conjunto de tarefas elaboradas e que serviram como base à realização deste projeto.

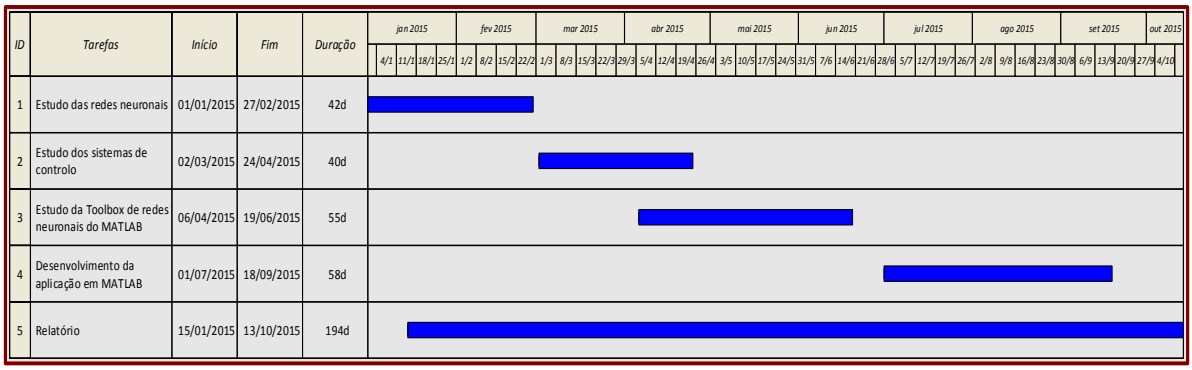


Figura 1 Calendarização do projeto

1.4. ORGANIZAÇÃO DO RELATÓRIO

No Capítulo 1 é apresentada uma breve contextualização às redes neuronais, assim como os objetivos do trabalho, a sua calendarização e a organização.

No Capítulo 2 é apresentado o estado da arte das redes neuronais: a sua história, biologia adjacente, e os aspetos técnicos envolvidos nas redes neuronais artificias, como por exemplo, o modelo de um neurónio e regras de aprendizagem. Por fim são descritas algumas aplicações práticas para diferentes tarefas e áreas.

No Capítulo 3 são apresentados diferentes sistemas de controlo aplicáveis, bem como um pequena introdução ao *software* utilizado para desenvolver os sistemas de controlo de redes neuronais.

No Capítulo 4 são descritos os sistemas dos controladores utilizados. Os sistemas apresentados no capítulo anterior e escolhidos para serem implementados são posteriormente aplicados no estudo desenvolvido. Nesse capítulo são explicitados todos os pormenores desse desenvolvimento.

No Capítulo 5 são expostos os testes realizados e os seus resultados. Além disso, são também efetuadas comparações entre os desempenhos dos diferentes sistemas de controlo implementados.

No Capítulo 6 é apresentado um resumo do trabalho desenvolvido, as respetivas conclusões, e perspetivas futuras.

2. ESTADO DA ARTE DAS REDES NEURONAIS

As RNA são um método de processamento de informação inspirado na forma como o sistema biológico nervoso, mais propriamente o cérebro, processa informação e adquirem conhecimento através da experiência. Assim, uma rede neuronal consiste na implementação de uma rede de circuitos que simule o funcionamento do cérebro humano. Sendo esta rede capaz de processar informação, reconhecer erros, e aprender com esses mesmos erros e adaptar-se.

Uma RNA é composta por um número variável de elementos de processamento (neurónios) e pela respetiva rede de ligações direcionadas e pesos associados. Cada unidade processa uma função de um número limitado de saídas de outros neurónios. Essas saídas são pesadas e tornam-se entradas da unidade atual.

Uma particularidade destes sistemas é o facto de ser capaz de aprender através de exemplos. Cada problema específico é diferente, sendo que cada um dos processos associados têm respostas diferentes para a mesma entrada. As redes neuronais ajustam os variados parâmetros inerentes para aproximar o sistema o mais possível ao desejado.

A estimação ou aproximação de funções que dependem de um grande número de entradas são, na maior parte, o tipo de problema que requerem a utilização deste tipo de sistemas, pois não podem ser resolvidas por nenhum algoritmo. Assim, pode-se dizer que as RNA são um tipo de sistemas de processamento não-linear. Estas são também usadas para resolver problemas como: reconhecimento de formas; classificação e tratamento de sinais, e previsões baseadas na análise historial.

Neste capítulo, é apresentado o estado da arte das redes neuronais, envolvendo todos os fundamentos adjacentes. Inicialmente será apresentada a história da evolução das redes neuronais ao longo dos anos, e a biologia inerente à inspiração da conceção destes sistemas.

Seguidamente é apresentado o modelo de neurónio artificial bem como as características que o definem. São também abordadas topologias de redes e métodos de aprendizagem das mesmas. Por fim, são enumeradas algumas aplicações das RNA, casos de estudo no qual foram implementadas com sucesso as redes neuronais.

2.1. HISTÓRIA

As raízes do trabalho com redes neuronais estão, de acordo com estudos da neurobiologia, datados de mais de um século atrás. Durante muitas décadas, biologistas especularam sobre o funcionamento exato do sistema nervoso. *Como se comportam os neurónios quando estimulados por diferentes magnitudes de corrente elétrica? Existe um limite mínimo (quantidade de corrente) necessária para ativar os neurónios?* Embora pudessem ser formuladas hipóteses, as respostas a estas e muitas mais outras perguntas não podiam ser dadas até meados do século XX, com o avanço da neurologia como ciência.

Nos anos 30, Nicolas Rashevsky deu início ao estudo da dinâmica neuronal, também conhecida como teoria do campo neuronal, representando a ativação e propagação na rede neuronal em termos de equações diferenciais [2].

2.1.1. PRINCÍPIO

No início dos anos 40, e seguindo os estudos inicias desenvolvidos pelo seu orientador (Nicolas Rashevsky), Walter Pitts (Figura 2), em conjunto com Warren McCulloch, projetou o que são geralmente consideradas como as primeiras redes neuronais [3]. Estes investigadores reconheceram que a combinação de vários neurónios simples em sistemas neuronais eram a fonte para um aumento do poder computacional. Os pesos são definidos

para que o neurónio realize uma função lógica simples, com diferentes neurónios realizando diferentes funções. Os neurónios podem ser dispostos numa rede que produz uma qualquer saída que possa ser representada por uma combinação de funções lógicas. O fluxo de informações da rede assume um intervalo de tempo de uma unidade para o sinal viajar de um neurónio para o seguinte.

Figura 2 Walter Pitts [4]

Uns anos mais tarde, impulsionados pelo primeiro desenvolvimento, um segundo trabalho dos mesmos investigadores abordara questões que ainda hoje são importantes áreas de investigação, tais como o reconhecimento de padrões, entre outras [5].

Em 1949, Donald O. Hebb, psicólogo da McGill University, projetou a primeira regra de aprendizagem para RNA [6]. A regra implica que se dois neurónios estivessem ativos simultaneamente, então a força (ou peso) da sua ligação deveria ser aumentada. Hebb assumia como verdadeira a sua declaração, mas devido à ausência de investigação neurológica ele não foi capaz de a verificar. Contudo, seria mais tarde confirmada a veracidade da sua afirmação e feitos melhoramentos a esta regra.

Em 1950, o neuropsicólogo Karl Lashley defendeu a tese que o armazenamento da informação do cérebro é realizado como um sistema distribuído. A sua tese era baseada em experiências com ratos, onde apenas a destruição extensa e não localizada dos tecidos nervosos influenciavam o desempenho dos ratos a descobrir a saída de um labirinto.

2.1.2. ERA DE OURO

A era do ouro da redes neuronais iniciou-se em 1951, quando Marvin Minsky desenvolveu um computador neuronal chamado de Snark, que era capaz de ajustar os pesos das ligações automaticamente. Contudo, não teve grande sucesso e nunca foi implementado na prática [7]. Durante as duas décadas seguintes a área das redes neuronais foi sendo explorada e inovações foram desabrochando ao longo destes anos.

No MIT (*Massachusetts Institute of Technology*), Frank Rosenblatt, Charles Wightman e seus colaboradores desenvolveram com sucesso o primeiro computador neuronal, denominado de MARK I PERCEPTRON. Este era capaz de reconhecer simples valores numéricos através da uma imagem *pixel* de 20x20. Este perceptrão consistia numa camada de entrada (retina) ligada aos neurónios associados com pesos ajustáveis representados por potenciómetros. A regra de aprendizagem do perceptrão usa um ajuste iterativo dos pesos mais eficaz que a regra de Hebb. A sua aprendizagem provou convergir para os pesos corretos que permitem resolver o problema.

O sucesso deste computador conduziu a um entusiasmo momentâneo. Isto porque a prova matemática da convergência do processo iterativo de aprendizagem seguia limitações sobre o que o tipo de rede do perceptrão podia aprender.

Bernad Widrow e Marcian E. Hoff introduziram, em 1960, o ADALINE (*ADAptive LInear NEuron*) [8]. Um sistema rápido e preciso de aprendizagem que foi a primeira rede neuronal amplamente usada comercialmente, em particular, nos telefones analógicos. Esta regra é geralmente designada por erro quadrático ou regra Delta, e está relacionada com a regra de aprendizagem do perceptrão. Uma vantagem da regra Delta sobre o algoritmo de aprendizagem é a sua adaptabilidade, ou seja, no caso de a diferença entre a saída do sistema e a saída desejada ser grande, os pesos das ligações também têm grandes alterações. Isto significa que na regra delta grandes diferenças originam grandes alterações, e vice-versa. Na altura Hoff, mais tarde fundador da Intel, era um estudante doutorado de Windrow, sendo que ele próprio é reconhecido como o inventor dos microprocessadores modernos.

No livro *Learning Marchines*, Nils Nilsson ofereceu uma visão geral do progresso e obras deste período de investigação das redes neuronais. Assumiram-se os princípios básicos da "auto-aprendizagem" e, portanto, de um modo geral, sistemas "inteligentes" já tinham sido

descobertos. Hoje em dia esta suposição parece ser uma sobrestimação excessiva, mas, nessa altura, permitiu uma grande popularidade e fundos para pesquisas posteriores [5].

Porém, foi publicado em 1969 por Marvin Minsky e Seymour Papert [9] uma análise precisa do perceptrão, mostrando que o seu modelo não era capaz de representar muitos problemas importantes. Esta implicação pôs em causa toda a investigação desenvolvida até então, resultando num declínio nos fundos durante os 15 anos seguintes.

2.1.3. ANOS SILENCIOSOS

Tal como já foi dito, os fundos para investigação sofreram uma grande diminuição. A investigação individualizada continuava, dando azo a paradigmas das redes neuronais. Contudo, estes não eram tratados nem conversados pois não existiam conferências ou eventos, e muito poucas publicações foram feitas durante este período.

Em 1972, Teuvo Khonen introduziu um modelo de associação linear, uma memória associativa [10]. Este modelo viria mais tarde a sofrer desenvolvimentos, e a ser aplicado a sistemas de reconhecimento de voz e composição de música. Ainda no mesmo ano, o mesmo modelo foi apresentado independentemente do ponto de vista de um neurofisiologista chamado James A. Anderson [11].

Em 1974, na sua dissertação em Harvard, Paul Werbos desenvolveu um procedimento de aprendizagem chamada retropropagação do erro [12], mas só uma década mais tarde é que este método atingiu a importância que tem hoje em dia.

De 1976 a 1980, Stephen Grossberg, em conjunto com muitos dos seus colegas e coautores, apresentou várias dissertações onde inúmeros modelos de redes neuronais são analisados matematicamente. Além disso, ele dedicou-se ao problema de manter a capacidade de uma rede neuronal aprender sem destruir as associações já conhecidas. Isto levou ao desenvolvimento de modelos da teoria de ressonância adaptativa (ART).

Em 1982, Teuvo Kohonen desenvolveu os mapas Kohonen que usava uma estrutura topológica para unidades fragmentadas [13]. Ele procurava por mecanismos que envolviam auto-organização no cérebro.

John Hopfield também inventou as chamadas redes Hopfield [14] que são inspiradas pelas leis do magnetismo. Estas redes são baseadas em pesos fixos e ativações adaptativas. Estas

não eram amplamente usadas em aplicações técnicas, mas fez com que a área das redes neuronais fosse lentamente ganhando importância.

2.1.4. RENASCENÇA

Duas razões para os "anos silenciosos" da década de 70-80, foram os fracassos do perceptrão de uma camada para resolver problemas como a função XOR e a falta de um método geral de aprendizagem para a rede multicamada. O método de retropropagação (anteriormente descoberto por Werbos) foi divulgado por David Parker e LeCun, antes de ser amplamente conhecido. O trabalho de Parker chamou a atenção de um grupo liderado pelos psicólogos David Rumelhart, da Universidade da California, e James McClelland, da Universidade de Carnegie-Mellon, que aperfeiçoaram e publicaram este método [5].

As deficiências detetadas de uma rede auto-organizada foram ultrapassadas. Esta rede surgiu depois de, em 1975, Kunihiko Fukushima e os seus colegas nos laboratórios NHK em Tóquio, desenvolverem uma série de redes neuronais especializadas em reconhecimento de caracteres. Contudo, numa fase inicial, estas redes falhavam em reconhecer caracteres distorcidos. Falhas estas que foram resolvidas em 1983.

Um número amplo de investigadores estiveram envolvidos no desenvolvimento de redes neuronais não-determinísticas, ou seja, redes que têm pesos que são alterados com base numa função probabilística de densidade.

Outra razão da renovação do interesse nas redes neuronais é o aperfeiçoamento das capacidades computacionais. Como exemplos têm-se as redes neuronais óticas e implementações de criação de circuitos integrados através de transístores (VLSI).

A partir desta época, o desenvolvimento desta área de investigação tem sido grande, não podendo ser mais discriminada. Alguns resultados destes estudos aqui descritos são apresentados nas secções seguintes.

2.2. BIOLOGIA

Antes de ser abordada a parte mais técnica das redes neuronais, seria útil conhecer a biologia destas redes. Ao contrário das redes artificiais, os neurónios são células vivas com axônios e dendrites que formam interligações. Impulsos elétricos carregados com informação viajam através destes vários componentes.

2.2.1. SISTEMA NERVOSO

O sistema de processamento de informação (o sistema nervoso - Figura 3) consiste num sistema nervoso central e um periférico. O primeiro é a estrutura principal deste sistema. Este é o lugar onde a informação recebida pelos órgãos sensores é guardada e gerida. O segundo consiste em nervos situados no exterior do cérebro, excluindo a medula espinal. Esses nervos formam uma rede densa e ramificada que cobre todo o corpo humano.

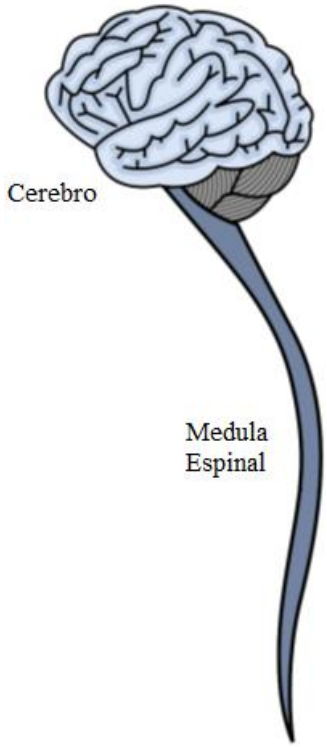


Figura 3 Sistema Nervoso Central [7]

Contudo, a área a focar é o sistema nervoso central, mais concretamente o cérebro. É possível dividi-lo em quatro áreas distintas de interesse para este trabalho (Figura 4) [7]:

- Telencéfalo – Divido em dois hemisférios ligados por vários cordões nervosos. Grande parte dos neurónios encontram-se no córtex cerebral, divididos em diferentes córtices, sendo que cada um destes campos desempenha uma função específica. Estes campos são responsáveis pelo processamento de informação qualitativa, como a gestão de diferentes perceções, ou seja, tratam processos de pensamento abstrato.

- Cerebelo – Encontra-se abaixo do Telencéfalo. Grande parte da coordenação motora é realizada nesta área do cérebro. Para o efeito, este tem uma informação sensorial direta sobre comportamentos musculares, bem como informações auditivas e visuais.

- Diencéfalo – Encontra-se dividido em duas partes. A primeira é mediadora entre o conjunto sensitivo e o motor, e o cérebro. Isto é, este decide que informação segue para o cérebro. A segunda parte do Diencéfalo controla o número de processo do corpo.

- Tronco cerebral (ou encefálico) – Liga o cérebro à medula espinhal, e é responsável pelo controlo dos reflexos, como o piscar dos olhos.

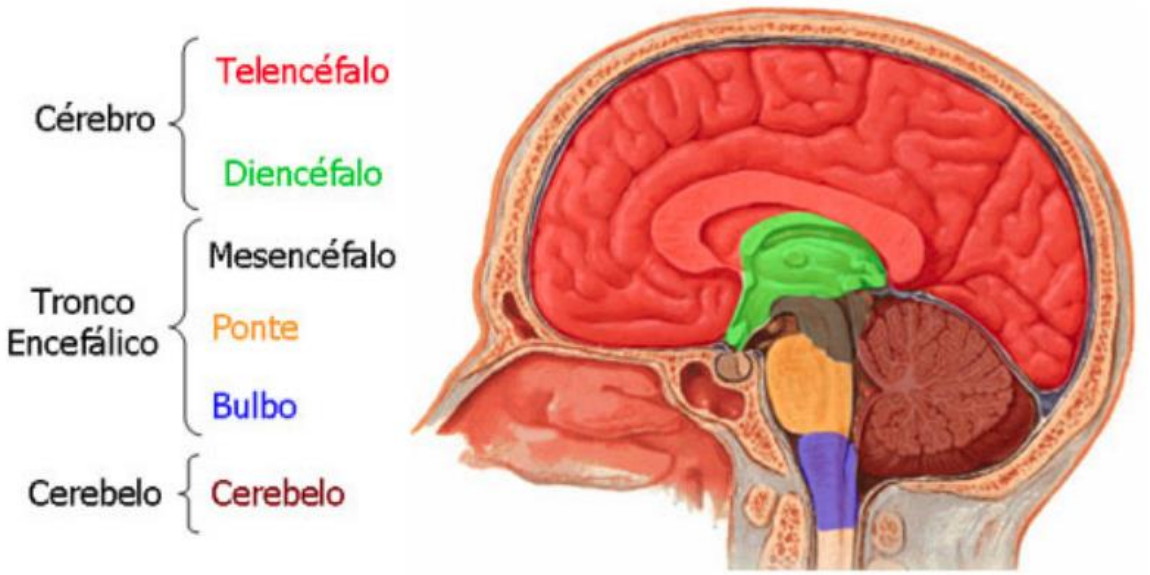


Figura 4 Divisões do cérebro [45]

2.2.2. NEURÓNIOS

A nível celular do cérebro encontram-se os neurónios. Estes podem desempenhar funções diferentes, que por sua vez conduzem a uma morfologia muito variável. Se se analisar o córtex humano sob um microscópio, pode-se encontrar vários tipos de neurónios.

Mais aprofundadamente, a estrutura geral de um neurónio é mostrada na Figura 5. Os ramos à direita são os canais de transmissão de informações de entrada e são chamados dendrites. Estes são como ramos para o núcleo do neurónio, e recebem os sinais elétricos de diferentes fontes, que são então transferidos para o núcleo da célula. A ramificação das dendrites pode ser também chamada de "árvore dendrítica". Os sinais de saída da célula são transmitidos posteriormente pelo axónio, que por sua vez o conduz para outros neurónios.

Estes quatro elementos, dendrites, sinapses, célula (ou corpo celular) e axônio, são a estrutura mínima que se adota a partir do modelo biológico. Os neurônios artificiais para a

computação têm canais de entrada, uma célula e um canal de saída. As sinapses são simuladas por pontos de contacto entre o corpo celular e ligações de entrada e saída, sendo associado um peso nesses pontos.

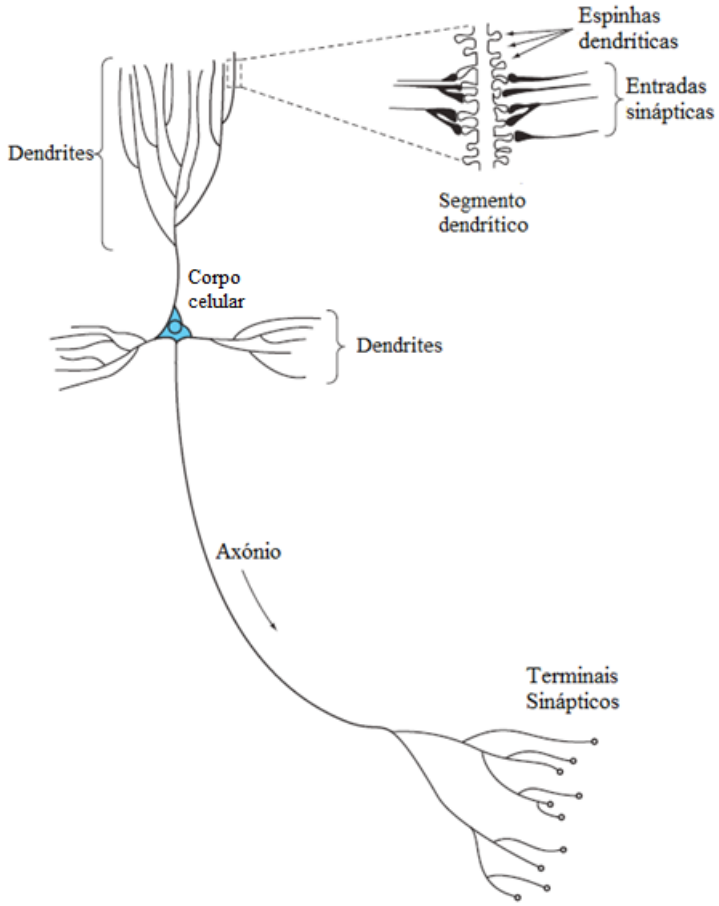


Figura 5 Estrutura de um neurónio [16]

Os neurônios transmitem informações utilizando os potenciais de ação (Figura 6). O processamento desta informação envolve uma combinação de processos elétricos e químicos, regulados na maior parte na interface entre os neurónios.

Os sinais de entrada de outros neurónios ou células são transferidos para o neurónio recetor por meio de ligações especiais – as sinapses. Estas ligações podem ser encontradas nas dendrites do neurónio, como é visível do lado direito da Figura 5. O processo de transferência de informação ocorre no espaço sináptico. Esse espaço separa eletricamente o lado pré-sináptico do pós-sináptico. Quando o impulso elétrico chega à sinapse do lado pré-sináptico, as vesículas sinápticas juntam-se à membrana celular. Estas contêm transmissores químicos – os neurotransmissores. Estes fluem para o espaço sináptico e alguns deles ligam-se aos canais iônicos. Se o transmissor for do tipo certo, os canais iônicos são abertos e mais iões

podem fluir do exterior para o interior da célula. O potencial da célula é assim alterado. Se o potencial do interior da célula for aumentado, isso provoca uma excitação da célula e ajuda a preparar um potencial de ação. Se os iões negativos forem transportados para dentro da célula, a probabilidade de se iniciar um potencial de ação é diminuída por algum tempo – sinapse inibitória.

Um aspeto fundamental é o facto de, comparativamente com o seu meio ambiente, os neurónios mostrarem uma diferença de carga elétrica, um potencial. Na membrada do neurónio a carga é diferente da carga exterior. Esta diferença é um conceito que é importante para entender os processos dentro do neurónio. Esta diferença é chamada de potencial da membrana, e é criada por diversos tipos de átomos (iões) que se dissolvem no fluido intracelular e extracelular, e se desagregam em iões positivos e negativos. Cloreto de sódio, por exemplo, associa-se em iões positivos de sódio (Na^+) e iões negativos de cloro (Cl^-). Outros iões positivos presentes no interior ou exterior das células são o potássio (K^+) e o cálcio (Ca_2^+). A permeabilidade é determinada pelo número e tamanho dos poros na membrana, os chamados canais iónicos. Estes têm formas e cargas permitindo certos iões atravessarem a membrana. A permeabilidade específica da membrana leva a diferentes distribuições de iões no interior e exterior da célula.

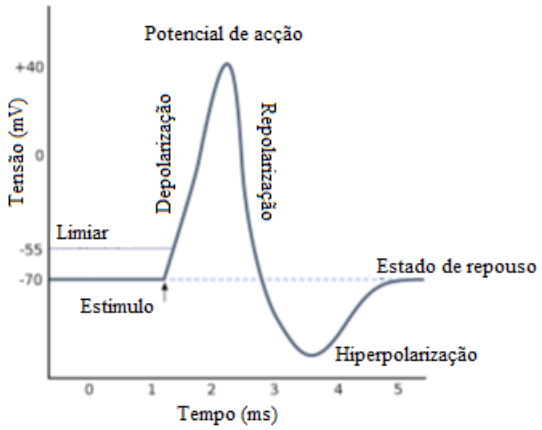


Figura 6 Potencial de ação ao longo do tempo [7]

Depois do núcleo da célula receber uma quantidade de sinais de ativação e inibição por sinapses através das dendrites, a célula acumula estes sinais. Logo que o conjunto de todos estes ultrapasse um valor limiar, o núcleo da célula do neurónio ativa um impulso elétrico que é então transmitido para os neurónios ligado ao neurónio atual, e são também abertos os canais iónicos. O valor limiar encontra-se a -55 mV. Logo que o estímulo recebido alcance

este valor, o neurónio é ativado e um sinal elétrico é iniciado (0). Abaixo são descritos os vários estados deste sinal.

- Estado de Descanso – Apenas um número limitado de canais de sódio e potássio são permeáveis. O potencial da membrana é de -70 mV e é mantido assim pelo neurónio.

- Estímulo até ao limiar: Um estímulo abre canais para a entrada de iões de sódio no exterior da membrana. O número de canais abertos deve ser suficiente para ultrapassar o valor limiar que se encontra a -55 mV. Isto pode ser conseguido pela chegada simultânea ou num curto intervalo de tempo de vários impulsos excitantes. Ultrapassado o valor limiar, são abertos os canais de sódio e inicia-se o potencial de ação.

- Despolarização: Predomina o "ambiente negativo" na célula, atraindo os iões de sódio do lado de fora da membrana compensando a depolarização da membrana. O influxo de sódio aumenta drasticamente o potencial de membrana.

- Repolarização: Os canais de sódio são fechados e os de potássio abertos. Os iões positivos deixam o ambiente "positivo" da célula. Além disso, a concentração intracelular é muito maior que a extracelular, aumentando ainda mais o fluxo.

- Hiperpolarização: Os canais de sódio e potássio estão fechados. Ao início, o potencial da membrana é ligeiramente mais negativo que o potencial de descanso. Depois de um período de 1 a 2 ms, o estado de descanso é restabelecido para que o neurónio possa reagir a um novo estímulo com um potencial de ação.

O pulso é transferido para outros neurónios através do axónio. Este é uma extensão longa e fina do núcleo da célula, e é eletricamente isolado de modo a atingir uma melhor condução do sinal elétrico. Este conduz os sinais até às dendrites, local onde é transferida a informação para os neurónios posteriores. E assim recomeça todo este processo. Na Figura 7 é mostrado como é uma sequência de impulsos a serem transmitidos.

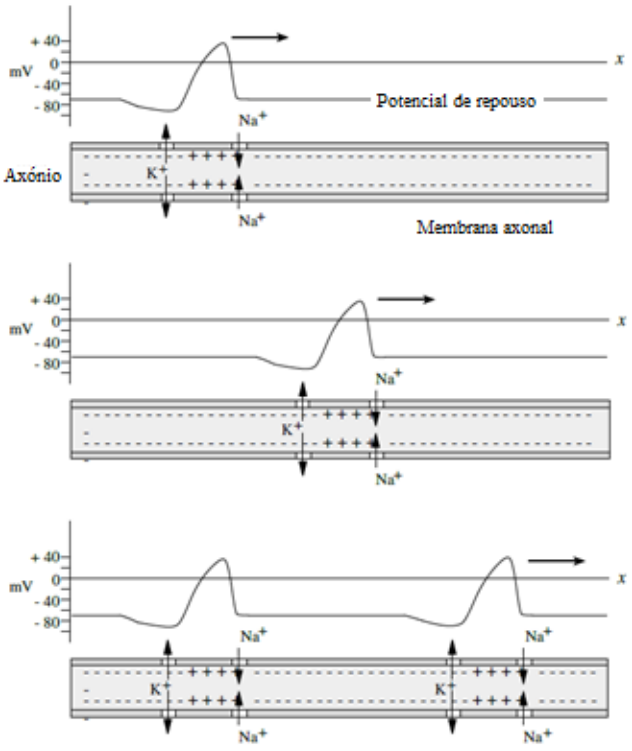


Figura 7 Transmissão de um potencial de ação [15]

2.2.3. APRENDIZAGEM

Nas redes neuronais, a informação é armazenada nas sinapses. Outras formas de armazenamento de informação podem estar presentes, mas estas estão ainda por ser descobertas ou não são bem entendidas. A eficiência da sinapse em induzir a despolarização da célula pode ser aumentada se forem abertos mais canais iónicos. Há cerca de uma década que os recetores NMDA têm sido estudados, pois apresentam algumas propriedades que poderiam ajudar a explicar algumas formas de aprendizagem nos neurônios [15].

Os recetores NMDA são canais iónicos permeáveis a diferentes tipos de moléculas. Estes canais estão bloqueados por um ião magnésio, para que a permeabilidade ao sódio e cálcio seja baixa. Se a célula é exposta a um certo nível de estímulo, os canais iónicos perdem o ião magnésio e são desbloqueados. A permeabilidade dos iões de cálcio aumenta imediatamente, e devido ao fluxo de iões de cálcio, uma cadeia de reações é iniciada, que produz uma mudança perdurável do nível do limiar da célula.

Os recetores NMDA são apenas um dos mecanismos usados pelos neurónios para aumentar a sua plasticidade, isto é, a sua capacidade de adaptação a novas circunstâncias. Através da modificação da permeabilidade da célula, esta pode ser treinada para ativar com maior

frequência diminuindo o limiar. Estes recetores também oferecem uma explicação para o fenômeno para o qual as células que não são estimuladas a ativar, tendem a ter um limiar mais elevado. A informação armazenada é atualizada periodicamente, de modo a manter a permeabilidade ideal da membrana celular [15].

2.3. MODELO DE NEURÓNIO

Tal como já foi visto, o neurónio é uma unidade de processamento de informação. A Figura 8 mostra o modelo de um neurónio artificial, que é a base de uma RNA. Podem-se identificar os seguintes elementos:

- Um conjunto de sinapses, ou ligações de entrada. Cada uma caracterizada por um peso próprio. Um sinal x_j da entrada j que está ligado ao neurónio k é multiplicado pelo peso ω_{kj}. Ao contrário dos pesos observáveis no cérebro, o peso da ligação do neurónio artificial tem um alcance que inclui tanto valores positivos como negativos.

- Um somatório que junta todos os sinais pesados pelas respetivas ligações sinápticas do neurónio. Aqui é incluído um valor de polarização aplicado externamente à soma.

- Uma função de ativação para limitar a amplitude da saída do neurónio. Geralmente, o intervalo da amplitude da saída é de [0,1] ou [-1,1].

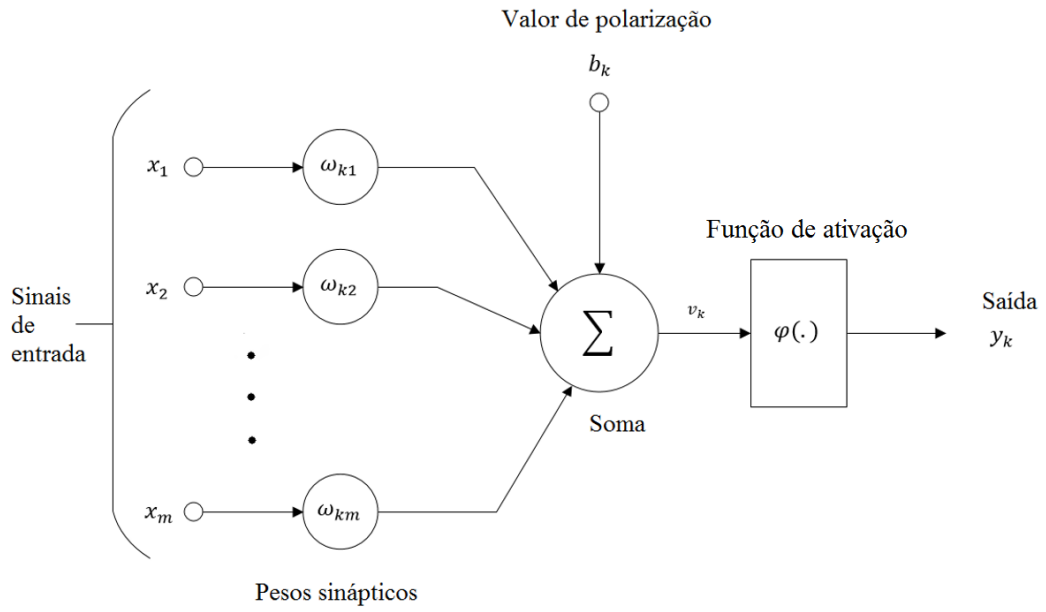


Figura 8 Modelo de um neurónio [16]

O modelo da Figura 8 inclui também um *bias*, definido por b_k. O valor de polarização tem o efeito de diminuir ou aumentar o limiar da função da ativação, isto é, define o valor mínimo que a acumulação dos sinais de entrada devem alcançar para o neurónio emitir uma saída. Pode-se então dizer que o limiar será o valor negativo do valor de polarização.

Em termos matemáticos, pode-se descrever o neurónio k pelo seguinte par de equações:

$$u_k = \sum_{j=1}^{m} \omega_{kj} x_j \tag{1}$$

$$y_k = \varphi(u_k + b_k) \tag{2}$$

Na primeira expressão, o conjunto de valores x_1, x_2, …, x_m são sinais de entrada, e o conjunto ω_{k1}, ω_{k2}, …, ω_{km} são os pesos das ligações de entrada do neurónio k. Na segunda expressão, tem-se o valor u_k, que é a saída do combinador linear dos sinais de entrada, b_k, que é o valor de polarização, $\varphi(.)$, que é a função de ativação, e y_k, que é o valor de saída do neurónio. A utilização do valor de polarização tem o efeito de aplicar uma transformação à saída do combinador linear, u_k, como mostrado na expressão a seguir:

$$v_k = u_k + b_k \tag{3}$$

Dependendo do valor de polarização, quer seja positivo ou negativo, a relação entre o potencial de ativação, v_k, e a saída do combinador linear, u_k, é alterada de forma exibida na Figura 9.

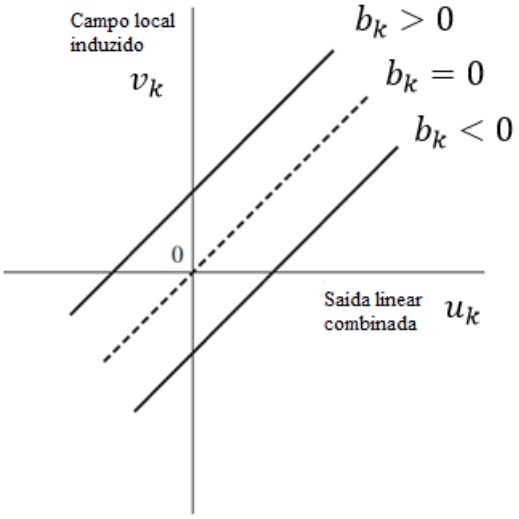


Figura 9 Transformação produzida pela presença do valor de polarização [16]

O valor de polarização é um parâmetro externo do neurónio k. Existem várias formas de integrar este valor. Uma delas já foi explicada acima. De forma equivalente, é possível formular uma combinação das expressões acima da seguinte maneira:

$$v_k = \sum_{j=0}^{m} \omega_{kj} x_j \tag{4}$$

$$y_k = \varphi(v_k) \tag{5}$$

A diferença é que, na primeira destas expressões, é adicionado uma nova sinapse ou entrada. Isto é:

$$x_0 = +1 \tag{6}$$

Com um peso:

$$\omega_{k0} = b_k \tag{7}$$

Assim, é possível reformular o modelo acima apresentado, de forma a considerar o valor de polarização, com as seguintes alterações: é adicionada uma nova entrada fixa a $+1$, e o peso dessa entrada será o valor de polarização, b_k.

2.3.1. FUNÇÕES DE ATIVAÇÃO

Tal como já foi referido, a função de ativação define a saída do neurónio e é definida por $\varphi(v)$. Aqui são apresentadas algumas das funções mais usadas.

A primeira delas é a função limiar (Figura 10), ou função Heaviside. Esta é definida pela seguinte expressão:

$$\varphi(v) = \begin{cases} 1, v \geq 0 \\ 0, v < 0 \end{cases} \tag{8}$$

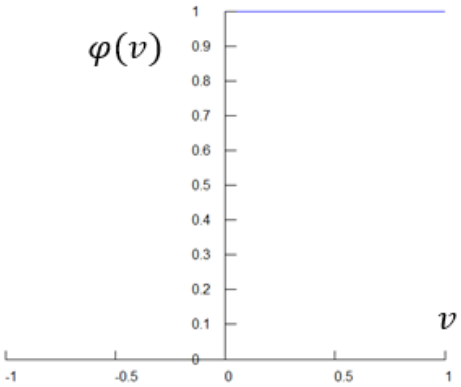


Figura 10 Função Heaviside

A saída do neurónio é igual a:

$$y_k = \begin{cases} 1, v_k \geq 0 \\ 0, v_k < 0 \end{cases} \tag{9}$$

Para este modelo de função aplicado, a saída do neurónio assume o valor 1 se o valor acumulado no somador for positivo e 0 no caso contrário. Esta afirmação descreve a prioridade tudo-ou-nada do modelo de McCulloch-Pitts. Este modelo pode ter em alguns casos um intervalo de valores diferentes:

$$\varphi(v) = \begin{cases} 1, v \geq 0 \\ -1, v < 0 \end{cases} \tag{10}$$

Este tipo de funções são usadas em esquemas de classificação binários. Isto é, quando se quer classificar a entrada em um de dois grupos. Outros dos usos é a criação de um conjunto pequeno de identificadores de características. Cada identificador seria uma pequena rede que geraria o valor 1 na saída se uma característica particular estivesse presente, e 0 caso esta não esteja presente. Combinando múltiplos identificadores de características numa única rede, permitiria resolver problemas complexos de agrupamento ou classificação.

A segunda função é conhecida como Linear (Figura 11), e é definida pela seguinte expressão:

$$\varphi(v) = \begin{cases} 1, v \geq \frac{1}{2} \\ v + \frac{1}{2}, -\frac{1}{2} < v < \frac{1}{2} \\ 0, v \leq -\frac{1}{2} \end{cases} \tag{11}$$

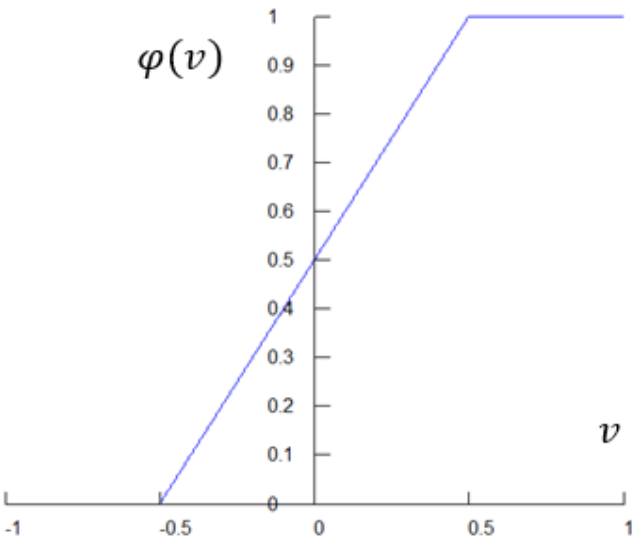


Figura 11 Função linear

O fator de amplificação na região linear da expressão é assumido como sendo igual à unidade, sendo que esta função pode ser vista como uma aproximação a um amplificador não-linear. As duas situações seguintes podem ser vistas como formas especiais desta função:

- É um combinador linear (somador), se a região linear de operação for mantida sem ocorrer saturação.

- Esta função é reduzida a uma função Heaviside se o fator de amplificação da região linear for infinitamente grande.

A terceira função é a Sigmóide (Figura 12) ou logarítmica. O gráfico desta função tem forma de S, e é de longe a função de ativação mais usada na construção de redes neuronais. É definida como uma função estritamente crescente que demonstra um equilíbrio entre o comportamento linear e não-linear. A expressão abaixo é demonstrativa desta função:

$$\varphi(v) = \frac{1}{1 + e^{-av}} \tag{12}$$

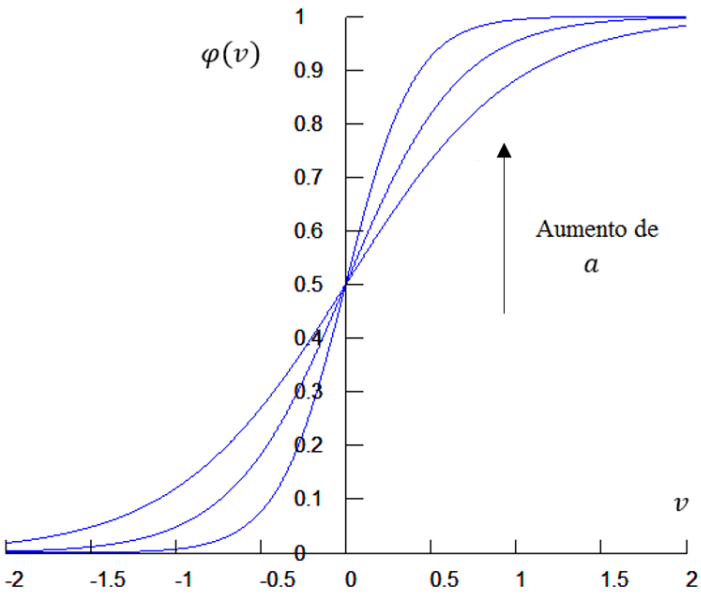


Figura 12 Função sigmóide

Onde a é o parâmetro de inclinação da função Sigmóide. Variando este parâmetro, é possível obter diferentes inclinações, como é visível na Figura 12. No limite, quando o parâmetro de inclinação se aproxima de infinito, a função torna-se uma simples função Heaviside. De notar que a função Sigmóide é diferenciável, enquanto a função Heaviside não o é.

No caso de ser necessário alterar o alcance de valores para o intervalo compreendido entre -1 e $+1$, é usada uma função formulada da Sigmóide – a função tangente hiperbólica. Esta permite assumir valores negativos que podem ser benéficos em termos práticos, e é definida pela seguinte expressão:

$$\varphi(v) = \tanh(v) \tag{13}$$

As funções de base radial ou gaussianas são funções de ativação com curvas em forma de sino, ou seja, com um campo local de resposta. A resposta desta função é não negativa para todos os valores de x. Contudo, esta diminui para 0 quando $|x - c| \to \infty$. Um exemplo habitual é a função ilustrada na Figura 13. A qual pode ser definida pela expressão:

$$f(x) = \exp(-x^2) \tag{14}$$

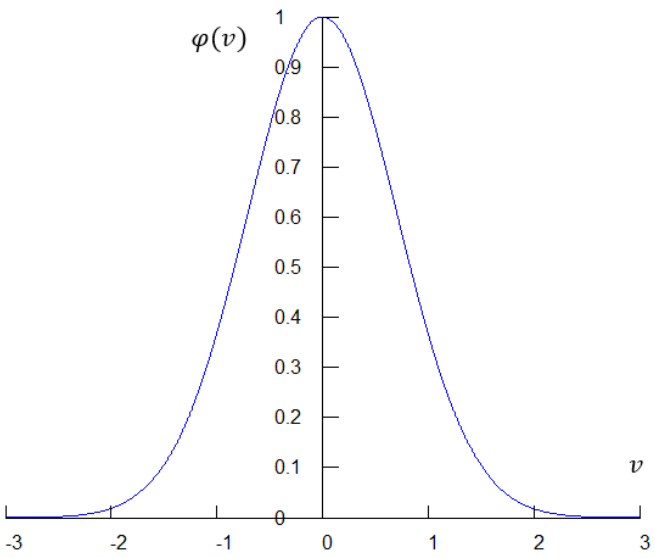


Figura 13 Função gaussiana

As redes de funções de base radial podem ser usadas para aproximação de funções e reconhecimento de padrões. Para este último caso, é possível interpretar a classe do padrão através da distância da entrada da rede a um valor de *x*.

2.4. ARQUITETURAS

Um único neurónio é insuficiente para muitos problemas práticos, e as redes com um grande número de neurónios são usadas com bastante frequência. A forma como estes estão ligados entre si determina como procedem os cálculos e constitui uma importante decisão do programador da rede neuronal. Esta estrutura de ligação está intimamente ligada com a regra utilizada para treinar a rede neuronal.

Também no sistema nervoso central existem partes estruturadas de diferentes formas. O córtex cerebral, onde se acredita ocorrer a maior parte do processamento de informação, é composto por cinco e sete camadas de neurónios com cada camada a fornecer as entradas da camada seguinte. No entanto, os limites não são rigorosos e existem ligações que atravessam várias camadas e também ligações de realimentação. Cada neurónio pode estar ligado a muitos outros, que são seus "vizinhos" dentro da mesma camada. A maioria destas ligações é excitatória, mas existe uma minoria de ligações inibitória. Outras delas são de autoexcitação indireta, isto é, a excitação de um neurónio provoca uma reação de excitação no neurónio "vizinho" que por sua vez excita o primeiro [2].

2.4.1. REDES TOTALMENTE CONECTADAS

Pode-se começar por introduzir o tipo arquitetura da qual todas as outras derivam. Esta denomina-se *fully connected* (totalmente conectadas). Tal como o nome indica, todos os neurónios se encontram ligados entre si, como é possível observar na Figura 14. Numa rede são geralmente definidos três tipos de neurónios: de entrada, de saída, ou invisíveis. Estes últimos são caracterizados por não serem visíveis diretamente pela entrada ou saída da rede. A sua função é de intervenção entre os neurónios de entrada e saída de uma maneira útil para o sistema. Outros neurónios podem ser de entrada e de saída.

Esta é a arquitetura de rede mais geral imaginável, e, tal como foi dito, todas as outras arquiteturas podem ser derivadas desta alterando o valor dos pesos das ligações. Isto é, por exemplo, definindo o peso com um valor nulo anula a ligação correspondente. Numa rede assimétrica totalmente conectada, a ligação de um neurónio ao outro pode ser diferente da ligação de sentido contrário. As redes com ligações com o mesmo valor nos dois sentidos denominam-se de redes simétricas totalmente conectadas. Estas são utilizadas normalmente para tarefas de memória associativa.

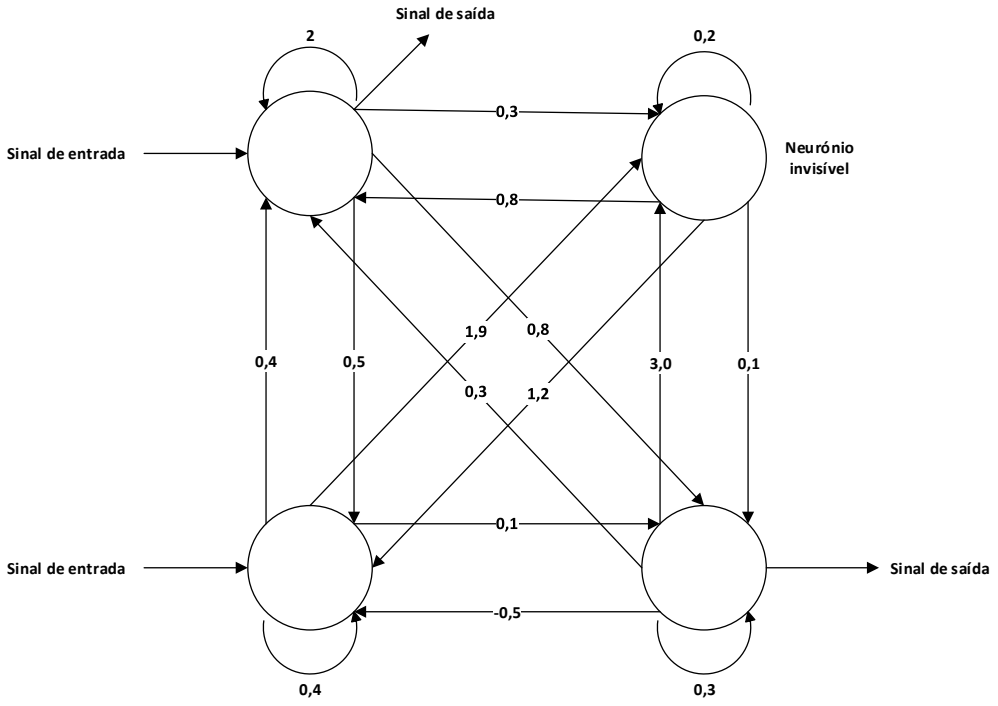


Figura 14 Rede totalmente conectada

Contudo, este tipo de estrutura é raramente usada devido ao grande número de parâmetros. Uma rede com *n* neurónios tem n^2 pesos. É difícil de conceber sistemas de aprendizagem

rápida que possam produzir uma rede totalmente conectada. Na prática, não existem casos onde cada neurónio tem influência direta sobre todos os outros neurónios, independentemente da distância.

2.4.2. REDES PROACTIVAS

Este é um dos tipos de redes em que os neurónios se encontram repartidos em subconjuntos, denominados de camadas. Existem diferentes tipos de ligações possíveis numa rede de camadas. Neste caso em particular, existe geralmente um agrupamento de neurónios com as seguintes camadas: uma camada de entrada, *n* camadas invisíveis, e uma camada de saída. Este tipo de redes – redes proactivas multicamada – são caracterizadas por ligações entre camadas adjacentes, isto é, os neurónios pertencentes a uma camada só podem estar ligados aos neurónios da camada que o antecede e sucede.

Na Figura 15 tem-se uma rede proactiva, e é descrita com a seguinte sequência de números: 3-2-3-2. Ou seja, tem três neurónios na camada de entrada, dois neurónios na primeira camada invisível, três neurónios na segunda camada invisível, e dois neurónios na camada de saída.

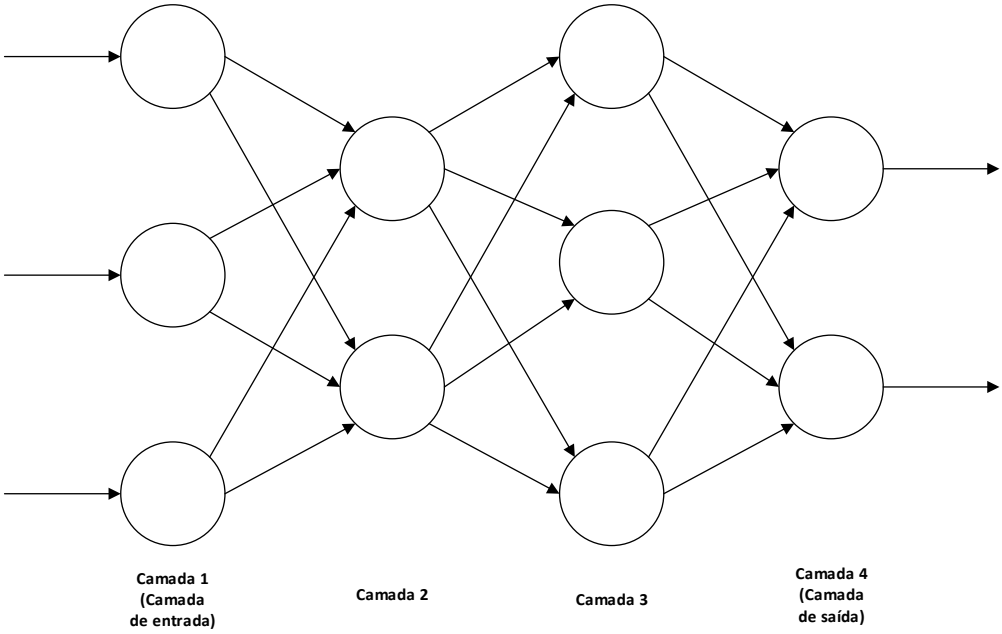


Figura 15 Rede proactiva multicamada

Os neurónios na camada de entrada abastecem a camada seguinte (camada invisível) com o padrão de ativação dos mesmos. Os sinais de saída da segunda camada são usados como sinais de entrada para a terceira camada, e assim sucessivamente. Assim, os neurónios de

uma camada tem como sinal de entrada os sinais de saída da camada que a precede. O conjunto de sinais da última camada constitui a resposta da rede ao estímulo recebido na camada de entrada.

Estas redes têm normalmente não mais que quatro camadas, e estão entre as mais usadas. Tanto que para muitos utilizadores as "redes neuronais" englobam apenas redes proactivas [2].

É possível ainda existirem redes proactivas de camada única (Figura 16). Esta é a forma mais simples de uma rede de camadas, onde existe apenas uma camada de neurónios que está ligada diretamente tanto à saída quanto à entrada. É contabilizada apenas a camada composta pelos nós computacionais, e não pelos nós de entrada.

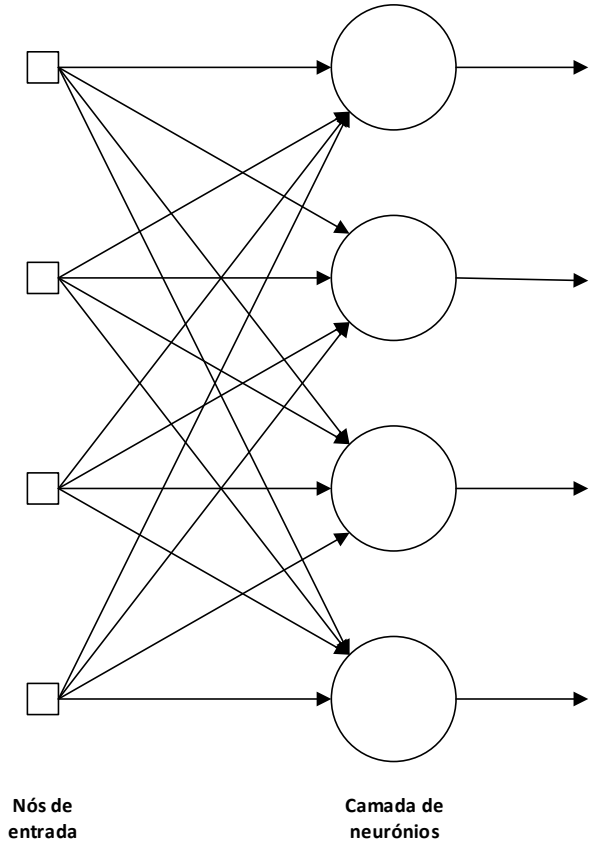


Figura 16 Rede proactiva de uma camada

2.4.3. REDES ACÍCLICAS

Existe ainda outra variante das redes proactivas. Algumas delas têm ligações que saltam um ou mais níveis. Por exemplo, pode existir uma ligação entre a camada de entrada e a camada de saída com várias camadas invisíveis pelo meio, não existindo assim a limitação de

ligações entre camadas sucessivas. Contudo, o sentido da ligação tem que ser conservado, não podendo um neurónio se ligar a outro que se encontre numa camada que antecede a sua. A esta subclasse dá-se o nome de redes Acíclicas.

Na Figura 17 está representada uma rede acíclica que contém duas ligações em camadas não sucessivas. Uma das ligações encontra-se entre neurónios da camada de entrada e da segunda camada invisível, e a outra entre a primeira camada invisível e a camada de saída.

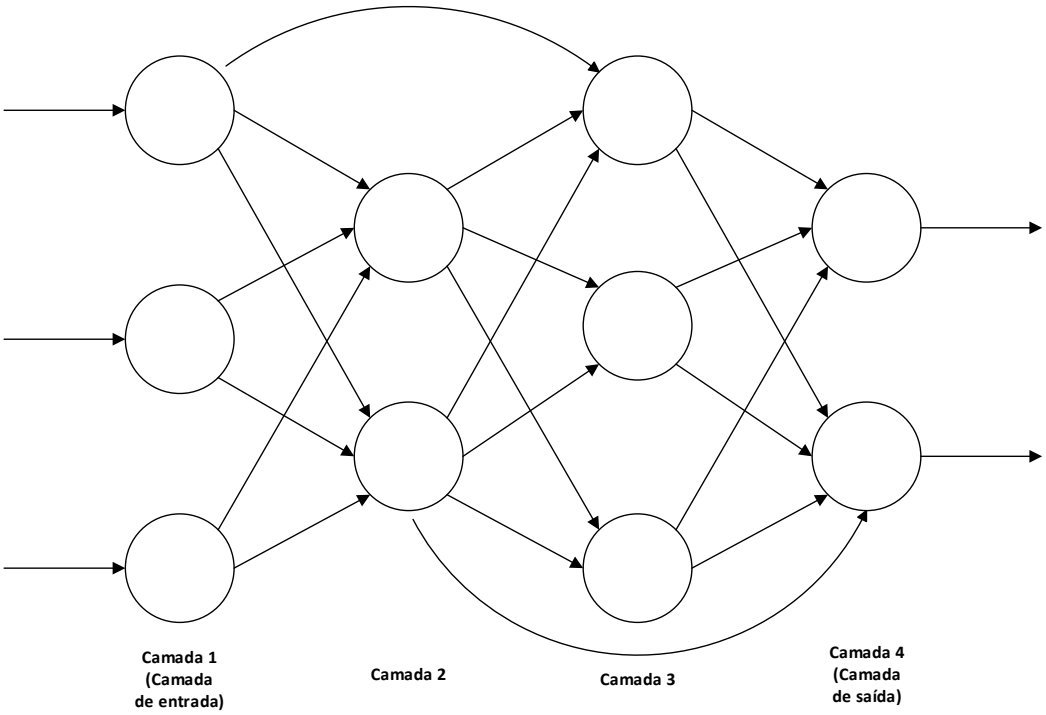


Figura 17 Rede acíclica

2.4.4. REDES RECURSIVAS

Este tipo de redes são uma subclasse das redes por camadas. Estas caracterizam-se pelo processo dos neurónios se influenciarem a si mesmos. Existem diferentes formas de obter este efeito.

A forma mais simples é a admissão de ligações a si próprios, isto é, um neurónio pode ter uma ligação direta entre a sua saída e entrada, como uma espécie de realimentação. A este método denomina-se de recorrência direta (a) – Figura 18), resultando numa autoestimulação de modo a atingir o ponto de ativação pretendido.

Outras redes usam o método da recorrência indireta. Este permite que hajam ligações adicionais entre um neurónio e as suas camadas anteriores. No caso b) da Figura 18 é demonstrado um exemplo, onde, por exemplo, o neurónio 4 tem ligações aos neurónios 1 e 2 que se encontram numa camada que antecede a sua. Ao influenciar a entrada dos neurónios das camadas anteriores, o neurónio está indiretamente a influenciar a sua própria entrada, daí o nome deste método.

No último caso (c) – Figura 18) é apresentado um exemplo de recorrência lateral. Tal como é visível na figura, existem ligações entre neurónios da mesma camada. Geralmente, cada neurónio inibe seus "vizinhos" e reforça-se. Como resultado apenas o neurónio mais forte fica ativo (esquema *winner takes-all*) [7]. Este esquema é usado para tarefas de especialização de padrões, onde um neurónio fica encarregue de identificar um padrão que se insira na subclasse correspondente à especializada pelo mesmo.

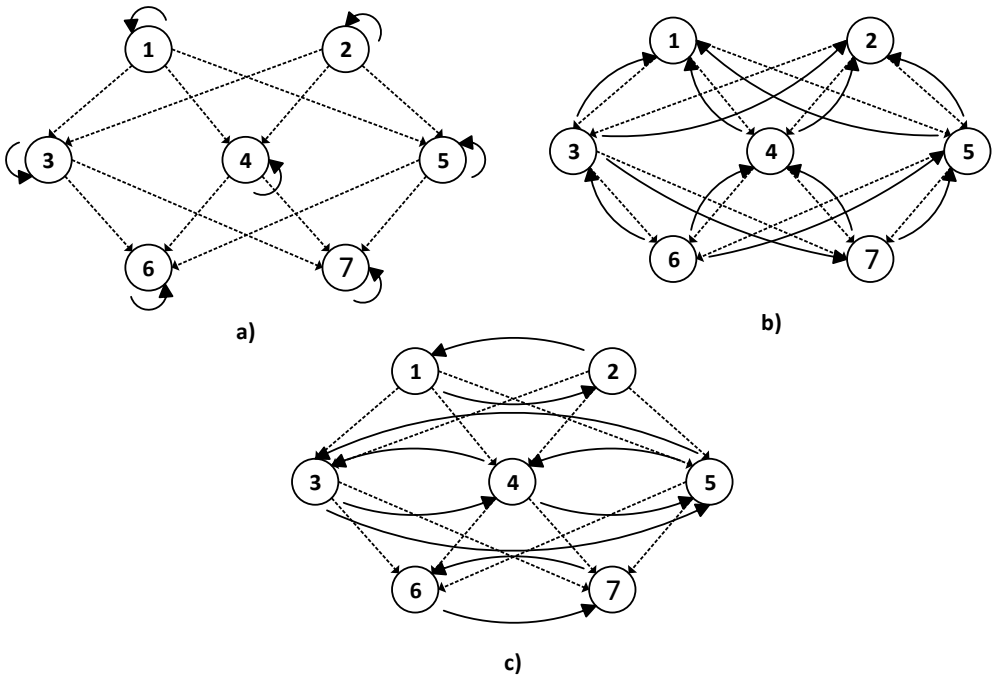


Figura 18 a) Recorrência direta; b) Recorrência indireta; c) Recorrência lateral

2.5. APRENDIZAGEM

De entre as interessantes propriedades de uma rede neuronal, a habilidade de aprender através do ambiente exterior e de melhorar a seu desempenho pela aprendizagem é a que se destaca. O processo da aprendizagem consiste numa adaptação dos parâmetros da rede neuronal através de um processo contínuo de estimulação por parte do ambiente no qual a

rede está inserida. Isto permite uma familiarização com o problema proposto de modo a ser capaz de resolver problemas da mesma classe – generalização.

Uma rede neuronal pode aprender de diferentes formas, por exemplo, criando ou excluindo ligações entre neurónios. Contudo, quase todas estas podem ser conseguidas através de modificações aplicadas aos pesos das sinapses e aos limiares dos neurónios. Sendo que estes últimos podem também ser definidos como pesos, como foi abordado na secção 2.3. Assim, a aprendizagem da rede neuronal efetua-se através da alteração dos pesos das ligações de acordo com as regras que podem ser formulados como algoritmos.

A definição do processo de aprendizagem implica a seguinte sequência de eventos:

1. Estimulação pelo ambiente;
2. Alterações resultantes da estimulação;
3. Nova resposta, devido às alterações ocorridas na estrutura interna.

Um conjunto predeterminado de regras bem definidas para a solução de um problema de aprendizagem é chamado de regra de aprendizagem. Como seria de esperar, não há uma regra de aprendizagem exclusiva para uma rede neural. Em vez disso, existem várias ferramentas representadas por uma grande variedade de regras, cada uma das quais oferecendo vantagens específicas. Em resumo, os algoritmos de aprendizagem diferem uns dos outros na forma como o ajustamento para o valor $\Delta\omega_{kj}$ é formulado, sendo ω_{kj} o peso da ligação da entrada j para o neurónio k [17].

2.5.1. PROCESSOS DE APRENDIZAGEM

De modo geral, os processos de aprendizagem podem ser classificados através do seu funcionamento e interação com as redes neuronais: aprendizagem com supervisão (*Supervised Learning*) e sem supervisão. Da mesma forma, esta última forma de aprendizagem pode ser subcategorizada em aprendizagem não-supervisionada (*Unsupervised Learning*) e por reforço (*Reinforcement Learning*). Estas diferentes formas de aprendizagem são idênticas às da aprendizagem humana.

Na aprendizagem com supervisão (Figura 19), o ingrediente principal é disponibilidade de um supervisor que fornece à rede neural as respostas desejadas. Para o conjunto de respostas

que são providenciadas, existe um conjunto de valores de saída que são comparados. Os pesos da rede alteram de acordo com essa diferença, ou seja, de acordo com o erro existente. Este ajustamento é realizado iterativamente passo-a-passo. O objetivo é que a rede não só consiga associar independentemente entradas e saídas, mas também fornecer resultados plausíveis para padrões de entrada semelhantes.

Nesta categoria, estão inseridos os seguintes métodos de aprendizagem: Regra Delta ou aprendizagem por correção do erro, e a regra de retropropagação, que é uma versão generalizada da regra Delta. Estes métodos são descritos nas subsecções seguintes.

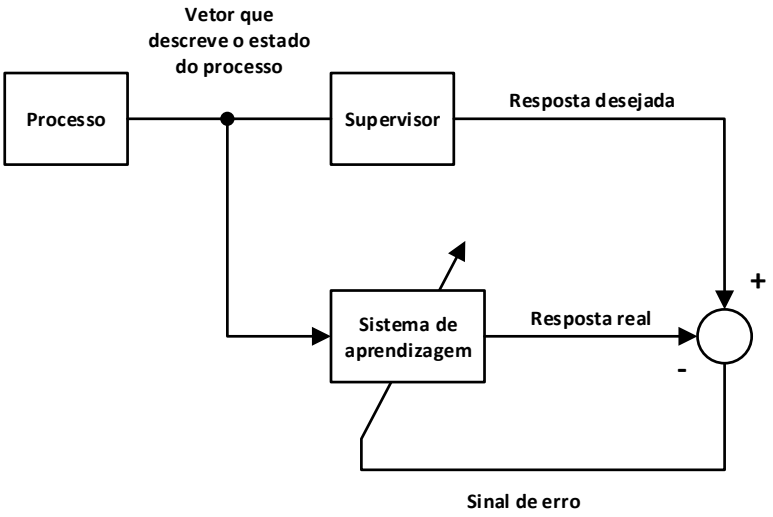


Figura 19 Aprendizagem com supervisão

A aprendizagem sem supervisão (Figura 20) é o método biológico mais plausível, mas não é adequado para todos os problemas. Tal como o nome indica, não existe supervisor que forneça amostras específicas da função a ser aprendida pela rede neuronal. O conjunto de resultados de treino apenas contempla os padrões de entrada, e a rede tenta por si detetar as semelhanças. Quando a rede ficar sintonizada com as regularidades estatísticas dos dados de entrada, esta desenvolve a capacidade de formar representações internas para características codificadas da entrada e, assim, cria classes automaticamente.

Para realizar este tipo de aprendizagem é usada uma regra de aprendizagem competitiva. Esta regra consiste numa camada de neurónios que competem entre si pela oportunidade de responder a características contidas no sinal de entrada. O neurónio com o maior valor total da entrada "ganha" a competição, e é o único a dar uma resposta. Todos os outros são desligados. É fácil de entender que para esta regra é usada a topologia de rede com recorrência lateral que usa o esquema *winner taskes-all*.

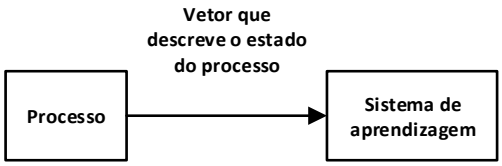


Figura 20 Aprendizagem sem supervisão

Na aprendizagem por reforço (Figura 21), é feito o mapeamento da entrada-saída através de uma interação contínua com o ambiente, tal como no método anterior. Contudo, o sistema é concebido para aprender sob um reforço atrasado, isto é, este observa a sequência de estímulos do ambiente (entrada), que eventualmente resulta na geração de um sinal heurístico de reforço, indicando se as repostas dadas à sequência de estímulos foram corretas ou não.

O objetivo deste tipo de aprendizagem é minimizar o custo acumulativo das ações sobre uma sequência de passos. Pode ser que certas ações tomadas anteriormente na sequência de passos temporais sejam as melhores no comportamento do sistema. A função deste tipo de sistemas é encontrar essas ações e voltar a alimentar o ambiente com as mesmas ações [16].

Contudo, este tipo de aprendizagem é difícil de realizar, não só porque não é fornecida a resposta desejável, mas também porque existe um atraso no sinal heurístico recebido. O que significa que o sistema de aprendizagem deve temporariamente atribuir crédito e culpa individualmente a cada componente para cada ação da sequência de passos, enquanto o reforço não avaliar o resultado.

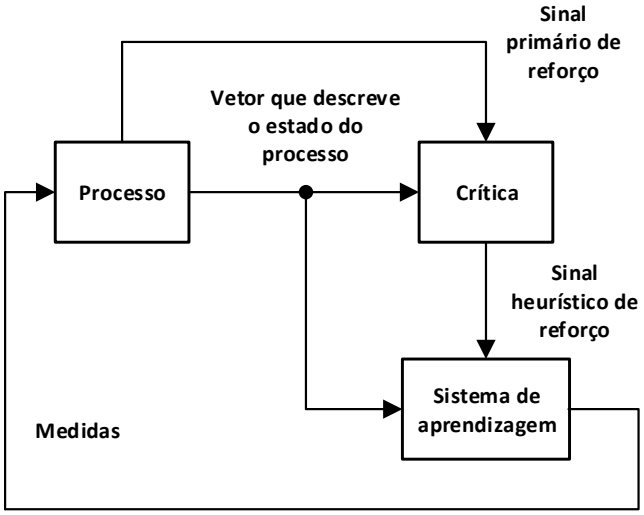


Figura 21 Aprendizagem por reforço

2.5.2. REGRA DE HEBB

Tal como já foi dito na secção sobre a história das redes neuronais, em 1949, Donald O. Hebb elaborou a regra de Hebb que é a mais antiga e famosa regra de aprendizagem, e a base de muitas outras regras usadas nas RNA.

A definição inicial desta regra é [17]:

1. Se dois neurónios de cada lado da ligação sináptica estão ativos simultaneamente (de forma síncrona), então a força da ligação da sinapse (peso) deve ser aumentada.

2. Se dois neurónios de cada lado da ligação sináptica estão ativos de forma assíncrona, então a força da ligação da sinapse de ser diminuída ou eliminada.

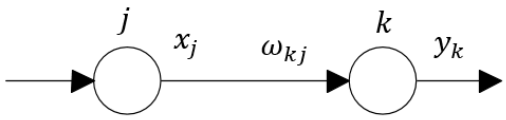


Figura 22 Ligação sináptica

Para formular a forma de aprendizagem acima descrita em termos matemáticos, considera-se o peso sináptico ω_{kj} com atividades pré e pós-sinápticas designadas por x_j e y_k, respetivamente. De acordo com o pressuposto de Hebb, o ajustamento aplicado ao peso sináptico ω_{kj} no tempo n é:

$$\Delta\omega_{kj}(n) = F(y_k(n), x_j(n)) \tag{15}$$

É possível usar ainda a regra do produto da atividade:

$$\Delta\omega_{kj}(n) = \eta y_k(n) x_j(n) \tag{16}$$

Onde η é uma constante positiva que determina a taxa de aprendizagem. Esta regra realça claramente a natureza relacional da sinapse Hebbiana.

Pela representação da sequência de equações (17), (18) e (19), é possível verificar que a aplicação repetida do sinal de entrada x_j leva a um crescimento exponencial do peso da ligação, e, eventualmente, à sua saturação.

$$\omega_{kj}(n+1) = \omega_{kj}(n) + \eta y_k(n) x_j(n) = \omega_{kj}(n)(1 + \eta x_j^2(n)) \tag{17}$$

$$\begin{aligned}\omega_{kj}(n+2) &= \omega_{kj}(n+1) + \eta y_k(n+1)x_j(n+1) \\ &= \omega_{kj}(n)(1+\eta x_j^2(n)))(1+\eta x_j^2(n+1)))\end{aligned} \tag{18}$$

Se x_j se mantiver constante, então:

$$\omega_{kj}(n+N) = \omega_{kj}(1+\eta x_j^2)^N \tag{19}$$

Para evitar que surjam situações como esta, é necessário impor um limite no crescimento dos pesos sinápticos. Um método que limita este crescimento é a introdução de um fator não-linear de esquecimento na fórmula de ajustamento ($\Delta\omega_{kj}(n)$). Esta fórmula é, assim, redefinida como a forma generalizada da regra de Hebb, e enunciada do seguinte modo:

$$\Delta\omega_{k,j}(n) = \eta y_k(n)x_j(n) - \alpha y_k(n)\omega_{k,j}(n) = \alpha y_k(n)\left[cx_j(n) - \omega_{k,j}(n)\right] \tag{20}$$

Onde $c = \frac{\eta}{\alpha}$. Se o peso $\omega_{kj}(n)$ aumentar até ao ponto (21), é alcançado um ponto de equilíbrio, e este pára de ser atualizado.

$$cx_j(n) - \omega_{k,j}(n) = 0 \tag{21}$$

2.5.3. REGRA DELTA

A regra Delta (Figura 23), ou regra de Windrow-Hoff, é uma regra de aprendizagem supervisionada, pois requere um sinal desejado para a resposta do sistema. Esta regra ajusta os pesos das ligações de modo a reduzir a diferença entre a saída da rede e a saída desejada. Isto resulta no menor erro quadrático (gradiente descendente). Esta é aplicada em redes de apenas uma camada e é a percursora da regra de retropropagação para redes com múltiplas camadas [5].

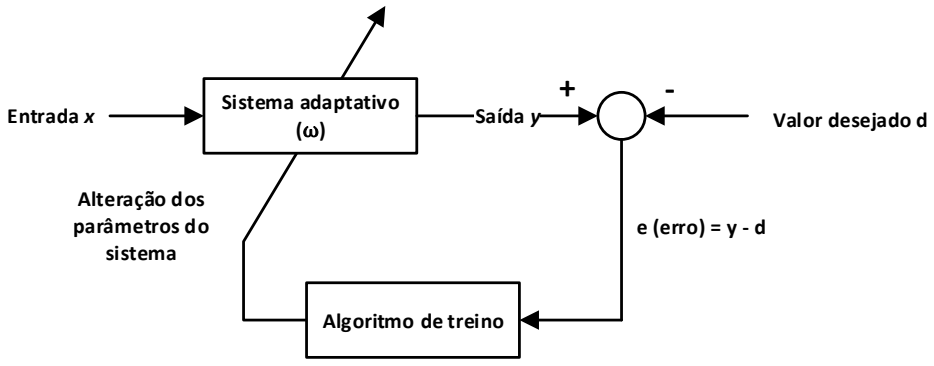


Figura 23 Diagrama da regra Delta

Assumindo uma resposta desejada $d_k(n)$ para o neurónio k no tempo n. Sendo o valor correspondente à resposta real do neurónio dado por $y_k(n)$. Esta reposta é produzida por um estímulo aplicado à entrada da rede no qual o neurónio k está associado. O vector de entrada e a resposta desejada constituem uma amostra particular apresentada à rede no tempo n.

Geralmente, a resposta real do neurónio é diferente da resposta desejada. Por isso, define-se o sinal de erro como:

$$e_k(n) = y_k(n) - d_k(n) \tag{22}$$

O critério usado para a função de custo é o erro quadrático:

$$E = \sum_k e_k^2(n) \tag{23}$$

A rede é depois otimizada minimizando E relativamente aos pesos sinápticos da rede. Assim, de acordo com esta regra, o ajustamento do peso sináptico é dado por:

$$\Delta\omega_{k,j} = \eta e_k(n) x_j(n) \tag{24}$$

2.5.4. REGRA DA RETROPROPAGAÇÃO

Esta é a regra mais usada nas aplicações que envolvem redes neuronais e de aprendizagem supervisionada. A ideia central por detrás desta solução é que os erros das unidades das camadas escondidas da rede sejam determinados pela retropropagação dos erros das unidades da camada de saída. Por esta razão, este método é chamado regra da retropropagação.

A aplicação da regra delta generalizada (retropropagação) envolve duas fases: na primeira, a entrada é apresentada e propagada pela rede para calcular o valor da saída para cada unidade de saída. Essa saída é comparada com o seu valor desejado, resultando no sinal de erro para cada unidade de saída. A segunda fase envolve uma propagação inversa ao sentido da rede (retropropagação) durante o qual o sinal de erro é passado a cada unidade da rede e as alterações aos pesos são calculadas. Este processo repete-se sucessivamente até que o critério de paragem seja satisfeito. Este critério poderá ser, por exemplo, um número máximo de iterações ou uma margem de erro máxima.

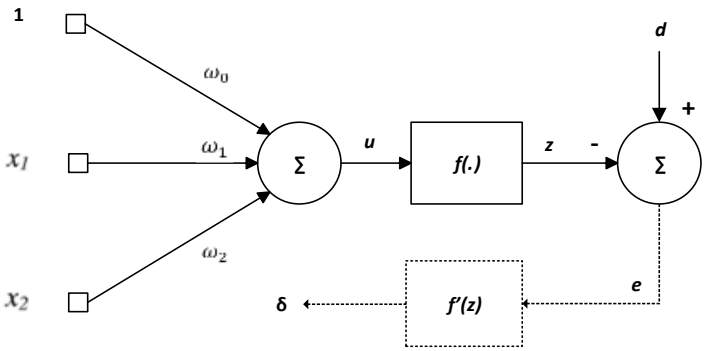


Figura 24 Exemplo de retropropagação para um simples neurónio

Considere-se primeiro um exemplo simples que consiste num único neurónio para ilustrar este procedimento. A Figura 24 representa um neurónio com duas partes distintas: uma unidade de soma para calcular a função *u*, e uma função de ativação $z = f(u)$. A saída *z* é comparada com o valor desejado, e a sua diferença (erro) será calculada. Existem duas entradas $[x_1\ x_2]$ com os pesos w_1 e w_2, respetivamente. A entrada marcada com o par composto pela constante 1 e o peso w_0 representa o termo limiar. A função da rede é calculada pela equação:

$$u = \sum_{i=0}^{2} \omega_i x_i = Wx \tag{25}$$

Onde $x_0 = 1$, $W = [\omega_0\ \omega_1\ \omega_2]$ é a matriz de pesos, e $x = [1\ x_1\ x_2]^T$ é o vetor de entrada.

Dado um conjunto de amostras, o treino da rede por retropropagação começa por alimentar todas as entradas e calcular a saída correspondente. O conjunto inicial de valores dos pesos representa a primeira tentativa de ajuste para o problema. Ao contrário de outros métodos, o uso desta técnica não depende do fato de fazer uma boa estimativa inicial. De seguida, é calculado a soma do erro quadrático correspondente à diferença do valor real e desejado.

$$E = \sum_{k=1}^{K} [e(k)]^2 = \sum_{k=1}^{K} [d(k) - z(k)]^2 = \sum_{k=1}^{K} [d(k) - f\big(Wx(k)\big)]^2 \tag{26}$$

O objetivo é minimizar o erro *E*, e a forma mais simples de o fazer é alterar as ligações da rede neuronal (pesos) de tal maneira que, na próxima iteração, o erro irá diminuir para este padrão em particular. Isto leva a um problema não-linear de otimização de erros quadráticos.

Existem vários algoritmos de otimização disponíveis para resolver estes problemas. Estes algoritmos adotam uma formulação iterativa semelhante:

$$W(t+1) = W(t) + \Delta W(t) \tag{27}$$

Onde $\Delta W(t)$ é a correção feita aos pesos correntes $W(t)$. Para determinar a direção no qual os pesos têm que ser alterados, calcula-se o valor negativo do gradiente E, ∇E, no que diz respeito aos pesos, ω_{ij}. A seguir, ajusta-se os valores dos pesos de modo que o erro total seja reduzido. Geralmente, pensa-se em E como sendo uma superfície no espaço dos pesos.

Na Figura 25 é apresentado um exemplo simples de uma rede com um peso. Neste caso, a superfície é bastante simples, mas à medida que as variáveis (pesos) vão aumentando esta vai tornando-se mais complexa. As alterações dos pesos devem ocorrer na direção do gradiente negativo, que é a direção do gradiente descendente da superfície num ponto. Além disso, as alterações dos pesos deve ser feita iterativamente até o E alcançar o ponto mínimo. Esse valor mínimo pode ser local ou global (Figura 25 – b)). Embora não desejável, é provável que o erro *E* estabilize num mínimo local. Contudo, existe uma forma de contornar este problema que será abordada mais à frente nesta subsecção.

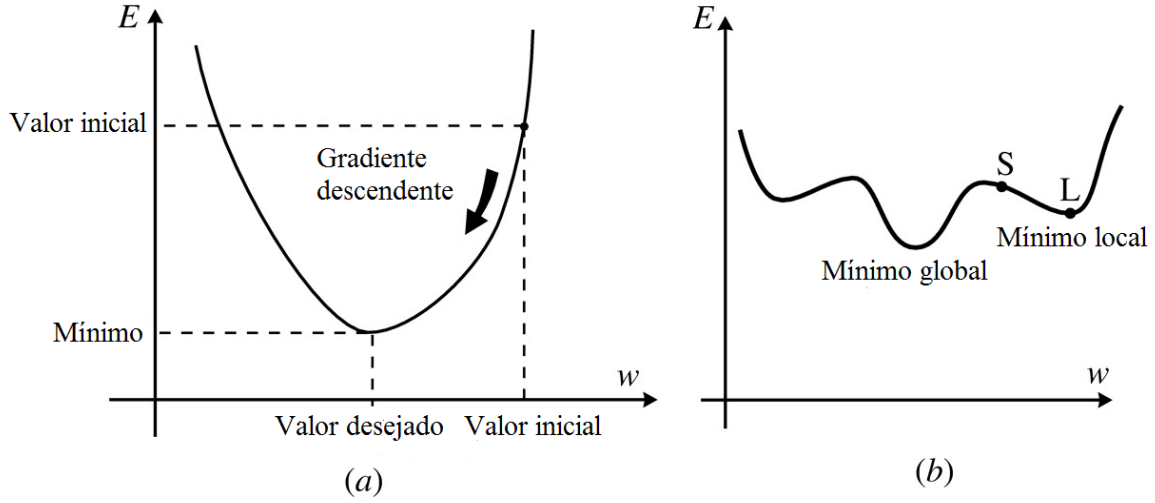


Figura 25 Gradiente descendente: a) Esquema de um exemplo da superfície do erro (sem mínimos locais); b) Esquema de uma superfície com mínimo local [46]

Acima foi descrito o algoritmo de gradiente descente, que é o utilizado na regra de retropropagação. A fórmula do gradiente pode ser representada por:

$$\Delta W(t) = -\eta g(t) = -\eta \frac{\partial E}{\partial W} \tag{28}$$

Onde g é o vetor gradiente, e η a taxa de aprendizagem. A derivada do escalar *E* em relação aos pesos individuais pode ser calculado da forma que se segue:

$$\frac{\partial E}{\partial \omega_i} = \sum_{k=1}^{K} \frac{\partial [e(k)]^2}{\partial \omega_i} = \sum_{k=1}^{K} 2[d(k) - z(k)](-\frac{\partial z(k)}{\partial \omega_i}) \; para \; i = 0,1,2 \tag{29}$$

Onde

$$\frac{\partial z(k)}{\partial \omega_i} = \frac{\partial f(u)}{\partial u}\frac{\partial u}{\partial \omega_i} = f'(u)\frac{\partial}{\partial \omega_i}\left(\sum_{j=0}^{2} \omega_j x_j\right) = f'(u)x_i \tag{30}$$

Assim,

$$\frac{\partial E}{\partial \omega_i} = -2\sum_{k=1}^{K} \delta(k)x_i(k) \tag{31}$$

Onde $\delta(k)$ é o sinal de erro $e(k) = d(k) - z(k)$ modelado pela derivada da função de ativação $f'(u(k))$ e assim representando a valor da correção que é necessária ser aplicada ao peso ω_i para a entrada $x_i(k)$. A mudança $\Delta\omega_i$ é assim a soma de todas as amostras de treino. Por isso, a fórmula de atualização do peso é descrita da seguinte forma:

$$\omega_i(t+1) = \omega_i(t) + \eta\sum_{k=1}^{K} \delta(k)x_i(k) \tag{32}$$

Se for usada uma função de ativação sigmóide, que é uma função diferenciável, $\delta(k)$ pode ser calculado da seguinte forma:

$$\delta(k) = \frac{\partial E}{\partial u} = [d(k) - z(k)]z(k)[1 - z(k)] \tag{33}$$

Onde $z(k)[1 - z(k)]$ corresponde à derivada da função sigmóide. Na Figura 26, a rede e a saída correspondente à amostra *k* de treino do neurónio *j* da camada *L–1* são indicados por $u_j^{L-1}(k)$ e $z_j^{L-1}(k)$, respetivamente. A saída alimenta o neurónio *i* da camada *L* através do peso sináptico $\omega_{ij}^{L}(t)$ ou ω_{ij}^{L}.

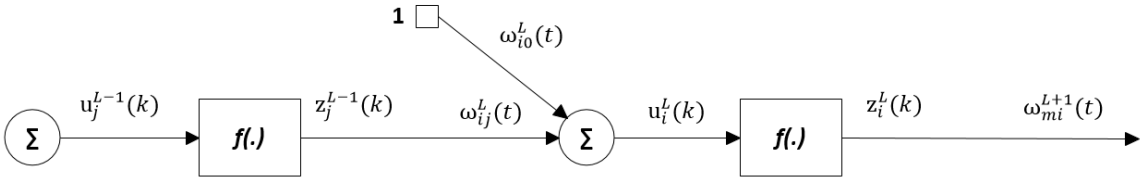


Figura 26 Modelo de neurónio

Para derivar a equação de adaptação dos pesos $\frac{\partial E}{\partial \omega_i}$ deve ser calculado:

$$\begin{aligned}\frac{\partial E}{\partial \omega_{ij}^{L}} &= -2\sum_{k=1}^{K}\frac{\partial E}{\partial u_i^{L}(k)}\frac{\partial u_i^{L}(k)}{\partial \omega_{ij}^{L}} = -2\sum_{k=1}^{K}\left[\delta_i^{L}(k)\frac{\partial}{\partial \omega_{ij}^{L}}\sum_{m}\omega_{im}^{L}z_m^{L-1}(k)\right] \\ &= -2\sum_{k=1}^{K}\delta_i^{L}(k)z_j^{L-1}(k)\end{aligned} \tag{34}$$

Nesta equação, a saída $z_j^{L-1}(k)$ pode ser avaliada aplicando a amostra *k* de treino à rede com pesos fixos a ω_{ij}^{L}. Contudo, o termo de erro delta $\delta_i^{L}(k)$ não está disponível e tem que ser calculado.

Sabendo que o erro delta é definido por $\delta_i^{L}(k) = \frac{\partial E}{\partial u_i^{L}(k)}$, é ilustrado na Figura 27 como é calculado iterativamente $\delta_i^{L}(k)$ de $\delta_m^{L+1}(k)$ e os pesos da camada *L+1*.

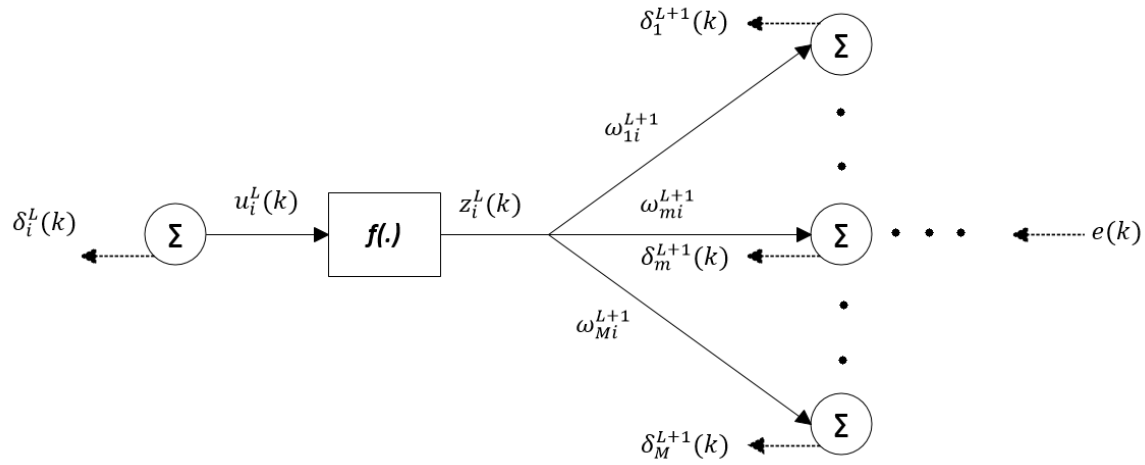


Figura 27 Ilustração de como o erro é calculado no método de retropropagação

De notar que $z_i^{L}(k)$ alimenta todos os neurónios da cada *L+1*. Assim:

$$\delta_i^L(k) = \frac{\partial E}{\partial u_i^L(k)} = \sum_{m=1}^{M} \frac{\partial E}{\partial u_m^{L+1}(k)} \frac{\partial u_m^{L+1}(k)}{\partial u_i^L(k)} = \sum_{m=1}^{M} \left[\omega_{mj}^L f(u_j^L(k))\right] = f'(u_i^L(k)) \sum_{m=1}^{M} \delta_m^{L+1}(k)\omega_{mi}^L \tag{35}$$

A equação acima é a fórmula do erro de retropropagação que calcula o erro delta desde a camada de saída até à de entrada, num processo camada a camada.

Dado o erro delta, os pesos são atualizados de acordo com a fórmula seguinte:

$$\omega_{ij}^L(t+1) = \omega_{ij}^L(t) + \eta \sum_{k=1}^{K} \delta_i^L(k) z_j^{L-1}(k) + \mu\left[\omega_{ij}^L(t) - \omega_{ij}^L(t-1)\right] + \varepsilon_{ij}^L(t) \tag{36}$$

Do lado direito da equação (36), o segundo termo é o erro quadrático médio em relação a ω_{ij}^L. O terceiro termo é conhecido como *momentum* e a sua função é demonstrada na Figura 28. Este fornece um mecanismo que adaptativamente ajusta o tamanho dos passos dados. Quando vetores do gradiente em intervalos de tempo sucessivos aponta na mesma direção, o tamanho dos passos aumenta. Quando os vetores do gradiente sucessivos formam um padrão de pesquisa em "zigzag", a direção do gradiente é regulado pelo termo *momentum* para se minimizar o erro quadrático.

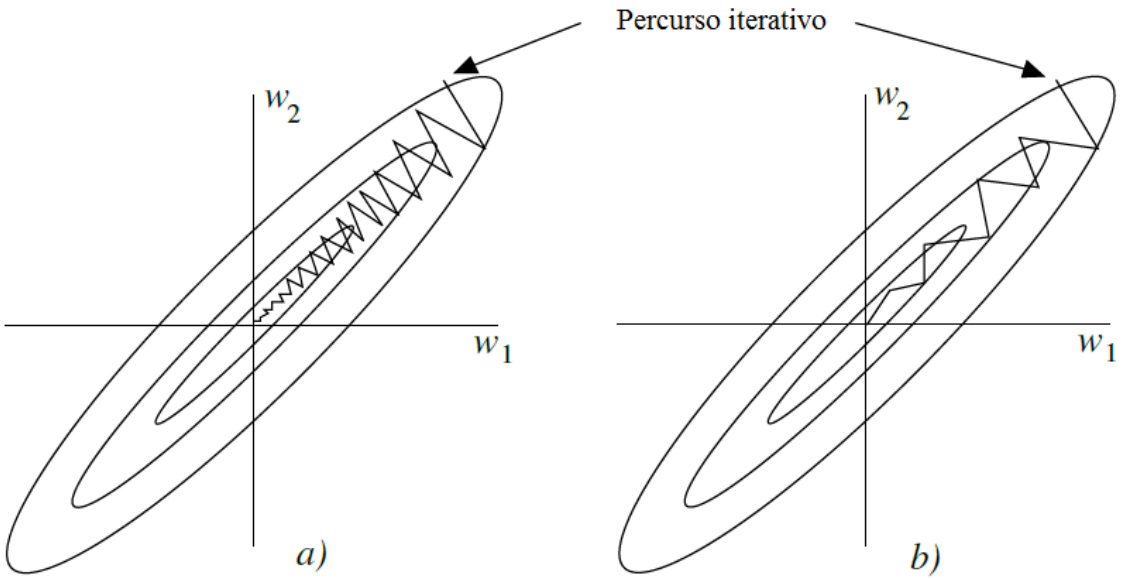


Figura 28 Método descendente no espaço dos pesos das ligações: a) η sem *momentum* ; b) η com *momentum* [15]

Existem dois parâmetros que devem ser escolhidos: a taxa de aprendizagem, e a constante de *momentum*. O valor de ambos os parâmetros deve ser compreendido no intervalo [0,1]. Na pratica, η assume geralmente um valor pequeno, por exemplo, $0 < \eta < 0{,}3$, e μ assume um valor mais alto , como por exemplo, $0{,}6 < \mu < 0{,}9$.

O último termo da equação é um termo de ruído aleatório que tem pouco efeito quando os outros termos têm grandes amplitudes. Quando a procura atinge um mínimo local, a magnitude do vetor de gradiente ou *momentum* é provável que diminua. Nesta situação, o termo de ruído ajuda o algoritmo a "saltar" do mínimo local e continuar a procurar a solução ideal.

2.6. APLICAÇÕES

Praticamente todas as tarefas não-mecânicas realizadas por animais requerem a interação de redes neuronais. Perceção, reconhecimento, memória e pensamento consciente, são alguns dos exemplos. Cada uma destas tarefas pode ter várias aplicações em áreas diversas e de diferentes géneros. Áreas como a medicina e as telecomunicações, e aplicações desde projetos de investigação inovadores até outras que se tornaram sucessos comerciais. Esta amplitude de aplicabilidade torna o campo das redes neuronais interdisciplinar.

Nesta secção são apresentadas as tarefas envolvidas no uso das redes neuronais, e exemplos de aplicações para cada uma delas.

2.6.1. CLASSIFICAÇÃO

Classificação é definida como a atribuição de um objeto a uma classe específica. Esta categoria é de fundamental importância em várias áreas, desde o tratamento de imagem até ao reconhecimento de voz. Fornecendo um conjunto de exemplos que consistem em padrões de amostra representativos de todas as classes, e informações de associações para cada classe de cada padrão, é possível deduzir regra de associação e criar um *classificador*. Este pode ser usado para atribuir outros padrões à sua respetiva classe.

Na Figura 29 são apresentados alguns exemplos de classificação.

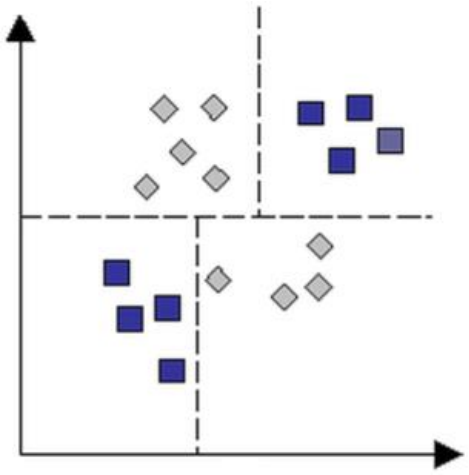


Figura 29 Exemplos de classificação [21]

As redes neuronais têm sido bastante usadas para classificar amostras, mapeando padrões de entrada para diferentes classes. Um dos métodos usado é o de definir cada neurónio de saída como representante de uma classe. O padrão de entrada pertence a uma certa classe, se a saída que a representa calcula o valor mais alto de entre os neurónios de saída.

De seguida são apresentados alguns exemplos de tarefas de classificação prática que se tornaram em casos de sucesso com o uso das redes neuronais:

- Reconhecimento de Caracteres – São usadas em grande parte redes neuronais de múltiplas camadas. Mesmo com aplicações que são baseadas num algoritmo padrão de treino, é bastante comum personalizar a arquitetura de modo a melhorar a execução da aplicação. Numa aplicação deste tipo é necessário, geralmente, que o utilizador forneça os padrões de escrita como modo de comparação [24].

- Produção de Voz – Um dos exemplos de uma abordagem das redes neuronais a este problema é o NETtalk. Em contraste com a necessidade de construir regras e tabelas para exceções, o único requerimento do NETtalk é um conjunto de exemplos de entrada, combinado com uma pronunciação correta de cada elemento. A entrada inclui a letra que é falada, e as três letras que a antecedem e sucedem, fornecendo contexto. A rede foi treinada usando as 1.000 (mil) palavras mais comuns inglesas. Depois de algum treino, a rede é capaz de ler novas palavras com muito poucos erros [25].

- Reconhecimento de Voz – Diferentes tipos de redes neuronais têm sido usadas nesta área, incluído redes de múltiplas camadas e redes com ligações recursivas. Uma delas de grande interesse são as desenvolvidas por Kohonen usando mapas auto-organizáveis. Esta aplicação designa-se de *Phonetic Typewriter*. As unidades de saída são dispostas num *array* bidimensional. Depois dos sinais de entrada da voz serem mapeados para as regiões do fonema, as unidades de saída podem ser ligadas para a tecla da máquina de escrever apropriada [26].

2.6.2. AGRUPAMENTO

Agrupamento (Figura 30) requere a associação de objetos semelhantes uns aos outros [27][28]. Este tipo de problema é idêntico ao de classificação. Porém, a informação disponível é apenas composta por um conjunto de amostras e relações de distância das quais podem ser derivadas descrições de amostras.

Algumas redes neuronais usam o seguinte método para realizar o agrupamento dos objetos. Inicialmente, cada neurónio reage aleatoriamente às amostras de entrada apresentadas. Os neurónios com altos valores de saída a uma entrada aprendem a reagir ainda mais forte a essa mesma entrada ou a uma outra semelhante. A este processo dá-se o nome de especialização [2].

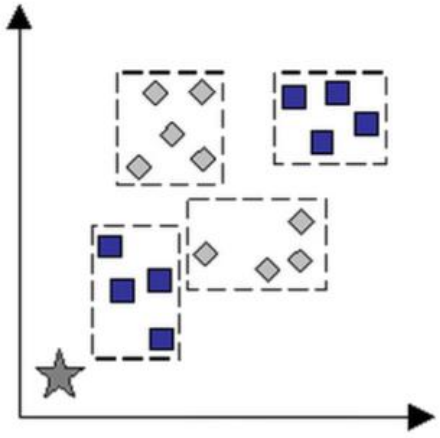


Figura 30 Exemplo do resultado de um agrupamento [21]

2.6.3. ASSOCIAÇÃO DE PADRÕES

Nesta categoria, a apresentação de uma amostra na entrada deve provocar a conceção de um padrão de saída específico. Existem dois tipos de associação de padrões: autoassociativos ou memória associativa, e hétero-associativos.

Na primeira, presume-se que o padrão de entrada é uma versão corrompida, ruidosa, ou parcial, do padrão de saída desejado. Um exemplo deste tipo de tarefas é a conceção completa de uma imagem (não corrompida), tendo sido apresentada na entrada uma versão corrompida dessa mesma imagem.

Uma das aplicações deste tipo de tarefa é no processamento de sinal [29], na área das telecomunicações. Uma das primeiras aplicações comercializadas foi (e ainda é) uma rede para suprimir o ruído de uma linha telefónica. A necessidade de filtros antirruído adaptativos tornou-se mais urgente com o desenvolvimento de ligações satélite transcontinentais para circuito telefónico de longa distância. A comutação envolvida na supressão do eco convencional é perturbadora devido ao atraso incluído nestes tipos de ligações. Mesmo no caso de transmissões telefónicas com fios, os amplificadores repetidores introduzem ecos no sinal [5].

A ideia do método de cancelamento de ruído é bastante simples. No final da linha de um dos pares da ligação, o sinal de entrada é aplicado ao componente do sistema telefónico e ao filtro adaptativo. A diferença entre a saída dos dois elementos é o erro, que é usado para ajustar os pesos da rede neuronal. Esta é treinada para remover o ruído (eco) da saída do sistema telefónico.

No segundo tipo de tarefa, a saída pode ser um padrão aleatório que é posteriormente associado a um conjunto de padrões de entrada. Um exemplo de associação hétero-associação é a produção de um nome quando uma imagem de uma face é apresentada como entrada [30][31].

Uma de muitos exemplos de aplicações das redes neuronais na medicina foi desenvolvida em meados dos anos 80. A aplicação é chamada de *Instant Physician*. A ideia por detrás desta aplicação é treinar uma rede neuronal de memória autoassociativa para armazenar um grande número de registos médicos, cada um dos quais inclui informação sobre sintomas, diagnóstico, e tratamento para um caso particular. Depois do treino, a rede pode ser apresentada com uma entrada que consiste num conjunto de sintomas, ao qual a rede encontra um padrão armazenado que representa o melhor diagnóstico e tratamento [5].

A rede tem um desempenho muito bom, dada a sua estrutura simples. Quando um conjunto particular de sintomas ocorre com frequência no conjunto de treino, a rede fornece o mesmo diagnóstico e tratamento. Em casos onde existem ambiguidades nas informações de treino,

a rede fornece o diagnóstico e tratamento mais comum. Em situações novas, a rede prescreve um tratamento correspondente aos sintomas que já foram vistos, independentemente dos outros sintomas presentes [5].

2.6.4. APROXIMAÇÃO DE FUNÇÕES

Muitos modelos computacionais podem ser descritos como funções que mapeiam vetores numéricos de entrada [32][33]. A aproximação de funções é uma tarefa de aprendizagem ou construção de uma função que gera aproximadamente as mesmas saídas que o processo a ser modelado para um conjunto de vetores de entrada.

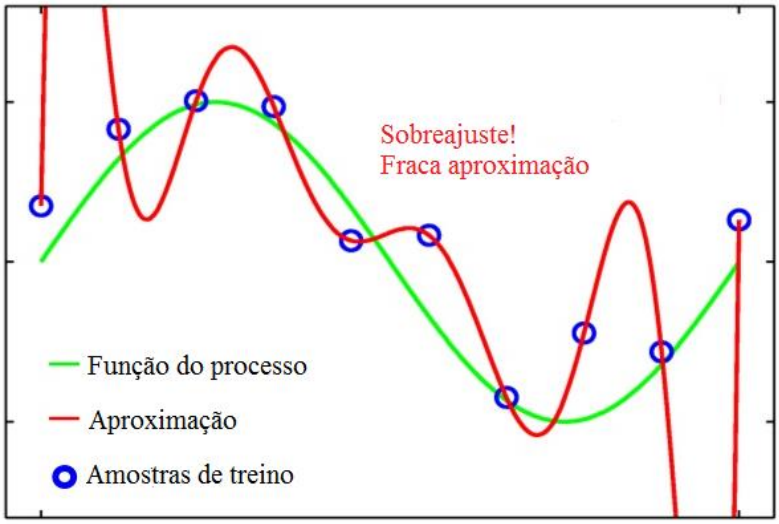


Figura 31 Exemplo de uma aproximação de funções [47]

Na Figura 31 é demonstrado um exemplo de uma aproximação de funções. Os pontos apresentados no gráfico são as amostras fornecidas à rede, e as linhas representam as funções encontradas. As funções devem coincidir, ou estar muito próximos, com os valores das amostras para representarem o processo a modelar. Por fim, deve ser escolhida uma dessas funções.

Contudo, existe um número infinito de funções que coincidem com um conjunto finito de pontos. Assim, são necessários critérios adicionais para decidir qual das funções é a desejável. Um aspeto a ter em conta, aquando da escolha de uma função, é a sua continuidade e suavidade. Tal como é demonstrado na Figura 31, a função mais estável torna-se a "verdadeira função" do sistema. Relativamente à rede neuronal, esta deve ser o mais simples possível. Quanto menos parâmetros a rede neuronal tiver melhor será.

Muitos problemas industriais e de manufaturação envolvem a estabilização do comportamento de um objeto, ou a monitorização do comportamento de um objeto em movimento. Estes podem também ser vistos como problema de aproximação de funções, na qual a função desejada representa o comportamento do objeto em questão ao longo do tempo.

2.6.5. PREVISÃO

Existem muitos problemas da vida real na qual eventos futuros têm que ser previstos com base no passado [34][35]. Embora uma previsão completamente acertada seja dificilmente possível, as redes neuronais pode ser usadas para obter previsões razoáveis num grande número de casos. Num exemplo concreto, as redes neuronais foram capazes de aprender com sucesso o ciclo de 11 anos de mudança de atividade solar sem terem sido informadas *a priori* da existência desse ciclo.

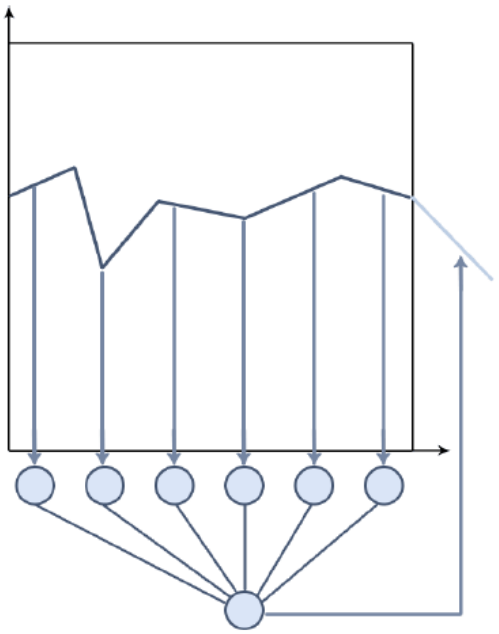


Figura 32 Exemplo de processo de previsão [16]

Como é visível na Figura 32, inicialmente é apresentado um conjunto dos *n* valores mais recentes da variável à rede, onde é extraído um conjunto de *n+1* valores. O último destes valores é a saída desejada, ou seja, o valor calculado pela rede e o próximo na sequência de tempo. Assim, o problema de previsão reduz-se a um problema de aproximação de funções. A diferença entre a aproximação e a previsão é que os valores da função são representados usando séries de tempo, ou seja, sequências de valores medidos ao longo do tempo. Podendo estes ser medidos em unidades discretas ou contínuas.

Neste tipo de problemas é importante considerar que existem previsões de curto-prazo e de longo-prazo. No primeiro caso, a previsão é realizada para o valor seguinte baseada apenas em valores passados. No segundo caso, alguns valores previstos são também usados para prever futuros valores.

Um dos exemplos deste tipo de problemas é o trabalho de avaliação de hipotecas [36]. Neste negócio é difícil de especificar o processo do qual especialistas tomam as suas decisões. Além disso, existe uma recompensa financeira pela redução do número de hipotecas desrespeitadas. A ideia por detrás desta abordagem do uso de redes neuronais para avaliação do risco de hipoteca é o uso de experiências passadas para treinar a rede de modo a fornecer uma avaliação mais consistente e fidedigna para as aplicações de hipotecas. Contudo, segue um método diferente do explicitado acima.

Usando a informação de vários avaliadores experientes, redes neuronais foram treinadas para fazer a triagem de aplicações de hipoteca. O objetivo é o de determinar se o aplicante deve ou não receber um empréstimo. O conjunto de sinais de entrada de treino incluem informação sobre os anos de desemprego do aplicante, número de dependentes, rendimento corrente, entre outras, tais como características relacionadas com a hipoteca em si, taxa empréstimo-valor, características da propriedade e o valor de avaliação [5].

Os resultados revelaram um elevado nível de concordância com especialistas humanos. Usando uma medida independente de qualidade das hipotecas certificadas, as redes neuronais tem consistentemente melhores julgamentos do que os especialistas. Assim, a rede aprendeu a formar um consenso com a experiência de todos os especialistas cujas ações foram a base para a sua formação [5].

Para um melhor entendimento de problemas de grande dificuldade obtido pelo estudo de variáveis relacionadas, em vez do estudo de uma só variável. Uma sequência de tempo com múltiplas variáveis consiste numa série de valores de diferentes variáveis simultaneamente alteradas com o passar do tempo. Este tipo de previsão obtém resultados de elevada exatidão.

2.6.6. OTIMIZAÇÃO

Muitos problemas em negócios e modelações científicas podem ser representadas como problemas de otimização, no qual o objetivo é otimizar algumas funções sujeitas a restrições [37][38]. Um exemplo é a tarefa de organizar componentes numa placa de circuito impresso,

em que o tamanho das ligações devem ser minimizadas, e outras restrições que requerem que certos componentes estejam ligados entre si.

2.6.7. CONTROLO

Muitas aplicações industriais e de manufaturação têm relações implícitas complexas entre as entradas e saídas. O controlo visa determinar os valores corretos a aplicar às variáveis de entrada de modo a ser alcançado o valor desejado nas variáveis de saída [39][40]. Os diferentes tipos de controlo irão ser abordados no capítulo seguinte.

Um tipo de exemplo aplicável é a tarefa que envolve manobrar um camião com atrelado [41][42]. As dificuldades desta tarefa são óbvias para qualquer um. Contudo, um condutor experiente completa-a com relativa facilidade. Considere-se a tarefa de treinar uma rede que forneça direções a um atrelado na tentativa de o colocar numa plataforma de carregamento. A informação disponível relata a posição da cabine do camião, a posição da traseira do camião, a posição fixa da plataforma, e os ângulos do camião e do atrelado com a plataforma. A rede neuronal é capaz de aprender como direcionar o camião de modo a que este chegue à plataforma, iniciando o camião e o atrelado numa qualquer configuração que permita folga suficiente para que aja uma solução.

Tal como com o condutor, a execução melhora com a prática, e o controlador neuronal aprende a fornecer uma série de sinais de direção que encaminhem o camião e o atrelado para a plataforma.

Neste exemplo são usadas as redes proactivas, tal como em muitos outros problemas de controlo: controlo de um braço robótico, estabilidade de um poste, cinemática inversa de um robô, etc. Todos estes sistemas têm em comum as seguintes características [2]:

- Realização de rápida tomada de decisão e controlo através de computação paralela;
- Habilidade de se adaptar a um grande número de parâmetros;
- Tolerância natural a falhas devido à representação distribuída de informação;
- Robustez a variações nos parâmetros não modelados, devido às propriedades de generalização das redes.

2.7. CASOS DE ESTUDO

Nesta secção são apresentados dois casos de estudo em que a aplicação das redes neuronais aos problemas que são aqui descritos solucionou os mesmos de uma forma eficaz.

2.7.1. PROCESSO AUTÓNOMO VISUAL DE ACOMPANHAMENTO DE UM TRAJETO

Este caso aborda a deteção automática de um trajeto e orientação automática de um carro, de modo a que este siga um trajeto. Embora a orientação automática já tenha sido abundantemente explorada, a deteção de um trajeto é uma tarefa complexas, especialmente quando se lidam com cenários ao ar livre, pois é necessário um sistema que se adapte à mudança de condições.

As redes neuronais são bastante úteis neste caso, uma vez que já provaram ser capazes de solucionar problemas de reconhecimento de padrões. Assim, estas foram adotadas para as tarefas de navegação autónoma, nomeadamente, para o processo visual de acompanhamento de um trajeto.

O modelo proposto (Figura 33 – a)) consistia em duas etapas de processamento: seleção da característica e classificação. Na primeira são usados algoritmos de transformação da imagem do trajeto obtida para análise posterior. Os algoritmos usados podem ser: deteção de contornos, que transforma a imagem em código binário marcando as extremidades com 1's e o resto da imagem a 0's; transformada de Hough, que permite identificar características na imagem, como por exemplo, linhas, círculos, elipses, etc; por último, existe a opção de não usar nenhum algoritmo de tratamento de imagem.

Na segunda etapa define-se qual o classificador a usar. É possível a utilização de 4 classificadores: mapas auto-organizativos concorrentes (CSOM), perceptrão multicamada (MLP), mapas auto-organizativos supervisionados (SOM), e *K-mean* (método de agrupamento).

Foi quantificado o "caminho a ser seguindo" em três classes: esquerda, direita, e frente. Deste modo a saída do sistema será apenas uma destas instruções.

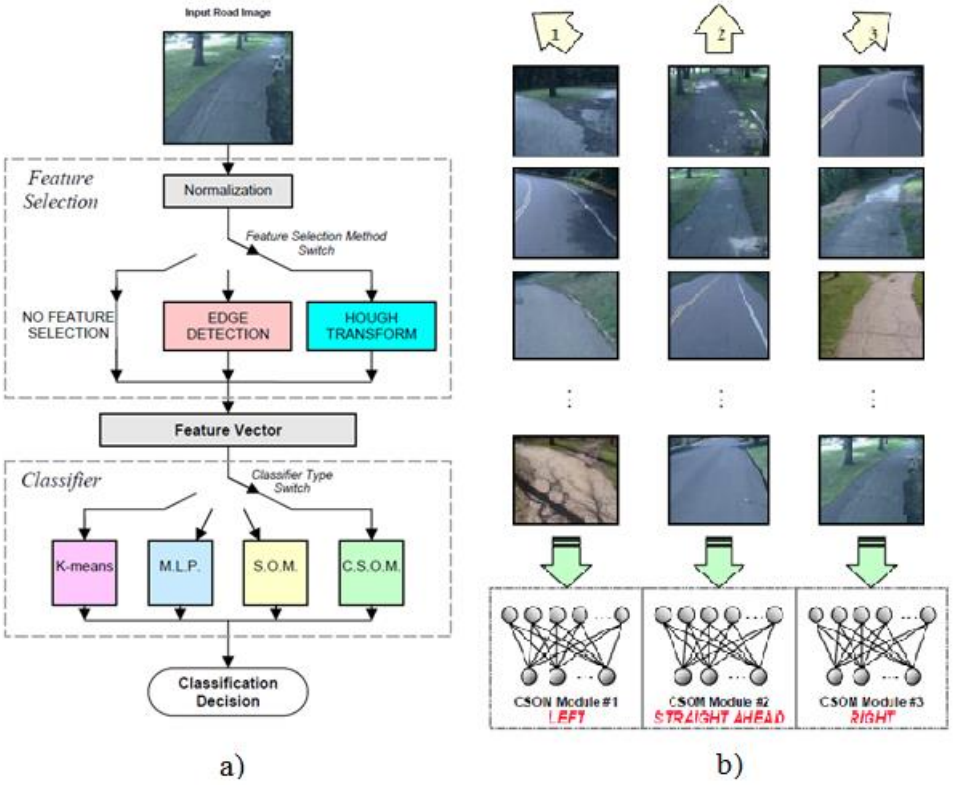


Figura 33 a) – Esquema da arquitetura do sistema; b) – Fase de treino [22]

Relativamente à arquitetura do modelo CSOM, este representa uma coleção de pequenos SOMs usados numa estratégia global de competição. O número deste módulo é igual ao número de classes. Portanto, neste caso, existem 3 módulos idênticos, cada um treinado individualmente para fornecer os melhores resultados para uma das classes, correspondente a uma direção do trajeto possível. Este processo de treino encontra-se apresentado na Figura 33 – b).

Depois da fase de treino, o sistema deve ser capaz de classificar corretamente uma imagem desconhecida numa das três classes, usando a informação armazenada nos pesos da rede.

A imagem a ser classificada é aplicada a todos os três módulos do sistema (Figura 34). A distância entre o vetor de entrada e todos os neurónios dos três modelos são calculadas e, de seguida, o melhor neurónio correspondente é determinado como sendo o mais próximo ao vetor de entrada. A imagem do trajeto é classificada, assim, como pertencente ao neurónio menos distante, e a legenda do módulo que contém o melhor neurónio correspondente é atribuído à imagem de entrada.

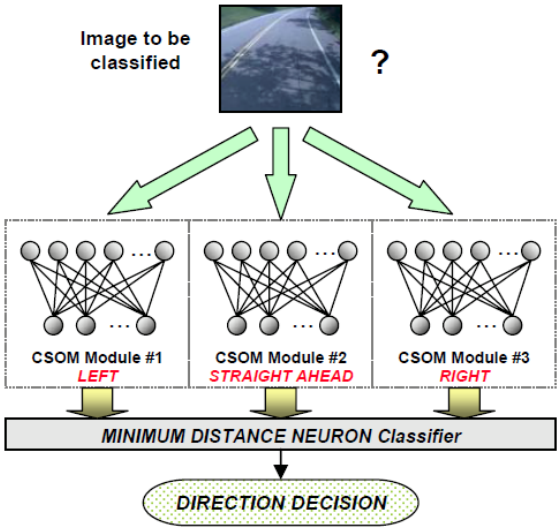


Figura 34 Fase de classificação [22]

Os resultados experimentais deste sistema são apresentados na Figura 35. São apresentados os resultados de acordo com os vários parâmetros relativos ao sistema, nomeadamente, tipos de classificador, cor da imagem, e método de seleção de característica.

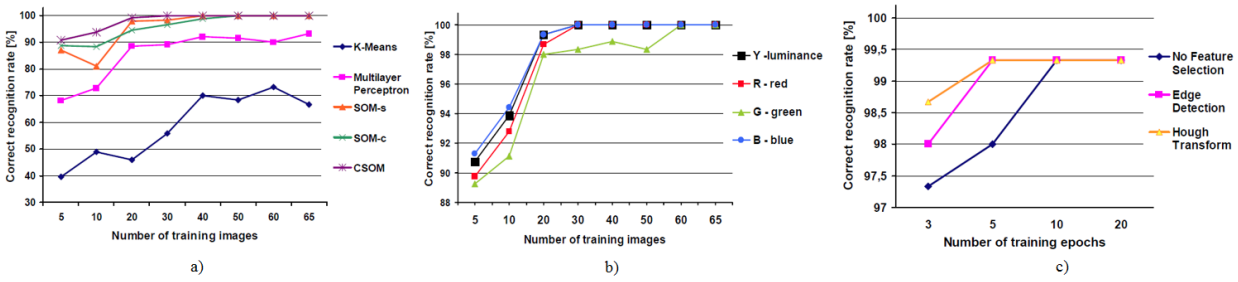


Figura 35 a) – Desempenho da identificação do trajeto de acordo com o tipo de classificador; b) – Desempenho do CSOM em função da componente da cor; c) – Desempenho do CSOM em função do método de seleção de característica [22]

Os resultados experimentais demonstraram que o melhor classificador é o CSOM. A sua arquitetura permite uma implementação do algoritmo mais rápida e flexibilidade em comparação com os outros classificadores. A cor da imagem que permite melhores desempenhos é a cor azul. E, por último, a seleção de característica mais eficaz é o algoritmo pela transformada de Hough.

2.7.2. PREVISÃO DE BOLSA DE VALORES

A previsão dos valores dos índices da bolsa é um assunto importante no setor financeiro. O objetivo deste caso de estudo é ilustrar que as RNA podem ser eficazmente usadas para

prever os valores dos índices da *Istanbul Stock Exchange* (ISE) usando os valores do índices do dia anterior, a taxa de câmbio TL/USD (Lira Turca/Dólar), a taxa de *overnight* (taxa de juro do mercado financeiro), e 5 outras variáveis representando os dias úteis da semana. Foram utilizados modelos de aprendizagem supervisionada na qual certos nós da saída receberam treino para responder a padrões de entrada específicos. As alterações efetuadas nos pesos através da aprendizagem permitiram que o sistema respondesse a classes mais gerais de padrões.

Foi reunida informação experimental diretamente do banco central da república da Turquia durante um período de 417 dias. Deste conjunto, os primeiros 376 casos foram usados como treino e 41 como exemplos de teste.

Para o modelo de sistema descrito, são usados dois modelos diferentes de redes: proactivas, designadas por MLP, e Acíclico, designado por GFF. São aplicados 3 exemplos de cada modelo: com 1, 2, ou 4, camadas invisíveis de neurónios.

Neste estudo, foram aplicados 6 modelos de redes ao modelo do sistema. Os desempenhos das redes podem ser medidos pelo coeficiente de determinação (R^2) ou o erro médio relativo percentual. Este coeficiente é uma medida de precisão da previsão dos modelos de redes treinados. Valores mais altos de R^2 indicam melhor precisão. O erro médio relativo percentual pode também ser usado para medir a precisão da previsão através da representação do grau de dispersão. Para cada modelo de previsão, foi utilizada a equação (37) para calcular o erro relativo para cada caso do conjunto de teste. De seguida, os valores calculados foram avaliados e multiplicados por 100 para serem exprimidos em termos percentuais.

$$\frac{|(f_{ISE})_{actual} - (f_{ISE})_{previsto}|}{(f_{ISE})_{actual}} \tag{37}$$

A Tabela 1 mostra os valores de R^2 para os modelos de rede MLP e GFF aplicados ao sistema.

Tabela 1 Coeficientes de determinação para os modelos de redes neuronais artificias [23]

Number of Hidden Layers	ANN Model	
	MLP	GFF
1	0.81	0.82
2	0.79	0.81
4	0.78	0.81

Os desempenhos das redes podem ser comparados com a abordagem das médias móveis (EWMA ou MA). O EWMA é a média de valores desfasados do índice ao longo de um período passado específico (5 e 10 dias). Os erros médios relativos percentuais foram calculados como 0,022 por 5 dias e 0,03 por 10 dias. A Tabela 2 mostra os erros para cada modelo.

Tabela 2 Erro médio relativo percentual para todos os modelos [23]

Model	Mean Relative Percentage Error (%)
MLP - 1 Hidden Layer	1.62
MLP - 2 Hidden Layers	1.65
MLP - 4 Hidden Layers	1.70
GFF - 1 Hidden Layer	1.59
GFF - 2 Hidden Layers	1.65
GFF - 4 Hidden Layers	1.71
MA - 5 days	2.17
MA - 10 days	3.03

A precisão da previsão para cada modelo foi comparada através do coeficiente de determinação. A eficiência dos modelos variaram com o número de camadas invisíveis. Para ambos os modelos MLP e GFF, são registadas maiores precisões para o modelo de uma camada invisível.

Depois de calculados os erros médios relativos percentuais, verificou-se que os modelos de redes neuronais foram superiores aos modelos EWMA.

Com base nos resultados destes estudo, concluiu-se que:

- Os modelos de previsão baseados em RNA são mais precisos que os modelos EWMA;

- De entre os modelos de redes neuronais, o GFF (Acíclico) foi considerado o mais apropriado para esta previsão.

3. ESTRUTURAS DE CONTROLO NEURONAIS

Neste capítulo são apresentados aspetos relacionados com a implementação de sistemas de controlo neuronais. Inicialmente são expostas as formas de aprendizagem realizadas pelas redes neuronais: indireta e direta. São também apresentados alguns dos modelos de controlo mais utilizados envolvendo estas redes.

De seguida, são expostas ferramentas de *software* utilizadas no desenvolvimento e simulação de redes neuronais. Além da apresentação do ambiente gráfico, são também enumerados alguns aspetos positivos e negativos de cada uma das ferramentas. É abordado em maior detalhe o *software* utilizado no desenvolvimento deste trabalho, o MATLAB, em particular a sua *Toolbox* de redes neuronais. Este capítulo explora tanto os ambientes gráficos mais intuitivos e simples da *Toolbox*, como as suas funções de criação, treino, e simulação, para desenvolvimento em código MATLAB.

3.1. CONTROLO

O controlo de um processo é uma tarefa de aprendizagem adequada para as redes neuronais. Designa-se por um processo um sistema que é mantido numa condição de controlo. No contexto do controlo, o cérebro é a prova viva de que é possível construir um controlador

generalizado que tira proveito do *hardware* distribuído em paralelo, controlando milhares de atuadores (fibras musculares) em paralelo, lidando com ruído e não-linearidades, e podendo otimizar processos durante um planeamento a longo prazo.

Considere-se um sistema de controlo com realimentação demonstrado na Figura 36. O sistema envolve o uso de uma unidade de realimentação em torno do processo a ser controlado, ou seja, a saída do processo alimenta diretamente a entrada. Assim, a saída do processo é subtraída a um sinal de referência fornecido por uma fonte externa. O sinal de erro produzido é aplicado ao controlador neuronal com a finalidade de ajustar os seus parâmetros. O objetivo primário do controlador é fornecer entradas adequadas para que a saída do processo acompanhe o sinal de referência. Noutras palavras, o controlador inverte o comportamento entrada-saída do processo [16].

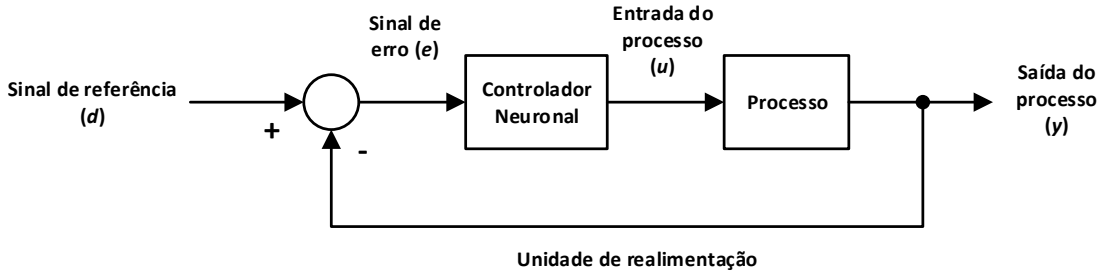


Figura 36 Diagrama de blocos de um sistema de controlo com realimentação

De notar que na Figura 36, o sinal de erro propaga-se através do controlador neuronal antes de atingir o processo. Consequentemente, para realizar ajustes aos parâmetros do processo, é necessário calcular o Jacobiano, constituído por uma matriz de derivadas parciais, como mostrado por:

$$J = \left\{\frac{\partial y_k}{\partial u_j}\right\}_{j,k} \tag{38}$$

Onde y_k é um elemento da saída do processo y e u_j é um elemento da entrada do processo u. Infelizmente, as derivadas parciais $\frac{\partial y_k}{\partial u_j}$ para diversos k e j dependem de ponto de operação do processo, portanto, não são conhecidas. Pode-se usar uma de duas abordagens para os contabilizar [16]:

- Aprendizagem indireta – Usando medições de reais entrada-saída do processo, constrói-se primeiro uma rede neuronal que mapeie as entradas e saídas da mesma forma. Este modelo é usado para fornecer uma estimativa do Jacobiano. As derivadas

parciais que constituem o Jacobiano são posteriormente utilizadas na regra de aprendizagem para calcular os ajustamentos aos parâmetros do controlador neuronal.

- Aprendizagem direta – Os sinais das derivadas parciais $\frac{\partial y_k}{\partial u_j}$ são geralmente conhecidos e permanecem constantes ao longo de uma intervalo dinâmico do processo. Isto sugere que se pode aproximar estas derivadas pelos seus sinais individuais. Aos seus valores absolutos são dados uma representação distribuída nos parâmetros do controlador. O controlador neuronal fica habilitado a aprender as alterações aos parâmetros diretamente a partir do processo.

Antes de serem apresentadas algumas das mais populares arquiteturas de rede usadas para o controlo, é importante referir e perceber que nestes casos as redes neuronais são aplicadas com a intenção de aproximar um mapeamento entrada-saída desconhecido, ou seja, pode imitar o processo ou o seu inverso. Estes dois métodos podem ser denominados por:

- Identificação do sistema – Neste primeiro caso a relação entrada-saída do sistema é treinada para modelar um certo sistema ou processo. Na Figura 37 é apresentado um diagrama de blocos que exemplifica este mesmo método. Aqui, tal como no método seguinte, o erro é utilizado para ajustar os parâmetros da rede e minimizar a diferença entre as saídas do processo desconhecido e as da rede neuronal do ponto de vista estatístico.

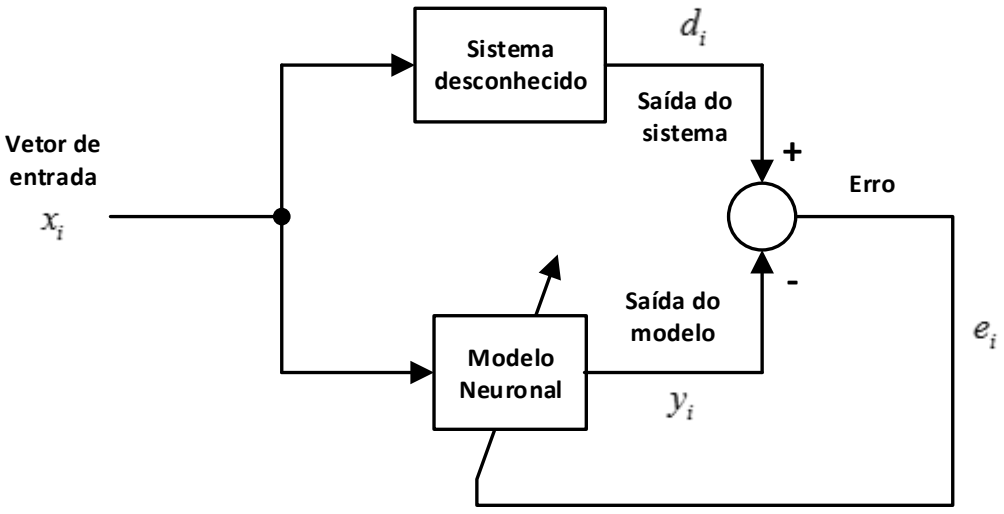


Figura 37 Diagrama de blocos da identificação de um sistema

- Modelação Inversa – Para este caso, o requisito é o de construir um modelo inverso ao do processo. Um sistema inverso pode ser definido por:

$$x = f^{-1}(d)$$

Em muitos casos práticos, a função $f(.)$ é demasiado complexa e inibe uma formulação direta da função inversa $f^{-1}(.)$. Dado um conjunto de exemplos, é possível contruir uma aproximação usando o esquema da Figura 38. Nesta situação aqui descrita, d_i é usado como vetor de entrada e x_i como a resposta desejada. O vetor do sinal de erro e_i indica a diferença entre x_i e a saída real y_i da rede neuronal produzida pela resposta d_i. Geralmente, a modelação inversa é uma tarefa mais difícil que a de identificação de um processo, pois nunca existe apenas uma única solução.

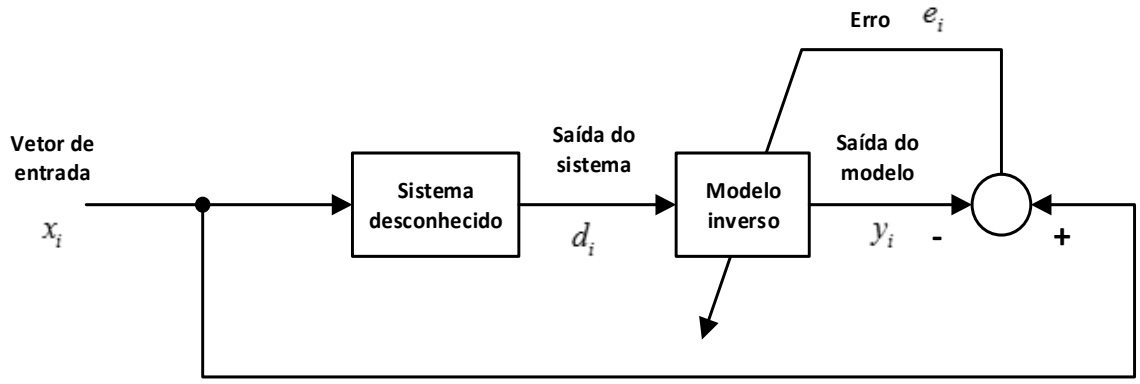


Figura 38 Diagrama de blocos da modelação inversa de um sistema

3.1.1. CONTROLADOR ESTABILIZADOR FIXO

Este tipo de controlador tem sido aplicado, por exemplo, no controlo de trajetória de um braço robótico, onde é usado um controlador proporcional com ganho como controlador estabilizador de realimentação. Da Figura 39 pode-se ver que o total da entrada que chega ao processo é a soma do sinal de controlo de realimentação e um sinal de controlo de alimentação direta, que é calculado do inverso do modelo dinâmico (rede neuronal). Esse modelo usa a trajetória desejada como entrada e realimentação como sinal de erro. Com o avanço do treino da rede, a entrada irá convergir para zero [43].

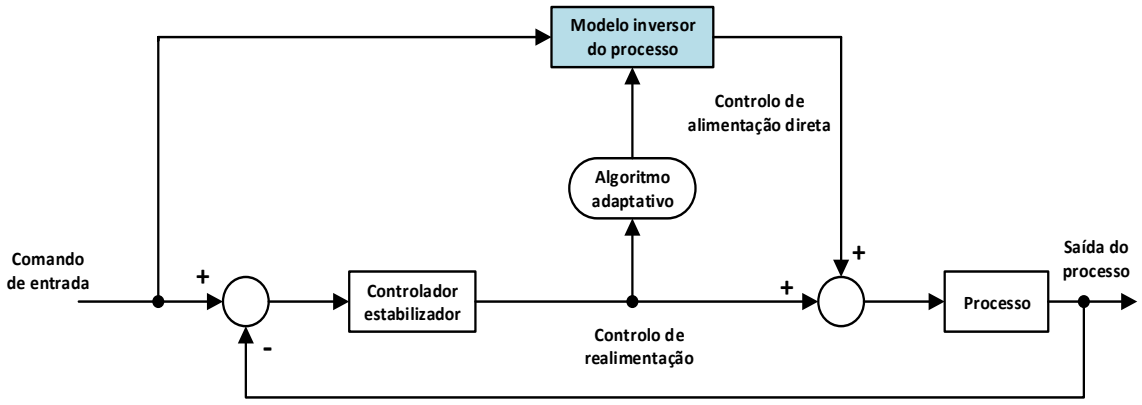


Figura 39 Controlador estabilizador

3.1.2. CONTROLO ADAPTATIVO INVERSO

A Figura 40 mostra a estrutura de um modelo de referência de controlo adaptativo inverso [44]. O algoritmo adaptativo recebe o erro entre a saída do processo e a saída do modelo de referência. Os parâmetros do controlador são atualizados para minimizar o erro de acompanhamento. A abordagem do modelo básico de referência de controlo adaptativo pode ser afetado pelo ruído do sensor e distúrbios no processo. Uma alternativa que permite o cancelamento do ruído inclui um modelo neuronal do processo em paralelo com o mesmo sistema. Esse modelo é treinado para receber a mesma entrada do processo e produzir a mesma saída. A diferença entre as saídas é interpretada como o efeito do ruído e distúrbios no processo. Esse sinal serve como entrada de um modelo inverso do sistema para gerar um sinal filtrado que é subtraído à entrada do processo.

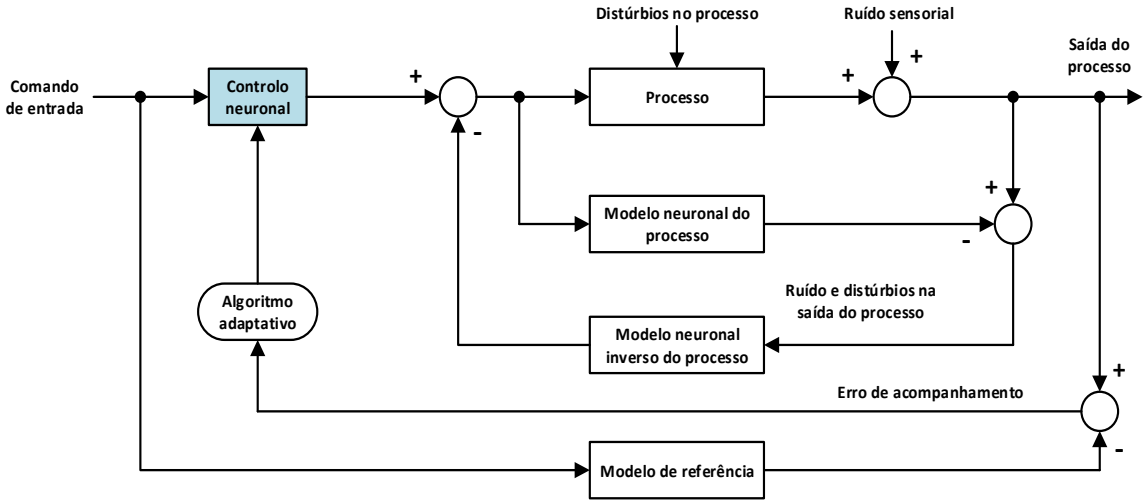


Figura 40 Sistema de controlo adaptativo inverso

3.1.3. MODELO DO CONTROLO INTERNO NÃO-LINEAR

O modelo de controlo interno não-linear (Figura 41) consiste num controlador neuronal, um modelo neuronal do processo, e um filtro de robustez com um único parâmetro de ajuste. A rede neuronal que atua como controlador é geralmente treinado para representar o inverso do processo, se o mesmo existir. O erro entre a saída do modelo e a do processo é usado como entrada de realimentação no filtro de robustez, que, de seguida, alimenta o controlador neuronal.

O modelo neuronal do processo e o controlador neuronal podem ser treinados *offline*, usando informações recolhidas sobre as operações do sistema. O filtro de robustez é um filtro de primeira ordem cuja constante de tempo é escolhida de modo a garantir a estabilidade do sistema.

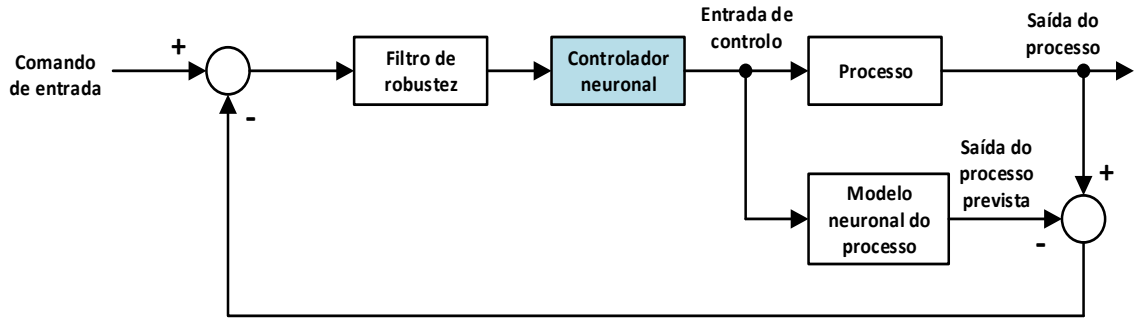


Figura 41 Modelo de controlo interno não-linear

3.1.4. MODELO DE CONTROLO PREDITIVO

O modelo de controlo preditivo (Figura 42) otimiza a resposta do processo ao longo de um período de tempo específico. Esta arquitetura requer um modelo neuronal do processo, um controlador neuronal, uma função de desempenho para avaliar a resposta do sistema, e um procedimento de otimização para selecionar a melhor entrada de controlo.

O procedimento de otimização pode ser computacionalmente dispendiosa. Requer o cálculo de múltiplos passos à frente, na qual o modelo neuronal é usado para prever a resposta do processo. O controlador neuronal aprende a reproduzir a entrada selecionada pelo processo de otimização. Quando o treino estiver completo, o passo de otimização pode ser completamente substituído pelo controlador neuronal.

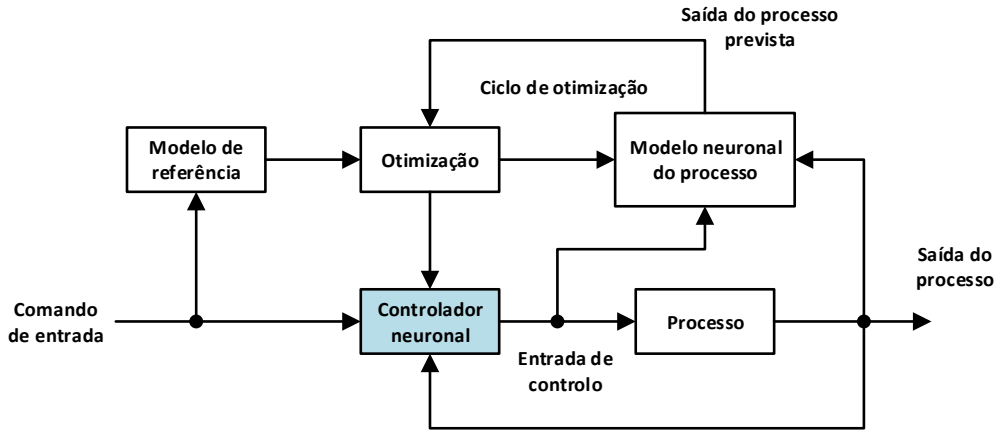


Figura 42 Modelo de controlo preditivo

3.1.5. CONTROLO DE MODELO DE REFERÊNCIA OU CONTROLO DE REFERÊNCIA ADAPTATIVO

Tal como em outras técnicas, a configuração do controlo de modelo de referência usa duas redes neuronais: uma rede controladora e uma rede modelo (Figura 43). O modelo neuronal pode ser treinado *offline* usando medições do processo já recolhidas. O controlador é treinado adaptativamente para forçar a saída do processo a seguir a saída do modelo de referência. O modelo neuronal é usado para prever o efeito das alterações do controlador no processo, permitindo a atualização dos parâmetros do controlador.

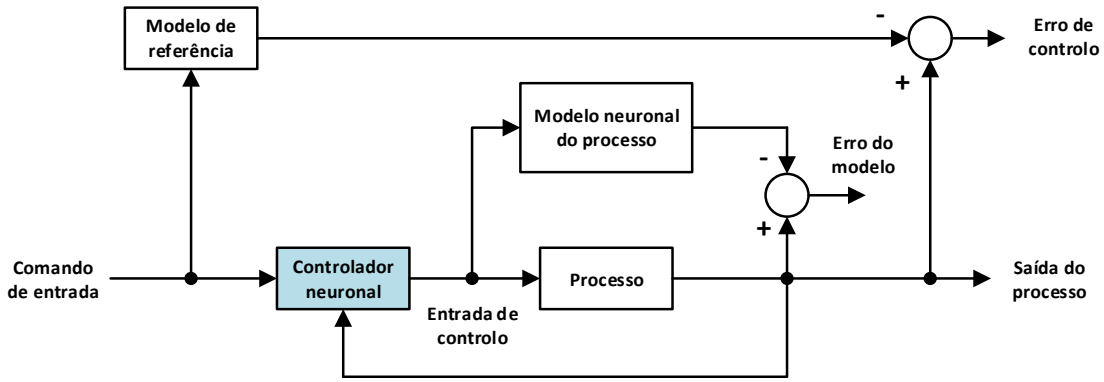


Figura 43 Modelo de controlo adaptativo de referência

3.1.6. CRÍTICA ADAPTATIVA

Tal como é apresentado na Figura 44, o controlador de crítica adaptativa consiste em duas redes neuronais. A primeira age como um controlador inverso e é designada como rede de atuação. A segunda, chamada de rede crítica, é treinada para otimizar o desempenho futuro.

O treino é realizado usando a aprendizagem por reforço, que é uma aproximação da programação dinâmica. Têm existido muitas variações do controlador adaptativo crítico propostas nos últimos anos [43].

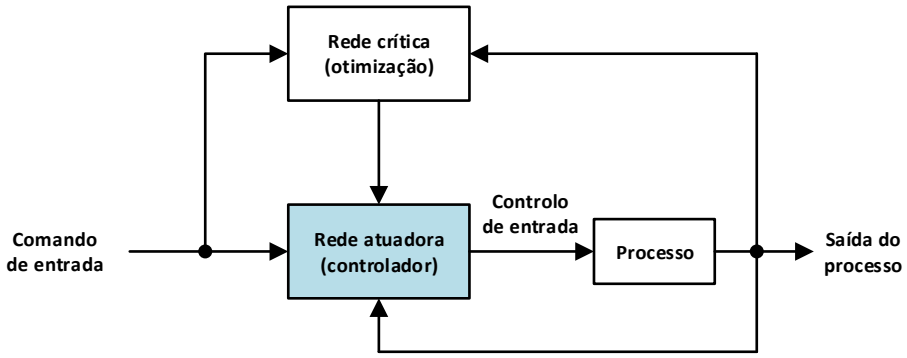


Figura 44 Crítica adaptativa

3.2. *SOFTWARE* DE DESENVOLVIMENTO

Dada a versatilidade das RNAs, estas podem ser aplicadas a diferentes áreas. Esta diversidade levou a uma necessidade crescente e ao desenvolvimento de ferramentas informáticas que, de uma forma intuitiva, permitisse aos utilizadores individuais o uso desta tecnologia nos seus projetos.

Existem inúmeras aplicações informáticas atualmente no mercado. Desde aplicações *freeware* a *shareware*, sendo o código de implementação das redes escrito em várias línguagens de programação.

Um dos *software*s de simulação de redes neuronais é o SNNS (*Stuttgart Neural Network Simulator*). Este foi desenvolvido na Universidade de Estugarda e inicialmente construído para estações de trabalho UNIX. O objetivo do projeto SNNS é a criação de um ambiente de simulação eficiente e flexível para investigação e aplicação das redes neuronais.

O simulador consiste em 4 componentes principais: o núcleo do simulador, interface gráfica (Figura 45), interface de execução por lotes, e um compilador de rede. O núcleo do simulador opera nas estruturas internas de informação da rede neuronal e executa todas as suas operações. A interface gráfica XGUI (X *Graphical User Interface*), construída sob o núcleo, oferece uma representação gráfica e controlo das redes neuronais durante a simulação. Adicionalmente, a interface pode ser usada diretamente para criar, manipular e visualizar as redes neuronais de diferentes formas. Podem ser criadas rapidamente redes complexas.

Porém, a XGUI é também adequada para utilizadores inexperientes, que querem aprender mais sobre os modelos de ligações através do simulador [48].

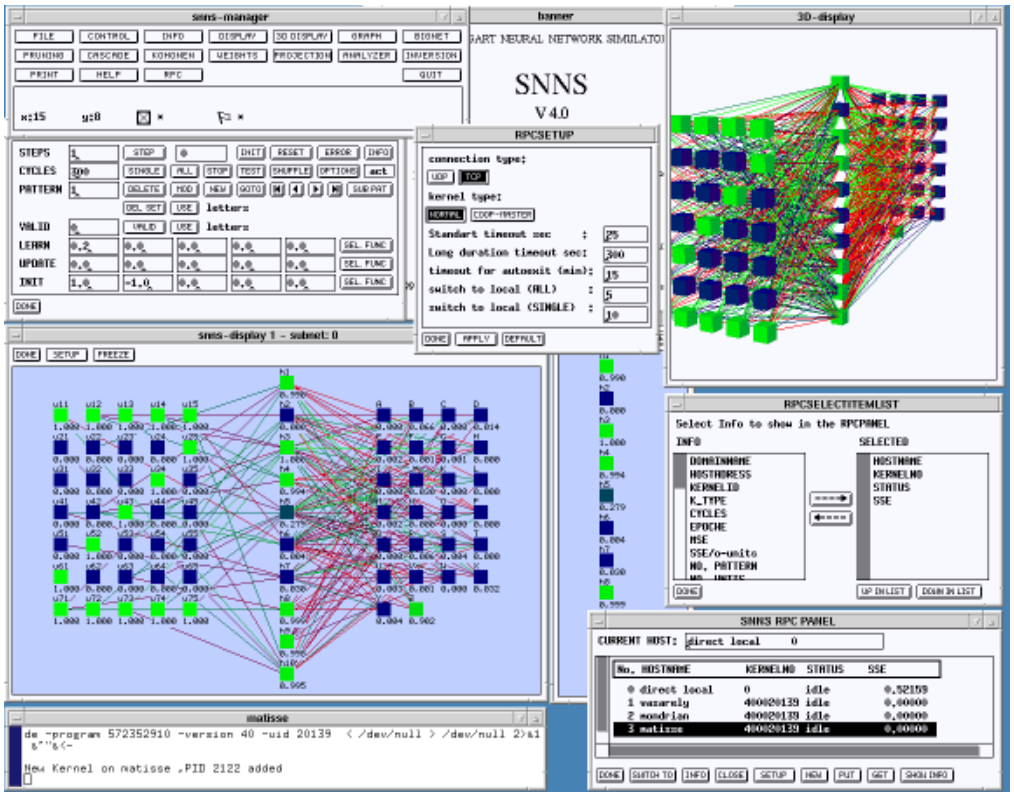


Figura 45 Ambiente gráfico do *software* SNNS [50]

Um conceito importante do projeto da aplicação é a possibilidade do utilizador poder selecionar apenas as características da representação visual em que está interessado, incluindo a descrição de diversos aspetos e partes da rede com múltiplas janelas, tal como a supressão de informação indesejada [48].

O SNNS pode também ser expandido através de novas funções de ativação, funções de saída, procedimentos de aprendizagem, etc, definidas e escritas pelo utilizador em programas em C associados ao núcleo do simulador.

São enumeradas de seguida algumas arquiteturas de rede e procedimentos de aprendizagem incluídos no simulador [49]:

- Retropropagação para redes proativas;
- Contra-propagação;
- *Quickprog* (método de aprendizagem derivado da retropropagação);

- Funções de base radial (RBF);
- Retropropagação através do tempo (para redes recursivas);
- Mapas auto-organizados (mapas Kohonen);
- Redes Jordan;
- Memória associativa.

O segundo *software* aqui abordado é o JavaNNS. Este simulador para redes neuronais foi desenvolvido no Instituto de Wilhelm-Schickard (WSI) na Alemanha, e é baseado no núcleo do simulador SNNS. Como consequência, as suas capacidades são na maioria iguais às do seu antecessor. Contudo, apresenta uma interface gráfica (Figura 46) completamente nova, escrita em Java, e mais intuitiva e fácil de utilizar. Algumas das características (mais complexas) do SNNS foram deixadas de fora [51].

Além da nova interface, a grande vantagem do JavaNNS é a particularidade de ser multiplataforma. Enquanto o SNNS foi criado principalmente para estações de trabalho UNIX, este novo simulador pode ser executado em qualquer máquina que tenha o Java *Runtime Enviroment* instalado [51].

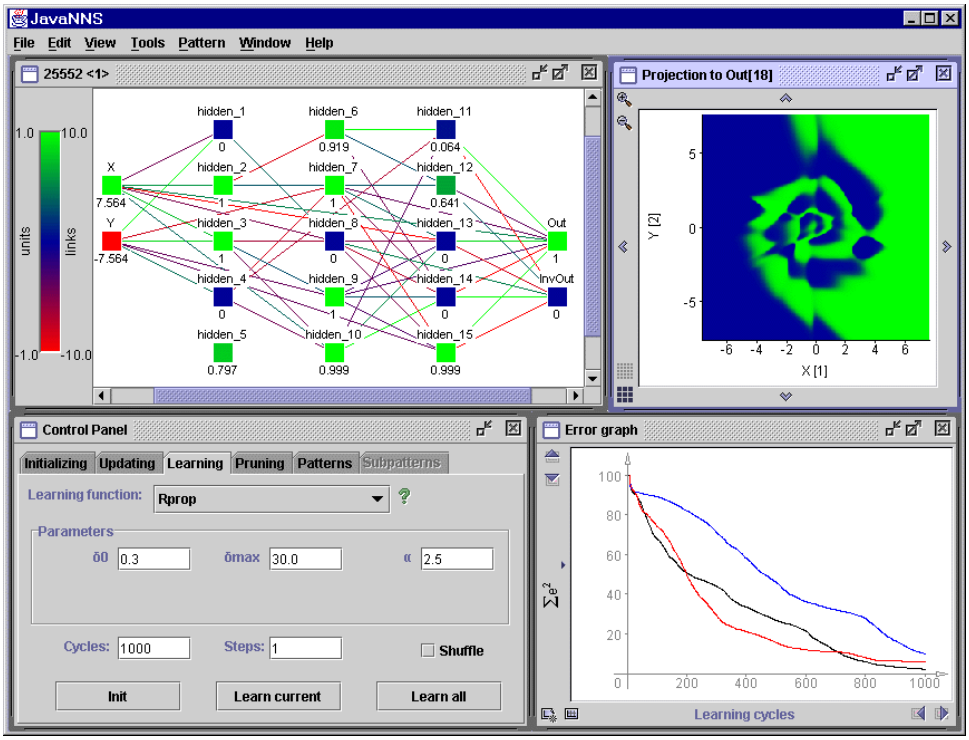


Figura 46 Ambiente gráfico do *software* JavaNNS [52]

Outra das aplicações, também *freeware*, utilizadas para construção e simulação das redes neuronais, é o NetMaker. Este foi desenvolvido com o objetivo de apoiar a classificação de partículas em experiências no CERN (*Conseil Européen pour la Recherche Nucléaire*). Foi também usado para experiências com diversos algoritmos. Mais recentemente, tornou-se um programa de projeto de redes neuronais para inúmeras aplicações.

Algumas das características deste *software* são [53]:

- O ambiente gráfico (Figura 47) apresenta os dados de entrada e resultados da rede, através de gráficos especializados, histogramas, diagramas de dispersão, parâmetros de aprendizagem, etc;
- É escrito em C#;
- Cálculos otimizados em C.

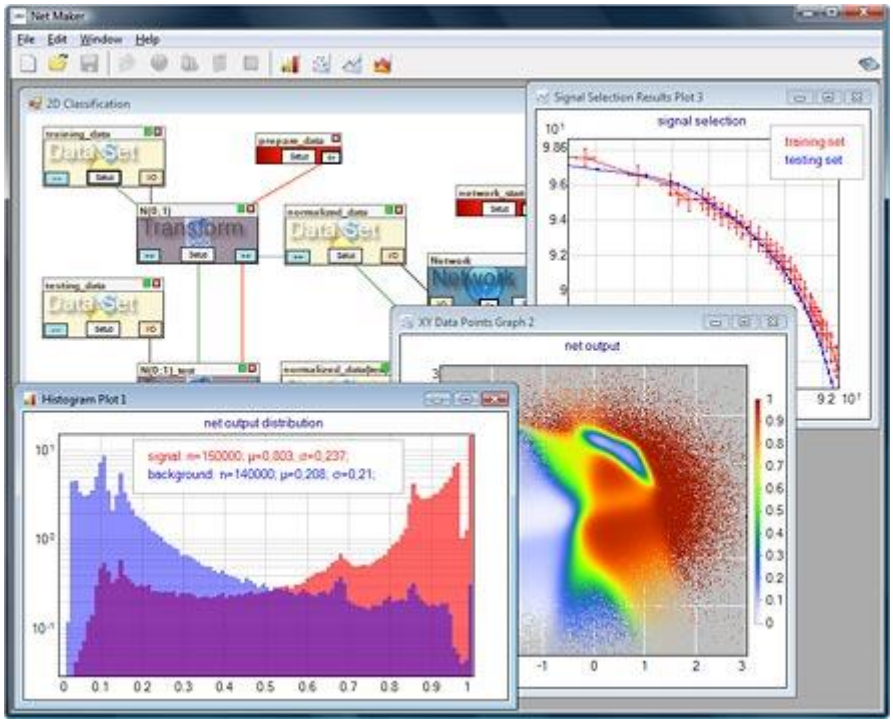


Figura 47 Ambiente gráfico do *software* NetMaker [53]

Uma das aplicações mais conhecida e utilizada é o MATLAB. Desenvolvido pela MathWorks, o MATLAB é um *software* interativo de alto desempenho voltado para o cálculo numérico. Este permite análise numérica, cálculo com matrizes, processamento de sinais, construção de gráficos e interfaces, e interface com programas escritos em outras linguagens, como C, C++, Java, Python, etc.

Embora na sua versão base o MATLAB já possua um vasto conjunto de funções de carácter genérico, existem várias bibliotecas de funções adicionais (*Toolboxes*) que expandem as suas capacidades em domínios de aplicação mais específicos. As *Toolboxes* constituem um vasto conjunto de funções que permitem resolver rápida e eficientemente classes particulares de problemas de grande complexidade em diversas áreas de interesse (entre outras, sistemas de controlo, otimização e RNAs).

3.3. A BIBLIOTECA DA *TOOLBOX* DE REDES NEURONAIS DO MATLAB

A *Neural Networks Toolbox* consiste num conjunto de funções e estruturas que permite manipular as redes neuronais para modelar sistemas não-lineares, sem necessidade de escrever qualquer tipo de código para funções de ativação, algoritmos de treino, etc. Nela é possível projetar, treinar, visualizar, e simular redes neuronais, podendo ser utilizada para aplicações de aproximação de funções, reconhecimento de padrões, agrupamento, previsão, e controlo e modelação de sistemas dinâmicos.

3.3.1. INTERFACES *NNTOLL* E *NNSTART*

Para facilitar o uso desta *Toolbox*, estão à disposição do utilizador os seguintes comandos: *nntool* e *nnstart*. Estes podem ser executados na linha de comandos do MATLAB, e fornecem interfaces gráficas para facilitar a construção e simulação de redes neuronais, tornando o processo mais amigo do utilizador e intuitivo.

O primeiro dos comandos, *nntool*, apresenta a interface exibida na Figura 48. Esta janela, denominada de gestor de redes/dados neuronais, é dividida em diferentes áreas. Destacam-se as áreas onde são expostos os dados de entrada, dados desejados e as redes neuronais criadas, que equivalem aos campos *Input Data*, *Target Data*, e *Networks*, respetivamente. Aqui é possível criar uma rede, ver o seu aspeto, simulá-la, e exportar os resultados para o espaço de trabalho do MATLAB. Da mesma forma, é possível importar dados do espaço de trabalho para serem utilizados nesta interface.

Para adicionar conjuntos de valores ou uma nova rede, é necessário premir o botão *New*. Aparece, então, uma janela com dois separadores (Figura 49). O primeiro dos separadores, denominado de *Network*, exibe opções de configuração da rede neuronal a criar. Opções como dados de entrada, dados desejados, função de treino, função de aprendizagem, função de desempenho, e número de camadas e respetivo número de neurónios e funções de

ativação. O segundo dos separadores, denominado de *Data*, é utilizado para gerar conjuntos de valores. Este separador é dividido em três áreas, representando cada uma um aspeto do conjunto: nome, valores, e tipo. Este último classifica os valores do conjunto como: de entrada, valores desejados, estados de atraso de entrada, estado de atraso de camada, saída ou erros.

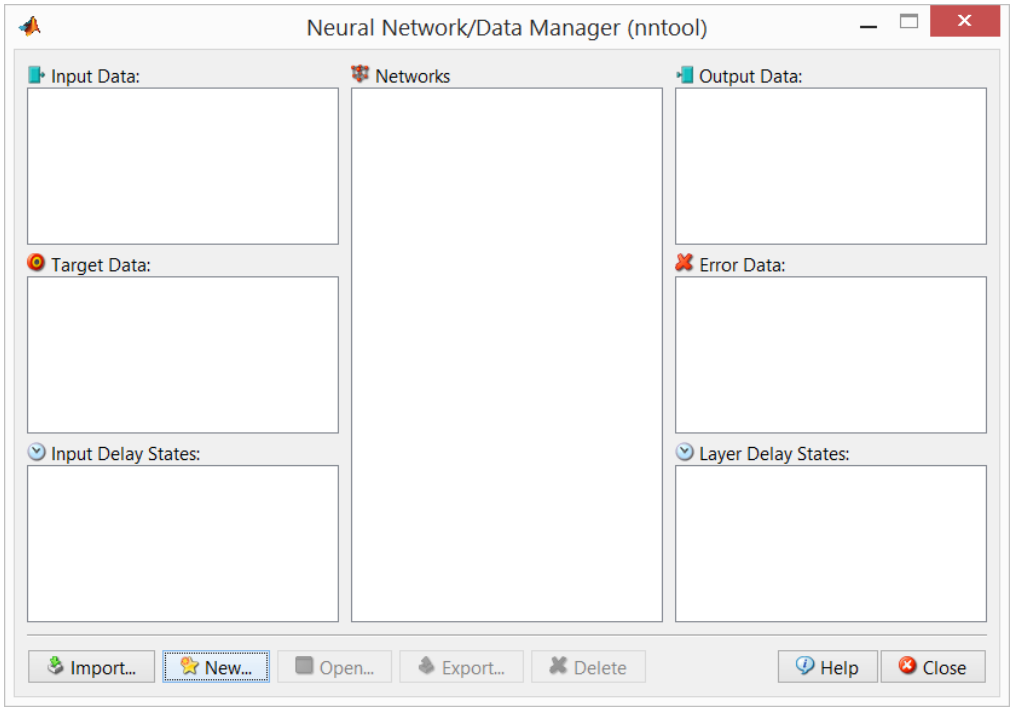


Figura 48 Janela do gestor de redes/dados neuronais

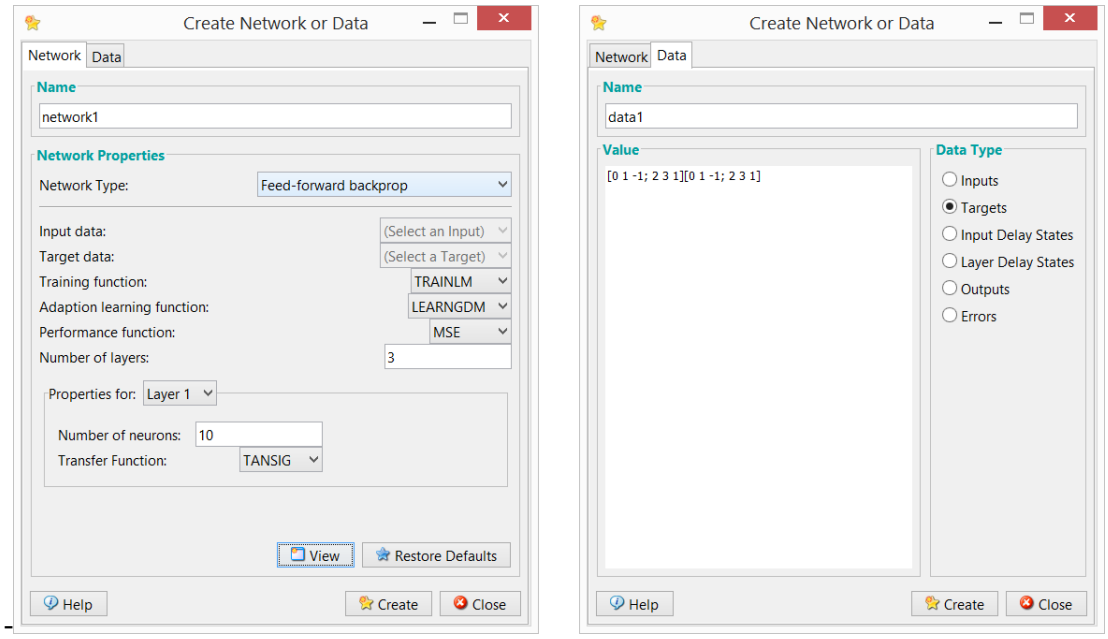


Figura 49 Janela de criação de uma rede ou dados

O comando *nnstart* foi introduzido nesta *Toolbox* mais recentemente que o *nntool*, para que, de certa forma, pudesse substituir este segundo. Se este comando for executado, é aberta a janela exibida na Figura 50. Esta exibe dois separadores. O primeiro permite lançar um de quatro sistemas de resolução de problemas onde as redes neuronais são aplicadas. Cada uma das aplicações é acompanhada pelo respetivo comando e conecção à documentação da aplicação.

O tipo de problemas correspondentes às aplicações aqui disponíveis são:

- Ajuste entrada-saída e de curva;
- Reconhecimento de padrões e classificação;
- Associação e extração de características;
- Modelação dinâmica e problemas de previsão.

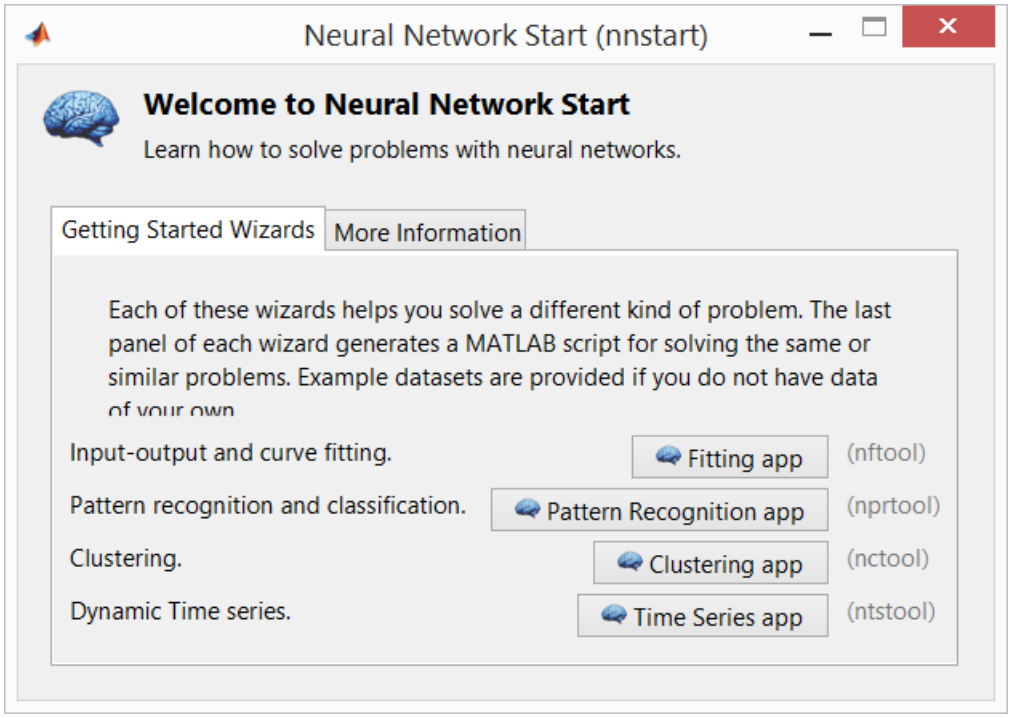


Figura 50 Janela *Neural Network Start*

O segundo separador fornece ligações para outros tipos de recursos. Por exemplo, é possível abrir o guia da *Toolbox*, ou ver uma lista de demonstrações e conjuntos de dados. A cada conjunto está associado um exemplo de código para o resolver, podendo ser resolvido através de uma das aplicações disponíveis no primeiro separador. Existe ainda uma ligação

para um ficheiro de texto, que fornece um olhar mais profundo sobre a matemática por detrás dos algoritmos neuronais e das várias arquiteturas. Podem também ser encontrados muitos exemplos interativos incluídos na *Toolbox*. Cada exemplo permite uma análise mais detalhada sobre o funcionamento de cada um dos pontos de resolução de problemas.

Para melhor entender o ambiente gráfico, foi testada a aplicação de ajuste com um dos exemplos disponíveis na *Toolbox*. Na Figura 51 é apresentada a janela inicial da aplicação escolhida. Nesta janela é exibida a rede de ajuste, e descrito o tipo de problema para que é adequada.

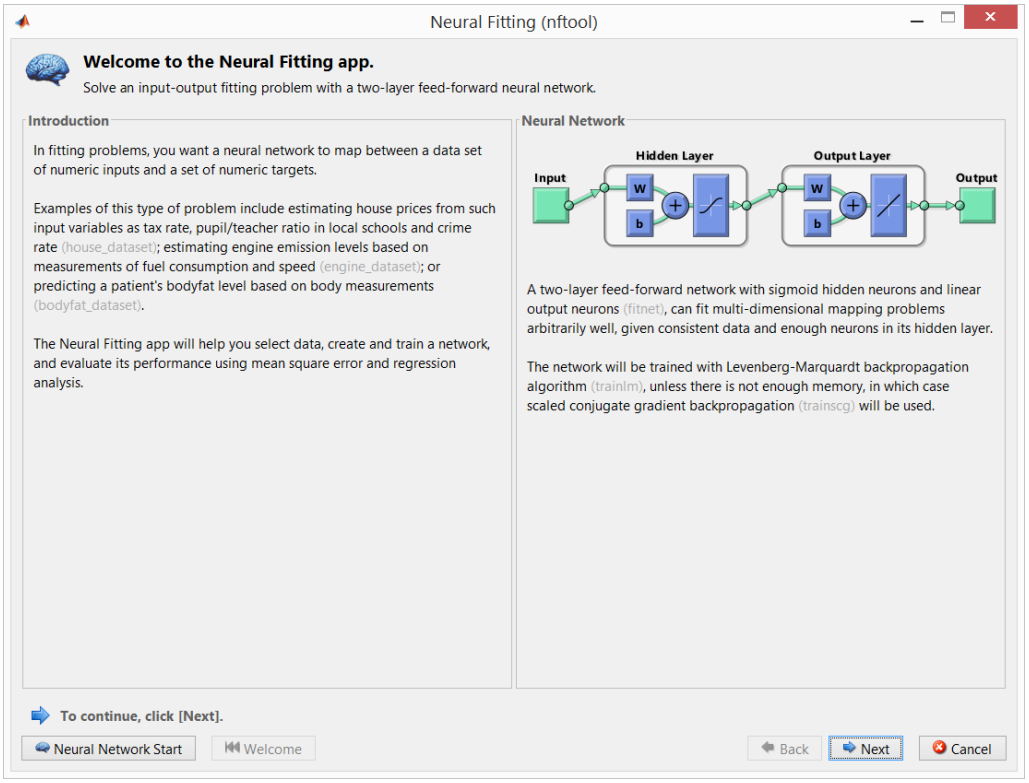


Figura 51 Janela inicial da aplicação de ajuste

O segundo painel (Figura 52) permite a seleção dos dados do problema do espaço de trabalho do MATLAB. Além disso, é também possível carregar um dos exemplos disponibilizados pela *Toolbox* para o espaço de trabalho. Para este problema, foi carregado um exemplo de forma a analisar os diferentes tipos de painéis da aplicação.

O terceiro painel (Figura 53) permite fazer a divisão dos dados do problema, ou seja, os dados são divididos em *treino*, *validação*, e *teste*. Os primeiros dois conjuntos são usados para treino e indicação de quando o treino pára. O terceiro conjunto fornece um teste independente da rede.

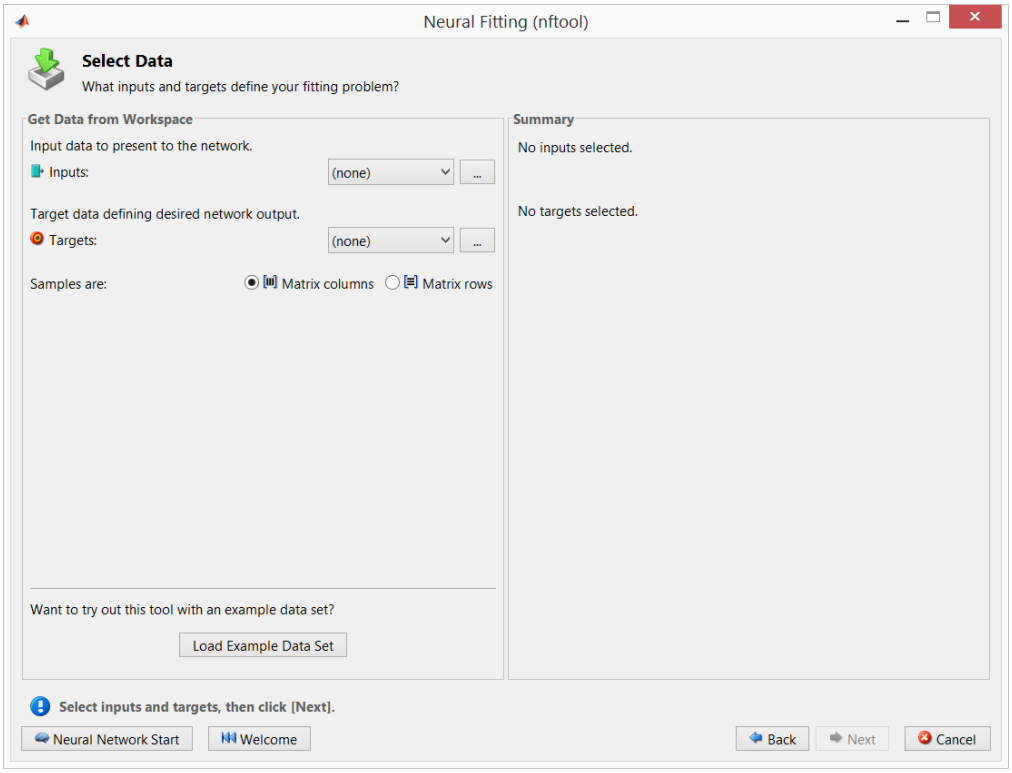


Figura 52 Janela de escolha dos dados do problema

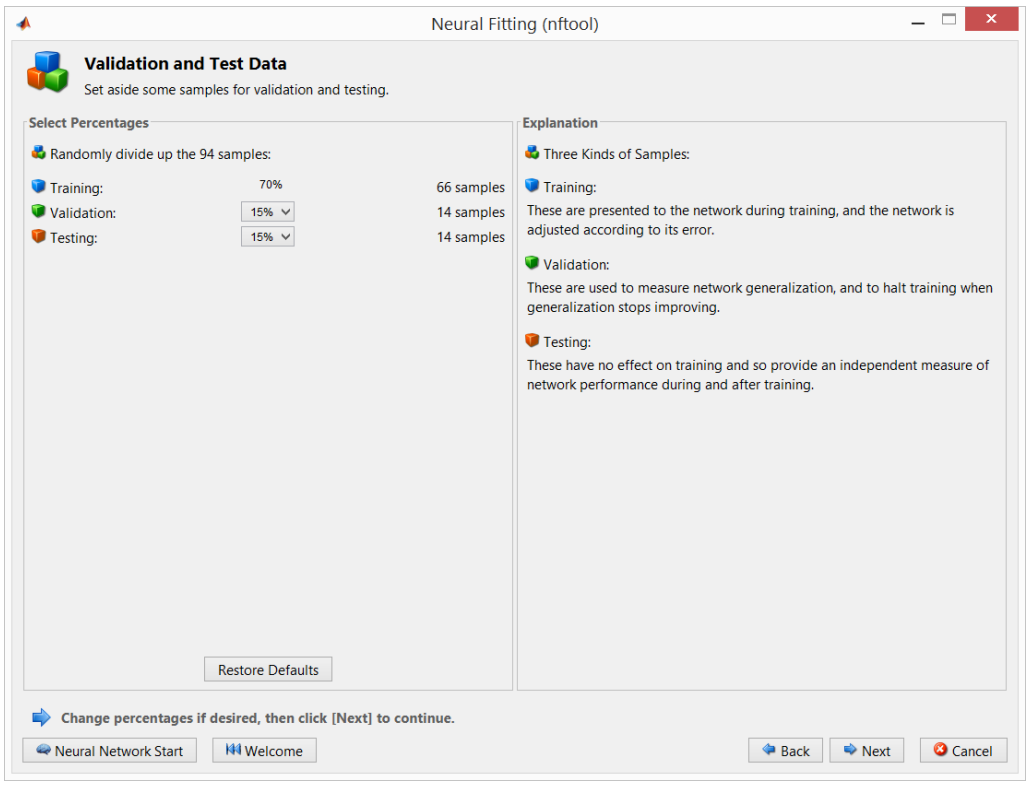


Figura 53 Janela de divisão dos dados

O quarto painel (Figura 54), ou painel de arquitetura da rede, permite uma escolha mais detalhada da rede que irá ser usada para resolver o problema. Podem-se usar os parâmetros por omissão, e se não se obtiver um bom resultado, é sempre possível retornar a este ponto e fazer alterações.

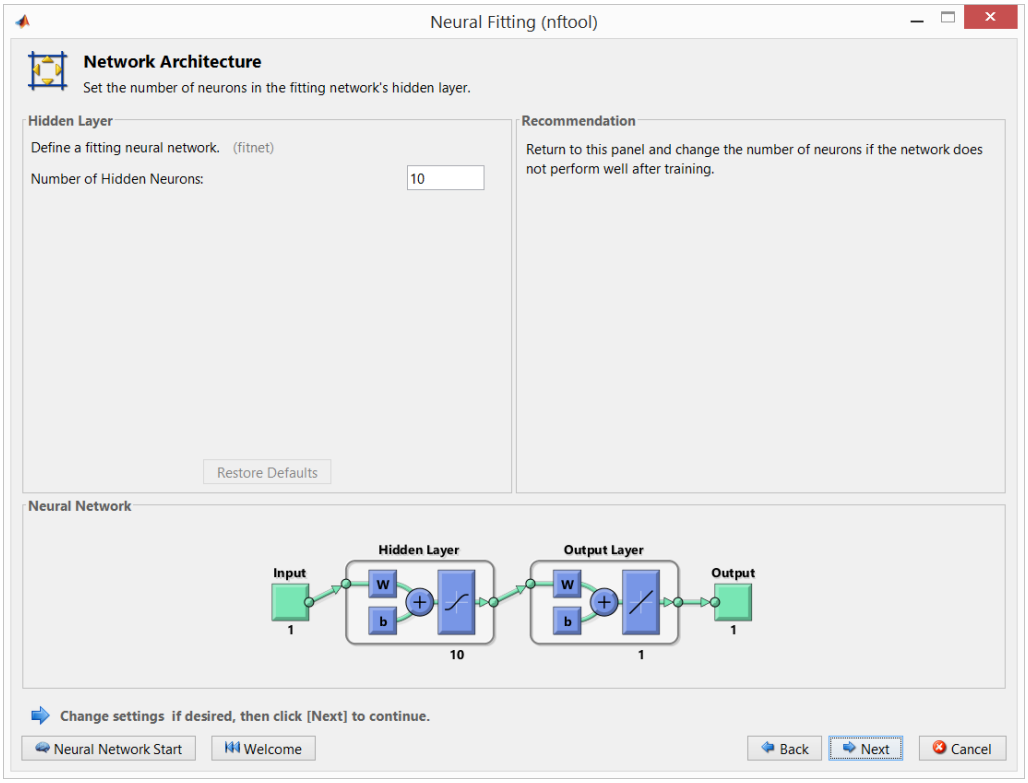


Figura 54 Janela da arquitetura da rede

Depois de criada a rede, o passo seguinte é treiná-la. Na Figura 55 é apresentado o quinto painel. Nele é possível escolher o algoritmo usado para treino, tal como no comando *nntool*. Escolhido o algoritmo, deves-se pressionar o botão *train* para treinar a rede.

A janela onde é possível acompanhar e observar os resultados do treino da rede, denomina-se de ferramenta de treino da rede e é representada pela Figura 56. Esta ferramenta mostra detalhes sobre o algoritmo de treino e o seu estado, incluindo a precisão da rede. Forneceu também gráficos úteis para analisar o desempenho da rede, o histograma do erro, entre outros.

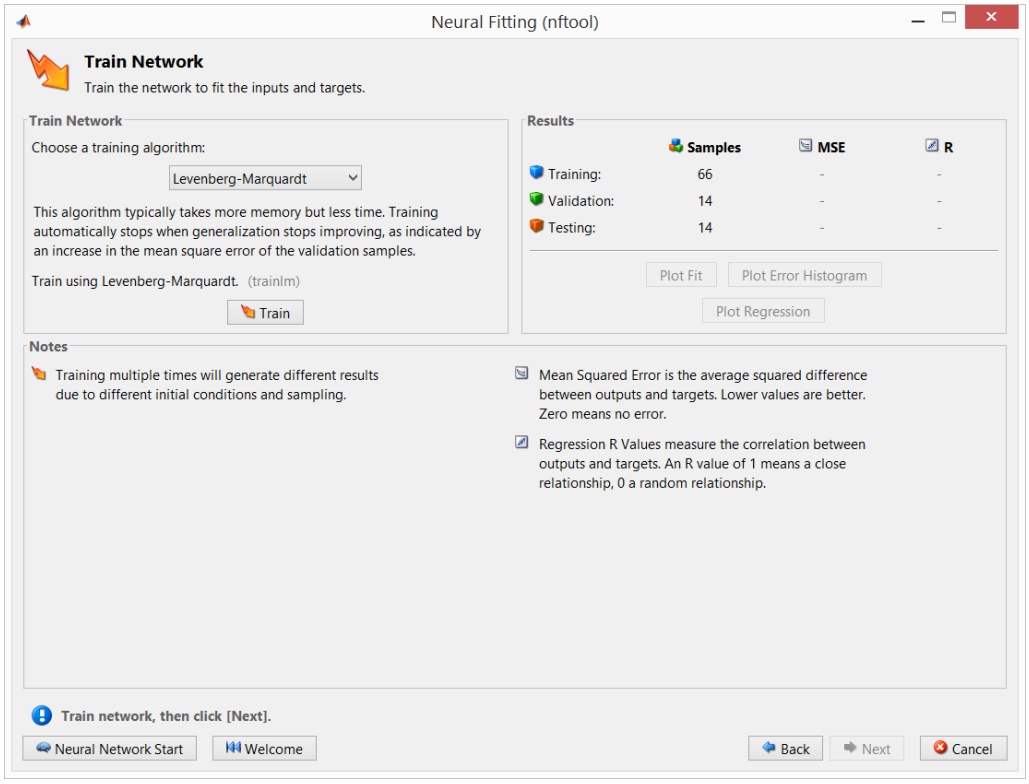


Figura 55 Janela de treino da rede

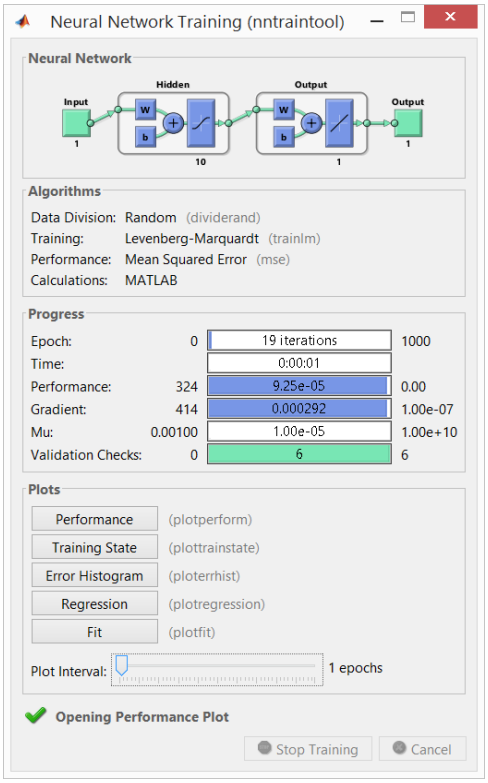


Figura 56 Ferramenta de treino da rede

O sexto painel (Figura 57) é o de avaliação da rede. Este permite que a rede treinada possa ser usada para analisar dados adicionais, e fornece sugestões sobre como melhorar a rede, se necessário.

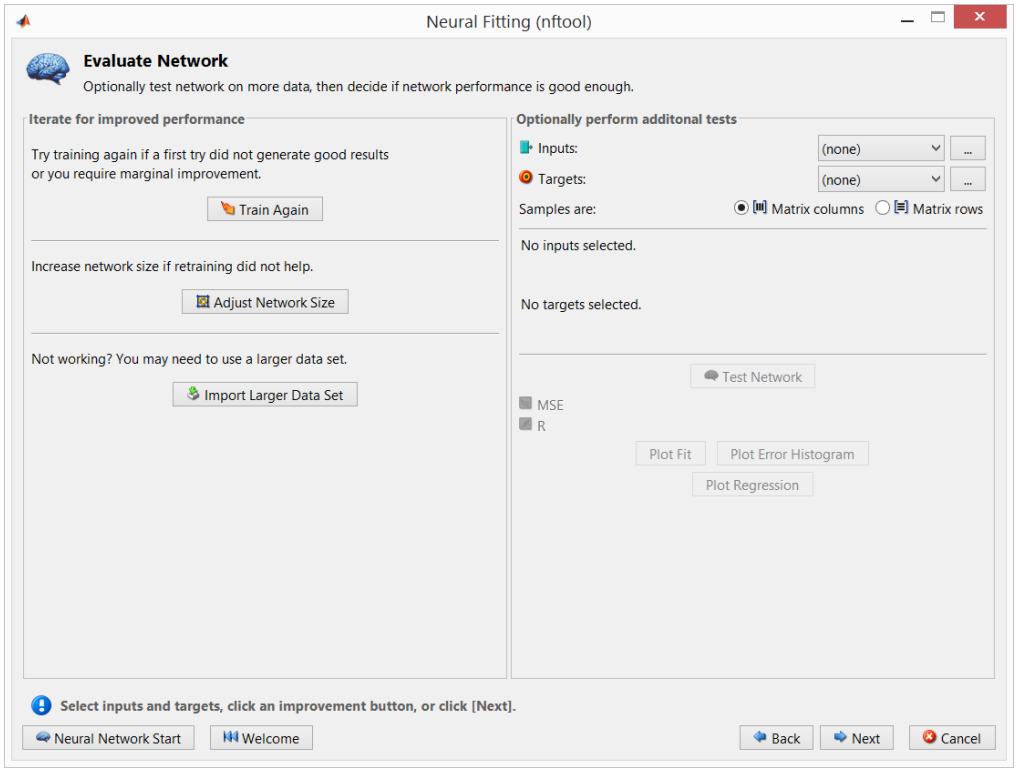


Figura 57 Janela de avaliação da rede

O sétimo painel (Figura 58) é usado para implementação da rede desenvolvida. A implementação pode passar pela criação de uma função usando a rede neuronal, ou através do *Simulink*, onde pode ser usada como bloco em diferentes sistemas, seja de controlo ou não.

O painel final (Figura 59) permite exportar os resultados para o espaço de trabalho do MATLAB. Além disso, fornece dois tipos de *script*s. O primeiro, mais simples, são exibidos os comandos equivalentes ao processo exibido nas páginas anteriores. É possível também personalizar este *script* para outras aplicações. O segundo *script*, mais avançado, exibe opções adicionais para obter uma solução o mais exata possível.

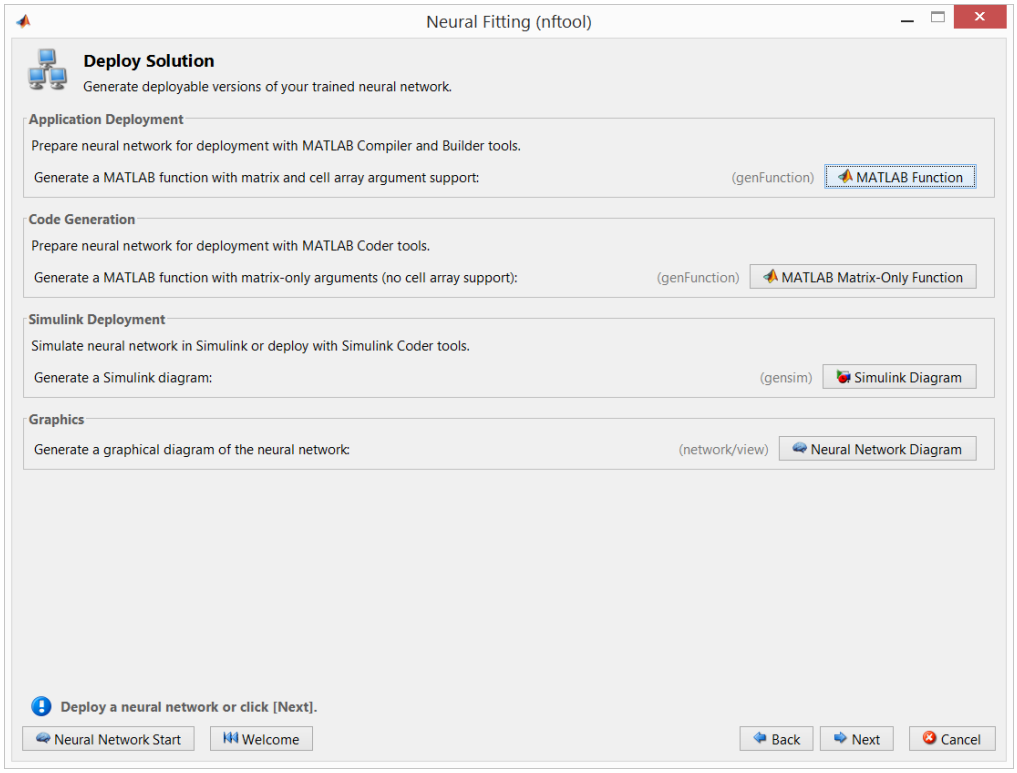


Figura 58 Janela de implementação da solução

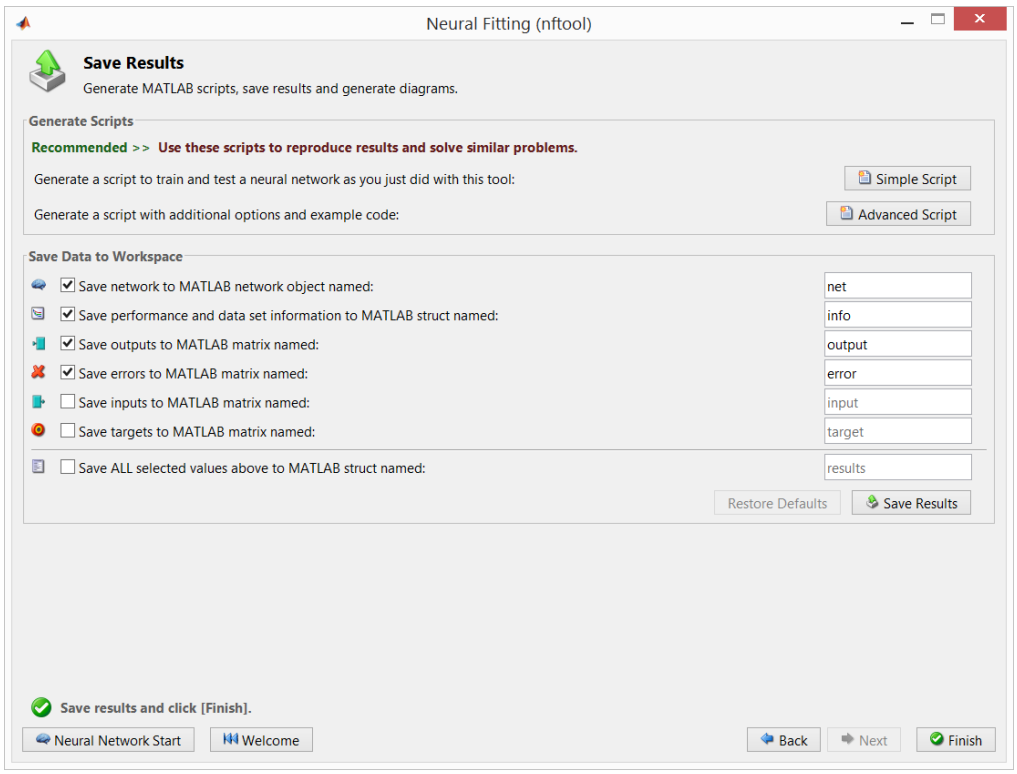


Figura 59 Janela de transferência dos resultados para a área de trabalho

3.3.2. COMANDOS MATLAB

São apresentados nas páginas seguintes alguns comandos ou funções suportados pela *Toolbox* de redes neuronais do MATLAB. Comandos esses que permitem a criação, treino e simulação de uma rede neuronal. Todos estes comandos assumem a existência de um objeto chamado 'net'. A definição desse mesmo objeto, é descrita pela seguinte sequência de código:

```
>> net = network;
```

Onde é criada uma rede "em branco", ou seja, sem quaisquer propriedades. Posteriormente é possível escolher as propriedades da rede neuronal a definir.

Esta *Toolbox* permite redes com múltiplas camadas de entrada, embora não seja muito usual. Entenda-se que esta camada de entrada referida é um conjunto de entradas da rede e não a primeira camada de neurónios que é abastecida com essas mesmas entradas. Para definir o número de camadas de entrada utiliza-se o seguinte excerto de código:

```
>> net.numInputs = 1;
```

Nas subsecções seguintes são exemplificados os diferentes comandos de criação de uma rede neuronal da *toolbox*. Para isso, dever-se-á ter em conta o diagrama da Figura 60. Esta rede será desenvolvida através dos comandos disponíveis.

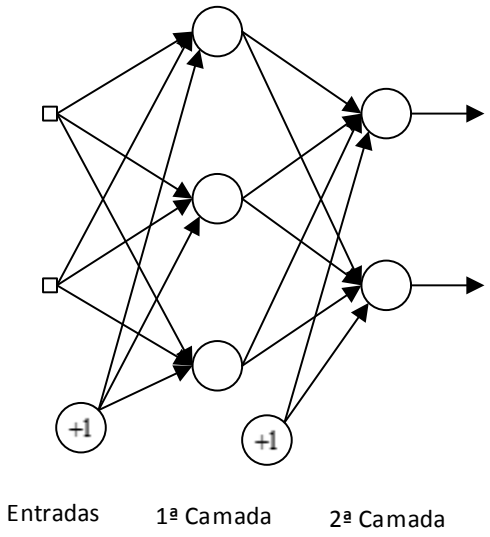


Figura 60 Diagrama da rede a implementar

CAMADAS DA REDE

Definiu-se que a rede apresenta apenas uma camada de entrada, como na maior parte dos casos. De seguida, é necessário definir quantos neurónios deverá ter a camada de entrada. O

número de neurónios deve ser igual ao conjunto dimensional dos dados. Para definir esta propriedade é usado o comando `net.inputs{i}.size`, onde `i` é o índice que representa o número da camada de entrada. Para criar uma rede com um vetor de entrada com duas dimensões, e como apenas foi definida uma camada de entrada, define-se o seguinte excerto de código:

```
>> net.inputs{1}.size = 2;
```

A propriedade seguinte a definir é o número de camadas da rede, que é traduzida pelo comando `net.numLayers`. Em conjunto com esta propriedade, é importante também definir quantos neurónios existem por camada. No excerto de código seguinte define-se que a rede tem duas camadas, sendo que a primeira terá três neurónios e a segunda um.

```
>> net.numLayers = 2;
>> net.layers{1}.size = 3;
>> net.layers{2}.size = 1;
```

Agora é tempo de definir as ligações entre camadas. Primeiro define-se a que camadas estão ligadas as entradas com o comando `net.inputConnect(i)`, sendo `i` o número da camada a ligar. As ligações entre as restantes camadas são definidas por uma matriz de conectividade (`net.layerConnect(i,j)`). Se o comando for igualado a 1, então as saídas de `i` são ligadas às entradas de `j`. De seguida, define-se qual a camada de saída configurando `net.outputConnect(i)` como 1, sendo `i` a camada a ser definida. Por último, e no caso de ser utilizado o paradigma da aprendizagem supervisionada, define-se que camada está ligada aos valores desejados configurando o comando `net.targetConnect(i)` a 1.

No excerto de código seguinte, é exibido um exemplo do processo descrito no parágrafo anterior.

```
>> net.inputConnect(1) = 1;
>> net.layerConnect(2, 1) = 1;
>> net.outputConnect(2) = 1;
>> net.targetConnect(2) = 1;
```

O último dos comandos apresentados no excerto acima é redundante, pois na última versão da *Toolbox* de redes neuronais presume-se que os valores desejados estão ligados à camada de saída. Assim, o comando `net.outputConnect` contempla o comando `net.targetConnect`.

Falta ainda definir que funções de ativação são utilizadas pelos neurónios. Nesta ferramenta a cada camada é atribuída uma função, ou seja, todos os neurónios pertencentes à mesma

camada tem a mesma função de ativação. Esta propriedade é definida pelo comando `net.layers{i}.transferFcn`, e é aplicada segundo o excerto de código abaixo.

```
>> net.layers{1}.transferFcn = 'logsig';
>> net.layers{2}.transferFcn = 'purelin';
```

PESOS E LIMIAR

A parte seguinte da configuração é a de definir pesos e *bias*. Relativamente aos *bias* é necessário atribui-los a cada camada com o código `net.biasConnect(i) = 1`, sendo *i* a camada à qual o valor de polarização está ligado. De forma semelhante, atribui-se o valor de polarização a todas as camadas da rede usando o comando:

```
>> net.biasConnect = [ 1 , 1 ];
```

Onde cada valor da matriz igualada representa uma camada, sendo que o valor 1 significa que o valor de polarização está ligado à camada respetiva.

De seguida, devem ser decididos quais os valores iniciais dos pesos das ligações (incluindo o valor de polarização). Para reiniciar os pesos utiliza-se o comando:

```
>> net = init(net);
```

Para esta inicialização utiliza-se o comando `net.initFcn`. Se o utilizador não possuir uma rotina de inicialização, deve-se igualar o comando a `'initlay'`. Assim, cada camada terá a sua rotina de inicialização. Para atribuir uma rotina a uma camada é usado o comando `net.layers{i}.initFcn`. A opção mais usual é a inicialização Nguyen-Widrow, representada pelo valor `'initlnw'`. Se o utilizador preferir escolher os valores dos pesos deve utilizar o valor `'initlwb'`.

Em baixo é apresentado um excerto de código das diferentes formas de inicialização dos pesos descritas no parágrafo anterior.

```
>> net.initFcn = 'initlay';
>> net.layers{i}.initFcn = 'initnw';
>> net.layers{i}.initFcn = 'initwb';
```

FUNÇÕES DE TREINO E PARÂMETROS

Um dos aspetos mais contraintuitivos da *Toolbox* é a distinção entre *treino* e *adaptação*. Ambas as funções são usadas para treinar uma rede neuronal, e na maioria dos casos ambas podem ser usadas pela mesma rede.

A diferença mais importante tem a ver com o treino incremental (atualização dos pesos depois da apresentação de cada amostra de treino) *versus* o treino por lote (atualização dos pesos depois da apresentação de um conjunto de dados).

Quando se usa a *adaptação*, ambas os métodos de treino podem ser usados. Dependendo do formato do conjunto de treino, pode ser selecionado o treino incremental ou por lote. Se o conjunto consistir em duas matrizes de vetores de entrada e valores desejados, a rede é atualizada usando o treino por lote. Se o conjunto de treino for dado em forma de matriz de células, então é usado o treino incremental. Por outro lado, quando se utiliza o método de *treino*, apenas é usado o treino por lote, independentemente do formato dos dados.

A grande vantagem do *treino* é que este dá uma maior escolha de funções de treino (gradiente descendente com ou sem *momentum*, Levenberg-Marquardt, etc.) que são implementadas de forma muito eficiente. Portanto, se não existir uma boa razão para aplicar o treino incremental, o método *treino* é provavelmente a melhor escolha.

Como na maior parte dos casos se utiliza o método de *treino* e não o de *adaptação*, apenas são abordados nesta subsecção os parâmetros configuráveis para o primeiro método.

Primeiro é necessário configurar qual o algoritmo ou regra a usar, e os seus parâmetros, para o treino da rede. Estes valores devem ser especificados no comando `net.trainFcn`, onde é identificado o algoritmo, e no comando `net.trainParam`, para os parâmetros.

No excerto de código seguinte são preenchidos os campos acima descritos. O método de treino definido é gradiente descendente com *momentum*.

```
>> net.trainFcn = 'traingdm';
>> net.trainParam.lr = 0.1;
>> net.trainParam.mc = 0.9;
```

Onde `lr` representa o taxa de aprendizagem, e `mc` o termo de *momentum*.

Dois outros parâmetros úteis são: `net.trainParam.epochs`, que representa o número máximo de vezes que o conjunto de dados pode ser usado para treino, e `net.trainParam.show`, que representa o tempo entre relatórios de estado da função de treino. No excerto de código seguinte é demonstrado um exemplo:

```
>> net.trainParam.epochs = 10000;
>> net.trainParam.show = 100;
```

O passo seguinte é a execução do comando treino da rede criada, e, posteriormente, a sua simulação. Os parâmetros de treino já abordados são meramente opcionais, pois se não forem especificados, o MATLAB assume os valores de omissão. Contudo, é necessário existir uma matriz de entrada, que contêm os padrões de treino/entrada, e uma matriz de saída, que contêm os valores desejados para cada padrão. As matrizes devem respeitar a estrutura da rede, isto é, cada padrão de treino de ter a mesma dimensão que o número de entradas da rede e o mesmo acontece com a matriz de saída. Assim, e tendo em conta a estrutura da rede a implementar, é apresentado abaixo um excerto de código com duas matrizes possíveis a utilizar, sendo P a matriz de entrada e T a matriz dos valores desejados.

```
P = {[0.5;0] [0.3;0.6] [0.2;0.4] [0.5;0.4]
[0.2;0.6] [1;0] [0.1;0.2] [0.6;0.1] [0.9;0.1]
[0.2;0.2]}
T = {0.5 0.9 0.6 0.9 0.8 1 0.3 0.7 1 0.4}
```

A lógica destes valores segue a equação $T = P_A + P_B$ com intervalo entre 0 e 1, ou seja, a soma dos valores de entrada é igual ao valor de saída que será o valor entre 0 e 1. Desta forma torna-se mais fácil testar a rede para valores aleatórios.

Para treinar a rede usando as matrizes acima, é usado o comando `train`. Os argumentos deste comando são: o objeto criado para representar a rede, a matriz que contêm os padrões de entrada, e a matriz com os valores desejados. Estes deverão ser colocadas na mesma ordem em que foram mencionados como sugere o excerto de código abaixo.

```
>> net = train(net,P,T);
```

Depois de treinada é necessário avaliar o desempenho da rede. As duas opções mais usadas são o erro médio absoluto (*mae*) e erro médio quadrático (*mse*). A primeira das opções é geralmente usada em redes de classificação, enquanto a segunda é usada para redes de aproximação de funções.

A função de desempenho é definida no comando `net.performFcn`, como é demonstrado no excerto de código seguinte:

```
>> net.performFcn = 'mse';
```

Por fim, para simular a rede é usado o comando `sim`, sendo os argumentos o objeto que representa a rede e a matriz de entrada, como sugere o excerto de código abaixo. Na variável `Y` é guardado o valor de saída da rede. O objetivo é que a rede seja capaz de generalizar, ou seja, com diferentes padrões de entrada esta seja capaz de mapear a relação entrada-saída da mesma forma.

```
>> Y = sim(net,P);
```

Depois de elaborado o ficheiro para a conceção da rede neuronal a desenvolver, é possível testar e analisar os resultados obtidos pela rede neuronal. Inicialmente é executado o comando `view(net)` que mostra a configuração da rede. Através da Figura 61 pode-se afirmar que a rede tem a configuração pretendida.

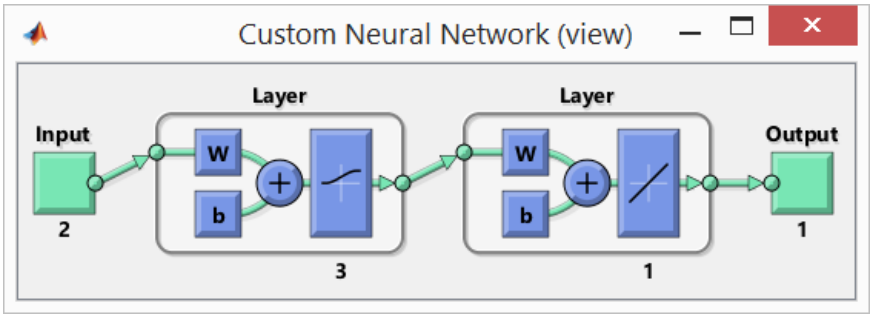


Figura 61 Diagrama da rede neuronal (MATLAB)

Relativamente aos resultados, é possível fazer uma análise rápida através da Figura 62. Verifica-se que, depois de treinada, a rede apresenta resultados relativamente satisfatórios quando estimulada pelos padrões de treino.

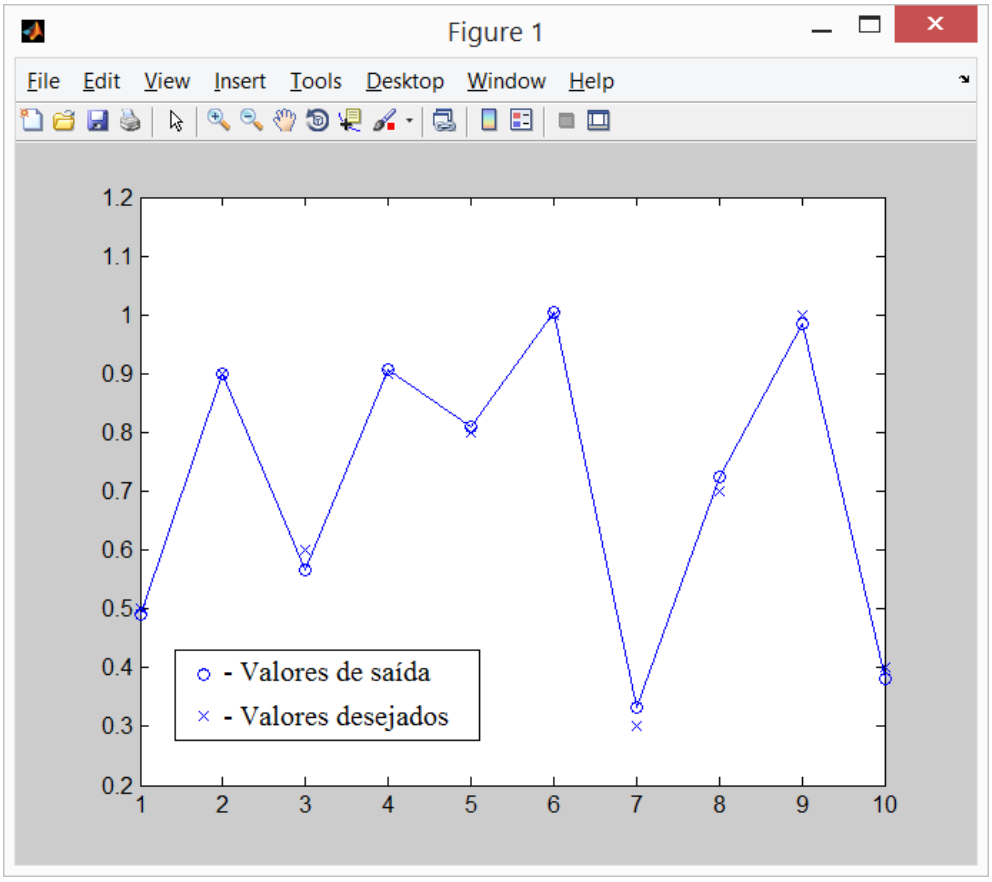


Figura 62 Gráfico de comparação dos resultados da rede neuronal desenvolvida

Para finalizar, é testado um padrão de entrada aleatório, de modo a confirmar que a rede neuronal foi capaz de adquirir a competência para generalizar. O comando de teste e o respetivo resultado é apresentado no excerto de código abaixo.

```
>> net([0.3;0.6])

ans =

    0.8995
```

Resume-se no excerto de código abaixo todos os comandos utilizados para gerar, treinar e simular a rede neuronal apresentada nesta secção. Este pode ser usado como modelo de partida para o desenvolvimento de uma rede mais complexa.

```
net = network;                 % Criação da rede
como objeto
net.numInputs = 1;             % Nº de camadas
de entrada
net.inputs{1}.size = 2;        % Nº entradas na
camada 1

net.numLayers = 2;              % Nº de camadas
de neurónios
net.layers{1}.size = 3;        % Nº de neurónios
na camada 1
net.layers{2}.size = 1;        % Nº de neurónios
na camada 2

net.inputConnect(1) = 1;       % Ligação entre
as entradas e camada 1
net.layerConnect(2,1) = 1;     % Ligação entre a
camada 1 e 2
net.outputConnect(2) = 1;      % Definição que a
camada de saída e a camada 2

net.layers{1}.transferFcn = 'logsig';     %
Definir a função sigmoid para a camada 1
net.layers{2}.transferFcn = 'purelin';    %
Definir a função linear para a camada 2

net.biasConnect = [1;1];       % Valor de
polarização ligado às duas camadas

net.initFcn = 'initlay';
net.layers{1}.initFcn = 'initnw';
net.layers{2}.initFcn = 'initnw';

net = init(net)

view(net)
```

```
net.trainFcn = 'traingdm';
net.trainParam.lr = 0.1;         % Taxa de
aprendizagem
net.trainParam.mc = 0.9;         % Momentum

net.trainParam.epochs = 10000;    % Máximo de
iterações
net.trainParam.show = 100;        % Intervalo
de iterações que são exibidas

% Matrizes de entrada e saída -> valores
desejados são a soma dos valores
% de entrada
P = {[0.5;0] [0.3;0.6] [0.2;0.4] [0.5;0.4]
[0.2;0.6] [1;0] [0.1;0.2] [0.6;0.1] [0.9;0.1]
[0.2;0.2]}
T = {0.5 0.9 0.6 0.9 0.8 1 0.3 0.7 1 0.4}

% Comando de treino
net = train(net,P,T)

% Simulação da rede
Y = sim(net,P)

% Graficos (Comparação entre valores obtidos
e desejados)
plot(1:10,cell2mat(T),'x')
hold on
plot(1:10,cell2mat(Y),'-o')

% TESTE
%Resultado = net([a;b])
```

4. DESENVOLVIMENTO

Neste capítulo são apresentadas e explicitadas as experiências realizadas, tendo em vista a comparação de diferentes sistemas de controlo de um processo. Alguns dos modelos implementados já foram descritos no capítulo anterior, não sendo, contudo, abordados com tanta profundidade como nas secções seguintes.

O processo escolhido foi o de um motor DC, que é controlado para executar uma manobra mecânica precisa. A velocidade do eixo deverá seguir uma trajetória especificada, independentemente das variações de carga e outros parâmetros desconhecidos.

As redes neuronais devem ser treinadas para emular a dinâmica não linear do processo mediante a apresentação de padrões de entrada/saída gerados pelo processo. Assim que a dinâmica do sistema for identificada pela rede neuronal, podem ser aplicadas várias técnicas convencionais de controlo para que o processo acompanhe a trajetória de referência.

Os modelos de controlo aqui utilizados e testados são: o controlador estabilizador e o controlador de referência adaptativo.

De entre os vários sistemas de controlo aplicados, será feito um estudo do desempenho de modo a encontrar qual o melhor para o processo em questão.

Este estudo envolve também uma comparação na diferença de resultados obtidos entre sistemas que contêm apenas o controlador PID e sistemas com PID e rede neuronal, e em que medida a inclusão desta última melhora a resposta do sistema. Assim, cada sistema deve apresentar um controlo híbrido PID-Neuronal.

Para melhor entendimento do tipo de controlo PID, foi incluída uma secção com uma pequena introdução e contextualização a este controlador, explicitando também como este funciona.

Por fim, são abordados índices de desempenho utilizados no tratamento dos dados de modo a avaliar o desempenho de cada sistema implementado, retirando as devidas conclusões.

4.1. DESCRIÇÃO DO PROCESSO

O processo a controlar, tal como já foi dito, é um motor DC, e nesta secção será apresentado o modelo deste processo. Embora não seja obrigatório adquirir o seu modelo, para este caso torna-se bastante útil. Através da sua simulação, é possível extrair os padrões de entrada/saída para posteriormente treinar a rede neuronal, tal como acontece numa situação de utilização real.

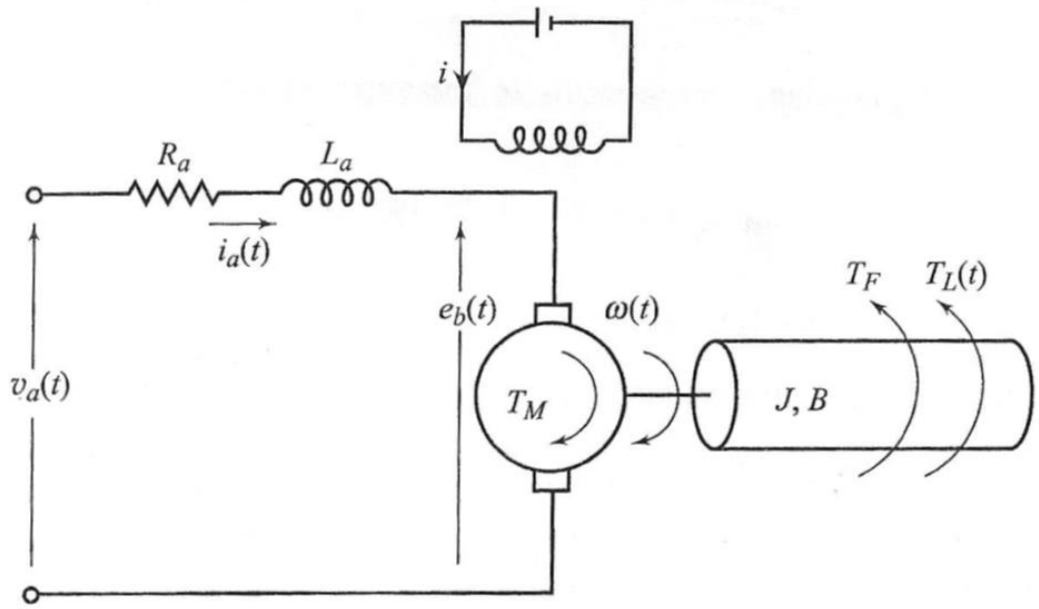


Figura 63 Dinâmicas do motor DC [57]

As dinâmicas do motor (Figura 63) são dadas pelas seguintes expressões:

$$v_a(t) = R_a i_a(t) + L_a \frac{di_a}{dt} + e_b(t) \tag{39}$$

$$e_b(t) = K_b \omega(t) \tag{40}$$

$$T_M(t) = K_T i_a(t) = J \frac{d\omega(t)}{dt} + B\omega(t) + T_L(t) + T_F \tag{41}$$

Onde:

$v_a(t)$ = Tensão aplicada à armadura;

$e_b(t)$ = Força eletromotriz inversa;

$i_a(t)$ = Corrente na armadura;

R_a = Resistência dos rolamentos da armadura;

L_a = Indutância dos rolamentos da armadura;

$\omega(t)$ = Velocidade angular do rotor do motor;

$T_M(t)$ = Binário desenvolvido pelo motor;

K_T = Constante de binário;

K_b = Constante da força eletromotriz inversa;

J = Momento de inércia do rotor do motor com a carga mecânica anexada;

B = Coeficiente de fricção do rotor do motor com a carga mecânica anexada;

$T_L(t)$ = Distúrbios do binário de carga;

T_F = Binário friccional.

O binário de carga $T_L(t)$ pode ser expresso por:

$$T_L(t) = \psi(\omega) \tag{42}$$

Onde a função $\psi(.)$ depende da natureza da carga.

Para a maioria dos sistemas de condução de carga, a função $\psi(.)$ toma a seguinte forma:

$$T_L(t) = \mu\omega^2[sgn\ \omega(t)] \tag{43}$$

Onde μ é uma constante.

O sistema de condução do motor é expresso por um processo SISO combinando as equações (39) e (40):

$$\begin{aligned} L_aJ\frac{d^2\omega(t)}{dt^2} + (R_aJ + L_aB)\frac{d\omega(t)}{dt} + (R_aB + K_bK_T)\omega(t) + L_a\frac{dT_L(t)}{dt} \\ + R_a[T_L(t) + T_F] + K_Tv_a(t) = 0 \end{aligned} \tag{44}$$

O modelo discreto é derivado substituindo todas as diferenças contínuas utilizando o método das diferenças atrasadas [56]. Desenvolvendo a expressão, é possível chegar ao seguinte modelo do processo:

$$\begin{aligned} \omega(k+1) = K_1\omega(k) + K_2\omega(k-1) + K_3[sgn(\omega(k))]\omega^2(k) \\ + K_4[sgn(\omega(k))]\omega^2(k-1) + K_5v_a(k) + K_6 \end{aligned} \tag{45}$$

$$\begin{aligned} K_1 &= \frac{2L_aJ + T(R_aJ + L_aB) - T^2(R_aB + K_bK_T)}{L_aJ + T(R_aJ + L_aB)} \\ K_2 &= -\frac{L_aJ}{L_aJ + T(R_aJ + L_aB)} \\ K_3 &= -\frac{T(\mu L_a + \mu R_aT)}{L_aJ + T(R_aJ + L_aB)} \\ K_4 &= \frac{T\mu L_a}{L_aJ + T(R_aJ + L_aB)} \\ K_5 &= \frac{K_TT^2}{L_aJ + T(R_aJ + L_aB)} \\ K_6 &= -\frac{T_FR_aT^2}{L_aJ + T(R_aJ + L_aB)} \end{aligned} \tag{46}$$

Os valores dos parâmetros associados ao motor DC podem ser encontrados na Tabela 3 [57].

Tabela 3 Valores dos parâmetros do motor DC

J (kg-m2)	0,068
B (N-m/(rad/s))	0,03475
R_a (Ω)	7,56
L_a (H)	0,055
K_T (N-m/amp)	3,475
K_b (volts/(rad/s))	3,475
μ (N-m/(rad/s)^2)	0,0039
T_F (N-m)	0,212
T (ms)	40

Com estes parâmetros, as constantes K_1, K_2, K_3, K_4, K_5 e K_6, tomam os valores apresentados na Tabela 4.

Tabela 4 Constantes do modelo discreto do motor DC

K_1	0,34366
K_2	$-0{,}1534069$
K_3	$-2{,}286928 * 10^{-3}$
K_4	$3{,}5193358 * 10^{-4}$
K_5	0,2280595
K_6	$-0{,}105184$

Para emular o motor DC é utilizada a expressão (45) que, complementada com os valores da tabela acima, resulta no esquema apresentado no Anexo A, Secção 7. Este é utilizado nas simulações realizadas em *Simulink*, mais especificamente para o controlador estabilizador. Outro método análogo de imitação do processo é o uso de uma rede neuronal depois de treinada. Porém, as entradas e saídas da mesma devem ser restringidas aos limites de operação do processo [57]. Assim, em conformidade com as limitações do *hardware* mecânico e elétrico do motor, são definidos os seguintes limites de operação:

$$-30 < \omega(k) < 30 \text{ rad/s}$$

$$|\omega(k-1) - \omega(k)| < 1{,}0 \text{ rad/s} \tag{47}$$

$$|v_a(k)| < 100 \text{ volts}$$

4.2. CONTROLADOR PID

O controlador PID (controlador proporcional integrativo derivativo) é a forma mais comum de controlo por realimentação. Devido à sua simplicidade e excelente desempenho em muitas aplicações, estes controladores são usados em 95% dos processos industriais de malha-fechada [54]. Tal utilização deve-se ao facto de ser facilmente implementável, de baixo custo, versátil e capaz de alterar os comportamentos transitórios e de regime permanente dos processos sob controlo. Atualmente, a maioria dos processos automatizados que utilizam PLCs (*Programmable Logic Controller*), possuem malhas de controlo de algoritmos PID, sendo responsabilidade dos engenheiros/técnicos a tarefa de sintonia dos parâmetros dos controladores. A sua adaptação às novas mudanças na tecnologia permitiu o aumento de recursos adicionais tais como o ajuste automático, a programação dos ganhos e a adaptação contínua.

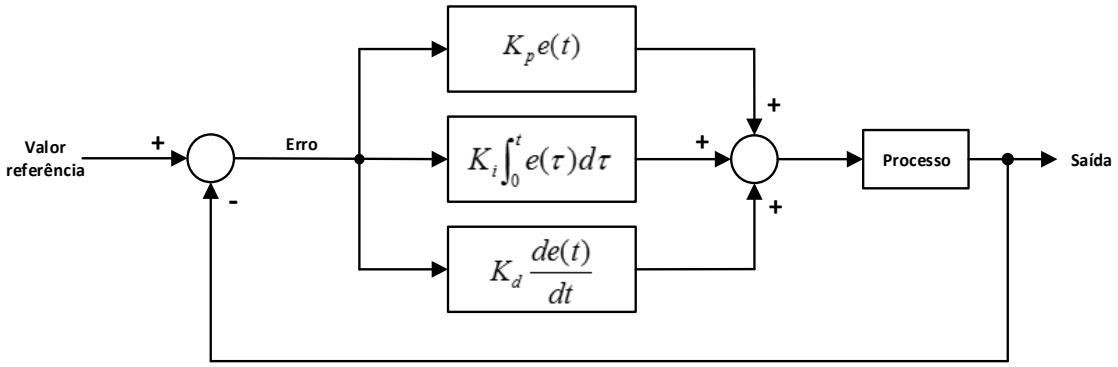


Figura 64 Esquema do controlador PID num sistema de realimentação

Muito resumidamente, este tipo de controlador calcula o erro entre o valor medido e o valor desejado ou de referência, e tenta minimizá-lo ajustando as entradas que controlam o processo, como sugere a Figura 64. O cálculo da ação do controlador envolve, separadamente, três ações: proporcional (P), integral (I), e derivativa (D). Com a soma destas três componentes é possível calcular a ação de controlo a tomar, como sugerido na equação seguinte:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt} \tag{48}$$

Sendo os ganhos integral (K_i) e derivativo (K_d) dados pelas seguintes equações, respetivamente:

$$K_i = K_p * \frac{1}{T_i} \tag{49}$$

$$K_d = K_p * T_d \tag{50}$$

Cada ganho provoca diferentes reações na resposta do processo a controlar. Dependendo do seu valor, a sua utilização pode melhorar ou piorar a resposta. Assim, na hora de definir estes valores, há que escolher qual a melhor forma de o fazer. O método mais básico e simples é o da *tentativa-erro*. Nesta forma de sintonia, o utilizador ajusta os ganhos conforme as suas influências na resposta do processo, até encontrar a melhor saída possível. Contudo, este está longe de ser o melhor método de sintonia.

Abaixo são abordados os esquemas de ajuste dos ganhos utilizados no desenvolvimento deste trabalho.

4.2.1. ZIEGLER-NICHOLS

De entre diferentes e variados métodos de sintonia, o Ziegler-Nichols é o mais conhecido e utilizado. Este foi desenvolvido por John G. Ziegler e Nathaniel B. Nichols em 1942, através do reconhecimento das respostas de vários processos a um degrau. Essas respostas exibiam uma curva de reação como a apresentada na Figura 65. A curva em *S* é característica de muitos sistemas físicos, e as suas funções de transferência podem ser aproximadas por:

$$\frac{Y(s)}{U(s)} = \frac{Ke^{-Ls}}{Ts + 1} \tag{51}$$

Sendo esta expressão um sistema de primeira ordem com o tempo de atraso de *L* segundos. As constantes da equação podem ser determinada pela resposta ao degrau (Figura 65) do processo. Se for desenhada uma reta tangente no ponto de inflexão da curva, o declive dessa linha será igual à expressão (52), e a intersecção com o eixo do tempo igual ao atraso da resposta (*L*).

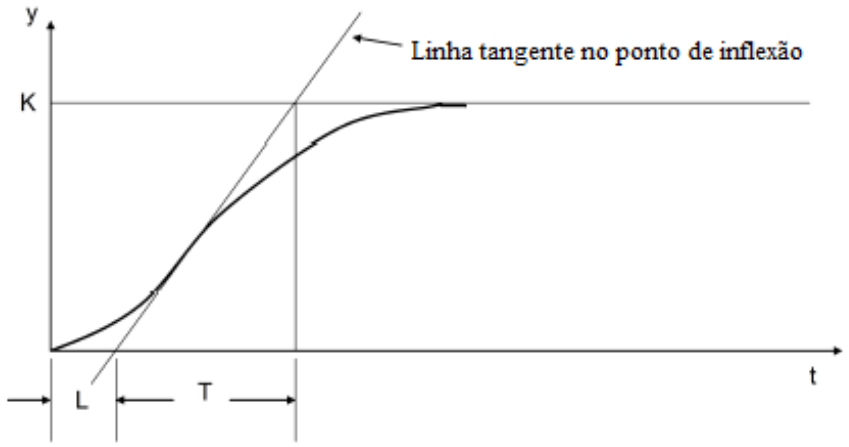


Figura 65 Esquematização dos parâmetros da resposta do sistema a um degrau

$$R = \frac{K}{T} \tag{52}$$

Usando este modelo, são fornecidos dois métodos heurísticos de sintonia do controlador. Para o primeiro, a escolha dos ganhos é baseado numa taxa de decaimento de aproximadamente 0,25. Isto significa que após um período de oscilação, a resposta cai para um quarto do seu valor inicial. Este quarto de queda garante uma boa relação entre uma resposta rápida e uma margem de estabilidade adequada. Os autores simularam as equações e ajustaram os parâmetros de controlo até ser exibido este tipo de comportamento, sugerindo as expressões expostas na Tabela 5.

Tabela 5 Resumo das regras de sintonia para a heurística de Ziegler-Nichols em malha aberta

Tipo de controlador	K_p	T_i	T_d
P	$\frac{1}{RL}$	∞	0
PI	$\frac{0,9}{RL}$	$\frac{L}{0,3}$	0
PID	$\frac{1,2}{RL}$	$2L$	$0,5L$

Para o segundo método, os critérios de ajuste dos parâmetros são baseados na avaliação do sistema no seu limite de estabilidade, ou seja, quando este exibe uma resposta oscilatória estável. O processo a controlar é simulado num sistema de realimentação, e o ganho proporcional é aumentado gradualmente até serem observadas oscilações contínuas (Figura

66). O valor do ganho correspondente é designado de ganho crítico (K_u), e o período da onda de período crítico (P_u).

Por último, são calculados os parâmetros do controlador escolhido de acordo com as expressões da Tabela 6.

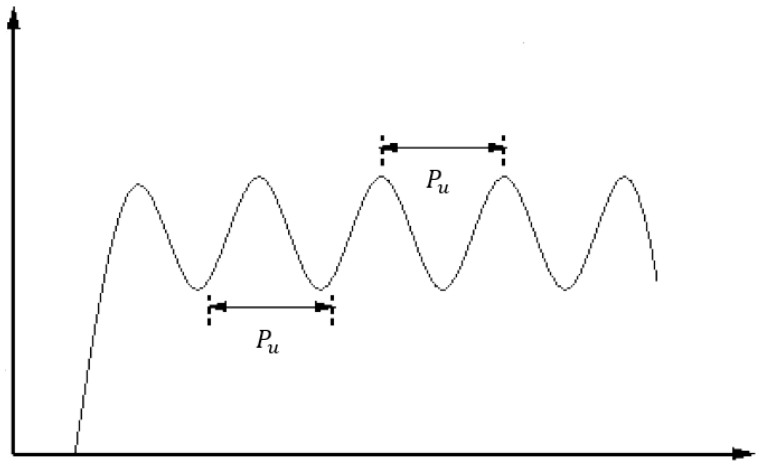


Figura 66 Resposta ao ganho crítico

Tabela 6 Resumo das regras de sintonia para a heurística de Ziegler-Nichols em malha fechada

Tipo de controlador	K_p	T_i	T_d
P	$0{,}5K_u$	∞	0
PI	$0{,}45K_u$	$\frac{1}{1{,}2}P_u$	0
PID	$0{,}6K_u$	$\frac{1}{2}P_u$	$\frac{1}{8}P_u$

4.2.2. COON-COHEN

O método de sintonia Coon-Cohen é similar ao de Ziegler-Nichols pois também se trata de um método heurístico. Este destaca-se por uma maior flexibilidade em comparação com o Ziegler-Nichols, adequado para um maior número de aplicações, e por ser um método com regras de sintonia para controladores PD.

O método de sintonia inicia-se da mesma forma que o Ziegler-Nichols em malha aberta, pois é necessário calcular o modelo de primeira ordem do processo com atraso descrito pela equação (51). Posteriormente, utilizando as equações (53) e (54), é possível calcular os parâmetros para o controlador através das expressões da Tabela 7.

$$a = \frac{KL}{T} \tag{53}$$

$$\tau = \frac{L}{L+T} \tag{54}$$

Tabela 7 Resumo das regras de sintonia para a heurística de Coon-Cohen

Tipo de controlador	K_p	T_i	T_d
P	$\frac{1}{a}\left(1+\frac{0{,}35\tau}{1-\tau}\right)$	0	0
PI	$\frac{0{,}9}{a}\left(1+\frac{0{,}92\tau}{1-\tau}\right)$	$\frac{3{,}3-3\tau}{1+1{,}2\tau}L$	0
PD	$\frac{1{,}24}{a}\left(1+\frac{0{,}13\tau}{1-\tau}\right)$	0	$\frac{0{,}27-0{,}36\tau}{1-0{,}87\tau}L$
PID	$\frac{1{,}35}{a}\left(1+\frac{0{,}18\tau}{1-\tau}\right)$	$\frac{2{,}5-2\tau}{1-0{,}39\tau}L$	$\frac{0{,}37-0{,}37\tau}{1-0{,}81\tau}L$

4.2.3. ZHUANG-ATHERTON

Este é um método prático para sintonia de controladores PI e PID, desenvolvido por Zhuang e Atherton em 1993. Ao contrário dos outros dois explicitados acima, este é um método de sintonia ótimo baseado no critério integral do erro. Considera-se a seguinte expressão como critério de otimização:

$$J_n(\theta) = \int_0^\infty [t^n e(\theta, t)]^2 \, dt \tag{55}$$

Onde $e(\theta, t)$ é o sinal do erro que entra no controlador PID, e θ seus parâmetros. São também considerados os seguintes valores de n, $n = 0{,}1{,}2$. Estes três casos correspondem, respetivamente, a três critérios de otimização: integral quadrático do erro (ISE), integral do quadrado do produto do tempo com o erro (ISTE), e integral do quadrado do produto do tempo quadrático com o erro (IST^{2}E).

Deste procedimento são propostas duas estratégias: uma baseada na entrada de referência e outra baseada na rejeição a perturbações. Neste estudo foi preferido o uso da segunda das estratégias, de modo a perceber se este tipo de sintonia é capaz de lidar com o ruído a ser aplicado ao sistema.

Para este procedimento, se o processo em questão for capaz de ser representado pela função de transferência de primeira ordem (51), podem ser utilizadas as seguintes expressões para cálculo dos parâmetros do PID:

$$K_p = \frac{a_1}{T}\left(\frac{L}{T}\right)^{b_1}$$

$$T_i = \frac{T}{a_2}\left(\frac{L}{T}\right)^{b_2} \tag{56}$$

$$T_d = a_3 T\left(\frac{L}{T}\right)^{b_3}$$

Consoante o critério de que se pretenda utilizar e a extensão de L/T, substitui-se as variáveis das expressões em (56) pelo valores apresentados na Tabela 8.

Tabela 8 Parâmetros para o controlador da sintonia de Zhuang-Atherton para rejeição de perturbações

Extensão de L/T	$0{,}1-1$			$1{,}1-2$		
Critério	ISE	ISTE	IST^{2}E	ISE	ISTE	IST^{2}E
a_1	1,473	1,468	1,531	1,524	1,515	1,592
b_1	$-0{,}970$	$-0{,}970$	$-0{,}960$	$-0{,}735$	$-0{,}730$	$-0{,}705$
a_2	1,115	0,942	0,971	1,130	0,957	0,957
b_2	0,753	0,725	0,746	0,641	0,598	0,597
a_3	0,550	0,443	0,413	0,552	0,444	0,414
b_3	0,948	0,939	0,933	0,851	0,847	0,850

4.3. CONTROLADOR ESTABILIZADOR FIXO

O primeiro dos sistemas é denominado por controlo estabilizador fixo. Este tipo de controlo é abordado na subsecção 3.1.1, e pode ser dividido em duas partes: um controlo por realimentação e um controlo direto. A vantagem de um sistema com dois sistemas de controlo é a possibilidade de dividir tarefas de controlo e melhorar assim o desempenho da resposta do processo.

O primeiro tipo de controlo é responsável por controlar a saída do processo, e garantir que esta siga o sinal de entrada de referência. A rede neuronal é usada para garantir o cumprimento desta tarefa.

O segundo tipo de controlo tem pouca ou nenhuma influência na saída do processo. Contudo, é este que reage a distúrbios que deteriorem os sinais do sistema que posteriormente influenciem a saída do processo. O controlador PID é usado para garantir o cumprimento desta tarefa.

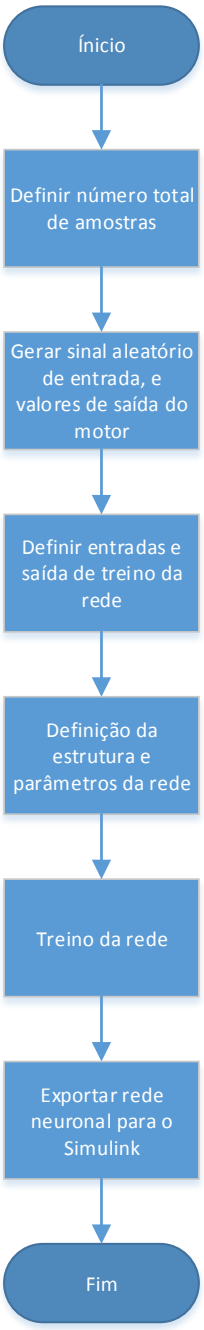


Figura 67 Fluxograma do *script* para treino do modelo inverso do processo

Antes da simulação do sistema, é necessário treinar a rede neuronal para recriar o mapeamento saída-entrada do processo (modelação inversa). Para isso é criado um *script* no MATLAB. A sequência de passos realizados encontra-se no fluxograma da Figura 67. O código correspondente pode ser encontrado no Anexo A, Secção 1. As subsecções permitem a divisão das diferentes etapas de treino da rede de modo a associar às descrições realizadas nos parágrafos seguintes.

Inicialmente é necessário definir o número total de amostras para os valores de treino. Esse valor é utilizado para limitar o número de pares entrada-saída gerados. O sinal aleatório de tensão é a entrada da rede, e a respetiva resposta do motor a saída desejada da rede.

Em seguida, é gerada e configurada a rede neuronal. Primeiro definem-se as entradas e saída da rede usadas para o treino. As entradas são compostas pelos valores de velocidade angular do processo nos tempos de amostragem iguais a $k+2$, $k+1$, e k, como é demonstrado no código Anexo A, Secção 1.2. A saída corresponde ao valor desejado que se pretende que a rede neuronal reproduza, pois o modelo neuronal deve representar a dinâmica inversa do sistema. O valor desejado é a tensão correspondente ao último valor de saída, daí estar registado no tempo de amostragem $k+1$.

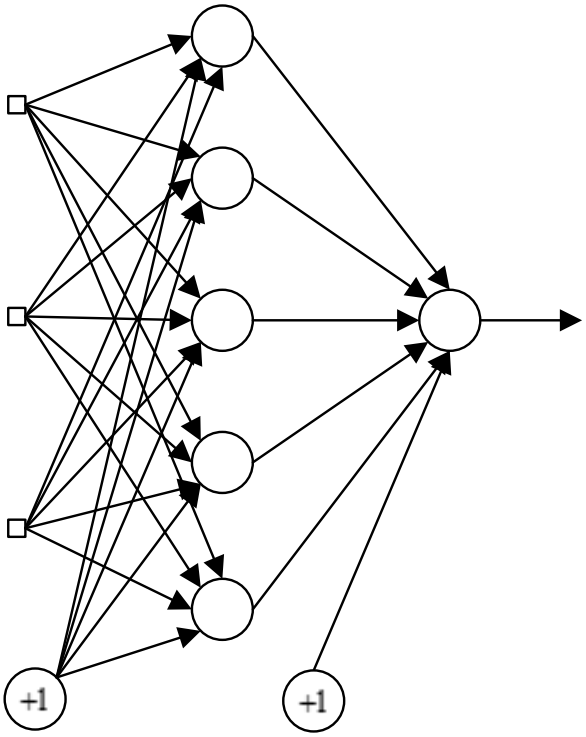


Figura 68 Diagrama da rede neuronal do controlador estabilizador

Posteriormente, foi configurada a rede com a estrutura apresentada na Figura 68. De acordo com as definições de valores de entrada desejados é obrigatório que a rede neuronal tenha três entradas. É também composta por duas camadas de neurónios: uma invisível e uma de saída. A camada invisível é composta por 5 neurónios. Foram testados vários valores relativos ao número de neurónios desta camada, sendo este o que demonstrou melhor desempenho. A camada de saída é composta por um neurónio, pois apenas existe uma variável de entrada do sistema, que é a tensão v_a.

As funções de transferência são definidas por camada. Para a camada invisível foi definida a função logarítmica com intervalo [-1;1], e para a camada de saída foi definida a função linear. A escolha das funções de ativação seguem o critério de desempenho da rede, ou seja, a definição das mesmas para as respetivas camadas garante um melhor desempenho comparativamente a outras funções. Relativamente aos pesos das ligações, estes são inicializados com valores aleatórios entre [-0,5;0,5].

Por último, é necessário configurar os parâmetros de treino e inicializar a rede, tais como: algoritmo de treino (*trainFcn*), número máximo de iterações (*epochs*), valor de erro mínimo (*goal*), taxa de aprendizagem (*lr*), e função de desempenho (*performFcn*). O algoritmo de treino utilizado é o de retropropagação de Levenberg-Marquard. Este é um método padrão de minimização do erro quadrático, devido às suas propriedades de rapidez de convergência e robustez. A função de desempenho utilizada é a do erro quadrático.

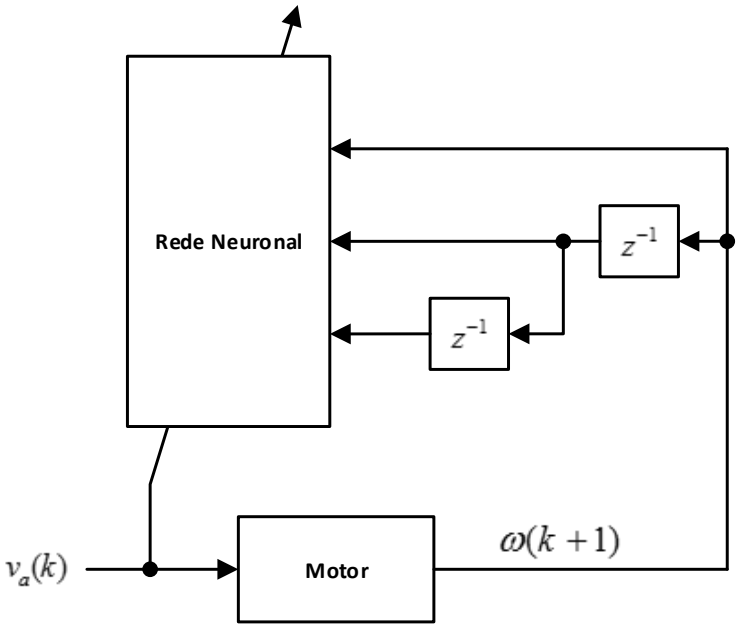


Figura 69 Diagrama de treino da rede neuronal (Modelação Inversa)

Estão reunidas todas as condições para o treino e teste da rede neuronal. O treino é executado através do comando `train` como já foi visto no capítulo anterior. Utilizando o conjunto de valores e a estrutura da rede gerados nas etapas anteriores, é realizado o treino da rede segundo o diagrama da Figura 69. Para o teste da rede, é elaborado um sinal de referência (velocidade angular) que é usado como entrada da rede neuronal, segundo o esquema da Figura 70.

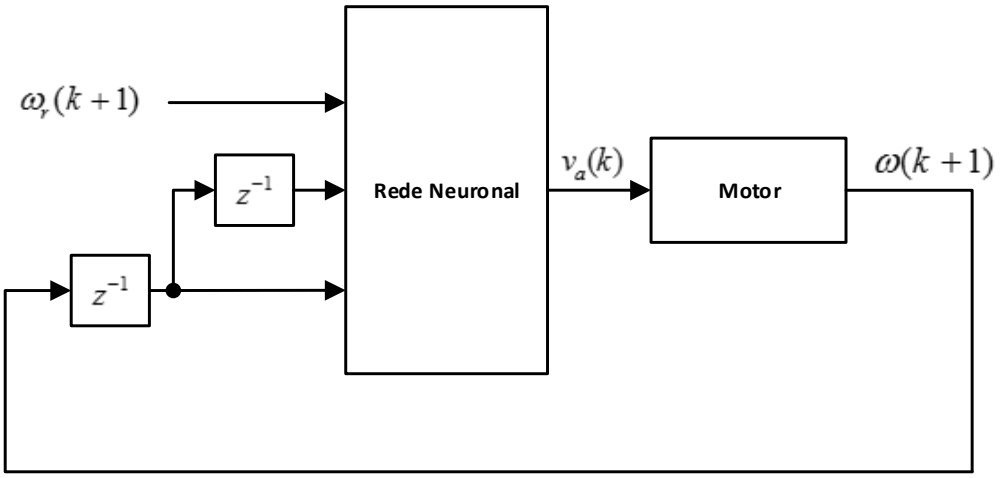


Figura 70 Diagrama de teste da rede neuronal

Se o teste efetuado demonstrar claramente que a rede neuronal foi capaz de reproduzir a dinâmica inversa do sistema, é possível avançar para a fase seguinte: implementação da rede neuronal no sistema de controlo.

O esquema do sistema de controlo a implementar, controlador estabilizador, é apresentado na Figura 39. Adaptando esse mesmo esquema ao trabalho desenvolvido, chega-se ao diagrama de blocos da Figura 71. As diferenças dos dois esquemas são três: a primeira é a adição de dois blocos de normalização que são nada mais do que ganhos utilizados para converter os valores de entrada e saída para intervalos de valores compatíveis com a rede neuronal; a segunda é a adição de um bloco de saturação de modo a limitar a tensão aplicada ao processo entre [-100;100] volts; e, por fim, a terceira é a falta do bloco relativo ao algoritmo adaptativo, pois a rede neuronal é treinada *a priori*, daí não ser necessário essa mesma componente.

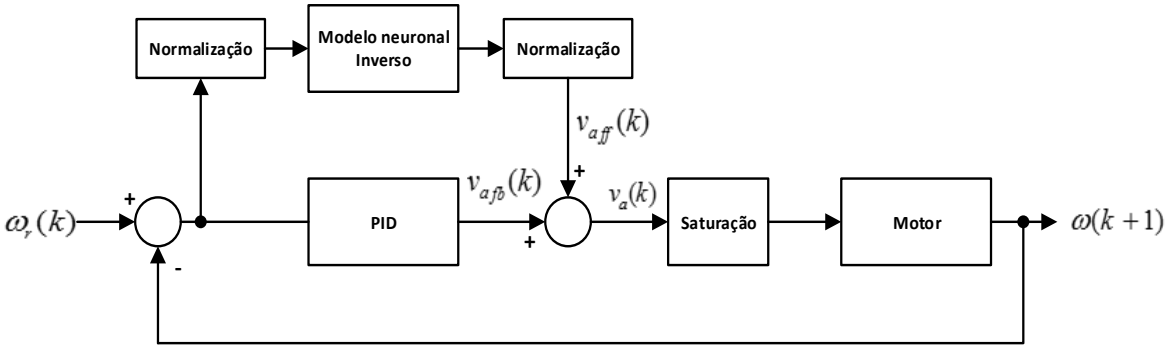


Figura 71 Diagrama de aplicação do controlador estabilizador

Um dos blocos mais importantes do esquema do controlador estabilizador é o do PID, pois este também é um dos alvos do estudo aqui desenvolvido. O PID é sintonizado com os métodos descritos na secção 4.2. Este procedimento inicia-se com simulação do processo com entrada em degrau unitário e malha aberta, de modo a obter as variáveis da equação de primeira ordem com atraso. Depois de extraída a resposta do processo, é utilizada a função do MATLAB `fmincon`. Esta permite encontrar um mínimo restrito de uma função objetivo de múltiplas variáveis. A função objetivo é a equação de primeira ordem descrita em (51) que tem como variáveis K, T, e L.

No excerto de código exibido no Anexo A, Secção 2, `x0` é o vetor de valores iniciais das variáveis, `Lb` e `Ub` são os limites mínimos e máximo das mesmas, `fun_equacao1` é a função objetivo, `x` é o vetor que contém os valores das variáveis para o mínimo encontrado, e `fval` é o valor mínimo da função encontrado. Para este último parâmetro, é necessário passar em argumento os valores da resposta obtida ao degrau unitário em malha aberta de modo a calcular o erro entre a mesma e a função objetivo. Por fim, são aplicadas as expressões de cálculo dos métodos de sintonia já apresentadas na secção 4.2.

Os testes respetivos a este controlador são realizados no ambiente de simulação *Simulink* no *software* MATLAB, e são apresentados no capítulo seguinte.

4.4. CONTROLADOR DE REFERÊNCIA ADAPTATIVO

O controlador de referência adaptativo, ou controlador de modelo de referência adaptativo, é outro tipo de controlo utilizado neste estudo. Este controlador caracteriza-se pelo uso de duas redes neuronais como já foi explicitado acima. Contudo, para este trabalho, este

controlador deve incorporar um PID. Assim, serão utilizados dois blocos neuronais e um controlador PID. A primeira rede neuronal mapeia a entrada-saída do processo e é treinada *a priori*. A segunda rede neuronal é treinada *online* e atualiza os ganhos do PID para que a resposta do processo seja semelhante à da referência.

Como é previsível o primeiro passo para a elaboração deste sistema de controlo, é o treino da modelo neuronal do motor. Tal como no desenvolvimento do controlador anterior foi criado um *script* para a configuração da rede e treino. A sequência de etapas é similar à efetuada para o treino da rede do controlador anterior. Contudo, como se quer mapear a entrada-saída (identificação do processo) e não a saída-entrada, existem algumas características de configuração e treino que se distinguem.

A primeira é a definição das entradas e saída, como sugere o código apresentado no Anexo B, Secção 1.1. Relativamente à saída desejada da rede, esta passa a ser a velocidade angular do processo, de modo a que a rede consiga reproduzir fielmente as respostas do motor. Já as entradas são definidas tendo em conta a influência das mesmas na saída. O número de entradas escolhidas influencia a alteração de outros dois parâmetros: estrutura da rede e esquema de treino.

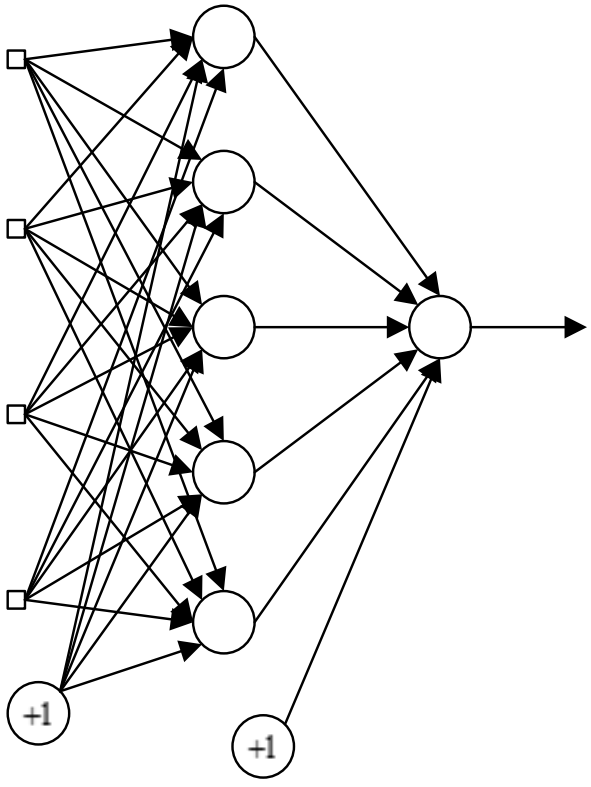


Figura 72 Diagrama do modelo neuronal do controlador de referência adaptativo

A estrutura da rede não tem uma organização premeditada, mas deve garantir que a rede apresente o melhor desempenho possível. O esquema da mesma é exibido na Figura 72. Contudo, a definição das entradas está interligada com a estrutura. O número das entradas da estrutura deve ser igual ao número de entradas definidas para o treino.

O esquema de treino é, à partida, diferente do utilizado para o controlador anterior. Este deve estar de acordo com o diagrama da Figura 73, sendo utilizada a saída do motor para treinar a rede neuronal. Além disto, é necessário ter em conta o número de entradas da rede. Como é constatado, o esquema de treino está em concordância com a estrutura da rede (e por sua vez com a definição das entradas de treino) pois apresenta uma rede com 4 entradas.

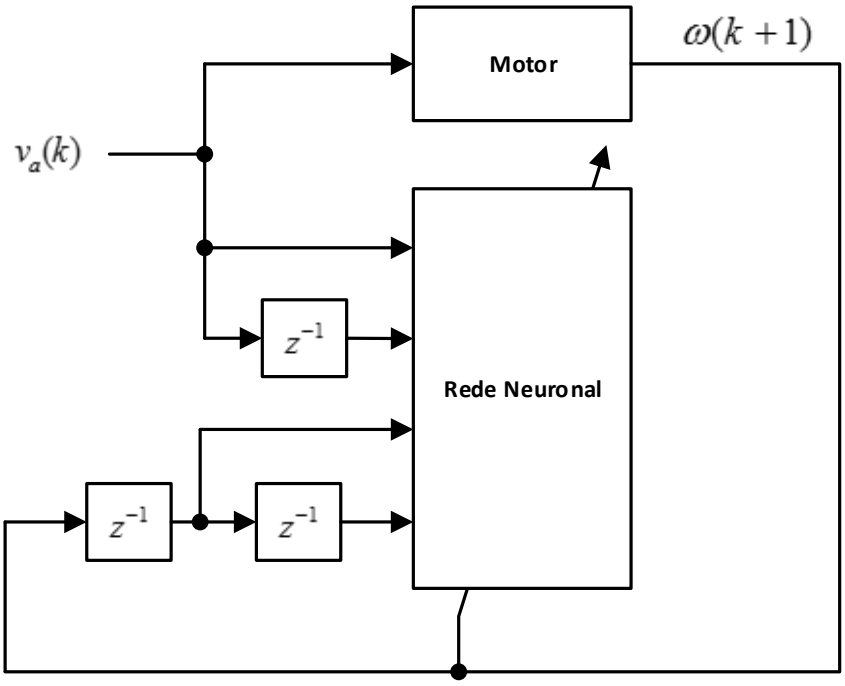


Figura 73 Diagrama de treino da rede neuronal (Identificação do processo)

Por último, são guardados os pesos das ligações dos neurónios para serem utilizados posteriormente. Este passo é necessário no desenvolvimento deste controlador pois a simulação não é concebida da mesma forma que no tipo de controlo anterior. Assim, depois de treinada e de ser feito um teste de generalização, os pesos são guardados em variáveis de acordo com o excerto de código do Anexo B, Secção 1.2.

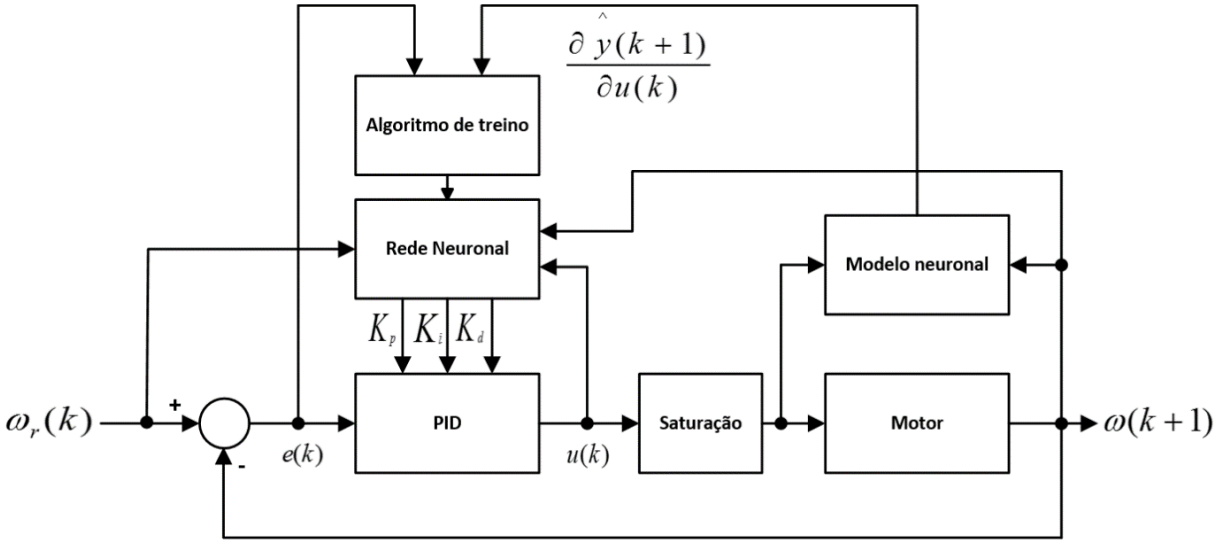


Figura 74 Diagrama de aplicação do controlador de referência adaptativo

A partir desta fase é possível desenvolver o sistema de simulação. O diagrama deste tipo de controlo encontra-se na Figura 74. Tal como já foi dito, este sistema utiliza dois blocos neuronais: um modelo neuronal que imita o processo, e outro de atualização de ganhos do PID. Além disso é aplicado um bloco de saturação à entrada do processo devido aos limites impostos pelo mesmo (ver secção 4.1).

O sistema começa por analisar o comportamento do processo na amostra anterior, e calcular o erro. Para que se compute o sinal de entrada correto na amostra presente, é necessário aplicar o algoritmo de treino de modo a corrigir os pesos da rede de atualização dos ganhos. Além dos pesos da rede neuronal e outras variáveis já calculadas que envolvem a mesma, são também necessárias neste procedimento variáveis externas. Como é possível observar no esquema do sistema, os dois parâmetros de entrada do algoritmo de treino são: o erro, e o Jacobiano, calculado pelo modelo neuronal. Mais à frente o algoritmo de treino será explicado em maior detalhe.

Depois de serem atualizados os pesos da rede, são também processados os novos ganhos do controlador PID. Torna-se assim possível calcular o sinal de controlo. Esse sinal será sujeito a um teste de saturação antes de ser aplicado ao processo. Por fim, o processo exibe a resposta correspondente ao sinal de controlo e calcula-se o Jacobiano para ser utilizado no treino da rede neuronal na amostra seguinte.

Nas subsecções seguintes é explicitado em maior detalhe os procedimentos envolvidos em cada bloco do diagrama.

4.4.1. CONTROLADOR PID

A expressão (48) representa controlador PID no domínio contínuo. Devido às características do sistema é necessário discretizar a equação do PID. A simulação do sistema a implementar é amostra-a-amostra, sendo assim necessário desenvolver uma expressão diferente capaz de produzir uma resposta discreta. Para isso, através da expressão (48), é formulada a seguinte equação no domínio de Laplace:

$$\frac{U(s)}{E(s)} = K_p + \frac{K_i}{s} + K_d s \tag{57}$$

De seguida aplica-se um dos métodos equivalentes discretos por integração numérica. Neste caso foi utilizada a regra dos retângulos atrás. Esta regra dita a seguinte aproximação:

$$s \to \frac{z-1}{Tz} \tag{58}$$

Assim,

$$\frac{U(z)}{E(z)} = K_p + K_i \frac{Tz}{z-1} + K_d \frac{z-1}{Tz}$$

$$\Leftrightarrow \frac{U(z)(z-1)}{E(z)} = K_p(z-1) + K_i Tz + K_d \frac{(z-1)^2}{Tz} \tag{59}$$

$$\Leftrightarrow U(z)(z-1) = K_p E(z)(z-1) + K_i E(z) Tz + K_d E(z) \frac{z^2 - 2z + 1}{Tz}$$

Para $T = 1\ s$, a equação amostrada a utilizar na simulação do sistema será igual a:

$$\begin{aligned} u(k+1) = u(k) + K_p(e(k+1) - e(k)) + K_i e(k+1) \\ + K_d(e(k+1) - 2e(k) + e(k-1)) \end{aligned} \tag{60}$$

4.4.2. REDE NEURONAL DE SINTONIA DOS GANHOS

A rede neuronal de sintonia dos ganhos do PID envolve dois blocos do diagrama da Figura 74: *rede neuronal* e *algoritmo de treino*. O primeiro representa o cálculo dos ganhos do controlador PID, processando as entradas da rede e calculando os valores de saída da rede neuronal. O segundo representa o algoritmo de treino da rede, e engloba todas as equações necessárias para a atualização dos pesos consoante o erro obtido.

A estrutura da rede neuronal é apresentada na Figura 75. A rede pode ainda ser definida pela equação matemática MIMO (*multiple-input and multiple-output*) (61), sendo f uma função não-linear e os seus argumentos as entradas da rede. O número de entradas da rede, os valores de entrada, e os neurónios da camada invisível foram escolhidos conforme o desempenho e os sinais a considerar para a respetiva saída.

$$\left[K_p(k), K_i(k), K_d(k)\right] = f[u(k-1), u(k-2), \omega(k), \omega(k-1), \omega_r(k), \omega_r(k-1)] \quad (61)$$

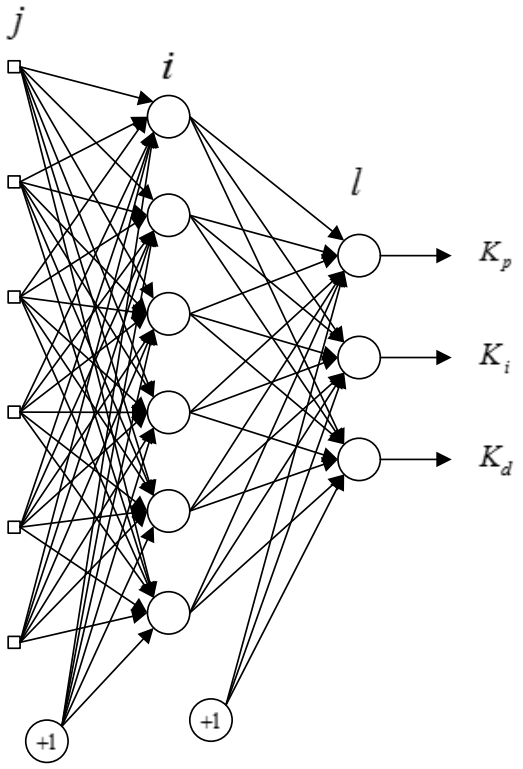


Figura 75 Diagrama da rede neuronal de sintonia dos ganhos do controlador PID

O bloco *rede neuronal* é apenas composto pelos cálculos de cada neurónio até aos valores finais de saída (ganhos). Os cálculos são processados por camada de neurónios.

Para a camada de entrada, os seus valores são iguais às entradas da rede já referidas acima, com o acréscimo do valor limiar:

$$x_0^{(1)} = u(k-1),$$

$$x_1^{(1)} = u(k-2),$$

$$x_2^{(1)} = \omega(k),$$

$$x_3^{(1)} = \omega(k-1), \tag{62}$$

$$x_4^{(1)} = \omega_r(k),$$

$$x_5^{(1)} = \omega_r(k-1),$$

$$x_6^{(1)} = 1.$$

Onde, por exemplo, $x_2^{(1)}$ é o segundo ponto de entrada da primeira camada, e $x_6^{(1)}$ é o valor limiar. Nas equações, os expoentes (1), (2) e (3), correspondem às camadas de entrada, invisível, e de saída, respetivamente.

A entrada e saída da camada invisível é expressa pelas seguintes equações:

$$net_i^{(2)} = \sum_{j=0}^{6} w_{ij}^{(2)}\, x_j^{(1)}(k)\,,$$

$$o_i^{(2)}(k) = f\left[net_i^{(2)}(k)\right], i = 0,1,\dots,6, \tag{63}$$

$$o_6^{(2)}(k) = 1.$$

Onde, $net_i^{(2)}$ e $o_i^{(2)}$ são a entrada e saída do neurónio i da camada invisível, respetivamente, $w_{ij}^{(2)}$ é o peso da ligação entre a camada de entrada e a invisível, e $f[x]$ é a função de ativação dada por $f[x] = \tanh(x)$. Esta função é também conhecida como logarítmica com intervalo de valores [-1;1]. O valor de saída $o_6^{(2)}$ equivale ao limiar para a camada de saída.

A entrada e saída da camada de saída é expressa pelas seguintes equações:

$$net_l^{(3)} = \sum_{i=0}^{6} w_{li}^{(3)}\, o_i^{(2)}(k)\,,$$

$$o_l^{(3)}(k) = g\left[net_l^{(3)}(k)\right], l = 0,1,2, \tag{64}$$

$$K_p(k) = o_0^{(3)}(k),$$

$$K_i(k) = o_1^{(3)}(k),$$

$$K_d(k) = o_2^{(3)}(k).$$

Onde, $w_{li}^{(3)}$ é o peso das ligações entre a camada invisível e de saída, e $g[x]$ é a função de ativação dada por $g[x] = [1 + \tanh(x)]/2$. Esta função é também conhecida como logarítmica com intervalo [0;1]. Na camada de saída não é possível usar a função linear, como nos outros casos, porque os ganhos devem ser positivos. No entanto, estes podem ter um intervalo maior do que a função logarítmica toma. Assim, são adicionadas constantes aos valores de saída da rede neuronal, de acordo com a equação (65), de modo a que os ganhos não estivessem restringidos a um intervalo tão pequeno.

$$\begin{aligned} K_p(k) &= K_0 o_0^{(3)}(k), \\ K_i(k) &= K_1 o_1^{(3)}(k), \\ K_d(k) &= K_2 o_2^{(3)}(k). \end{aligned} \tag{65}$$

Os cálculos dos ganhos apresentados acima são definidos pelo código exibido Anexo B, Secção 1.3. Inicialmente são configuradas as variáveis que servem de entrada para a rede neuronal de sintonia. Os ciclos *for* seguintes representam os cálculos de cada camada. O primeiro é referente à camada invisível, e o segundo à camada de saída.

Contudo, para que os valores dos ganhos calculados sejam os corretos e mais indicados, é necessário treinar a rede. Tal como já foi visto, o treino da rede é feito *online*. O algoritmo de treino utilizado é o da retropropagação do erro por ser o de uso mais fácil e o mais adotado. Este altera os pesos iteração a iteração de modo a que a rede neuronal conceba os ganhos do PID ideais para corrigir o erro entre a velocidade de referência e a do processo.

Antes de se aplicar o algoritmo é necessário escolher os valores iniciais dos pesos. Estes são determinantes para todo o processo de aprendizagem, pois influenciam em grande parte o tempo que o algoritmo demora a convergir para um valor ótimo. Definiu-se os valores dos pesos iniciais como aleatórios no intervalo [-0,5;0,5], os valores de entrada como 0,5, e os valores de saída dos neurónios (K_p, K_i, e K_d) também como 0,5. Estes valores são os mais aconselháveis de modo a melhorar a convergência da rede [55].

Relativamente ao algoritmo de retropropagação, tal como já foi visto, é baseado no método de gradiente descendente. O objetivo é a minimização de uma função de custo E que pode ser expressa por:

$$E = \frac{1}{2}[r(k+1) - y(k+1)]^2 = \frac{1}{2}e^2(k+1) \tag{66}$$

Esta equação é usada para modificar os pesos, através da procura e sintonia na direção do gradiente negativo e adicionando um coeficiente de inércia (*momentum*) para uma convergência mais rápida:

$$\Delta w_{li}^{(3)}(k+1) = -\eta \frac{\partial E}{\partial w_{li}^{(3)}} + \mu \Delta w_{li}^{(3)}(k) \tag{67}$$

$$\frac{\partial E}{\partial w_{li}^{(3)}} = \frac{\partial E}{\partial y(k+1)} \frac{\partial y(k+1)}{\partial u(k)} \frac{\partial u(k)}{\partial o_l^{(3)}(k)} \frac{\partial o_l^{(3)}(k)}{\partial net_l^{(3)}(k)} \frac{\partial net_l^{(3)}(k)}{\partial w_{li}^{(3)}} \tag{68}$$

Em (68), o Jacobiano do processo $\left(\frac{\partial y(k+1)}{\partial u(k)}\right)$ não é conhecido, mas pode ser substituído pelo seu valor previsto calculado através do modelo não-linear neuronal $\left(\frac{\partial \hat{y}(k+1)}{\partial u(k)}\right)$, como é abordado mais à frente.

Da equação (60), é possível derivar:

$$\frac{\partial u(k)}{\partial o_0^{(3)}(k)} = e(k) - e(k-1),$$

$$\frac{\partial u(k)}{\partial o_1^{(3)}(k)} = e(k), \tag{69}$$

$$\frac{\partial u(k)}{\partial o_2^{(3)}(k)} = e(k) - 2e(k-1) + e(k-2).$$

Assim, a equação de atualização dos pesos da camada de saída da rede neuronal é:

$$\Delta w_{li}^{(3)}(k+1) = \eta \delta_l^{(3)} o_i^{(2)} + \mu \Delta w_{li}^{(3)}(k), \tag{70}$$

$$\delta_l^{(3)} = e(k+1)\frac{\partial \hat{y}(k+1)}{\partial u(k)}\frac{\partial u(k)}{\partial o_l^{(3)}(k)}g'\left[net_l^{(3)}(k)\right],$$

$$l = 0,1,2.$$

A equação de atualização dos pesos da camada invisível é:

$$\Delta w_{ij}^{(2)}(k+1) = \eta\delta_i^{(2)}x_j^{(1)} + \mu\Delta w_{ij}^{(2)}(k),$$

$$\delta_i^{(2)} = f'\left[net_i^{(2)}(k)\right]\sum_l^2 \delta_l^{(3)}w_{li}^{(3)}(k), \tag{71}$$

$$i = 0,1,\ldots,6.$$

Onde,

$$g'[x] = g(x)[1-g(x)],$$
$$f'[x] = [1-f^2(x)]/2. \tag{72}$$

O método de treino (retropropagação) representado pelas equações anteriores é implementado através do Anexo B, Secção 1.4. Este é executado por uma função de treino da rede, utilizando as variáveis globais do erro do processo, Jacobiano, ganhos do PID, e outras variáveis relativas à rede, como taxa de aprendizagem (`Lrt`), *momentum* (`Mom`), entre outras.

4.4.3. MODELO NEURONAL

O bloco *Modelo Neuronal* corresponde a uma função que contêm os cálculos do modelo neuronal do processo. A rede neuronal processa as entradas e calcula o valor previsto da saída do processo. Para este caso este cálculo torna-se redundante, pois a rede já se encontra treinada, ou seja, os pesos das ligações da estrutura da rede (Figura 72) já se encontram otimizados, tornando-a capaz de emular o processo. Assim, se o modelo neuronal estiver bem implementado é espectável que o erro entre o modelo e o processo seja próximo de 0 (zero).

Considerando as entradas do modelo neuronal utilizadas para o treino e as especificidades do sistema, pode-se expressar esta rede neuronal pela seguinte expressão matemática:

$$[\hat{y}(k+1), J(k+1)] = f[\omega(k), \omega(k-1), u(k), u(k-1)] \tag{73}$$

Sendo os argumentos da função as suas entradas, e as saídas o valor da resposta do processo previsto e o Jacobiano (abordado mais à frente).

A entrada e saída da camada invisível é expressa pelas seguintes equações:

$$net_i^{(2)} = \sum_{j=0}^{4} w_{ij}^{(2)}\, x_j^{(1)}(k)\,,$$

$$x_4^{(1)}(k) = 1,$$

$$o_i^{(2)}(k) = f\left[net_i^{(2)}(k)\right], i = 0,1,\ldots,5, \tag{74}$$

$$o_5^{(2)}(k) = 1.$$

Onde, i e $o_i^{(2)}$é o número dos neurónios e do limiar da camada invisível e o valor da sua saída, respetivamente, j e $x_j^{(1)}$ é o numero das entradas e do limiar da primeira camada e o valor da entrada, respetivamente, $w_{ij}^{(2)}$ é o peso da ligação entre a camada de entrada e a invisível, e $f[x]$ é a função de ativação dada por $f[x] = \tanh(x)$.

A entrada e saída da camada de saída é expressa pela seguinte equação:

$$\hat{y}(k+1) = \sum_{i=0}^{6} w_i^{(3)}\, o_i^{(2)}(k) \tag{75}$$

Onde, $w_i^{(3)}$ é o peso das ligações entre a camada invisível e de saída, e $\hat{y}$ é o valor previsto da resposta do processo. A função de ativação usada no neurónio de saída é linear, ou seja, não é aplicada nenhuma restrição ao somatório, e o valor resultante do mesmo é a saída da rede.

Contudo, a razão deste bloco estar implementado no sistema, prende-se principalmente com o cálculo do Jacobiano (38). O Jacobiano é a derivada da saída em relação à entrada, e como

tal, não é possível ser calculado através do processo. No entanto, é possível fazer a seguinte aproximação:

$$\frac{\partial y(k+1)}{\partial u(k)} \approx \frac{\partial \hat{y}(k+1)}{\partial u(k)} \tag{76}$$

A expressão anterior aproxima o Jacobiano calculado através da saída real do processo com o calculado através da saída prevista, ou seja, a saída do modelo neuronal. A equação seguinte expõe o procedimento da aquisição do Jacobiano:

$$\frac{\partial \hat{y}(k+1)}{\partial u(k)} = \sum_{i=0}^{5}\left(\frac{\partial \hat{y}(k+1)}{\partial o_i^{(2)}(k)}\frac{\partial o_i^{(2)}(k)}{\partial net_i^{(2)}(k)}\frac{\partial net_i^{(2)}(k)}{\partial u(k)}\right) = \sum_{i=0}^{5}\left(w_i^{(3)} f'\left[net_i^{(2)}(k)\right] w_{i3}^{(2)}\right) \tag{77}$$

Onde, i é o número dos neurónios da camada invisível, $f'[x]$ é a derivava da função de ativação dada por $f'[x] = [1 - f^2(x)]/2$, e $w_{i3}^{(2)}$ é o peso da ligação à entrada correspondente ao sinal de controlo da presente amostra.

Os cálculos do modelo neuronal são implementados através do código Anexo B, Secção 1.5. Primeiramente a função define as variáveis de entrada da rede e depois realiza os cálculos camada-a-camada de modo a encontrar o valor da velocidade previsto. Por fim, é computado e retornado o Jacobiano do processo para a amostra *k*.

4.5. ÍNDICES DE DESEMPENHO

Depois da simulação dos sistemas e de efetuar diferentes testes de desempenho, é necessário analisar os resultados obtidos. A exposição das respostas por meio de gráfico torna-se útil para uma primeira análise. Essa mesma análise, embora visual, permite perceber as principais diferenças entre as respostas.

Contudo, as respostas obtidas podem ter diferenças mínimas não sendo percetíveis visualmente. Assim, é requerida uma análise mais precisa e cuidada, que trate os sinais e garanta que mesmo os erros mais pequenos sejam contabilizados, podendo retirar conclusões sobre qual o sistema mais eficaz.

Para análise dos sistemas implementados são aplicados índices de desempenho que medem o erro de diferentes formas: erro médio quadrático (MSE), erro médio absoluto (MAE), integral do erro quadrático (ISE), e integral do erro absoluto (IAE).

O MSE, tal como o nome indica, calcula a média de todos os erros quadráticos. É uma medida frequentemente usada na verificação de precisão de modelos numéricos. Este método é bastante sensível a grandes erros pois eleva as diferenças individuais ao quadrado. O seu cálculo é definido pela seguinte expressão:

$$MSE = \frac{1}{N}\sum_{i=0}^{N}(r_i - y_i)^2 \tag{78}$$

Onde, i e N são o número da amostra e o número total de amostras da simulação efetuada, respetivamente, r é a velocidade de referência à entrada do sistema, e y é a velocidade real do processo.

Este índice é calculado de duas maneiras diferentes. Se a simulação for realizada por *Simulink* é apenas possível calcular a componente da soma com os dados da simulação. O código deste método é apresentado abaixo.

```
valor_soma=0;
i=1;
while (<variável_resposta>(i,1)<N)
    valor_soma=valor_soma+(<variável_resposta>(i,2) -
<variável_resposta>(i,3))^2;
    i=i+1;
end
<variável_índice>=valor_soma/N;
fprintf('Índice <índice>: %d \n',<variável_índice>);
```

A segunda forma é calcular a componente da soma durante as iterações da simulação. Este é o método utilizado para a simulação do controlador de referência adaptativo. O ciclo *while* desaparece pois a soma já é realizada no ciclo de simulação, e no fim deste processo o valor resultante é dividido pelo número de iterações.

O MAE, tal como o MSE, mede quão perto se encontram os valores de referência dos valores de resposta do processo. Este não leva em conta o erro sobrestimado ou subestimado, caracterizando-se por ser a média dos erros cometidos pelo modelo de previsão durante uma série de períodos de tempo. O seu cálculo é definido pela seguinte expressão:

$$MAE = \frac{1}{N}\sum_{i=0}^{N}|r_i - y_i| \tag{79}$$

O cálculo deste índice é bastante parecido com o do MSE e o mesmo se pode afirmar para a sua implementação. O código e os métodos de cálculo são exatamente iguais, com a exceção de ser calculado o valor absoluto do erro e não o valor quadrático.

As duas medidas seguintes são bastantes utilizadas em publicações e estudos de simulações. Estas requerem uma experiência a ser realizada num sistema, como por exemplo, um sinal de referência, e é posteriormente avaliado o integral do erro ao longo de um período de tempo. No entanto, é necessário que o erro seja capaz de se aproximar bastante de 0 (zero), pois se tal não acontecer estas medidas terão valores muito elevados devido às características do integral.

A primeira medida é o ISE (*integral squared error*), e efetua o integral do error quadrático. Este índice penaliza em grande parte os maiores erros em detrimento dos menores. O ISE é definido pela seguinte expressão:

$$ISE = \int_0^t (r_i - y_i)^2 \tag{80}$$

A segunda medida é o IAE (*integral squared error*) e efetua o integral do erro absoluto. Ao contrário do ISE, este não adiciona peso aos erros do sistema. O IAE é definido pela seguinte expressão:

$$IAE = \int_0^t |r_i - y_i| \tag{81}$$

O cálculo destes dois últimos índices, que utilizam o integral do erro, é realizado através do comando `trapz`. Este utiliza a regra trapezoidal para calcular a área da função de erro. Para a sua implementação deve ser definido a variável `X` que é nada mais que um vetor dos elementos da função que se pretende integrar. O código abaixo demonstra como foi aplicado o índice ISE, sendo que para o IAE apenas é necessário alterar o quadrado do erro para o seu valor absoluto, tal como nas expressões (80) e (81).

```
ISE = trapz(X,(r(X+2)-w_sim(X+3)).^2);
```

5. TESTES E RESULTADOS

Neste capítulo são apresentados os resultados dos testes realizados em cada sistema desenvolvido abordados no capítulo anterior, sendo feita uma análise de desempenho desses mesmos sistemas.

O derradeiro objetivo destes testes é provar que o uso das redes neuronais é capaz de melhorar a resposta do processo em comparação com a obtida com um controlador PID isolado. Dois modelos são apresentados para cada sistema: um com ruído e outro sem ruído. Isto serve também para examinar a capacidade dos controladores para reagirem a distúrbios externos. Além destes modelos são efetuados alguns testes de desempenho para melhorar a capacidade de controlo da rede, nomeadamente no controlador de referência adaptativo onde a rede é treinada *online.*

5.1. CONTROLADOR ESTABILIZADOR

Os testes a realizar envolvem a comparação de respostas obtidas com ou sem rede neuronal, isto é, são simulados dois sistemas equivalentes e calculados os seus desempenhos. O primeiro é um simples sistema de realimentação que usa um controlador PID isolado. O PID, depois de sintonizado, minimiza o erro do sistema de modo que a saída do processo siga uma entrada de referência. O segundo é o controlador estabilizador. A diferença para o primeiro

sistema é a adição da rede neuronal. Esta adicionará ao sinal de controlo de realimentação, um sinal de controlo direto que não tomará em conta o erro do sistema.

Os testes de desempenho têm por base pequenas alterações. São realizadas simulações com e sem ruído, utilizando três métodos de sintonia do PID (Ziegler-Nichols, Coon-Cohen, Zhuang-Atherton), e diferentes entradas. Assim, é possível fazer análises em diferentes condições e tirar uma conclusão mais geral.

Na simulação do controlador estabilizador, a rede neuronal já se encontra treinada, emulando fielmente o modelo inverso do processo. Assim, as propriedades da rede não são alteradas pois já se encontram otimizadas, não sendo necessário realizar testes sobre a influência das modificações das variáveis da rede.

Nas próximas subsecções são apresentados os resultados de cada um dos testes realizados, sendo estes divididos em *sem ruído* e *com ruído*. Todas as simulações são realizadas utilizando a ferramenta *Simulink* do MATLAB. Os esquemas utilizados para a simulação encontram-se no Anexo A.

5.1.1. SEM RUÍDO

A primeira simulação a realizar foi a dos sistemas sem ruído. Esta é dividida em 4 partes, numa combinação entre os três métodos de sintonia aplicados, e de duas entradas diferentes: onda sinusoidal (Anexo A, Secção 3) e quadrada (Anexo A, Secção 4).

Aplicando os métodos de sintonia para o controlador PID (Ziegler-Nichols, Coon-Cohen, e Zhuang-Atherton), foram obtidos os ganhos exibidos na Tabela 9. Para a sintonia ótima de Zhuang-Atherton foi utilizado o critério ISTE.

Tabela 9 Valores dos ganhos para os métodos de sintonia utilizados

	Ziegler-Nichols	Coon-Cohen	Zhuang-Atherton
K_p	0,7489	2,4422	1,7862
K_i	0,8038	5,0110	9,4559
K_d	0,1745	0,1402	0,2578

Estes ganhos são aplicados ao controlador PID, e obtidos os gráficos apresentados na Figura 76 e Figura 77 para as duas entradas.

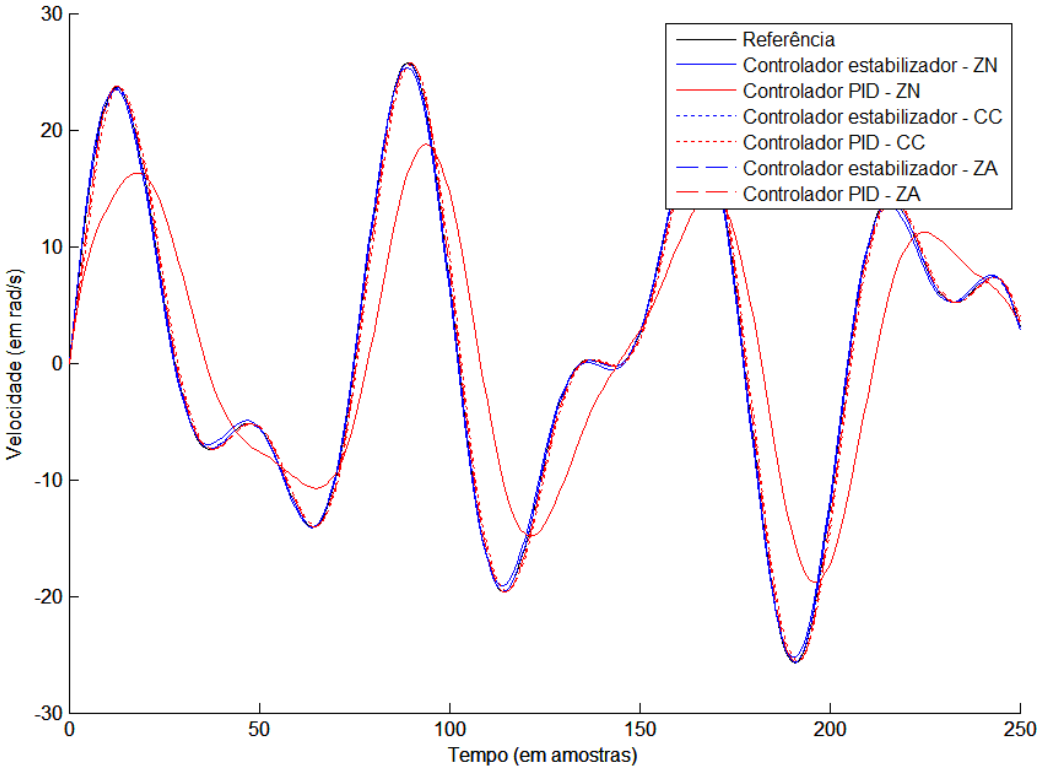


Figura 76 Comparação das respostas para o controlador estabilizador com uma entrada sinusoidal

Numa primeira análise pode-se observar que para a entrada sinusoidal as respostas do controlador estabilizador para as várias sintonias seguem quase na perfeição a referência. Já o controlador PID apresenta uma resposta bastante má para a sintonia de Ziegler-Nichols. Esta tem um grande atraso e não consegue ter a amplitude da entrada de referência.

Relativamente às outras sintonias, as suas respostas são bem melhores. No entanto, estas parecem ter sempre um desempenho ligeiramente inferior em comparação com as respostas equivalentes do controlador estabilizador, ou seja, com as respostas que utilizam a mesma sintonia.

Para a entrada quadrada, verifica-se que a respostas do controlador estabilizador são quase todas melhores que as respostas do PID. A exceção é a sintonia de Zhaung-Atherton do controlador estabilizador, que parece exibir um desempenho inferior relativamente à resposta do controlador PID sintonizado com o método de Coon-Cohen. Entre as diferentes sintonias, as melhores são a Ziegler-Nichols e a Coon-Cohen, sendo que esta última apresenta uma maior oscilação mas um menor tempo de estabelecimento que a primeira.

Em relação às respostas do controlador PID, destaca-se o grande *undershoot* em cada "passo" da onda, possuindo, contudo, um *overshoot* similar ao do controlador estabilizador. Para a sintonia de Ziegler-Nichols, o sistema de realimentação atinge a estabilidade, mas apresenta um valor do erro final bastante superior a todas as outras respostas do controlador PID.

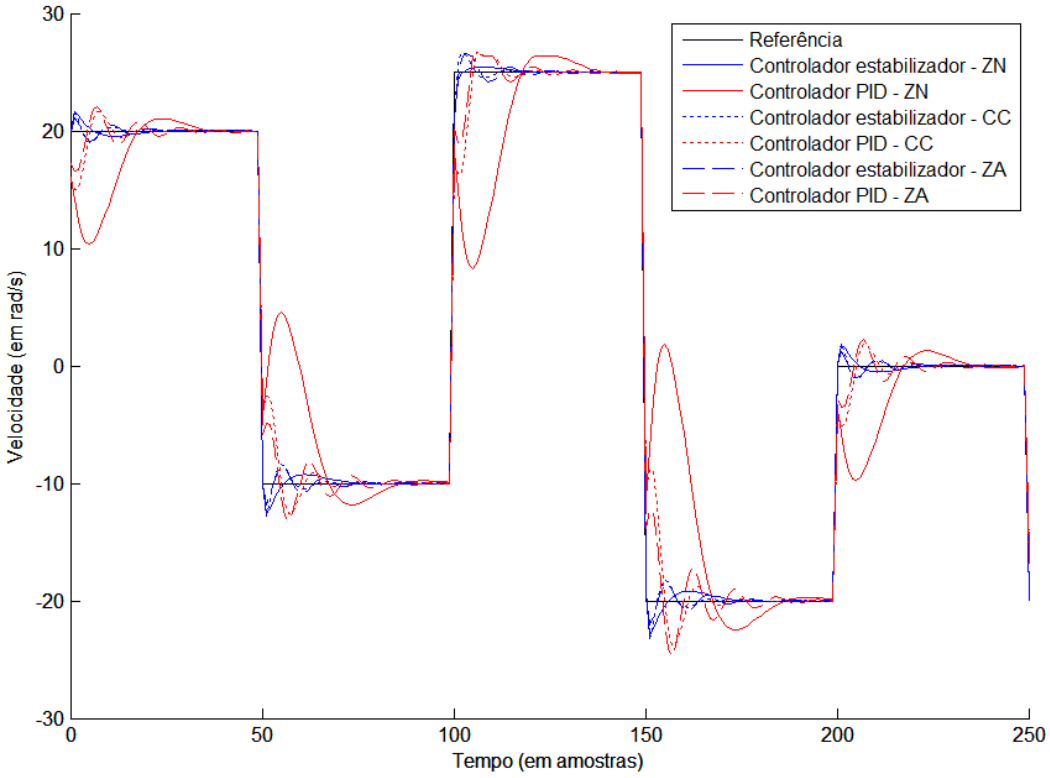


Figura 77 Comparação das respostas para o controlador estabilizador com uma entrada quadrada

Na generalidade das respostas sem a adição de ruído, apenas pela análise visual dos gráficos, é possível verificar que o sistema do controlador estabilizador têm um melhor desempenho que o sistema de realimentação do controlador PID. As maiores diferenças são verificadas para o sistema que utiliza o método de sintonia de Ziegler-Nichols.

5.1.2. COM RUÍDO

De seguida foi adicionado ruído ao sensor de leitura da velocidade de saída do processo. Este é gerado por um bloco no S*imulink* que gera números aleatórios distribuídos e que é introduzido no sistema como ruído branco. O ruído branco é um sinal aleatório com uma densidade espectral de potência (PSD) constante ao longo do espectro de frequências. O sinal caracteriza-se por ter uma média de valores igual a zero e uma variância finita. A PSD,

define como a variância se encontra distribuída num domínio de frequências. No bloco utilizado, definiu-se o tamanho da PSD, ou potência de ruído, como 0,1.

Depois de definido e implementado devidamente o bloco que gera o ruído, são simulados os dois sistemas. Tal como nas simulações anteriores, são realizados testes com duas entradas diferentes, utilizando os esquemas do Anexo A, Secção 5 e Secção 6, e com os três métodos de sintonia. Como o processo não se altera, também os valores dos ganhos calculados anteriormente para as três formas de sintonia permanecem os mesmos.

Analisando os gráficos correspondentes aos testes realizados aos sistemas para as duas entradas utilizadas (Figura 78 e Figura 79), é possível afirmar que ambos os controladores possuem respostas idênticas às simulações sem ruído. Contudo, o ruído é percetível para ambas as entradas, não sendo completamente anulado por parte dos controladores abordados. Para a onda sinusoidal este não é constante ao longo do tempo de simulação, enquanto para a onda quadrada a sua presença é mais marcada.

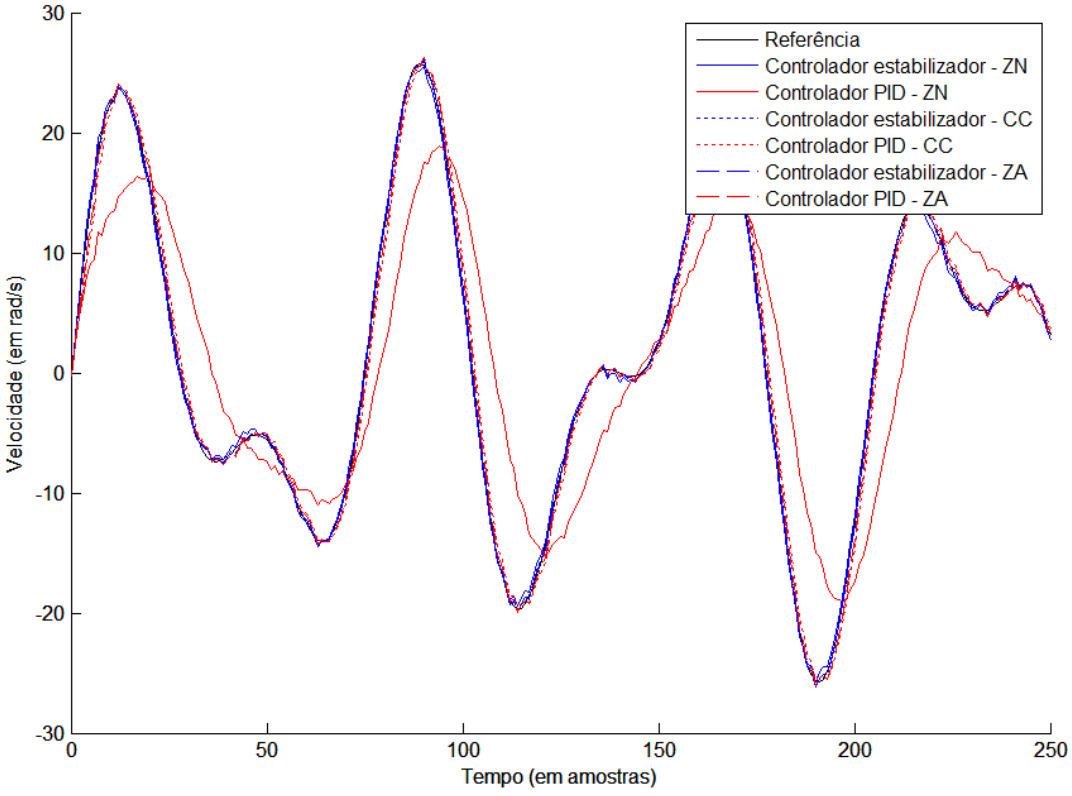


Figura 78 Comparação das respostas para o controlador estabilizador com uma entrada sinusoidal e com ruído

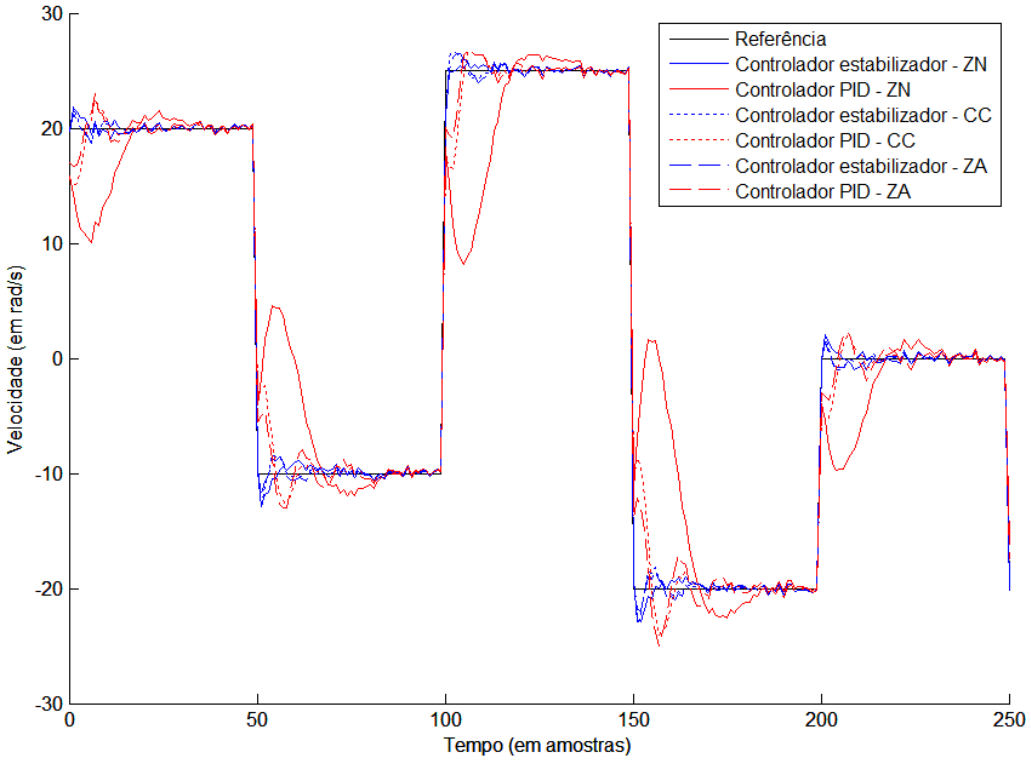


Figura 79 Comparação das respostas para o controlador estabilizador com uma entrada quadrada e com ruído

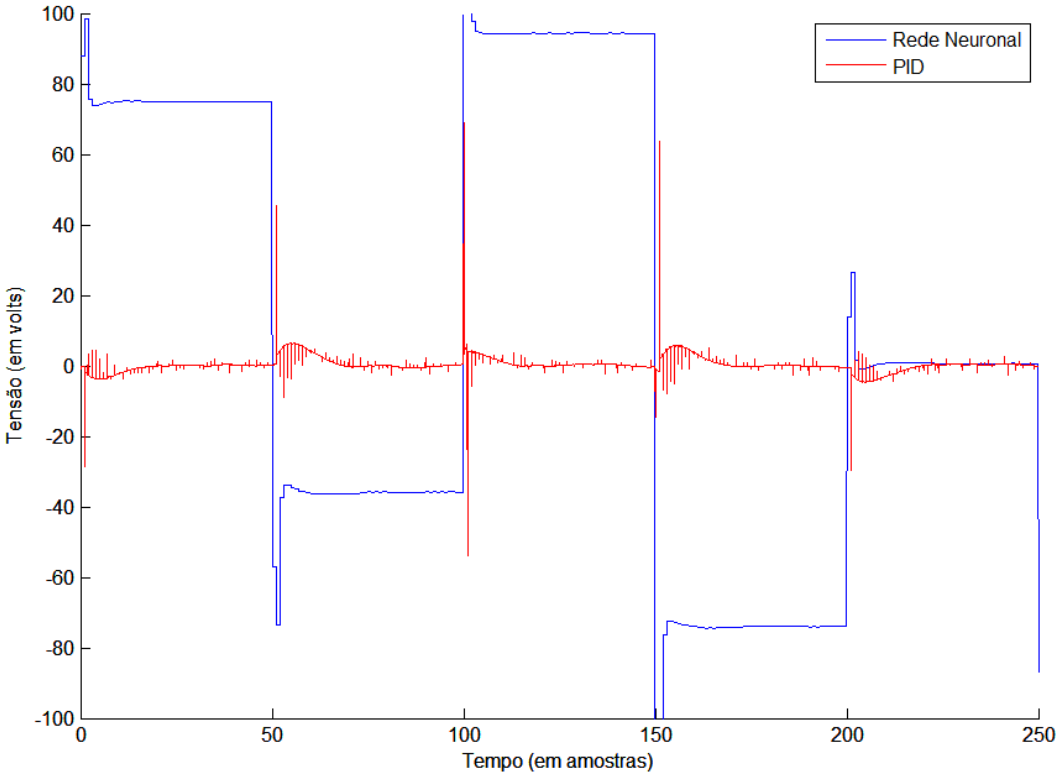


Figura 80 Comparação entre os sinais de controlo do controlador estabilizador com uma entrada quadrada e ruído

Na Figura 80 é confirmada a teoria do comportamento do sistema do controlador estabilizador. Tal como era previsto, a rede neuronal é responsável pelo controlo da velocidade do processo, sendo o controlador PID relegado para segundo plano. No entanto, este fica responsável por monitorizar e cancelar o erro, como acontece nos intervalos de tempo onda a velocidade de referência é estável e o erro é mínimo.

5.1.3. ÍNDICES DE DESEMPENHO

Para as simulações realizadas acima, são calculados os índices de desempenho correspondentes a essas respostas. Os índices utilizados foram: o MSE, MAE, ISE, e IAE. Os valores dos mesmos encontram-se na Tabela 10.

Tabela 10 Valores dos índices de desempenho do controlador estabilizador

			MSE	MAE	ISE	IAE
Ziegler-Nichols	S/ Ruído	Sinusoidal	0,192	0,374	48,151	93,686
		Quadrada	0,301	0,253	75,606	63,620
	C/ Ruído	Sinusoidal	0,250	0,415	62,750	103,887
		Quadrada	0,376	0,377	94,393	94,552
Coon-Cohen	S/ Ruído	Sinusoidal	0,010	0,078	2,377	19,418
		Quadrada	0,327	0,224	82,175	56,197
	C/ Ruído	Sinusoidal	0,071	0,212	17,654	53,007
		Quadrada	0,391	0,365	97,962	91,418
Zhuang-Atherton	S/ Ruído	Sinusoidal	0,004	0,045	0,975	11,296
		Quadrada	0,255	0,260	64,095	65,151
	C/ Ruído	Sinusoidal	0,077	0,221	19,310	55,265
		Quadrada	0,315	0,378	79,087	94,492

Dos valores apresentados na tabela é possível retirar as seguintes conclusões, sendo algumas delas já percetíveis na análise dos gráficos:

- O controlador PID encontra-se melhor sintonizado com a heurística de Zhuang-Atherton. O método de Coon-Cohen também apresenta valores de desempenho bastante bons, especialmente para os índices que consideram o erro absoluto, onde até existem valores melhores que os da sintonia Zhuang-Atherton. Contudo, os erros quadráticos são preponderantes, demonstrando que esta última sintonia é superior.

- Em relação aos sistemas com ruído, os desempenhos têm valores semelhantes para as três sintonias. Tal como já foi visto e é agora confirmado, a presença do ruído reflete-se nos valores dos índices através de uma pequena diminuição no desempenho. Independente da sintonia, as diferenças entre os valores do índices para os sistemas (com e sem ruído) com a mesma entrada são quase idênticas, não se destacando uma sintonia que seja capaz de lidar melhor com o ruído.

Depois terem sido retiradas as devidas conclusões entre os diferentes sistemas que envolvem o controlador estabilizador, são expostos os valores dos índices de desempenho da resposta do sistema que contêm apenas o controlador PID em realimentação (Tabela 11). Esses valores serão utilizadas para comparação com os já calculados, de modo a compreender se o controlador estabilizador é capaz de obter uma melhor resposta, e se isso se verifica também na presença de ruído.

Tabela 11 Valores dos índices de desempenho do controlador PID

			MSE	MAE	ISE	IAE
Ziegler-Nichols	S/ Ruído	Sinusoidal	47,376	5,528	$1{,}189 * 10^4$	$1{,}388 * 10^3$
		Quadrada	37,904	3,483	$9{,}545 * 10^3$	872,244
	C/ Ruído	Sinusoidal	47,722	5,554	$1{,}198 * 10^4$	$1{,}393 * 10^3$
		Quadrada	37,904	3,509	$9{,}505 * 10^3$	878,684
Coon-Cohen	S/ Ruído	Sinusoidal	1,284	0,904	321,976	226,475
		Quadrada	4,520	0,857	$1{,}126 * 10^3$	213,078
	C/ Ruído	Sinusoidal	1,390	0,944	348,717	236,546
		Quadrada	4,568	0,966	$1{,}137 * 10^3$	240,105
Zhuang-Atherton	S/ Ruído	Sinusoidal	0,373	0,482	93,578	120,846
		Quadrada	2,563	0,855	639,458	213,142
	C/ Ruído	Sinusoidal	0,470	0,546	118,039	136,835
		Quadrada	2,655	0,944	661,856	235,339

Para o controlador PID, os valores dos índices apresentados mostram a mesma tendência que o controlador estabilizador, tanto para sistemas sem ruído como com ruído.

Neste trabalho, contudo, o aspeto de maior relevância é a comparação de desempenhos entre os controladores. Uma aspeto que evidente, é a grande diferença de valores entre os sistemas que utilizam a sintonia Ziegler-Nichols. Todos os índices calculados demonstram que para essa sintonia o controlador estabilizador tem muito melhores resultados que o controlador

PID. Para a sintonia de Coon-Cohen, na análise dos gráficos não era percetível distinguir quem apresentava um desempenho melhor. Depois de calculados os índices, é claro que o controlador estabilizador é superior, dominando em todos os índices calculados, sendo as diferenças de muito menor amplitude do que as verificadas para a sintonia Ziegler-Nichols. Para a última das sintonias, Zhuang-Atherton, é seguida a mesma tendência onde o controlador estabilizador se mostra bastante superior ao PID em termos de desempenho.

5.2. CONTROLADOR DE REFERÊNCIA ADAPTATIVO

Diferente do controlador estabilizador, o sistema do controlador de referência adaptativo tem uma rede neuronal que é atualizada *online*. Assim, o processo de aprendizagem, que inclui um grande número de variáveis, pode influenciar em grande parte as respostas obtidas dependendo da definição das suas características, tais como: taxa de aprendizagem, *momentum*, e número de neurónios são algumas dessas variáveis. Contudo, a mais importante é o valor de inicialização dos pesos. Os melhores valores de inicialização permitem que o algoritmo de treino convirga mais rapidamente para o mínimo da função objetivo. Piores valores de inicialização não garantem que o algoritmo atinja mínimo global da função, podendo o gradiente ficar "preso" num mínimo local ou saturar.

Para combater esse problema, são realizados um determinado número de simulações até encontrar os valores dos pesos iniciais que permitam obter os melhores índices de desempenhos possíveis. Contudo, é possível estabelecer algumas propriedades dos pesos para melhorar a convergência sem que seja necessário utilizar métodos de cálculo para maximizar o desempenho do sistema pois o cálculo dos mesmos antecipadamente iria contra o objetivo do treino do algoritmo de retropropagação.

É importante que os pesos iniciais aleatórios tenham valores pequenos. Se o seu valor for próximo de 0 (zero), o limiar não terá grande influência antes da rede ser treinada. Contudo, estes não podem ter o mesmo valor pois terão o mesmo gradiente durante o treino e o ajuste será idêntico. Assim, as suas variações deverão ser pequenas, garantindo que os pesos aumentem a cada iteração e ajudando a rede a convergir mais rapidamente. Depois de consideradas todas estas premissas, foi definido que os pesos iniciais têm um valor aleatório no intervalo $[-0{,}5; 0{,}5]$.

5.2.1. CONFIGURAÇÃO DA REDE NEURONAL DE SINTONIA

O objetivo do estudo foca-se, no entanto, também nas outras variáveis. É verificado se estas poderão influenciar de alguma forma o desempenho do sistema. Essa é a primeira fase de comparação de desempenhos. Nela, variáveis como valor da taxa de aprendizagem, *momentum*, o número de neurónios da camada invisível, e o intervalo que os ganhos podem tomar, estão sobre maior atenção. Os testes do desempenho são utilizados para descobrir quais os melhores valores dessas mesmas variáveis.

Tal como já foi visto, a taxa de aprendizagem deve ter um valor baixo de modo que a rede neuronal não entre em oscilação, daí ser estabelecido que esta tomará valores no intervalo $[0{,}1; 0{,}3]$. O *momemtum* terá valores no intervalo $[0{,}7; 0{,}9]$. Por fim, sem ser previsto que esta última variável tenha uma grande influência sobre o resultado final, são apenas testadas redes com 5 (cinco) e 6 (seis) neurónios na camada invisível.

Numa primeira fase, o intervalo que os ganhos do PID podem tomar é $[0; 1]$. Estes não se alteram durante os testes para encontrar a melhor combinação de valores da taxa de aprendizagem, *momentum*, e número de neurónios.

Tabela 12 Valores do índice MSE para uma combinação de diferentes valores da taxa de aprendizagem, *momentum*, e número de neurónios da camada invisível

		Entrada Quadrada		Entrada Sinusoidal	
Taxa de aprendizagem	Momentum	5 Neurónios	6 Neurónios	5 Neurónios	6 Neurónios
0,1	0,7	40,814	32,127	23,267	23,260
	0,8	30,646	38,623	23,248	23,306
	0,9	39,197	52,527	23,237	23,273
0,2	0,7	39,197	61,152	23,237	23,238
	0,8	42,824	41,734	23,233	23,231
	0,9	34,674	34,198	23,197	23,223
0,3	0,7	36,343	45,505	23,250	23,241
	0,8	24,330	36,808	23,207	23,257
	0,9	35,010	38,290	23,224	23,232

Na Tabela 12 são apresentados os valores do índice MSE das simulações realizadas envolvendo diferentes valores das variáveis já referidas. O resultado apresentado é o melhor

de 50 simulações efetuadas. Nelas foi necessário usar um grande número de amostras pois a rede neuronal de sintonia não se encontra treinada. Assim, se o tempo de simulação não for suficientemente grande para que o gradiente esteja próximo do mínimo global, a resposta do processo não será capaz de seguir a referência e os valores do índice calculado serão enormes.

Olhando para a tabela, é visível que tanto a entrada sinusoidal como a entrada quadrada apresentam valores menores para a rede com 5 neurónios na camada invisível. Ainda que para a última a diferença não seja tão significativa. Relativamente às outras variáveis, os melhores valores do índice encontram-se para a taxa de aprendizagem mais alta, ou seja, 0,3, e nesse caso o valor de *momentum* indicado parece ser igual a 0,8.

Não sendo este o método mais preciso, é uma forma simples e rápida para encontrar os valores mais indicados para este sistema. Estes são utilizados para as fases de testes seguintes, nomeadamente para a definição dos intervalos que os ganhos devem tomar, e a medição do desempenho do sistema adicionando ruído.

Tabela 13 Valores do índice MSE para uma combinação de diferentes intervalos dos ganhos do controlador PID

		Entrada Quadrada			Entrada Sinusoidal		
K_p	K_i	K_d			K_d		
		$]0;1]$	$]0;3]$	$]0;5]$	$]0;1]$	$]0;3]$	$]0;5]$
$]0;1]$	$]0;1]$	50,449	47,015	88,560	23,213	23,256	23,731
	$]0;3]$	30,890	60,961	65,062	3,387	3,399	3,523
	$]0;5]$	21,806	75,710	91,148	1,285	1,308	1,282
$]0;3]$	$]0;1]$	42,254	89,387	88,559	18,165	18,444	18,638
	$]0;3]$	18,139	52,119	172,991	3,230	3,321	3,580
	$]0;5]$	28,013	92,801	82,673	1,293	1,305	1,431
$]0;5]$	$]0;1]$	44,125	95,112	90,878	14,755	14,488	14,939
	$]0;3]$	46,949	93,882	189,820	3,212	3,558	3,481
	$]0;5]$	58,828	119,277	187,735	2,011	51,740	9,504

O intervalo dos ganhos é definido da mesma forma que as variáveis de taxa de aprendizagem e *momentum*. Estes podem tomar os seguintes valores: $]0;1]$, $]0;3]$, e $]0;5]$. Os intervalos foram escolhidos tendo em conta os valores obtidos pelos métodos de sintonia do controlador anterior, e de modo a que a rede possa convergir mais rápido para os valores de saída ideais.

Na Tabela 13 são apresentados os valores calculados do índice MSE para as simulações com os respetivos intervalos dos ganhos.

Analisando as tabelas apresentadas, decide-se que intervalos dos ganhos a usar. O intervalo de mais clara escolha talvez seja a do ganho K_i. Para a entrada sinusoidal, destacam-se dois deles, e não sendo grande a diferença, optou-se pelo mais regular: $]0; 5]$. Relativamente ao ganho K_d, observa-se que os piores resultados surgem nos maiores intervalos. Assim, escolhe-se $]0; 1]$ para esse mesmo ganho. Por fim, para o ganho K_p, opta-se pelo intervalo $]0; 3]$, embora os valores sejam bastante idênticos.

5.2.2. SEM RUÍDO

A última fase de testes deste controlador é simulação e cálculo dos índices de desempenho. O sistema é simulado com os valores escolhidos nas fases anteriores. Contudo, este não se realiza da mesma forma. O objetivo é encontrar o melhor resultado possível. Assim sendo, o número de simulações será o dobro do efetuado até aqui (100 simulações).

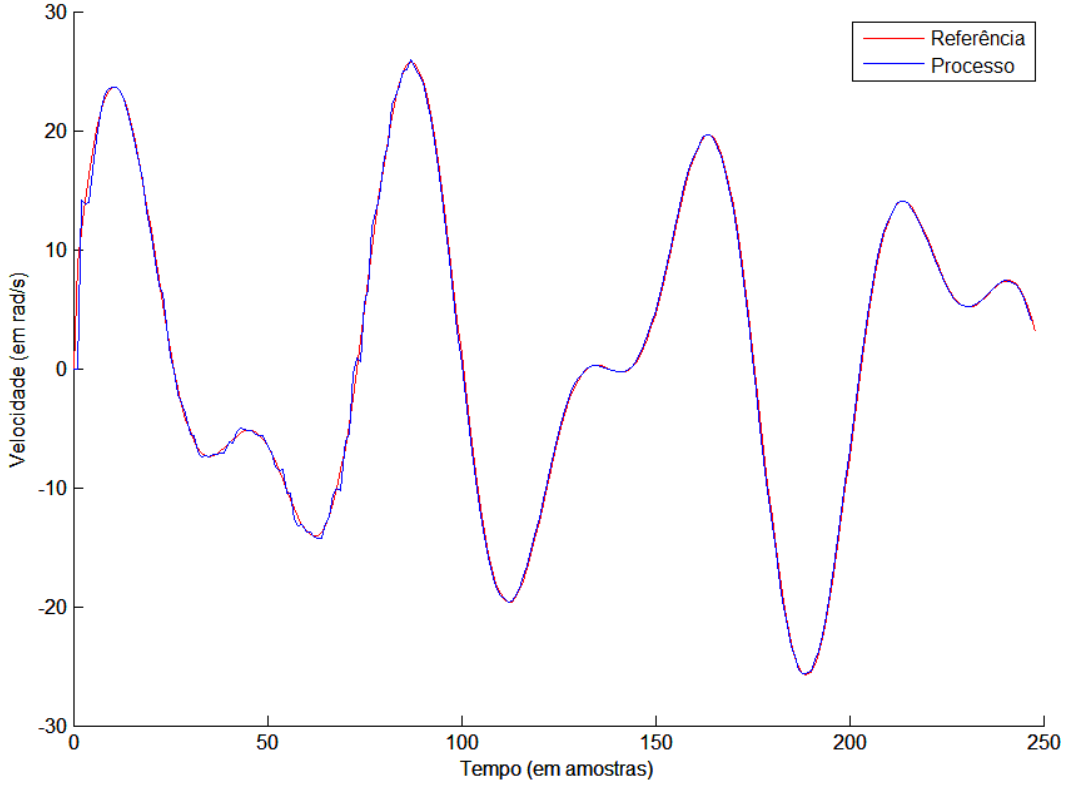


Figura 81 Resposta do processo para o controlador de referência adaptativo com entrada sinusoidal (sem ruído)

Para o sistema sem ruído na saída, são obtidas as respostas da Figura 81 e Figura 82. No Anexo B, Secção 2, podem também ser encontrados os gráficos de variação dos ganhos do PID respetivas a estas simulações. Seguindo a mesma tendência do controlador anterior, para a entrada sinusoidal o controlador consegue uma resposta bastante boa, acompanhando quase na perfeição o sinal de entrada aplicado.

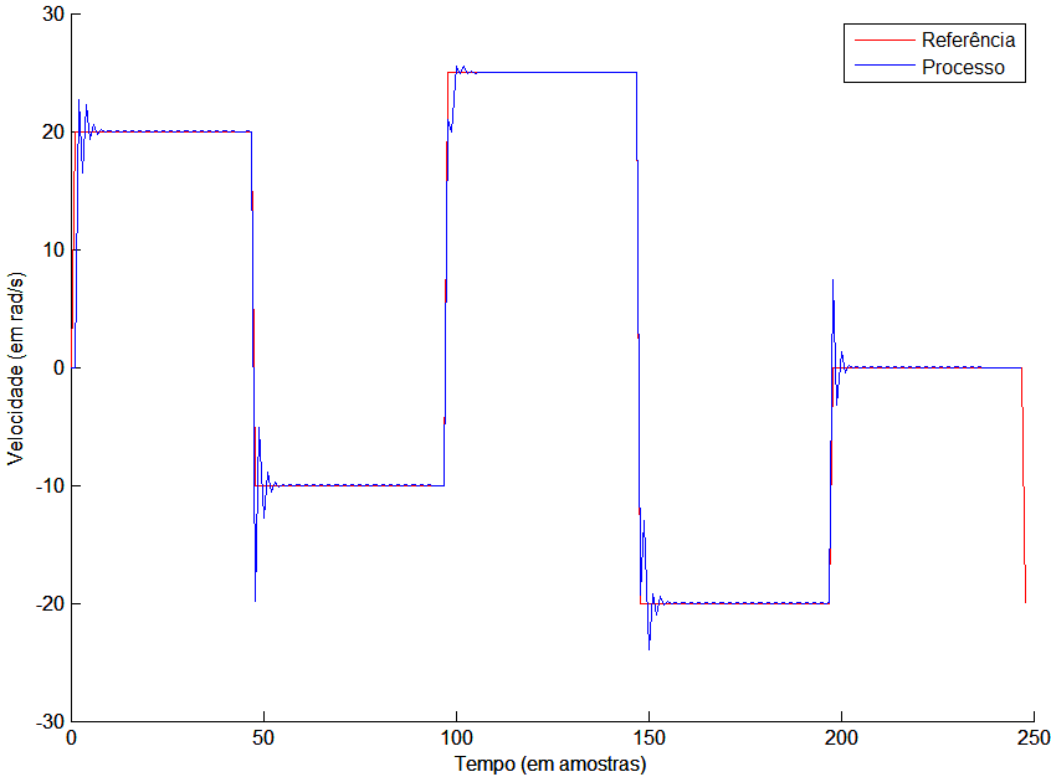


Figura 82 Resposta do processo para o controlador de referência adaptativo com entrada quadrada (sem ruído)

Para a entrada quadrada é possível verificar que, embora a rede de sintonia não seja treinada *a priori*, o seu desempenho é bastante aceitável. A resposta do processo consegue seguir de certa forma a referência aplicada ao sistema, destacando-se o rápido tempo de estabelecimento. Contudo, os maiores problemas encontrados são o *overshoot* e algumas oscilações. O *overshoot* é constante, de grande valor, e não é eliminado à medida que a rede é treinada.

Tabela 14 Valores dos índices de desempenho para o controlador de referência adaptativo sem ruído

	MSE	MAE	ISE	IAE
Entrada sinusoidal	0,586	0,390	105,089	91,613
Entrada quadrada	2,987	0,356	535,771	77,973

Depois de uma primeira análise dos gráficos, é necessária a comparação entre os valores dos índices de desempenho expostos na Tabela 14. Para a entrada sinusoidal, todos os valores calculados podem ser considerados baixos, tal como era esperado. Para a entrada quadrada, isto já não se verifica. Embora os índices que envolvem no seu cálculo o valor absoluto do erro (MAE e IAE) sejam semelhantes, ou até mais baixos aos da entrada sinusoidal, não se pode afirmar que o sistema apresenta um melhor desempenho na entrada quadrada. Empregando o erro quadrático, verifica-se que para a entrada quadrada existem erros de maior amplitude com mais frequência. Isto só é possível pois os índices MSE e ISE penalizam este tipo de diferenças entre a resposta e a referência. Relativamente aos valores desses mesmos índices, verifica-se que são superiores para a entrada quadrada.

5.2.3. COM RUÍDO

O passo seguinte é a implementação do sistema com ruído na saída, isto é, no sensor que mede a velocidade. A instrução que introduz ruído utiliza o comando `awgn`. Este ruído é do tipo Gaussiano, ou seja, é estatístico e tem uma função de densidade igual à distribuição normal. Contudo, se configurado da forma correta, este gera um sinal idêntico ao do aplicado no sistema do primeiro controlador.

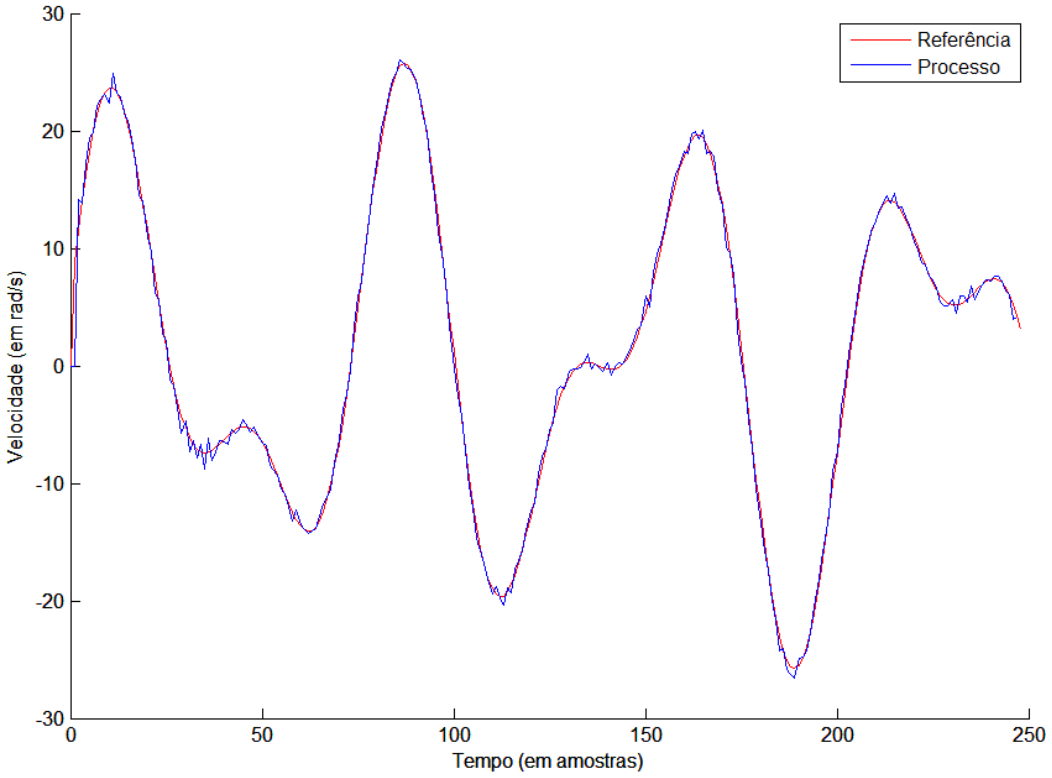


Figura 83 Resposta do processo para o controlador de referência adaptativo com entrada sinusoidal (com ruído)

Depois de realizadas as simulações, são obtidos os gráficos apresentados na Figura 83 e Figura 84. No Anexo B, Secção 3, podem também ser encontrados os gráficos de variação dos ganhos do PID respetivos a estas simulações. Para a entrada sinusoidal, o primeiro dos gráficos demonstra que o controlador não é capaz de lidar o ruído. É possível observar que os fragmentos da onda onde não existe uma grande variação no sinal durante um período de tempo relativamente amplo são mais suscetíveis ao ruído, onde surgem algumas oscilações no sinal de saída.

Para a onda quadrada, podem-se tirar as mesmas conclusões que para a entrada sinusoidal: o controlador não é capaz de eliminar o ruído do sistema. A resposta do processo segue aceitavelmente a referência. No entanto, não é capaz de estabilizar quando o sinal de entrada é linear, permanecendo o ruído sempre evidente.

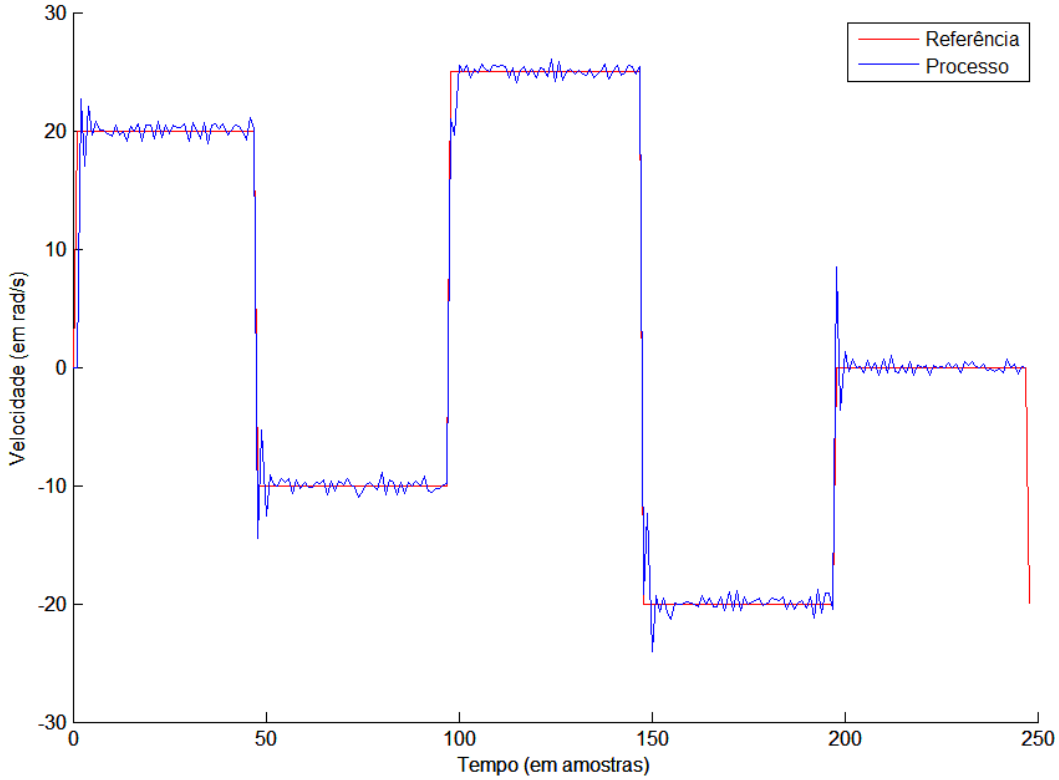


Figura 84 Resposta do processo para o controlador de referência adaptativo com entrada quadrada (com ruído)

Tabela 15 Valores dos índices de desempenho para o controlador de referência adaptativo com ruído

	MSE	MAE	ISE	IAE
Entrada sinusoidal	0,805	0,586	160,677	140,440
Entrada quadrada	3,306	0,736	619,234	171,910

Relativamente ao valor dos índices de desempenho, no geral os valores são piores que os calculados para a simulação dos sistemas sem ruído, com os valores a sofrerem aumentos ligeiros. Seguindo a mesma tendência desses sistemas, quando é introduzido o ruído a entrada quadrada apresenta valores de desempenho significativamente superiores à da entrada sinusoidal. Neste caso incluem-se os índices que utilizam o erro absoluto, o que não se passava nos sistemas sem ruído. No entanto, e apesar de existir uma diminuição no desempenho dos sistemas com ruído, os valores calculados não permitem classificar as suas respostas como más ou inaceitáveis.

5.3. COMPARAÇÃO ENTRE OS CONTROLADORES

A comparação entre controladores é concretizada através de uma análise visual dos gráficos da respostas e dos valores dos índices calculados e apresentados nas tabelas deste capítulo. Estes mostram que o controlador estabilizador é ligeiramente superior ao controlador de referência adaptativo, independentemente da sintonia utilizada. As diferenças acentuam-se mais para a entrada quadrada, onde o controlador de referência adaptativo não é capaz de apresentar um desempenho semelhante à da onda sinusoidal, enquanto o estabilizador sofre "uma queda mínima" no desempenho.

Contudo, com análise mais detalhada dos gráficos, é observável que para a onda quadrada o controlador de referência adaptativo tem um tempo de estabelecimento mais baixo que o controlador estabilizador. Além disso, apresenta um erro de estabelecimento muito próximo de zero, sendo inferior ao deste último. Os gráficos demonstram também que o controlador de referência adaptativo exibe *overshoot*s maiores que o estabilizador.

Para os sistemas com ruído, numa primeira fase, os gráficos parecem indicar que o controlador estabilizador está mais apto a lidar com ruído que o adaptativo. Isto torna-se evidente quando comparados os valores dos índices de desempenho exibidos nas tabelas dos respetivos controladores. Para a onda sinusoidal, ambas as respostas dos controladores apresentam instabilidades, mas os índices das mesmas mostram que o controlador estabilizador tem um desempenho melhor. Para a entrada quadrada, o controlador de referência adaptativo apresenta por vezes elevados *overshoot*s, e até *undershoot*s, que não são tão acentuados, ou não estão presentes, na resposta do controlador estabilizador para a mesma entrada. No entanto, pelos gráficos não é percetível qual dos controladores tem maior capacidade de lidar com os ruído quando a referência é linear. Se comparados os índices de

desempenho, as diferenças entre sistemas (sem e com ruído) do controlador estabilizador são mais pequenas que para o adaptativo, demonstrando claramente, mais uma vez, que o primeiro tem uma maior capacidade em lidar com o ruído.

Em relação ao desempenho do controlador PID, já foi visto que este não é capaz de atingir os valores de desempenho do estabilizador, mas não foi realizada essa comparação para o controlador de referência adaptativo. Isto deve-se ao facto das simulações dos controladores estabilizador e PID serem feitas em simultâneo no *Simulink*, como é observável pelos esquemas exibidos no Anexo A. A simulação do controlador estabilizador é efetuada num *script* à parte, daí a sua comparação entre este controlador e o PID ser feita apenas nesta secção.

Nos sistemas sem ruído, os índices de desempenho demonstram que o controlador de referência adaptativo é superior ao controlador PID para as sintonias Ziegler-Nichols e Coon-Cohen. Contudo, para a de Zhaung-Atherton o controlador PID apresenta melhores valores, embora com uma superioridade ligeira.

Quando comparados, em detalhe, os gráficos das respostas destes dois controladores, para a entrada sinusoidal parece que o controlador PID é ligeiramente superior para a sintonia Zhuang-Atherton devido à presença de algumas instabilidades na resposta do controlador de referência adaptativo. Essa conceção que é confirmada pelos índices de desempenho das respetivas respostas. Todavia, o mesmo não ocorre para entrada quadrada, considerando a melhor das sintonias (Zhuang-Atherton). Apesar dos índices indicarem o contrário, nomeadamente os mais importantes (MSE e ISE), a resposta do controlador de referência adaptativo é melhor que a do PID. O primeiro controlador apresenta um *overshoot* por vezes superior aos da resposta exibida pelo PID. Porém, o tempo de estabelecimento do controlador de referência adaptativo é bastante mais curto, e o erro de estabelecimento é muito próximo de zero, originando em valores de índices que utilizam o erro absoluto (MAE e IAE) mais baixos. Os índices com erro quadrático penalizam muito características como as oscilações da resposta e *overshoot*, mas em termos de estabilidade este controlador é sem dúvida superior, ideal para processos que necessitem de um erro de estabilidade mínimo.

Para os sistemas com ruído, os índices comprovam a tendência inicial dos sistemas sem ruído: a resposta do controlador PID é ligeiramente superior à do controlador de referência adaptativo para a sintonia Zhuang-Atherton. Este último controlador é bastante prejudicado

por apresentar valores de erro grandes garantindo maiores valores de MSE e ISE, especialmente para a onda quadrada.

Através da comparação dos índices, é também possível perceber como ambos os controladores se comportam na presença de ruído. O controlador PID consegue anular melhor o ruído, com diferenças mínimas entre os índices dos sistemas sem e com ruído. O controlador de referência adaptativo apresenta diferenças maiores, apesar destas não serem muito superiores às do controlador PID.

Os gráficos dos sistemas, com a presença de ruído, destes dois controladores são analisados ao pormenor de forma a ajudar a confirmar as conclusões retiradas da comparação dos valores dos índices. Para a entrada sinusoidal não é percetível qual a melhor resposta, pois ambas são bastante boas apesar de exibirem porções onde se destaca o erro introduzido, o que não permite atestar a conclusões dos índices que indicam que o PID tem uma resposta melhor. Para a entrada quadrada, o ruído está presente na resposta de ambos os controladores, não conseguindo ser anulado por nenhum deles. Contudo, o facto de exibir em algumas situações um *overshoot* significativamente maior, a resposta do controlador de referência adaptativo estabiliza muito mais rapidamente que a do PID, independentemente da sintonia. Este tem uma resposta mais suave, mas, em oposição, perde na rapidez de estabelecimento, o que pode ter grande impacto no sistema.

6. CONCLUSÃO

As RNAs são modelos estatísticos de aprendizagem inspirados pelas redes neuronais biológicas, ou seja, pelo sistema nervoso central dos animais, em particular o cérebro, e são utilizados para estimar ou aproximar funções desconhecidas que podem depender num grande número de entradas.

O elemento fundamental das redes neuronais é o neurónio. Este é caracterizado por transformar o seu sinal de entrada num de saída aplicando uma função de ativação. Contudo, para que estes sinais sejam conduzidos, é necessário um elemento importante para a definição das diferentes estruturas que as RNAs podem tomar: as ligações entre neurónios. Estas ligações são caracterizadas por conduzirem sinais entre neurónios empregando em cada uma delas um peso específico da ligação. Os pesos são definidos consoante a relevância que o sinal de saída do neurónio onde se inicia a ligação possui, para o neurónio onde a mesma termina. De referir que pode ainda ser aplicada ainda uma ligação (independente) a cada neurónio caracterizada por ter um valor de saída igual ao seu peso, denominada de valor de polarização, e caracterizada por definir um valor limiar para a função de ativação do neurónio.

Os valores que os pesos das ligações tomam são aprimorados e aperfeiçoados durante o treino da rede. Nessa fase entram os algoritmos de treino. Existem inúmeros algoritmos com

vantagens e desvantagens inerentes a cada um deles, mas destaca-se o método da retropropagação por ser fiável e fácil de aplicar.

Na área do controlo, as redes neuronais são aplicadas de forma a emular um processo, isto é, são utilizados padrões entradas-saída ou saída-entrada do processo para que a rede seja treinada até conseguir imitar uma dessas relações. O procedimento seguinte é a introdução da rede neuronal num sistema de controlo, que independentemente da estrutura escolhida, terá sempre que ser simulada através de uma plataforma de *software* que suporte as redes neuronais. De entre as plataformas existentes, foi escolhida a do MATLAB por ser muito versátil.

Na fase seguinte é realizada a implementação e simulação dos sistemas escolhidos (estabilizador e de referência adaptativo) através do MATLAB para controlar um motor DC. No entanto, o objetivo deste estudo é perceber em que medida é que a introdução das redes neuronais pode influenciar o controlo de um processo. Assim, é implementado nos sistemas de controlo um controlador PID em conjunto com as redes neuronais. No sistema do controlador estabilizador é aplicado um controlador PID (sintonizado) em realimentação, e um controlador neuronal em controlo direto. No sistema do controlador de referência adaptativo é aplicado um controlador PID (não sintonizado) em realimentação, uma rede neuronal de sintonia, e um modelo neuronal do processo que auxilia na sintonia realizada pela primeira rede. É também implementado um sistema de realimentação, apenas com um controlador PID, juntamente com os esquemas do primeiro sistema de controlo, para uma comparação de respostas e desempenhos, verificando a influência das redes neuronais no controlo.

Para que os testes demonstrem na generalidade quais os melhores sistemas, foram realizadas simulações com entradas quadradas e sinusoidais, e com a adição ou não de ruído.

Os resultados para o controlador estabilizador indicam uma clara superioridade em todos os gráficos de resposta e índices de desempenho. Independentemente da sintonia utilizada, este controlador é sempre superior ao sistema composto apenas pelo controlador PID para a mesma sintonia. A maior diferença verifica-se para a sintonia com piores resultados: Ziegler-Nichols. Enquanto o PID tem um desempenho muito pobre, se for adicionado um controlador neuronal, relegando o controlo efetuado pelo PID para segundo plano, o

desempenho do sistema melhora substancialmente. Este argumento é atestado pelo facto do controlador estabilizador ter desempenhos semelhantes para as diferentes sintonias.

Os resultados do controlador de referência adaptativo indicam que em termos de desempenho é inferior ao estabilizador e ao do PID, mas com valores muito próximos deste último quando considerada a melhor sintonia (Zhuang-Atherton). Contudo, o controlador adaptativo exibe respostas bastante boas. Propriedades como tempo e erro de estabelecimento são melhores do que aquelas demonstradas pelas respostas do sistema do controlador PID em realimentação. Daí não se poder afirmar que o sistema do controlador de referência adaptativo, tal como está configurado, é inferior ao sistema do PID. No entanto, existe outra conclusão a extrair: este sistema tem uma capacidade menor em lidar com o ruído relativamente às dos outros dois controladores. Como os gráficos das simulações deste trabalho não são esclarecedores, recorre-se aos índices de desempenho para extrair as devidas conclusões, e nesse caso quase todos estes têm valores superiores para o controlador de referência adaptativo.

Porém, é importante referir que as características definidas para o controlador de referência adaptativo não se encontram otimizadas. Para a rede de sintonia do controlador PID do sistema são definidas propriedades variáveis como a taxa de aprendizagem, *momentum*, pesos iniciais, entre outras, mas a escolha dos seus valores é apenas realizada de modo a permitir a convergência efetuada pelo algoritmo de treino. Ora se a configuração destas variáveis não garantirem a convergência para um mínimo absoluto, a rede de sintonia, e por sua vez o sistema do controlador de referência adaptativo, estará sempre em sub-rendimento pois o treino não foi eficaz.

No entanto, não só as variáveis têm influência no desempenho. Sendo a rede treinada *online*, o algoritmo de treino é também importante. O método de treino utilizado foi o da retropropagação do erro por ser o mais simples de aplicar e muito utilizado, mas este é bastante lento no que toca à sua convergência. Talvez se fosse utilizado outro método de treino mais eficaz, como por exemplo o de Levenberg-Marquardt, ou se fossem realizados testes com maior tempo de simulação, o sistema do controlador de referência adaptativo obtivesse índices de desempenho inferiores aos calculados.

Por fim, não obstante os resultados obtidos, é possível realçar a capacidade notável das redes neuronais de extração de significado de dados imprecisos, através da deteção de padrões e tendência complexas. Estas apresentam vantagens como:

- Aprendizagem adaptativa – A habilidade de aprender a realizar tarefas com base em dados apresentados ou através da experiência;
- Auto-organização – Uma RNA pode criar a sua própria organização ou uma representação da informação que recebe durante o período de treino;
- Operação em tempo real – A possibilidade dos seus cálculos serem realizados em paralelo;
- Tolerância a falhas via informação redundante – A degradação do desempenho face à destruição parcial de uma rede, não significa que a mesma perca todas as suas capacidades.

Para perspetivas futuras podem ser considerados os seguintes pontos de abordagem:

- Aplicar o esquema do controlador de referência adaptativo utilizando métodos das suas variáveis, como por exemplo a utilização de algoritmos genéticos na otimização de pesos da rede de sintonia, de forma a melhorar o seu desempenho;
- Aplicação de outros sistemas de controlo apresentados neste relatório utilizando as redes neuronais, e comparação de resultados;
- Aplicação dos sistemas aqui implementados usando tecnologias diferentes (tais como *fuzzy*, neuro-*fuzzy*, ou algoritmos genéticos) e comparação dos resultados;

Referências Documentais

[1] SPALL, James C., "A Neural Network Controller for Systems with Unmodeled Dynamics with Applications to Wastewater Treatment", IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics, vol. 27, Junho 1997

[2] Ranka, S., Mehrotra, K., Mohan, C. K., "Elements of Artificial Neural Networks", MIT Press, 1996

[3] McCulloch, W.S., Pitts, W.;"A logical calculus of the ideas immanent in nervous activity", Bulletin of Mathematical Biology, 5(4):115–133, 1943

[4] *Walter Pitts*
http://web.csulb.edu/~cwallis/artificialn/walter_pitts.html (16/02/2015)

[5] Fausett, L. V., "Fundamentals of Neural Networks: Architectures, Algorithms And Application", Pearson 1st, Dezembro 1993

[6] Hebb, D. O., "The Organization of Behavior", 1949

[7] Kriesel, D., "A Brief Introduction to Neural Networks", 2007, disponível em http://www.dkriesel.com

[8] Widrow, B., Hoff, M. E., "Adaptive switching circuits", WESCON, páginas 96–104, 1960

[9] Minsky, M., Papert, S., "Perceptrons", MIT Press, Cambridge, Mass, 1969

[10] Kohonen, T., "Correlation matrix memories", IEEEtC, C-21:353–359, 1972

[11] Anderson, J. A., "A simple neural network generating an interactive memory", Mathematical Biosciences, 14:197–220, 1972

[12] Werbos, P. J., "Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences", PhD thesis, Harvard University, 1974

[13] Kohonen, T., "Self-organized formation of topologically correct feature maps", Biological Cybernetics, 43:59–69, 1982

[14] Hopfield, J.J., "Neurons with graded response have collective computational properties like those of two-state neurons", Proceedings of the National Academy of Sciences, 81(10):3088–3092, 1984

[15] Rojas, R., *Neural Networks: A Systematic Introduction*, Springer-Verlag, Berlin, 1996

[16] Haykin, S., "Neural Networks and Learning Machines", Pearson, 3rd Edition, 2009

[17] Hajek, M., "Neural Networks", University of KwaZulu-Natal, 2005

[18] Hu, Y. H., Hwang, J., "Handbook of Neural Network Signal Processing", The Electrical Engineering and Applied Signal Processing Series, CRC Press, 2002

[19] Freeman, J. A., "Neural Networks: Algorithms, Application and Programming Techniques", Addison-Wesley Publishing Company, 1991

[20] Krose, B., Van der Smagt, P., "An Introduction to Neural Networks", The University of Amsterdam, 8th Edition, November 1996

[21] Gahegan, M., West, G, "The Classification of Complex Geographic Datasets: An Operational Comparison of Artificial Neural Network and Decision Tree Classifiers", School of Computing, Curtin University of Technology, 1998 Disponível em http://www.geocomputation.org/1998/61/gc_61.htm, e consultado em 13/04/2015

[22] Tudoran, C., Neagoe, V., "A New Neural Network Approach for Visual Autonomous Road Following", Departamento de Eletrónica, Universidade de Bucareste, Bucareste, Roménia

[23] Egeli, B., Ozturan, M., Badur, B., "Stock Market Precidtion Using Artificial Neural Networks", Departamento de Sistemas de Gestão de Informação, Universidade Bogazici, Istambul, Turquia

[24] Le Cun, Y., "Learning Processes in a Asymmetric Threshold Network" In E. Bienenstock, F. Fogelman-Souli, & G. Weisbuch, Disordered Systems and Biological Organization, série NATO ASI, Berlim, Springer Verlag, 1986

[25] Sejnowski,T. J., Rosenberg, C. R. (1986), "NETtalk: A Parallel Network That Learns to Read Aloud", The Johns Hopkins University Electrical Engineering and Computer Science Technical Report JHU/EECS-86/01, 33 pp. Reprinted in Anderson & Rosenfeld, pp 663-672, 1988

[26] Lippmann, R. P., "Review of Neural Networks for Speech Recognition", Neural Computation, 1ª edição, 1989

[27] Fritsch, Th. et al, "A self-organizing neural net clustering Parkison patients and control persons using motor data", Computer-Based Medical Systems, 1995

[28] Sitte, J., Korner, T., Ruckert, U., “An analog-current mode local cluster neural net”, Emerging Technologies and Factory Automation Proceedings, 1997

[29] Widrow, B., Stearns, S. D., “Adaptive Signal Processing”, Eglewood Cliffs, NJ: Prentice-Hall, 1985

[30] Anderson, J. A., “Cognitive Capabilities of a Parallel System”, Disordered Systems and Biological Organization, Berlim: Springer.Verlag, 1986

[31] Anderson, J. A., Golden, R. M., Murphy, G. L., “Concepts in Distributed Systems”, Optical and Hybrid Computing, WA: Society of Photo-Optical Instrumentation Engineers, 1986

[32] Koiran, P., “Function approximation using a partition of the input space”, International Joint Conference on Neural Nets, Volume 1, 1992

[33] Geva, S., Sitte, J., “A Constructive Method for Multivariate Function Approximation by Multilayer Perceptrons”, IEEE Transaction on Neural Networks, 1992

[34] Sun, Y., Qian, J., “Forecast the Dangerous Level of Gas Outburst Based on GASA Neural Networks in Coal Mine”, 3rd International Conference on Natural Computation, 2007

[35] Xiao-Wen, W. et al, “The Short-Term Load Forecasting by Applying the Fuzzy Neural Net, 6th International Conference on Intelligent Networks and Intelligent Systems (ICINIS), 2013

[36] Collins, E., Ghosh, S., Scofield, C. L., “An Application of a Multiple Neural Network Learning System to Emulation of Mortgage Underwriting Judgements”, IEEE International Conference on Neural Networks, 1988

[37] Maa, C.-Y., Chiu, C., Shanblatt, M. A., “A constrained optimization neural net technique for economic power dispatch”, IEEE International Symposium on Circuits and Systems, 1990

[38] Krawczak, M., Sotirov, S., Sotirova, E., “Generalized net model for parallel optimization of multilayer neural network with time limit”, 6th IEEE International Conference on Intelligent Systems (IS), 2012

[39] Koh, K. C., Cho, H. S., “A neural net-based feedfoward control scheme for mobile robots”, IEEE/RSJ Internation Workshop on Intelligent Robots and Systems, 1991

[40] Xuejing, Y., Peng, L., Xiuyan, P., "Online Neural-net Control System for Ship Motion Stabilization", International Conference on Mechatronics and Automation, 2007

[41] Nguyen, D., Widrow, B., "Truck Backer-Upper: An Example of Self-Learning in Neural Networks", International Joint Conference on Neural Networks, Washington, 1989

[42] Miller, W. T., Sutton, R. S., Werbos, P. J., "Neural Networks for Control", MIT Press, Cambridge, 1990

[43] Hagan, M. T., Demuth, H. B., "Neural Networks for Control", American Control Conference, Junho 1999

[44] B. Widrow and E. Walach, "Adaptive Inverse Control", New Jersey: Prentice Hall, 1996

[45] *Sistema Nervoso*
htpp://www.auladeanatomia.com/neurologia/sneroso.htm, consultado em 01/04/2015

[46] Gurney, K., "Neural networks for perceptual processing: from simulation tools to theories", Adaptive behavior research group, Department of Psychology, University of Sheffield, 8 de Janeiro de 2007
Disponível em http://rstb.royalsocietypublishing.org/content/362/1479/339, e consultado em 09/04/2015

[47] Breckon, T., "Machine Learning for Computer Vision – a whirlwind tour of key concepts for the uninitiated", School of Engineering and Computing Sciences, Durham University, 2014
Disponível em http://www.slideshare.net/potaters/bmva-ss-2014breckonml, e consultado em 13/04/2015

[48] Zell, A, e outros, "SNNS Stuttgart Neural Network Simulator", Institute for Parallel and Distributed High Performance Systems (IPVR), University of Stuttgart, Versão 4.2

[49] *What is SNNS?*
Disponível em http://www.ra.cs.uni-tuebingen.de/SNNS/announce.html, e consultado em 28/04/2015

[50] *Stuttgart Neural Network Simulator*
Disponível em http://www.ra.cs.uni-tuebingen.de/SNNS/, e consultado em 28/04/2015

[51] Fischer, I., "JavaNNS Java Neural Network Simulator", Wilhelm-schickard-institute for computer science, University of Tubingen, Versão 1.1

[52] *JavaNNS: Java Neural Network Simulator*
Disponível em http://www.ra.cs.uni-tuebingen.de/software/JavaNNS/, e consultado em 29/04/2015

[53] *NetMaker Neural networks simulator and designer*
Disponível em http://www.ire.pw.edu.pl/~rsulej/NetMaker/, consultado em 30/04/2015

[54] Astrom K. J. and Hagglund T. H., "New tuning methods for PID controllers", Proceedings of the 3rd European Control Conference, 1995

[55] D. Nguyen and B. Widrow. Improving the Learning Speed of 2-Layer Neural Networks by Choosing Initial Values of the Adaptive Weights. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN) San Diego,* vol. III, pp. 21–26, Edward Brothers, 1990

[56] Franklin, G. F. *et al.*, "Digital Control of Dynamic Systems", Addison-Wesley, 3rd Edition, 1997

[57] Gopal, M., "Digital Control and State Variable Methods: Conventional and Intelligent Control Systems" , McGraw-Hill Higher Education, 3rd Edition, 2010

Anexo A. Elementos de desenvolvimento e simulação do controlador estabilizador

Secção 1 – Código de desenvolvimento e treino do controlador neuronal

Secção 1.1 – Definição das variáveis; gerar sinal aleatório de entrada; gerar valores de saída do motor para treino

```
%% Variáveis

global r T N

N=500;  % Numero de amostras
T=1;   % Tempo de amostragem


%% Modelo do processo e tensão de entrada


% Constantes
K1=0.34366;
K2=-0.1534069;
K3=-2.286928e-3;
K4=3.5193358e-4;
K5=0.2280595;
K6=-0.105184;


% Dados de treino
valor=0;
va(1)=0;
for k=1:N
    if mod(k,50)==0
        valor=(0.5-rand)*200; % -100<va<100
    end
    va(k+1)=valor;
end


% Modelo do processo (saída-velocidade)
w(1)=0;w(2)=0;
for k=1:(N-2)

w(k+2)=K1*w(k+1)+K2*w(k)+sign(w(k+1))*(K3*w(k+1)*w(k+
1)+K4*w(k)*w(k))+K5*va(k+1)+K6;
end
```

Secção 1.2 – Controlador Neuronal

```
%% Controlador Neuronal

% Entradas da rede
for k=1:(N-2)
    input(1,k)=w(k+2)/30;
    input(2,k)=w(k+1)/30;
    input(3,k)=w(k)/30;
    target(k)=va(k+1)/100;
end


% Configuração da rede

net = network(1,2,[1;1],[1;0],[0 0;1 0],[0 1]);
% N° de camadas de entrada, N° de camadas, Camadas
com ligação ao bias,
% Camadas com ligação a camada de entrada, Ligação
entre camadas, Camada
% de saída

net.inputs{1}.size = 3;      % N° entradas na camada
de entrada 1
net.layers{1}.size = 5;      % N° de neurónios na
camada 1
net.layers{2}.size = 1;      % N° de neurónios na
camada 2

net.layers{1}.transferFcn = 'tansig';     % Definir a
função sigmoid para a camada 1
net.layers{2}.transferFcn = 'purelin';     % Definir a
função linear para a camada 2

net.initFcn = 'initlay';    % Cada camada tem a sua
rotina de inicialização
net.layers{1}.initFcn = 'initwb';
net.layers{2}.initFcn = 'initwb';
net.inputWeights{1,1}.initFcn = 'rands';
Inicialização com valores aleatórios
net.layerWeights{2,1}.initFcn = 'rands';
net.biases{1}.initFcn='rands';
net.biases{2}.initFcn='rands';

view(net)


% Parâmetros de treino e inicialização
net.trainFcn = 'trainlm';    % Algoritmo de Default no
comando newff
```

```
net.trainparam.epochs=1000;
net.trainparam.goal=1e-7;
net.trainparam.show=50;
net.trainparam.lr=0.1;
net.performFcn='msereg';    % Utilizar MSEREG em vez
de MSE para melhor generalização

net = init(net);    % Inicializar a rede
```

Secção 1.3 – Treino da rede e gráfico de desempenho

```
%% Treino da rede e grafico de desempenho

[net,tr]=train(net,input,target);

y_NN=sim(net,input)*100;

figure(1);
plot(va);hold on;plot(y_NN,'r');
legend('Sinal aleatório','Controlador');
```

Secção 1.4 – Teste de generalização

```
%% Teste

% Sinal de referência
% Degraus;
Passos;

% Simulação do sistema
w_sim(1)=0;
w_sim(2)=0;
for k=1:(N-2)
    NN_input=[r(k+2);w_sim(k+1);w_sim(k)]/30;
    va_sim(k+1)=sim(net,NN_input)*100;

w_sim(k+2)=K1*w_sim(k+1)+K2*w_sim(k)+sign(w_sim(k+1))
*(K3*w_sim(k+1)*w_sim(k+1)+K4*w_sim(k)*w_sim(k))+K5*v
a_sim(k+1)+K6;
end

figure(2)
plot(r);hold on; plot(w_sim,'r');
legend('Referência','Processo');
```

Secção 1.5 – Exportação para *Simulink*

```
%% Exportar rede para o simulink

gensim(net);
```

Secção 2 – Código de calculo da função de primeira ordem do motor

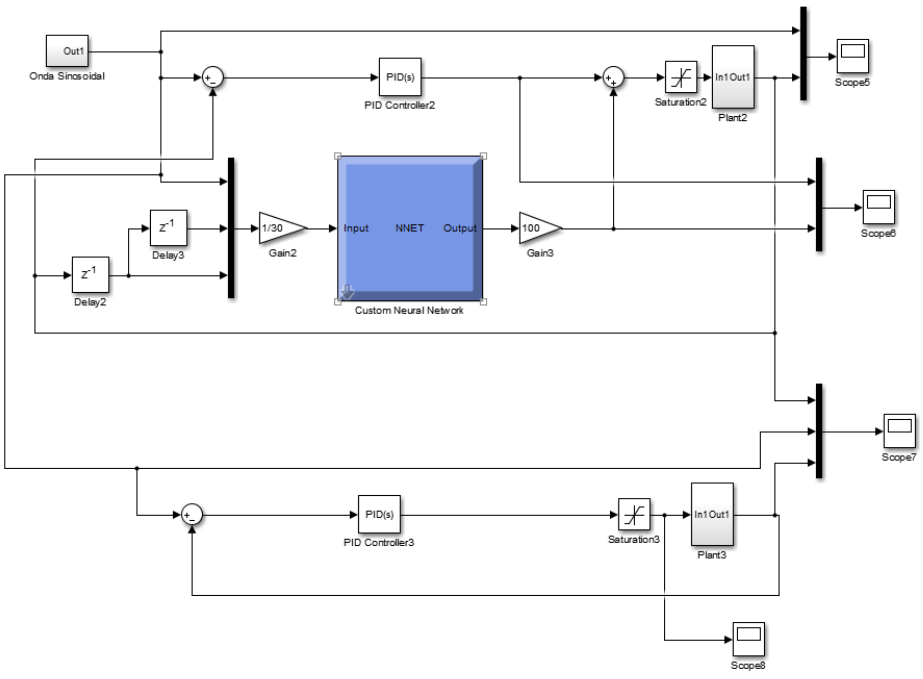
Secção 2.1 - Função Objetivo

```
function J = fun_ordem(x,t,y)

K = x(1);
L = x(2);
T = x(3);

s = tf('s');
G = K*exp(-L*s)/(T*s+1);
y1 = step(G,t);

J = sum((y(:)-y1(:)).^2);  % Calcula valor da função
objectivo
```

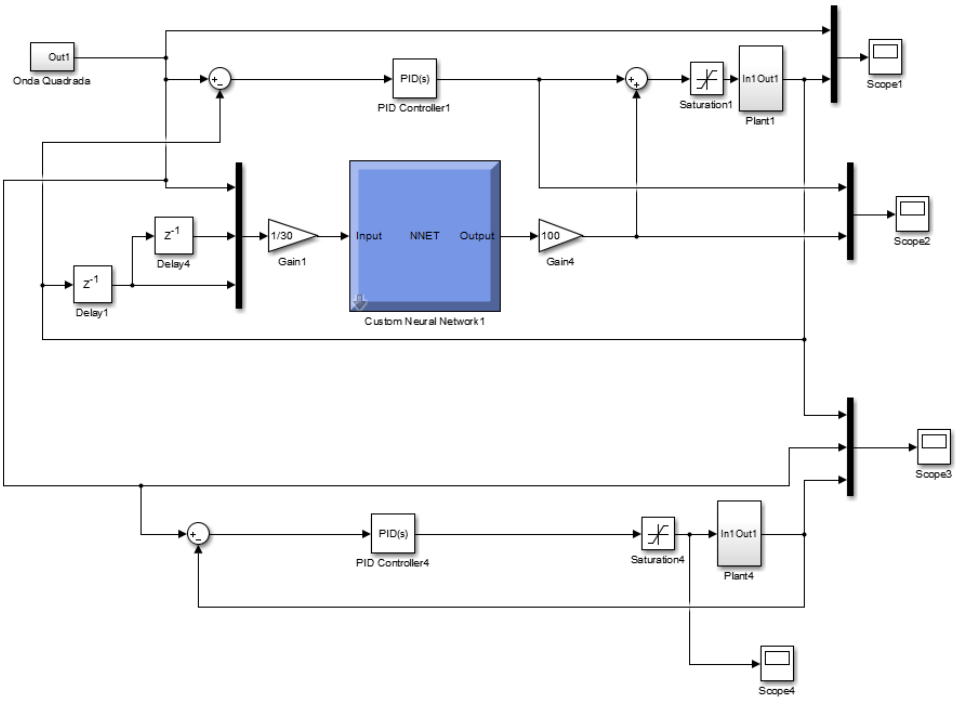
Secção 2.2 – Implementação da função de `fmincon`

```
t = y_zn_ma(:,1);
y = y_zn_ma(:,2);

x0 = [1 1 1];
Lb = [0 0 0]; Ub =[2 3 2]
[x fval] =
fmincon('fun_equacao1',x0,[],[],[],[],Lb,Ub,[],[],t,y
)
```

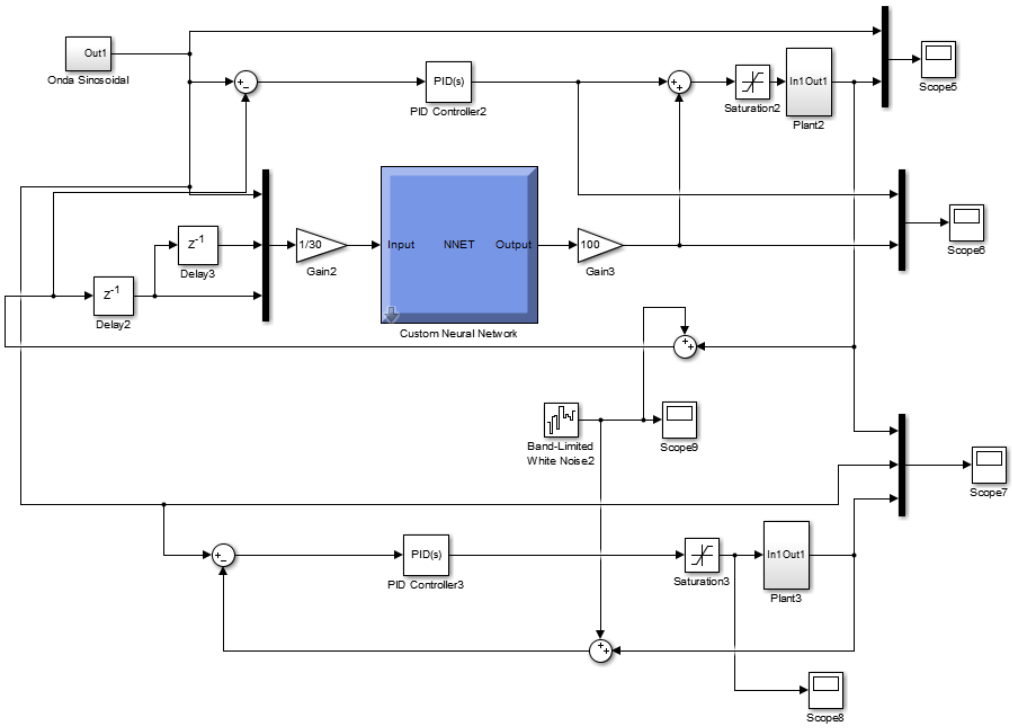
Secção 3 – Esquema de simulação do controlador estabilizador para uma entrada sinusoidal



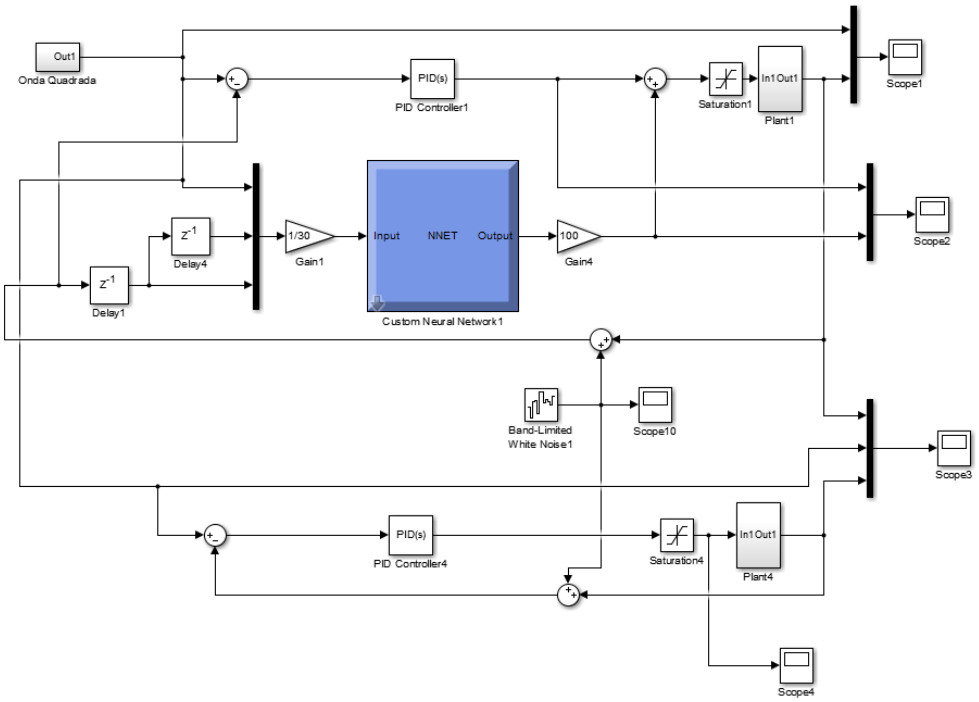
Secção 4 - Esquema de simulação do controlador estabilizador para uma entrada quadrada



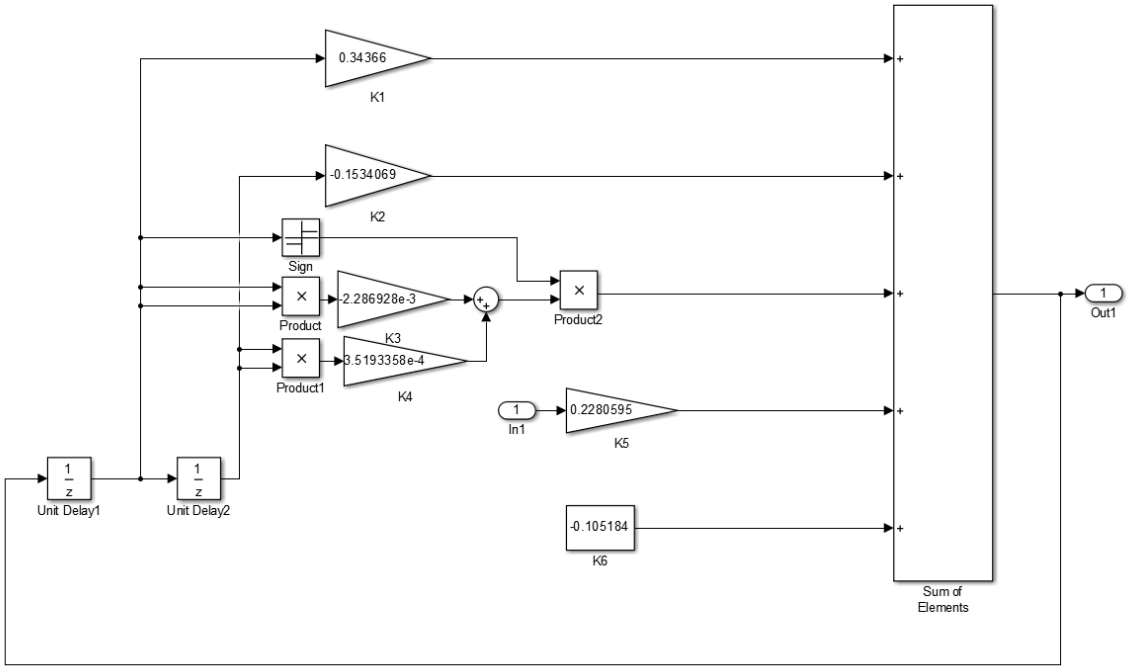
Secção 5 - Esquema de simulação do controlador estabilizador para uma entrada sinusoidal e com ruído no sensor



Secção 6 - Esquema de simulação do controlador estabilizador para uma entrada quadrada e com ruído no sensor



Secção 7 – Esquema do motor DC



Anexo B. Esquemas de simulação do controlador de referência adaptativo

Secção 1 – Código de desenvolvimento e treino do modelo neuronal do motor

Secção 1.1 – Definição das entradas e saída da rede para treino

```
% Entradas e saída do treino
for k=1:(N-2)
    input(1,k)=w(k+1)/30;
    input(2,k)=w(k)/30;
    input(3,k)=va(k+1)/100;
    input(4,k)=va(k)/100;
    target(k)=w(k+2)/30;
end
```

Secção 1.2 – Exportar pesos da rede treinada

```
%% Pesos da rede treinada
wij=net.IW{1,1}
bw=net.b{1}
vjl=net.LW{2,1}
bv=net.b{2}
```

Secção 1.3 – Cálculos da rede neuronal de sintonia

```
%% Definição das entradas

xt(1,k)=u(k-1)/100;
xt(2,k)=u(k-2)/100;
xt(3,k)=w_sim(k)/30;
xt(4,k)=w_sim(k-1)/30;
xt(5,k)=r(k)/30;
xt(6,k)=r(k-1)/30;

%% Calculo da saída da rede

sum=0;

for i=1:Nut
    sum=wt0(i,k); % Limiar
    for j=1:6
        sum=sum+wt(i,j,k)*xt(j,k);
    end
    zt(i,k)=tansig(sum);
end
```

```
for i=1:3
    sum=vt0(i,k); % Limiar
    for j=1:Nut
        sum=sum+zt(i,k)*vt(i,j,k);
    end
    if i==1
        Kp=logsig(sum)*Kp_K;
        if Kp<0.01
            Kp=0.01;
        end
    else if i==2
            Ki=logsig(sum)*Ki_K;
            if Ki<0.01
                Ki=0.01;
            end
        else
            Kd=logsig(sum)*Kd_K;
            if Kd<0.01
                Kd=0.01;
            end
        end
    end
end
```

Secção 1.4 – Método de retropropagação na rede neuronal de sintonia

```
%% Atualização dos pesos

theta31=e(k)*J(k)*(e(k)-e(k-1))*(logsig(Kp(k-1))*(1-
logsig(Kp(k-1))));
theta32=e(k)*J(k)*e(k)*(logsig(Ki(k-1))*(1-
logsig(Ki(k-1))));
theta33=e(k)*J(k)*(e(k)-2*e(k-1)+e(k-
2))*(logsig(Ki(k-1))*(1-logsig(Ki(k-1))));

for l=1:3
    if l==1
        vt0(l,k)=vt(l,k-
1)+Lrt*theta31*1+Mom*(vt0(l,k-1)-vt0(l,k-2));
        else if l==2
                vt0(l,k)=vt(l,k-
1)+Lrt*theta32*1+Mom*(vt0(l,k-1)-vt0(l,k-2));
            else
                vt0(l,k)=vt(l,k-
1)+Lrt*theta33*1+Mom*(vt0(l,k-1)-vt0(l,k-2));
            end
    end

    for i=1:Nut
       if l==1
            vt(l,i,k)=vt(l,i,k-1)+Lrt*theta31*zt(i,k-
1)...
```

```
                +Mom*(vt(l,i,k-1)-vt(l,i,k-2));
            else if l==2
                    vt(l,i,k)=vt(l,i,k-
1)+Lrt*theta32*zt(i,k-1)...
                        +Mom*(vt(l,i,k-1)-vt(l,i,k-
2));
                else
                    vt(l,i,k)=vt(l,i,k-
1)+Lrt*theta33*zt(i,k-1)...
                        +Mom*(vt(l,i,k-1)-vt(l,i,k-
2));
                end
        end
         wt0(i,k)=wt0(i,k-1)+Lrt*1*(theta31*vt(1,i,k-
1)+theta32*vt(2,i,k-1)...
            +theta33*vt(3,i,k-1))*((1-(zt(i,k-
1))^2)/2)+Mom*(wt0(i,k-1)-wt0(i,k-2));
         for j=1:6
            wt(i,j,k)=wt(i,j,k-1)+Lrt*xt(j,k-1)*((1-
(zt(i,k-1))^2)/2)...
                *(theta31*vt(1,i,k-
1)+theta32*vt(2,i,k-1)+theta33...
                *vt(3,i,k-1))+Mom*(wt(i,j,k-1)-
wt(i,j,k-2));
         end
    end
end
```

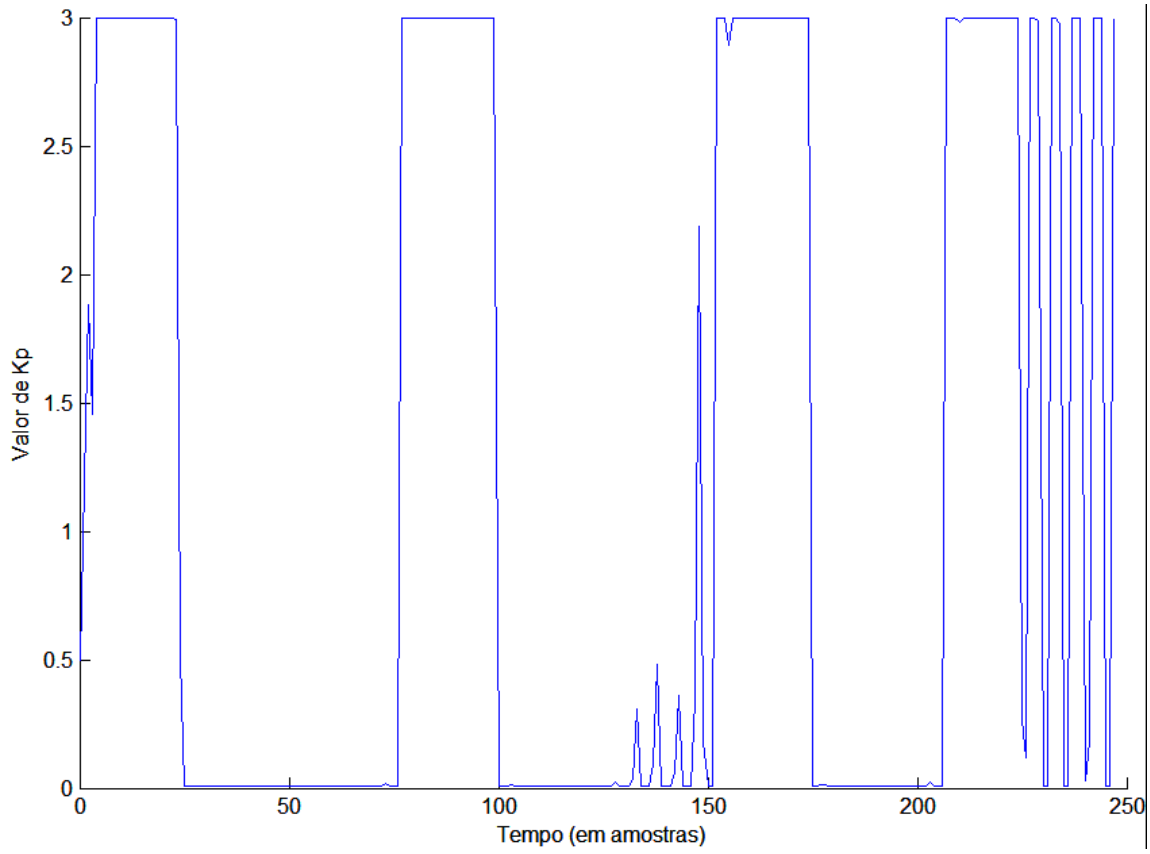
Secção 1.5 – Cálculos do modelo neuronal do motor

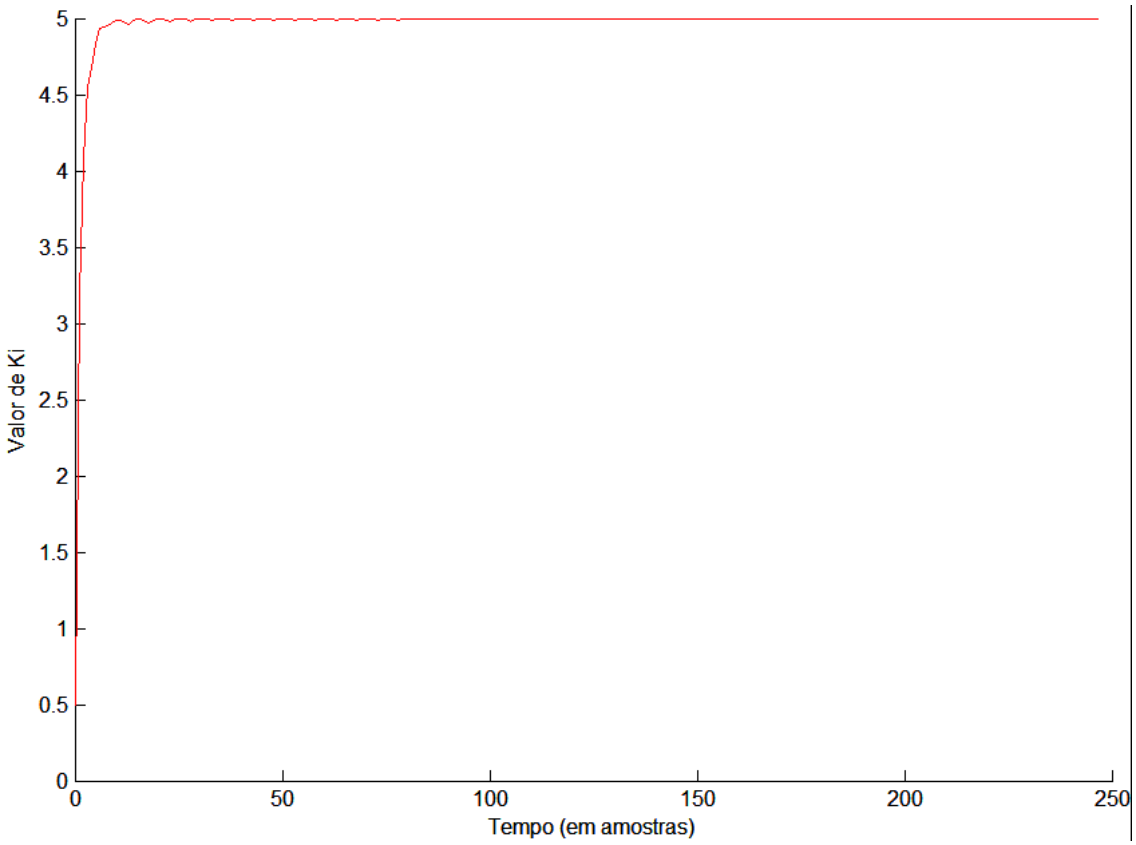
```
%% Definição das entradas

xm(1,k)=w_sim(k)/30;
xm(2,k)=w_sim(k-1)/30;
xm(3,k)=u(k)/100;
xm(4,k)=u(k-1)/100;

%% Calculo da saída da rede
sum=0;
for i=1:Num
    sum=bw(i);
    for j=1:4
        sum=sum+wij(i,j)*xm(j,k);
    end
    zm(i)=tansig(sum);
end

sum=bv;
for i=1:Num
    sum=sum+vjl(1,i)*zm(i);
end
```

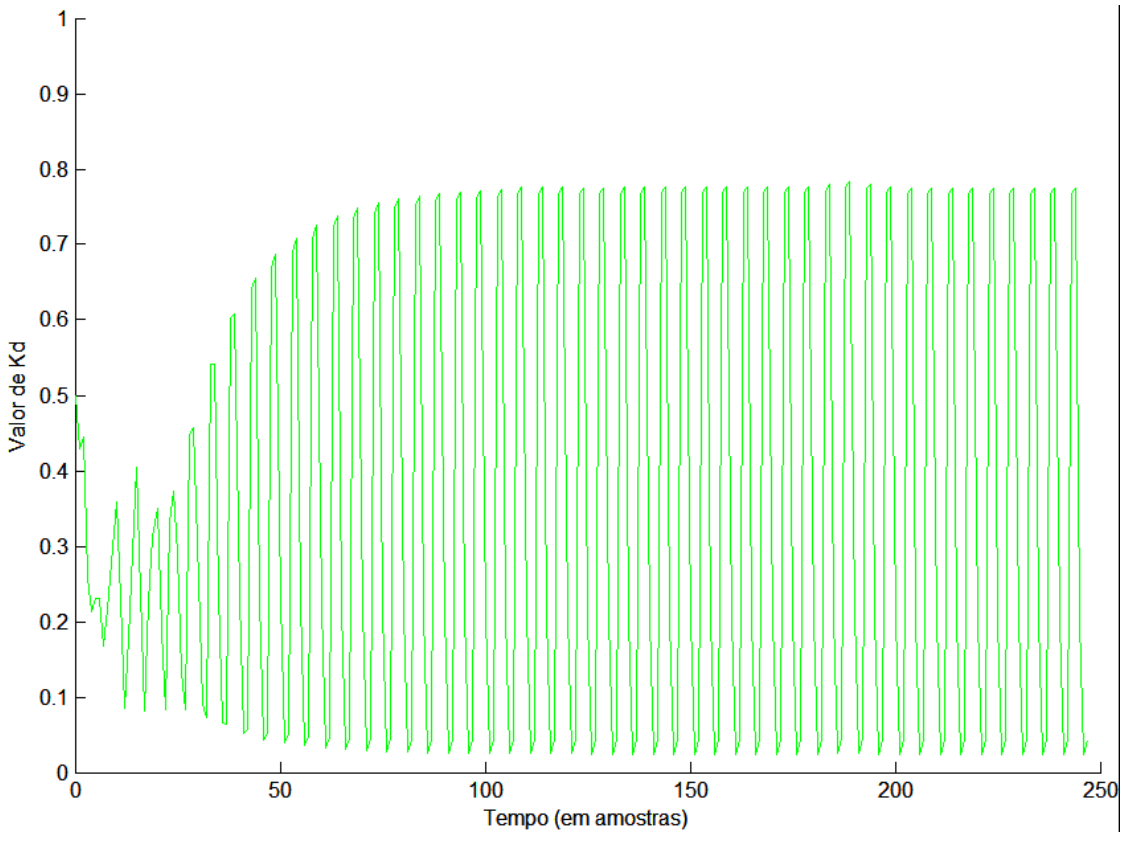
```
y_previsto=sum*30;

%% Calculo do Jacobiano

sum=0;
for i=1:Num
    sum=sum+((1-zm(i)*zm(i))/2)*vjl(i)*wij(i,3);
end

Jacobian=sum;
```

Secção 2 – Gráficos da variação dos ganhos para o sistema sem ruído

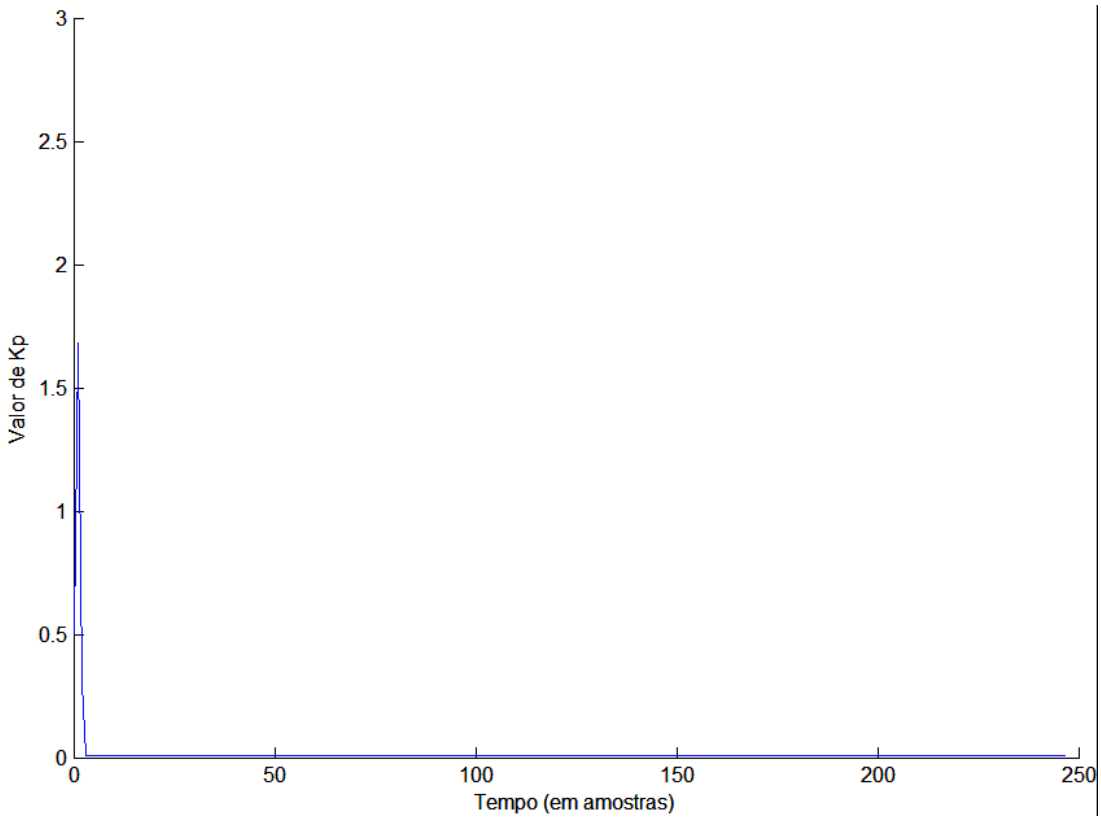
Secção 2.1 – Entrada sinusoidal

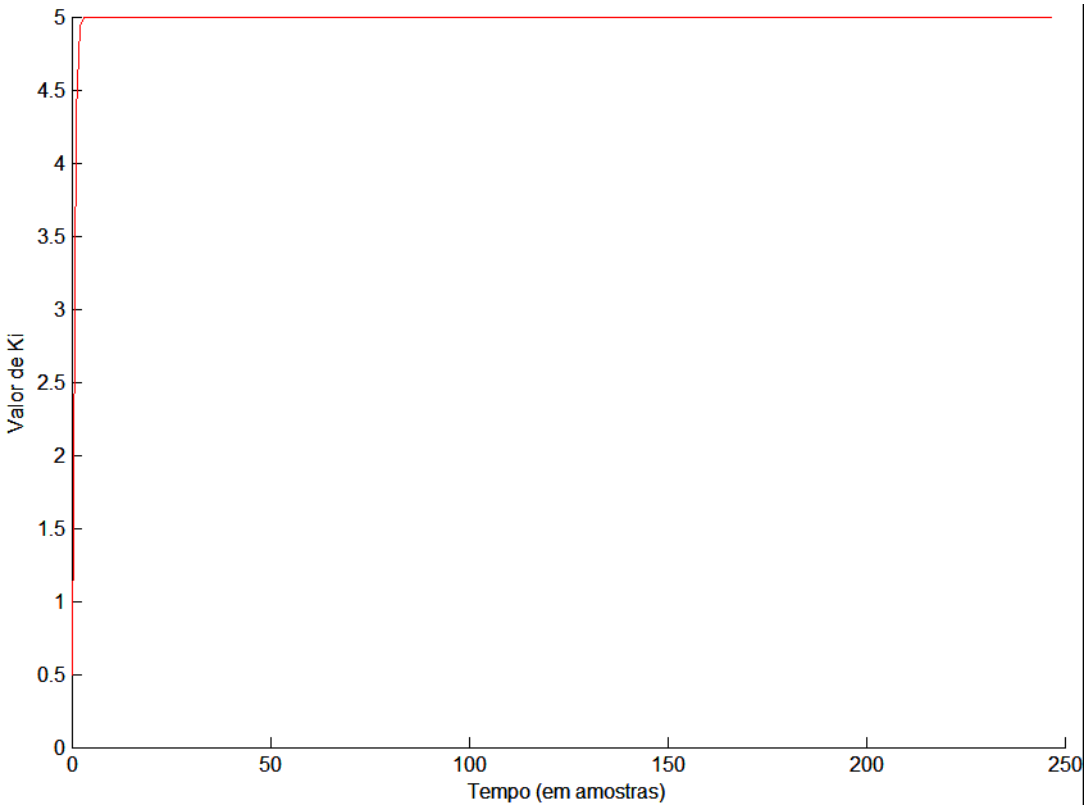


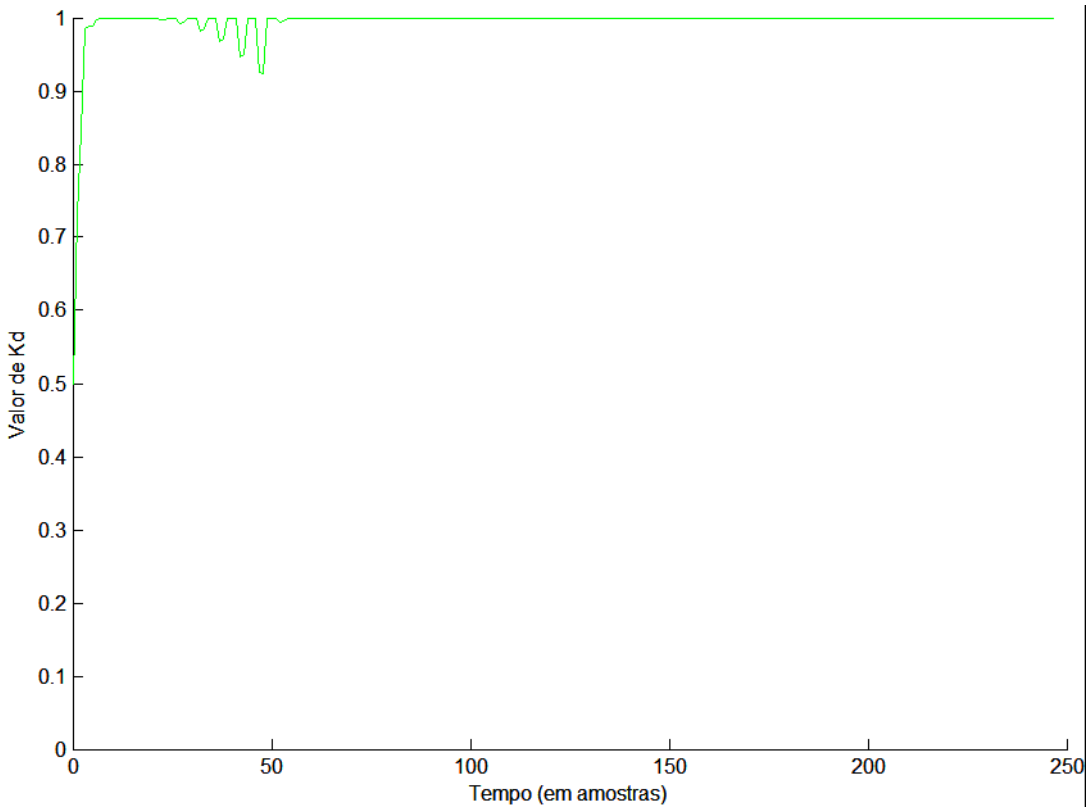

Valor de Ki
Tempo (em amostras)
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
0
50
100
150
200
250


Valor de Kd
Tempo (em amostras)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
50
100
150
200
250

Secção 2.2 – Entrada quadrada

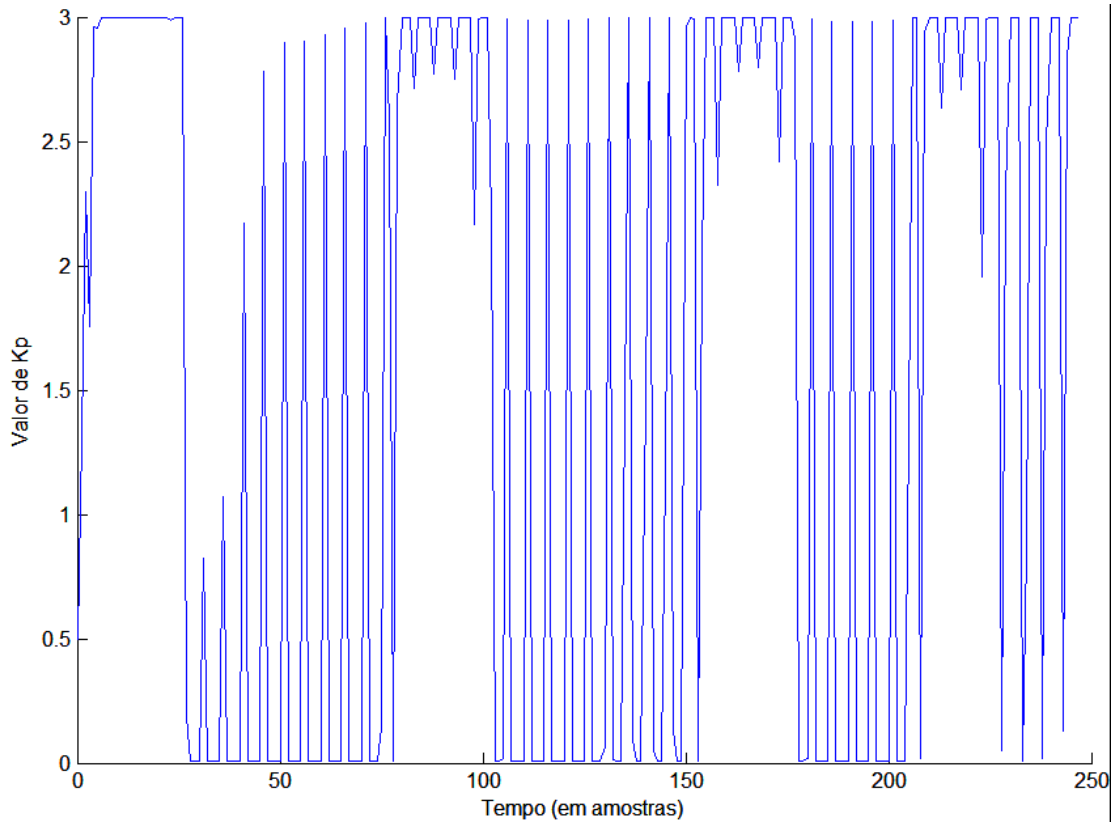


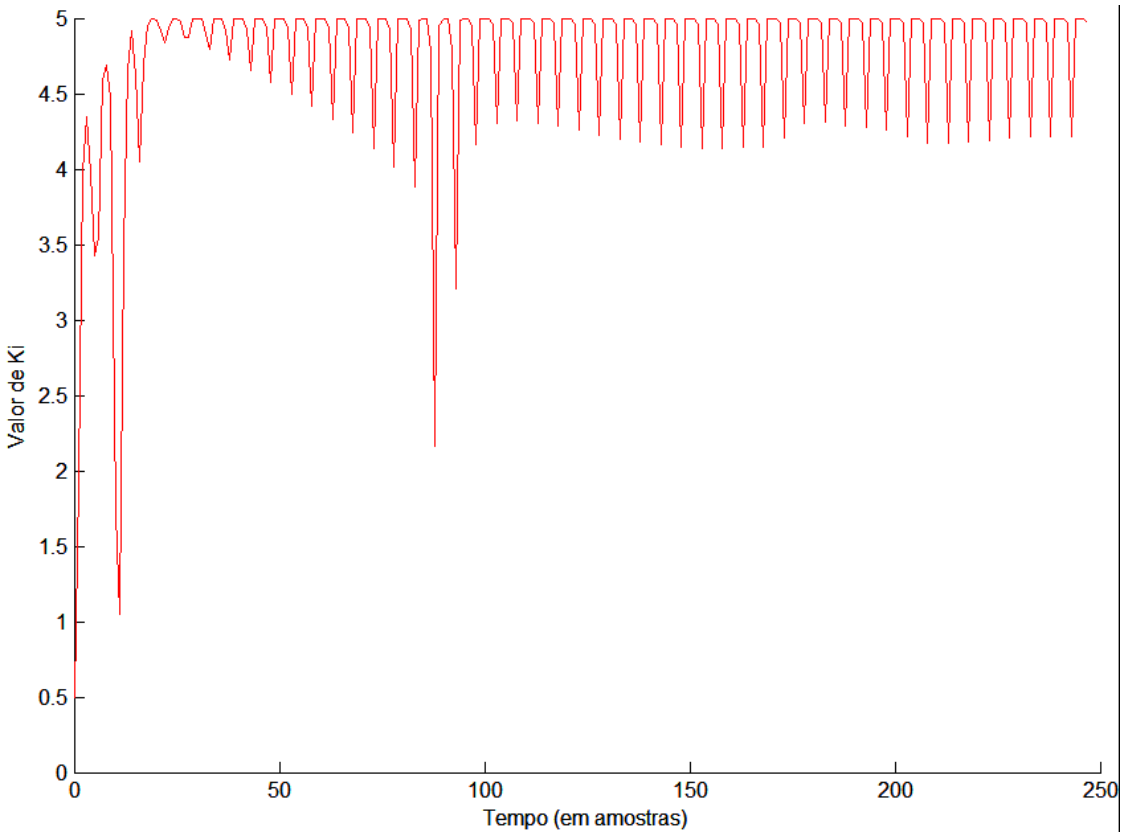


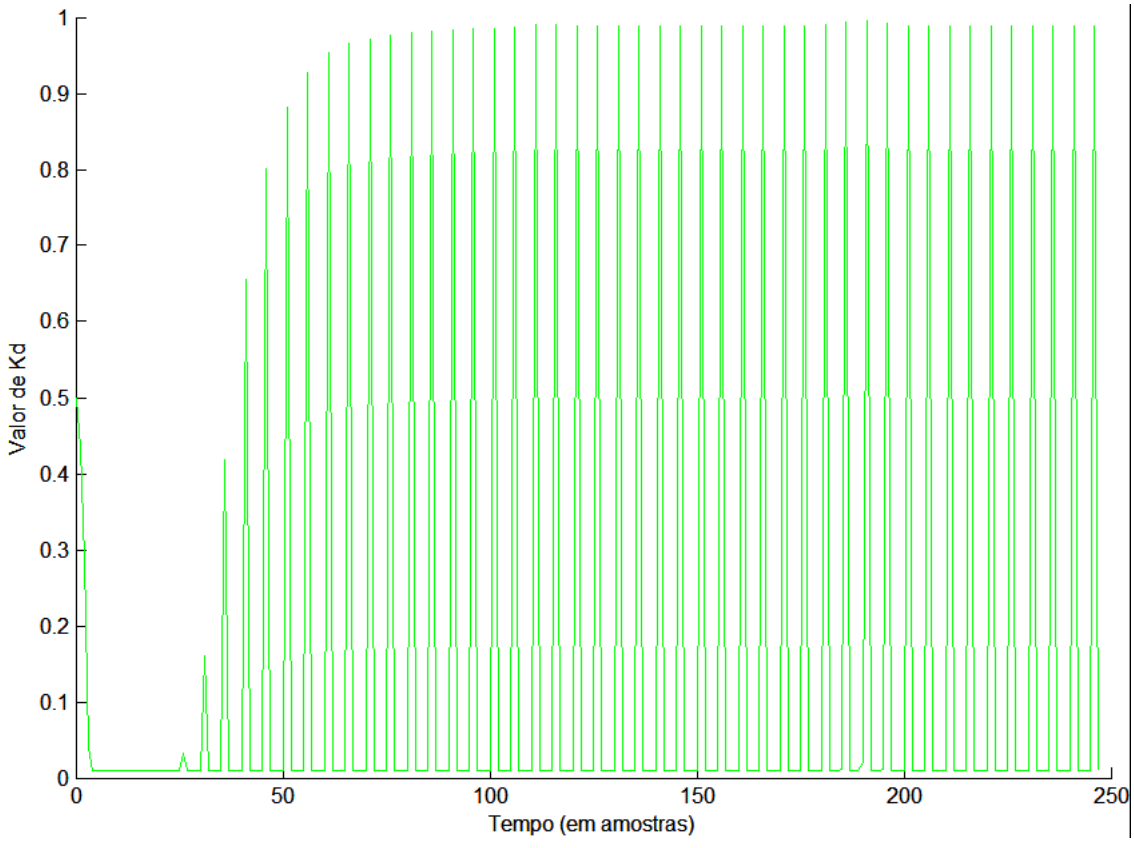


Secção 3 – Gráficos da variação dos ganhos para o sistema com ruído

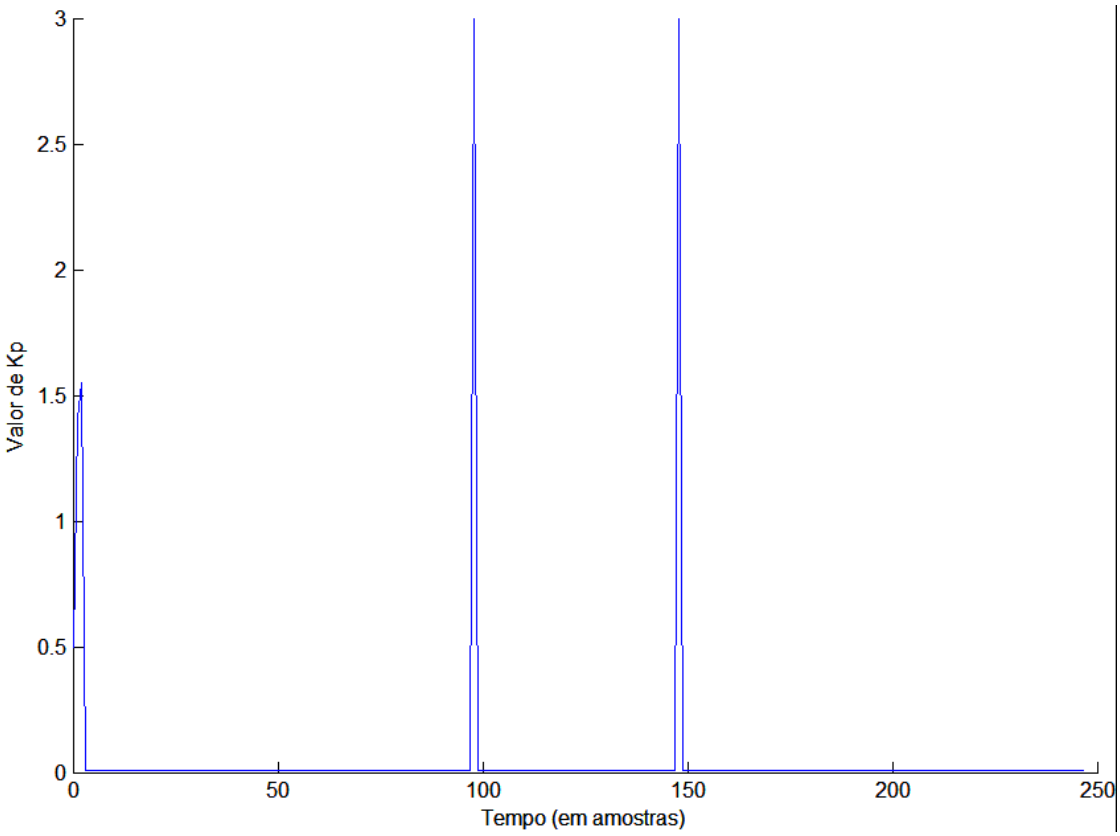
Secção 3.1 – Entrada sinusoidal

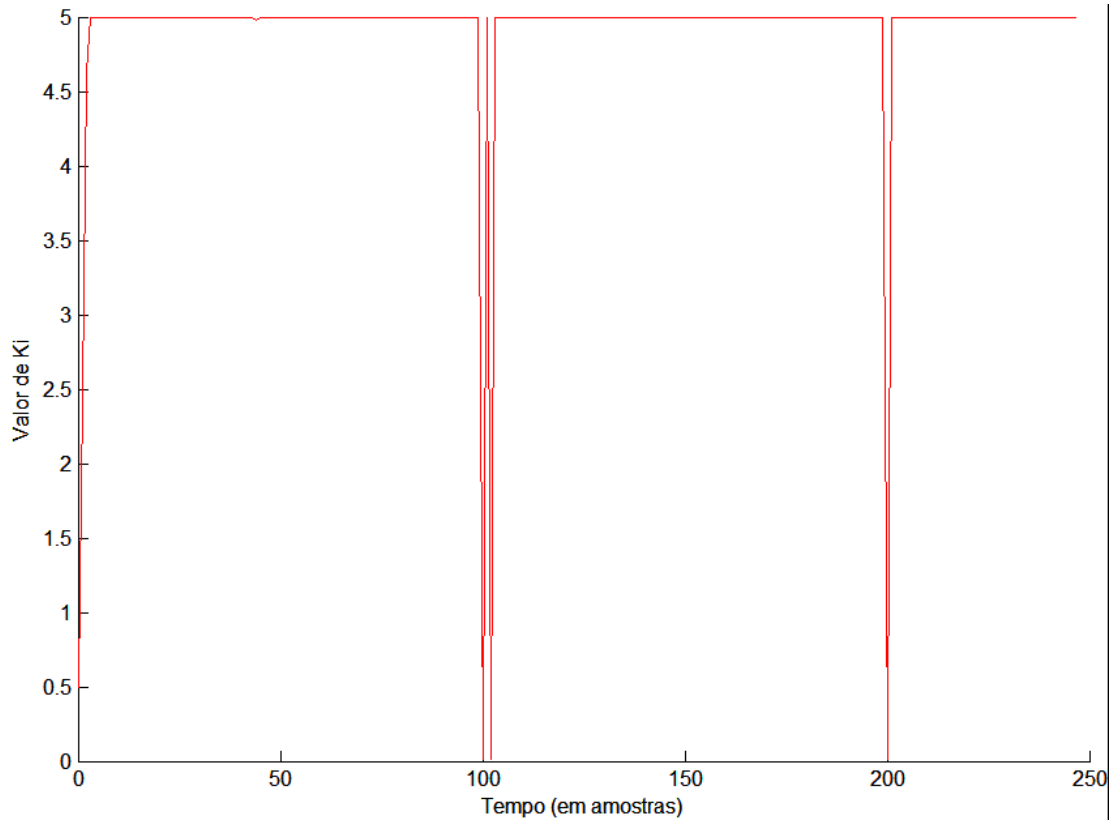


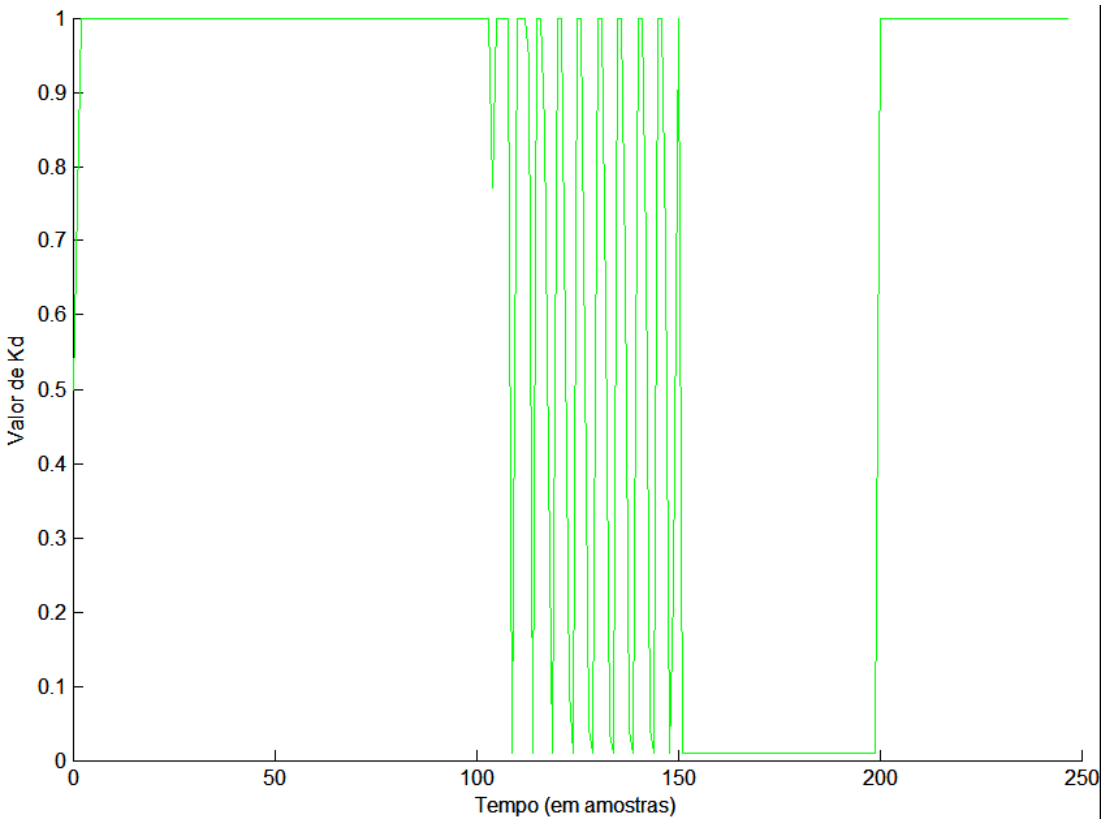

Valor de Ki
Tempo (em amostras)
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
0
50
100
150
200
250


Valor de Kd
Tempo (em amostras)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
50
100
150
200
250

Secção 3.2 – Entrada Quadrada






Valor de Kd
Tempo (em amostras)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
50
100
150
200
250