3
Inteligência Computacional

Inteligência Computacional (IC) é um ramo da ciência da computação que desenvolve, através de técnicas inspiradas na natureza, algoritmos capazes de imitar algumas habilidades cognitivas, como reconhecimento, aprendizado e percepção. Dentre as principais técnicas desenvolvidas encontram-se: Redes Neurais Artificiais e Computação Evolucionária. Neste trabalho estas duas técnicas foram utilizadas para a aproximação de dados e otimização de parâmetros, respectivamente.

3.1
Redes Neurais

Inspirada na estrutura e operação do cérebro humano, uma Rede Neural Artificial (RNA) é um modelo matemático não-linear usado para encontrar relacionamentos complexos entre a entrada e a saída de dados. Esse modelo é usado em problemas da predição de séries temporais, reconhecimento de padrões e aproximação de funções. Três conceitos básicos caracterizam os diversos tipos de RNAs: o modelo do neurônio artificial, sua estrutura de interconexão (topologia) e a regra de aprendizado.

Assim como o sistema nervoso é composto por bilhões de neurônios, a RNA também é formada por unidades elementares, denominadas neurônios artificiais ou processadores, que efetuam operações simples. Nas redes, esses neurônios encontram-se interconectados, transmitindo seus resultados aos processadores vizinhos. As RNAs são eficazes na aproximação de funções não-lineares a partir de dados não-lineares, incompletos, com ruído ou compostos por exemplos contraditórios, sendo essa potencialidade de modelar sistemas não-lineares a principal vantagem sobre outros métodos de interpolação. Na maioria dos casos, uma RNA é um sistema adaptável que muda sua estrutura com base na informação externa ou interna da rede.

3.1.1 Histórico

O ano de 1943 é considerado o marco inicial no desenvolvimento das Redes Neurais Artificiais. McCulloch e Pitts modelaram o primeiro modelo formal de um neurônio computacional [28]. Nesse modelo, o neurônio é uma estrutura binária em que o estado de saída pode assumir os valores lógicos verdadeiro ou falso, em função dos sinais de entrada. Caso o somatório dos sinais de entrada seja superior a um determinado limiar, o estado de saída assume o valor verdadeiro, caso contrário, falso. Tal modelo nunca obteve significância técnica, mas se tornou a principal referência da Teoria das Redes Neurais Artificiais.

Donald Hebb, em 1949, foi o primeiro a propor um método de aprendizado para atualizar as conexões entre neurônios [29] que hoje é conhecido como aprendizado *Hebbiano*. Ele enunciou que a informação pode ser armazenada nas conexões, e postulou a técnica de aprendizado que teve um profundo impacto nos desenvolvimentos futuros nesse ramo.

As redes neurais compostas por múltiplos neurônios foram introduzidas por Rosenblatt, em 1957 [30]. O modelo *perceptron*, com grande sucesso em certas aplicações, apresentou problemas em outras aparentemente similares, fazendo com que o uso e difusão das redes neurais na década de 1960 fossem irrelevantes.

No final dos anos 60, as limitações das redes tipo *perceptron* se tornaram públicas. Em 1969, Minsky e Papert provaram matematicamente que essas redes eram incapazes de solucionar problemas simples como o caso do ou-exclusivo [31], causando mais um período de diminuição no uso de tal técnica.

Somente em 1986, McClelland e Rummelhart desenvolveram um algoritmo de treinamento de redes multicamadas, denominado retropropagação (*backpropagation*), tornando as redes aplicáveis [32] a qualquer problema. Trata-se de um algoritmo de otimização que utiliza método do gradiente decrescente para minimizar a função de erro. Este algoritmo será posteriormente apresentado.

A partir de então as redes neurais evoluíram muito e rapidamente, desenvolvendo-se vários tipos de arquitetura, métodos de treinamento e aplicativos comerciais.

3.1.2 Estrutura de uma Rede

Neurônio artificial

O neurônio artificial i, tipicamente denominado elemento processador, é inspirado no neurônio biológico, possuindo um conjunto de entradas x_m (dendritos) e uma saída y_i (axônio), conforme ilustrado na figura 3.1. As entradas são ponderadas por pesos sinápticos w_{im} (sinapses), que determinam o efeito da entrada x_m sobre o processador i. Estas entradas ponderadas são somadas, fornecendo o potencial interno do processador v_i. A saída ou estado de ativação y_i do elemento processador i é finalmente calculada através de uma função de ativação $\phi(\cdot)$. O estado de ativação pode então ser definido pela equação 3-1.

$$y_i = \phi\left(\sum_{j=1}^{m} x_j w_{ij} + \theta_i\right) \tag{3-1}$$

onde m é o número de entradas do neurônio i e θ_i é um termo de polarização do neurônio (bias).

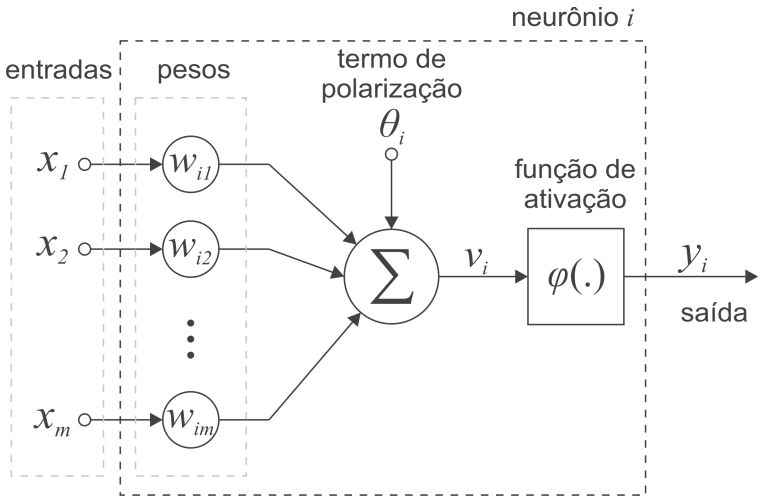


Figura 3.1: Representação gráfica de um neurônio artificial.

Função de ativação

A função de ativação é responsável por majorar ou minorar a informação recebida por um neurônio antes de passá-la adiante. Qualquer função pode ser utilizada para tal fim. Porém, a diferenciabilidade é uma característica importante na teoria das redes neurais, possibilitando o seu uso em algoritmos de treinamento baseados no método do gradiente. A tabela 3.1 apresenta algumas funções de ativação usuais.

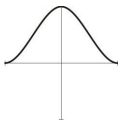
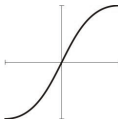
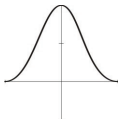
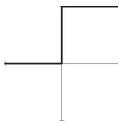
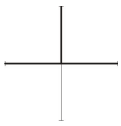
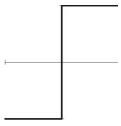
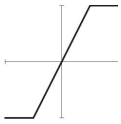
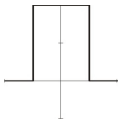
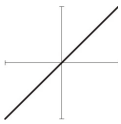
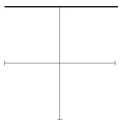
Nome	Função $\phi(v)$	Derivada $\partial\phi/\partial v$
Logística	 $\frac{1}{1+\exp(-\lambda v)}$	 $\lambda\phi(v)(1-\phi(v))$
Tanh	 $\frac{2}{1+\exp(-\lambda v)} - 1$	 $\lambda(1-\phi(v)^2)$
Degrau	 $\begin{cases} 0 & \text{se } v \leq 0 \\ 1 & \text{se } v > 0 \end{cases}$	 $\delta(v)$
Sinal	 $\begin{cases} -1 & \text{se } v \leq 0 \\ 1 & \text{se } v > 0 \end{cases}$	 $2\delta(v)$
Linear saturada	 $\begin{cases} -1 & v \leq -1/\lambda \\ \lambda v & -1/\lambda < v \leq 1/\lambda \\ 1 & v > 1/\lambda \end{cases}$	 $\begin{cases} 0 & v \leq -1/\lambda \\ \lambda & -1/\lambda < v \leq 1/\lambda \\ 0 & v > 1/\lambda \end{cases}$
Linear	 v	 1

Tabela 3.1: Funções de ativação e suas respectivas derivadas.

A função linear é geralmente utilizada quando a saída do neurônio pode atingir qualquer valor, ou seja, não possui valores limites. Quando há muitas entradas nesse neurônio, esta função pode produzir valores elevados nos resultados. É geralmente utilizada nos neurônios da camada de saída.

As funções sigmóides formam uma família de funções cujo gráfico tem a forma de s. Tal forma é a mais comumente utilizada, sendo definida como uma função estritamente crescente que apresenta na sua estrutura intervalos linear e não-lineares. Esta função possibilita o mapeamento de dados com tal comportamento. Um exemplo de função sigmóide é a função logística (tabela 3.1), em que os valores de saída dos neurônios são unipolares e estão contidos no intervalo [0,1]. Nessa função, λ é um parâmetro de suavização da curva. Variando-se o parâmetro λ, obtém-se funções sigmóides com inclinações diferentes. Quando $\lambda \to \infty$, a função tende à função degrau unitário.

Algumas vezes é desejável que a função se estenda ao intervalo [-1,1]. Nesse caso, a tangente hiperbólica é utilizada (3.1). Essa função também pertence às sigmóides, porém é bipolar, possibilitando o neurônio assumir valores de saída negativos.

Topologia

As topologias das redes neurais podem ser divididas em duas classes: não recorrentes e recorrentes. Redes não recorrentes são aquelas que não possuem realimentação de suas saídas para suas entradas e por isso são ditas "sem memória". A estrutura dessas redes é em camadas, podendo ser formadas por uma camada única ou múltiplas camadas. A figura 3.2 ilustra uma rede multi-camadas, em que os neurônios são representados por círculos (nós) e as conexões por retas (arcos). Essas redes contêm um conjunto de neurônios de entrada, uma camada de saída e uma ou mais camadas escondidas. A entrada não é considerada uma camada da rede, pelo fato de apenas distribuir os padrões para a camada seguinte [33]. A camada de saída contém os neurônios que fornecem o resultado da rede. As camadas que não possuem ligações diretas nem com a entrada, nem com a saída são denominadas camadas escondidas. No caso de redes não recorrentes não existem conexões ligando um neurônio de uma camada a outro de uma camada anterior, nem a um neurônio da mesma camada.

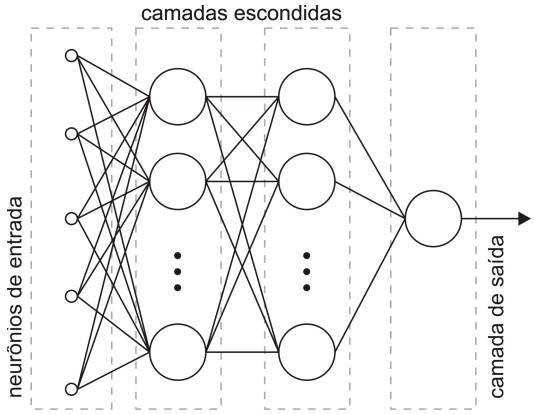


Figura 3.2: Exemplo de uma rede neural não recorrente.

As RNAs recorrentes são redes em que as saídas realimentam as entradas, sendo suas saídas determinadas pelas entradas atuais e pelas saídas anteriores. As redes recorrentes, quando organizadas em camadas, possuem interligações entre neurônios da mesma camada e entre camadas não consecutivas, gerando interconexões bem mais complexas que as redes neurais não recorrentes (figura 3.3).

As redes neurais recorrentes respondem a estímulos dinamicamente, isto é, após aplicar uma nova entrada, a saída é calculada e então realimentada para modificar a entrada. Para uma rede se tornar estável, este processo é repetido várias vezes, produzindo pequenas mudanças nas saídas, até que estas tendam a ficar constantes. Todavia, as redes neurais recorrentes não são necessariamente estáveis, mesmo com entradas constantes. O fato de não se conseguir prever

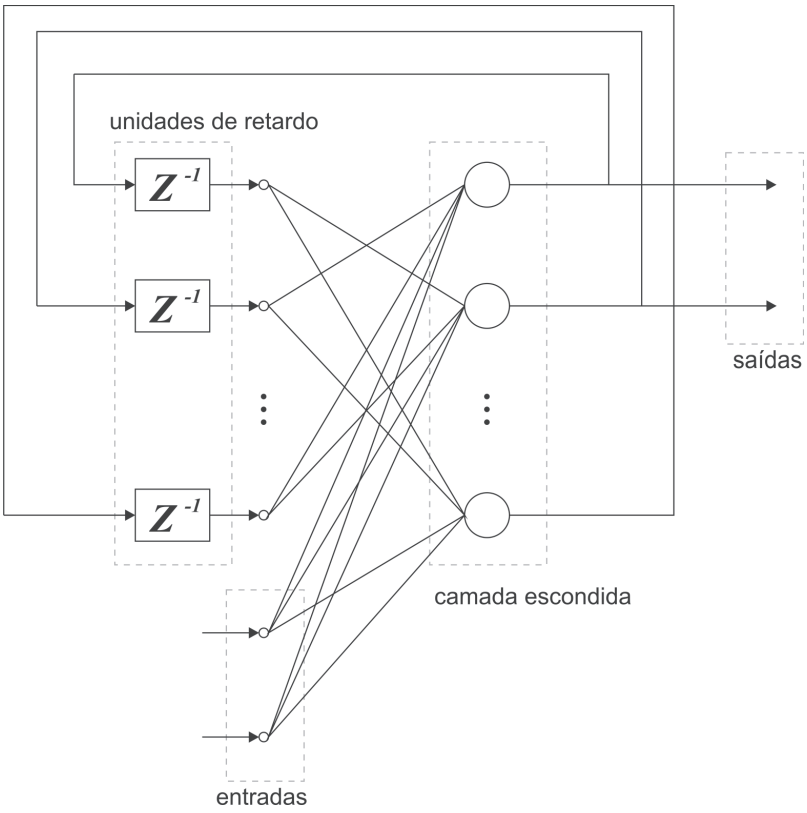


Figura 3.3: Exemplo de uma rede neural recorrente com duas entradas e duas saídas.

quais redes seriam estáveis foi um problema que preocupou os pesquisadores até o início da década de 80, quando Cohen e Grossberg provaram um teorema para definir quando as redes neurais recorrentes são estáveis [33]. Este teorema diz que, para as RNAs recorrentes alcançarem um estado estável, é necessário que as funções de ativação sejam monotônicas, hajam conexões simétricas, e um neurônio não possa realimentar a si mesmo. Contribuições importantes também foram dadas por John Hopfield [34], sendo que algumas configurações passaram a ser chamadas de redes de Hopfield em sua homenagem, e por Hinton e Sejnowski, que introduziram regras gerais para treinamento.

3.1.3 Tratamento dos dados

Como as ordens de grandeza de cada entrada e saída podem ser distintas, é necessário empregar algum tipo de normalização antes de executar o treinamento da rede neural. Desta forma, o valor de uma variável não se torna mais significativo que o de outra quando as escalas forem diferentes, ou seja, uma variável que assume valores de 1 a 2500 não afetará mais o sistema que outra com valores entre 0 e 0,5.

A normalização linear consiste em transformar os dados de modo que

se encontrem em um determinado intervalo. Tal normalização é definida pela equação 3-2, onde x é o valor do dado real, y o dado normalizado e os índices max e min são seus valores máximos e mínimos, respectivamente.

$$f(x) = y = (y_{max} - y_{min})\frac{(x - x_{min})}{(x_{max} - x_{min})} + y_{min} \tag{3-2}$$

Quando os dados se encontram concentrados em um subespaço do intervalo admissível de uma variável, utiliza-se um método para dispersá-los. Um dos métodos mais simples de se implementar é a normalização linear por partes. Tal normalização determina subintervalos a partir de um valor intermediário, assim, uma parte dos dados é normalizada entre um dado intervalo $[y_{min}; y_{int}]$ e outra parte é normalizada em outro intervalo $[y_{int}; y_{max}]$, conforme a equação 3-3.

$$f(x) = y = \begin{cases} (y_{int} - y_{min})\frac{(x - x_{min})}{(x_{int} - x_{min})} + y_{min} & \text{se } x \leq x_{int} \\ (y_{max} - y_{int})\frac{(x - x_{int})}{(x_{max} - x_{int})} + y_{int} & \text{se } x > x_{int} \end{cases} \tag{3-3}$$

3.1.4 Treinamento

O objetivo do treinamento de uma RNA é fazer com que a aplicação de um conjunto de entradas produza um conjunto de saídas desejado. Cada conjunto de entradas e saídas desejadas, ou alvo, é chamado de padrão de treinamento. O treinamento é realizado pela aplicação seqüencial dos vetores de entrada e, em alguns casos, também os de saída, enquanto os pesos da rede são ajustados de acordo com um procedimento de treinamento pré-determinado. Durante o treinamento, os pesos da rede gradualmente convergem para determinados valores, de modo que a aplicação dos vetores de entrada produza as saídas necessárias. Os procedimentos de treinamento podem ser classificados de duas formas: supervisionado e não supervisionado. Ambos usufruem de um conjunto de treinamento, entretanto o primeiro necessita de valores alvo para as saídas, enquanto que o segundo não.

O conjunto de treinamento modifica os pesos da rede de forma a produzir saídas que sejam consistentes, isto é, tanto a apresentação de um dos vetores de treinamento, como a apresentação de um vetor que é suficientemente similar, irá produzir o mesmo padrão nas saídas.

O treinamento supervisionado necessita de um par de vetores composto das entradas e do vetor alvo que se deseja obter como as respectivas saídas. Juntos, estes vetores são chamados de par de treinamento ou vetor de treinamento, sendo que geralmente a rede é treinada com vários vetores de treinamento.

Existe uma grande variedade de algoritmos de treinamento, tanto para

o treinamento supervisionado, quanto para o não supervisionado. Entre estes, o mais difundido é o algoritmo de retropropagação (*backpropagation*) [35]. O algoritmo de retropropagação contém duas etapas. A primeira é o método com o qual a aplicação da regra-da-cadeia obtém a derivada parcial da função de erro da rede em relação a cada um de seus pesos. A segunda é o algoritmo de atualização dos pesos, que consiste basicamente do gradiente decrescente.

Em suma, para a realização do treinamento supervisionado, é preciso que haja um conjunto de padrões de entrada e suas respectivas saídas, além de uma função de erro (função de custo) para medir o custo da diferença entre as saídas da rede e os valores desejados. Tal treinamento é iniciado com a apresentação e propagação de um padrão através da rede para obter as saídas. Uma vez calculadas, as saídas são comparadas com os respectivos valores alvo e o erro é então calculado a partir de alguma métrica. O erro então é utilizado para atualizar os pesos de acordo com um algoritmo de minimização, conforme ilustrado na figura 3.4 (adaptada de [36]). Este processo de treinamento é repetido até que o erro, para todos os vetores de treinamento, tenha alcançado o nível especificado ou até que um número máximo de iterações seja atingido. Cada iteração desse processo é conhecida como época.

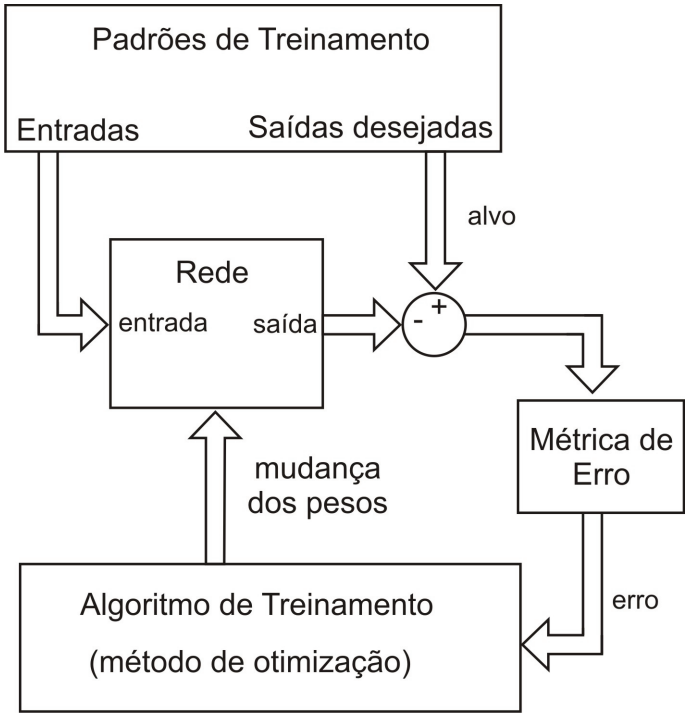


Figura 3.4: Treinamento supervisionado de uma rede neural.

Propagação das entradas

Os próximos algoritmos são explicados com base em uma rede exemplo multi-camada. Tal rede possui uma entrada de q nós, duas camadas escondidas

e um único neurônio de saída conforme a figura 3.2. Os elementos do vetor de pesos $\mathbf{w}$ estão ordenados por camadas (a partir da primeira camada escondida). Dentro de cada camada, os pesos estão ordenados por neurônios, seguindo a ordem das entradas de cada neurônio. Sendo $w_{ij}^{(l)}$ o peso sináptico do neurônio i da camada $l-1$ para o neurônio j na camada l. Para $l = 1$, a primeira camada escondida, o índice i se refere ao nó de entrada, ao invés de um neurônio.

Em tal rede, a propagação do sinal em cada camada l pode ser definida como $\mathbf{y}^{(l)} = F(\mathbf{w}, \mathbf{x})$ para uma entrada específica $\mathbf{x} = [x_1, x_2, \ldots, x_q]^T$ e um respectivo vetor de pesos $\mathbf{w}$. As equações 3-4 demonstram como, ao se utilizar uma função de ativação $\phi(\cdot)$, o sinal é propagado através de cada neurônio j camada a camada.

$$\begin{aligned} y_j^{(1)} &= \phi\left(\mathbf{x}^T \mathbf{w}_j^{(1)}\right) \\ y_j^{(2)} &= \phi\left((\mathbf{y}^{(1)})^T \mathbf{w}_j^{(2)}\right) \\ y_j^{(3)} &= \phi\left((\mathbf{y}^{(2)})^T \mathbf{w}_j^{(3)}\right) \end{aligned} \tag{3-4}$$

Métricas de erro

A menos que a rede esteja perfeitamente treinada, as saídas da rede estarão destoantes dos valores desejados. A significância dessas diferenças é medida por uma função de erro $\mathcal{E}$. A soma dos erros quadráticos (SSE) é uma métrica comumente utilizada

$$\mathcal{E}_{SSE} = \sum_p \sum_i (t_{pi} - y_{pi})^2 \tag{3-5}$$

onde, p indica o conjunto de padrões de treinamento, i os nós de saída, t_{pi} e y_{pi}, respectivamente, o alvo e o valor de saída da rede para o padrão p do nó i. O erro quadrático médio (MSE) normaliza o SSE para o número P de padrões de treinamento e as N saídas da rede, como segue:

$$\mathcal{E}_{MSE} = \frac{1}{PN} \mathcal{E}_{SSE} \tag{3-6}$$

As funções SSE e MSE têm a vantagem de serem facilmente diferenciáveis e que seus custos dependem somente da magnitude do erro [36].

Na avaliação final da rede, o erro percentual médio absoluto (MAPE) é outra métrica muito utilizada. Esta é definida na equação 3-7. O MAPE permite uma melhor sensibilidade na análise dos resultados, pois representa a distância percentual entre o valor obtido e o desejado.

$$\mathcal{E}_{MAPE} = \frac{1}{NP}\sum_{p}\sum_{i}\left|\frac{t_{pi} - y_{pi}}{t_{pi}}\right| \times 100 \tag{3-7}$$

Retropropagação do erro

Como dito anteriormente, a retropropagação é constituída de duas etapas: o cálculo da derivada do erro e a atualização de pesos.

Na primeira etapa, geralmente utiliza-se o SSE como métrica de erro, porém qualquer função de erro que seja derivável pode ser utilizada. A derivada depende da localização do neurônio, isto é, se pertence à camada de saída ou não. O resultado final da utilização da regra da cadeia para a métrica SSE é apresentado a seguir, sendo o desenvolvimento facilmente encontrado em diversas referências [35, 36, 37]. A derivada do erro pode ser vista como o somatório das derivadas do erro relativo a cada padrão de treinamento, conforme a equação 3-8.

$$\frac{\partial \mathcal{E}}{\partial w_{ij}} = \sum_{p}\frac{\partial \mathcal{E}_p}{\partial w_{ij}} \tag{3-8}$$

Cada uma dessas derivadas pode ser representada como a equação 3-9, em que o valor de δ_i muda de acordo com a localização do neurônio (equação 3-10).

$$\frac{\partial \mathcal{E}_p}{\partial w_{ij}} = \delta_i y_i \tag{3-9}$$

$$\delta_i = \begin{cases} -(t_{pi} - y_{pi})\phi_i' & \text{para neurônios de saída} \\ \phi_i' \sum_k w_{ki}\delta_k & \text{para neurônios escondidos} \end{cases} \tag{3-10}$$

A segunda etapa do algoritmo da retropropagação (atualização dos pesos) é praticamente equivalente ao método do gradiente decrescente. Por definição o gradiente de $\mathcal{E}$ indica a direção de maior crescimento de $\mathcal{E}$, sendo que para sua minimização é necessário caminhar na direção contrária. A retropropagação atribui uma taxa de aprendizado η que determina o passo do algoritmo. A equação 3-11 apresenta a variação dos pesos em função do gradiente do erro.

$$\Delta w_{ij} = -\eta\frac{\partial \mathcal{E}}{\partial w_{ij}} \tag{3-11}$$

Algoritmos baseados na retropropagação

Existem diversos algoritmos que tiram proveito da forma em que o gradiente do erro em função dos pesos é calculado segundo o método de retropropagação. Tais algoritmos diferenciam-se pela maneira como atualizam os pesos da rede. A primeira variação da retropropagação a se tornar popular foi o gradiente decrescente com momento (equação 3-12). Nela, a última atualização de pesos é ponderada por uma taxa de momento μ, evitando

mudanças bruscas na direção.

$$\Delta \mathbf{w}(t) = -\eta \frac{\partial \mathcal{E}}{\partial w}(t) + \mu \Delta \mathbf{w}(t-1) \tag{3-12}$$

Outros métodos que utilizam derivadas de segunda ordem são muito eficientes sob certas condições. Enquanto os métodos de primeira ordem utilizam uma aproximação linear da superfície de erro, os métodos de segunda ordem utilizam uma aproximação quadrática. O algortimo de Newton é o mais conhecido para a minimização de funções, sua adaptação para atualizar os pesos é apresentada na equação 3-13.

$$\Delta \mathbf{w}(t) = -\eta \mathbf{H}^{-1} \frac{\partial \mathcal{E}}{\partial w}(t) \tag{3-13}$$

onde $\mathbf{H}$ é a matriz Hessiana. Como calcular a inversa da matriz Hessiana é custoso, são utilizados métodos, conhecidos como Quasi-Newton, que aproximam esse valor. Os dois métodos Quasi-Newton mais difundidos são o algoritmo de Davidon-Fletcher-Powell (DFP) e Broyden-Fletcher-Goldfarb-Shanno (BFGS), sendo o segundo mais recomendado [36].

O método de Levenberg-Marquadt (LM) varia entre o método do gradiente decrescente e o de Newton. Esse método tira proveito da rápida convergência do método de Newton quando próximo de um mínimo, evitando que esse divirja pela utilização do gradiente. A direção de busca é uma combinação linear da direção do gradiente decrescente $\mathbf{g}$ e do método de newton ($\mathbf{H}^{-1}\mathbf{g}$), como segue:

$$\Delta \mathbf{w} = -(\mathbf{H} + \lambda \mathbf{I})^{-1} \mathbf{g} \tag{3-14}$$

onde, $\mathbf{I}$ é a matriz identidade e o parâmetro λ controla o compromisso entre o gradiente e o método de Newton. O algoritmo começa com o valor de λ grande, fazendo com que a equação tenda ao gradiente, esse valor é decrementado a cada iteração, tendendo ao método de Newton para a convergência final.

A Regularização Bayesiana (RB) minimiza uma combinação linear entre o SSE e a magnitude dos pesos. Ela também modifica essa combinação linear a fim de atingir uma boa qualidade de generalização da rede [38]. Essa combinação linear é então utilizada como a função de custo no algoritmo LM.

Validação

O treinamento de redes neurais pode causar um super treinamento (*overfitting*). Esse termo é utilizado quando a rede se torna capaz de prever apenas o conjunto de dados utilizados no treinamento, perdendo sua capacidade de prever dados nunca apresentados para a rede (generalização). Para evitar esse super treinamento, utiliza-se um conjunto de validação. Esse conjunto de padrões é propagado pela rede a cada iteração do algoritmo de minimização. O

valor do erro desse conjunto é então monitorado, sendo que seu aumento indica que a rede está se tornando muito especializada. Então, a rede que deve ser escolhida é aquela em que o erro do conjunto de validação é o menor, como ilustrado na figura 3.5, onde os círculos (∘) representam os dados de treinamento e as cruzes (+) os de validação.

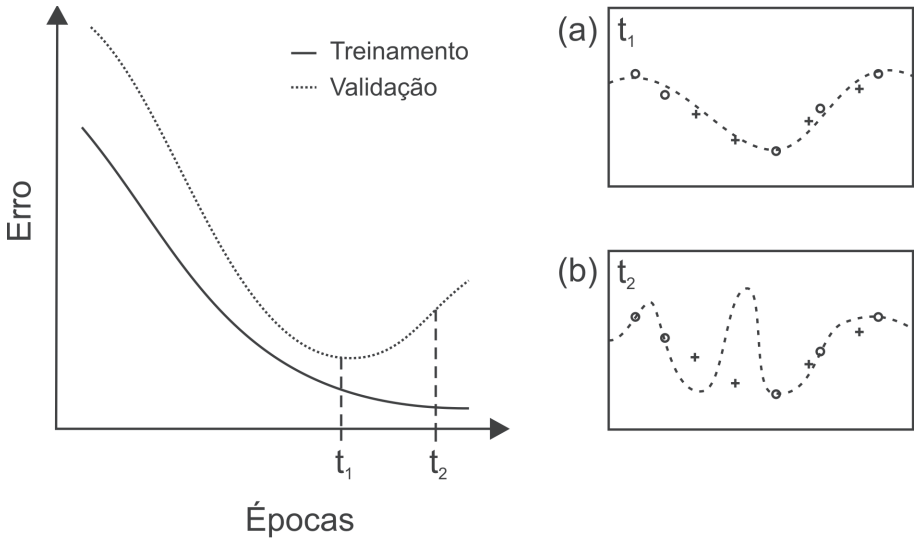


Figura 3.5: Validação cruzada. Ilustração de dois momentos distintos do treinamento: generalização da rede (a) e supertreinamento (b).

Inicialização dos pesos

O valor final do treinamento de uma rede depende do ponto inicial, isto é, sua configuração inicial de pesos. Isso devido aos algoritmos de minimização utilizados dependerem desse ponto. Um exemplo desse problema pode ser visto na figura 3.6, onde as curvas de nível representam a superfície do erro em função de duas variáveis (pesos), em que o vermelho é o erro mais alto e o azul o mais baixo. Nessa figura, os dois círculos azuis representam duas configurações de pesos distintas, e a cada iteração do algoritmo de minimização, um novo ponto (∘) é obtido. Nota-se que em cada inicialização o ponto de mínimo encontrado é distinto, isso porque a função de erro possui diversos mínimos locais. Para evitar esse acontecimento, um experimento deve ser feito com diversas configurações iniciais de peso, a fim de descobrir qual é a melhor rede.

3.2 Algoritmos Genéticos

Essencialmente, Algoritmos Genéticos (AGs) são métodos de busca e otimização, que têm sua inspiração nos conceitos da teoria de seleção natural das espécies proposta por Darwin. Os sistemas desenvolvidos a partir deste

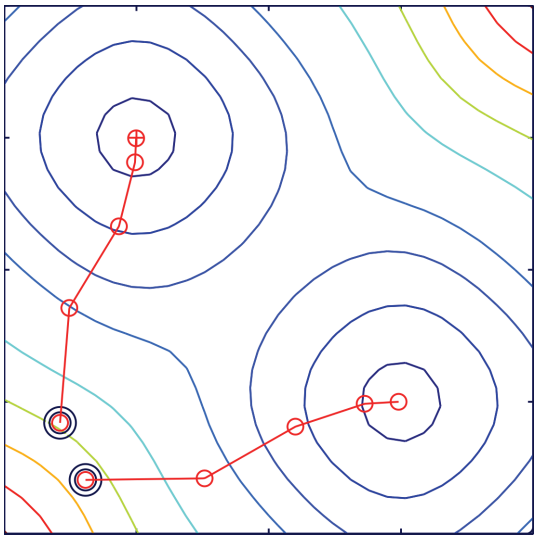


Figura 3.6: Dependência entre o ponto inicial e o erro final obtido.

princípio são utilizados para procurar soluções de problemas complexos ou com espaço de soluções (espaço de busca) muito grande, o que os tornam problemas de difícil modelagem e solução quando se aplicam métodos de otimização convencionais.

Estes algoritmos são baseados nos processos genéticos de organismos biológicos para procurar soluções ótimas ou sub-ótimas. Para tanto, procede-se da seguinte maneira: codifica-se cada possível solução de um problema em uma estrutura chamada de "cromossomo", que é composta por uma cadeia de bits ou caracteres. Estes cromossomos representam indivíduos, que são evoluídos ao longo de várias gerações, de forma similar aos seres vivos, de acordo com os princípios de seleção natural e sobrevivência dos mais aptos, descritos pela primeira vez por Charles Darwin em seu livro "A origem das espécies". Emulando estes processos, os Algoritmos Genéticos são capazes de "evoluir" soluções de problemas do mundo real.

Os cromossomos são então submetidos a um processo evolucionário que envolve avaliação, seleção, cruzamento e mutação. Após vários ciclos de evolução a população deverá conter indivíduos mais aptos. Os AGs utilizam uma analogia direta deste fenômeno de evolução na natureza, onde cada indivíduo representa uma possível solução para um problema dado. A cada indivíduo atribui-se um valor de adaptação: sua aptidão, que indica o quanto a solução representada por este indivíduo é boa em relação às outras soluções da população, cujo análogo em programação matemática é a função objetivo. Desta maneira o termo população refere-se ao conjunto de todas as soluções com as quais o sistema trabalha. Aos indivíduos mais adaptados é dada a oportunidade de se reproduzir mediante cruzamentos com outros indivíduos da

população, produzindo descendentes com características de ambas as partes. A mutação também tem um papel significativo, ao introduzir na população novos indivíduos gerados de maneira aleatória.

O processo de evolução começa com a criação aleatória dos indivíduos que formarão a população inicial. A partir de um processo de seleção baseado na aptidão de cada indivíduo, são escolhidos indivíduos para a fase de reprodução, que cria novas soluções através de um conjunto de operadores genéticos. Deste modo, a aptidão do indivíduo determina o seu grau de sobrevivência e, assim, a possibilidade do cromossomo fazer parte das gerações seguintes. O procedimento básico de um AG é resumido no algoritmo 1 [39].

$t \leftarrow 1$
$P(t) \leftarrow$ IniciaPopulação()
Avalia($P(t)$)
enquanto CondiçãoDeParada() = $falso$ **faça**
  $t \leftarrow t + 1$
  $P(t) \leftarrow$ SelecionaPopulação($P(t-1)$)
  AplicaOperadores($P(t)$)
  Avalia($P(t)$)
fim enquanto

Algoritmo 1: Procedimento básico do Algoritmo Genético.

Para determinar o final da evolução pode-se fixar o número de iterações do algoritmo (gerações), o número de indivíduos criados, ou ainda condicionar o algoritmo à obtenção de uma solução satisfatória, isto é, quando se atingir um ponto ótimo. Outras condições de parada incluem o tempo de processamento e o grau de similaridade entre os elementos numa população (convergência).

Uma das principais vantagens dos AGs frente aos métodos clássicos é que eles requerem pouco conhecimento específico do problema. Isso ocorre por serem baseados no conceito de população, não sendo tão dependente de um ponto inicial. Diversos estudos já foram realizados provando que AGs possuem uma convergência global [40].

As seções seguintes apresentam em mais detalhes cada um dos componentes de um algoritmo genético.

3.2.1 Representações

A solução de um problema pode ser representada por um conjunto de parâmetros (genes), unidos para formar um conjunto de valores (cromossomo); a este processo chama-se codificação. As soluções (cromossomos) são codificadas através de uma seqüência formada por caracteres de um sistema alfabético. Originalmente, utilizou-se o alfabeto binário, $0, 1$. Porém, novos mo-

delos de AGs codificam as soluções com outros alfabetos, como por exemplo com números inteiros ou reais.

Assim a representação é um aspecto fundamental na modelagem de um AG para a solução de um problema. Ela define a estrutura do cromossomo, com os respectivos genes que o compõem, de maneira que este seja capaz de descrever todo o espaço de busca relevante do problema.

A decodificação do cromossomo consiste basicamente na construção da solução real do problema a partir de sua representação. Isto é, o processo de decodificação constrói a solução para que esta seja avaliada pelo problema.

3.2.2 Operadores

Operadores genéticos são algoritmos que modificam os genes possibilitando a evolução da solução. Existem duas classes de operadores: cruzamento e mutação. A geração de novos cromossomos pode ser realizada tanto pelo cruzamento quanto pela mutação, ou ainda pelos dois simultaneamente. Os operadores de mutação são responsáveis pela divergência das soluções, explorando o espaço de busca; enquanto que os operadores de cruzamento fazem com que as soluções convirjam, tirando proveito de um determinado subespaço de busca. Esses operadores variam de acordo com a representação utilizada. A seguir alguns operadores para as representações inteira e real serão apresentados.

Reais

O operador real mais usual é o cruzamento aritmético. Este consiste de uma combinação linear entre dois genes, conforme mostrado na equação 3-15.

$$v_f = f(v_{p1}, v_{p2}) = \alpha \cdot v_{p1} + (1 - \alpha) \cdot v_{p2} \tag{3-15}$$

onde $\alpha \in [0, 1]$ é uma variável aleatória e v_f é o valor do gene do filho, gerado em função dos valores dos genes dos pais v_{p1} e v_{p2}.

Entre as mutações, os dois tipos mais comuns são a mutação uniforme e a não uniforme. A uniforme explora todo o espaço de busca igualmente, equação 3-16, sendo independente dos pais.

$$v_f = \alpha \cdot (v_{max} - v_{min}) + v_{min} \tag{3-16}$$

onde v_{max} e v_{min} são, respectivamente, os valores máximo e mínimo que tal gene pode assumir, ou seja, as restrições de caixa desse gene.

Já a mutação não uniforme faz com que os genes sorteados no espaço se concentrem em torno do gene do pai, em função do número de gerações

decorrido e de um *bit* aleatório.

$$v_f = f(v_{p1}) = \begin{cases} v_{p1} + \Delta(t, v_{max} - v_{p1}) & \text{se o bit for 0} \\ v_{p1} - \Delta(t, v_{p1} - v_{min}) & \text{se o bit for 1} \end{cases} \tag{3-17}$$

$$\Delta(t, y) = y \cdot \left(1 - \alpha^{\left(1 - \frac{t}{T}\right)^b}\right) \tag{3-18}$$

onde t é a geração atual e T o número máximo de iterações do algoritmo. Por fim, b é um parâmetro que determina o grau de dependência de acordo com o número da geração.

Inteiros

Entre os operadores inteiros, alguns se assemelham com os reais, porém são truncados. Esse é o caso do cruzamento aritmético e da mutação uniforme. Entretanto, outros tipos de cruzamentos são bastante usuais, é o caso dos cruzamentos de um e dois pontos. O cruzamento de um ponto, diferentemente dos operadores vistos até então, é aplicado ao cromossomo como um todo e não gene a gene. Nele, escolhe-se um ponto de corte aleatório e a partir desse ponto todos os demais genes são trocados entre os progenitores, como ilustra a figura 3.7. O cruzamento de dois pontos é semelhante, possuindo dois pontos de corte aleatórios, sendo trocados apenas os genes entre esses pontos.

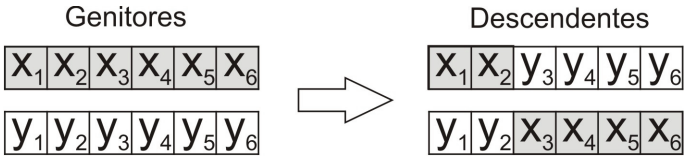


Figura 3.7: Cruzamento de um ponto.

3.2.3 Múltiplos objetivos

Muitas aplicações reais possuem múltiplos objetivos. Em geral busca-se minimizar o custo das operações junto com um ganho em outra característica do processo que está sendo aprimorado. A otimização de cada objetivo separado não produz bons resultados, pois ao otimizar um deles, os demais podem não ser satisfeitos. Técnicas tradicionais de otimização trabalham com um único valor a ser otimizado. Para isso, estratégias de combinação de objetivos foram desenvolvidas. As principais formas de processamento de múltiplos objetivos por computação evolutiva disponíveis na literatura são: agregação de objetivos, estratégia de Pareto [41] e minimização de energia [42].

A agregação de objetivos consiste na soma ponderada de cada função objetivo f_i (equação 3-19). A principal desvantagem desse método é a necessidade

de conhecimentos específicos do problema, de modo a definir corretamente os pesos de cada objetivo.

$$F = \sum_i w_i f_i \tag{3-19}$$

A estratégia de Pareto compara soluções sem combinar as avaliações, usando o conceito de dominância. Uma solução v domina outra solução u se para nenhum objetivo é verificado que a avaliação de v seja pior que a de u; e para pelo menos um objetivo é verificado que a avaliação de v é superior a de u. Define-se o conjunto Pareto-Ótimo como sendo o conjunto de todas as soluções que não são dominadas por nenhuma outra. Qualquer que seja o critério para a avaliação global, é garantido que a melhor solução fará parte desse conjunto. As principais desvantagens dessa estratégia é a necessidade de um critério adicional para a escolha da solução ótima entre as pertencentes ao conjunto Pareto-Ótimo. Na prática, essa estratégia pode não orientar a evolução no sentido desejado, levando a resultados insatisfatórios para certos problemas.

A minimização de energia é um método adaptativo de agregação de objetivos. Este método atualiza os pesos dinamicamente ao longo do processo evolutivo, estabelecendo pesos maiores para os objetivos que forem menos satisfeitos pela população do AG. O cálculo da aptidão é feito em função dos pesos, das aptidões e da média das aptidões de cada objetivo. Para a atualização dos pesos, precisa-se especificar um valor alvo para cada objetivo. A principal vantagem desse método é a capacidade de adaptar a orientação da evolução aos problemas encontrados durante o processo. As desvantagens são: a necessidade de se especificar um valor alvo para cada objetivo e a possibilidade do surgimento de subgrupos especializados em satisfazer cada objetivo.