Redes Neuronais Artificiais

Paulo Cortez e José Neves

Unidade de Ensino
Departamento de Informática
Escola de Engenharia
Universidade do Minho

Braga, Portugal
2000

Capítulo 1

Redes Neuronais Artificiais

1.1 O que é uma Rede Neuronal Artificial?

As *Redes Neuronais Artificiais (RNAs),* também designadas por *sistemas conexionistas*, são modelos simplificados do sistema nervoso central do ser humano. Trata-se de uma estrutura extremamente interconectada de *unidades computacionais*, frequentemente designadas por *neurónios ou nodos,* com capacidade de aprendizagem. Eis uma definição de uma RNA vista como uma máquina adaptativa [12]:

> Uma *RNA* é um processador eminentemente paralelo, composto por simples unidades de processamento, que possui uma propensão natural para armazenar conhecimento empírico e tornar-lo acessível ao utilizador. Assemelha-se ao comportamento do cérebro em dois aspectos:
>
> - O conhecimento é adquirido a partir de um ambiente, através de um processo de aprendizagem.
> - O conhecimento é armazenado nas *conexões,* também designadas por *ligações* ou *sinapses*, entre nodos.

Durante o processo de aprendizagem, dado por um *algoritmo de aprendizagem* ou *de treino*, a *força* (ou *peso)* das conexões é ajustada de forma a se

atingir um desejado objectivo ou estado de conhecimento da rede. Embora seja esta a forma tradicional de construir *RNAs* também é possível modificar a sua própria estrutura interna (ou *topologia*), à semelhança do que se passa no cérebro, onde neurónios podem morrer e novas sinapses (e mesmo neurónios) se podem desenvolver.

1.2 Inspiração Biológica: O Cérebro Humano

A maior parte da investigação em *RNAs* foi inspirada e influenciada pelos sistemas nervosos dos seres vivos, em particular do ser humano. Muitos investigadores acreditam que as *RNAs* oferecem a aproximação mais promissora para a construção de verdadeiros sistemas inteligentes, tendo capacidade para ultrapassar a explosão combinatorial associada à computação simbólica baseada em arquitecturas de *von Neumann*[1] [25].

O sistema nervoso central fornece uma forte base de sustentação a esta tese. O *cérebro* é uma estrutura altamente complexa, não linear e paralela. Possui uma capacidade de organizar os seus constituintes, conhecidos por *neurónios*, de modo a executarem certas tarefas complexas (e.g. reconhecimento de padrões ou voz), de uma forma inatingível pelo computador mais potente até hoje concebido.

Apesar dos grandes avanços científicos, o conhecimento acerca do modo como o cérebro humano opera está longe de estar completo. No entanto, alguns factos importantes são já conhecidos. Quando alguém nasce, o seu cérebro apresenta-se já com uma estrutura fortemente conexionista, com capacidade de aprender através da *experiência*. Este conhecimento evolui em função do tempo, apresentando-se com um desenvolvimento mais acentuado nos primeiros dois anos de vida. Em termos de velocidade de processamento, um neurónio é cerca de 5 a 6 vezes mais lento do que uma porta lógica de silício. Todavia, o cérebro ultrapassa esta lentidão utilizando uma estrutura

[1] John von Neumann (1903-1957), matemático húngaro-americano que teve uma grande contribuição na definição da arquitectura de máquinas sequenciais, onde um programa é armazenado na mesma memória de dados que o programa utiliza. Hoje em dia, quase todos computadores são do tipo von Neumann.

maciçamente paralela. Estima-se que o *cortex* humano possui cerca de 10 biliões de neurónios e 60 triliões de sinapses [12].

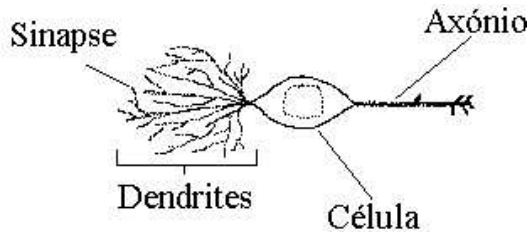

Figura 1.1: Estrutura de um neurónio natural

Um neurónio é uma célula complexa que responde a sinais electro-químicos, sendo composto por um *núcleo*, por um *corpo celular*, por um numeroso conjunto de *dendrites, entidades* que recebem sinais de outros neurónios via sinapses, e por um *axónio*, que transmite um estímulo a outros neurónios, através das já referidas sinapses (Figura 1.1) [3]. Um único neurónio pode estar ligado a centenas ou mesmo dezenas de milhares neurónios. Num cérebro existem estruturas anatómicas de pequena, média e alta complexidade com diferentes funções, sendo possíveis parcerias. Os néuronios tendem a agrupar-se em camadas, existindo três principais tipos de conexões (Figura 1.2):

- *divergentes*, onde um neurónio pode estar ligado a outros neurónios via uma arborização do axónio;

- *convergentes*, onde vários neurónios podem estar conectados a um único neurónio; e

- *encadeadas* ou *cíclicas*; as quais podem envolver vários neurónios e formarem ciclos.

Por sua vez, as conexões são constituídas por dois tipos de sinapses: *inibitórias* e *excitatórias*.

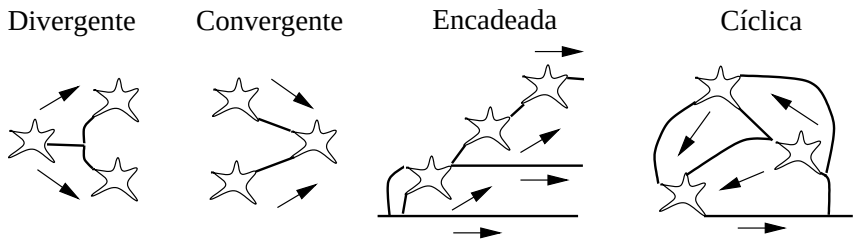

Figura 1.2: Os diferentes tipos de conexões

1.3 Benefícios das *RNA*s

O poder computacional de uma *RNA* alicerça-se em dois aspectos fundamentais: numa topologia que premeia o paralelismo, e por outro lado, na sua capacidade de aprendizagem e generalização; i.e., conseguir responder adequadamente a novas situações com base em experiências passadas. São estas duas características que tornam possível a resolução de problemas, que de outra forma seriam intratáveis. Isto não quer dizer que as *RNAs* sejam caixas mágicas que consigam por si dar resposta a qualquer problema. Em vez disso, precisam não raras vezes de serem integradas com outros sistemas ou paradigmas. Convém, ainda, reconhecer-se que ainda se está muito longe de atingir uma arquitectura que mimetize o cérebro humano [12].

As *RNAs* apresentam características únicas, que não se encontram em outros mecanismos ou técnicas [25][1][12]:

- *aprendizagem e generalização*; i.e., conseguindo descrever o todo a partir de algumas partes, constituindo-se como formas eficientes de aprendizagem e armazenamento de conhecimento;
- *processamento maciçamente paralelo*; i.e., permitindo que tafefas complexas sejam realizadas num curto espaço de tempo
- *transparência*; i.e., podendo ser vistas como uma caixa negra que transforma vectores de entrada em vectores de saída, via uma função desconhecida.
- *não linearidade*; i.e, atendendo a que muitos dos problemas reais a equacionar e resolver são de natureza não linear;

- *adaptatividade*; i.e., podendo adaptar a sua topologia de acordo com mudanças do ambiente;
- *resposta evidencial*; i.e., onde uma saída da rede traduz não só um processo de decisão mas também o grau de confiança a conferir a esta;
- *robustez e degradação suave*; i.e., permitindo processar o ruído ou informação incompleta de forma eficiênte, assim como sendo capazes de manter o seu desempenho quando há desactivação de algumas das suas conexões e/ou nodos; e
- *flexibilidade*; i.e., com um grande domínio de aplicabilidade.

Finalmente, convém referir que apesar do facto das RNAs partilharem muitas das características com do cérebro humano, sao carentes relativamente a outras, como o *esquecimento*.

1.4 Neurónio Artificial ou Nodo

Um *nodo*, termo usado para distingir entre um neurónio natural e artificial, é a unidade de processamento chave para a operação de uma *RNA*. Embora existam diversos tipos de nodos, em princípio, comporta-se como um comparador que produz uma saída quando o efeito cumulativo das entrada excede um dado valor limite. Um nodo pode ser dissecado de acordo com o exposto na Figura 1.3 [30]:

- *Um conjunto de conexões* (w_{ij}), cada uma etiquetada por um *peso*; i.e., um número real ou binário (embora seja mais comum na forma real) que tem um efeito excitatório para valores positivos e inibitório para valores negativos. Assim, o *sinal* ou *estímulo* (x_j) como entrada da conexão é multiplicado pelo corresponte peso w_{ij}, onde i denota o nodo objecto de estudo e j o nodo de onde partiu o sinal. Pode ainda existir uma conexão extra, denominada de *bias*, cuja entrada toma o valor $+1$, que estabelece uma certa tendência ou inclinação no processo computacional; i.e., adiciona uma constante (w_{i0}) para que se estabeleçam as correctas condições operacionais para o nodo.

- Um *integrador (g),* que reduz os n argumentos de entrada (estímulos) a um único valor. Frequentemente é utilizada a função *adição* ($\sum$), pesando todas as entradas numa combinação linear.

- Uma *função de activação* (f), que pode condiciona o sinal de saída, introduzindo uma componente de não linearidade no processo computacional.

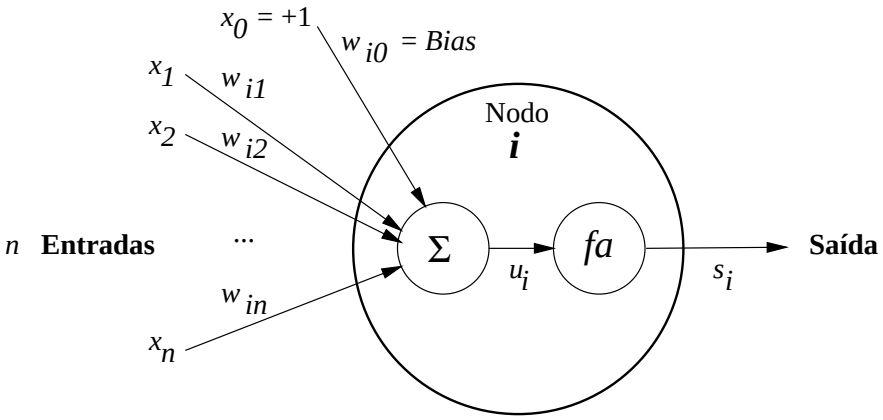

Figura 1.3: Estrutura geral de um nodo

Em termos formais tem-se que este nodo é descrito pelas seguintes equações:

$$u_i = g(1 \times w_{io}, x_1 \times w_{i1}, x_2 \times w_{i2}, ..., x_n \times w_{in}) \tag{1.1}$$

$$s_i = f(u_i) \tag{1.2}$$

para um nodo i com n entradas e uma saída, onde u_i representa o ganho do nodo i e s_i a *saída* do nodo.

A Tabela 1.1 mostra algumas das funções de activação mais utilizadas, onde k denota a *inclinação* da função, *mod* o resto de uma divisão inteira e $sign(x) = \frac{x}{|x|}$ [1] (Figura 1.4). A primeira função, também designada por função de *heaviside,* é normalmente utilizada em nodos do tipo McCulloch-Pitts [21], em que a saída toma o valor $+1$ apenas se o ganho for não-negativo,

Tabela 1.1: Funções de activação

Nome	**Função *f***	**Contradomínio**
limiar	$\begin{cases} 1 & , \ u_i \geq 0 \\ 0 & , \ u_i < 0 \end{cases}$	$\{0, 1\}$
linear	u_i	$]-\infty, +\infty[$
por troços	$\begin{cases} 1 & , \ u_i \geq 0.5 \\ ku_i & , \ -0.5 < u_i < 0.5 \\ 0 & , \ u_i \leq -0.5 \end{cases}$	$[0, 1]$
logística	$\frac{1}{1+\exp(-ku_i)}$	$[0, 1]$
tangente hiperbólica	$\tanh(ku_i)$	$[-1, 1]$
sin	$\sin(u_i \ mod \ 2\Pi)$	$[-1, 1]$
cos	$\cos(u_i \ mod \ 2\Pi)$	$[-1, 1]$
gaussiana	$\exp(\frac{-u_i^2}{2k^2})$	$[-1, 1]$
quadrada	$-sign(u_i)u_i^2$	$]-\infty, +\infty[$

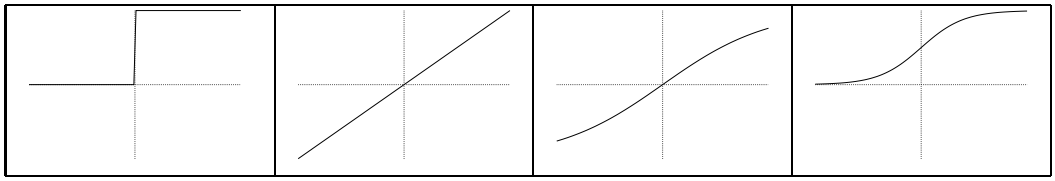

Figura 1.4: Funcões de activação típicas.

de acordo com uma filosofia do *tudo-ou-nada*. Em seguida, aparecem duas outras funções lineares, com a última a ter como contradomínio o intervalo $[0, 1]$. A não linearidade é definida em termos das restantes funções, onde uma especial atenção deve ser dada à função *logística*, também conhecida por função *sigmoid*. Esta função, cuja forma é modelada a um *S,* é de longe a função mais utilizada no uso de *RNAs*. É uma função crescente que exibe um balanceamento gracioso entre um comportamento linear e não linear (Figura 1.5)[14]. Ao se variar a *inclinação* (k) obtêm-se funções com diferentes declives; no limite, em que k se aproxima do infinito, a função tende para a *limiar*.

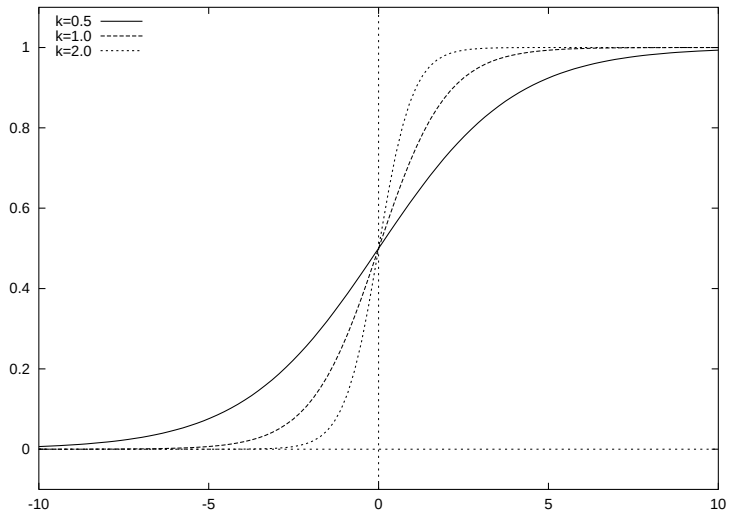

Figura 1.5: A função *sigmoid* para inclinações de $k = 0.5$, $k = 1.0$ e $k = 2.0$.

1.5 Arquitecturas de Rede

A forma como os nodos se interligam numa estrutura de rede é denominada por *arquitectura* ou *topologia.* Existem inúmeros tipos de arquitecturas de *RNAs* ou *topologias*, cada um com as suas próprias potencialidades. Em geral, caem dentro de três categorias [12]:

- *Redes Feedforward de uma Só Camada (RFSC).* Uma *RNA feedforward* pode ser organizada por camadas, pois não existem ciclos, dado que as conexões são sempre unidirecionais (*convergentes* ou *divergentes*). Na sua forma mais simples uma rede é composta por uma *camada de entrada*, cujos valores de saída são fixados externamente, e por uma *camada de saída* (Figura 1.6). De referir que a camada de entrada não é contabilizada como camada numa *RNA* devido ao facto de nesta não se efectuarem quaisquer cálculos.

- *Redes Feedforward MultiCamada (RFMC).* A segunda classe de redes *feedforward* destingue-se pelo facto de possuir uma ou mais *camadas intermédias,* cujos nodos são designados por *nodos intermédios* (Figura 1.7*).* A função destes é de intervir de forma útil entre a entrada e a saída da rede. Ao se acrescentarem camadas intemédias está-se a aumentar

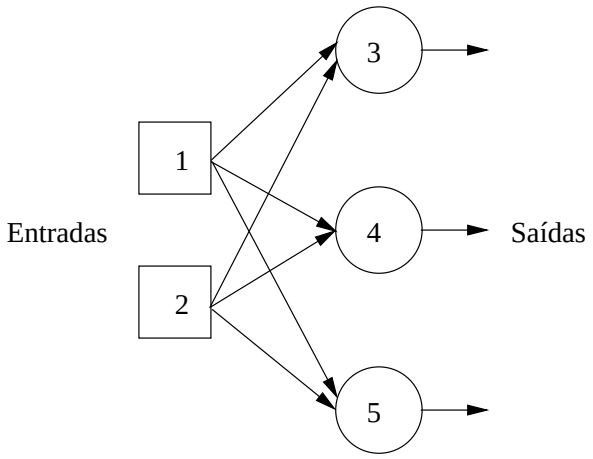

Figura 1.6: Arquitectura de uma *RFSC*

a capacidade da rede em modelar funções de maior complexidade, uma particularidade bastante útil quando o número de nodos na camada de entrada é elevado. Por outro lado, este aumento também transporta um senão, uma vez que o tempo de aprendizagem aumenta de forma exponencial.

- *Redes Recorrentes (RR).* A recorrência existe em sistemas dinâmicos quando uma saída de um elemento influência de algum modo a entrada para esse mesmo elemento, criando-se assim um ou mais circuitos fechados tal como apresentado na Figura 1.8. Assim que uma ou mais conexões cíclicas são incluídas numa rede, esta passa a ter um comportamento não linear, de natureza espaciail e/ou temporal, que podem ser utilizados para modelar novas funções cognitivas tais como as de *memoria associativa* e/ou *temporal* [3]. Ao conter ciclos, as saídas não são função exclusivamente das conexões entre nodos, mas também de uma *dimensão temporal*; i.e., está-se na presença de uma cálculo recursiv*o*, que obedecerá naturalmente a uma certa condição de paragem, com a última iteração a ser dada como a saída para o nodo [30].

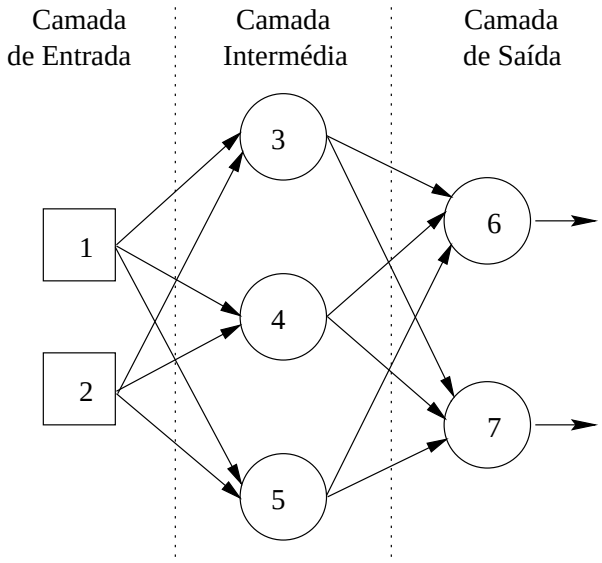

Figura 1.7: Arquitectura de uma RFMC

1.6 Aprendizagem

Como já foi referido, uma propriedade importante das *RNAs* é a sua capacidade apara aprender a partir do seu ambiente. A aprendizagem de uma *RNA* envolve a seguinte sequência de eventos [12]:

- a *RNA* é estimulada por um dado ambiente;
- certos parâmetros livres, normalmente *os pesos* das conexões, são alterados em resultado deste estímulo; e

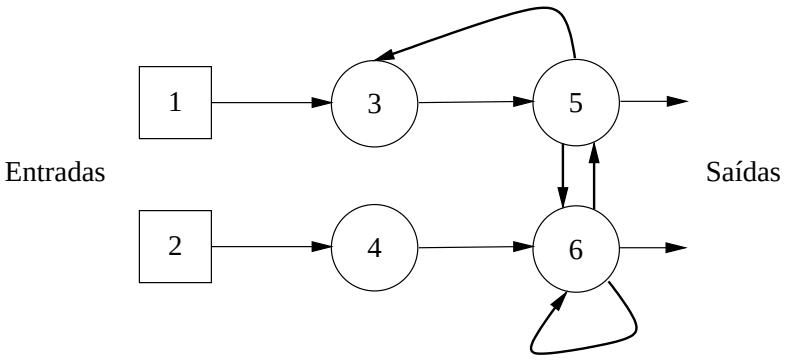

Figura 1.8: Arquitectura de uma *RC*

- a RNA responde de uma nova forma ao ambiente em virtude das alterações na sua estrutura interna.

Esta aprendizagem é executada a partir de um conjunto bem conhecido de *regras*, chamado de *algoritmo de aprendizagem* ou *treino*. Outro facto a ter em conta é forma como uma *RNA* se relaciona como seu ambiente. Neste contexto, está-se a falar de um *paradigma*; i.e., o modelo do ambiente em que a rede opera. Assim, os diversos algoritmos de treino distinguem-se pelo paradigma e regras de aprendizagem que utilizam, cada um oferecendo as suas vantagens.

1.6.1 Paradigmas de Aprendizagem

Existem três paradigmas fundamentais de aprendizagem [25]:

1. *Supervisionada.* Trata-se de um paradigma bastante popular que envolve um "professor", ou seja, são fornecidas respostas correctas à rede. A rede aprende a partir de um conjunto de padrões (P), onde cada *padrão* (p), também chamado de *exemplo* ou *caso de treino*, é composto por um vector de entrada (x^p) e por um vector de *resposta* ou *saída* (s^p). Durante o processo de aprendizagem, é efectuada uma comparação entre o valor desejado (t^p) com o valor de saída da rede, originando um *erro* ($e^p = t^p \ominus y^p$, sendo $\ominus$ a função de erro). O erro resultante é utilizado para de alguma forma ajustar os *pesos* da rede, de forma a que este seja reduzido. Uma *iteração* do algoritmo de treino é composta por ajustamentos *iterativos* ou *em lote* para todos os casos de treino. A aprendizagem é então conseguida quando, após $v$ariadas *iterações*, o erro é reduzido para valores aceitáveis.

2. *De Reforço.* Neste tipo de aprendizagem também se assume a presença de um "professor" embora a resposta correcta não seja apresentada à rede. Assim, apenas se fornece uma indicação sobre se a resposta da rede é correcta ou errada, tendo a rede de usar esta informação para melhorar a sua eficácia. No caso típico, um prémio é dado pelo reforço

dos pesos que dão uma resposta correcta e uma penalidade para a situação oposta.

3. *Não Supervisionada.* Trata-se de uma abordagem diferente, onde não é fornecida a um sistema uma indicação externa sobre acerca da resposta correcta. Por conseguinte, a aprendizagem é feita pela descoberta de características nos dados de entrada, adaptando-se a regularidades estatísticas ou agrupamentos de padrões dos exemplos de treino. Como exemplo deste tipo de aprendizagem temos as redes *Kohonen* [17].

1.6.2 Regras de Aprendizagem

As regras de aprendizagem dividem-se em cinco tipos básicos [12]:

1. *Hebbian.* Trata-se do mais antigo e mais famoso postulado de aprendizagem, proposto por Hebb[2] em 1949: "Quando um axónio da célula A está demasiado próximo para exitar uma célula B e dispara esta de forma repetitiva ou persistente, então algum processo metabólico ocorre numa ou ambas as células de forma a aumentar a eficiência da célula A em disparar B". Este postulado, constituido num contexto neurobiológico, foi expandido para uma regra de duas partes:

   - Se dois nodos em cada lado de uma conexão são activados simultaneamente, então a *força* dessa conexão é progressivamente aumentada.
   - Se dois nodos em cada lado de uma conexão são activados de forma assíncrona, então a conexão é progressivamente enfraquecida ou eliminada.

   Tal conexão, designada por sinapse de *Hebbian,* depende assim do tempo e da sua localização espacial. Esta regra é bastante utilizada na

[2]Donald Hebb (1904-1985). Psicólogo canadiano, tornou-se um teórico influente na relação entre o cérebro e o comportamento.

aprendizagem não supervisionada, com por exemplo em redes *auto-associativas* (e.g. *Hopfield* [13]) ou *heteroassociativas* (e.g. *Memória Associativa Bidirectional* [18]) (ver Secção 1.7).

2. *Competitiva.* Como o nome indica, as saídas dos nodos da mesma camada competem entre si para se tornarem activas, com apenas um nodo a ser activado num dado instante, segundo a estratégia do *vencedor-toma-tudo.* No começo do processo, os nodos de uma camada possuiem conexões com pesos pequenos mas desiguais. Quando um padrão é fornecido à rede, um dos elementos da camada responderá melhor do que os outros, sendo por isso "premiado" com um reforço dos pesos das suas conexões. Em alguns casos, os pesos dos nodos vizinhos também podem ser reforçados. Esta regra é bastante apropriada para a *extracção de características*, utilizada em problemas de *classificação* (ver Secção 1.7). As redes *SOM (*do inglês *Self-Organizing Maps)* como as *Kohonen* [17] e *ART* [6] utilizam este tipo de aprendizagem.

3. *Estocástica.* Neste tipo de aprendizagem, os pesos são ajustados de um modo probabilístico. Como exemplos temos o *Simulated Annealing* aplicado a máquinas de *Cauchy* e *Boltzmann* onde os estados dos nodos são determinados por uma distribuição probabilística [30].

4. *Baseada na Memória.* Esta forma baseia-se no armazenamento de todas (ou quase todas) experiências passadas, que são explícitamente armazenadas numa grande memória de pares entrada-saída. Quando surge um novo vector, x^p, nunca antes fornecido à rede, o algoritmo responde pela procura de vectores na região vizinha de x^p. Normalmente, o algoritmo e composto por dois componentes:

   - um critério de definição da vizinhança local do vector x^p; e
   - uma regra de aprendizagem aplicada aos exemplos de treino da vizinhança local de x^p.

   Como exemplo deste tipo de aprendizagem temos as redes *Radial-Basis Functions (RBF)* [5].

5. *Gradiente Descendente.* Como foi anteriormente referido, na aprendizagem supervisionada pretende-se reduzir o erro entre o valor pretendido com o valor de saída da rede. O *erro* actua assim como um mecanismo de controlo, onde os sucessivos ajustamentos deste vão originar melhores respostas. O objectivo é minimizar uma função de custo, ξ, definida em termos do sinal de erro e^p. A regra de aprendizagem, conhecida como a regra *delta* ou *Widrow-Hoff*, baseia-se na resolução da seguinte equação [28]:

$$\Delta w = \eta \nabla \xi \tag{1.3}$$

onde η denota uma constante positiva, chamada de *taxa de aprendizagem*, e $\nabla \xi$ o *gradiente* da função de custo. Trata-se de uma aprendizagem deveras utilizada e conhecida, associada a *RMC* e ao popular algoritmo de *Back-Propagation* [31].

1.7 Tarefas de Aprendizagem

A escolha da arquitectura e do método de aprendizagem é influênciada pela tarefa de aprendizagem a ser desempenhada pela *RNA*. Neste contexto existêm sete categorias principais [25][12]:

1. *Memória Associativa.* Trata-se de uma mémoria distribuída que aprende por *associação.* Esta pode tomar duas formas: *autoassociação* ou *heteroassociação.* No primeiro caso, a rede armazena um conjunto de padrões. Mais tarde, é apresendado à rede uma versão distorcida do padrão original, pelo que a rede deve ser capaz de devolver o padrão original. Na segunda situação, a diferença reside no facto dos padrões estarem organizados por pares de entradas e saídas. Assim, está-se na presença de uma aprendizagem supervisionada, ao contrário da autoassociação que usa uma aprendizagem não supervisionada.

2. *Diagnóstico.* Esta é uma tarefa deveras comum em àreas distintas como a *medicina*, a *engenharia* ou a *produção.* Trata-se essencialmente

de uma tafefa de *classificação*; i.e., exige uma correcta associação entre entradas, que representam dados indicadores de um estado (e.g. sintomas ou comportamento anormal), com o correspondente diagnóstico (e.g. doença ou falha do equipamento). De uma forma geral, as *RNAs* são integradas em sistemas periciais complexos, envolvendo outros paradigmas (e.g. *aprendizagem por regras*).

3. *Reconhecimento de padrões.* Formalmente, esta tarefa define-se como o processo pelo qual um sinal/padrão recebido é atribuído a uma de diversas categorias possíveis. Em primeiro lugar, é necessário treinar uma rede, onde os padrões, associados à respectiva categoria, são alimentados à rede de forma repetitiva. Mais tarde, um padrão novo é fornecido à rede, que terá de ser capaz de identificar a categoria correcta, de acordo com a informação assimilada. De uma forma geral, o reconhecimento de padrões por *RNAs* pode tomar duas formas. Na forma mais simples, utiliza-se uma única *RFMC* com um algoritmo de aprendizagem supervisionado, tendo os nodos intermédios a função de *extracção de características*; i.e., uma transformação da entrada (x) num ponto intermédio (y) pertencente a uma dimensão inferior. Esta redução facilita a tarefa de *classificação*, descrita como uma transformação do ponto intermédio (y) em uma das classes possíveis num espaço de decisão r-dimensional, para r classes distintas. Na segunda forma, a máquina de aprendizagem é composta por duas partes, a primeira para a *extracção de características,* via uma rede não supervisionada, e a restante para a *classificação*, via uma rede supervisionada. Importa referir ainda que as *RNAs* são bastante eficazes na aprendizagem de tarefas perceptivas como *o processamento de imagem,* o *reconhecimento de* voz *ou escrita,* chegando a ultrapassar métodos estatísticos convencionais (e.g. *classificadores Bayesianos*).

4. *Regressão/Previsão.* A ideia é conceber um*a RNA* capaz de modelar uma função desconhecida $f(\cdot)$ que se aproxime da função $F(\cdot)$ dada por um conjunto de vectores etiquetados; i.e., compostos por um par entrada-saída ($x \rightarrow y$), de forma a distância *euclidiana* seja pequena

para todos as entradas; i.e.:

$$\forall x,\ \|F(x) - f(x)\| < \rho \tag{1.4}$$

onde ρ representa um valor pequeno. A regressão é uma tarefa perfeita para a aprendizagem supervisionada. A *previsão*, caso particular da regressão, é uma tarefa comum em diversas àreas como a *economia*, vendas ou a produção, pretendendo-se "adivinhar" os valores futuros com base nos valores passados. As *RNAs* têm-se mostrado eficazes como ferramentas de previsão de variadas formas, desde a ocorrência (ou não ocorrência) de eventos assim como o tempo e nível de intensidade destes.

5. *Controlo.* Esta tarefa envolve um processo ou uma parte crítica de um sistema que tem de ser mantida numa condição controlada. O principal objectivo do *controlador* é fornecer os apropriados sinais para um dado sistema de forma que a saída (y) deste acompanhe uma entrada de referência (x). Como exemplos temos o controlo de *veiculos autónomos*, *robôs* ou *processos de fabrico*, onde as *RNAs* têm sido utilizadas com bastante sucesso.

6. *Optimização.* Existe um variado conjunto de problemas que envolvem a procura de soluções óptimas ou razoavelmente óptimas, compreendendo variados parâmetros, envolvendo um espaço de procura demasiado elevado, onde a exaustão de todas alternativas se torna incomportável em termos computacionais (e.g. o *problema do caixeiro viajante* ou o *escalonamento de tarefas*). A resolução deste tipo de problemas passa pelo uso de heurísticas e/ou máquinas de aprendizagem (incluindo as *RNAs*) que reduzem o espaço de procura de forma a que se atinjam soluções aceitáveis.

7. *Filtragem/Compressão de dados.* O termo *filtro* usualmente designa um dispositivo ou algoritmo que extrai informação de interesse a partir de um conjunto "bruto" de dados que contém ruído. Por sua vez, a *compressão* envolve uma redução de um espaço n-dimensional para

um espaço m-dimensional, sendo $n > m$. Ambas tarefas se tornam particularmente importantes quando existem quantidades enormes de dados a serem processados (e.g. processamento de imagens via satélite).

1.8 A Inteligência Artificial e as *RNAs*

O objectivo da *Inteligência Artificial (IA)* é o desenvolvimento de paradigmas e algoritmos para realizar tarefas cognitivas, as quais são actualmente executadas pelos os seres humanos de uma forma mais eficaz. Um sistema de *IA* deve ser capaz de perfazer três requisitos [12] (Figura 1.9):

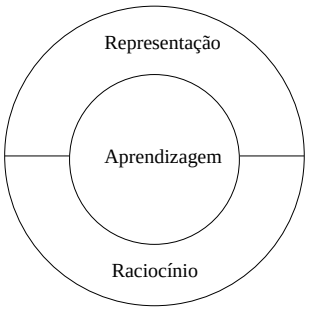

Figura 1.9: Os três componentes fundamentais de um sistema de *IA*

1. armazenar conhecimento (*representação do conhecimento*);
2. aplicar o conhecimento adquirido para resolver problemas (*raciocínio*); e
3. adquirir novo conhecimento através da experiência (*aprendizagem*).

O primeiro requisito envolve uma *representação do conhecimento*, que provém de dois tipos de informação:

- o estado conhecido, representado por factos sobre o que é e o que se conhece; e
- observações ou medições sobre o ambiente, que podem conter ruído e estar sujeitas a erros.

Na *IA,* este conhecimento tem sido representado pelo uso de uma linguagem *simbólica*, através de uma colecção de factos e regras para manipular estes.

O segundo requisito diz respeito ao *raciocínio*, definido como a capacidade para a *resolução de problemas,* que pode ser vista como um problema de *procura.* Uma forma de lidar com a procura é o uso de regras heurísticas, dados, controlo ou raciocínio probabilístico.

Por último, temos a *aprendizagem*, permitindo aumentar o conhecimento através da experiência. A aprendizagem pode envolver duas formas diferentes de processamento de informação: *indutiva* e *dedutiva.* Na primeira forma, padrões e regras são determinadas a partir dos dados e experiência (e.g. *aprendizagem baseada em casos*), enquanto que na última, as regras são utilizadas para criar novos factos (e.g. *prova de teoremas*).

As máquinas de *IA* podem ser comparadas às *RNAs* em três níveis [22]:

1. *Explicação.* Na *IA* clássica, o enfâse está na construção de representações simbólicas, admitindo-se a existência de uma representação mental para a cognição. Por outro lado, a enfâse das *RNAs* está no desenvolvimento de modelos de *Processamento Paralelo Distribuído (PPD)*, assumindo que o processamento depende de um elevado número de unidades, procurando-se uma explicação neurobiológica para os fenómenos cognitivos.

2. *Estilo de Processamento.* Na *IA* convencional, o processamento é sequencial e mesmo quando não existe uma ordem pré-definida, as operações são executadas *passo a passo.* Este estilo é influenciado pela natureza sequencial da linguagem natural e lógica, assim como da estrutura da máquina de *von Neumann.* Em contraste, o paralelismo é essencial às RNAs, sendo a sua fonte de flexibilidade, robustez e imunidade ao ruído.

3. *Estrutura de Representação.*A *IA* clássica utiliza uma linguagem simbólica, com uma estrutura quase linguística, onde surgem expressões complexas a partir de símbolos simples. Ao contrário, nas *RNAs* o conhecimento é armazenado na sua estrutura interna, no valor dos pesos das suas conexões, em valores numéricos.

O movimento conexionista surgiu dentro da *IA* como uma reacção aos modelos simbólicos, considerados demasiado rígidos e especializados; i.e., esperava-se deste novo paradigma a criação de sistemas de aprendizagem mais flexíveis e universais. Contudo, depois de um grande entusiasmo inicial com o modelo conexionista, hoje caminha-se para uma integração de ambos os paradigmas, em sistemas híbridos, adicionando as múltiplas capacidades das *RNA*s, como a adaptividade e robustez, à representação, inferência e universalidade da *IA* simbólica [23].

1.9 O Movimento Conexionista ou uma Perspectiva Histórica

A reflexão filosófica sobre a consciência e qual o órgão que a contém dura desde acerca de dois mil anos, com os filósofos gregos a serem dos primeiros a especular sobre a localização da alma. O conhecimento sobre como o cérebro funciona é resultado da investigação feita nos últimos 100 anos. Ramón y Cajal[3] em 1894 foi o primeiro a propor a teoria dos neurónios [37]. Desde então, numerosos foram os progressos obtidos na compreensão do cérebro humano: os axónios, as dendrites, as sinapses e as activações electro-químicas [30].

A era moderna das *RNAs* surgiu com o trabalho pioneiro de McCulloch e Pitts [1943], descrevendo um cálculo lógico de *RNAs* que unia os estudos neurobiológicos com a matemática lógica. Desde então, um enorme progresso tem sido feito na colaboração de ambas as àreas, com um particular destaque para os anos 80, onde se deu um ressurgimento do interesse em *RNA*s, com contribuições em diversas frentes:

- Grossberg [80] estabeleceu um novo tipo de *SOM* conhecido como *ART* (do inglês *Adaptative Resonance Theory*);

[3]Ramón y Cajal (1852-1934). Médico e histologista espanhol, conhecido pelo seu trabalho sobre o cérebro e nervos, isolando o neurónio e descobrindo como os impulsos nervosos são transmitidos às células do cérebro.

- Em 1982, Hopfield desenvolve um novo tipo de *RNAs* baseado em *RR* com conexões simétricas. No mesmo ano surgem as redes *SOM* do tipo *Kohonen* [17].

- O famoso algoritmo de *Back-Propagation* surgiu em 1986 pelo trabalho de Rumelhart, Hinton e Williams [86], tornando-se no algoritmo de treino mais popular para *RMC*.

- Em 1988, Broomhead e Lowe [5] descrevem as redes *RBF*, fornecendo assim uma alternativa às *RMC*.

- No início dos anos 90, Vapnik e seus colaboradores inventam uma poderosa classe de redes supervisionadas, designadas de *Support Vector Machines*, para a regressão e o reconhecimento de padrões [4].

Hoje em dia procuram-se não só redes mais eficientes como também melhores algoritmos de treino [35]. Por outro lado, espera-se que a aplicação de *RNAs* a outras áreas do conhecimento se generalize, seja à Economia, ao Processamento de Sinal, à Visão por Computador, à Robótica, à Automação ou aos Sistemas Periciais, para além da Estatística [15][10]. Neste último domínio temos a estimação não paramétrica, a aprendizagem por agentes económicos e a modelação de séries temporais [35][24].

Capítulo 2

Redes MultiCamada com Aprendizagem *Back-Propagation*

2.1 A Arquitectura *Feedforward* MultiCamada

As *Redes Feedforward MultiCamada (RFMC),* também conhecidas por *Redes Perceptrão Multicamada*, constituem uma das mais importantes e populares classes de *RNAs*, com um vasto leque de aplicabilidade, utilizáveis em problemas de *memória associativa*, *classificação*, *reconhecimento de padrões*, *optimização* e *regressão* [25].

De um modo geral, as *RFMC* são definidas pelo tuplo (E, I, S, C, F) [30][11] (Figura 2.1):

- um conjunto de *nodos de entrada* (E), onde surgem os estímulos do ambiente;
- um conjunto de *nodos intermédios* (I), unidades internas de processamento que aumentam a capacidade de aprendizagem de tarefas complexas, pela extracção progressiva de mais características;
- um conjunto de *nodos de saída* (S), que devolvem a resposta da rede;

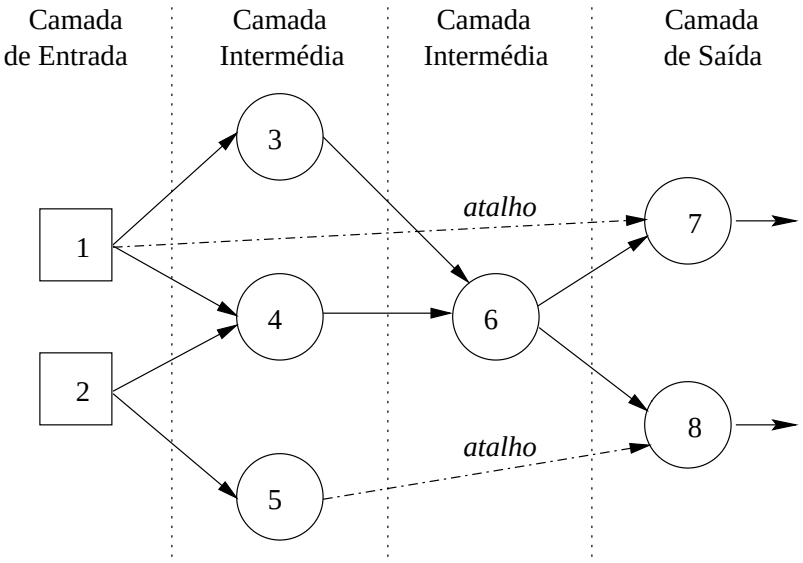

Figura 2.1: Estrutura de uma *RFMC* com a topologia *2-3-1-2*

- um conjunto de *conexões pesadas unidirrecionais* (C), definidos pelo tuplo (i,j,w) ou abreviadamente w_{ij}, em que $i \in I \cup S$, $j \in E \cup I$, $j < i$ e $w \in \Re$; e

- um conjunto de *funções de activação* (F), normalmente do tipo não linear e diferenciável, sendo a função *logística* uma das mais utilizadas (Tabela 1.1).

O sinal de entrada propaga-se assim para a frente, através da rede, camada por camada, não existindo ciclos. Nas redes multicamada temos que as unidades intermédias (I) se dividem em c subconjuntos, ou camadas, $(I_1, I_2, \cdots, I_c)$. É comum representar as camadas pela forma $E - I_1 - \cdots - S$.

Uma *RNA* designa-se *por completamente interligada* quando contém todas as ligações possíveis entre os nodos de duas camadas adjacentes (Figura 1.7). Caso contrário, designa-se por p*arcialmente interligada*. Por vezes existem ligações directas entre as entradas e os neurónios de saída ou que "saltam" camadas (Figura 2.1). Estas conexões designam-se por *atalhos* [26].

A não linearidade, a existência de nodos intermédios e o seu alto grau de conectividade tornam a arquitectura *RFMC* muito poderosa como uma

máquina de aprendizagem. Por outro lado, são estas mesmas características que dificultam uma análise teórica ao processo de aprendizagem [12].

2.2 O Algoritmo de *Back-Propagation*

Dentro dos algoritmos supervisionados, o mais popular e mais usado é o algoritmo de *Back Propagation (BP)* ou seus derivados [11]. Este algoritmo prevalece como um marco para a comunidade das *RNAs,* já que constitui um método eficiente de computação para o treino de *RFMCs*, procurando o mínimo da função de erro no espaço de procura dos pesos, baseiando-se em métodos de gradiente descendente. A combinação de pesos que minimiza a função de erro é considerada a solução para o problema de aprendizagem. Dado que este método exige o cálculo do gradiente, torna-se necessário que a função de erro (ξ) seja contínua e diferenciável, o que acontece quando se usam funções de activação diferenciáveis [30]. O algoritmo de *BP* utiliza dois passos [3][28]:

1. *Em frente,* onde o vector de entrada (x^p) é fornecido aos nodos de entrada, propagando-se em frente, camada por camada, de acordo com as equações 1.1 e 1.2 em que o integrador (g) utiliza a função adição ($\sum$); i.e.,

$$u_i = \sum_j s_j w_{ij} \tag{2.1}$$

   em que $s_0 = 1$ (valor de entrada da conexão de *bias*), $s_1 = x_1^p, \cdots, s_E = x_E^p$, para E nodos de entrada. Em seguida, calcula-se o erro, com base na função de custo, normalmente dada por

$$\xi = \frac{1}{2} \sum_{k \in S} (t_k^p - s_k^p)^2 \tag{2.2}$$

   Durante este passo, os pesos da rede estão fixos.

2. *Retropropagação,* onde o erro é propagado para atrás, desde a saída até os nodos de entrada. De seguida, os pesos são ajustados segundo a

regra de *Widrow-Hoff* (equação 1.3). Para um único peso temos:

$$\Delta w_{ij}(t) = -\eta * \frac{\delta \xi}{\delta w_{ij}}(t) \tag{2.3}$$

em que t representa a ordem da iteração e η a taxa de aprendizagem [28]. As derivadas parciais são calculadas segundo o clausulado

$$\frac{\delta E}{\delta w_{ij}} = \frac{\delta \xi}{\delta s_i} \frac{\delta s_i}{w_{ij}} \tag{2.4}$$

onde

$$\frac{\delta s_i}{w_{ij}} = f'(u_i) sj \tag{2.5}$$

Para se obter $\frac{\delta \xi}{\delta s_i}$, ou seja a influência da saída s_i do nodo i no erro global ξ, é necessário atender ao tipo de nodo

$$\frac{\delta \xi}{\delta s_i} = \begin{cases} -(t_i - s_i) & , \ i \in S \\ \sum_{j \in succ(i)} \frac{\delta \xi}{\delta s_j} f'(u_j) w_{ji} & , \ i \notin S \end{cases}$$

onde $succ(i)$ representa o conjunto dos nodos j da camada com que o nodo i se relaciona (ou estabelece ligações).

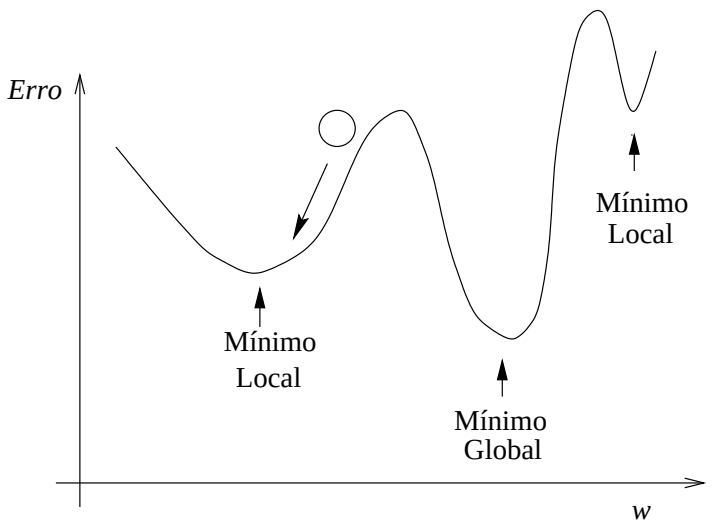

Figura 2.2: Mínimos locais e globais

Antes de se iniciar o treino de uma rede, há que proceder à escolha dos valores iniciais dos pesos associados às ligações entre nodos, que deverão ser pequenos e gerados de forma aleatória. Pode-se então partir para o treino da rede, começando-se por seleccionar um caso de treino, na forma iterativa, ou todos os casos, na forma *em lote*. Normalmente, convém que esta selecção seja aleatória. Em seguida, calcula-se o gradiente e ajustam-se os pesos. Uma *iteração* termina quando todos os casos disponíveis tiverem sido considerados. O processo é terminado por critérios de paragem, por exemplo, quando as mudanças nos pesos e na função de erro forem insignificantes. De notar que o algoritmo pode convergir para um mínimo local (Figura 2.2); na prática, porém, constata-se que quando se parte de um número elevado de casos de treino, esta questão não se coloca, ou então não se assume como um problema sério.

2.3 Alterações ao Algoritmo de *BP*

A apresentação do algoritmo alterou o panorama da investigação em *RFMCs* e desde então têm surgido uma miríade de novos algoritmos de treino. Esta explosão deve-se a duas razões. Por um lado, o algoritmo de *BP* é pesado, de convergência lenta. Por outro lado, baseia-se no gradiente descendente, pelo que todas técnicas de optimização não linear do gradiente podem ser aplicadas [3]. No pior dos casos, a aprendizagem de *RNAs* é *NP-completa*; i.e., o esforço computacional envolvido cresce de um modo exponencial com o aumento dos parâmetros livres (pesos). Assim, existe espaço para a proposta de alternativas que possam acelerar o processo de aprendizagem, embora seja sempre possível enganar o "melhor" método com uma dada tarefa de aprendizagem, desde que se conheca os seus pontos fortes e fracos [30]. Por conseguinte, diversas variantes rápidas do algoritmo de *BP*, que se baseiam no uso de uma topologia fixa, foram propostas nos últimos anos. Todavia, talvez as melhorias realmente significativas advenham do uso de topologias *adaptativas*; i.e., algoritmos que adaptam não só os pesos mas também a topologia interna da rede a uma dada tarefa.

2.3.1 Melhorias ao *BP*

Existem formas simples de acelerar o algoritmo de *BP* [12][30]:

1. *Taxa de Aprendizagem.* Quanto mais pequena for η menores vão ser as mudanças nos pesos das conexões, de modo que a procura do mínimo global será favorecida pelo uso de saltos mais suaves. O problema é que desta forma tem-se uma aprendizagem mais lenta. Por outro lado, se se aumentar em demasia o valor de η, então os elevados saltos nas mudanças dos pesos poderão provocar instabilidade no treino (e.g. movimento oscilatório). Uma forma de aumentar a taxa de aprendizagem sem provocar instabilidade é alterar a regra de aprendizagem de modo que esta inclua um termo de *momentum*:

$$\Delta w_{ij}(t) = -\eta * \frac{\delta \xi}{\delta w_{ij}}(t) + \mu \Delta w_{ij}(t-1) \tag{2.6}$$

   onde μ representa a constante de *momemtum,* usualmente dada por um valor positivo escolhido do intervalo $[0, 1]$. Este termo de *momentum* funciona como uma mémoria que acelera descidas do gradiente, tendo um esfeito estabilizador em situações oscilatórias. Outra forma distinta para a lidar com este problema reside no uso diferentes taxas de aprendizagem, uma por cada nodo. LeCun [20] sugere a seguinte formula $\eta = \frac{1}{\sqrt{z}}$, para um nodo com z conexões.

2. *Modo de Treino.* Durante a aprendizagem, uma *iteração* corresponde a uma completa apresentação de todos os exemplos de treino à rede. Convém que a apresentação seja aleatória, de forma a que não exista uma tendência para valorizar certos casos de treino. Esta apresentação pode ser efectuada de duas formas:

   - *modo sequencial*, onde se calculam os ajustamentos aos pesos logo após a escolha de um exemplo de treino; e
   - *modo em lote* ou *por iteração*, onde os ajustamentos aos pesos só são efectuados após a apresentação de todos os casos de treino.

O primeiro modo exige uma menor memória local de armazenamento, para além da apresentação aleatória tornar difícil que o algoritmo fique preso em mínimos locais, sendo deveras útil quando o conjunto de dados de treino é elevado e existe bastante informação redundante. Em contraste, o modo *em lote* melhora a estimativa do vector de gradiente, sendo também um processo mais facil de se paralelizar.

3. *Funções de Activação.* Embora seja habitual o uso de funções de activação com saídas dentro do intervalo $[0, 1]$, como no caso da função *logística* padrão, existem evidências teóricas e empíricas que favorecem saídas dentro do domínio $[-1, 1]$. A função *tanh* (Tabela 1.1) é um exemplo deste tipo de funções. A função logística também pode ser adaptada a este intervalo adoptando a seguinte fórmula:

$$s_i = \frac{1 - \exp(-u_i)}{1 + \exp(-u_i)} \tag{2.7}$$

Outro aspecto está relacionado com o número de operações de vírgula flutuante, que poderão ser reduzidas pelo uso de funções de activação mais simples, por exemplo, para a função logística existe uma versão desta que não necessita do cálculo de expoentes [8]:

$$s_i = \frac{1}{(1 + |u_i|)^2} \tag{2.8}$$

4. *Maximizando o Conteúdo de Informação.* Existem heurísticas simples para melhorar a informação fornecida à rede, por exemplo [12]:

   - escolha do exemplo que origina o maior erro de treino; e
   - escolha do exemplo mais distinto em relação aos exemplos já fornecidos.

5. *Pré-Processamento de Dados.* Em muitas situações torna-se útil pré-processar os dados de um problema antes de estes serem alimentados à rede. A ideia é ajustar os dados de alguma forma tal que a rede

os possa tratar de forma eficiente. Existem diversas formas de pré-processamento que serão descritas adiante (Secção 2.5).

6. *Escolha dos Pesos Iniciais.* Uma boa escolha dos valores iniciais dos pesos pode acelerar bastante o processo de aprendizagem. O problema que surge é: O que é uma boa escolha? Valores elevados podem originar uma saturação da rede, atrasando o processo de aprendizagem. No entanto, valores demasiado pequenos podem levar a que o algoritmo de *BP* opere numa área demasiado plana por volta da origem da superfície de erro. Assim quer valores demasiado elevados ou demasiado pequenos devem ser evitados. Uma boa estratégia é gerar valores aleatórios com uma média de zero que dependem do número de conexões de um nodo. Por exemplo, Gallant [11] sugere o intervalo $[\frac{-2}{z}; \frac{2}{z}]$, para um nodo com z entradas.

2.3.2 Algoritmos de Adaptação Local

As variantes do algoritmo de *BP* podem ser classificadas em duas categorias: de adaptação global ou local [28]. Os primeiros utilizam um conhecimento global do estado completo da rede, como a direcção de todo o vector de actualização dos pesos. Em contraste, os últimos são baseados na informação específica de um peso, como o comportamento temporal da sua derivada parcial. Este tipo de estratégia é mais próxima ao conceito de *RNAs,* sendo mais facilmente paralelizável. Para além disso, estas técnicas tendem a ser mais eficazes e robustas, apesar de usarem menos informação. Nesto contexto, importa descrever dois algoritmos importantes de adaptação local: o *QuickProp* e o *RPROP*:

QuickProp

Fahlman [9] propôs uma variante do *BP* chamada *QuickProp (QP)*, que parece levar a uma rápida convergência em muitos casos, sendo por isso bastante utilizado. Trata-se de um método de segunda ordem combinado com algumas heurísticas. Parte de duas assunções que podem não se verificar

sempre [3]:

1. o erro *versus* a curva do peso para cada conexão pode ser aproximado por uma parábola aberta para cima; e

2. a mudança no peso não é afectada por todos outros pesos que estão a ser alterados.

O algoritmo comporta-se como o algoritmo tradicional de *BP*, utilizando uma combinação da mesma regra de aprendizagem (equação 2.3) e com a seguinte regra de aprendizagem:

$$\Delta w_{ij}(t) = \frac{D(t)}{D(t-1) - D(t)} \Delta w_{ij}(t-1) \tag{2.9}$$

em que $D(t) = \frac{\partial \xi(t)}{\partial w_{ij}(t)}$. Para além disso, de modo a evitar elevadas variações, o salto actual dos pesos é limitado a um máximo de ν vezes em relação ao salto anterior. Assim, este algoritmo contém dois parâmetros: η, taxa de aprendizagem para o gradiente descendente e ν, um limitador do tamanho do salto, tendo o valor padrão de 1,75 [28].

RPROP

O nome do algoritmo, ***RPROP***, vem do inglês "***R****esilient back****PROP****agation***" [27], e trata-se de uma variante sofisticada do algoritmo de *BP* [29].

A regra de cálculo é dada por :

$$\Delta_{ij}(t) = \begin{cases} \eta^+ \times \Delta_{ij}(t-1) & , \quad \frac{\delta \xi}{\delta w_{ij}}(t-1) \times \frac{\delta \xi}{\delta w_{ij}}(t) > 0 \\ \eta^- \times \Delta_{ij}(t-1) & , \quad \frac{\delta \xi}{\delta w_{ij}}(t-1) \times \frac{\delta \xi}{\delta w_{ij}}(t) < 0 \\ \Delta_{ij}(t-1) & , \quad \frac{\partial \xi}{\partial w_{ij}}(t-1) \times \frac{\partial \xi}{\partial w_{ij}}(t) = 0 \end{cases} \tag{2.10}$$

para $0 < \eta^- < 1 < \eta^+$. Baseados em considerações de ordem teórica e empírica, estes factores foram fixados em $\eta^+ = 1,2$ e $\eta^- = 0,5$. Os pesos das

ligações entre neurónios podem então ser alterados de acordo com a relação:

$$\Delta w_{ij}(t) = \begin{cases} -\Delta_{ij}(t) & , \quad \frac{\delta \xi}{\delta w_{ij}}(t) > 0 \\ +\Delta_{ij}(t) & , \quad \frac{\delta \xi}{\delta w_{ij}}(t) < 0 \\ 0 & , \ \frac{\delta \xi}{\delta w_{ij}}(t) = 0 \end{cases} \tag{2.11}$$

Estudos comparativos mostraram que este algoritmo converge mais rapidamente que outros algoritmos do género [28]. Além disso, dependende apenas de dois parâmetros, Δ_0 e Δ_{max}, cuja escolha de valores não é muito crítica, sendo um algoritmo deveras robusto; i.e., regra geral, não é necessária nenhuma alteração aos valores aconselhados para Δ_0 e Δ_{max} ($\Delta_0 = 0,1$ e $\Delta_{max} = 50,0$) [29].

2.4 Capacidades e Limitações das *RFMCs*

Uma rede *RFMC* treinada por *Backpropagation* pode ser vista como uma forma prática para efectuar uma qualquer correspondência não linear. A questão que surge é: dada uma função g desconhecida que faz uma correspondência entre padrões de um espaco n-dimensional para um espaço m-dimensional, $g : \Re^n \rightarrow \Re^m$, será possível encontrar uma *RFMC* que realize a correspondência exigida ou pelo menos se aproxime dela com alguma eficácia? A resposta vem do *teorema universal de aproximação de funções* que estipula que uma camada intermédia é suficiente para uma *RFMC* computar uma aproximação de uma qualquer função contínua, embora o teorema não afirme que esta estrutura seja óptima em termos de tempo de aprendizagem e generalização [12]. Também está provado que com duas camadas intermédias até funções descontínuas podem ser representadas [32].

Outro aspecto importante diz respeito ao tempo de aprendizagem. A aprendizagem implica a procura dos elementos desconhecidos de uma *RNA,* normalmente pelo ajuste dos pesos. Ora, a aprendizagem numa rede com 100 pesos é bastante mais pesada em termos computacionais do que a de uma rede com 10 pesos, sendo uma relação bem maior do que o factor 1 : 10 poderia sugerir. Seria muito útil que o tempo de aprendizagem fosse limitado

por uma função polinomial sobre o número de variáveis. No entanto, tal nao se sucede. Está provado que o problema geral de aprendizagem em *RNAs* é intractável; i.e., não pode ser resolvido de forma eficiente para todas as instâncias. Não é conhecido um algoritmo que consiga realizar a aprendizagem num tempo polinomial, sendo improvável que tal algoritmo venha a existir. Assim, diz-se que em geral o problema de aprendizagem em *RNAs* é NP-completo [30]. Todavia, um problema que seja de difícil aprendizagem para uma dada estrutura de rede poderá ser de fácil aprendizagem para outra. Assim, uma das possibilidades para ultrapassar a aprendizagem NP-completa das *RNAs* reside no uso de arquitecturas adaptativas [19].

2.5 Pré-processamento dos Dados

O pré-processamento dos dados deve ser seriamente considerado antes de treinar uma rede. Embora não seja essencial em certos casos, torna-se vital noutros. São várias as operações em que se podem desdobrar o pré-processamento [1][34]:

1. *Verificação da Integridade dos Dados.* Trata-se de um procedimento de validação dos dados, verificando-se se existem erros de alguma espécie.

2. *Representação dos Dados.* Esta fase envolve uma conversão de dados. Dado que as *RNAs* apenas lidam com números, os dados terão de ser codificados numa representação binária, discreta ou real. Por outro lado, uma variável pode ser representada por um ou mais nodos. Por exemplo, em problemas de classificação, com M classes, é comum representar uma saída por um vector binário M-dimensional. A classe k poderá então representada por um vector em que o k elemento toma o valor 1 (verdadeiro) e os restante elementos o valor 0 (falso).

3. *Escalonamento dos Dados.*O objectivo deste procedimento é realizar uma transformação nos dados de modo a acelerar e melhorar o processo de aprendizagem. Este escalonamento depende do tipo de dados:

(a) *Entradas.* O escalonamento das variáveis de entrada tem diversos efeitos conforme os distintos algoritmos de aprendizagem. De um modo particular, algoritmos de gradiente descendente, como o conhecido *BP*, são bastante sensíveis ao escalonamento. Assim, cada variável de entrada deve ser escalonada de forma a que a sua média sobre o conjunto de casos de treino esteja à volta do valor zero, pelo que o escalonamento para o intervalo $[-1, 1]$ funcionará melhor do que para o intervalo $[0, 1]$. Existem duas formas eficientes de realizar este escalonamento:

- *Média Zero e Desvio Padrão 1*

$$y = \frac{x - \overline{x}}{std(x)} \tag{2.12}$$

onde $\overline{x}$ denota a média de x e $std(x)$ o desvio padrão de x; i.e., $std(x) = \sqrt{\frac{\sum_i (x_i - \overline{x})^2}{n-1}}$, para n exemplos de treino.

- *Valor Mínimo -1 e Valor Máximo 1*

$$y = \frac{x - (max + min)/2}{(max - min)/2} \tag{2.13}$$

em que min e max representam o mínimo e máximo da variável x.

(b) *Saídas.* Existem duas fortes razões para escalonar as saídas. Em primeiro lugar, quando se usa mais de uma saída e se a função de erro é sensível à escala, como acontece no caso da aprendizagem do gradiente descendente, então a diferença de escalas entre as saídas pode afectar a forma como a rede aprende. Por exemplo, se uma saída tem valores entre 0 e 1 enquando outra tem valores entre 0 e 1000000, então o algoritmo de treino irá dispender a maior parte do seu esforço na aprendizagem da segunda saída. Assim, as saídas que têm a mesma importância devem ser escalonadas para a mesma escala. Neste caso, deve-se proceder a um escalonamento com a mesma escala ou desvio padrao (usando por exemplo a

equação 2.12). Em segundo lugar, pode-se querer ajustar as variáveis de saída aos valores do contradomínio da função de activação (e.g. a função *logística* produz valores de saída dentro do domínio $[0, 1]$). Neste caso, é importante conhecer os limites máximos e mínimos da variável em vez dos valores máximos e mínimos dos casos de treino. Neste caso poderá se utilizar a seguinte fórmula:

$$y = \frac{(x - L_{min})(B - A)}{L_{max} - L_{min}} + A \tag{2.14}$$

para um escalonamento no domínio $[A, B]$, sendo L_{max} e L_{min} os limites máximo e mínimo da variável. Se a variável de saída limites desconhecidos então é preferível utilizar a função *linear* (ou outra não limitada) para os nodos de saída.

4. *Redução da Dimensão.* O número de elementos num vector de entrada, sua *dimensão*, pode atingir proporcões demasiado elevadas, principalmente quando não existe um grande conhecimento sobre a natureza e relevância destes. Em geral, redes maiores têm uma aprendizagem mais díficil. Uma das formas de aliviar este problema consiste na redução da dimensão dos vectores de entrada, extraindo-se a informação relevante. Existem várias técnicas para o fazer:

   (a) *RFMC para Compressão.* Uma forma simples consiste no uso de uma *RFMC* com uma camada intermédia, que contém menos nodos do que os nodos de entrada, em que apenas conexões entre camada adjacentes são permitidas. Esta rede contém um igual número de nodos de entrada e de sáida, que são iguais à dimensão original do problema. Os vectores de saída também são iguais aos vectores de entrada, pelo que a rede é treinada para reproduzir a sua própria entrada. Devido ao desenho em funil, a camada intermédia aprende a comprimir os dados. No final do treino, existirá um erro residual, que depende do grau de compressão. Caso este não possa ser reduzido a um valor aceitável, é sinal de que a compressão não se adequa ao problema. Por fim, a camada de saída é

removida e os pesos entre a camada de entrada e intermédia são fixos. A estrutura resultante pode então ser acrescentada a uma rede de dimensão reduzida, que irá aprender a tarefa original.

(b) *Análise de Componentes Principais (ACP).* O critério crucial na selecção das variáveis é a sua relevância para o objectivo pretendido. Ora, certas variáveis podem influênciar outras, existindo por isso uma duplicação de informação ou *correlação.* A separação da informação relevante de cada variável facilita a aprendizagem. Este processo, designado de *descorrelação,* pode ser visto como uma rotação dos eixos cartesianos das entradas. Uma técnica poderosa para a descorrelação consiste no uso da *ACP.*

(c) *Transformada Wavelet.* Esta fornece uma nova representação dos dados, com vectores formados por funcões *wavelet,* cujos componentes correspondem aos diversos graus de detalhe dos dados originais. A compressão de dados é então obtida pelo uso de um subconjunto destes componentes.

5. *Filtragem de Ruído.* Por vezes torna-se útil o uso de técnicas de filtragem para a eliminação de ruído, suavizando a função de apredizagem, facilitando o treino. Como exemplo temos a *análise espectral,* uma técnica muito usada na manipulação de imagens, onde um sinal de entrada é decomposto em ondas sinusoidais, pela *transformada de fourier.*

2.6 Generalização

Diz-se que uma *RNA* possui uma boa *generalização* quando a correspondência entre entradas e saídas é correcta (ou próxima disso) para *dados de teste*, retirados da mesma população, nunca antes utilizados na criação ou treino da rede. O processo de aprendizagem pode ser visto como um problema de ajustamento de curvas ou de aproximação de funções, onde a rede tenta efectuar uma boa interpolação não linear dos dados [30]. A Figura 2.3 mostra como podem ocorrer duas generalizações distintas para o mesmo conjunto de dados de treino. Aqui, uma boa generalização ocorre com a curva A, com

um erro mínimo para os dados de teste. O mesmo já não se sucede com a curva B, que origina um erro maior para os casos de teste, isto apesar de apresentar um menor erro para os dados de treino. Tal fenómeno, designado de *overfitting,* ocorre quando uma *RNA* memoriza em demasia os exemplos de treino, tratando-se de um dos problemas mais sérios relacionados com o uso de *RNAs* [33]. Durante o processo de aprendizagem, a rede pode captar certas características, como o ruído, que estão presentes nos dados de treino mas não na função implícita a ser aprendida. Este exemplo ilustra os dois objectivos contraditórios da aproximação funcional. Por um lado temos a minimização do erro de treino, pelo outro temos a minimização do erro para as entradas desconhecidas. Assim, uma *RNA* que seja treinada em demasia perde capacidade para generalizar.

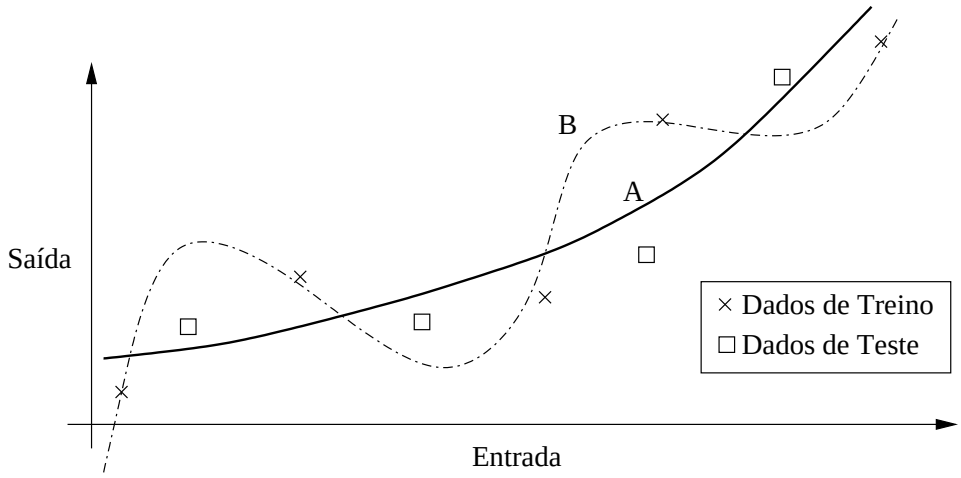

Figura 2.3: Generalização e *overfitting*

A generalização é influenciada por três factores [12][34]:

1. *A complexidade do problema a ser aprendido.* Trata-se de um factor de difícil controlo. As entradas devem conter informação suficiente para permitir a obtenção das saídas desejadas; i.e., tem de existir uma função matemática com algum grau de precisão que relacione as entradas com as saídas. Por outro lado, convém que esta função seja *suave*; i.e., pequenas alterações nas entradas devem provocar pequenas alterações nas saídas, para a maior parte dos casos. Por vezes, uma transformação

não linear nas entradas pode melhorar a suavidade (e.g. transformação logarítmica).

2. *Os casos de treino.* O seu tamanho deve ser suficientemente grande, com exemplos (ou *amostras* na terminologia estatística) que sejam suficientemente representativas devem serquão representativos do ambiente (ou *população*). A generalização é sempre efectuada a partir de dois tipos de duas situações: *interpolação* e *extrapolação.* No primeiro caso, um valor é determinado a partir da informação dos valores vizinhos. A segunda situação engloba tudo o resto, ou seja, casos fora do domíno dos dados de treino. Enquanto que a interpolação pode ser efectuada com relativa acuidade, o mesmo já não se passa com a extrapolação, notoriamente menos fiável.

3. *A arquitectura da RNA*; i.e., o número de parametros livres que contém (pesos) e sua capacidade de aprendizagem ou complexidade. Uma rede que contenha uma insuficiente complexidade irá falhar na aproximação da função a aprender. Por outro lado, uma rede demasiado complexa irá fixar o ruído, provocando *overfitting.* A Figura 2.4 mostra uma variação típica do erro de uma *RNA* com uma camada intermédia, para os casos de treino e de teste, com o incremento do número de nodos intermédios. À medida que estes aumentam o erro de treino diminui. A dada altura, a curva de erro para os casos de teste inflecte, originando-se uma perda de generalização.

A melhor forma de evitar o *overfitting* é utilizar uma quantidade elevada de casos de treino. Quando este número for pelo menos 30 vezes superior ao número de conexões, então é muito improvável que ocorra overfitting. O problema é que nem sempre existem muitos casos de treino disponíveis e não se deve reduzir o número de conexões de um modo arbitrário, devido a problemas de insuficiência de complexidade. Dado um número fixo de dados de treino existem duas grandes alternativas eficientes para controlar a complexidade da rede [33][34]: *regularização* e *selecção de modelos.*

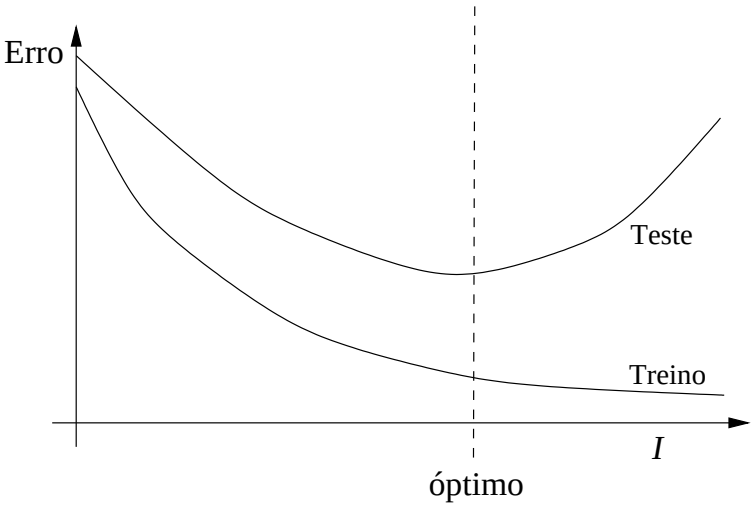

Figura 2.4: Erro típico com um aumento do número de nodos intermédios

2.7 Regularização

A regularização baseia-se num controlo dos valores dos pesos das conexões para obter uma boa generalização, envolvendo o uso de restrições ou penalidades sobre estes, de modo que a rede aprenda funções mais suaves. Entre os diversos métodos de regularização temos [33][34]:

2.7.1 Dacaimento dos Pesos

A ideia é acrescentar uma penalidade à função de erro, de modo a reduzir os pesos elevados, visto que estes prejudicam a generalização, originando funções irregulares, por vezes na vizinhança de discontinuidades. Noutras palavras, pesos elevados causam uma excessiva variância nas saídas [2]. Normalmente esta penalidade é calculada pela expressão:

$$d \times \sum w_{ij}^2 \tag{2.15}$$

onde d representa a *constante de decaimento,* cuja escolha é crucial para uma boa generalização.

2.7.2 Adição de Ruído

O objectivo é acrescentar deliberadamente ruído artificial às entradas durante o treino. Esta estratégia funciona porque a maior parte das funções a ser aprendidas pela rede são suaves. Assim, em cada iteração do algoritmo de treino, novos casos de treino são criados, pelo acrescento de quantidades de ruído. Este nem deve ser demasiado pequeno, produzindo pouco efeito, nem demasiado grande, pois obviamente eliminará a função implícita a ser aprendida. Este ruído é produzido por um gerador de números aleatórios, usualmente através de uma distribuição normal com média 0 e desvio padrão s, cujo valor deverá ser estimado de algum modo. Por exemplo, de modo a que seja menor do que o erro de generalização, medido por um estimador.

2.7.3 Paragem Antecipada

Trata-se de um dos mais populares métodos de regularização, onde os dados de treino são divididos em dois tipos de casos: *de treino* e *de validação*. Os primeiros são utilizados na aprendizagem da rede, enquando que os últimos são utilizados para aferir da qualidade da aprendizagem; i.e., para estimar o erro de generalização. De notar que podem ser utilizados novos *casos de teste* para medir o desempenho da rede após o treino. Durante a fase de treino, calcula-se o erro de validação de forma periódica, parando-se quando este começa aumentar. Todavia, esquemas de paragem mais elaborados têm de ser adoptados, dado que a função de erro pode apresentar diversos mínimos locais. Por exemplo, Prechelt [26] defende o uso de três critérios de paragem:

1. *Falha de Progresso do Treino.* A ideia é tentar avaliar o progresso do treino; i.e., qual a diminuição do erro sobre os casos de treino, ξ_{tr}, durante uma dada *faixa de treino*, de k iterações. A função de progresso, avaliada em cada k iterações, toma a forma:

$$P_k(t) = 1000 * (\frac{\sum_{t' \in t-k+1 \ldots t} \xi_{tr}(t')}{k * min_{t' \in t-k+1 \ldots t} \xi_{tr}(t')} - 1) \qquad (2.16)$$

   O progresso no treino é elevado nas fases instáveis, onde o erro nos casos de treino sobe em vez de descer. No entanto, tende para zero a

longo prazo, a não ser que o treino se torne oscilante. O treino é parado se $P_k(t) < \beta$, em que β é uma medida de erro de estado estacionário.

2. *Perda de Generalização.* Esta ocorre sempre que há uma inversão de sinal nos valores da derivada da função de erro para os casos de validação, ξ_{va}, passando estes de negativos a positivos. A função de avaliação, também medida de k em k iterações, toma a forma:

$$G_k(t) = 100 * (\frac{\xi_{va}(t)}{min_{t' \leq t}\xi_{va}(t')} - 1) \tag{2.17}$$

Uma grande perda de generalização é uma boa razão para se parar o treino, pelo que o treino termina se $G_k(t) > \alpha$, onde α denota a perda de poder de generalização aconselhável

3. *Número Máximo de Iterações.* Este critério é aplicado quando os anteriores critérios de paragem falham, de modo a garantir que o treino termine.

A paragem antecipada é bastante utilizada porque é simples e rápida, podendo ser aplicada a *RNAs* com um grande número de conexões. Todavia, possui algumas desvantagens. Em primeiro lugar, é bastante sensível à forma como é feita a divisão entre casos de treino e de validação; i.e., que e quantos casos devo usar em cada categoria. Por outro lado, não aproveita toda informação disponível para a aprendizagem. Daí que certos autores defendam o uso de estimadores mais sofisticados para o erro de generalização [16].

2.8 Selecção de Modelos

A regularização diminui o efeito de *overfitting* pelo estímulo na aprendizagem de funções suaves. No entanto, utiliza uma estrutura fixa, que deve ser especificada em avanço pelo utilizador. Embora se possa utilizar uma grande estrutura, com um grande número de nodos intermédios, na prática, a optimização dos pesos torna-se de díficil ajuste, exigindo um grande esforço computacional. Mais ainda, em geral, são métodos que exigem um delicado

balanço, controlado por um (ou mais) parâmetro(s) de regularização. Mais recentemente, métodos *Bayesianos* têm sido incorporados na regularização, para eliminar alguns destes problemas. Trata-se de uma aproximação promissora embora ainda pouco desenvolvida. Para além disso, estes métodos assumem certas distribuições que podem falhar quando o número de conexões da rede é grande, quando comparado com o tamanho dos exemplos de treino [19]. Uma alternativa distinta baseia-se na procura da topologia correcta de uma *RNA*, em termos do número de conexões, número de nodos e camadas intermédias. Os defensores desta estratégia argumentam que é mais fácil adaptar a complexidade da rede ao problema a ser resolvido. Assim, um problema que seja de difícil aprendizagem para uma rede poderá ser facilmente aprendido por outra rede.

A abordagem estatística passa pela estimativa do erro de generalização para cada um dos modelos, ou topologias de rede, escolhendo-se o modelo que minimiza essa estimativa.

2.8.1 Métodos de Estimação do Erro de Generalização

Existem diversos métodos para estimar a capacidade de generalização de uma *RNA* [7][16]:

1. *Estatísticas Simples.* Diversas métricas foram desenvolvidas, tendo em conta modelos lineares, baseando-se em assunções sobre as amostras de dados. Entre estas, temos [10]:

   - *Critério de Informação de Akaike*, conhecido por *AIC*, que tende a admitir o *overfitting* em *RNAs*:

     $$AIC = n \ln(SSE/n) + 2p \tag{2.18}$$

     onde SSE representa o somatório do quadrado dos erros para todos os exemplos de treino, n representa o número de exemplos de treino e p o número de parâmetros livres da rede, ou seja, o número de pesos ($\#C$); e

- *Critério de Informação de Bayes*, designado por *BIC* ou *SBC,* que normalmente funciona bem com *RNAs*:

$$BIC = n\ln(SSE/n) + p\ln(n) + p? \tag{2.19}$$

2. *Validação com Divisão da Amostra.* O método mais popular para a estimação do erro de generalização, geralmente associado à paragem antecipada (Secção 2.7.3), baseia-se numa divisão dos dados do problema em *casos de treino*, para a rede aprender, e *casos de validação,* para estimar o erro de validação. Como ponto fortes temos a sua simplicidade e rapidez embora produza uma redução efectiva dos dados disponíveis para treino.

3. *Validação Cruzada.* Trata-se de um melhoramento do método anterior, que permite utilizar todos os casos disponíveis. Na validação cruzada *k-desdobrável,* os dados (P) são divididos em k subconjuntos mutualmente exclusivos $(P_1, P_2, \cdots, P_k)$ de comprimentos aproximadamente iguais. A rede é então treinada e testada k vezes. Em cada tempo $t \in \{1, 2, \cdots, k\}$, a rede aprende a partir do conjunto com $P \setminus P_t$, estimando-se o erro de validação a partir dos dados restantes, P_t. O erro final de generalização é então dado pela média dos erros de validação obtidos durante k vezes. Os valores de k podem variar entre 2 e n, embora o valor 10 seja bastante popular. Um valor pequeno para k origina uma má estimativa da generalização enquanto que um valor elevado aumenta consideravelmente o esforço de computação, visto que a rede é treinada k vezes para cada iteração do algoritmo de treino. A validação cruzada é notavelmente superior à validação com divisão da amostra para pequenos conjuntos de exemplos de treino. Todavia, tal realização é conseguida à custa de um maior esforço computacional.

4. *Bootstrapping.* Dado um conjunto de dados com n elementos, uma amostra de *bootstrap* é criada pela amostragem uniforme de n instâncias a partir dos dados, com repetição. A probabilidade de uma dada instância não ser escolhida após n amostragens é $(1 - 1/n)^n \approx 0,368$.

Assim, o número esperado de instâncias distintas a serem amostradas é então $0,623n$. O método consiste em utilizar a amostra de *bootstrap* para treinar a rede e as intâncias restantes para validação. A estimativa é então dada por:

$$\frac{1}{b}\sum_{i=1}^{b}(0,632\xi_{va_i} + 0,368\xi_{tr}) \tag{2.20}$$

onde ξ_{va_i} e ξ_{tr_i}denotam respectivamente a estimativa do erro de validação e de treino para cada amostra i. Os valores típicos para b variam entre 20 e 200. O *bootstrapping* parece funcionar melhor que a validação cruzada em muitas situações, embora também à custa de ainda mais esforço computacional.

2.8.2 Escolha da Topologia de Rede

A topologia ideal (em termos de camadas, nodos e conexões) de uma *RMFC* depende fundamentalmente do problema a ser resolvido. Normalmente tenta-se reduzir o número de escolhas (ou espaço de procura) adoptando certos pressupostos, como o uso de redes completamente interligadas, evitando-se neste caso a preocupação sobre que conexões devem existir. Também é comum adoptar-se apenas uma camada intermédia, ficando a escolha dos nodos intermédios a ser feita decidida por regras empíricas. Trata-se de uma estratégia que resulta em diversas situações, principalmente se não existe uma grande exigência, em parte devido ao enorme poder das *RMFC*. Estratégias mais elaboradas passam por procedimentos de *tentativa-e-erro,* utilizando uma procura aleatória, embora seja altamente desejável a utilização de métodos eficientes e automáticos, com base em heurísticas (e.g. *hill-climbing* ou *algoritmos genéticos*) [32]. Neste caso existem duas importantes estratégias: *algoritmos de corte* e *algoritmos construtivos* [25].

Escolha do número de camadas intermédias

Escolha do número de nodos intermédios

Algoritmos de Corte

Os algoritmos de corte envolvem a remoção de nodos e conexões desnecessárias, segundo oprincípio de *Ockham*[1] ; i.e., tentando-se obter a rede mais simples que explique um dado problema. Estes algoritmos começam com uma *RNA* que seja suficientemente grande para garantir um treino com sucesso. A seguir, algumas conexões e/ou nodos são removidos e a rede volta a ser treinada. Este processo continua até que uma solução aceitável seja encontrada, devolvendo-se no final a rede mais simples que garante uma boa aprendizagem. Independentemente do método de corte, é díficil avaliar a qualidade da rede. Tal critério deve depender do número de camadas intermédias, nodos, conexões e restrições da aplicação (e.g. requisitos de tempo real podem implicar um limite para a quantidade de camadas intermédias). Outro critério importante a ter em conta é a capaciade de generalização da rede. Em geral, a avaliação da rede deve depende depender do tempo de treino, do tamanho da rede e a da sua capacidade de generalização.

Os principais decisões de um algoritmo de corte envolvem a escolha da(s) unidade(s) a ser cortada(s), quando efectuar o corte e quando parar o treino. Existem vários algoritmos de corte que diferem fundamentalmente na primeira decisão, a parte mais crítica do processo. Esta escolha, sobre o que cortar, é feita com base em heurísticas (e.g. a heurística simples da remoção das conexões com os pesos mais pequenos) [36].

Algoritmos Construtivos

Trata-se de uma estratégia oposta, seguindo-se outra direcção de procura: começa-se com uma rede simples e depois nodos intermédios e conexões adicionais são acrescentadas a esta, até que uma solução satisfatória seja en-

[1]William of Ockham – 1349. Filósofo e frade franciscano inglês, conhecido pela defesa da aplicação da regra económica ontologica à teologia: "entidades não são multiplicadas para além da necessidade", tão frequentemente utilizada que se tornou famosa como a *navalha de Ockham*.

contrada. Existem várias vantagens desta aproximação sobre a anterior. Em primeiro lugar, é fácil especificar a topologia inicial da rede, enquanto que nos algoritmos de corte não se sabe quão grande deverá ser a rede. Depois, soluções mais pequenas são procuradas no início, sendo assim mais económicos em termos de custos de computação. Mais ainda, dado que diversas redes de tamanhos distintos podem ser capazes de constituir soluções aceitáveis, os algoritmos construtivos têm uma maior probabilidade de encontrar redes mais pequenas do que os de corte. Outra motivação para este tipo de algoritmo está relacionada com o tempo de aprendizagem. A flexibilidade destes torna as redes em máquinas universais de aprendizagem; i.e., capazes de resolver em tempo polinomial qualquer classe de problemas que possa ser aprendida em tempo polinomial por outros métodos. Como dificuldades mais importantes desta estratégia tem-se a altura em que o processo deve terminar, as heurísticas de procura (que podem ser sub-óptimas) e cuidados de generalização da rede.

Os diversos algoritmos construtivos diferenciam-se fundamentalmente pela sua *estratégia de construção* ou *de transição de estado*, existindo duas importantes categorias: *de um só valor* ou *multi-valor*. A primeira estratégia caracteriza-se pelo uso de apenas um único estado possível de transição, ao contrário da procura multi-valor, onde existem vários estados possíveis, designados por *candidatos*, sendo cada um estruturalmente distinto dos outros. Os algoritmos construtivos também se podem distinguir pelo tipo de treino aplicado à nova estrutura, que pode ser *total*, treinando-se todos os pesos da rede, ou *parcial,* congelando-se os valores dos pesos antigos e treinando-se apenas as conexões dos novos nodos. Trata-se de uma aproximação ávida, mais eficiente, mas que poderá não resultar sempre. Assim, por vezes aplicam-se estratégias mais complexas (e.g. ajustamentos cíclicos dos pesos que afectam as novas conexões inseridas) [19].

2.9 Aplicações

As *RFMCs* têm sido empregadas com sucesso em diversas domínios, incluindo Engenharia, Direito, Ciencias da Computação, Processos de Controlo,

Robótica e Automação, Estatística, Medicina, Produção, Economia, entre outros. Esta grande popularidade das *RFMCs* advém das suas capacidades de correspondência não linear entre padrões. Normalmente, para um bom conjunto de dados de treino, uma rede com apenas uma (ou quanto muito duas) camada(s) intermédia(s) serve para aprender uma dada tarefa com mestria. Como exemplos de aplicações de *RFMCs* temos [25].

- *Classificação e Diagnóstico*
  - Classificação de células para um diagnóstico de cancro
  - Identificação de falhas na comutação de circuitos telefónicos
- *Controlo e Optimização*
  - condução de veículos autónomos
  - controladores inteligentes para a produção de ferro
- *Regressão/Previsão*
  - atribuição de montantes de crédito
  - previsão de séries temporais
  - previsão da evolução de acções da bolsa
  - previsão de resultados desportivos
- *Reconhecimento de Padrões*
  - reconhecimento de escrita manual
  - identificação automática de indivíduos

Bibliografia

[1] E. Azoff. *Neural Network Time Series Forecasting of Financial Markets.* John Wiley & Sons, 1994.

[2] P. Barlett. For valid generalization, the size of the weights is more important that the size of the network. *Advances in Neural Information Processing Systems*, 9:134–140, 1997.

[3] N. Bose and P. Liang. *Neural Network Fundamentals with Graphs, Algorithms and Applications.* McGraw-Hill, USA, 1996.

[4] B. Boser, I. Guyon, and V. Vapnik. A training algoritm for optimal margin classifiers. In *Fifth Annual Workshop on Computational Learning Theory*, pages 144–152. San Mateo, CA: Morgan Kaufmann, 1992.

[5] D. Broomhead and D. Lowe. Multivariable functional interpolation and adaptative networks. *Complex Systems*, 2:321–355, 1988.

[6] G. Carpinter and S. Grossberg. The ART of Adapttive Pattern Recognition by a Self-Organizing Neural Network. *Computer*, 21:77–88, 1988.

[7] B. Efron and R. Tibshirani. *An Introduction to the Bootstrap.* Chapman & Hall, USA, 1993.

[8] D. Elliot. a better activation function for artificial neural networks. Technical report tr 93-8, Institute for Systems Research, University of Maryland, Maryland, USA, January 1993.

[9] S. Fahlman. Faster Learning Variations on Back-Propagation: An Empirical Study. in: [Touretzky et al. 1989], 1989.

[10] J. Faraday and C. Chatfield. Times Series Forecasting with Neural Networks: A Case Study. *Applied Statistics*, 47:231–250, 1998.

[11] S. Gallant. *Neural Network Learning and Expert Systems*. MIT Press, Cambridge, USA, 1993.

[12] S. Haykin. *Neural Networks - A Compreensive Foundation*. Prentice-Hall, New Jersey, 2nd edition, 1999.

[13] J. Hopfield. Neural Networks and Physical Systems with Emergent Collective Computational Abilities. In *the National Academy of Science*, volume 79, pages 2554–8, 1982.

[14] M. Jordan. Why the logistic function? a tutorial discussion on probabilites and neural networks. 9503, Massachusetts Institute of Technology, USA, August 1995.

[15] D. Kemsley and T. Martinez. A Survey Of Neural Network Research And Fielded Applications. *International Journal of Neural Networks: Research and Applications*, 2:123–133, 1992.

[16] R. Kohavi. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, Montreal, Quebec, Canada, August 1995.

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

[18] B. Kosko. Bidirectional Associative Memories. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-18:49–60, 1988.

[19] T. Kwok and D. Yeung. Constructive algorithms for structure learning in feedforward neural networks for regression problems problems: A survey. *IEEE Transactions on Neural Networks*, 8(3):630–645, May 1999.

[20] Y. LeCun. Efficient Learning and Second-order Methods. A Tuturial at NIPS 93, 1993.

[21] W. McCulloch and W. Pits. A Logical Calculus of the Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[22] D. Memmi. Connectionism and artificial intelligence. In *Neuro-Nimes'89 International Workshop on Neural Networks and their Applications*, pages 17–34, Nimes, France, 1989.

[23] M. Minsky. Logical vs. Analogical or Symbolic vs. Connectionist or Neat vs. Scruffy. *MIT Press*, 1, 1990.

[24] J. Neves and P. Cortez. Combining Genetic Algorithms, Neural Networks and Data Filtering for Times Series Forecasting. In Nicos E. Mastorakis, editor, *2nd IMACS International Conference on: Circuits, Systems and Computers - IMACS-CSC'98*, volume 2, pages 933–939, Piraeus, Greece, October 1998. IMACS.

[25] D. Patterson. *Artificial Neural Networks - Theory and Applications*. Prentice Hall, Singapore, 1996.

[26] L. Prechelt. PROBEN1 - A Set of Neural Network Benchmark Problems and Benchmarking Rules. Research Report, Fakultät für Informatik, Universität Karlsruhe Germany, 1994.

[27] M. Riedmiller. Rprop - Description and Implementation Details. Technical report, University of Karlsruhe, Karlsruhe, Germany, January 1994.

[28] M. Riedmiller. Supervised Learning in Multilayer Perceptrons - from Backpropagation to Adaptive Learning Techniques . *Computer Standards and Interfaces*, 16, 1994.

[29] M. Riedmiller and H. Braun. A Direct Adaptative Method for Faster Backpropagation Learning: The RPROP Algorithm. In *Proceeedings of*

the IEEE International Conference on Neural Networks, San Francisco, CA, USA, 1993.

[30] R. Rojas. *Neural Networks - A Systematic Introduction.* Springer-Verlag, Germany, 1996.

[31] D. Rumelhart, G. Hinton, and R. Williams. Learning Internal Representations by Error Propagation. In D. Rulmelhart and J. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, volume 1, pages 318–362, MIT Press, Cambridge MA, 1986.

[32] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice-Hall, New Jersey, USA, 1995.

[33] W. Sarle. Stopped Training and Other Remedies for Overfitting. In *Proceedings of the 27th Symposium on the Interface of Computer Science and Statistics*, pages 352–360, 1995.

[34] W. Sarle. Neural network faq. ftp://ftp.sas.com/pub/neural/FAQ.html, 1999.

[35] R. Sharda and R. Rampal. Neural Networks and Management Science/Operations Research: A Bibliographic Essay. *Encyclopedia of Library and Information Science*, 61:247–259, 1996.

[36] G. Thimm and E. Fiesler. Pruning of Neural Networks. Research report idiap-rr 97-03, IDIAP, 1997.

[37] S. R. y Cajal. *New Ideas on the Structure of the Nervous System in Man and Vertebrates.* MIT Press, Cambridge, MA, translation of the French edition of 1894, 1990.