Revista de Ciências da Computação, 2019, n.º 14

Uma breve viagem pela Inteligência Artificial

Jorge Soares jorgemfsoares@gmail.com
Companhia IBM Portuguesa

Resumo

O presente artigo apresenta uma viagem rápida pela inteligência artificial, começando por um lado mais generalista e entrando depois genericamente nas principais ferramentas ao dispor dos humanos para treinar a inteligência artificial. Por fim, metade do texto será dedicado a explorar os fundamentos básicos de uma rede neuronal, por ser a abordagem que está mais em voga.

Palavras-chave: inteligência artificial, aprendizagem automática, redes neuronais

Title: A brief trip through Artificial Intelligence

Abstract

The present article presents a rapid journey through artificial intelligence, starting from a more generalist side and then entering generically the main tools available to humans to train artificial intelligence. Finally, half of the text will be devoted to exploring the basics of a neural network, as it is the approach that is most in vogue.

Keywords: artificial intelligence, machine learning, neural networks

1. Introdução

Hoje é comum ouvirmos falar de inteligência artificial através dos diversos meios de comunicação, e de alguma forma existe uma dicotomia entre o que o nosso imaginário preenche em relação à inteligência artificial e o que de facto temos ao dispor para nos auxiliar no nosso mundo. Ainda existem várias diferenças significativas entre, se assim o quisermos designar, as máquinas de carbono que são os humanos e as máquinas de silício que são os computadores. A consciência será talvez a que maior expressão tem. Tal não significa, contudo, que a inteligência artificial não esteja aí já a auxiliar os humanos em muitas tarefas.

Este artigo parte do tema mais geral de inteligência artificial, para particularizar na aprendizagem automática, particularizando de seguida nas redes neuronais.

Este artigo apresenta a seguinte estrutura, podendo ser ilustrada pela Figura 1. Na secção 2 discute-se a inteligência artificial e temas como a 'artificial narrow intelligence' e a 'artificial general intelligence'. Na secção 3 introduz-se a aprendizagem automática em temas como *supervised learning, unsupervised learning, semi-supervised learning, reinforcement learning* e *deep learning*. Na secção 4 são apresentadas as redes neuronais e na secção 5 é apresentado um exemplo detalhado do treino de uma rede neuronal. Finalmente na secção 6 são apresentadas as conclusões.

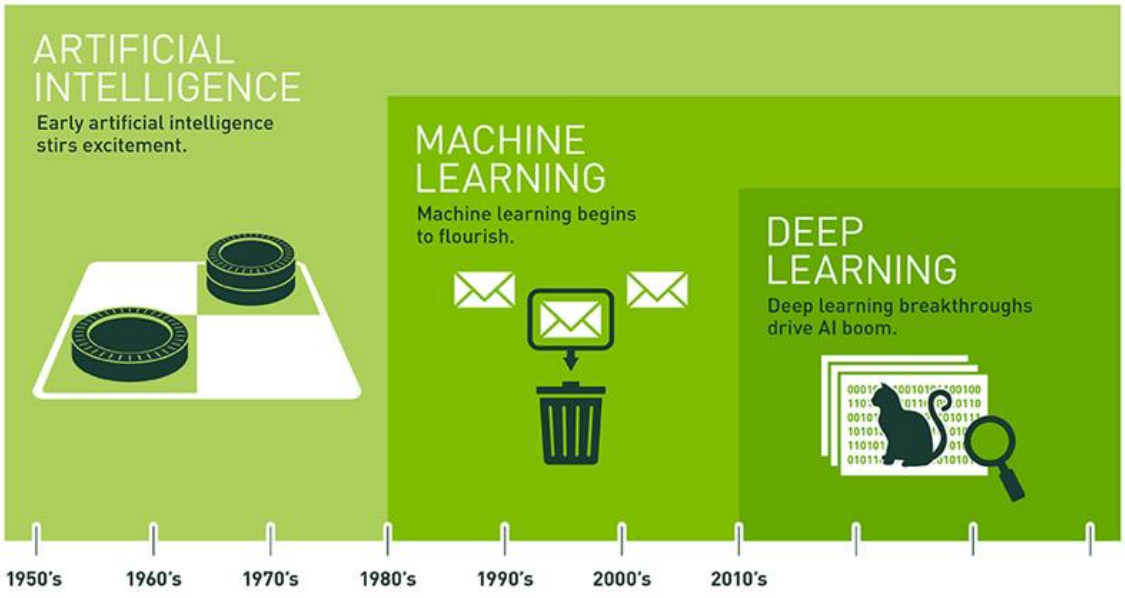

Figura 1. *Inteligência Artificial, Machine Learning e Deep Learning*

2. Inteligência Artificial

A Inteligência Artificial (ou AI) tem vindo a entrar de uma forma consistente nas nossas vidas, das mais variadas formas, por via tecnológica, escrita, cinema, televisão, ou rádio, para mencionar algumas.

Na generalidade, o nosso imaginário é preenchido por muitas destas descrições ou manifestações idealizadas de inteligência artificial, contrastando com as evidências tecnológicas atuais do estado da arte dessa inteligência.

Não se pretende de todo definir aqui o que é inteligência, no entanto entre a inteligência artificial e a humana, existem várias diferenças significativas aos dias de hoje, sendo a consciência [Koch 2004] talvez a que maior expressão tem. De facto, o reino animal, à exceção dos invertebrados, é dotado de consciência para além de inteligência, consciência do mundo em redor, da sua própria existência, das ações e consequências, não necessariamente na mesma forma de profundidade em todo o ecossistema.

Uma lagosta colocada numa panela de água a ferver tem consciência do que lhe está a acontecer? Pelo facto de não se queixar ou manifestar o desagrado, tal não significa que não tenha consciência. Já as máquinas de silício, os computadores ou robots se assim o quisermos ver, não têm consciência nem tão pouco inteligência ao nível humano, ao nosso nível de máquinas de carbono. Que se saiba ainda não existem algoritmos para consciência embora se possa dizer que existem para inteligência artificial.

Não tendo consciência, afinal o que é a inteligência artificial? Segundo Russell e Norvig [2009], "A inteligência artificial ou AI, abrange um conjunto alargado de áreas que vão do geral (aprendizagem e perceção) ao específico, como jogar xadrez, provar teoremas matemáticos, escrever poesia, dirigir um carro numa rua movimentada e diagnosticar doenças. A AI é relevante para qualquer tarefa intelectual e é verdadeiramente um campo universal."

Há também aqui na AI, várias definições, mas pode dizer-se que seja qual for a definição, a inteligência artificial se divide em duas grandes aproximações ou categorias: AI estreita (também conhecida como ANI ou 'Artificial Narrow Intelligence') e AI genérica (também conhecida como AGI ou 'Artificial General Intelligence').

Hoje em dia, e só por curiosidade, também existe quem defenda uma nova categorização, a ASI ('Artificial Super Intelligence') [Bostrom 2016], termo utilizado para quando a AI não conseguindo simular ou ter um comportamento próximo da inteligência humana, consegue, no entanto, superá-la.

Voltando às duas designações mais comuns, a AGI e a 'narrow' AI, a AGI tem basicamente a ver com a possibilidade do sistema ou máquina inteligente ter a capacidade de executar qualquer atividade intelectual que um ser humano seja capaz de executar. Muita investigação tem vindo a ser feita no domínio da AGI no sentido de copiar o mais possível o funcionamento inteligente do ser humano, sendo esta a aproximação que mais perto está da ficção científica e, diga-se, esta realidade ainda não foi atingida pelo que se pode considerar que a AGI ainda não existe. Há quem defenda que estamos à distância de 30-40 anos de ter AGI, mas o facto é que provavelmente não sabemos de todo se são 30-40 ou 100 anos ou mais ou menos.

Por outro lado, nesta aproximação de AGI, onde a investigação se tem focalizado na descoberta de novos algoritmos muito ainda pela aproximação do funcionamento do ser humano, tentando capturar em algoritmos alguns aspetos distintivos, coloca-se também a questão de virmos a ter diversos algoritmos especializados ou será que seremos capazes de descobrir um algoritmo mestre [Domingos 2015] tal como defende o Prof. Pedro Domingos.

O algoritmo mestre não tem de ser único como se fosse uma fórmula de deus, o que levantaria certamente outros temas mais filosóficos, mas pode existir um algoritmo mestre por cada uma das grandes "escolas" de AI, como por exemplo as redes neuronais (*neural networks*), ou as representações simbólicas tão em uso faz uns anos. As construções das

arquiteturas mais complexas de AI não têm de ser apoiadas numa única abordagem, mas podem ser uma combinação de várias abordagens como por exemplo ter módulos com redes neuronais e outros com redes Bayesianas (*Bayesian Networks*).

Ainda, atingir a AGI pode significar ter atingido a singularidade [Oliveira 2018] tão debatida hoje em dia e também abordada pelo Prof. Arlindo Oliveira, em que a AI será hipoteticamente capaz de superar o ser humano, o que tem colocado algum alarmismo em relação a uma hipotética ameaça de fim da raça humana tal qual a conhecemos. Mais uma vez o horizonte temporal a que estamos desta hipotética realidade, não se sabe bem, nem tão pouco se realmente a raça humana fica ostracizada face a uma superinteligência. São discussões atuais e interessantes, mas fora do âmbito deste texto.

Deveremos também ter em consideração que na generalidade a evolução da AI se tem apoiado no funcionamento do cérebro do ser humano, resultando que os modelos matemáticos, ou algoritmos construídos, acabam por ser representações aproximadas da forma como os seres humanos funcionam. Será que esta é a aproximação que nos levará a ter a AGI e eventualmente a singularidade ou a inteligência artificial poderá vir a beneficiar de uma aproximação diferente?

Tomando por comparação a ação de voar, não foi preciso copiar o batimento das asas das aves para termos criado o avião. O objetivo foi cumprido sem ter copiado o reino animal. Possivelmente a AI também não terá de copiar o funcionamento do cérebro humano. E depois como se copia a consciência? Apesar de toda a investigação nesse sentido, a realidade é que ainda não sabemos muito sobre o funcionamento do cérebro humano.

Por outro lado, como vai existir esta AGI? Na *cloud* ao género da Skynet [The Terminator 1984], em máquinas robóticas, seres biónicos ou outros como Tegmark [2018]? Qualquer das aproximações levanta um conjunto de temas éticos e sociais bem como de convivência com os seres humanos, que não são de desprezar, mas assumindo que estamos a falar do futuro, deixemos para já a AGI, ou de outra forma mais simplista, coloquemos de lado as visões cinematográficas da inteligência artificial e o futuro potencial de AGI.

O que nos resta então, hoje em dia? Afinal existe AI ou não? Sim, existe e trata-se de 'narrow' AI, ou AI específica. Esta, ao contrário da genérica, acaba por ser uma AI especializada em domínios, sendo os principais, ou com maior expressão atualmente, o processamento de linguagem (ou NLP – 'Natural Language Processing'), o reconhecimento de voz ou a verbalização de texto, e também o reconhecimento de imagens.

A 'narrow' AI pode não ter a abrangência de uma AGI, mas tem vindo a instalar-se e a fazer parte das nossas vidas, desde há alguns anos, ajudando-nos tal como outras ferramentas tecnológicas o têm feito, no fundo, convidando e auxiliando os humanos. Um exemplo antigo disso, é a utilização de caracteres manuscritos, ou ICR ('Intelligent Character Recognition') que utilizam redes neuronais como tecnologia. Outro exemplo disso são os motores de recomendação que hoje em dia nos bombardeiam com sugestões baseadas nas nossas pesquisas e em escolhas de outros humanos que fizeram pesquisas

semelhantes e se decidiram por este ou aquele item. Ainda outro exemplo são as Siris e Alexas desta vida com as quais interagimos por voz. Os 'bots' (*robots* de *software*) que começam a aparecer por todo o lado e com os quais interagimos teclando ou por voz, dependendo da finalidade e âmbito da interação.

De facto, a AI deve ser vista como existindo para nos ajudar e não para nos substituir. Claro que ainda assim existe muito debate em redor das ameaças da AI, debates válidos e necessários à convivência do ser humano com a AI. Há, portanto, e claramente, necessidade de regular, há necessidade de levar em consideração temas éticos e sociais, analisar os impactos, dotar o ser humano da educação adequada. Mas tudo isso também faz parte do processo de evolução e não porque estamos perante uma ameaça.

3. Machine Learning

Há que ir ajustando o nosso conhecimento e aprendizagem à utilização da AI, tal como o temos feito ao longo dos anos com a introdução de outras diversas tecnologias. Não nascemos ensinados para utilizar computadores, mas aprendemos, não nascemos ensinados para conduzir um carro, mas aprendemos, e assim sucessivamente com tantos aspetos da nossa vida quotidiana.

Com a AI, e daqui em diante referimo-nos tão somente à *narrow* AI, acontece o mesmo, ela tem de aprender sendo ensinada ou aprender por si, como veremos adiante. Esse acontecimento designa-se por ML ou Machine Learning e é o estado atual nos domínios já referidos de linguagem, voz e imagem. Por isso se diz simplisticamente, ser o ML o que existe como AI hoje em dia.

O que se pode fazer com a AI depende tão somente da nossa imaginação, e fruto disso ela tem estado a ser incorporada, com maior ou menor grau de profundidade ou expressão mediática, em diversas indústrias desde o consumo à área da saúde, passando pelos assistentes pessoais, ou pelos BOTs (que são no fundo robots que automatizam um diálogo com o ser humano num determinado contexto). A AI pode ser dotada de mecanismos de aprendizagem continua, haja capacidade de cálculo, mas isso pode ser arriscado já que a AI pode ficar tendenciosa (ou *biased* como se costuma designar) se não for feita a curadoria da informação que lhe é fornecida. Tal já ocorreu faz cerca de 3 anos, com um BOT que se tornou racista.

Está de facto na ordem do dia como se costuma dizer, a AI, e não é de estranhar que todas as indústrias ou ramos de negócio queiram ter artigos ou soluções inteligentes como muitas vezes ouvimos serem referidos na comunicação social.

De uma forma simplista pode-se hoje consumir AI usando APIs (Application Program Interface) disponíveis na *cloud*, como a da IBM, da Google, da Microsoft, da Amazon, para mencionar as principais. Essas APIs estão disponíveis nos domínios já mencionados de voz, imagem e processamento de linguagem, sendo previamente treinadas pelos

fornecedores do serviço, e complementadas em treino para o uso especifico que se pretende, por exemplo, se quisermos criar um classificador de imagens para reconhecer danos num automóvel, podemos recorrer a estes serviços enviando-lhes imagens de danos em automóveis, para treino, e depois usar a API treinada no nosso caso de uso em benefício da solução que se decida implementar, facilitando a atividade humana. Já por exemplo em processamento de linguagem podemos também classificar um conjunto de termos e relações específicas de um determinado domínio linguístico, como por exemplo linguagem jurídica, mais uma vez de forma a podermos usufruir desta ajuda adicional. De facto aqui no domínio linguístico, quantos de nós têm tempo de ler tudo o que aparece hoje numa determinada área do nosso interesse. É humanamente impossível sem um auxiliar destes, ou um auxiliar, se quisermos chamar-lhe assim, cognitivo.

Mas voltemos ao ML ou *machine learning*. O que é o ML afinal? Mais do que uma ciência é uma arte, desde a preparação dos dados à escolha dos algoritmos, passando pelos ajustes necessários até obter eventualmente os resultados esperados, ou seja, que a AI possa realizar a ajuda para a qual foi idealizada, com grau de confiança tão perto quanto possível do ser humano realizando a mesma tarefa.

O paradigma é diferente da programação tradicional. Em ML fornecemos os dados e deixamos que os algoritmos aprendam ou se ajustem de forma a obter os resultados esperados ou inteligentes em futuras situações para as quais não tenham sido ensinados. No fundo a AI face a novo conjunto de dados que nunca tenha visto, consegue inferir um potencial resultado. Não precisamos de saber a relação dos dados com o resultado, sendo essa a função do ML. Vamos é afinando um conjunto de parâmetros tendo em vista melhorar a fiabilidade dos resultados. Já na programação tradicional, sabemos os dados, sabemos os resultados, sabemos a relação entre os dados, e construímos essa relação programaticamente de forma a ter os resultados esperados de uma forma mais rápida e fiável. Mas se as regras mudarem, teremos de recodificar o programa. Já no ML é esperado que ele infira essa mudança de regras ao ser novamente treinado com novos dados e resultados esperados em função desses dados.

Em poucas palavras, ML é um ramo da Inteligência Artificial focalizado no desenho e desenvolvimento de algoritmos que permitam aos computadores desenvolver determinados comportamentos baseados em dados empíricos.

E existem hoje dezenas de milhares de algoritmos de *machine learning*, surgindo centenas deles a cada ano. Mas basicamente e de uma forma conceptual, cada algoritmo pode dizer-se que é composto por 3 elementos: Representação, Avaliação e Otimização.

E existem centenas de algoritmos a aparecer a cada novo ano seja pelo meio académico, seja pelas empresas de tecnologia, e a sua aplicabilidade é variada, dependendo muito do tipo de dados que temos em mão para servir de ensinamento à AI.

Quando se fala em representação, estamos a falar dos modelos ou aproximações de ML que é possível utilizar face a um determinado conjunto de dados, e desses modelos, os mais

conhecidos, não pretendendo aqui defini-los todos, são as árvores de decisão (Decision Trees), os modelos gráficos (Bayes/Markov Nets), as redes neuronais (Neural Networks), as SVMs (Support Vector Machines), as CNNs (Convolutional Neural Network) e as RNNs (Recurrent Neural Network).

Cada um destes modelos, no seu treino incorporará a componente de avaliação, ou seja, uma forma de avaliar a performance do modelo, quão distante ou perto está do resultado esperado (por exemplo no reconhecimento de imagem o erro que está a ser obtido é aceitável ou não?). E aqui entra mais um conjunto de técnicas, ou algoritmos, que se passam a enumerar sem as descrever, como sejam, Accuracy, Squared Error, Cost / Utility, Margin, Entropy, K-L Divergence, etc.

E por fim, também existem um conjunto de técnicas ou algoritmos para otimizar a performance dos modelos, sendo as mais utilizadas ou conhecidas, Combinatorial Optimization (e.g.: Linear Programming, Greedy Search), Convex Optimization (e.g.: Gradient Descent) e Constrained Optimization.

Definido o modelo a seguir, é preciso treiná-lo, sendo que este processo de treino pode ser executado de diversas formas, a saber, *Supervised Learning*, *Unsupervised Learning*, *Semi-supervised Learning*, *Reinforcement Learning* e *Deep Learning*.

Na utilização de *Supervised Learning*, o conceito é o mesmo de aprendizagem assistida na escola, em que somos guiados na aprendizagem. Aqui fornecemos os dados de treino ou aprendizagem e também o resultado desejado, isto é, fornecem-se ao modelo (algoritmo) os dados já previamente classificados, por exemplo, se pretendermos treinar uma rede neuronal para reconhecer peras, prepara-se um conjunto de dados (data set) com fotografias de peras e com fotografias de outros frutos como maças, para que o algoritmo possa "aprender" a classificar peras, apresentado na Figura 2. No fundo funciona como aprendemos de forma assistida na escola.

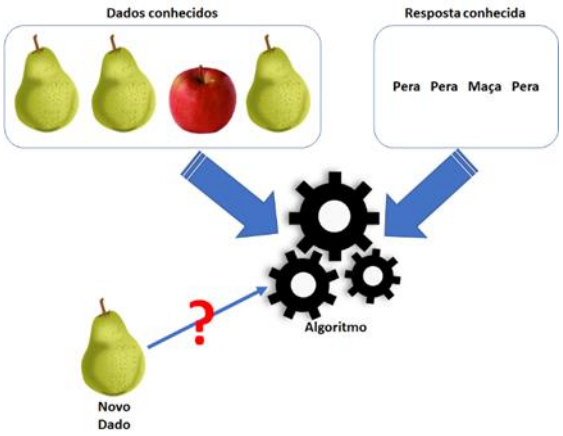

Figura 2. *Supervised Learning*

Quem faz a curadoria dos dados, e os classifica, tem também a responsabilidade em não tornar o modelo de AI tendencioso, ou como se costuma designar, *biased*, isto é, reconhecer só peras amarelas e não as verdes. Muitas vezes os dados a que temos acesso já estejam pré-classificados, como é o caso de imagens obtidas na internet já com meta dados associados, referenciando tratar-se de, por exemplo, um gato, um cão, o que for, e que alguém previamente classificou.

Já na abordagem de *Unsupervised Learning*, ver Figura 3, os dados de treino não incluem o resultado desejado, isto é, fornecem-se um conjunto de dados ao modelo (algoritmo) e este deverá ser capaz de inferir as relações potenciais entre os dados, já que os dados não estão previamente classificados.

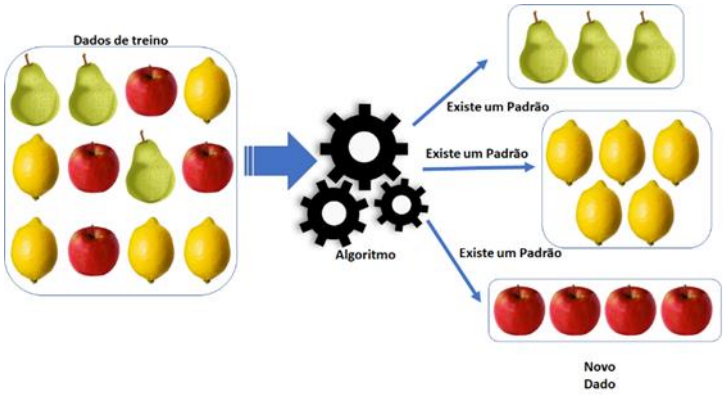

Figura 3. *Unsupervised Learning*

É como se aprendêssemos sozinhos, sem professor, isto é, se por exemplo fornecêssemos fotografias de gatos, cães e patos, seguindo o exemplo anterior, o algoritmo usando *unsupervised learning* seria capaz de agrupar as fotografias em classes, os cães, os gatos e os patos. Outro exemplo seria a análise de uma rede social de forma a definir grupos de amigos, e ainda outro exemplo seria uma segmentação de mercado das companhias, por localização, indústria e área de especialização.

Quanto à *Semi-supervised Learning*, ver Figura 4, os dados de treino ou aprendizagem incluem alguns dos resultados desejados. Por outras palavras, misturam-se as abordagens *supervised* e *unsupervised*, ou seja, do conjunto global de dados, uns estarão pré-classificados e outros não.

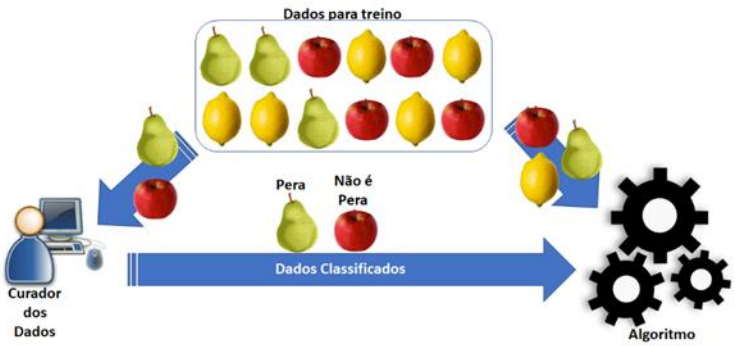

Figura 4. *Semi-supervised Learning*

Genericamente, o objetivo é o de treinar um modelo que consiga prever a partir de dados futuros com base no que aprendeu com o treino baseado em dados pré classificados. A motivação por trás da *semi-supervised learning* tem a ver com o seu valor prático no que diz respeito a aprender mais depressa e melhor. No mundo real é possível obter um alargado conjunto de dados não categorizados, como por exemplo, documentos obtidos na internet ou imagens obtidas em câmaras de vigilância, etc. Contudo as classificações correspondentes, como deteção de intrusos, requerem uma tarefa humana de anotação, que é em si um processo lento.

Já no *reinforcement learning* existem recompensas na sequência de ações, como que um prémio por ter atingido uma determinada meta, e um castigo por não ter atingido. No mundo humano e animal alguma aprendizagem pode ser feita dessa forma, poder-se-ia até dizer idealisticamente que será uma aprendizagem pavloviana. Assim, de uma forma geral, o objetivo do *reinforcement learning* (RL) é aprender como fazer corresponder um conjunto de observações e medições, com um conjunto de ações, enquanto tenta maximizar uma recompensa de longo prazo. Imaginemos uma criança a quem damos um *tablet*. Primeiro, a criança vai observar e construir a sua própria representação da interação com o *tablet.* Depois, a curiosidade da criança vai levá-la a tomar um conjunto de ações como ligar o *tablet* e carregar no ecrã, observando como reage o *tablet*. Se o *tablet* não reagir ou não mostrar o que a criança poderá ter interesse, ela não vai gostar o que a levará a tomar menos ações em relação ao *tablet*, ao contrário de conseguir obter o resultado idealizado, o que a levará a explorar mais. A criança poderá repetir este processo até encontrar um resultado que a satisfaça. Uma área de aplicação de *reinforcement learning* é o mundo da robótica, representado na Figura 5.

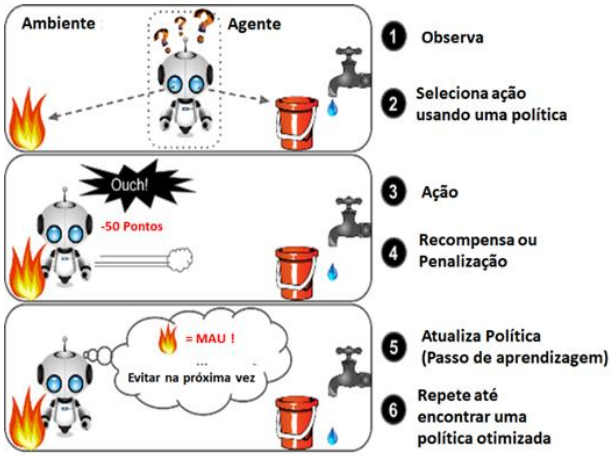

Figura 5. *Reinforcement learning*

Por fim, a abordagem de D*eep learning*, ver Figura 6, que podendo ir para além das redes neuronais, é aí de facto que tem maior expressão. Trata-se de uma técnica de *machine learning* que permite ensinar os algoritmos a fazer o que é natural nos humanos: aprender pelo exemplo. Algumas técnicas anteriores já descritas permitem fazer o mesmo, no entanto em *deep learning*, as redes neuronais que a suportam podem ter dezenas de níveis com milhares de nós ou "neurónios", daí o nome 'deep', de profundidade de relações neuronais, mais uma vez à semelhança dos humanos.

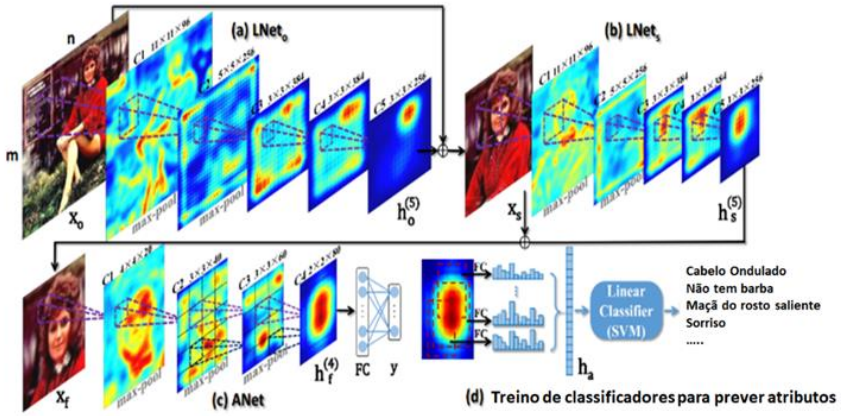

Figura 6. *Deep learning*

E é isso que permite hoje em dia associar *deep learning* a, por exemplo, carros de condução autónoma permitindo-lhes reconhecer um sinal de stop, distinguir um pedestre de um semáforo, ou ainda o som de uma ambulância. Para além de reconhecimento de imagem, é também utilizada em reconhecimento de voz e tem aparecido embebida em telemóveis, *tablets* e televisões, por exemplo. O *deep learning* tem atraído as atenções nos últimos anos, e por uma boa razão, a de finalmente estar a atingir resultados que não foi possível atingir antes.

E existem dois grandes fatores para estes resultados estarem agora a começar a aparecer, por um lado a capacidade de processamento tem aumentado bastante (lei de Moore) e por outro, os dados digitais, necessários ao treino dos algoritmos, têm aumentado exponencialmente, e também importante, grande parte deles com classificação associada, isto é, não só temos fotografias de gatos e cães, como elas estão classificadas como sendo de gatos e cães.

E aqui, nós os humanos, ou as máquinas de carbono, temos vindo a digitalizar o mundo e a classificá-lo, para que as máquinas de silício possam "aprender" e ajudar-nos.

Mas voltando ao *deep learning* e à sua utilização com redes neuronais, existem hoje duas grandes abordagens em uso: as CNN ou ConvNets (*Convolutional Neural Networks*) e as RNN (*Recurrent Neural Network*), para as quais existe também muita informação explicativa publicada na internet, e que se aconselha a procura. Sendo as CNN e as RNN apoiadas em redes neuronais, opta-se por explicar aqui os fundamentos base das redes neuronais, que são um modelo de *machine learning*.

4. Rede Neuronal

Uma rede neuronal, representada na Figura 7, não é mais do que uma versão cartoonista do funcionamento elétrico do neurónio humano, inspirou-se neste, e nas suas ligações com neurónios adjacentes, tal como podemos ver na figura, onde se representa para efeitos de ilustração o neurónio humano (*cell body*) com as suas dendrites e axónios. Na zona inferior da figura pode ver-se o modelo de neurónio da rede neuronal que pretendemos analisar daqui em diante. Podemos ver aí o corpo da célula (círculo) bem como os sinais de entrada (in_1 a in_n) e ainda o sinal de saída depois de ter sido processado pelo neurónio (out), que passamos a designar por nó.

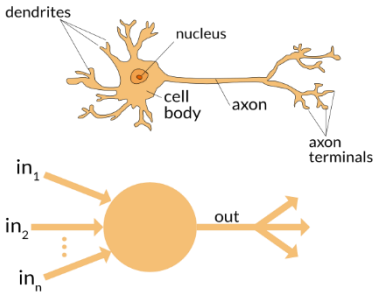

Figura 7. Rede neuronal

Olhando para o funcionamento deste nó em mais detalhe, de acordo com a Figura 7 que segue, os sinais de entrada (representados na figura por 1 a X_m), podem aí chegar com maior ou menor constrangimento, pelo que se introduz a noção de peso da ligação (W_1 a W_m). Isto é, dado um determinado sinal X_m ele será multiplicado por um determinado peso W_m sendo o sinal fortalecido ou desvanecido conforme o valor de W, à chegada ao nó. Já

dentro do nó, todos estes sinais amplificados ou diminuídos, são somados, isto é matematicamente teremos $1*W_0 + X_1*W_1 + X_2*W_2 + \ldots X_m*W_m$, o que é representado na figura pelo sinal de matemático de somatório ($\sum$). Por fim há que decidir qual será o sinal de saída e para isso convencionou-se na literatura da matéria designar e usar uma função de ativação para esse efeito, ver Figura 8.

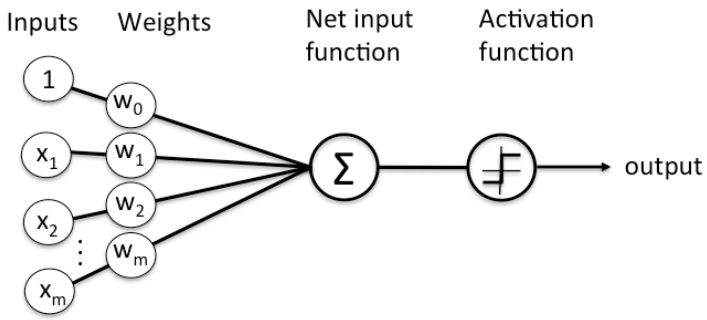

Figura 8. Função de ativação

A função de ativação tem também uma expressão matemática e é comum usar por exemplo uma função sigmoide como função de ativação, bem como uma função ReLu, que se representam graficamente na Figura 9.

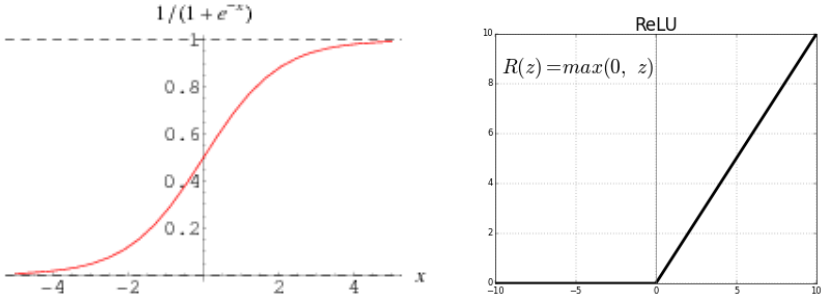

Figura 9. Função sigmoide e função ReLu

O objetivo da função de ativação é o de decidir com que intensidade o sinal sai de um determinado nó para o nó seguinte, e como se pode ver nas representações anteriores, se por exemplo um sinal chegar à função sigmoide de ativação com um valor de -4 o seu valor de saída será próximo de 0, já se chegar com um valor de 4 o seu valor de saída será próximo de 1. Por outro lado, quando se utiliza a função ReLu como ativação, se o sinal for de valor -4 o seu valor de saída será 0 e se for 4 o seu valor de saída será 4. De facto, na função ReLu, como se vê, qualquer valor negativo terá como resultado 0, e qualquer valor positivo terá de resultado o seu próprio valor. Uma ou outra aproximação são hoje utilizadas separadamente ou em conjunto, sendo que a função ReLu tem um custo

computacional menor, o que poderá ser uma vantagem para redes neuronais com alguma profundidade ou níveis de processamento.

Definido o modelo de funcionamento do nó unitariamente, ele fará parte de uma rede neuronal em número suficiente para poder ser treinado eficazmente. A rede neuronal pode ter diversos níveis, e o número de nós em cada nível depende essencialmente do tipo de dados que é necessário processar bem como dos algoritmos escolhidos. Os níveis que definem a rede neuronal são essencialmente de 3 tipos, o nível de entrada (ou *input layer*), o nível de saída (*output layer*) e os níveis que existirem entre estes dois, em número de um ou mais, convencionaram chamar-se de *hidden layers*.

Um exemplo de uma rede neuronal com 4 nós de entrada e 3 de saída, ver Figura 9, contendo dois *hidden layers* com respetivamente 5 e 7 nós, é a que se mostra na imagem adjacente. Pode verificar-se que o número de nós nos níveis de entrada e saída não é igual, sendo tipicamente inferior no nível de saída e dependente do que se pretende que a rede "aprenda". Já o nível de entrada está dependente do tipo de dados com que queremos treinar a rede e com ela obter as previsões pretendidas.

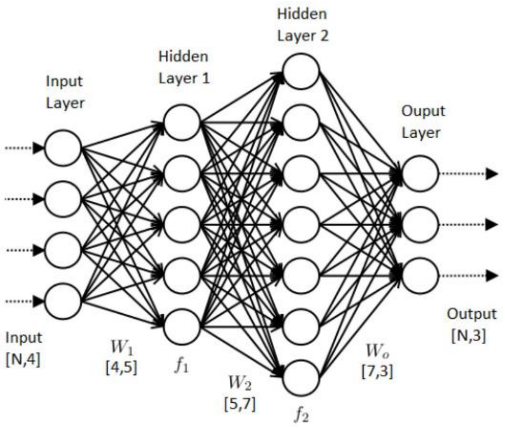

Figura 10. Rede neuronal com 4 nós de entrada e 3 de saída

Também se pode ver na Figura 10 que as ligações neuronais entre os níveis da rede têm associado um peso W_m, já descrito anteriormente.

Quanto à dependência do número de nós no primeiro nível, do tipo de dados de treino, tomemos por exemplo que queríamos treinar uma rede neuronal para reconhecer fotografias de gatos e que cada fotografia é quadrada e com uma resolução de 300 pixels ou pontos. Isso significa que a fotografia, que é bidimensional, tem 90.000 (300 x 300) pontos no total. Assuma-se também que a fotografia já está em formato RGB (*Red*, *Green*, *Blue*) ou seja, que a cor de cada ponto da fotografia pode ser representada pelas suas componentes cromáticas base, vermelho, verde e azul. Isso significa então que cada ponto é composto por 3 elementos e daí a dimensionalidade total do dado fotografia para alimentar uma rede neuronal será de 3x90.000, ou seja 270.000 pontos.

Concluindo, e neste exemplo, o número de nós no nível de entrada da rede seria de 270.000 nós. Ainda neste exemplo, para o nível de saída bastaria 1 nó com os valores 1 (é gato) ou 0 (não é gato). Pelo meio haveria que definir quantos *hidden layers* deveriam ser usados e com quantos nós cada um devia ser constituído.

Ainda de forma ilustrativa imaginemos que pretendíamos construir uma rede neuronal que fosse capaz de reconhecer algarismos manuscritos, onde cada imagem a preto e branco de um algarismo teria uma dimensão de 28 por 28 pontos ou *pixel*. Isso significaria que no nível de entrada da rede neuronal teríamos de ter 784 (28x28) nós. Já para o nível de saída ou resultado, e neste caso de reconhecimento de algarismos, deveríamos ter 10 nós, cada um correspondente a um algarismo, isto é, se o algarismo reconhecido fosse 4, então o quinto nó (já que normalmente se convenciona incluir o 0 como primeiro) deveria ter um valor 1, e os restantes um valor de 0. Se escolhêssemos só um *hidden layer*, então a representação gráfica desta rede poderia ser como se representa na Figura 11.

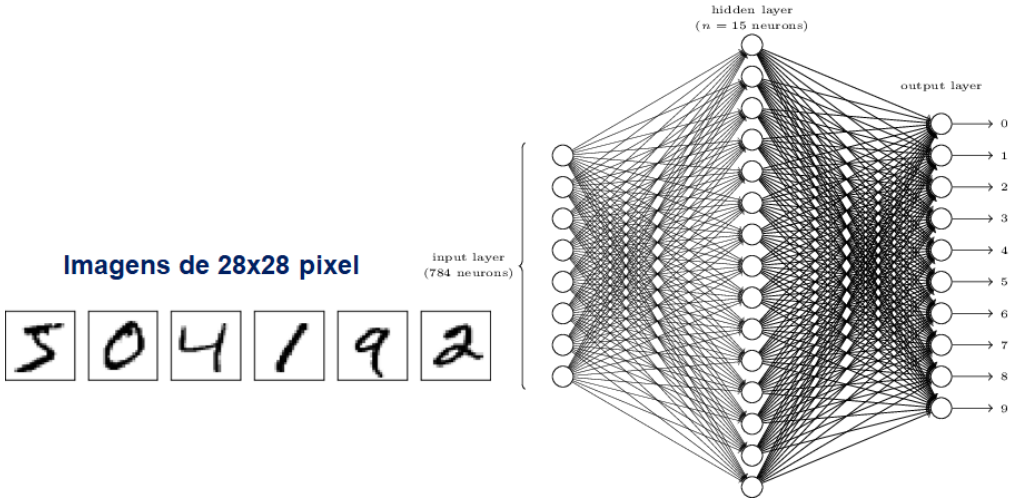

Figura 11. Rede neuronal com um só *hidden layer*

Desta forma mais detalhada vimos a representação de uma rede neuronal. Seguindo com os 3 elementos que compõem o algoritmo, representação (rede neuronal no nosso caso de exemplo), avaliação e otimização, cabe agora escolher o que vamos usar para otimização e para avaliação. Para exemplificação de conceitos, e tendo escolhido a rede neuronal, vamos agora escolher o erro quadrático médio (*mean squared error*) como avaliação e o gradiente (*gradient descent*) como otimização. Existem claro outras possibilidades, mas para exemplificação de conceitos seguimos com estas.

O erro quadrático médio enquanto mecanismo de avaliação, tem como função avaliar quão longe uma previsão de uma determinada imagem, por exemplo, está do que deveria ser, isto é, se alimentarmos a rede com a imagem de um gato e a rede fizer a previsão de um cão, essa diferença é medida pelo erro quadrático médio e será refletida no ciclo de aprendizagem da rede, como veremos adiante.

Já que se volta a falar de aprendizagem, é de referir que para a rede ser ensinada, ela terá de ser alimentada com um volume de dados significativo (por exemplo fotografias) e as respetivas classificações individuais (por exemplo cão, cão, gato, cão, gato, etc). E é a média da soma de cada diferença de cada dado apresentado à rede neuronal, que é avaliada pelo erro quadrático médio, como se pode ver na sua fórmula matemática $MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$ muitas vezes também representada por $MSE = \frac{1}{2}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$.

Nesta fórmula, n representa o número de dados existentes para treinar a rede neuronal, e i refere-se a um determinado elemento entre 1 e n. Por outro lado, y_i representa o resultado esperado para o dado i, e $\tilde{y_i}$ representa o resultado obtido para o dado i.

Como se pode depreender da fórmula, calcula-se o quadrado das diferenças entre o resultado esperado e o obtido, para todos os dados existentes, soma-se e divide-se o resultado pelo número de dados, obtendo assim o erro quadrático médio.

Esta diferença entre o resultado esperado e o obtido deverá ir sendo diminuída de forma a ter uma rede neuronal com o mínimo de erro possível. E essa é a função da otimização com gradiente (*gradient descent*), que passamos a descrever. Como já se disse existem outras abordagens, mas a título exemplificativo de conceitos, vamos usar o gradiente, que será talvez um dos mais usados em *machine learning*.

O gradiente (*gradient descent*), como já se referiu, tem como função principal minimizar uma função de custo (*cost function*), representada muitas vezes como J(x), e que mais não é que uma medida de quão errado está o modelo, e já vimos que isso será obtido no nosso exemplo pelo erro quadrático médio já apresentado.

A minimização de uma função, por outro lado, implica encontrar os chamados mínimos de uma função, que podem ser locais ou globais. Não sendo objetivo aqui entrar nos detalhes matemáticos, exemplificamos de seguida graficamente uma função J(w), ilustrando o objetivo de minimizar o erro, ou seja, atingir o mínimo desta função.

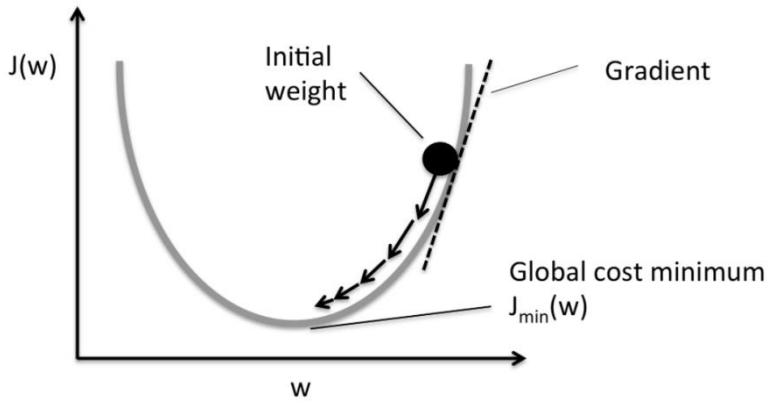

Figura 12. Otimização com gradiente (*gradient descent*)

Podemos ver na Figura 12, que na base da curva ou parábola é referenciado o mínimo $J_{min}(w)$, e que esse mínimo é atingido começando de um ponto inicial e descendo a função com maior ou menor declive. Isso são os gradientes que pretendemos. Estes gradientes ou declives são obtidos por cálculo diferencial, isto é, por $\frac{\partial J}{\partial w}$, ou seja, pela derivada da função de custo J relativamente a um valor w, e isto é feito para todos os valores w. Mais uma vez o objetivo não é entrar nos detalhes matemáticos, mas explicar o conceito por detrás.

Como se infere, o gradiente tem a ver com a descida da curva representada até atingir o valor mínimo. A velocidade com que se desce essa curva pode ser controlada, e em ML designa-se por *Learning Rate* sendo muitas vezes representada matematicamente como veremos adiante a sua inclusão nas fórmulas que compõe o treino da rede neuronal. Neste momento, já decidimos o modelo (rede neuronal), a avaliação (erro quadrático médio) e a otimização (gradiente).

Então como se ensina a rede neuronal? Basicamente apresentam-se os dados de treino com os respetivos valores a atingir, passam-se pela rede neuronal (*forward propagation*), avalia-se a diferença entre o resultado esperado e o obtido, define-se o valor de otimização com um determinado *learning rate*, propaga-se essa otimização pela rede em sentido inverso (referido como *back propagation*), e repete-se o ciclo, isto é, voltam-se a passar os dados pela rede seguindo todos os passos mencionados de *forward* e *backpropagation*, tantas vezes quantas as que forem definidas (tem a ver com a profundidade do treino) e passando todos os dados do conjunto de dados de treino. Só uma nota, não existem unicamente dados de treino, também deverão existir dados de teste para verificar se a rede está a dar o resultado esperado, e tipicamente a divisão é de 80% de dados de treino e 20% de dados de teste.

5. Treino da Rede Neuronal

Para ilustrar melhor o treino da rede neuronal, munidos dos conceitos anteriormente descritos, vamos necessitar de dados de treino. Para isso vamos utilizar um conjunto de dados simples, relativos ao naufrágio do navio Titanic, dados esses obtidos a partir do site https://www.kaggle.com/azeembootwala/titanic. Desse *dataset* vamos utilizar as colunas 4, 5, 6 e 10 relativas respetivamente ao sexo do passageiro, idade, tarifa e tamanho da família. Os dados já estão tratados e normalizados, sendo nosso objetivo treinar a rede para os resultados da coluna 3, ou seja, se o passageiro sobrevive ou não. Escolhidas então 4 variáveis de entrada, isso significa que a nossa rede neuronal terá de ter 4 nós de entrada e um nó de saída. Para exemplificação decidem-se usar 3 nós no *hidden layer*. Assim sendo, esta rede com somente quatro nós de entrada (I_1, I_2. I_3 e I_4), um nó de saída (O_1) e três nós no *hidden layer* (h_1, h_2 e h_3), pode ser representada tal como na Figura 13.

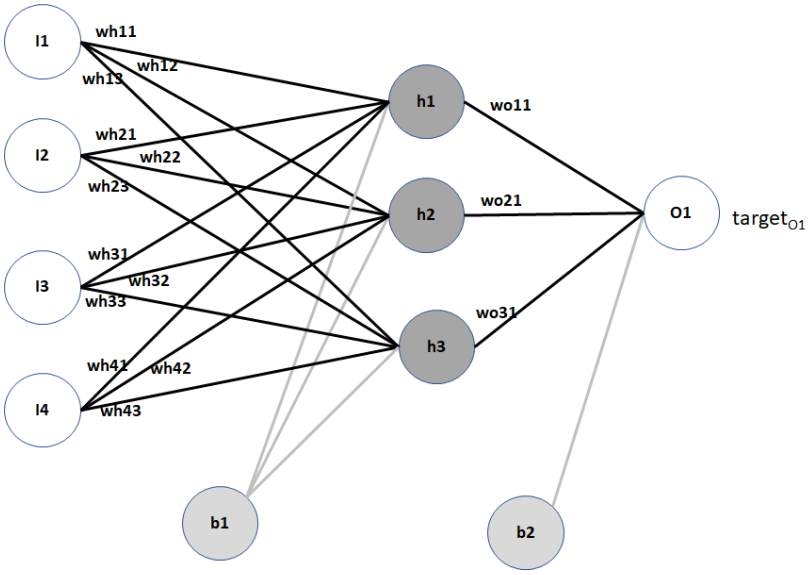

Figura 13. Rede neuronal para o exemplo de treino

Usaremos um só nó de saída, mas poderiam ser vários, como já vimos anteriormente no exemplo de reconhecimento de carateres manuscritos.

A explicação que segue teve por base um artigo encontrado na internet mas que não se conseguiu recuperar o *link* pelo que não sendo possível referenciá-lo formalmente, se queria deixar aqui no entanto, essa nota de referência.

Neste modelo apresentado, podemos ver a introdução de dois nós, b_1 e b_2, designados por *bias*, e cuja função é a de facilitar ou dificultar a passagem de sinal pelo nó, isto é, para valores de *bias* elevados, o resultado do processamento do nó tenderá a ser também elevado, e por conseguinte próximo de 1 (de acordo com a função de ativação já descrita). O valor de *bias* pode também ser negativo e neste caso o nó tenderá a dar resultados próximos de 0.
Vemos também no diagrama a presença dos pesos das ligações entre os nós de entrada e os do *hidden layer* (w_{ij}, em que i representa o nó inicial e j o nó destino, em numérico), bem como entre o *hidden layer* e os nós de saída (w_{ij}).

Também está aí representado o valor $target_{O1}$ que representa qual o valor que deve ser obtido para um conjunto de dados (I_1, I_2, I_3, I_4), isto é $target_{O1} = f(I_1, I_2, I_3, I_4)$, ou seja, f é como se fosse a função rede neuronal.

Como já se referiu anteriormente, todos estes valores mencionados são normalmente representados de forma matricial. Assim, e neste exemplo, os dados apresentados à rede serão constituídos pelo vetor [I_1, I_2, I_3, I_4], o *hidden layer* pelo vetor [h_1, h_2, h_3], e a saída tem um único nó e, portanto, um único valor O_1.

Para efeitos de demonstração consideremos um conjunto de dados de 7 exemplos já pré-classificados, obtidos das 7 primeiras linhas do dataset mencionado anteriormente, e que servirão para treinar a rede neuronal, isto é, consideramos os seguintes valores dos vetores I e O_1:

$$I = \begin{bmatrix} 1 & 0 & & & 0 & 0 & 1 & 1 & 1 & & \\ 0{,}275 & 0{,}475 & 0{,}325 & 0{,}4375 & 0{,}4375 & 0{,}35 & 0{,}675 & & & & \\ 0{,}0142 & 0{,}1391 & 0{,}0155 & 0{,}1036 & 0{,}0157 & 0{,}0165 & 0{,}1012 & & & & \\ 0{,}1 & 0{,}1 & & 0{,}0 & 0{,}1 & 0{,}0 & 0{,}0 & 0{,}0 & & & \end{bmatrix}$$

$$O_1 = [0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0]$$

Do conjunto de 7 exemplos para treino, vamos utilizar somente o segundo, ou seja, os valores que estão na segunda coluna da matriz I:

$$\begin{bmatrix} 0 \\ 0{,}475 \\ 0{,}1391 \\ 0{,}1 \end{bmatrix}$$

Escolhido o segundo exemplo, isso significa que vamos usar também o segundo valor da matriz O1, no valor de 1 como se viu anteriormente, representando assim que o resultado esperado da rede neuronal para o processamento do primeiro exemplo, é 1.

Os pesos das ligações dentro da rede neuronal W_{ij} são normalmente inicializados aleatoriamente, pelo que vamos utilizar os seguintes valores obtidos aleatoriamente, para efeitos ilustrativos da explicação que segue:

$$Ligações\ entre\ I\ e\ H = wh_{i,j} = \begin{bmatrix} 0{,}1 & 0{,}7 & 0{,}4 \\ 0{,}6 & 0{,}2 & 0{,}8 \\ 0{,}5 & 0{,}9 & 0{,}3 \\ 0{,}2 & 0{,}4 & 0{,}1 \end{bmatrix}$$

$$Ligações\ entre\ H\ e\ O = wo_{i,j} = \begin{bmatrix} 0{,}3 \\ 0{,}7 \\ 0{,}2 \end{bmatrix}$$

Colocando estes valores sobre o modelo de rede neuronal, obtemos a representação na Figura 14.

Vamos então de seguida ver como se treina a rede neuronal. Relembrando, existe uma primeira passagem pela rede, *forward propagation*, é calculado o erro (função da diferença entre o valor esperado e o obtido), depois é efetuada a *back propagation*, que leva em linha de conta as variações calculadas de otimização (com gradiente) dos pesos das ligações.

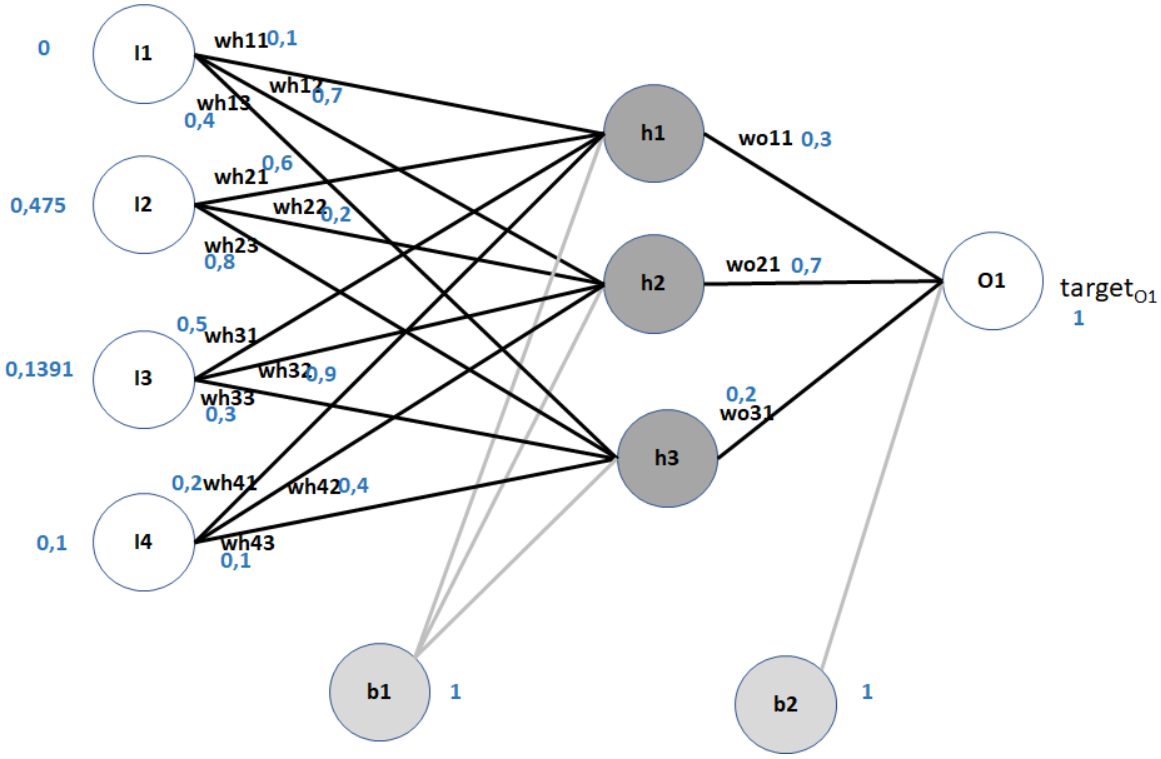

Figura 14. Rede neuronal de treino com pesos entre I e H e entre H e O

Começamos por ver que sinais chegam aos nós do *hidden layer*. Pelo que já vimos anteriormente, à entrada dos nós do *hidden layer* (que designaremos por IN_{h1}, IN_{h2} e IN_{h3}), e assumindo o valor de 1 para os nós de *bias* (b_1 e b_2), teremos os seguintes valores calculados:

$$IN_{h1} = wh_{11} * I_1 + wh_{21} * I_2 + wh_{31} * I_3 + wh_{41} * I_4 + b_1 =$$
$$= 0{,}1 * 0 + 0{,}6 * 0{,}475 + 0{,}5 * 0{,}1391 + 0{,}2 * 0{,}1 + 1 = 1{,}3746$$

$$IN_{h2} = wh_{12} * I_1 + wh_{22} * I_2 + wh_{32} * I_3 + wh_{42} * I_4 + b_1 =$$
$$= 0{,}7 * 0 + 0{,}2 * 0{,}475 + 0{,}9 * 0{,}1391 + 0{,}4 * 0{,}1 + 1 = 1{,}26019$$

$$IN_{h3} = wh_{13} * I_1 + wh_{23} * I_2 + wh_{33} * I_3 + wh_{43} * I_4 + b_1 =$$
$$= 0{,}4 * 0 + 0{,}8 * 0{,}475 + 0{,}3 * 0{,}1391 + 0{,}1 * 0{,}1 + 1 = 1{,}43173$$

Imaginado que existiam mais do que 4 nós de entrada na rede neuronal, não é prático fazer este cálculo para cada uma das ligações, pelo que se recorre a cálculo matricial, onde como

vimos se pretende calcular o produto dos pesos das ligações de entrada no nó pelos valores de sinal que passam nessas ligações, somando-lhe o valor de *bias* para esse nó do nível que está a ser processado, isto é, pretende usar-se a fórmula (IN_h representa os valores calculados à entrada dos nós do *hidden layer*):

$$IN_h = w^T * I + b$$

em que w^T é a matriz transposta de w. Executando matematicamente o cálculo matricial, temos:

$$IN_h = w^T * I + b = \begin{bmatrix} 0{,}1 & 0{,}7 & 0{,}4 \\ 0{,}6 & 0{,}2 & 0{,}8 \\ 0{,}5 & 0{,}9 & 0{,}3 \\ 0{,}2 & 0{,}4 & 0{,}1 \end{bmatrix}^T * \begin{bmatrix} 0 \\ 0{,}475 \\ 0{,}1391 \\ 0{,}1 \end{bmatrix} + 1 =$$

$$= \begin{bmatrix} 0{,}1 & 0{,}6 & 0{,}5 & 0{,}2 \\ 0{,}7 & 0{,}2 & 0{,}9 & 0{,}4 \\ 0{,}4 & 0{,}8 & 0{,}3 & 0{,}1 \end{bmatrix} * \begin{bmatrix} 0 \\ 0{,}475 \\ 0{,}1391 \\ 0{,}1 \end{bmatrix} + 1 =$$

$$= \begin{bmatrix} 0{,}1 * 0 + 0{,}6 * 0{,}475 + 0{,}5 * 0{,}1391 + 02 * 0{,}1 \\ 0{,}7 * 0 + 0{,}2 * 0{,}475 + 0{,}9 * 0{,}1391 + 0{,}4 * 0{,}1 \\ 0{,}4 * 0 + 0{,}8 * 0{,}475 + 0{,}3 * 0{,}1391 + 0{,}1 * 0{,}1 \end{bmatrix} + 1 = \begin{bmatrix} 0{,}3746 \\ 0{,}26019 \\ 0{,}43173 \end{bmatrix} + 1 = \begin{bmatrix} 1{,}3746 \\ 1{,}26019 \\ 1{,}43173 \end{bmatrix}$$

O mesmo resultado obtido com as fórmulas anteriores como seria de esperar.

Já pensando numa implementação programática, por exemplo com Python, recorreríamos à seguinte fórmula para este cálculo:

```
INh= np.dot(w.T,I)+b
```

Em que IN_h, w, I e b são matrizes, e w.T é a matriz transposta de w.

Como vimos também anteriormente, estes valores apresentados à entrada de um nó vão passar pela função de ativação, e neste caso vamos usar a função sigmoide como referido atrás:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Aplicando os valores IN_{h1}, IN_{h2} e IN_{h3} à função sigmoide vamos obter o sinal de saída dos nós do primeiro nível (*hidden layer* neste caso), e que designaremos por OUT_{h1}, OUT_{h2} e OUT_{h3}:

$$OUT_{h1}(IN_{h1}) = \frac{1}{1 + e^{-IN_{h1}}} = \frac{1}{1 + e^{-1{,}3746}} = 0{,}7981$$

$$OUT_{h2}(IN_{h2}) = \frac{1}{1+e^{-IN_{h2}}} = \frac{1}{1+e^{-1,26019}} = 0,7791$$
$$OUT_{h3}(IN_{h3}) = \frac{1}{1+e^{-IN_{h3}}} = \frac{1}{1+e^{-1,43173}} = 0,8072$$

Mais uma vez, usando cálculo matricial (mais apropriado para grande número de nós), a função sigmoide $\sigma(x) = \frac{1}{1+e^{-x}}$ pode ser aplicada a IN_h:

$$OUT_h = \sigma(IN_h) = \frac{1}{1+e^{-IN_h}} = \sigma(\begin{bmatrix}1,3746\\1,26019\\1,43173\end{bmatrix}) = \begin{bmatrix}0,7981\\0,7791\\0,8072\end{bmatrix}$$

Já programaticamente, com Python, a fórmula seria:

OUT_h = sigmoid(IN_h)

em que sigmoide(z)= 1 / (1+ np.exp(-z)) é uma função definida.

Usando a nomenclatura acima, e num único passo, poderíamos simplificar a instrução em Python para:

OUT_h = sigmoid(np.dot(w_h.T,I)+b)

Ainda em Python deveríamos definir primeiro a função sigmoid, pelo que o código resultaria então em:

```
#define função sigmoide
def sigmoid(Z):
    A = 1/(1+np.exp(-Z))
    return A

OUTh = sigmoid(np.dot(wh.T,I)+b)
```

O processo repete-se agora entre os nós do *hidden layer* e o de saída, da mesma forma e usando as mesmas designações e fórmulas, pelo que resumimos aos cálculos:

$$IN_{O1} = wo_{11} * OUT_{h1} + wo_{21} * OUT_{h2} + wo_{31} * OUT_{h3} + b_2 =$$
$$= 0,3 * 0,7981 + 0,7 * 0,7791 + 0,2 * 0,8072 + 1 = 1,9462$$
$$OUT_{O1}(IN_{O1}) = sigmoid(IN_{O1}) = 0,8750$$

Já matricialmente, teríamos:

$$IN_{O1} = \begin{bmatrix}0,3\\0,7\\0,2\end{bmatrix}^T * \begin{bmatrix}0,7981\\0,7791\\0,8072\end{bmatrix} + 1 = [0,3 \quad 0,7 \quad 0,2] * \begin{bmatrix}0,79811\\0,7791\\0,8072\end{bmatrix} + 1 = 1,9462$$

$$OUT_{O1} = sigmoid(IN_{O1}) = sigmoide(1,9462) = 0,8750$$

Os mesmos resultados, como seria de esperar.

Em Python seria:

```
#define função sigmoide
def sigmoid(Z):
    A = 1/(1+np.exp(-Z))
    return A

                OUT_O =
  sigmoid(np.dot(w_o.T,OUT_h)+b)
```

Atingido o último nível da rede neuronal vamos agora avaliar através do erro quadrático médio, a performance da rede. Como se viu no diagrama anterior da rede, o valor esperado para o nó de saída é $target_{O1}$=1.

Aplicando a fórmula do erro quadrático médio em relação aos valores obtidos e esperados, vamos obter, e em que n=1 uma vez que só existe um nó de saída:

$$E_{total} = MSE = \frac{1}{n}\sum_{i=1}^{n}\left(y^{"}_{i} - y_i\right)^2 = \frac{1}{2}\left((OUT_{o1} - target_{o1})^2\right)$$

cujo cálculo do erro total (E_{total}) dá:

$$E_{total} = \frac{1}{2}(0,8750 - 1)^2 = 0,0078$$

O fator ½ é incluído para que o expoente 2 seja cancelado quando derivarmos esta equação mais tarde. Como tipicamente o resultado é depois multiplicado por um fator de *learning rate*, não é relevante para o resultado final, o valor que aqui se coloca.

Matricialmente usar-se-iam na mesma os valores obtidos e o cálculo seria idêntico.

A implementação em Python poderia ser da seguinte forma:

```
Etotal = (1/2)*(OUTo-targeto1)**2
```

Feita a avaliação da rede neuronal (0,0078), resultado do cálculo do erro quadrático médio, o valor obtido deveria ser o mais perto possível de 1, o que não acontece, significando que o erro é ainda grande. Há, portanto, que o otimizar e para isso vamos recorrer à função gradiente, e utilizar o mecanismo conhecido como *backward propagation* ou *backpropagation*.

O objetivo do *backpropagation* é o de atualizar os valores dos pesos (w_{ij}) da rede neuronal de forma a que eles permitam ir aproximando o resultado obtido do resultado esperado, isto é, minimizando o erro e assim otimizando o resultado obtido.

Estas atualizações são calculadas pela otimização via gradiente, que corresponde ao respetivo cálculo diferencial em relação aos valores dos pesos (w_{ij}) das diversas ligações da rede neuronal.

Vamos começar nas ligações entre o *hidden layer* e o nível de saída da rede. Como se vê no modelo da rede, estamos a falar dos pesos wo_{11}, wo_{21} e wo_{31}.

Tomemos por exemplo o peso wo_{11}. O que vamos querer saber é quanto é que uma mudança do valor do peso wo_{11} afeta o erro total E_{total}, isto é, matematicamente e usando cálculo diferencial, vamos querer calcular $\frac{\partial E_{total}}{\partial wo_{11}}$, que mais não é que a derivada parcial de E_{total} em relação a wo_{11}.

A dependência entre o peso wo_{11} e o erro E_{total}, está representada na Figura 15, para facilidade de interpretação da explicação que segue:

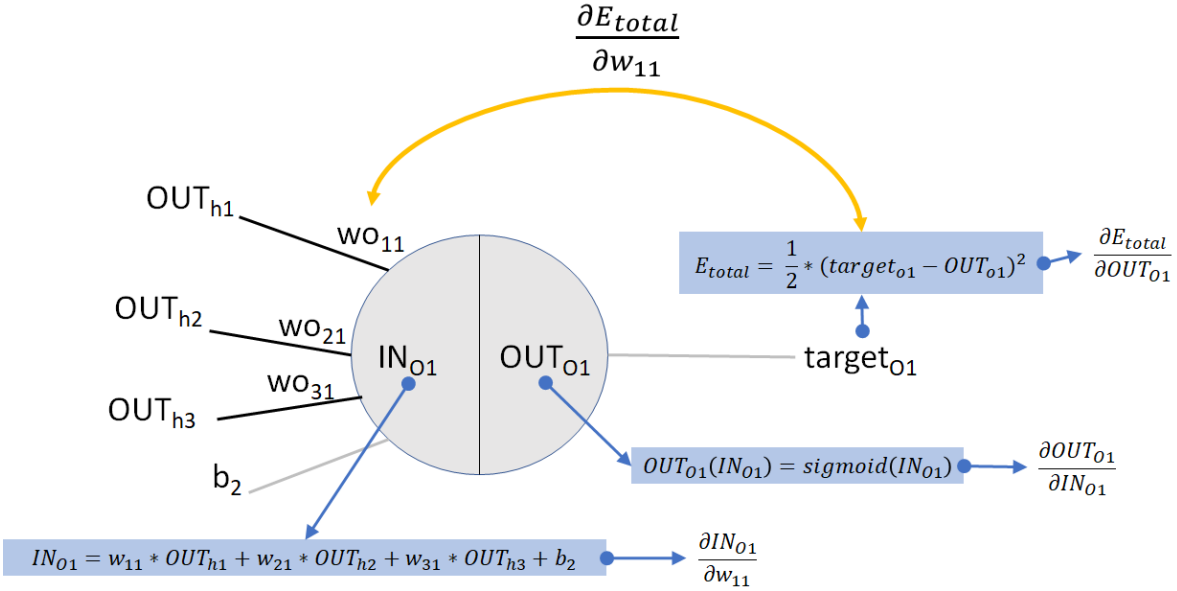

Figura 15. Detalhe do nó O1 da rede neuronal

Neste diagrama podemos ver claramente que, desde wo_{11}, e para o cálculo de E_{total}, contribuem 3 elementos: IN_{O1}, OUT_{O1} e o próprio erro E_{total}, e cujas fórmulas já conhecidas estão também representadas no diagrama.

Por esse motivo, para calcular $\frac{\partial E_{total}}{\partial wo_{11}}$, e olhando para os componentes representados no diagrama, matematicamente teremos que:

$$\frac{\partial E_{total}}{\partial wo_{11}} = \frac{\partial E_{total}}{\partial OUT_{O1}} \times \frac{\partial OUT_{O1}}{\partial IN_{O1}} \times \frac{\partial IN_{O1}}{\partial wo_{11}}$$

Isto corresponde aos elementos representados na figura com as respetivas fórmulas, isto é, para calcularmos quanto é que w_{11} contribui para o erro total, teremos de calcular quanto é que OUT_{O1} contribui para o erro total, quanto é que IN_{O1} contribui para OUT_{O1} e quanto é que Wo_{11} contribui para IN_{O1}, multiplicando no final estes fatores. No fundo estamos a analisar, mas no caminho inverso, todos os cálculos feitos no *forward propagation* para passagem pelo nó O_1 e que tenham direta ou indiretamente w_{11} incorporado.

Resumindo, temos então de calcular cada componente da equação, ou seja:

1) $\frac{\partial E_{total}}{\partial OUT_{O1}}$
2) $\frac{\partial OUT_{O1}}{\partial IN_{O1}}$
3) $\frac{\partial IN_{O1}}{\partial wo_{11}}$

Começando na primeira equação, e tendo por base que $E_{total} = \frac{1}{2}(\ (target_{O1} - OUT_{O1})^2)$ e calculando a derivada parcial, teremos que:

$$\frac{\partial E_{total}}{\partial OUT_{O1}} = 2 * \frac{1}{2}(target_{O1} - OUT_{O1})^{2-1} * (-1) \ = \ -1 * (target_{O1} - OUT_{O1})$$

cujo resultado será de:

$$\frac{\partial E_{total}}{\partial OUT_{O1}} = -(target_{O1} - OUT_{O1}) = \ -(1 - 0{,}8750) = -0{,}1250$$

Para a segunda equação $\frac{\partial OUT_{O1}}{\partial IN_{O1}}$, vamos ver quanto é que o OUT_{O1} muda em função da entrada IN_{O1}. Lembremos que aqui foi usada a função sigmoide, isto é, OUT_{O1}=sigmoid(IN_{O1}) como vimos anteriormente, ou seja $OUT_{O1}(IN_{O1}) = \frac{1}{1+ e^{-IN_{O1}}}$.

Calculando a derivada parcial teremos (dispensando as demonstrações matemáticas):

$$\frac{\partial OUT_{O1}}{\partial IN_{O1}} = \ \frac{\partial}{\partial IN_{O1}}\left(\frac{1}{1 + e^{-IN_{O1}}}\right) = \ \frac{e^{-IN_{O1}}}{(1 + e^{-IN_{O1}})^2} = \left(\frac{1}{1 + e^{-IN_{O1}}}\right) * \left(1 - \ \left(\frac{1}{1 + e^{-IN_{O1}}}\right)\right)$$

ou seja:

$$\frac{\partial OUT_{O1}}{\partial IN_{O1}} = OUT_{O1} * (1 - OUT_{O1})$$

Assim sendo, e colocando valores:

$$\frac{\partial OUT_{O1}}{\partial IN_{O1}} = OUT_{O1} * (1 - OUT_{O1}) = 0{,}8750 * (1 - 0{,}8750) = 0{,}1094$$

Por fim, calculamos quanto é que IN_{O1} varia com o peso wo_{11}. E, como vimos, o cálculo de IN_{O1} era dado por: $IN_{O1} = wo_{11} * OUT_{h1} + wo_{12} * OUT_{h2} + wo_{13} * OUT_{h3} + wo_2 + b_2$

Calculando a derivada parcial desta função, temos que:

$$\frac{\partial IN_{O1}}{\partial wo_{11}} = 1 * OUT_{h1} * wo_{11}^{(1-1)} + 0 + 0 + 0 = OUT_{h1} = 0{,}7981$$

Calculados os 3 elementos, e colocando tudo junto, teremos então:

$$\frac{\partial E_{total}}{\partial wo_{11}} = -0{,}1250 \times 0{,}1094 \times 0{,}7981 = -0{,}0109$$

Os cálculos anteriores de $\frac{\partial E_{total}}{\partial wo_{11}}$ podem ser representados matematicamente de uma forma agregada, ou seja:

$$\frac{\partial E_{total}}{\partial wo_{11}} = -(target_{O1} - OUT_{O1}) * OUT_{O1} * (1 - OUT_{O1}) * OUT_{h1}$$

Este valor terá agora de ser subtraído ao valor atual do peso wo_{11} de forma a diminuir o erro total. Vamos utilizar também um valor de *learning rate*, já explicado anteriormente, que designaremos por θ e ao qual atribuiremos como exemplo, o valor 0,5.

Este valor atuará sobre o valor obtido acima, isto é, o novo valor do peso w_{11} passará a ser de:

$$wo_{11}^{+} = wo_{11} - \theta * \frac{\partial E_{total}}{\partial w_{11}} = 0{,}3 - 0{,}5 * (-0{,}0109) = 0{,}2945$$

Usando o mesmo raciocínio, mas agora para os pesos wo_{21} e wo_{31} teremos que:

$$\frac{\partial E_{total}}{\partial wo_{21}} = -(target_{O1} - OUT_{O1}) * OUT_{O1} * (1 - OUT_{O1}) * OUT_{h2}$$

$$\frac{\partial E_{total}}{\partial wo_{21}} = -(1 - 0{,}8750) * 0{,}8750 * (1 - 0{,}8750) * 0{,}7791 = -0{,}0106$$

$$wo_{21}{}^{+} = wo_{21} - \theta * \frac{\partial E_{total}}{\partial wo_{21}} = 0{,}7 - 0{,}5 * (-0{,}0106) = 0{,}7053$$

$$\frac{\partial E_{total}}{\partial wo_{31}} = -(target_{O1} - OUT_{O1}) * OUT_{O1} * (1 - OUT_{O1}) * OUT_{h3}$$

$$\frac{\partial E_{total}}{\partial wo_{31}} = -(1 - 0{,}8750) * 0{,}8750 * (1 - 0{,}8750) * 0{,}8072 = -0{,}0110$$

$$wo_{31}{}^{+} = wo_{31} - \theta * \frac{\partial E_{total}}{\partial wo_{31}} = 0{,}2 - 0{,}5 * (-0{,}0110) = 0{,}2055$$

Matricialmente, a representação dos cálculos seria:

$$\frac{\partial E_{total}}{\partial wo_{ij}} = -(target_{O1} - OUT_{O1}) * OUT_{O1} * (1 - OUT_{O1}) * OUT_{hi}$$

$$wo_{ij}{}^{+} = wo_{ij} - \theta * \frac{\partial E_{total}}{\partial wo_{ij}}$$

Cuja implementação em Python poderia ser:

```
dwo = -(targeto1-OUTo)*OUTo*(1-
OUTo)*OUTh
wo = np.subtract(wo,np.dot(LRate,dwo))
```

Falta-nos agora calcular de que forma as variações dos pesos wh_{ij} contribuem para o erro total E_{total}. Estes pesos como vemos no modelo, são entradas do *hidden layer*.

Em mais detalhe, a propagação de dependências entre wh_{11} e E_{total} está representada na Figura 16.

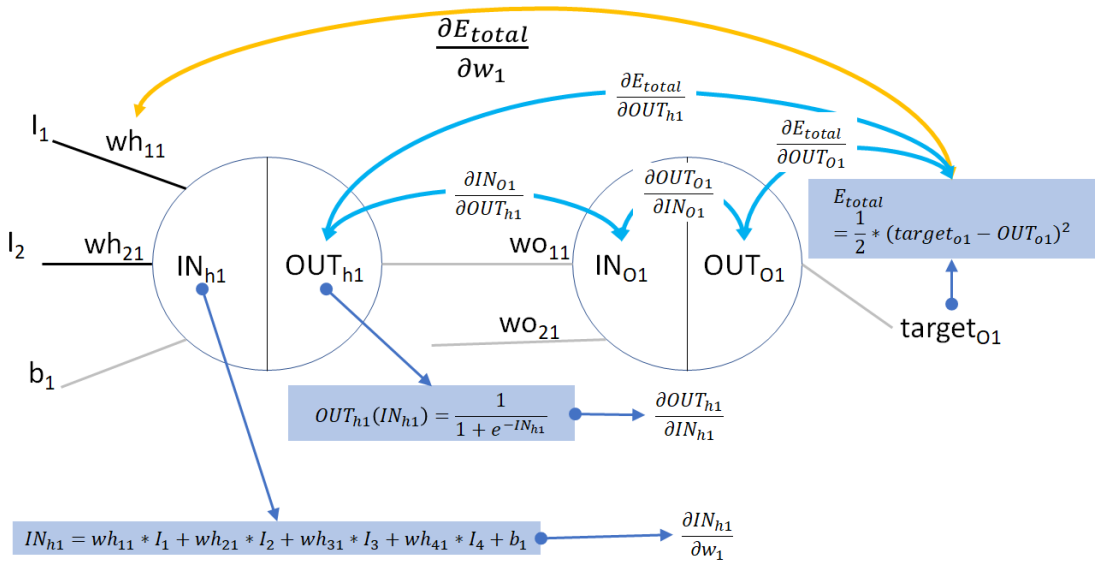

Figura 16. Detalhe dos nós O1 e H1 na fase de *back-propagation*

Como vemos no diagrama, para obter a variação de E_{total} em função de wh_{11}, temos 3 elementos: a variação de IN_{h1} com wh_{11}, a variação de OUT_{h1} com IN_{h1} e a variação de E_{total} com OUT_{h1}, o que pode ser representado matematicamente por:

$$\frac{\partial E_{total}}{\partial wh_{11}} = \frac{\partial E_{total}}{\partial OUT_{h1}} \times \frac{\partial OUT_{h1}}{\partial IN_{h1}} \times \frac{\partial IN_{h1}}{\partial wh_{11}}$$

Ou seja, teremos novamente 3 equações:

1) $\frac{\partial E_{total}}{\partial OUT_{h1}}$
2) $\frac{\partial OUT_{h1}}{\partial IN_{h1}}$
3) $\frac{\partial IN_{h1}}{\partial wh_{11}}$

A primeira equação tem incorporadas as variações já calculadas de passagem pelo último nó, e decompõe-se como se vê no diagrama anterior, em fatores já conhecidos e calculados, isto é:

$$\frac{\partial E_{total}}{\partial OUT_{h1}} = \frac{\partial E_{total}}{\partial IN_{O1}} * \frac{\partial IN_{O1}}{\partial OUT_{h1}}$$

onde

$$\frac{\partial E_{total}}{\partial IN_{O1}} = \frac{\partial E_{total}}{\partial OUT_{O1}} * \frac{\partial OUT_{O1}}{\partial IN_{O1}} = -0{,}1250 * 0{,}1094 = -0{,}0137$$

e

$$\frac{\partial IN_{O1}}{\partial OUT_{h1}} = wo_{11} = 0{,}3$$

Assim sendo:

$$\frac{\partial E_{total}}{\partial OUT_{h1}} = \frac{\partial E_{total}}{\partial IN_{O1}} * \frac{\partial IN_{O1}}{\partial OUT_{h1}} = -0{,}0137 * 0{,}3 = -0{,}0041$$

Como já vimos também anteriormente, $\frac{\partial OUT_{h1}}{\partial IN_{h1}}$ tem a ver com o processamento feito dentro do nó através da função de ativação sigmoide, e usando a mesma fórmula já vista, temos que:

$$\frac{\partial OUT_{h1}}{\partial IN_{h1}} = OUT_{h1} * (1 - OUT_{h1}) = 0{,}7981 * (1 - 0{,}7981) = 0{,}1611$$

Já para a terceira equação$\frac{\partial IN_{h1}}{\partial wh_{11}}$ e como vimos, o valor de IN_{h1} é obtido por:

$$IN_{h1} = wh_{11} * I_1 + wh_{21} * I_2 + wh_{31} * I_3 + wh_{41} * I_4 + b_1$$

e portanto,

$$\frac{\partial IN_{h1}}{\partial wh_{11}} = I_1 = 0$$

Assim sendo, temos que:

$$\frac{\partial E_{total}}{\partial wh_{11}} = \frac{\partial E_{total}}{\partial OUT_{h1}} \times \frac{\partial OUT_{h1}}{\partial IN_{h1}} \times \frac{\partial IN_{h1}}{\partial wh_{11}} = -0{,}0041 * 0{,}1611 * 0 = 0$$

Agregando tudo poderíamos representar a equação desta forma:

$$\frac{\partial E_{total}}{\partial wh_{11}} = -(target_{O1} - OUT_{O1}) * OUT_{O1} * (1 - OUT_{O1}) * wo_{11} \times OUT_{h1} * (1 - OUT_{h1}) \times I_1$$

que dará o mesmo resultado de 0.

Usando o mesmo valor de *learning rate*, o novo valor de wh_{11} passará a ser de:

$$wh_{11}{}^{+} = wh_{11} - \theta * \frac{\partial E_{total}}{\partial wh_{11}} = 0{,}1 - 0{,}5 * (0) = 0{,}1$$

Estas fórmulas são aplicáveis aos outros pesos da seguinte forma matricial:

$$\frac{\partial E_{total}}{\partial wh_{ij}} = -(target_{O1} - OUT_{O1}) * OUT_{O1} * (1 - OUT_{O1}) * wo_{ij} \times OUT_{hj} * \left(1 - OUT_{hj}\right) \times I_i$$

$$wh_{ij}{}^{+} = wh_{ij} - \theta * \frac{\partial E_{total}}{\partial wh_{ij}}$$

Note-se que as matrizes wo_{ij}, OUT_h e I têm dimensões diferentes pelo que o cálculo matricial referido acima, deverá ter em linha de conta estas dimensões e executar a fórmula de forma adequada.

Calculando matricialmente e usando esta fórmula:

$$\frac{\partial E_{total}}{\partial wh_{ij}} = \begin{bmatrix} 0 & 0 & 0 \\ -3.19466792\text{e}-04 & -7.88034678\text{e}-04 & -2.07631109\text{e}-04 \\ -9.35533280\text{e}-05 & -2.30769734\text{e}-04 & -6.08031310\text{e}-05 \\ -6.72561668\text{e}-05 & -1.65902037\text{e}-04 & -4.37118124\text{e}-05 \end{bmatrix}$$

$$wh_{ij}{}^{+} = wh_{ij} - \theta * \frac{\partial E_{total}}{\partial wh_{ij}} = \begin{bmatrix} 0.1 & 0.7 & 0.4 \\ 0.60015973 & 0.20039402 & 0.80010382 \\ 0.50004678 & 0.90011538 & 0.3000304 \\ 0.20003363 & 0.40008295 & 0.10002186 \end{bmatrix}$$

que são os novos pesos a aplicar à rede neuronal nas ligações entre os nós de entrada e os do *hidden layer*.

Dispensamos aqui os cálculos para obter os resultados acima, que serão exaustivos sem recurso a programação.

Programaticamente e em Python poderíamos implementar da seguinte forma:

```
dwh = np.empty(shape=wh.shape)
B = np.empty(shape=wh.shape)
A = -(targeto1-OUTo)*OUTo*(1-OUTo)*wo*OUTh*(1-OUTh)
B = np.multiply.outer(A, I)
C = np.squeeze(B)
Dwh = C.T
wh = np.subtract(wh,np.dot(LRate,dwh))
```

Os cálculos referidos até este ponto refletem uma única iteração com a rede neuronal para um conjunto de valores de entrada $I = \begin{bmatrix} 0 \\ 0{,}475 \\ 0{,}1391 \\ 0{,}1 \end{bmatrix}$ e sua classificação ($target_{o1}$=1).

De facto, sendo o princípio minimizar o erro obtido, vamos precisar de iterar estes cálculos diversas vezes. Para isso o melhor é recorrer mesmo à programação, pelo que copiamos abaixo o código Python usado, salvaguardando que tem fins puramente demonstrativos e não de elegância ou otimização de código.

Vamos também precisar de duas bibliotecas Python, numpy (para os cálculos) e pyplot, para desenhar os gráficos.

```
#!pip install numpy
import numpy as np

#define função sigmoide
def sigmoid(Z):
   A = 1/(1+np.exp(-Z))
   return A

#Inicializa variaveis
I = np.array( [[0],  [0.475], [0.1391],  [0.1]] )

wh = np.array( [[0.1, 0.7, 0.4], [0.6, 0.2, 0.8],
                [0.5, 0.9, 0.3],  [0.2, 0.4, 0.1]] )

wo = np.array( [[0.3], [0.7], [0.2]] )
b=1
targeto1 = 1
```

```
LRate=0.5
interacoes=100
erro = np.empty(shape=[interacoes])

for i in range(0,interacoes):
   #Forward Propagation
   OUTh = sigmoid(np.dot(wh.T,I)+b)
   OUTo = sigmoid(np.dot(wo.T,OUTh)+b)
   Etotal = (1/2)*(OUTo-targeto1)**2
   erro[i]=Etotal
   #print(OUTo)

   #BackPropagation
   dwo = -(targeto1-OUTo)*OUTo*(1-OUTo)*OUTh
   wo = np.subtract(wo,np.dot(LRate,dwo))

   dwh = np.empty(shape=wh.shape)
   B = np.empty(shape=wh.shape)
   A=-(targeto1-OUTo)*OUTo*(1-OUTo)*wo*OUTh*(1-OUTh)
   B=np.multiply.outer(A, I)
   C=np.squeeze(B)
   dwh=C.T
   wh = np.subtract(wh,np.dot(LRate,dwh))
```

Como podemos ver no código vamos recorrer a 100 iterações. Executado o programa acima, podemos traçar um gráfico com os valores de erro que foram obtidos nestas iterações. Para isso podemos usar o código em Python:

```
#!pip install matplotlib
import matplotlib.pyplot as plt

plt.plot(erro)
plt.ylabel('Interactions')
plt.show()
```

A execução deste código desenha o gráfico do erro obtido, que no nosso caso para os valores em causa resulta na Figura 17.

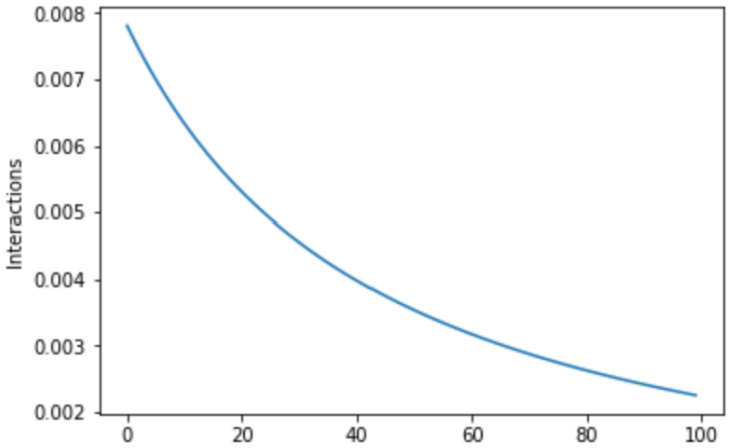

Figura 17. Minimização erro em 100 iterações

Se aumentarmos no código Python o número de iterações para 1000, o gráfico obtido é apresentado na Figura 18.

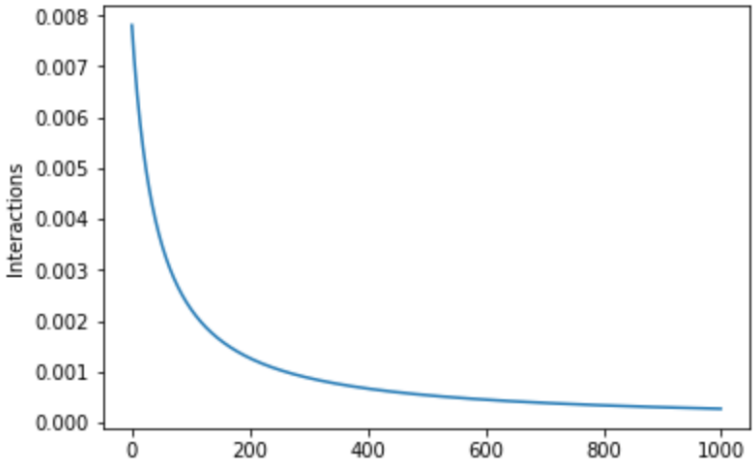

Figura 18. Minimização erro em 1000 iterações

Como se vê o erro aproxima-se mais de 0 com 1000 iterações.

Por outro lado, no código definimos também a *learning rate* como tendo o valor de 0,5. Experimentemos usar um valor de 4 para *learning rate*, e as mesmas 1000 iterações. O gráfico obtido é apresentado na Figura 19.

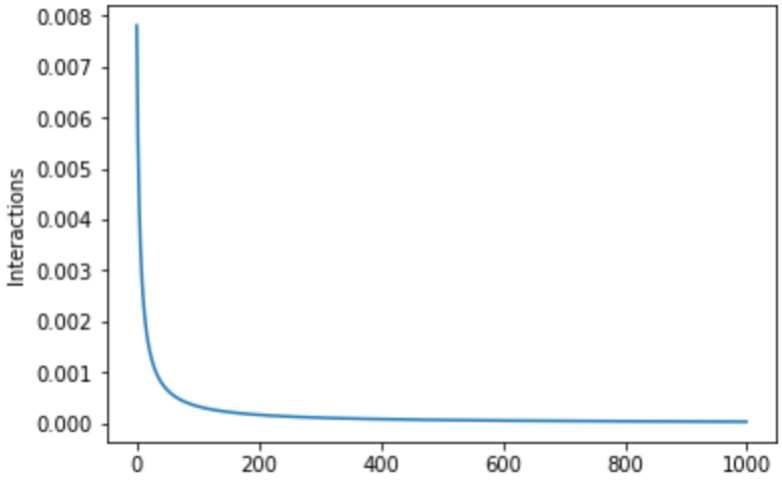

Figura 19. Minimização erro em 100 iterações *learning rate* de 4

Como se observa, e neste caso, um maior valor de *learning rate*, aproxima mais rapidamente o erro de zero.

As iterações feitas para um conjunto de valores à entrada da rede, e explicados até aqui, deveriam repetir-se para vários conjuntos de dados de entrada e respetiva classificação, isto é, a rede deverá ser treinada em diferentes exemplos de forma a generalizar melhor face a exemplos para os quais não tenha sido treinada.

O modelo explicado é muito simples, socorremo-nos de uma rede neuronal simples, e serviu somente para abordar os fundamentos básicos. Poderiam ter sido consideradas variantes [Grover 2018)], por exemplo, ao cálculo do erro, uma vez que basicamente estamos perante um classificador de ‘0’ e ‘1’.

Este modelo poderia ser utilizado para classificar imagens, peras ou maças por exemplo, mas, no entanto, não seria possível ter imagens com peras e maças ao mesmo tempo, teriam de ser só peras ou só maças, nem tão pouco seria possível dizer em que zona da imagem está localizada a pera ou a maça.

Entram então aqui as CNN já referidas e que permitem classificar uma série de atributos na imagem. Estas redes, para além de um número elevado de nós de entrada, têm também um número elevado de níveis, em milhares tipicamente, pelo que é possível imaginarmos o poder computacional envolvido para treinar um destes modelos, o que é facilitado pela utilização de GPUs ou processadores especializados

Estas redes com maior profundidade são muito utilizadas, como já se referiu, para reconhecimento de imagem, de voz e processamento de texto ou NLP (Natural Language Processing).

5. Conclusões

Esta breve viagem pela inteligência artificial iniciou-se numa visão geral do que é a inteligência artificial, o que está no horizonte como é o caso da AGI (*Artificial Generic Intelligence*), desceu à realidade da *narrow* AI e foi olhar para o interior de um dos modelos usados em *machine learning*, como é o caso de uma rede neuronal.

Viajou-se também para o interior da rede neuronal, numa das suas representações mais simples, e os dados utilizados para ilustrar o funcionamento da rede serviram unicamente para isso mesmo, ilustração.

Deste exemplo muito simples, só com 4 nós, pode extrapolar-se a dificuldade de criar redes com diversos graus de profundidade, bem como com milhares de nós de entrada, como se viu sempre dependentes da informação em estudo. Existem hoje em dia modelos com centenas de milhares de nós. Pode imaginar-se o tamanho dos vetores de entrada e das matrizes de cálculo intermédias.

É importante realçar que apesar desta viagem se ter feito com o objetivo de construir uma rede neuronal simples, a utilização de inteligência artificial hoje em dia, também pode ser feita recorrendo a modelos já treinados de reconhecimento de imagem ou texto que estão disponíveis na *cloud* para serem consumidos. Podem ser encontrados nas *clouds* da IBM, do Google, da Microsoft, da Amazon, para referir as principais.

Esses modelos existem como serviços (APIs) para serem incorporados em aplicações de lazer ou aplicações de negócio, e sempre no sentido de colaboração com o ser humano. De facto, a AI deverá ser vista como complementar ao ser humano, auxiliando-nos em muitas tarefas, e libertando-nos para o que sabemos fazer melhor que a AI, raciocinar.

A situação em que se utiliza a construção do modelo ou a reutilização do modelo, dependerá da solução que pretendemos construir e também do tipo de dados que temos em mão. Os modelos ou APIs existem na generalidade para voz, imagem e processamento de texto, e muitas vezes os dados que temos em mão, como os utilizados no exemplo explicado, são estruturados e específicos de uma determinada indústria ou setor, sendo também muitas vezes considerados confidenciais.

A inteligência artificial não se esgota neste texto, cuja intenção final foi de despertar a curiosidade para fazer outras viagens exploratórias no mundo da inteligência artificial através da *web* com todo o seu manancial de informação teórica e prática no assunto.

Bibliografia

[1] Koch C. (2004), The Quest for Consciousness: A Neurobiological Approach, Roberts and Company Publishers, ISBN-13: 978-1936221042.

[2] Russell S. and P. Norvig (2009), Artificial Intelligence: A Modern Approach, 3rd Edition, Pearson, ISBN-13: 978-1292153964.

[3] Bostrom N. (2016), Superintelligence: Paths, Dangers, Strategies, Oxford University Press, ISBN-13: 978-0198739838.

[4] Domingos P. (2015), The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World, Basic Books, ISBN-13: 978-0465094271.

[5] Oliveira A. (2018), The Digital Mind: How Science Is Redefining Humanity, The MIT Press, ISBN-13: 978-0262535236.

[6] The Terminator (1984), film directed by James Cameron, 1h 47 min.

[7] Tegmark M. (2018), Life 3.0: Being Human in the Age of Artificial Intelligence, Vintage Ed., ISBN-13: 978-1101970317.

[8] Grover P. (2018), Five Regression Loss Functions All Machine Learners Should Know: Choosing the right loss function for fitting a model, https://heartbeat.fritz.ai/5-regression-loss-functions-all-machine-learners-should-know-4fb140e9d4b0, acedida em março 2019.

Jorge Soares, trabalha na Companhia IBM Portuguesa desde 1988, e como arquiteto de *software* desde 2004. Participou em diversos projetos em variadas indústrias com mais incidência na área financeira, bancária mais propriamente. Nos últimos anos esteve envolvido em projetos na área de *smart cities*, e mais recentemente tem estado envolvido em projetos de inteligência artificial, com mais predominância na área preditiva e na utilização de assistentes virtuais ou bots (robots de software).