3
Técnicas de Inteligência Computacional

Inteligência computacional é um ramo da ciência da computação que utiliza algoritmos e técnicas que imitam algumas habilidades cognitivas, como reconhecimento, aprendizado e evolução, para criar programas, de alguma forma, inteligentes. Dentre os algoritmos mais conhecidos, tem-se: Computação Evolucionária, Redes Neurais Artificiais e Lógica Fuzzy. Neste trabalho, as técnicas utilizadas foram a Computação Evolucionária e as Redes Neurais Artificiais que são descritas a seguir.

3.1
Computação Evolucionária

Esta seção resume os principais conceitos sobre os algoritmos evolucionários. Primeiramente, é feita uma breve explicação sobre o princípio de funcionamento de um Algoritmo Genético (AG), descrevendo suas partes principais. Em seguida, a co-evolução é descrita. Por fim, uma alteração do AG é apresentada através do Algoritmo Genético com Inspiração Quântica (AGIQ).

3.1.1
Algoritmos Genéticos

Essencialmente, Algoritmos Genéticos são métodos de busca e otimização que têm sua inspiração nos conceitos da teoria de seleção natural das espécies proposta por Darwin (Mitchell, 1996, Koza, 1992, Goldberg, 1989, Back, 1996, Fogel, 1966). Os sistemas desenvolvidos a partir deste 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, ou programação matemática.

Estes algoritmos são baseados nos processos genéticos de organismos biológicos para procurar soluções 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 o resultado da função objetivo. Desta maneira, o termo População refere-se ao conjunto de todas as soluções com as quais trabalha o sistema. 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 utilizando-se para isto um conjunto de operadores genéticos. Deste modo, a aptidão do indivíduo determina o seu grau de sobrevivência e, assim, a possibilidade de que o cromossomo possa fazer parte das gerações seguintes. O procedimento básico de um algoritmo genético é resumido na Figura 3.1 (Davis, 1991).

```
Inicio
 t←1
 inicializar população P(t)
 avaliar população P(t)
 enquanto(não_condição_de_fim) faça
  t←t+1
  selecionar população P(t) a partir de P(t-1)
  aplicar operadores genéticos
```

Figura 3.1: Procedimento padrão do Algoritmo Genético.

Para determinar o final da evolução pode-se fixar o número de 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 para a parada incluem o tempo de processamento e o grau de similaridade entre os elementos numa população (convergência). As seções seguintes apresentam em mais detalhes cada um dos componentes de um algoritmo genético.

Representação

A solução de um problema pode ser representada por um conjunto de parâmetros (genes), unidos para formar uma cadeia de valores (cromossomo); a este processo chama-se codificação. As soluções (cromossomos) são codificadas através de uma sequência formada por caracteres de um sistema alfabético. Originalmente, utilizou-se o alfabeto binário (0, 1). Porém, novos modelos de AGs codificam as soluções com outros alfabetos, como por exemplo com números reais (Michalewicz, 1994). 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 do cromossomo. Isto é, o processo de decodificação constrói a solução para que esta seja avaliada pelo problema.

Avaliação

A avaliação é a ligação entre o AG e o problema a ser solucionado. Ela é feita através de uma função que melhor representa o problema e tem por objetivo oferecer uma medida de aptidão de cada indivíduo na população corrente, que irá dirigir o processo de busca. Dado um cromossomo, a função de avaliação consiste em associar um valor numérico, o qual supõe-se proporcional à "utilidade" ou "habilidade" do indivíduo representado em solucionar o problema em questão. Muitas vezes a avaliação é realizada por um simulador ou um código externo ao AG.

3.1.2 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 se combinem, tirando proveito de um determinado subespaço de busca. Esses operadores variam

de acordo com a representação utilizada. Neste trabalho, grande parte dos experimentos utilizam a representação real e os operadores relacionados serão descritos a seguir. Alguns estudos de casos fazem uso de outros tipos de operadores, estes serão descritos no momento adequado. Outros operadores podem ser encontrados na literatura (Michalewicz, 1994).

O operador real mais usual é o cruzamento aritmético. Este consiste de uma combinação linear entre dois genes, conforme mostrado na equação 3-1.

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

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 3-2, sendo independente dos pais.

$$v_f = \alpha \cdot (v_{max} - v_{min}) + v_{min} \tag{3-2}$$

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

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

$$\Delta(t, y) = y \cdot \left(1 - \alpha^{\left(1 - \frac{t}{T}\right)^b}\right) \tag{3-4}$$

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.

3.1.3 Co-evolução

Para se aplicar algoritmos evolucionários com sucesso em problemas com complexidade cada vez maior, torna-se necessário introduzir noções explícitas de modularidade nas soluções para que elas disponham de oportunidades razoáveis de evoluir na forma de subcomponentes co-adaptados (Potter, 2000).

Existem duas razões principais pelas quais algoritmos evolucionários convencionais não são totalmente adequados para resolver este tipo de problema. Em primeiro lugar, os algoritmos genéticos convencionais impedem, a longo prazo, a preservação de certos componentes da solução pois, por estarem codificados por

completo em um indivíduo, eles são avaliados como um todo e apenas os subcomponentes que pertencem a indivíduos com avaliações altas serão preservados. Em segundo lugar, o fato da representação estar relacionada a uma solução completa e por não haver interações entre os membros da população, não existe pressão evolucionária para a ocorrência de co-adaptação, ou seja, não existe pressão para a adaptação de um subcomponente dada a ocorrência de uma mudança em outro subcomponente (Potter, 2000).

No entanto, a decomposição do problema tem um aspecto importante que deve ser levado em conta e que está relacionado com a evolução de subcomponentes interdependentes. Se for possível decompor o problema em subcomponentes independentes, cada um deles pode evoluir sem haver necessidade de se preocupar com os outros. Pode-se visualizar isto imaginando que cada subcomponente evolui no seu próprio espaço de busca, desacoplado do espaço de busca dos outros componentes.

O modelo co-evolucionário cooperativo genérico (Potter, 2000) é mostrado na figura 3.2.

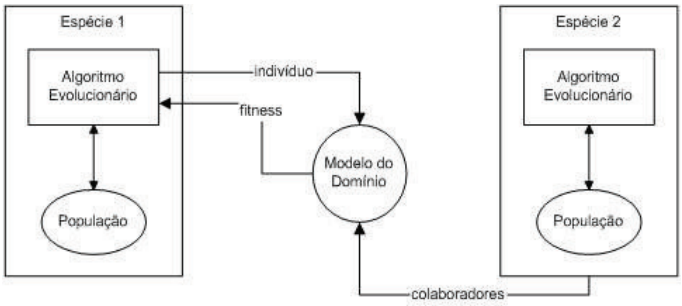


Figura 3.2: Modelo co-evolucionário genérico.

Apesar de serem mostradas apenas duas espécies neste modelo, o mesmo pode ser usado para n espécies diferentes. Cada espécie evolui em sua própria população (pois como já foi mencionado, elas são isoladas geneticamente) e se adaptam ao ambiente através de repetidas aplicações do algoritmo evolucionário. Para se calcular a aptidão dos indivíduos de uma determinada espécie, deve-se submetê-lo ao modelo do domínio (que contém a função de avaliação) juntamente com um ou mais colaboradores de cada uma das outras espécies (de modo a formar a solução completa).

3.1.4
Algoritmo Genético com Inspiração em Computação Quântica

A computação quântica é uma área recente de pesquisa, que utiliza fenômenos quânticos para resolver problemas de complexidade exponencial em um tempo aceitável. O uso de sistemas clássicos inspirados nesses fenômenos pode produzir um ganho substancial em relação ao tempo de processamento e, eventualmente, em relação à qualidade da solução e tem mostrado bons resultados em problemas *benchmark* (Han, 2000, Han, 2002, AbsDaCruz, 2004).

Algoritmos Genéticos com Inspiração Quântica

Os algoritmos evolucionários com inspiração quântica são baseados nos conceitos de bits quânticos, ou "qubits", e na superposição de estados da mecânica quântica (Han, 2000, Han, 2002, AbsDaCruz, 2004). O estado de um bit quântico pode ser representado por:

$$|\phi >= \alpha|0 > +\beta|1 > \tag{3-5}$$

onde α e β são números complexos que especificam as amplitudes de probabilidade dos estados correspondentes. $|\alpha^2|$ dá a probabilidade do qubit estar no estado 0 quando observado e $|\beta^2|$, por sua vez, dá a probabilidade do qubit estar no estado 1 quando observado. A normalização das amplitudes garante que

$$|\alpha^2| + |\beta^2| = 1 \tag{3-6}$$

O algoritmo evolucionário com representação binária (Han, 2000, Han, 2002) funciona bem somente quando esta representação é a mais adequada para o problema específico. No entanto, em alguns casos, a representação por números reais é mais eficiente (no caso de otimização de funções, por exemplo, onde se deseja encontrar um valor máximo, ou mínimo, a partir dos ajustes das variáveis) (AbsDaCruz, 2004). Mas como implementar essa representação usando-se o paradigma da inspiração quântica? Para se responder essa questão é necessário responder outras duas questões:

- Como representar a superposição de estados, dado que neste tipo de problema os genes podem assumir valores em um intervalo contínuo entre os limites das variáveis?
- Como atualizar estes valores de modo a levar o algoritmo a convergir para um valor ótimo ou sub-ótimo?

Para a primeira questão a resposta é simples: ao invés de se usar as probabilidades de observação de um estado, define-se uma distribuição de probabilidades para cada variável que permita sortear facilmente valores no universo de discurso

da variável. Para evitar um crescimento exponencial do armazenamento de informações, e reduzir o custo computacional, optou-se por usar um conjunto de pulsos retangulares para se representar estas distribuições (distribuição de probabilidade uniforme). Desta forma, há duas grandes vantagens: somente é necessário armazenar o centro e a largura de cada pulso; e simplifica-se o cálculo das funções de distribuição cumulativa de probabilidades, que são necessárias para o sorteio dos números aleatórios.

O procedimento para inicializar o algoritmo começa então com uma definição de um valor N que indica quantos pulsos serão usados para representar a distribuição de probabilidade para cada uma das variáveis. Feito isto, para cada pulso usado em cada variável do problema, define-se:

– Centro igual ao ponto médio do domínio;
– Altura igual ao inverso do tamanho do domínio dividido por N.

Ao final deste processo, a soma dos N pulsos de cada uma das variáveis terá área total igual a 1. Supondo, por exemplo, que se deseje inicializar uma variável com universo de discurso em [-50,50] e utilizar 4 pulsos retangulares para representar a distribuição de probabilidade para esta variável, cada pulso teria largura igual a 100 e altura igual a 1/100/4 = 0,0025. Com a superposição, a forma gráfica desta distribuição ficaria, após a inicialização, da seguinte forma:

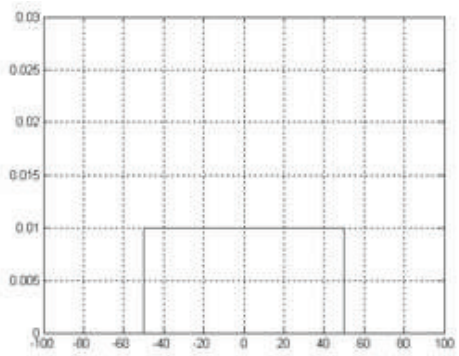


Figura 3.3: Gráfico com a superposição de 4 pulsos retangulares, cada um com largura 100 e altura 0,0025, representando a distribuição de probabilidade para uma variável

Esta curva representa, portanto, a superposição de todos os estados possíveis para a variável em questão. Este conjunto de distribuições de probabilidade de cada uma das variáveis (genes) do problema forma uma superposição $Q_i(t)$ para cada uma das i variáveis do problema.

A partir desta distribuição $Q_i(t)$ sorteia-se um conjunto de n pontos que formam a população P(t). Após o sorteio dos indivíduos que vão formar a população P(t) é necessário atualizar a distribuição de probabilidade $Q_i(t)$ de modo a obter a convergência para a solução ótima ou sub-ótima de maneira similar ao crossover convencional dos algoritmos genéticos. O método utilizado consiste em sortear um número m de indivíduos da população P(t) usando um processo de roleta idêntico ao dos algoritmos genéticos convencionais e, em seguida, redefinir o ponto central do primeiro pulso como o valor médio desses m indivíduos sorteados. Este processo é repetido para cada um dos N pulsos que definem a distribuição $Q_i(t)$. O valor m é dado por:

$$m = \frac{n}{N} \tag{3-7}$$

Além disso, em cada geração, a largura dos pulsos é reduzida de forma simétrica em relação a seus centros. Esta redução é feita de forma exponencial, de acordo com a fórmula:

$$\sigma = (u - l)^{(1-\frac{t}{T})^2} - 1 \tag{3-8}$$

Onde σ é a largura do pulso, u é o limite superior do domínio da variável, l é o limite inferior, t é a geração corrente do algoritmo, T é o total de gerações e λ é um parâmetro que define a velocidade de decaimento da largura do pulso.

É importante notar que, à medida que os pulsos têm suas larguras diminuídas e a posição de seus pontos centrais modificados, a soma deles deixa de ser um pulso e começa a assumir as mais diversas formas. Um exemplo de uma forma que a soma dos pulsos pode assumir é mostrado na figura a seguir, onde tem-se a soma de dois pulsos: o primeiro no intervalo [-50,50], com altura 0,005 e o segundo no intervalo [0,50] com altura 0,01.

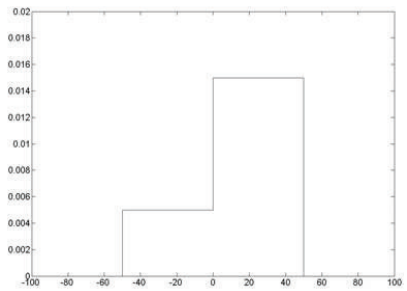


Figura 3.4: Forma da superposição de 2 pulsos de distribuição de probabilidade após algumas gerações do algoritmo.

3.2
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 elaboração de funções capazes de aproximar 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.

3.2.1
Estrutura de uma Rede

Neurônio artificial

O neurônio artificial *i*, tipicamente denominado de "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 3.5. 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, v_i, do processador. 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 3-9.

$$y_i = \phi\left(\sum_{j=1}^{m} x_j w_{ij} + \theta_i\right) \tag{3-9}$$

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

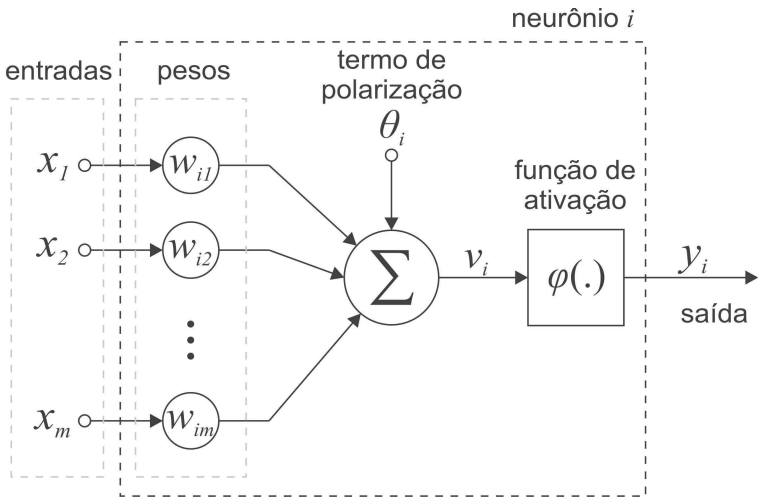


Figura 3.5: Representação gráfica de um neurônio artificial.

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.

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 comunmente 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 $\lambda \rightarrow \infty$, a função tende a 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.

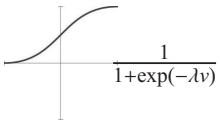
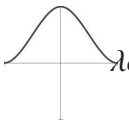
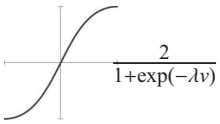
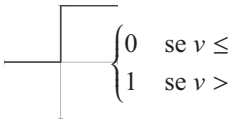
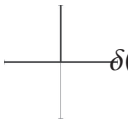
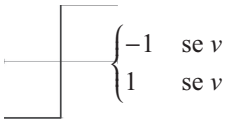
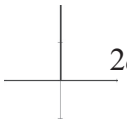
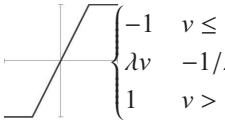
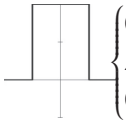
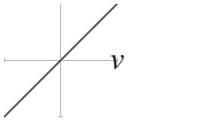
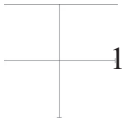
Nome	Função $\phi(v)$	Derivada $\partial\phi/\partial v$
Logística	 $\frac{1}{1+\exp(-\lambda v)}$	 $\lambda\phi(v)(1-\phi(v))$
Tanh	 $\frac{2}{1+\exp(-\lambda v)} - 1$	 $\lambda(1-\phi(v)^2)$
Degrau	 $\begin{cases} 0 & \text{se } v \le 0 \\ 1 & \text{se } v > 0 \end{cases}$	 $\delta(v)$
Sinal	 $\begin{cases}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

Tabela 3.1: Funções de ativação e suas respectivas derivadas.

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 3.6 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 (Wasserman, 1989). 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.

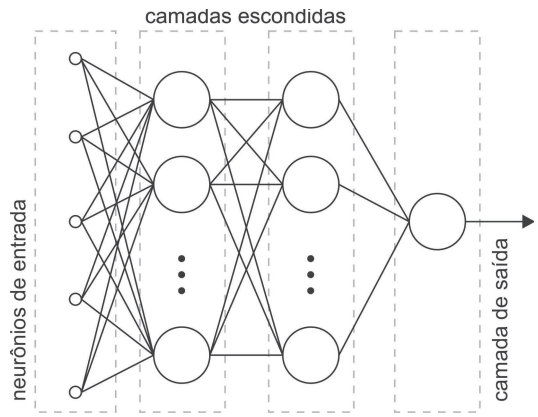


Figura 3.6: Exemplo de uma rede neural não recorrente.

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 3.7).

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 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 (Wasserman, 1989). 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 (Hopfield, 1982), 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.

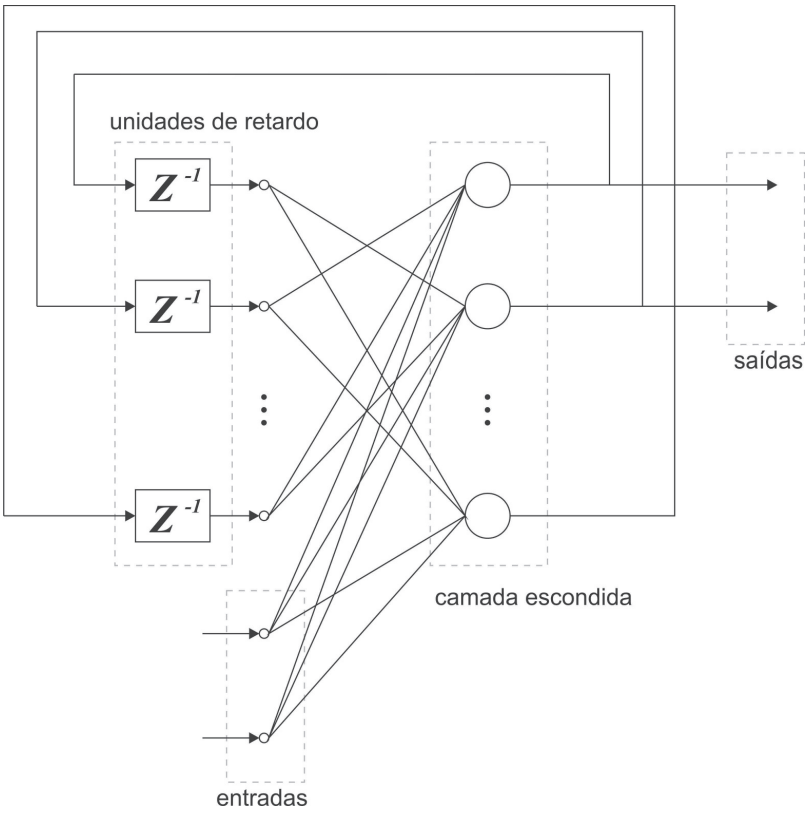


Figura 3.7: Exemplo de uma rede neural recorrente com duas entradas e duas saídas.

3.2.2 Tratamento dos dados

Como as ordens de grandeza dos parâmetros de entrada 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 3-10, 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.

$$f(x) = y = (y_{max} - y_{min})\frac{(x - x_{min})}{(x_{max} - x_{min})} + y_{min} \tag{3-10}$$

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 3-11.

$$f(x) = y = \begin{cases} (y_{int} - y_{min})\frac{(x - x_{min})}{(x_{int} - x_{min})} + y_{min} & \text{se } x \leq x_{int} \\ (y_{max} - y_{int})\frac{(x - x_{int})}{(x_{max} - x_{int})} + y_{int} & \text{se } x > x_{int} \end{cases} \quad (3\text{-}11)$$

3.2.3 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 sequencial 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, irão 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*). A 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 3.8. 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.

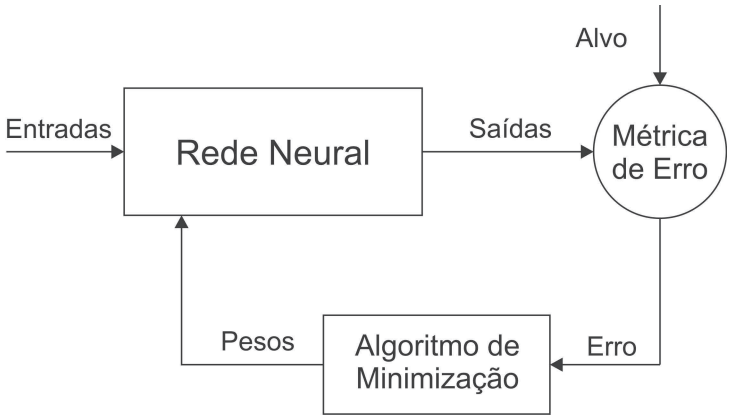


Figura 3.8: Treinamento supervisionado de uma rede neural. (Adaptado de (Reed, 1998))

Propagação das entradas

Os próximos algoritmos serão explicados com base em uma rede de exemplo com multi-camada. Tal rede possui uma entrada de q nós, duas camadas escondidas e um único neurônio de saída, conforme mostrado na figura 3.6. Os elementos do vetor de pesos $\mathbf{w}$ estão ordenados por camadas (a partir da primeira camada escondida) e em cada camada 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 3-12 demonstram como, ao se utilizar uma função de ativação $\phi(\cdot)$, o sinal é propagado através de cada neurônio j camada a camada.

$$\begin{aligned} y_j^{(1)} &= \phi\left(\mathbf{x}^T \mathbf{w}_j^{(1)}\right) \\ y_j^{(2)} &= \phi\left((\mathbf{y}^{(1)})^T \mathbf{w}_j^{(2)}\right) \\ y_j^{(3)} &= \phi\left((\mathbf{y}^{(2)})^T \mathbf{w}_j^{(3)}\right) \end{aligned} \tag{3-12}$$

Métricas de erro

A menos que a rede esteja perfeitamente treinada, as saídas da rede estarão destoantes com os 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

$$\mathcal{E}_{SSE} = \sum_p \sum_i (t_{pi} - y_{pi})^2 \tag{3-13}$$

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

$$\mathcal{E}_{MSE} = \frac{1}{PN}\mathcal{E}_{SSE} \tag{3-14}$$

As funções SSE e MSE têm a vantagem de serem facilmente diferenciáveis e que seus custos dependem somente da magnitude do erro (Reed, 1998).

Na avaliação final da rede, o erro percentual médio absoluto (MAPE) é outra métrica muito utilizada. Esta é definida na equação 3-15. O MAPE permite uma melhor sensibilidade na análise dos resultados, pois representa a distância percentual entre o valor obtido e o desejado.

$$\mathcal{E}_{MAPE} = \frac{1}{NP} \sum_p \sum_i \left| \frac{t_{pi} - y_{pi}}{t_{pi}} \right| \tag{3-15}$$

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 (Haykin, 1999, Reed, 1998, Zurada, 1992). A derivada do erro pode ser vista como o somatório das derivadas do erro relativo a cada padrão de treinamento, conforme a equação 3-16.

$$\frac{\partial \mathcal{E}}{\partial w_{ij}} = \sum_p \frac{\partial \mathcal{E}_p}{\partial w_{ij}} \tag{3-16}$$

Cada uma dessas derivadas pode ser representada como a equação 3-17, em que o valor de δ_i muda de acordo com a localização do neurônio (equação 3-18).

$$\frac{\partial \mathcal{E}_p}{\partial w_{ij}} = \delta_i y_i \tag{3-17}$$

$$\delta_i = \begin{cases} -(t_{pi} - y_{pi})\phi_i' & \text{para neurônios de saída} \\ \phi_i' \sum_k w_{ki}\delta_k & \text{para neurônios escondidos} \end{cases} \tag{3-18}$$

A segunda etapa do algoritmo da retropropagação é praticamente equivalente ao 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 3-19 apresenta a variação dos pesos em função do gradiente do erro.

$$\Delta w_{ij} = -\eta \frac{\partial \mathcal{E}}{\partial w_{ij}} \tag{3-19}$$

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 3-20). Nela, a última atualização de pesos é ponderada por uma taxa de momento μ, evitando mudanças bruscas na direção.

$$\Delta \mathbf{w}(t) = -\eta \frac{\partial \mathcal{E}}{\partial w}(t) + \mu \Delta \mathbf{w}(t-1) \tag{3-20}$$

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 3-21.

$$\Delta \mathbf{w}(t) = -\eta \mathbf{H}^{-1} \frac{\partial \mathcal{E}}{\partial w}(t) \tag{3-21}$$

onde **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 (Reed, 1998).

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 **g** e do método de newton ($\mathbf{H}^{-1}\mathbf{g}$), como segue:

$$\Delta\mathbf{w} = -(\mathbf{H} + \lambda\mathbf{I})^{-1}\mathbf{g} \tag{3-22}$$

onde, **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 o 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 (Mackay, 1992). 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 3.9, onde os círculos (○) representam os dados de treinamento e as cruzes (+) os de validação.

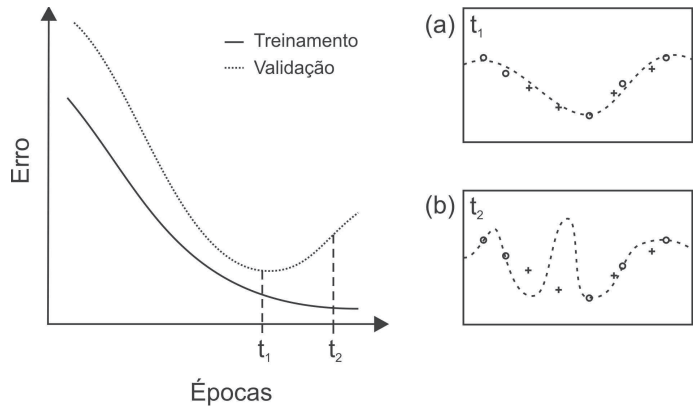


Figura 3.9: Validação cruzada. Ilustração de dois momentos distintos do treinamento: generalização da rede (a) e supertreinamento (b).

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 3.10, onde as curvas de nível representam a superfície do erro em função de duas variáveis (pesos). Nessa figura, os dois círculos menores 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.

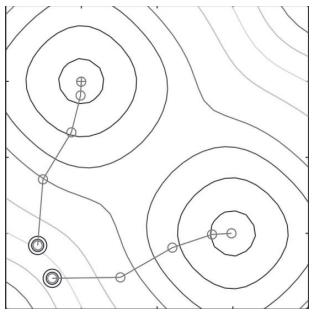


Figura 3.10: Dependência entre o ponto inicial e o erro final obtido.