UNIVERSIDADE ESTADUAL DE CAMPINAS

Faculdade de Engenharia Elétrica

Departamento de Engenharia de Computação e Automação Industrial

OTIMIZAÇÃO DE SISTEMAS ATRAVÉS DE REDES NEURAIS ARTIFICIAIS

Este exemplar corresponde à redação final da tese defendida por Roseli Aparecida Francelin e aprovada pela Comissão Julgadora em 28 / 07 / 93.

Orientador

Aluna : Roseli Aparecida Francelin Romero

Orientador : Prof. Dr. Fernando A. C. Gomide

Tese apresentada à Faculdade de Engenharia Elétrica da Universidade Estadual de Campinas, como parte dos requisitos exigidos para obtenção do Título de Doutor em Engenharia Elétrica.

Julho de 1993

AGRADECIMENTOS

Ao Prof. Dr. Fernando A. C. Gomide pela segurança e competência demonstradas na orientação deste trabalho, além do apoio e amizade dedicados.

Ao Departamento de Computação e Estatística - SCE, do Instituto de Ciências Matemáticas de São Carlos - USP pelo apoio dado para o desenvolvimento deste trabalho.

Aos meus familiares pelo incentivo recebido, em especial ao Airton, pela paciência e compreensão sempre constantes.

À Secretaria do Departamento de Computação e Automação Industrial - DCA e a seção de Pós-Graduação, da Faculdade de Engenharia Elétrica da UNICAMP, pela atenção e serviços prestados.

Contou-se com apoio financeiro da CAPES e do CNPq/RHAE.

Aos meus pais,

com muito carinho.

Sejam alegres! Busquem a perfeição,
tenham ânimo. Vivam em concórdia,
permaneçam em paz. E o Deus de amor e paz
estará com vocês. (2Cor 13,11)

Abstract

This thesis presents an artificial neural network with a three-layer feedback topology to solve continuous convex unconstrained and constrained optimization problems. A new scheme for updating the weights is introduced. This scheme is a modification of the *back-propagation* algorithm. It is based on the duality theory and subgradient methods. Computational results and a parallel implementation are presented which show the performance and validate the proposed approach. Further, details of implementation and comparative analysis with others optimization techniques are included.

Another class of artificial neural networks, with a two-layer feedback topology to solve nonlinear discrete dynamic optimization problems has also been developed. Generalized recurrent neurons are introduced. A direct method to assign the weights of neural networks is presented. The method is based on the Bellmann's Optimality Principle and in the interchange of information which occur during the synaptic chemical processing among neurons. A comparative analysis of the computational requirements has been performed. This analysis has highlighted advantages of the new approach when compared to the standard algorithm from dynamic programming. The technique has been applied to several important optimization problems, such as the knapsack and shortest path problems. In addition, two other applications: a power system long-range planning problem and discrete linear regulator problems have been tackled which demonstrate the applicability of the methodology.

Sumário

Esta tese apresenta uma Rede Neural Multi-Camadas com realimentação, visando a solução de problemas de otimização estáticos irrestritos e restritos. Um novo esquema de atualização dos pesos é proposto. Este esquema é uma modificação do algoritmo *back-propagation* e foi desenvolvido com base em resultados da teoria de dualidade e esquemas do tipo subgradientes. Resultados computacionais e uma implementação paralela são apresentados, que mostram o desempenho e a consistência do modelo proposto. Detalhes de implementação e análise comparativa do comportamento da rede em relação a outras abordagens são também incluídos.

Outra classe de Rede Neurais Artificiais constituída de redes de duas camadas com realimentação também é proposta, visando a solução de problemas de otimização dinâmica discreta não aditivamente separáveis. Esta abordagem propõe um modelo recorrente generalizado de neurônio e um método direto para designar os pesos da rede e incorporar conhecimento sobre o sistema dado. Este método fundamenta-se no Princípio de Otimalidade de Bellmann e na troca de mensagens que ocorrem entre os neurônios durante o processamento químico sináptico. Uma análise comparativa dos requisitos computacionais exigidos é realizada comprovando a vantagem da abordagem proposta com relação ao algoritmo convencional da Programação Dinâmica. Problemas conhecidos de otimização como o problema da mochila e o problema do caminho mínimo, problemas de reguladores lineares discretos e um problema de planejamento de sistemas de potência a longo prazo são resolvidos para mostrar o desempenho e utilização da abordagem proposta.

Conteúdo

Capítulo 1

Introdução

1.1 Motivação e Relevância

O crescente interesse em Redes Neurais Artificiais (RNA) nos últimos anos certamente será visto pelos historiadores da Ciência da Computação como um fenômeno notável dos anos de 1980, assim como as pesquisas em Inteligência Artificial (IA) marcaram os anos de 1970. Existe, entretanto, uma diferença. Enquanto a IA atraiu majoritariamente os cientistas de computação, RNA também despertou o interesse de pesquisadores de diversas áreas: físicos, matemáticos, engenheiros, psicólogos, neuro-fisiologistas e biólogos. O foco deste novo paradigma baseia-se no reconhecimento, por esta comunidade diversa, de que o cérebro processa informações de uma forma diferente do computador convencional [ALE 89].

Pesquisadores que têm dado contribuições importantes nesta área acreditam que o cérebro, constituído de neurônios, constrói suas próprias estratégias de processamento de informações, através do que é usualmente chamado "experiência", diferindo de pesquisadores em IA, que se baseiam na premissa de que, se um conhecimento é adquirido então ele pode ser expresso por um conjunto de regras possível de ser executado num computador [ALE 89].

RNA são motivadas por sistemas neurais biólogicos, os quais resolvem problemas complexos (visão, voz, etc...), muitos deles ainda não resolvidos satisfatoriamente em computadores digitais. Pode-se dizer que a pesquisa em redes neurais tem por objetivo a construção de sistemas computacionais com capacidades similares às do cérebro humano e, neste sentido, ela já tem alcançado resultados importantes. RNA proporcionam altas velocidades de processamento devido ao grande número de processadores (elementos de processamento simples) com alto grau de conectividade entre eles [TAGL 91].

Algumas das discussões atuais em RNA foram iniciadas cerca de 10 anos antes da invenção do computador digital. O modelo de neurônio proposto por McCulloch e Pitts [McCUL 43] em 1943, é ainda o elemento básico da maioria dos modelos de RNA. Por outro lado, os anos de 1960 foram muito produtivos, destacando-se neste período por exemplo, os trabalhos de Rosenblatt [ROSE 59], Widrow e Hoff [WIDR 60], que propuseram Perceptron e Adaline - *Adaptative Linear Element*, respectivamente.

Em 1969, Minsky e Papert [MINS 69] provocaram um esfriamento da pesquisa nesta área nos EUA, ao afirmarem que, enquanto que existia uma regra de aprendizado muito simples (regra do Perceptron) para os problemas que podem ser resolvidos por uma rede sem camadas intermediárias, não existia nenhuma para redes com camadas intermediárias. Eles mostraram também que mesmo para problemas simples como o do OU exclusivo, seria necessário, no mínimo, uma rede com 3 camadas (1 intermediária). Na Europa, contudo, alguns pesquisadores, como E. Caianiello [CAIA 75,CAIA 75], T. Kohonen [KOHO 84, KOHO 88], I. Aleksander [ALE 78,ALE 84] e outros, continuaram a desenvolver suas pesquisas com contribuições significativas.

O desenvolvimento de novos algoritmos de aprendizado, devido ao trabalho de Werbos [WERB 74] , Le Cun [LeCUN 86], Parker [PARK 85] , Rumelhart et al. [RUME 86]; fundamentos teóricos melhorados, devido a Grossberg [GROSS 76] , Amari [AMAR 72. AMAR 77] , Fukushima [FUKU] e Kohonen [KOHO 88]; novos sistemas de computação para estudos de simulação e melhores tecnologias de implementação [HAMM 90], [HOLL 89]. [MEAD 89], [MEAD 90], provocaram o ressurgimento da área.

Numerosas aplicações têm sido investigadas usando RNA, entre elas, Reconhecimento de Padrões [CARP 83,WIDR 88], Processamento de Sinais [WAIB 89,GORM 88], Processamento de Imagens [COTT 87,FUKU 88,FUKU 88,LIN 91], Processamento de Fala [SEJN 87,WAIB 89.HAMP 90], Robótica, Controle [NARE 90], Otimização [HOPF 85. TANK1 86,TANK2 86,KENN 88], Coloração de Grafos [DAHL 87]. Um exemplo particular de problema de Classificação de Óleos Vegetais submetidos a análise cromatográfica gasosa é discutido em [FRA9 93]. Neste caso, três importantes modelos de redes neurais: Rede de Hopfield, Rede de Hamming e Rede Multi-Camadas, foram usadas em Reconhecimento de Padrões.

Este trabalho é voltado ao estudo de redes neurais dentro da perspectiva de teoria de sistemas, particularmente da teoria de otimização de sistemas. Avanços na Teoria de Sistemas vêm sendo obtidos através da combinação da matemática, modelagem, computação e experimentação, elementos essenciais ao estudo de redes neurais. As diferentes áreas da pesquisa atual em Teoria de Sistemas, incluindo sistemas adaptativos e de aprendizagem, sistemas não-lineares, sistemas estocásticos, sistemas hierárquicos-multiníveis, são diretamente relevantes no estudo de sistemas neurais.

Este trabalho está relacionado principalmente com o uso de redes neurais na otimização de sistemas não lineares. Embora avanços expressivos na área de otimização de sistemas não-lineares, tenham sido testemunhados nas últimas décadas problemas complexos de otimização são ainda difíceis de serem tratados por técnicas e arquiteturas convencionais, fazendo-se necessária a pesquisa de métodos alternativos que exploram arquiteturas de processamento inerentemente paralelas e adaptativas. Além disso, métodos de otimização são essenciais ao desenvolvimento de mecanismos de aprendizagem e adaptação de sistemas em geral e de redes neurais em particular.

1.2 Trabalhos Relacionados

Primeiramente, dois enfoques devem ser considerados. O primeiro é o uso de RNA na solução de problemas de otimização e o segundo, é a aplicação de técnicas de otimização na concepção de RNA.

Vários problemas de otimização vêm sendo objeto de estudo nos últimos anos, onde têm-se proposto RNA para resolver problemas de Programação Linear (PL) [HOPF 86], problemas complexos como o Problema do Caixeiro Viajante (PCV) [HOPF 85] e problemas de Programação Não-Linear (PNL) [KENN 88].

Hopfield é um dos responsáveis pelo crescente interesse na área. Suas propostas, conhecidas como Redes de Hopfield, resolvem o PCV, e podem ser vistas também como *Content Addressable Memory* (CAM) [HOPF 85]. Esta classe de redes tem sido objeto de estudo de vários pesquisadores. Por exemplo, Carpenter, Kohen e Grossberg [CARP 83] mostraram que o requisito exigido por Hopfield em [HOPF 85], de que a matriz de interconexões deve ser simétrica, não é essencial para a existência de pontos de equilíbrio estáveis. Wilson e Pawley [WILS 88] apresentaram um estudo sobre a estabilidade da rede para resolver o PCV. Uma outra análise deste problema foi realizada por Davis [DAVI 89], análise esta feita em relação às entradas *bias* e as distâncias entre cidades. Uma modificação da rede proposta para o PCV foi sugerida por Mehta [MEHT 90]. Michel, Farrel e Porod [MICH 89] apresentam uma análise quantitativa de redes neurais, incluindo as Redes de Hopfield. Uma outra análise destas redes, tanto como CAM como para o PCV, baseada na geometria do subespaço gerado pelos auto-valores degenerados da matriz de interconexões, é apresentada por Aiyer et al. [AIYE 90]. Em [AIYE 90] e [KUH 89] é feito um estudo sobre a capacidade de armazenamento da CAM, onde um limitante que relaciona o número de padrões, M, que podem ser armazenados e a dimensão dos padrões, N, é fornecido.

Existem outros trabalhos de relevância na área de otimização. Ramanujam [RAMA 88] mostra como alguns problemas de Otimização Combinatorial, tais como, Particionamento

de Grafo, Cobertura de Vértices, Conjunto Independente de Nós, Clique Máximo podem ser mapeados na Rede de Hopfield. Dahl [DAHL 87] mostrou como o mapeamento pode ser feito para problemas de coloração de mapas e grafos. Muitos destes problemas vêm sendo resolvidos também pela técnica de *Simulated Annealing* (SA) [KIRK 83]. Embora a técnica SA tenha sido muito eficiente em problemas práticos, como os que surgem em projetos de chips VLSI, Problemas de Cobertura de Vértices, Particionamento de Grafo, em termos de qualidade de solução, tal técnica consome muito tempo de processamento. Por ser sequencial, a paralelização eficiente da SA é uma área ativa de pesquisa e é por isso que RNA pode ser considerada como uma alternativa para tais problemas.

Uma abordagem diferente daquela proposta por Hopfield, foi desenvolvida por Peng [PENG 89] para a solução de problemas de Cobertura de Vértices. Esta abordagem usa o grafo que representa o problema como a rede neural, sem quaisquer modificações. Convergência e propriedades deste modelo são estabelecidas com a introdução de uma função energia monótona não-crescente.

Hellstrom [HELS 90] mostrou que certos problemas de otimização não podem ser "mapeados" em Redes de Hopfield e propôs como alternativa, uma rede com unidades intermediárias para resolver o Problema da Mochila.

Problemas de *Job-Shop Scheduling* foram também resolvidos por rede neural para Programação Linear Inteira (PLI), também com base numa modificação da rede de Hopfield [FOO 88].

Mas, é o trabalho de Tagliarini et al. [TAGL 91], baseado também no modelo de Hopfield, que realmente enaltece o uso de RNA na área de Otimização. Ele demonstrou que é possível empregar uma abordagem sistemática para projetar RNA para aplicações de otimização, e que redes neurais de grande porte são capazes de produzir soluções de alta qualidade para problemas complexos.

Vale ressaltar aqui que, da mesma forma como RNA é uma abordagem alternativa e vantajosa para obtenção da solução de diferentes problemas de otimização, ela tem feito uso das técnicas existentes na teoria de otimização. Redes Neurais Multi-Camadas, cujo uso para Reconhecimento de Padrões e Identificação de Mapeamentos Não-Lineares é bem conhecido, pode ser citada como um exemplo típico.

A partir de conjuntos de pares de entradas e saídas que definem uma função estática, é possível propor uma rede neural multi-camadas que codifique tal função. O processo de aprendizado envolve a determinação dos pesos, de modo que a aproximação da função dada é obtida de maneira ótima. O algoritmo comumente usado para atualização dos pesos é um algoritmo do tipo gradiente, conhecido como *Back-Propagation* (BP). BP tem sido intensamente utilizado como algoritmo de aprendizado em redes neurais. No entanto, o tempo de aprendizado é geralmente grande, principalmente quando a dimensão

do problema também é grande.

Por ser um método do tipo gradiente, existem muitas variações do BP que têm sido implementadas pela comunidade de redes neurais. A mais popular destas variações é a que inclui um termo *momentum* na atualização dos pesos [RUME 86]. A maioria destas variações adotam implementações de técnicas conhecidas de otimização, tais como: Pesquisa Linear [JONE 90], Gradiente Conjugado [JOHA 90] e Métodos Quasi-Newton [BECK 88], Método de Newton [FAHL 88], como tentativas de tornar o algoritmo mais rápido. Outras propostas para melhoria do BP, bem como análise de seu desempenho podem ser encontradas em [STOR 87], [FUKU 91], [YANG 90], [BURR 91] e [NARE 90].

1.3 Proposta da Tese

Neste trabalho propõe-se uma Rede Neural Multi-Camadas com realimentação visando a solução de problemas de otimização convexos estáticos irrestritos, cujo esquema de atualização dos pesos é uma modificação do algoritmo BP. Uma generalização da rede é também proposta para a solução de problemas convexos restritos, usando teoria da dualidade e esquemas do tipo subgradientes.

Em adição, uma nova abordagem é proposta como um método sistemático de programação de redes neurais, incorporando conhecimento sobre a estrutura do problema de otimização associado a um sistema. Esta abordagem, baseada em um modelo recorrente de neurônio, fundamenta-se no Princípio de Otimalidade de Bellmann e viabiliza a criação de redes neurais para a solução de problemas de otimização que envolvem variáveis discretas de decisão. Este método é a seguir generalizado para incorporar o Princípio de Bellmann-Zadeh, visando a solução de problemas de otimização nebulosos. Finalmente, propõe-se uma generalização do modelo do neurônio recorrente e desenvolve-se um método de programação de redes para aplicações a problemas de controle ótimo discreto.

1.4 Organização da Tese

No capítulo II, faz-se uma revisão dos principais métodos de otimização irrestrita, salientando o fato que muitas RNA incorporam técnicas de otimização como mecanismos de aprendizagem. Baseados em resultados da teoria de otimização, uma rede neural artificial com realimentação da saída, constituída de três camadas, é proposta para resolver problemas de otimização irrestritos. Em seguida, uma generalização do esquema proposto é desenvolvida para resolver problemas de otimização restritos. Resultados computacionais são incluídos para mostrar o desempenho da rede neural proposta. Além disso, uma

implementação paralela é apresentada para um sistema de 32 *transputers*, cujo simulador contruído reflete a estrutura da rede. Detalhes de implementação e análise comparativa do comportamento da rede em relação a métodos tradicionais de otimização e outras abordagens são também apresentados.

No capítulo III, tanto a resolução de problemas determinísticos como a resolução de problemas de otimização *fuzzy*, através de programação dinâmica são focalizados. Uma rede neural constituída de duas camadas com realiamentação para resolver problemas de programação dinâmica é proposta. Mostra-se que, o esquema desenvolvido para programação da rede, baseado na troca de informações que ocorrem entre os neurônios durante o processamento químico sináptico, é equivalente ao algoritmo da programação dinâmica. Uma vez que este último algoritmo exige muitos recursos computacionais (e.g. memória, tempo computacional), uma análise dos requisitos computacionais exigidos pela abordagem proposta é realizada. Além disso, mostra-se que, com algumas modificações, a rede proposta pode também resolver problemas de otimização *fuzzy*. Aplicações, tais como o problema da mochila, problema do caminho mínimo e problema de planejamento de sistemas de potência a longo prazo são resolvidos para mostrar como a abordagem proposta pode ser utilizada na resolução de tais problemas.

No capítulo IV, problemas de controle ótimo multi-estágios e redes neurais são tratados. Diferentes abordagens sobre como incorporar redes neurais em estratégias de controle ótimo são revistas. Propõe-se uma generalização de modelos de neurônios recorrentes e, a partir desta generalização, uma rede neural para resolver problemas de controle ótimo multi-estágios é proposta, com um método direto para designar seus pesos e incorporar conhecimento sobre o sistema considerado. Aplicações são apresentadas para ilustrar o comportamento da rede.

Capítulo 2

Otimização de Sistemas: Métodos Estáticos

2.1 Introdução

Neste capítulo, inicialmente, apresenta-se resultados da teoria da otimização, que são necessários à compreensão das seções que se seguem. Em seguida, uma rede neural artificial com realimentação da saída, constituída de três camadas é proposta para resolver problemas de otimização irrestritos e restritos, convexos e contínuos. Resultados computacionais são também incluídos para mostrar o desempenho da abordagem proposta.

2.2 Problemas de Otimização Irrestritos

Consideram-se problemas de otimização da forma:

$$\begin{aligned} &\min_{x} \quad f(x) \\ &\text{sujeito a} \quad x \in \Omega \end{aligned} \tag{2.1}$$

onde f é uma função real e Ω, o conjunto de soluções factíveis, é um subconjunto aberto do R^n. Quando $\Omega = R^n$ o problema é chamado *problema de otimização irrestrito.*

Neste capítulo, primeiramente, propõe-se uma rede neural artificial para resolver o problema (2.1). A rede possui uma topologia com três camadas e com realimentação da saída. A primeira camada contém as entradas da rede, x, e a última camada, as saídas y,

que são realimentadas para a camada de entrada. A rede tem exatamente n processadores (nós) em cada camada onde n é a dimensão do problema (2.1).

Antes de apresentarmos a modificação na RDG (Regra Delta Generalizada) utilizada na solução do problema (2.1) e a rede correspondente, as técnicas de otimização existentes para a solução desse problema são revistas de forma suscinta. Em seguida, apresentaremos a RDG, bem como a modificação proposta como parte da solução do problema (2.1).

2.2.1 Métodos de Otimização Irrestrita

Existem vários métodos para resolver o problema (2.1), que podem ser classificados segundo o tipo de informação que requerem:

- Métodos de Busca: Estes métodos usam somente informações sobre a função $f(x)$ e não utilizam nenhuma informação sobre as derivadas da função $f(x)$. Em geral, são lentos e raramente usados na prática. Entretanto, métodos de busca unidimensional podem ser usados para incrementar o desempenho de métodos que usam informações sobre a derivada da função.

- Métodos que usam a primeira derivada (Gradiente): Estes métodos usam informações sobre a função e sua derivada. O gradiente $\nabla f(x)$ é um vetor dado por:

$$\nabla f(x) = (\frac{\partial f(x)}{\partial x_j}) \quad j = 1, 2, \ldots, n.$$

  Na prática, entre os métodos de otimização, eles são os mais utilizados porque são relativamente rápidos. Entre os métodos gradientes encontram-se:

  - Gradiente Simples
  - Gradiente Conjugado
  - Quase-Newton, etc $\cdots$

- Métodos que usam segunda derivada (Hessiana): Estes métodos usam informações sobre a função $f(x)$ e de suas derivadas de 1a. e 2a. ordem. A Hessiana $H(x)$ é uma matriz real, nxn, dada por:

$$H = (\frac{\partial f(x)}{\partial x_i \partial x_j}) \quad i = 1, 2, \ldots, n; \quad j = 1, 2, \ldots, n$$

  Esses métodos são em geral, menos usados que os métodos que usam primeira derivada, porque eles requerem mais informação e frequentemente mais tempo de cálculo. Eles são os mais rápidos, especialmente, quando o ponto x_0 inicial é próximo da solução.

Em geral, esses métodos podem ser usados para resolver (2.1) e encontram um ponto de mínimo local x^* tal que $f(x^*) \leq f(x)$ para todo x numa vizinhança de x^*. É possível que existam vários mínimos locais, a menos que a função objetivo $f(x)$ tenha propriedades especiais (e.g., convexidade).

Se x^* é um mínimo local de $f(x)$, então a norma de $\nabla f(x^*)$ é zero e $H(x^*)$ é definida semi-positiva. Somente, condições adicionais sobre $f(x)$, garantirão que este mínimo local é também global. Na prática, vários pontos iniciais são considerados, sendo o algoritmo usado para cada um deles, e o melhor x^* é escolhido como candidato a ponto de mínimo global.

Todos esses métodos seguem a seguinte estratégia:

- selecionar uma estimativa inicial da solução, x_0.
- encontrar uma direção de busca d_k, partindo-se de x_k, que garanta uma diminuição na função. Isto é feito se o gradiente $\nabla f(x_k^*)$ for diferente de zero. Se a $\| \nabla f(x^*) \|$ for suficientemente pequena, então a solução ótima foi obtida e o processo pára.
- determinar o tamanho do passo γ_k, na direção d_k, de modo que uma diminuição suficiente na função objetivo seja obtida, i.é.,

$$f(x_k + \gamma_k d_k) < f(x_k).$$

- atualizar a solução de acordo com

$$x_{k+1} = x_k + p_k \text{ onde } p_k = \gamma d_k = \Delta_{k+1}.$$

Métodos	Direções
Gradiente Simples	$p_k = \gamma d_k = -\gamma \nabla f(x_k^*)$
Gradiente Modificado	$p_k = \gamma d_k = -\gamma[\nabla f(x_k^*) + m p_{k-1}]$
Steepest Descent	$p_k = \gamma_k d_k = -\gamma_k \nabla f(x_k^*)$
Gradiente Conjugado	$p_k = \gamma_k d_k = -\gamma_k[\nabla f(x_k^*) + b_k p_{k-1}]$
Quase-Newton	$p_k = \gamma_k d_k = -\gamma_k S(x_k) \nabla f(x_k^*)$
Newton	$p_k = \gamma_k d_k = -\gamma_k [H(x_k)]^{-1} \nabla f(x_k^*)$

Tabela 2.1

Os métodos diferem entre si pela maneira com que p_k é calculado, como pode ser observado na Tabela 2.1. onde γ é o tamanho do passo, d é o vetor direção, m é um valor constante fixado, b_k é um valor constante variável a cada passo k, que envolve o cálculo de dois produtos internos, $S(x)$ é uma matriz de tamanho nxn, que é construída baseada nas diferenças entre gradientes sucessivos e vetores obtidos e, finalmente, $H(x)$ é uma matriz Hessiana de ordem n.

Informações mais detalhadas sobre esses métodos podem ser encontradas, por exemplo, em [LUEN 73].

2.2.2 Regra Delta Generalizada (RDG)

Muitas RNA incorporam técnicas de otimização como mecanismos de aprendizagem. Uma rede Multi-Camadas é um exemplo, visto que o esquema de atualização dos pesos, utilizado por esta rede e conhecido como RDG, implementa o método do Gradiente Simples (Tabela 2.1). No que segue, adotaremos a mesma notação empregada por Rumelhart [RUME 86].

Consideremos uma rede Multi-Camadas como a mostrada na Figura 2.1. A camada de entrada é a camada inferior da Figura 2.1 e a camada de saída é a camada superior da rede. As unidades (neurônios) que estão nas camadas de entrada e saída são chamadas unidades de entrada e unidades de saída, respectivamente. Podem existir inúmeras camadas, chamadas camadas intermediárias, entre as camadas de entrada e saída. Se a unidade está numa camada intermediária ela é chamada de unidade intermediária. Toda unidade envia sua saída para a camada superior a sua e recebe sua entrada da camada inferior, através das interconexões existentes entre as camadas. A cada interconexão está associado um peso w. Entre unidades de uma mesma camada, neste caso, não há comunicação.

A rede Multi-Camadas, geralmente, é usada para estabelecer um mapeamento entre dois conjuntos de dados. Desta forma, a cada entrada apresentada à rede, uma correspondente saída desejada deve também ser fornecida. O conjunto que contém todos os pares de vetores de entrada/saída é chamado conjunto de treinamento. Como as entradas e saída são fornecidas à rede, as variáveis restantes são os pesos correspondentes às interconexões existentes entre as várias camadas.

O esquema de aprendizado da rede pode ser descrito do seguinte modo. O sistema (a rede) utiliza o vetor de entrada para produzir seu próprio vetor de saída. A seguir, compara-se este com o vetor de saída desejado. Se eles são aproximadamente iguais, então diz-se que ocorreu o aprendizado. Caso contrário, os pesos são modificados com a finalidade de diminuir o erro.

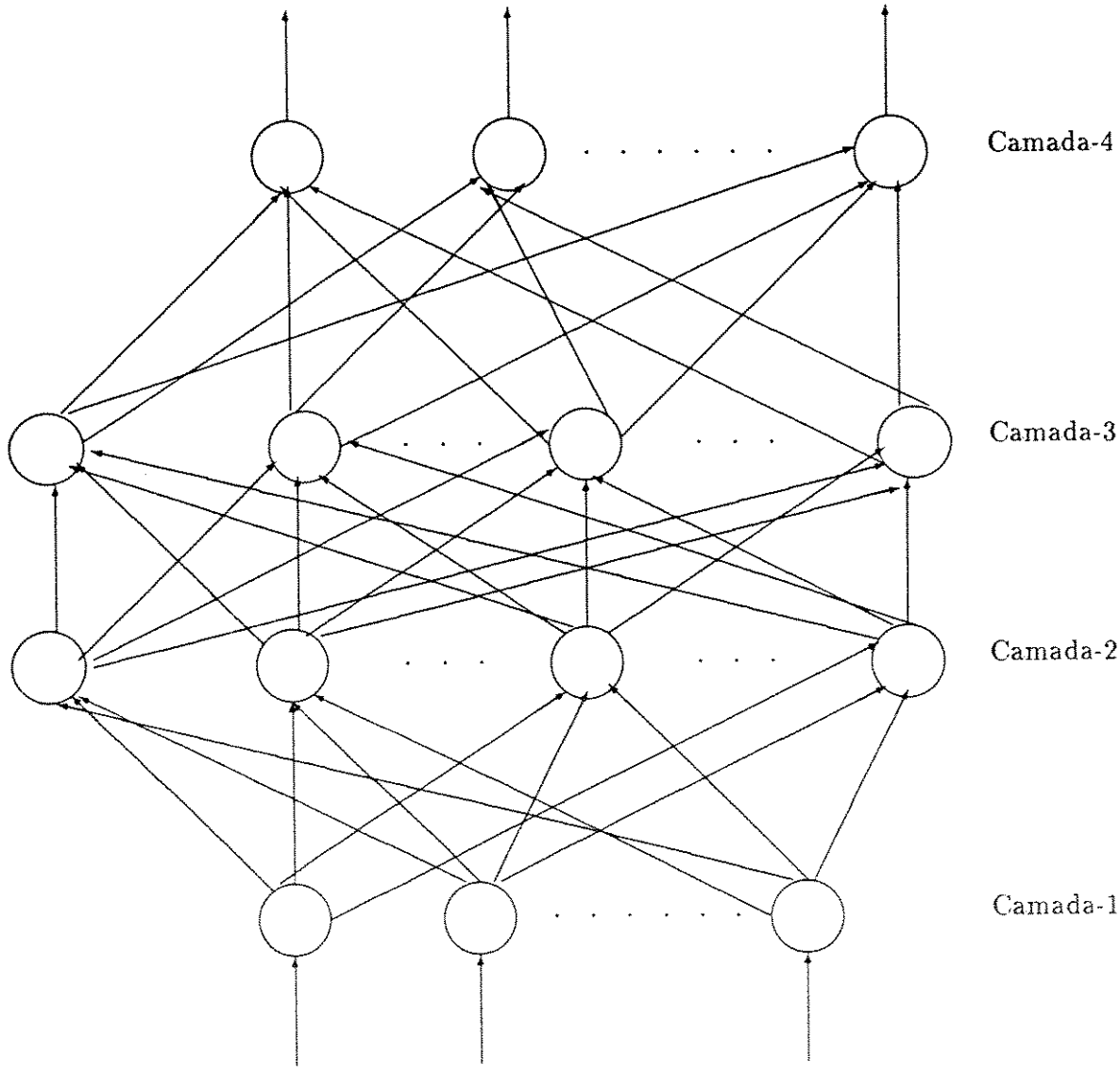


Figura 2.1: Uma rede neural multi-camadas

O erro quadrático, referente a cada par p apresentado a rede, pode ser definido por:

$$E_p = \frac{1}{2}\sum_j (t_{pj} - u_{pj})^2$$

onde t_{pj} é a j-ésima componente do vetor de saída desejado e u_{pj} é a j-ésima componente do vetor de saída obtido pela rede.

Desta forma, o problema de otimização associado à aprendizagem pode ser formulado como:

$$\min_{w_{ij}} \; E_p = \frac{1}{2}\sum_j (t_{pj} - u_{pj})^2$$

Como função de ativação de cada neurônio, adota-se a função de ativação não-linear, conhecida como função logistica ou *sigmoid.* Esta função é não-decrescente e diferenciável, sendo dada por:

$$u_{pj} = g_j(net_{pj}) = \frac{1}{1 + \epsilon^{-net_{pj}}} \tag{2.2}$$

onde

u_{pj} é a j-ésima componente do p-ésimo vetor de saída,

$net_{pj} = \sum_i w_{ij}.u_{pi}$ é a j-ésima componente do p-ésimo vetor de entrada.

Para mostrarmos que a RDG implementa o método do Gradiente Simples (ver Tabela 2.1), devemos mostrar que a direção p é:

$$p = -\gamma \; \frac{\partial E_p(w)}{\partial w_{ij}} = \Delta_p w_{ij}.$$

De acordo com Rumelhart [RUME 86], a RDG usando a função de ativação *sigmoid* é dada:

$$w_{ij}(k+1) = w_{ij} + \gamma \; \delta_{pj} u_{pi}$$

onde γ é uma constante positiva chamada velocidade de aprendizado e δ_{pj}, para uma unidade de saída, é dado por:

$$\delta_{pj} = (t_{pj} - u_{pj}) u_{pj}(1 - u_{pj}) \tag{2.3}$$

e, para uma unidade intermediária, é dada por:

$$\delta_{pj} = u_{pj}(1 - u_{pj}) \sum_k \delta_{pk} w_{kj}. \tag{2.4}$$

Assim sendo, devemos mostrar que:

$$\delta_{pj} u_{pj} \sim \frac{\partial E_p(w)}{\partial w_{ij}}$$

Usando a Regra da Cadeia, tem-se:

$$\frac{\partial E_p}{\partial w_{ij}} = \frac{\partial E_p}{\partial net_{pj}} \frac{\partial net_{pj}}{\partial w_{ij}} \tag{2.5}$$

O segundo fator de (2.5) é calculado diretamente:

$$\frac{\partial net_{pj}}{\partial w_{ij}} = \frac{\partial(\sum_k w_{kj} u_{pk})}{\partial w_{ij}} = u_{pi}$$

Para calcular o primeiro fator de (2.5) deve-se aplicar novamente a Regra da Cadeia:

$$\begin{aligned} \frac{\partial E_p}{\partial net_{pj}} &= \frac{\partial E_p}{\partial u_{pj}} \frac{\partial u_{pj}}{\partial net_{pj}} = \frac{\partial E_p}{\partial u_{pj}} g'_j(net_{pj}) \\ &= \frac{\partial E_p}{\partial u_{pj}} u_{pj}(1 - u_{pj}) \end{aligned} \tag{2.6}$$

Agora para se calcular $\frac{\partial E_p}{\partial u_{pj}}$, dois casos devem ser considerados:

1) u_{pj} é a saída de um nó na camada de saída da rede. Neste caso, a partir da definição de E_p tem-se:

$$\frac{\partial E_p}{\partial u_{pj}} = -(t_{pj} - u_{pj})$$

Assim, para este caso,

$$\frac{\partial E_p}{\partial net_{pj}} = -(t_{pj} - u_{pj})\, u_{pj}\, (1 - u_{pj}) = -\delta_{pj}.$$

e, portanto,

$$\delta_{pj} u_{pi} = -\frac{\partial E_p(w)}{\partial w_{ij}}.$$

2) u_{pj} é a saída de um nó na camada intermediária. Para obtermos $\frac{\partial E_p}{\partial u_{pj}}$, usamos mais uma vez a Regra da Cadeia:

$$\begin{aligned} \frac{\partial E_p}{\partial u_{pj}} &= \sum_k \frac{\partial E_p}{\partial net_{pk}} \frac{\partial net_{pk}}{\partial u_{pj}} = \sum_k \frac{\partial E_p}{\partial net_{pk}} w_{jk} \\ &= \sum_k -\delta_{pk} w_{jk} \end{aligned} \tag{2.7}$$

Logo:

$$\frac{\partial E_p}{\partial w_{ij}} = u_{pj}\ (1 - u_{pj})\ (\sum_k -\delta_{pk} w_{jk})\ u_{pi}$$

o que, de acordo com (2.4) implica que:

$$\delta_{pj} u_{pi} = -\frac{\partial E_p(w)}{\partial w_{ij}}$$

concluindo a demonstração.

A aplicação da RDG, envolve duas fases. Na primeira fase, a entrada é apresentada e propagada para frente (*forward*) através da rede, para calcular a saída u_{pj} para cada unidade. Esta saída é comparada com a saída desejada, resultando num erro δ_{pj} para cada unidade. A segunda fase, envolve um passo para trás (*backward*) através da rede, durante o qual o erro é propagado para cada unidade e as mudanças nos pesos realizadas. Por este motivo este procedimento (esquema de atualização dos pesos juntamente com aplicação da RDG) é conhecido como **algoritmo back-propagation (BP)**. Pode-se dizer que no sentido *forward* a rede avalia a função $E_p(w)$ e no sentido *backward* o seu gradiente. Redes Multi-Camadas que usam a RDG são também conhecidas como redes multi-camadas *feed-forward*.

Velocidade de Aprendizado: O procedimento de aprendizado requer apenas que a mudança no peso seja proporcional a $\frac{\partial E_p}{\partial w}$. A constante de proporcionalidade é a velocidade de aprendizado. Quanto maior é esta constante, maiores são as modificações nos pesos.

Rumelhart [RUME 86] sugere que se adote o maior valor possível para a constante velocidade de aprendizado, que não provoque oscilações. Sugere também que se acrescente um termo *momentum*, i.é.,

$$\Delta w_{ij}(k+1) = -\gamma\ \delta_{pj} u_{pi} + \alpha\ \Delta w_{ij}(k)$$

onde $0 < \alpha < 1$ é uma constante que determina o efeito anterior das mudanças no peso, na direção de movimento atual. Esta modificação corresponde ao método Gradiente Modificado na Tabela 2.1.

Apesar dessa modificação, o esquema back-propagation é em geral lento para convergir. Nota-se, que esses dois métodos são os únicos que constam da Tabela 2.1, que consideram o tamanho de passo fixo. Outros métodos de otimização, calculam a cada iteração, além do gradiente, também o tamanho do passo.

Outras modificações do algoritmo BP têm sido propostas, que utilizam técnicas de otimização, já citadas na Tabela 2.1, tais como, Gradientes Conjugados [JOHA 90], Métodos Quase-Newton [BECK 88], Método de Newton [FAHL 88], como tentativas de tornar o algoritmo mais rápido. Outras propostas para melhoria do BP, bem como análise do desempenho do mesmo podem ser encontradas, por exemplo, em [STOR 87], [FUKU 91], [YANG 90], [BURR 91], [KEST 58], [SARI 70], [SUTT 92], [BART 81] e [JACO 88].

Entretanto, convém ressaltar, que mesmo em face da existência destas modificações, com melhoria da velocidade de aprendizado comprovada, o algoritmo back-propagation, tal como ele é, ainda é o mais utilizado.

2.2.3 Rede de 3 Camadas para Solução de Problemas Irrestritos

Como vimos, a RDG permite a solução do problema:

$$\min_{w_{ij}}\ E_p = \frac{1}{2}\sum_j (t_{pj} - u_{pj})^2 \tag{2.8}$$

utilizando o método gradiente simples, onde E_p é uma função da diferença entre a saída obtida e a saída desejada.

Para resolver um problema de otimização, de acordo com o procedimento apresentado na seção anterior, partimos de uma solução inicial x_0 e caminhamos, segundo algum procedimento, passo a passo para a solução ótima. No entanto, não se conhece, no caso do problema original ser de otimização, a saída desejada, pois esta depende da solução ótima que se quer obter. Define-se, portanto, uma nova função erro $E(x)$, de acordo com:

$$E(x) = f(x) - f(x^*)$$

onde x^*, embora desconhecida, é suposta existir e ser única.

A topologia escolhida é uma rede constituída de três camadas com realimentação da saída. A escolha desta topologia foi realizada levando em consideração os seguintes fatos:

1) De acordo com [CYBE 89], [FUNA 89] e [HORN 89], sabe-se que qualquer função contínua pode ser aproximada num conjunto compacto, por redes multi-camadas constituídas de apenas três camadas (uma camada intermediária).

2) A rede é uma rede com realimentação porque, uma vez que não conhecemos a saída desejada, pode-se considerar a saída gerada pela rede, como uma nova entrada, supondo que esta esteja mais próxima da solução ótima que a anterior, uma vez que a Regra de Atualização dos Pesos obtida a seguir, também é do tipo gradiente.

3) Por ser uma rede do tipo multi-camadas, a Regra de Atualização dos Pesos pode ser obtida de forma análoga a RDG, porque, novamente, as variáveis da rede são os pesos.

Figura 2.2, mostra a topologia da rede considerada. Camada-0 e Camada-2 são, respectivamente, a camada de entrada e de saída da rede. A camada intermediária é denotada por Camada-1. Os pesos na Camada-1 são representados por $w_{ij}^{(1)}$ e na Camada-2, por $w_{ij}^{(2)}$. Da mesma forma, os vetores de entrada e de saída serão denotados por $u^{(k)}$ e $net^{(k)}$, $k = 0, 1, 2$. respectivamente, onde o índice k representa a camada que está sendo considerada, i.é.,

$$u_j^{(k)} = g(net_j^{(k)}) = \frac{1}{1 + e^{-net_j^{(k)}}}; \quad k = 0.1.2 \tag{2.9}$$

onde

$u_j^{(k)}$ é a j-ésima componente do vetor de saída da Camada-k.

$net_j^{(k)} = \sum_i w_{ij}^{(k)} u_i^{(k-1)}$ é a j-ésima componente do vetor de entrada da Camada-k e

$u_i^{(0)} = g(x_i^0)$.

Desde que as saídas da rede são dadas por (2.9), serão considerados somente problemas de otimização cujas soluções estão contidas no intervalo [0,1].

A rede é inicializada com pesos aleatórios entre [0,1] e uma solução inicial $x_0 = (x_0^1, x_0^2, \ldots, x_0^n)$ é fornecida como entrada. Um vetor de saída é então obtido baseado

na entrada e nos valores dos pesos. Os pesos são atualizados de acordo com a regra que desenvolveremos a seguir.

Regra de Atualização dos Pesos

Obtém-se a regra de atualização dos pesos de forma análoga àquela obtida para a RDG, ou seja.

$$\Delta w_{ij} \sim -\frac{\partial E}{\partial w_{ij}}$$

Como pode ser visto, fácilmente, no caso do problema (2.1):

$$\Delta w_{ij} \sim -\frac{\partial f}{\partial w_{ij}}$$

Consideremos dois casos:

(a) Para unidades na Camada-2:

$$\frac{\partial f}{\partial w_{ij}^{(2)}} = \frac{\partial f}{\partial net_j^{(2)}} \cdot \frac{\partial net_j^{(2)}}{\partial w_{ij}^{(2)}} = \frac{\partial f}{\partial net_j^{(2)}} . u_i^{(1)}$$

Colocando

$$\frac{\partial f}{\partial net_j^{(2)}} = -\delta_j^{(2)}$$

temos que:

$$-\delta_j^{(2)} = \frac{\partial f}{\partial net_j^{(2)}} = \frac{\partial f}{\partial u_j^{(2)}}(u^{(2)}) . \frac{\partial u_j^{(2)}}{\partial net_j^{(2)}} = \frac{\partial f}{\partial u_j^{(2)}}(u^{(2)}) . g'(net_j^{(2)})$$

Assim, obtemos:

$$\delta_j^{(2)} = -\frac{\partial f}{\partial u_j^{(2)}}(u^{(2)}) . g'(net_j^{(2)}) \tag{2.10}$$

(b) Para unidades na Camada-1:

$$\frac{\partial f}{\partial w_{ij}^{(1)}} = \frac{\partial f}{\partial net_j^{(1)}} \cdot \frac{\partial net_j^{(1)}}{\partial w_{ij}^{(1)}} = \frac{\partial f}{\partial net_j^{(1)}} . u_i^{(0)}$$

Analogamente, fazendo:

$$\frac{\partial f}{\partial net_j^{(1)}} = -\delta_j^{(1)}$$

temos que:

$$\frac{\partial f}{\partial net_j^{(1)}} = \frac{\partial f}{\partial u_j^{(1)}} \cdot \frac{\partial u_j^{(1)}}{\partial net_j^{(1)}} = \left(\sum_k \frac{\partial f}{\partial net_k^{(2)}} \cdot \frac{\partial net_k^{(2)}}{\partial u_j^{(1)}} \right) . g'(net_j^{(1)})$$

$$= \left(\sum_k (-\delta_k^{(2)}) . w_{jk}^{(2)} \right) . g'(net_j^{(1)})$$

Então,

$$\delta_j^{(1)} = \left(\sum_k \delta_k^{(2)} . w_{jk}^{(2)} \right) . g'(net_j^{(1)}) \qquad (2.11)$$

Calculando $\delta_j^{(1)}$ e $\delta_j^{(2)}$ por (2.11) e (2.10) respectivamente, finalmente obtemos:

$$\Delta w_{ij}^{(k)} = \eta . \delta_j^{(k)} . u_i^{(k-1)} \sim -\frac{\partial f}{\partial w_{ij}^{(k)}}$$

como desejávamos, onde η é um valor arbitrário pequeno no intervalo [0.1].

Resultados obtidos através de simulação da rede mostram que este esquema desempenha satisfatoriamente para problemas bem comportados, como os Exemplos 1 e 2. Investigações adicionais são necessárias para avaliar as soluções para problemas mais complicados, tais como, aqueles com funções objetivo não-convexas. Este caso deverá ser objeto de consideração futura.

Critério de Parada:

A rede neural proposta acima converge (o processo se estabiliza) se

$$\| \nabla f(x) \| \leq \epsilon$$

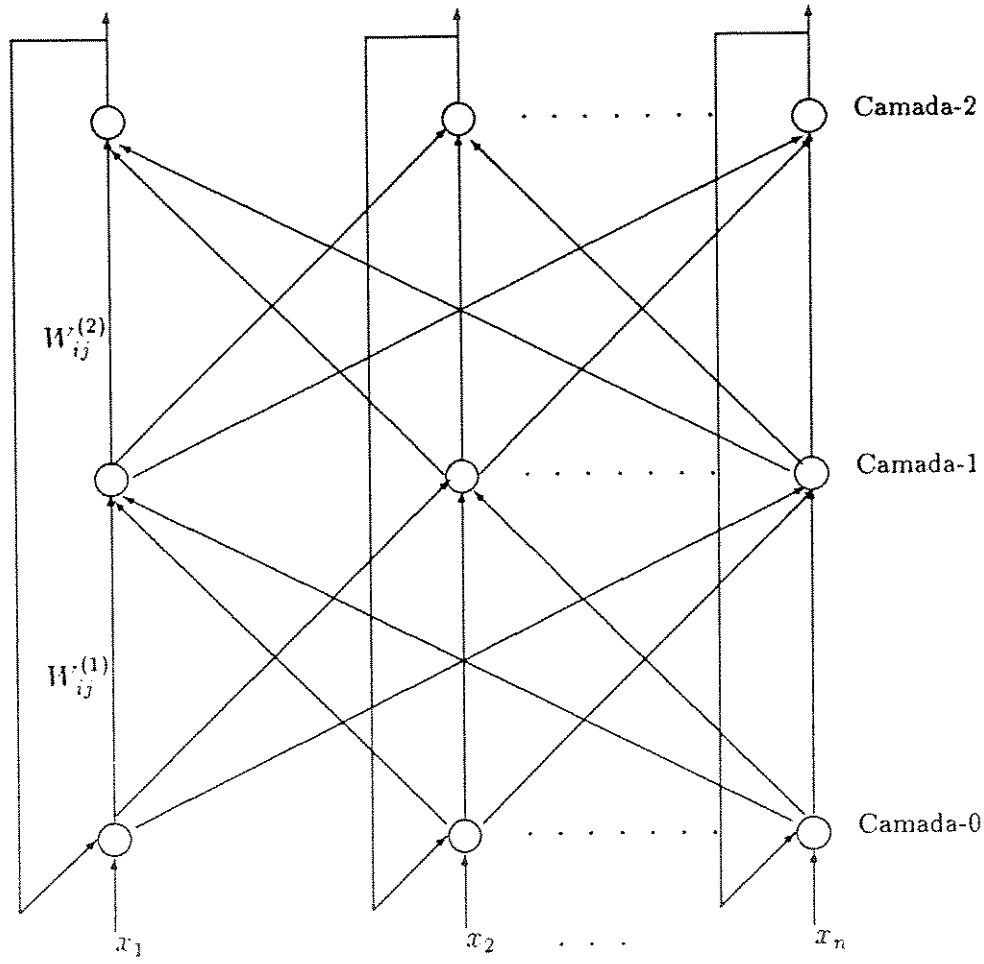


Figura 2.2: Uma rede neural de três camadas

onde ϵ é um valor arbitrário pequeno, e

$$\|v\| = \max_i |v_i|, \quad v^T = [v_1, ..., v_n].$$

Através deste esquema, caminhamos sempre em direção oposta ao gradiente de f, $-\frac{\partial f}{\partial w_{ij}}$ e, atualizamos w de acordo com o método do Gradiente Simples:

$$w_{ij}(t + \Delta t) = w_{ij}(t) - \alpha \frac{\partial f}{\partial w_{ij}}.$$

Assim, quando

$$\| \bigtriangledown f(w)\| \leq \epsilon$$

os pesos estabilizarão, i.é.,

$$w_{ij}(t + \Delta t) \sim w_{ij}(t) \Rightarrow x(t + \Delta t) = u^{(2)}(t + \Delta t) \sim u^{(2)}(t) = x(t)$$

o que implica que,

$$f'(x) = \lim_{\Delta t \to 0} \frac{f(x(t + \Delta t)) - f(x(t))}{\Delta t} \sim 0$$

numa vizinhança pequena de $x(t)$.

Logo, $x(t)$ obtido é um candidato a ponto de mínimo local de f.

Desta forma, quando os pesos estabilizam, a RNA convergiu para a solução ótima, se o problema for convexo. Para problemas não-convexos, um candidato a ponto de mínimo local é obtido.

Uma outra descrição da rede proposta pode ser encontrada em [FRA1 90] e [FRA3 91].

Os exemplos 1 e 2 ilustram o comportamento da rede neural proposta [Figura 2.2]:

Exemplo 1:

Como um primeiro exemplo, consideremos o problema:

$$min \quad (x_1 - 0.1)^2 + (x_2 - 0.2)^2 + (x_3 - 0.3)^2 + (x_4 - 0.4)^2$$

As soluções aproximadas x obtidas pela rede neural proposta na Figura 2.2 são mostradas na Tabela 2.2, considerando diferentes soluções iniciais para x^0, ϵ, η e matrizes de pesos [simétrica(S) ou nâo-simétrica(NS)].

OBS: o símbolo T denota transposto

$(x^0)^T$	ϵ	η	Matrizes Pesos	x^T	Itera ções
(.2,.15,.45,.6)	10^{-2}	0.5	S	(.104965,.199823,.299896,.399964)	159
(.2,.15,.45,.6)	10^{-3}	0.5	S	(.100498,.199975,.299990,.399996)	313
(.2,.15,.45,.6)	10^{-3}	0.5	NS	(.100496,.199977,.299991, .399997)	304
(.4,.32,.22,.11)	10^{-3}	0.5	S	(.100498,.199975,.299996,.399996)	313
(.4,.32,.22,.11)	10^{-3}	0.5	NS	(.100496,.199977,.299991,.399997)	304
(.4,.32,.22,.11)	10^{-3}	0.7	NS	(.100497,.199977..299991,.399996)	217
(.4,.32,.22,.11)	10^{-3}	0.3	S	(.1004956,.199977,.299990. 0.399997)	516

Tabela 2.2

Observe que, os resultados obtidos são próximos da solução ótima que obviamente é:

$$x^* = (0.1, 0.2, 0.3, 0.4)^T$$

A rede foi testada também, considerando diferentes soluções iniciais para x^0 e $\epsilon = 0.001$, obtendo os tempos de execução necessários para a convêrgencia quando os pesos iniciais são gerados aleatoriamente bem como, quando os pesos iniciais são pré-fixados, isto é, os valores para os pesos iniciais são fixados nos valores obtidos para os mesmos quando a rede convergiu para uma dada condição inicial. Além disso, utilizamos a rotina E04VDF da *NAG Fortran Library Routine* para resolver problemas irrestritos, obtendo a solução ótima e os respectivos tempos de execução, para cada solução inicial dada. Os resultados estão contidos na Tabela 2.3. Esses resultados dão uma idéia do desempenho da rede de três camadas proposta em relação a algoritmos convencionais. Os testes foram realizados num computador IBM-4341, utilizando a linguagem FORTRAN.

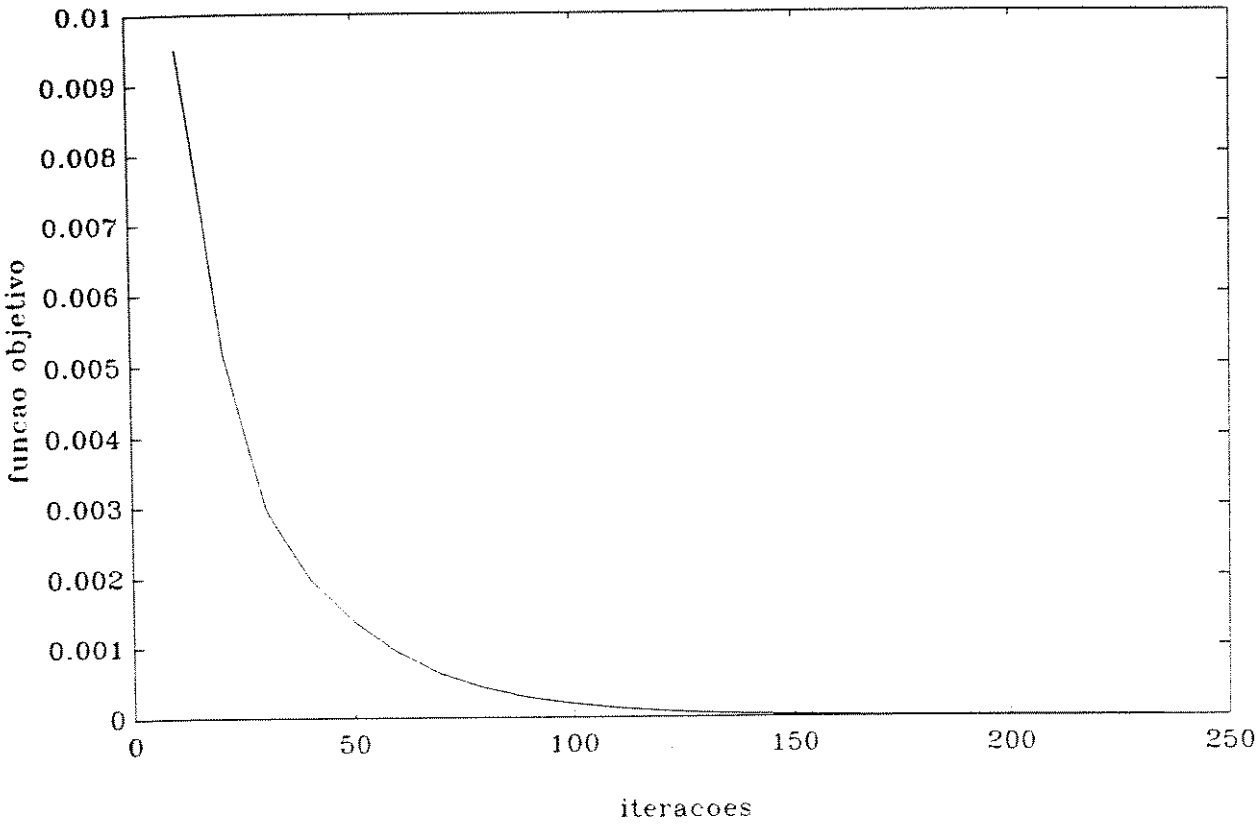


Figura 2.3: Comportamento da função objetivo para o Exemplo 1

$(x^0)^T$	NAG x^T / tempo	Pesos Aleatórios x^T / tempo	Pesos Pré-fixados x^T / tempo
(.4,.2,.1,.2)	(.1, .2, .3, .4) t = 0.101s	(.1004,.1997,.3001,.3999) t = 0.214s	(.0999,.1997,.3003,.4002) t = 0.042s
(-.1,.0,.1,.6)	(.1,.2,.3,.4) t = 0.122s	(.1004,.1997,.3001,.3999) t = 0.214s	(.1003,.1999,.3004,.3999) t = 0.015s
(.2,.15,.45,.6)	(.1,.2,.3,.4) t = 0.101s	(.1004,.1997,.3001,.3999) t = 0.214s	(.1003,.1999,.3004,.3999) t = 0.016s

Tabela 2.3

A Figura 2.3 mostra o comportamento da função objetivo em função do número de iterações, correspondente a solução obtida considerando $(x^0)^T = (0.4, 0.2, 0.1, 0.2)$, como solução inicial.

Exemplo 2:

Seja dado o problema:

$$min\ (x_1 - 0.5)^4 + (x_1 - 2x_2)^2$$

Neste caso, as soluções x obtidas pela rede neural são mostradas na Tabela 2.4, considerando diferentes soluções iniciais para x^0, ϵ, η e matrizes pesos [simétrica(S) ou não-simétrica(NS)].

$(x^0)^T$	ϵ	η	Matrizes Pesos	x^T	Itera ções
$(0.1, 0.2)$	10^{-2}	0.5	NS	$(0.6478911, 0.3246869)$	99
$(0.1, 0.2)$	10^{-2}	0.5	S	$(0.6449696, 0.323717)$	39
$(0.1, 0.2)$	10^{-3}	0.5	NS	$(0.5695122, 0.284842)$	1795
$(0.1, 0.2)$	10^{-3}	0.75	NS	$(0.5696032, 0.2848888)$	1024
$(0.1, 0.2)$	10^{-3}	0.75	S	$(0.5694886, 0.2848299)$	1142
$(0.2, 0.4)$	10^{-3}	0.5	NS	$(0.5696724, 0.2849246)$	1555
$(0.2, 0.4)$	10^{-3}	0.5	S	$(0.5694855, 0.2848284)$	1711
$(0.2, 0.4)$	10^{-3}	0.75	S	$(0.5694667, 0.2848188)$	1143

Tabela 2.4

Estes são resultados consideráveis desde que a solução ótima é:

$$x^* = (0.5, 0.25)^T.$$

Analogamente ao exemplo anterior, a Tabela 2.5 apresenta as soluções obtidas pela rede proposta e respectivos tempos de execução, tanto para o caso em que os pesos iniciais são gerados aleatoriamente, como para o caso em que eles são obtidos para uma dada solução inicial e fixados nestes valores. As soluções foram obtidas considerando diferentes soluções iniciais para x^0 e $\epsilon = 0.001$. Além disso, a tabela apresenta resultados obtidos através da execução da rotina E04VDF da NAG para resolver problemas irrestritos.

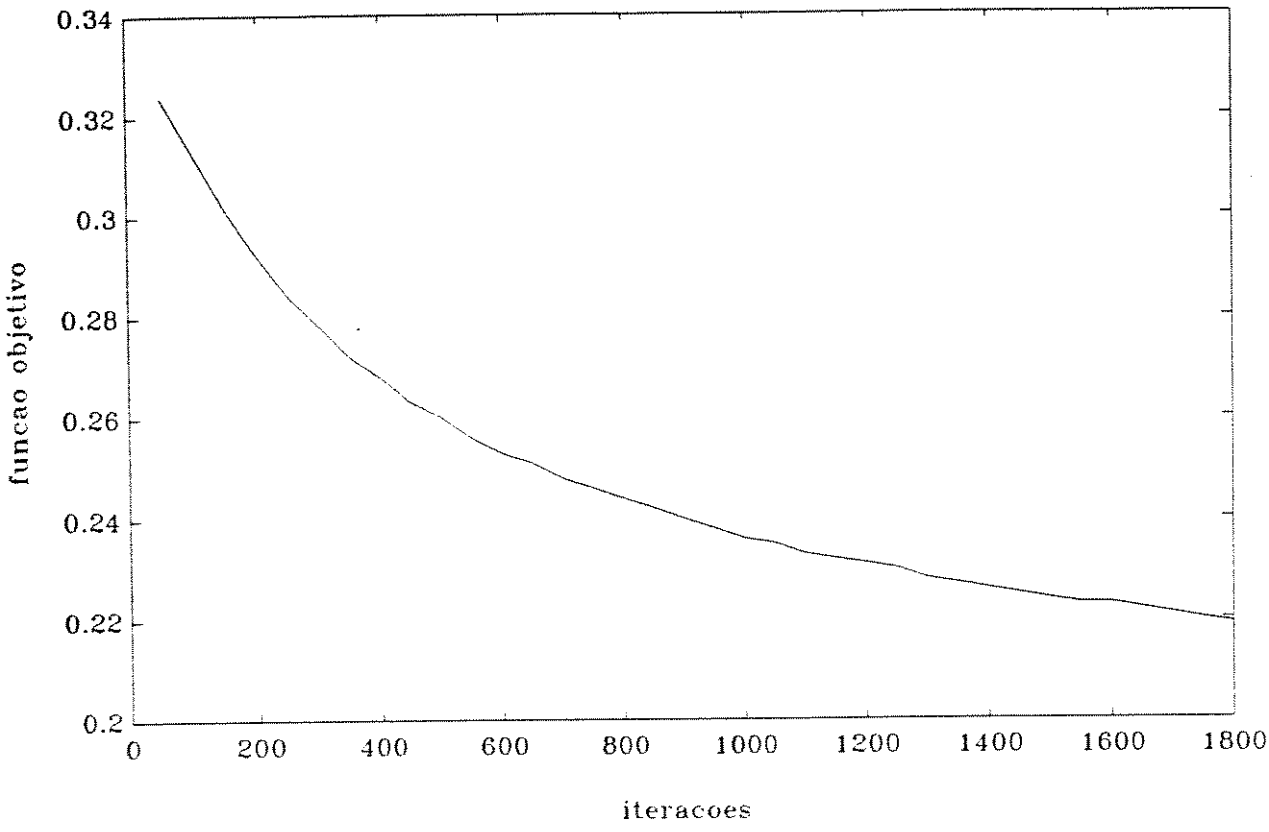


Figura 2.4: Comportamento da função objetivo para o Exemplo 2

$(x^0)^T$	NAG x^T / tempo	Pesos Aleatórios x^T / tempo	Pesos Pré-fixados x^T / tempo
(.1,.2)	(.499958, .2499788) t = 0.552s	(.557453,.278682) t = 0.602s	(.559564,.279751) t = 0.019s
(-.1,.1)	(.499948,.249974) t = .562s	(.557437,.278674) t = 0.598s	(.559491,.279715) t = 0.022s
(.1,-.1)	(.499958,.249979) t = .561s	(.557436,.278674) t = 0.598s	(.559429,.279683) t = 0.024s

Tabela 2.5

Na Figura 2.4, um esboço da função objetivo em função do número de iterações é apresentado, com base na solução obtida, partindo-se de um ponto inicial $(x^0)^T = (-0.1, 0.1)$.

Dos exemplos acima observa-se que, as soluções obtidas independem das matrizes de pesos iniciais, bem como, dos valores adotados para as constantes ϵ e η. O mesmo não acontece com o número de iterações que, como era esperado, pode aumentar ou diminuir à medida que os valores de ϵ e η são alterados.

2.3 Problemas de Otimização Restritos

Em geral, problemas de otimização não-lineares podem ser colocados na forma:

$$\begin{aligned} &\min_x \quad f(x) \\ &\text{sujeito a } r(x) = 0 \\ &\qquad g(x) \leq 0 \\ &\qquad x \in \Omega \subseteq R^n \end{aligned} \tag{2.12}$$

onde $r : R^n \rightarrow R^m$, $g : R^n \rightarrow R^p$, $m \leq n$ e as funções f, r_i, $i = 1, 2, ..., m$ e g_j, $j = 1, 2, ..., p$ são contínuas e de classe C^2. Estes problemas são *problemas de otimização restritos*.

O objetivo é o de resolver o problema (2.12), generalizando o esquema proposto anteriormente. Para isso, precisamos de alguns resultados da teoria de otimização, relativos a otimização com restrições de igualdade.

2.3.1 Otimização com Restrições de Igualdade

Se considerarmos somente restrições de igualdade, obtemos o seguinte problema:

$$\min_x \quad f(x)$$

$$\text{sujeito a} \quad r(x) = 0 \qquad (P) \tag{2.13}$$

que será chamado de problema Primal (P).

Definição[LUEN 73]: Um ponto x^* satisfazendo as restrições $r(x^*) = 0$ é chamado um *ponto regular das restrições* se os valores gradientes:

$$\nabla r_1(x^*), \nabla r_2(x^*), \ldots, \nabla r_m(x^*)$$

forem linearmente independentes.

Teorema[LUEN 73]: Seja x^* um ponto regular das restrições $r(x) = 0$ e um ponto extremo local (um mínimo ou um máximo) da função f(x) sujeito a essas restrições. Então, existe um vetor $\lambda^* \in R^m$ tal que:

$$\nabla f(x^*) + (\lambda^*)^T \nabla r(x^*) = 0.$$

O problema (P) pode ser transformado num problema de otimização irrestrito através das seguintes considerações.

Seja a função Lagrangeana associada à (P) definida por:

$$L(x, \lambda) = f(x) + \lambda^T r(x) \tag{2.14}$$

onde λ é um vetor do R^m e é chamado multiplicador de Lagrange. Além disso, consideremos também, a função dual:

$$h(\lambda) = \min_x L(x, \lambda)$$

onde o mínimo é procurado numa vizinhança pequena da solução x^*.

Associado ao problema primal (P), pode-se considerar o problema dual dado por:

$$\max_\lambda h(\lambda) \qquad (D)$$

Pode-se provar [LUEN 73] que, se x^* é um ponto regular das restrições e a matriz Hessiana de L é definida positiva, então, localmente, o problema (P) é equivalente ao problema de maximização local irrestrito da função dual $h(\lambda)$.

Desta forma, para resolver o problema (P), devemos resolver o problema de otimização irrestrito

$$h(\lambda) = \min_{x} L(x, \lambda)$$

para cada valor de λ e, em seguida, o valor máximo de $h(\lambda)$ deve ser determinado. Ainda mais, se a função dual for diferenciável em um ponto $\tilde{\lambda}$, então,

$$\nabla h(\tilde{\lambda}) = r(\tilde{x})$$

onde $\tilde{x}$ minimiza $L(x, \tilde{\lambda})$ [LUEN 73].

Isto significa que podemos atualizar o valor de λ usando um método do tipo gradiente. Mas, uma vez que, $h(\lambda)$ não é necessariamente diferenciável e geralmente não se dispõe dessa informação, o subgradiente da função dual calculado no ponto $\tilde{\lambda}$, pode ser considerado.

A definição de vetor subgradiente pode ser encontrada em Lasdon [LASD 70]. De acordo com sua definição, pode ser provado que $r(\tilde{x})$ é um subgradiente da função dual $h(\lambda)$ no ponto $\tilde{\lambda}$.

Assim, a partir destes resultados, os seguintes subproblemas podem ser resolvidos:

(1) Para λ fixado, resolver:

$$\min_{x} L(x, \lambda) \tag{2.15}$$

Este subproblema pode ser resolvido por uma abordagem similar ao caso irrestrito, determinando um valor $\hat{x}$ tal que $L(\hat{x}, \lambda)$ é mínimo.

(2) Para $\hat{x}$ obtido em (2.15), atualizar λ de acordo com a direção do subgradiente da função dual h, com um passo η.

$$\lambda^{k+1} = \lambda^k + \eta r(x^k), \tag{2.16}$$

uma vez que se deseja resolver:

$$\max_{\lambda} h(\lambda)$$

que é também um problema de otimização irrestrito.

A eficiência de métodos subgradientes depende da escolha do passo η. Alguns métodos subgradientes foram propostos por Polyack [POLI 87], Held, Wolfe e Crowder [HELD 74] e Kim, Ahn e Cho [KIM 91], onde várias sugestões sobre a escolha do parâmetro η podem ser encontradas. Entretanto, em nossa abordagem, consideramos, inicialmente, η constante.

De forma análoga a abordagem anterior (para minimização irrestrita) temos a seguinte regra de atualização dos pesos para a resolução do problema (D) (maximização irrestrita):

$$\Delta w_{ij}^{(k)} \sim \frac{\partial h}{\partial w_{ij}^{(k)}} = \eta_k \; \delta_j^{(k)} \; u_i^{(k-1)}; \;\; k = 1, 2$$

onde

$$\delta_j^{(2)} = r_j(\tilde{x}) \; g'(net_j^{(2)}); \;\; j = 1, 2, \ldots, m;$$

$$\delta_j^{(1)} = \sum_k (\delta_j^{(2)} \; w_{jk}^{(2)}).g'(net_j^{(1)}); \qquad j = 1, 2, \ldots, m$$

e η_k é um valor pequeno no intervalo [0,1].

Critério de Parada

Para cada λ obtido por (2.16), resolvemos o subproblema dado por (2.15) e o processo é repetido até que

$$\|r(x)\| \;\leq\; \epsilon$$

onde ϵ é um valor arbitrário pequeno.

A topologia da rede neural associada é mostrada na Figura 2.5. Maiores detalhes sobre a rede generalizada proposta podem ser encontrados- em [FRA2 91] e [FRA4 92].

Este esquema foi implementado e testado para problemas (convexos) simples.

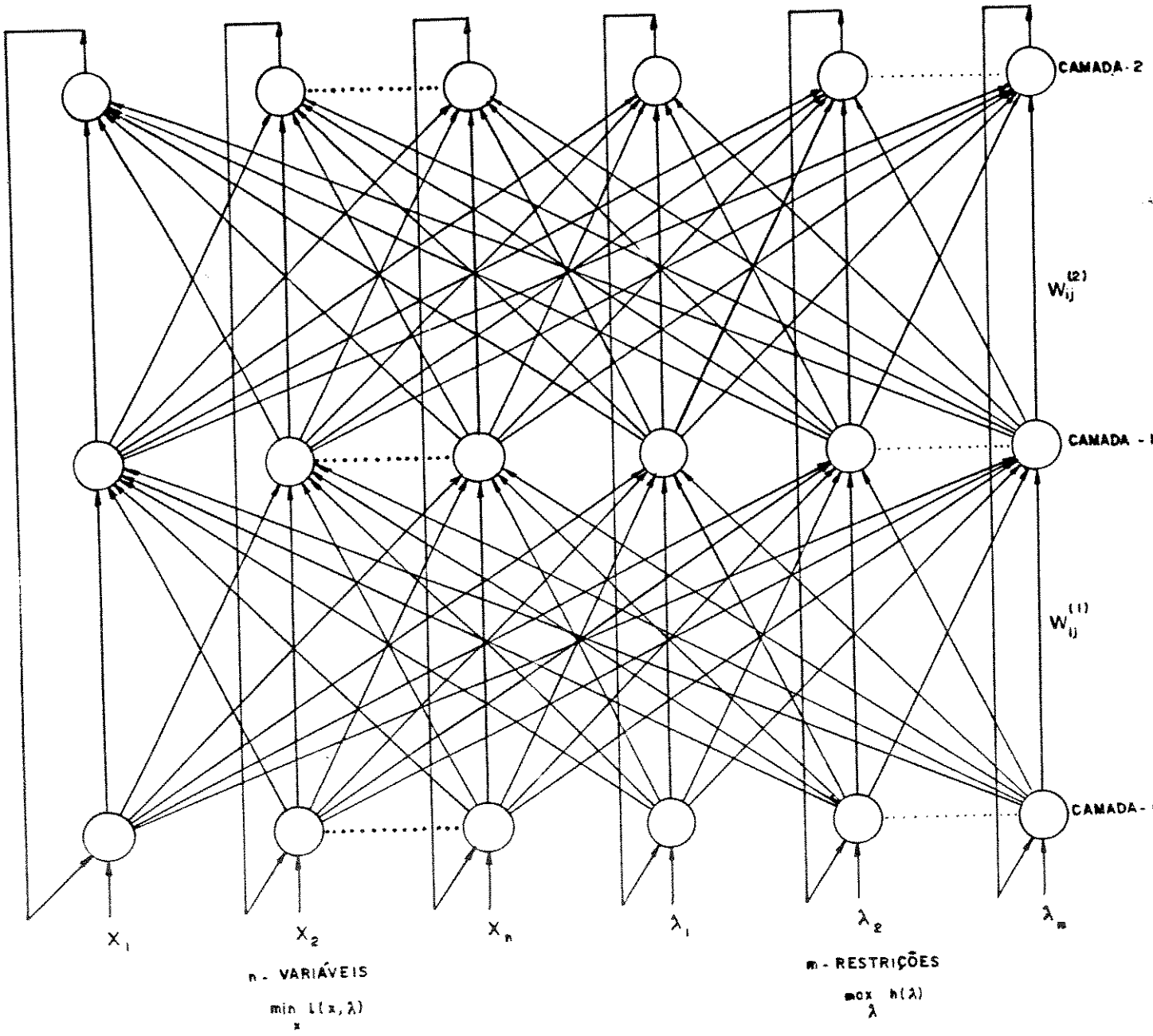


Figura 2.5: Topologia da rede neural para resolver problemas restritos

Exemplo 3:

Consideremos o problema:

$$min\ (x_1 - 0.5)^2 + (x_2 - 0.4)^2$$

$$\text{sujeito a } x_1 + x_2 = 0.8$$

As soluções aproximadas (x, λ) obtidas pela rede proposta na Figura 2.5 são mostradas na Tabela 2.6, considerando diferentes soluções iniciais (x^0, λ^0) e η.

$(x^0, \lambda^0)^T$	η	$(x, \lambda)^T$	Iterações
(.1,.2,.3)	0.5	(.44486,.34516,.10939)	1669
(.1,.2,.3)	0.75	(.44487,.34517,.10937)	1104
(.3,.1,.2)	0.5	(.44486,.34517,.10937)	1694
(.3,.1,.2)	0.75	(.44485,.34516,.10939)	1102
(-.1,.0,.6)	0.5	(.44485,.34516,.10939)	1534
(-.1,.0,.6)	0.75	(.44485,.34516,.10939)	1003

Tabela 2.6

Neste caso, a solução ótima é

$$x^* = (0.45, 0.35)^T, \quad \lambda^* = 0.1.$$

A Tabela 2.7 apresenta as soluções obtidas pela rede proposta e respectivos tempos de execução, tanto para o caso em que os pesos iniciais são gerados aleatoriamente, como para o caso em que eles são obtidos para uma dada solução inicial e fixados nestes valores. As soluções foram obtidas considerando diferentes soluções iniciais para x^0 e $\epsilon = 0.001$. Além disso, a tabela apresenta resultados obtidos através da execução da rotina E04VDF da NAG para resolver problemas irrestritos/restritos.

$(x^0, \lambda^0)^T$	NAG $(x, \lambda)^T$ / tempo	Pesos Aleatórios $(x, \lambda)^T$ / tempo	Pesos Pré-fixados x^T / tempo
(.1,.2,.3)	(.450,.3499,-.0999) t = 0.042s	(.4532,.3565,.0963) t = 2.158s	(.4532,.3565,.0963) t = 1.033s
(.2,.3,.6)	(.450,.3499,-.0999) t = 0.043s	(.4534,.3565,.0964) t = 1.462s	(.4531,.3567,.0962) t = 1.317s
(-.1,0.0,.6)	(.450,.3499,-.0999) t = 0.04s	(.448,.355,.0984) t = 1.035s	(.4534,.3565,.0964) t = 0.920s

Tabela 2.7

Na Figura 2.6, um esboço da função objetivo em função do número de iterações é apresentado, com base na solução obtida, considerando-se $(x^0, \lambda^0)^T = (0.3, 0.1, 0.2)$.

Exemplo 4:

Seja o problema não-linear:

$$min \; - \; x_1 x_2$$

$$\text{sujeito a } (x_1 - 0.3)^2 + x_2^2 = 0.05$$

Para este problema, as soluções aproximadas (x, λ) obtidas pela rede [Figure 2.5] são mostradas na Tabela 2.8, considerando diferentes valores para soluções iniciais (x^0, λ^0) e η.

$(x^0, \lambda^0)^T$	η	$(x, \lambda)^T$	Itera ções
(.40..35,.60)	0.5	(.40948110,.21179210..96974570)	1001
(.40..35,.60)	0.75	(.40723400,.20921750..98585700)	601
(.20..10,.40)	0.5	(.40988190,.21231440..96979670)	901
(.20,.10,.40)	0.75	(.41081660,.21376950..98343570)	501

Tabela 2.8

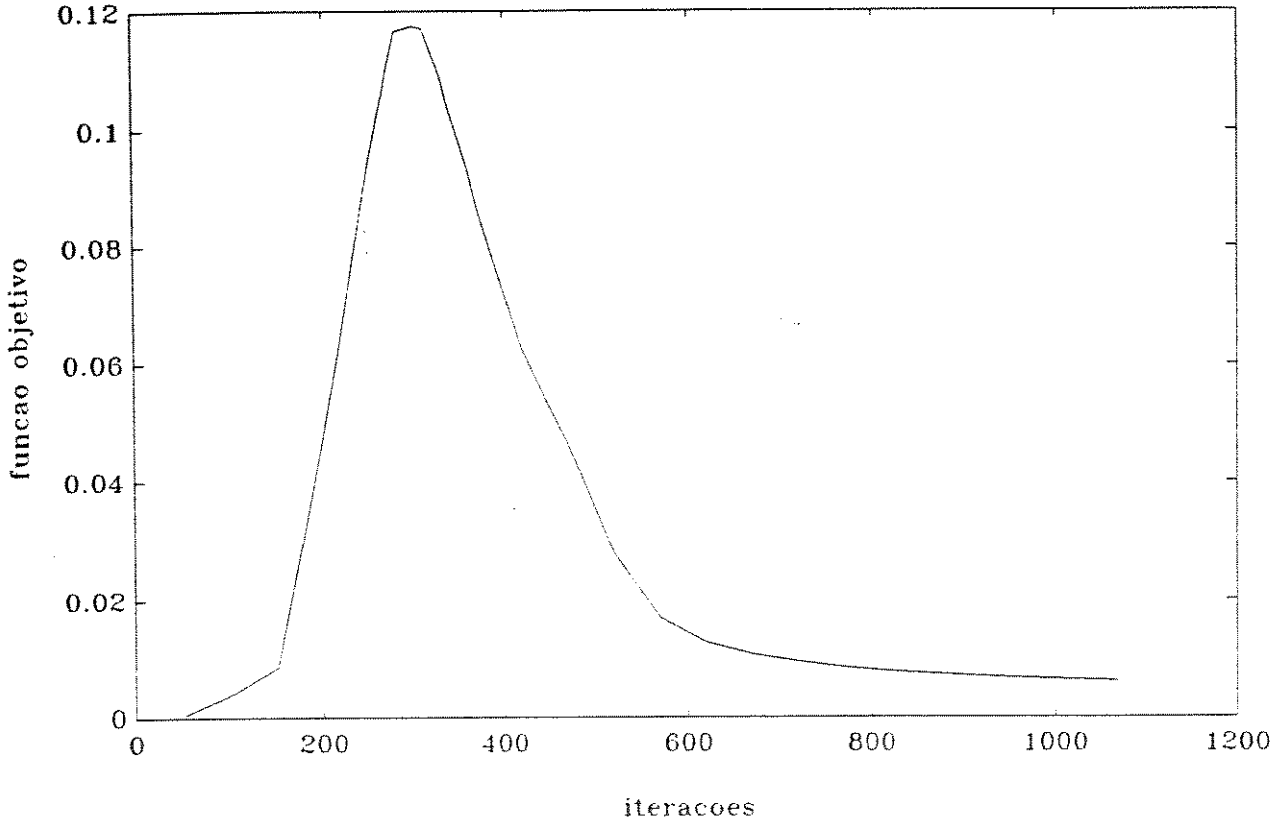


Figura 2.6: Comportamento da função para o Exemplo 3

Observe que estes resultados estão muito próximos da solução ótima:

$$x^* = (0.4, 0.2)^T, \quad \lambda^* = 1.0$$

Analogamente ao exemplo anterior, a Tabela 2.9 apresenta as soluções obtidas pela rede proposta e respectivos tempos de execução, tanto para o caso em que os pesos iniciais são gerados aleatoriamente, como para o caso em que eles são obtidos para uma dada solução inicial e fixados nestes valores. As soluções foram obtidas considerando diferentes soluções iniciais para x^0 e $\epsilon = 0.001$. Além disso, a tabela apresenta resultados obtidos através da execução da rotina E04VDF da NAG para resolver problemas irrestritos/restritos.

$(x^0, \lambda^0)^T$	NAG $(x, \lambda)^T$ / tempo	Pesos Aleatórios $(x, \lambda)^T$ / tempo	Pesos Pré-fixados x^T / tempo
(.4,.35,.6)	(.40,.199,-1.00) t = 0.092s	(.3939,.1927,1.000) t = .745s	(.3939,.1928,.999) t = .704s
(.2,.1,.4)	(.40,.199,-1.00) t = 0.113s	(.4066,.2034,1.000) t = 0.763s	(..3938,.1927,.9999) t = 0.833s
(.1,.1,.5)	(.40,.199,-1.00) t = 0.148s	(.3983,.1975,.9998) t = 0.967s	(.3939,.1928,.9999) t = 2.409s

Tabela 2.9

A Figura 2.7 mostra o comportamento da função objetivo em função do número de iterações correspondente à solução obtida considerando-se $(x^0, \lambda^0)^T = (0.2, 0.1, 0.4)$.

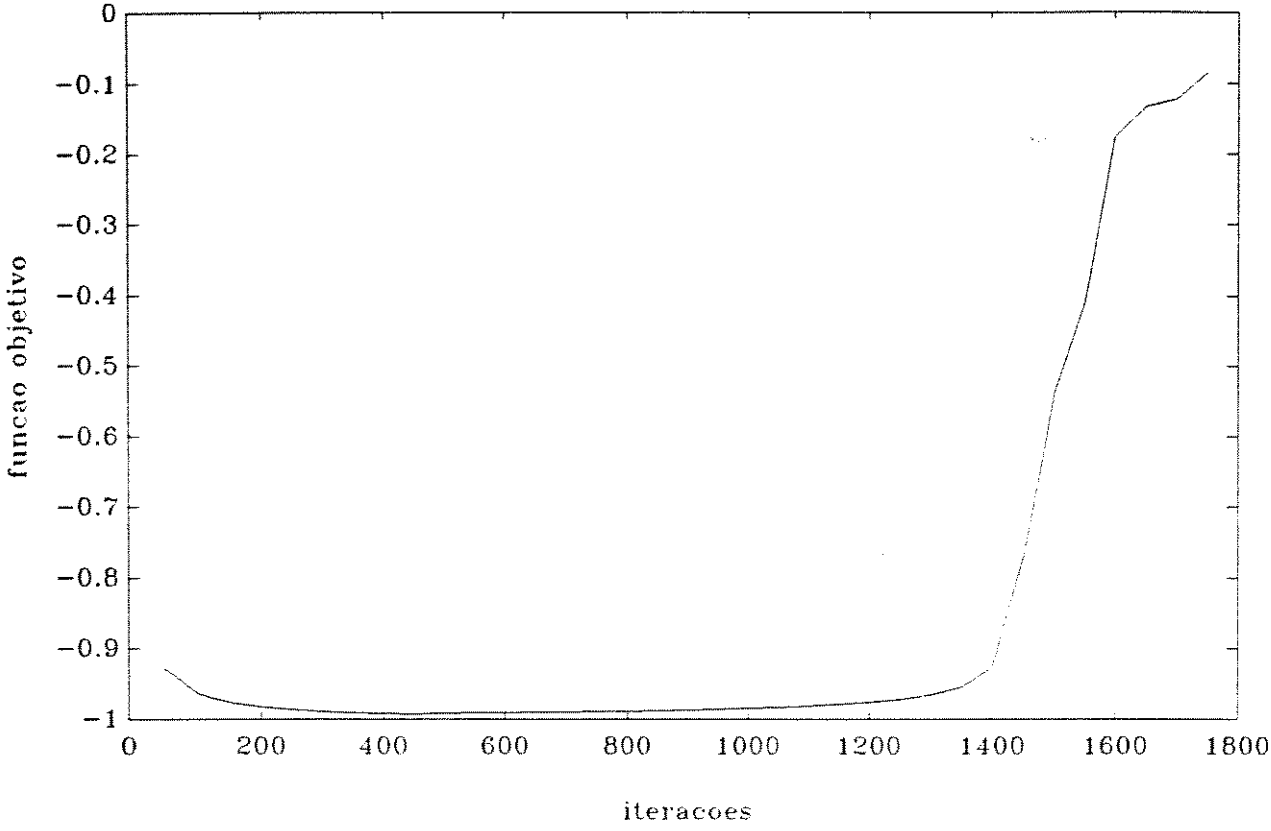


Figura 2.7: Comportamento da função objetivo para o Exemplo 4

2.4 Processamento da Rede Neural Proposta

Nesta seção apresentamos um algoritmo relativo ao processamento da rede neural proposta para resolver problemas de otimização irrestritos e restritos.

Sejam:

n - a dimensão do problema primal,

m - a dimensão do problema dual,

$x_i^t = (x_i^1, x_i^2)^t$ o vetor de entrada $(n+m)$-dimensional,

$y_i^t = (y_i^1, y_i^2)^t$ o vetor de saída $(n+m)$-dimensional,

w_{ih} - a matriz de pesos entre a camada de entrada e camada intermediária,

w_{ho} - a matriz de pesos entre a camada intermediária e a camada de saída,

Initialização

$$y_i^2 = 0$$
$$y_h^2 = 0$$
$$\delta_o^2 = 0$$
$$\delta_h^2 = 0$$

Passo Primal *Forward*

$$y_i^1 = g(x_i^1)$$
$$(x_h^1)^t = (w_{ih}^1)^t . y_i$$
$$y_h^1 = g(x_h^1)$$
$$(x_o^1)^t = (w_{ho}^1)^t . y_h$$
$$y_o^1 = g(x_o^1)$$
$$y_o^2 = x_i^2$$

Passo Primal *Backward*

$$\delta_o^1 = - \bigtriangledown f(y_o^1) . \bigtriangledown g(xo^1)$$

$$w_{ho}^1 = w_{ho}^1 + \alpha.\delta_o.y_h^1$$
$$\delta_h^1 = g'(x_h^1).(w_{ho}^{11})^t.\delta_o^1$$
$$w_{ih}^1 = w_{ih}^1 + \alpha.\delta_h.y_i^1$$
$$x_i^1 = y_o^1$$

Passo Dual

$$y_i^2 = g(x_i^2).(x_h^2)^t = (w_{ih}^2)^t.y_i y_h^2 = g(x_h^2(x_o^2)^t = (w_{ho}^2)^t.y_h y_o^2 = g(x_o^2)$$
$$\delta_o^2 = \bigtriangledown f(g_o^2).r(yo^2)$$
$$w_{ho}^2 = w_{ho}^2 + \alpha.\delta_o.y_h^2$$
$$\delta_h^2 = g'(x_h^2).(w_{ho}^{22})^t.\delta_o^2$$
$$w_{ih}^2 = w_{ih}^2 + \alpha.\delta_h.y_i^2$$
$$x_i^2 = y_o^2$$

Cálculo do Erro Local

$$\epsilon_l = \| \bigtriangledown f(y_o^1) + (x_i^2)^t.Jr(y_o^1 \|$$

Cálculo do Erro Global

$$\epsilon_g = \|r(yo^1)\|$$

Algorithmo

Faça

Início

executar o Passo Primal *Forward*

Se $\epsilon_l \leq$ limite erro local **então**

Se $\epsilon_g \leq$ limite erro global **então**

solução foi encontrada

senão

executar Passo Dual

senão

executar o Passo Primal *Backward*

Fim

enquanto solução não for encontrada

2.5 Implementação Paralela da Rede Proposta

Um implementação paralela da rede proposta foi realizada junto a University of Maryland, em College Park-MD, para verificar, preliminarmente, sua aplicabilidade prática. O sistema usado para esta implementação foi um Parsytec SuperCluster com 32 *transputers*. A cada nó da rede foi associado um *transputer* T-800 (um microprocessador com unidade de ponto flutuante de 20 MHz, 4 Kbyte de RAM e 4 canais de comunicação) com 2 Mbyte de memória local. Além desses nós na rede, um outro *transputer* com 4 Mbyte de memória local foi utilizado como responsável pela conexão com o *host*, uma Sparcstation Sun.

O simulador foi desenvolvido usando a linguagem de programação C para ser executado sob o sistema operacional Helios. Este sistema não oferece uma linguagem paralela própria para programação de *transputers* como outros compiladores. Neste ambiente, as construções paralelas são especificadas a nível de sistema operacional através de uma linguagem para especificação de como os diferentes módulos vão interagir.

A seguir, descreveremos brevemente, como os programas paralelos são desenvolvidos usando o sistema Helios. Maiores detalhes podem ser encontrados em [FRA2 91] e [FRA4 92].

Inicialmente, dado um certo problema, deve-se especificar como este problema pode ser particionado e associado a diferentes tarefas. Uma vez que o *transputer* suporta um esquema de *message-passing*, estas tarefas não compartilham estruturas de dados. Dois diferentes processadores compartilham da mesma estrutura de dados, somente através da troca de mensagens (*packets* de dados) através dos *links* de comunicação. Nesta fase, deve-se identificar as diferentes e independentes tarefas e como elas se comunicam entre si. O resultado desta fase é um **grafo paralelo**.

Em seguida, usa-se a linguagem CDL (Component Description Language) para descrever o grafo paralelo. Nesta linguagem, um nome lógico para cada nó (componente) e para cada *link* do grafo deve ser especificado. Para cada componente é associado um programa executável que define a tarefa correspondente, assim como que portas de comunicação (*streams*) são associadas com cada *link* lógico. Estes *streams* corresponderão a *pipes* que

podem ser acessado dos programas C.

Finalmente, o código C (ANSI) correspondente a cada componente deve ser desenvolvido. Como já mencionado, a comunicação entre tarefas é executada através de *pipes*, uma construção similar a arquivos. Portanto, de forma a enviar dados e receber dados, de uma tarefa para outra, os comandos *write* e *read* podem ser usados, respectivamente.

A especificação da rede física, i.é., o número de *transputers* a ser usado e o modo que eles são físicamente ligados, é definida por um mapa de recursos. Um mapa de recursos é um arquivo bem definido descrevendo como os 4 canais bi-direcionais do *transputer* são conectados na rede. Este mapa é armazenado em tempo de *login* e não pode ser trocado dinamicamente. Portanto, é muito importante definir a rede física adequada ao problema a ser resolvido. Nesta implementação, por exemplo, nós usamos uma rede de duas dimensões com 3xN *transputers*, onde N é a dimensão (número de variáveis) do problema.

2.5.1 Detalhes da Implementação

Nesta seção, a implementação paralela da rede neural, considerando um problema de tamanho $n + m = 3$ é descrita. Embora o *transputer* implemente uma máquina paralela *coarse-grained*, nós optamos por implementar uma configuração onde cada *transputer* simula um neurônio da rede neural. Esta configuração foi escolhida de forma a tornar a simulação mais próxima possível de uma possível implementação em *hardware* da rede.

Uma vez que a rede neural tem três camadas com $n + m$ nós em cada camada, esta configuração requer para a simulação, $3(n + m)$ *transputers* e mais um para realizar a comunicação com o *host*. Este último pode ser alocado para coordenar as tarefas desempenhadas pelos *transputers* restantes.

Figura 2.8 mostra a configuração correspondente ao problema de otimização de tamanho 3. Como podemos observar, decidimos mapear cada neurônio a cada *transputer*. Esta configuração é muito próxima do diagrama que representa a topologia dessa rede. A única diferença é que foi introduzido um nó denominado mestre, que coordena a rede. O mestre também é responsável por realimentar a rede enviando as saídas como entradas à rede. Nesta figura, os programas correspondentes aos nós nas camadas de entrada, intermediária e saída são chamados **Node1, Node2, Node3**, respectivamente. O programa correspondente ao coordenador da rede é chamado **Mestre**. Todo nó numa mesma camada executa o mesmo programa, pois eles realizam exatamente a mesma função.

O mapeamento desse grafo para um código CDL é praticamente direto. Vamos definir os seguintes nomes lógicos para os correspondentes componentes (nós): *mst*, para o mes-

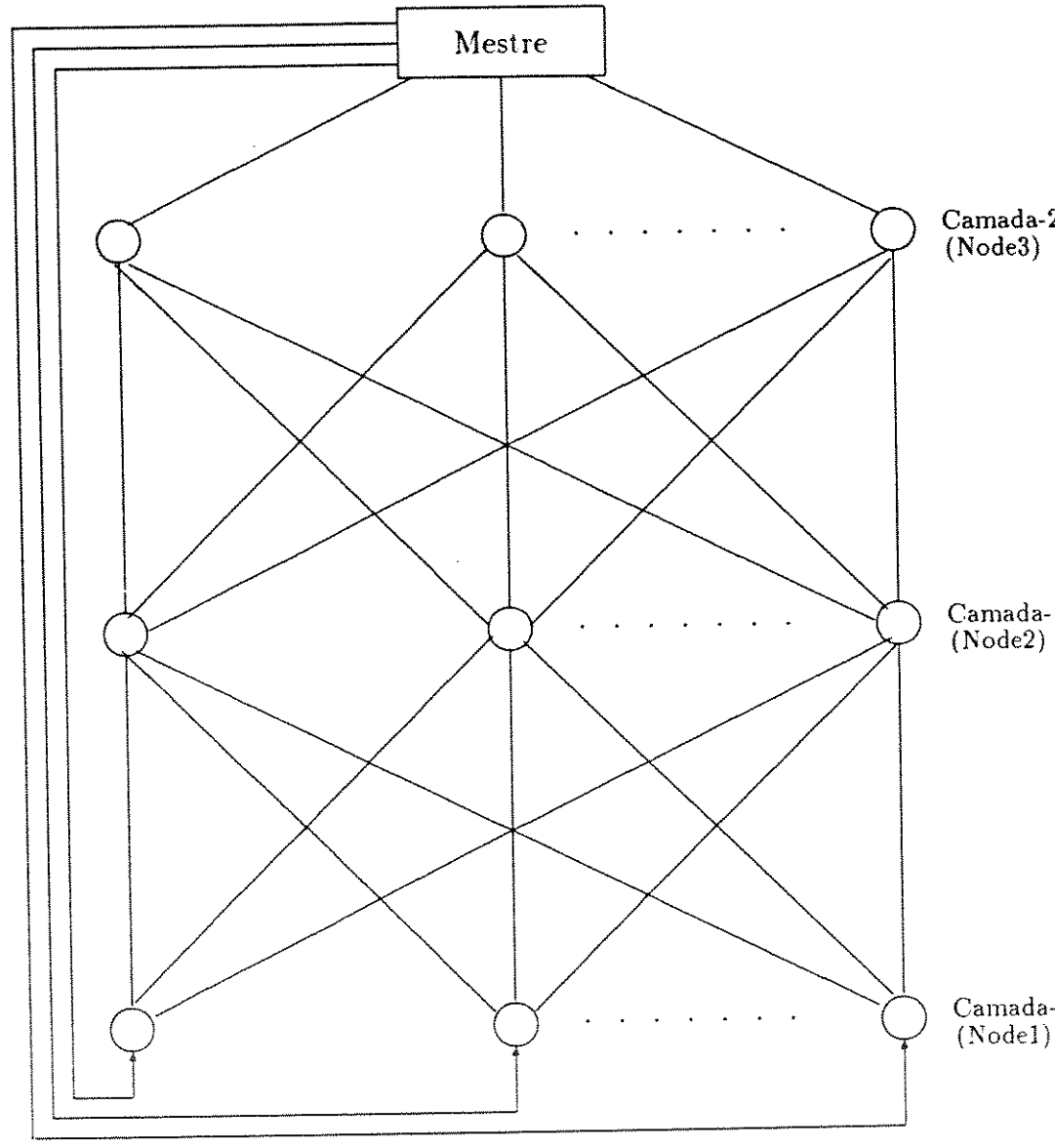


Figura 2.8: Configuração da Rede

tre, *inputlayer[i], hiddenlayer[i]* e *outputlayer[i]* como um *array* de componentes para as camadas de entrada, intermediária e de saída, respectivamente. Os nomes para os *links* lógicos são definidos como *arrays* do seguinte modo:

mn1: é um *link* do mestre para um nó na camada de entrada;

mn3: é um *link* do mestre para um nó na camada de saída;

n3m: é um *link* de um nó na camada de saída para o mestre;

n1n2: é um *link* de um nó na camada de entrada para um nó na camada intermediária;

n2n3: é um *link* de um nó na camada intermediária para um nó na camada de saída;

n3n2: é um *link* de um nó na camada de saída para um nó na camada intermediária;

n2n1: é um *link* de um nó na camada intermediária para um nó na camada de entrada.

Deve-se observar que estes *links* lógicos estabelecem comunicação em apenas uma direção. Portanto, para proporcionar o fluxo de informações em ambas as direções (para os passos *forward* e *backward*), dois *links* devem ser definidos entre dois nós. Na CDL, '<|' significa *reads from the stream whose name follows on this communication port*, enquanto '>|' significa *writes to the stream whose name follows on this communication port*. A seguir apresentamos o código CDL para este problema. Ele compreende as declarações das componentes. A última linha do código, simplesmente significa executar todas as seguintes componentes em paralelo. Isto é especificado pelo símbolo ^^.

```
component mst {
    code mestre;
    streams ,,,,
        <| n3m.0, >| mn3.0,
        <| n3m.1, >| mn3.1,
        <| n3m.2, >| mn3.2,
                , >| mn1.0,
                , >| mn1.1,
                , >| mn1.2;
    }
component hiddenlayer[i] {
    code node2;
    streams ,,,,
        <| n1n2{i}, >| n2n1{i},
```

```
component inputlayer[i] {
   code node1;
    streams <| mn1{i},,,,
       <| n2n1{3*i}, >| n1n2{3*i},
       <| n2n1{3*i+1}, >| n1n2{3*i+1},
       <| n2n1{3*i+2}, >| n1n2{3*i+2};
       }


 component outputlayer[i] {
   code node3;
   streams <| mn3{i}, >| n3m{i},,,
       <| n2n3{i}, >| n3n2{i},
```

```
        <| n1n2{i+3}, >| n2n1{i+3},          <| n2n3{i+3}, >| n3n2{i+3},
        <| n1n2{i+6}, >| n2n1{i+6},          <| n2n3{i+6}, >| n3n2{i+6};
        <| n3n2{3*i}, >| n2n3{3*i},       }
        <| n3n2{3*i+1}, >| n2n3{3*i+1},
        <| n3n2{3*i+2}, >| n2n3{3*i+2};
    }

mst ^^
outputlayer{0} ^^ outputlayer{1} ^^ outputlayer{2} ^^
inputlayer{0} ^^ inputlayer{1} ^^ inputlayer{2} ^^
hiddenlayer{0} ^^ hiddenlayer{1} ^^ hiddenlayer{2}
```

De acordo com a descrição na seção anterior, para implementar o simulador paralelo precisamos definir exatamente quais são as funções de cada nó na rede de *transputers*. Isto é, temos que definir precisamente quais são os programas *mestre*, *node1*, *node2* and *node3* declarados no código CDL. Estas funções, descritas a seguir, foram programadas na linguagem C.

Mestre

O mestre é responsável pela inicialização da rede (enviando uma identificação *id* para os nós da camada de entrada, que propagará este *setup* para as próximas camadas). Ele é também responsável pela sincronização do simulador, e pela realização das seguintes tarefas, a cada iteração:

- envia dados iniciais para a primeira camada para inicializar o passo *forward*, assim como, um *flag* indicando se a rede deve trabalhar com o subproblema primal ou com o subproblema dual. Para a primeira iteração, este *flag* é colocado como PRIMAL
- aguarda os dados de saída da camada de saída.
- calcula δ_o.
- envia δ_o a camada de saída para inicializar o passo *backward*.
- calcula os erros. Dependendo dos limites estabelecidos para os erros e dos valores calculados, o mestre decide se a próxima iteração corresponderá a um subproblema primal, a um subproblema dual, ou se o problema já está resolvido.

Camada de Entrada

Cada nó da camada de entrada executa as seguintes funções, a cada iteração:

- recebe dados de entrada do mestre,
- executa o passo *forward* local,
- envia a saída para a camada intermediária,
- recebe informação δ_h de cada nó na camada intermediária,
- atualiza pesos armazenados localmente (cada nó nesta camada armazena uma linha w_{ih})

Camada Intermediária

Cada nó na camada intermediária executa as seguintes funções, a cada iteração:

- recebe dados da camada de entrada,
- executa o passo *forward* local,
- envia a saída para a camada de saída,
- recebe informação δ_o de cada nó na camada de saída,
- atualiza pesos locais (cada nó nessa camada armazena uma linha da matriz w_{ho}),
- calcula δ_h,
- envia δ_h para a primeira camada

Camada de Saída

Cada nó na camada de saída executa as seguintes funções, a cada iteração:

- recebe dados da camada intermediária,
- executa o passo *forward* local,
- envia saída para o mestre,
- recebe informação δ_o do mestre,
- envia δ_o para a camada intermediária,

2.5.2 Descrição dos Procedimentos

Nesta seção, cada um dos módulos acima é detalhado usando pseudo-código.

Início do procedimento Mestre /* setup */

leia <xi>

envia *id* para a camada de entrada

flag = PRIMAL

/* ativação da rede */

faça {

envia <flag, xi> para a primeira camada

recebe <xo,yo> da camada de saída

calcula δ_o

envia δ_o para a camada de saída

Se flag == PRIMAL {

calcule ϵ_l

Se ϵ_l < limite erro local {

calcule ϵ_g

Se ϵ_g < limite erro global

flag = END

senão

flag = DUAL

}

senão

<xi> ← <yo>

}

```
            senão {
                <xi> ← <yo>
                flag = PRIMAL
            }
        } enquanto (flag != END)
        envia <flag; xi> para a primeira camada
Fim do procedimento Master


Início do procedimento Node1                    /* camada de entrada */
        /* setup */
        recebe id do mestre
        envia id para nós na camada intermediária
        inicializa pesos locais aleatoriamente
        yi = 0
        /* ativação da rede */
        faça {
            recebe <flag; xi> do mestre
            Se (flag && id) {
                /* nó é ativado */
                yi' ← g(xi)
                Para cada nó na cam. intermediária {
                    yi ← wih * yi'
                    envia <flag; yi> p/ a cam. intermediária
```

```
                }
                Para cada nó na cam. intermediária
                        recebe <δh >
                atualiza wih
        }
        senão {
                yi ← wih * yi'
                envia <flag; yi> para a camada intermediária
        }
    } enquanto (flag != FIM)
    envia <flag; yi> para a camada intermediária
Fim do procedimento Node1


Início do procedimento Node2                    /* camada intermediária */
    /* setup */
    recebe id camada de entrada
    envia id para nós na camada de saída
    yh = 0
    /* ativação da rede */
     faça {
            xh = 0
            Para cada nó na camada de entrada
                    recebe <flag; xh'>
```

Se (flag && *id*) {

xh ← sum xh'

yh' ← g(xh)

Para cada nó na camada de saída

yh ← who * yh'

envia <flag; yh> para a camada de saída

Para cada nó da camada de saída

recebe < δ_o >

atualiza pesos who

calcula <δ_h>

Para cada nó na primeira camada

envia <δ_h>

}

senão {

yh ← who * yh'

envia <flag; yh>

}

} **enquanto** (flag != FIM)

envia <flag; yh> para a camada de saída

Fim do procedimento Node2

Início do procedimento Node3 /* camada de saída */

/* setup */

recebe *id* da camada intermediária

/* ativação da rede */

faça {

xo = 0

recebe <flag; xo'> da camada intermediária

Se (flag && *id*) {

xo ← sum xo'

yo ← g(xo)

envia <flag; yo> para o mestre

recebe δ_o do mestre

envia δ_o para a camada intermediária

}

} **enquanto** (flag != END)

Fim do procedimento Node3

2.5.3 Resultados da Implementação

A implementação paralela foi executada no sistema de *transputers* descrito acima e comparada com uma implementação sequencial que utilizou também um processador T800 e linguagem C. Utilizamos os mesmos exemplos anteriores para analisar o comportamento da rede neural proposta. As soluções aproximadas obtidas são mostradas abaixo, considerando diferentes valores de x^0 e $\eta = 0.5, \epsilon_l = 0.001$.

Exemplo 1:

Considere o problema:

$$\min_x (x_1 - 0.1)^2 + (x_2 - 0.2)^2 + (x_3 - 0.3)^2 + (x_4 - 0.4)^2$$

Para $x^{0T} = (0.4, 0.2, 0.1, 0.2)$

	x^T	Iterações	Tempo(s)
Paralela	(.099507, .199954, .299960, .400010)	258	13.520
Sequencial	(.100422, .199810, .300175, .400019)	443	6.190

Para $x^{0T} = (-0.1, 0.0, 0.1, 0.6)$

	x^T	Iterações	Tempo(s)
Paralela	(.099508, .199954, .299960, .400010)	258	13.150
Sequencial	(.100422, .199810, .300175, .400019)	443	6.360

Para $x^{0T} = (0.2, 0.15, 0.45, 0.6)$

	x^T	Iterações	Tempo(s)
Paralela	(.099508, .199954, .299960, .400010)	258	13.280
Sequencial	(.100224, .199649, .300256, .399926)	405	.190

A solução ótima teórica é $x^* = (0.1, 0.2, 0.3, 0.4)^T$.

Exemplo 2:

Seja dado o problema:

$$\min_x (x_1 - 0.5)^4 + (x_1 - 2x_2)^2$$

Para x_0^T = (0.1, 0.2)

	x^T	iterações	Tempo(s)
Paralela	(.557852, .278885)	172	5.320
Sequencial	(.516331, .258273)	107	1.290

Para x_0^T = (-0.1, 0.1)

	x^T	iterações	Tempo(s)
Paralela	(.557824, .278870)	172	5.330
Sequencial	(.516265, .258239)	106	1.110

Para x_0^T = (0.1, -0.1)

	x^T	iterações	Tempo(s)
Paralela	(.557819, .278868)	172	5.250
Sequencial	(.516243, .258228)	106	1.080

A solução ótima teórica para este exemplo é:

$$x^* = (0.5, 0.25).^T$$

Exemplo 3:

$$\min_x (x_1 - 0.5)^2 + (x_2 - 0.4)^2$$

$$\text{sujeito a} \quad x_1 + x_2 = 0.8$$

As soluções aproximadas (x, λ) obtidas pela rede proposta são mostradas a seguir, considerando diferentes valores iniciais (x^0, λ^0), e fazendo $\eta = 0.5, \epsilon_l = 0.01, \epsilon_g = 0.001$.

Para $(x^0, \lambda^0)^T = (0.1, 0.2, 0.3)$

	(x, λ)	Iterações	Tempo(s)
Paralela	(.449505, .349499, .100923)	2045	62.760
Sequencial	(.450518, .350480, .099054)	6826	73.050

Para $(x^0, \lambda^0)^T = (-0.1, 0.0, 0.2)$

	(x, λ)	Iterações	Tempo(s)
Paralela	(.449504, .349499, .100916)	2027	61.920
Sequencial	(.450519, .350480, .099054)	7800	83.530

Para $(x^0, \lambda^0)^T = (0.2, 0.3, 0.6)$

	(x, λ)	Iterações	Tempo(s)
Paralela	(.449513, .349490, .100984)	4105	126.520
Sequencial	(.450518, .350480, .099054)	7071	75.210

Neste caso, a solução ótima é

$$x^* = (0.45, 0.35)^T, \quad \lambda^* = 0.1.$$

Exemplo 4:

Considere o problema não-linear :

$$\min_x -x_1 x_2$$

$$\text{sujeito a} \quad (x_1 - 0.3)^2 + x_2^2 = 0.05$$

Neste problema, as soluções aproximadas (x, λ) obtidas pela rede proposta são mostradas abaixo, considerando diferentes valores iniciais (x^0, λ^0), e fazendo $\eta = 0.5, \epsilon_l = 0.1$ e $\epsilon_g = 0.01$.

Para $(x^0, \lambda^0)^T = (0.4, 0.35, 0.6)$

	(x, λ)	Iterações	Tempo(s)
Paralela	(.413733, .216939, .954206)	6011	183.440
Sequencial	(.433633, .187869, .999892)	379	3.830

Para $(x^0, \lambda^0)^T = (0.2, 0.1, 0.4)$

	(x, λ)	Iterações	Tempo(s)
Paralela	(.413736, .216942, .954199)	6243	190.740
Sequencial	(***)		

Para $(x^0, \lambda^0)^T = (0.1, 0.1, 0.5)$

	(x, λ)	Iterações	Tempo(s)
Paralela	(.413736, .216942, .954199)	6598	201.130
Sequencial	(***)		

OBS: (***) indica: solução não obtida.

Observe que os valores obtidos estão bem próximos da solução ótima, dada por:

$$x^* = (0.4, 0.2)^T, \quad \lambda^* = 1.0.$$

2.6 Trabalhos Relacionados

É comum resolver um problema de otimização restrito transformando-o num problema de otimização irrestrito, no qual as restrições são incorporadas à função objetivo como termos de penalidades [LUEN 73]. Cada restrição $r_i(x)$ corresponde a um termo da forma $c_i r_i^2(x)$ onde c_i é uma constante.

Hopfield [HOPF 84] propôs uma implementação de rede neural para problemas irrestritos, onde as variáveis dinâmicas são identificadas com os neurônios, os pesos nas interconexões entre os neurônios são baseados na função objetivo e um procedimento do tipo gradiente especifica a regra de atualização para cada neurônio. Restrições são incorporadas como termos de penalidades à função objetivo [HOPF 85] e além disso, a rede realiza um mapeamento *sigmoidal* não-linear entre o estado interno de um neurônio e sua saída. As equações de movimento das redes de Hopfield são:

$$\frac{d\mu_i}{dt} = -\frac{\mu_i}{\tau} - \frac{\partial E}{\partial V_i}$$

onde E é a função objetivo, também chamada, função energia, V_i é a saída do i-ésimo neurônio e μ_i é o estado do i-ésimo neurônio.

A vantagem de aproximar um problema restrito por um irrestrito é que problemas irrestritos podem ser resolvido por métodos do tipo gradiente e estes, em geral, levam a implementações de redes [GIND 91]. Entretanto, algumas vezes, existe a necessidade de que as restrições sejam satisfeitas rigorosamente. Em tais casos, o método mais utilizado é o metodo dos multiplicadores de Lagrange [LUEN 73].

Na seção 2.3 propusemos uma rede neural de três camadas para resolver problemas restritos, cujo algoritmo de aprendizado é baseado nos multiplicadores de Lagrange.

Recentemente, Platt e Barr [PLAT 87] propuseram o método do multiplicador diferencial básico (BDMM), para resolver problemas restritos. O BDMM também se fundamenta nos multiplicadores de Lagrange. O algoritmo neural é baseado nas equações diferenciais apresentadas por Arrow [ARRO 58], cuja solução consiste, como no caso das redes de Hopfield, na minimização de uma determinada função energia. O BDMM será descrito, brevemente, a seguir.

- **BDMM**

Consideremos o problema primal (P) e a função Lagrangeana associada à (P) dados por (2.13) e (2.14), respectivamente. A função Lagrangeana, também chamada de função energia, pode ser escrita na forma:

$$L(x,\lambda) = f(x) + \sum_j \lambda_j r_j(x).$$

As condições necessárias para a solução do problema original (P) são dadas pelas seguintes equações diferenciais:

$$\nabla_x L(x_1, x_2, \ldots) = 0$$

$$r_j(x) = 0$$

Na implementação proposta por Platt e Barr [PLAT 87], estas equações correspondem a uma rede neural onde alguns neurônios (nós), denotados por V_i, representam as variáveis dinâmicas e outros, denotados por λ_i, representam os multiplicadores de Lagrange. As equações de movimento são:

$$\frac{dV_i}{dt} = -\frac{\partial f}{\partial V_i} - \sum_j \lambda_j \frac{\partial r_j}{\partial V_i}$$

$$\frac{d\lambda_i}{dt} = r_i(V_1, V_2, \ldots)$$

Platt [PLAT 87] demonstrou que BDMM sempre converge para o caso especial de problemas restritos: problemas quadráticos. Além disso, bons resultados foram obtidos usando o BDMM para resolver o PCV e o problema *analog decoding* [PLAT 87].

A rede proposta por Platt e Barr foi simulada e testada, resolvendo o sistema de equações diferenciais correspondentes aos Exemplos 3 e 4, utilizando uma rotina fornecida pela *NAG Fortran Library Routine* que implementa o método de Euler. Tanto a solução ótima como os respectivos tempos de execução, para cada solução inicial dada, foram obtidos. Os resultados estão contidos nas Tabelas 2.10 e 2.11. Estas tabelas apresentam os tempos de execução necessários para a convêrgencia da rede de três camadas proposta por nós, considerando diferentes soluções iniciais para x^0, $\epsilon_{local} = 0.01$ e $\epsilon_{global} = 0.001$, quando os pesos iniciais são gerados aleatoriamente. Esses resultados mostram o desempenho da rede proposta relativo à rede proposta por Platt e Barr. Os testes foram realizados utilizando-se um computador IBM-4341 e a linguagem FORTRAN.

- Para o Exemplo 3

$(x^0, \lambda^0)^T$	Platt e Barr $(x, \lambda)^T$ / tempo	Pesos Aleatórios x^T / tempo
(.1,.2,.3)	(.45062,.34937,.10000) $t = 0.025$s	(.453295,.356556,.096345) $t = 2.158$s
(.2,.3,.6)	(.44961,.35039,.10000) $t = 0.025$s	(.453453,.356467,.096416) $t = 1.462$s
(-.1,.0,.2)	(.44973,.35027,.10000) $t = 0.025$s	(.448488,.355195,.098423) $t = 1.035$s

Tabela 2.10

- Para o Exemplo 4

$(x^0, \lambda^0)^T$	Platt e Barr $(x, \lambda)^T$ / tempo	Pesos Aleatórios x^T / tempo
(.4..35..6)	(.39491,.20388,1.00166) $t = 0.0281$s	(.3939..1927,1.000) $t = 0.745$s
(.2..1..4)	(.39255,.19864,1.01178) t =0.0283s	(.4066..2034,1.000) $t = 0.763$s
(.1..1..5)	(.39526..19944,1.00691) t =0.034s	(.3983..1975..9998) $t = 0.967$s

Tabela 2.11

2.7 Análise do Comportamento da Rede

Sobre o desempenho da rede neural proposta para resolver problemas irrestritos e restritos, temos que fazer as seguintes considerações:

- Influência do ponto inicial x_0

A influência do ponto inicial x_0 na qualidade da solução é análoga aos resultados conhecidos da programação não-linear. Verificamos que para problemas convexos, a solução independe do ponto inicial, como nos Exemplos 1 e 2. Entretanto, isto não é válido para problemas não convexos, como o Exemplo 4.

- Tempo de CPU

O tempo de CPU relativo a implementação sequencial foi surpreendentemente baixo, diante do esperado, uma vez que a rede implementa um esquema do tipo BP e este, em geral, consome horas para convergir. Com relação a métodos tradicionais da programação não-linear (ver Tabelas 2.3, 2.5, 2.7 e 2.9) era de se esperar que a implementação sequencial fosse um pouco mais lenta que estes métodos, pois ela é uma simulação da rede proposta e nenhum conhecimento a respeito do problema é utilizado, tais como, se a função é ou não quadrática. Além disso, a rotina E04VDF da *NAG*, implementa um método de programação quadrática que, converge rapidamente para problemas quadráticos. Entretanto, os tempos de execução para alguns casos foram até menores, de acordo com Tabelas 2.3 e 2.5, com pesos iniciais fixados. Quanto aos testes realizados com a rede de Platt e Barr, os tempos foram bem maiores, mas vale ressaltar aqui, que esta rede foi simulada usando um dos mais simples métodos de solução numérica de equações diferenciais, método de Euler, que é um método de passo um e, portanto, consome menos tempos de CPU.

- Capacidade de Aprendizado

A rede possui capacidade de aprendizado como pode ser observado nas Tabelas 2.3, 2.5, 2.7, onde os tempos de execução são bem menores quando a rede foi inicializada com pesos fixados e não gerados aleatoriamente. Os pesos foram fixados de acordo com os valores obtidos para os mesmos quando a rede convergiu para uma dada condição inicial x_0. Em seguida, a rede foi utilizada para resolver o problema dado, para outras condições iniciais.

- Implementação Paralela

Pode-se observar que para problemas irrestritos, não existe melhoria a nível da simulação, o que pode ser observado no Exemplo 1 da seção 2.5. Isto se deve ao fato que a comunicação entre os *transputers* (nós) da rede, realizada através dos seus 4 canais bidirecionais, é uma operação que consome tempo. Entretanto, para problemas restritos, a implementação paralela forneceu resultados melhores que a implementação sequencial. Além da melhoria no desempenho, existem alguns casos onde a implementação sequencial converge lentamente para a solução, enquanto que no simulador paralelo a convergência é rápida e consistente. O simulador paralelo fornece uma implementação intrínseca da rede, isto é, uma implementação mais próxima possível de uma possível implementação de *hardware*, uma vez que ele reflete a estrutura da rede neural proposta. Contudo, a implementação paralela, tal como ela foi feita (os problemas podem ter no máximo tamanho

10, uma vez que o sistema conta com 32 *transputers*), é válida somente para problemas relativamente pequenos, uma vez que o número de processadores requeridos é igual ao número de nós na rede neural.

2.8 Conclusão

Neste capítulo, uma rede neural artificial com realimentação da saída, constituída de três camadas foi proposta para resolver problemas de otimização irrestritos. Esta abordagem foi estendida com base nos resultados da teoria de otimização, para resolver também problemas de otimização restritos. Análise comparativa do comportamento da rede proposta relativa a métodos convencionais de otimização e outras abordagens foram também apresentados. Uma implementação paralela que reflete a estrutura da rede foi desenvolvida. A solução de problemas mais complexos usando a abordagem proposta bem como a implementação da rede num *neurocomputer* são objetos de considerações futuras.

Agradecimentos

A implementação paralela foi desenvolvida usando o sistema *Parsytec SuperCluster* do *Systems Research Center* (SRC) na Universidade de Maryland em College Park-MD-USA. Agradecemos Dr. L. Kanal e Dr. B. Menezes pela oportunidade que nos foi dada para desenvolver esta implementação. Agradecemos também ao Conselho Nacional de Pesquisas - CNPq pelas bolsas concedidas pelo programa RHAE e através do processo número 300729/86-3.

Capítulo 3

Otimização de Sistemas: Métodos Multi-estágios

3.1 Introdução

Neste capítulo propomos uma rede neural para resolver problemas de programação dinâmica. Esta rede é constituída de neurônios do tipo Max e Min definidos a seguir. Mostramos que, com algumas modificações, esta rede pode resolver também problemas de decisão *fuzzy*. Em adição, problemas importantes de otimização são resolvidos pelas redes propostas para ilustrar os seus desempenhos.

3.2 Programação Dinâmica e Otimização

Embora a Programação Dinâmica seja uma abordagem poderosa na solução de problemas de otimização, sua utilização tem sido muito limitada por causa dos recursos computacionais (e.g. memória, tempo computacional) exigidos pelo algoritmo tradicional. Recentemente, novos procedimentos que reduzem os recursos computacionais necessários, têm sido desenvolvidos. Nesta seção, propomos uma RNA para resolver problemas de otimização discretos. Esta abordagem apresenta algumas vantagens com relação a outras, que são apontadas a seguir. Entretanto, antes de propormos a RNA, os conceitos básicos da programação dinâmica, bem como a sua utilização na solução de problemas de decisão *fuzzy*, são revistos.

3.2.1 Resolução de Problemas Determinísticos

Programação Dinâmica (PD) é um procedimento de otimização, que é particularmente aplicado a problemas que requerem uma sequência de decisões relacionadas entre si. Cada decisão transforma a situação atual numa nova situação. Num problema de otimização, uma sequência de decisões, que por sua vez produz uma sequência de situações que maximiza (ou minimiza) algum objetivo, é procurada.

Sabe-se que um problema de otimização consiste, dado um conjunto de várias alternativas, na seleção de uma, que satisfaz um conjunto de restrições e corresponde ao *valor ótimo* para uma função decisão (ou função objetivo) dada. Um problema desta natureza é resolvido formulando um modelo matemático associado e utilizando um método de solução adequado a este tipo de problema. As variáveis do modelo, que devem satisfazer algumas restrições, são determinadas simultaneamente.

Os conceitos básicos da PD são brevemente apresentados através de um simples exemplo, dado a seguir.

Exemplo :

Consideremos o problema de programação linear:

$$\begin{aligned} max \quad & f = 8x_1 + 10x_2 \\ \text{sujeito a} \quad & 4x_1 + 2x_2 \leq 12 \\ & x_1 , x_2 \geq 0 \end{aligned} \tag{3.1}$$

que pode ser interpretado por exemplo, como um problema de utilização ótima de capacidade. As variáveis x_1 e x_2 são quantidades produzidas por período de dois itens de produção, que requerem 4 e 2 horas-máquina por unidade. O lado direito das desigualdades representam o tempo máximo disponível pela máquina por período; os coeficientes na função decisão são unidades de lucro. Resolvendo esse problema pelo método simplex [BAZA 79] (ou geometricamente) obtém-se:

$$x_1 = 0;\ x_2 = 6;\ f = 60 \tag{3.2}$$

Uma abordagem alternativa para resolver o problema (3.1) é determinar uma variável por vez (sequencialmente), decompondo o problema em uma série de estágios, cada um correspondendo a um subproblema de uma única variável. Esta é a idéia fundamental da PD. Por exemplo, uma decomposição do problema 3.1 é ilustrada na Figura 3.1.

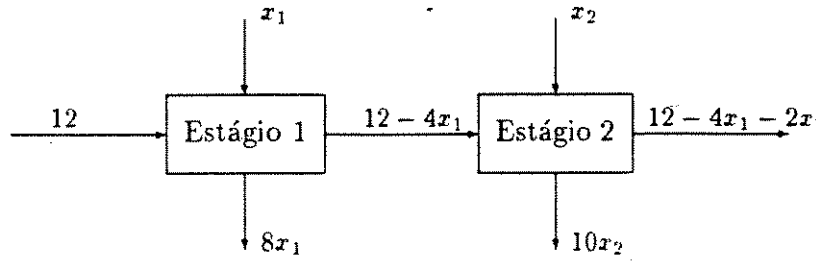


Figura 3.1: Diagrama de Fluxo

Vamos supor, por conveniência, que o item 1 é produzido primeiro que o item 2 (Estágio 1). Poderíamos ter começado pelo item 2, pois a ordem em que os itens são tomados não importa; ela é puramente formal pois a decomposição em estágios não reflete necessariamente uma sequência no tempo.

Para a produção do item 1, 4 horas-máquina são necessárias como mostrado na Figura 3.1. Se x_1 unidades são produzidas, $12 - 4x_1$ horas-máquina sobram como entrada para o segundo estágio. Depois de produzir x_2 unidades do item 2, obtém-se $12 - 4x_1 - 2x_2$ horas-máquina, correspondendo à variável de folga em (1) que representa capacidade não utilizada. Os dois estágios correspondem a um lucro total de $8x_1$ e $10x_2$, respectivamente.

Resolvendo o problema no sentido *backward*, x_1 é tratado como um parâmetro e o Estágio 2 é otimizado com respeito a variável x_2 (pode-se também resolver no sentido *forward*). Para $x_1 = x_1^*$ a capacidade máxima deixada para o Estágio 2 é $12 - 4x_1^*$ horas-máquina de modo que o subproblema parametrizado do Estágio 2 é dado por:

$$\max_{x_2} \quad f_1 = 10x_2$$

$$\text{sujeito a} \quad 2x_2 \leq 12 - 4x_1^*$$

$$x_2 \geq 0$$

que é um problema de programação linear análogo ao problema (3.1), com apenas uma única variável. A solução é dada por:

$$x_2 = 6 - 2x_1^*; \quad f_1 = 60 - 20x_1^* \tag{3.3}$$

onde x_1^* é um parâmetro.

A seguir, o Estágio 1 é otimizado com respeito à sua variável de decisão x_1. A capacidade disponível é 12 horas-máquina. A produção de x_1 unidades fornece uma contribuição de $8x_1$ para o lucro total, mas é necessário considerar que as horas-máquina usadas pelo Estágio 2 também afetam o lucro total, contribuindo com $f_1 = 60 - 20x_1$, que também depende de x_1. O problema de otimização correspondente ao Estágio 1 torna-se, portanto:

$$\max_{x_1} f_2 = 8x_1 + f_1 = 60 - 12x_1$$

$$\text{sujeito a} \quad 4x_1 \leq 12$$

$$x_1 \geq 0$$

que é, também, um problema linear. A solução, neste caso, é

$$x_1 = 0; \quad f_2 = 60$$

Tendo obtido o valor ótimo de x_1 e usando (3.3), obtemos

$$x_2 = 6.$$

Os valores de x_1 e x_2 obtidos coincidem com (3.2) e o lucro total $f = 60$ é igual a f_2. Assim, um problema de otimização com duas variáveis foi transformado numa série, ou numa sequência de problemas de uma única variável. Este é um exemplo de resolução via PD. Ele indica um procedimento geral para **decompor** um problema numa série de subproblemas envolvendo poucas variáveis, e combinar suas soluções para se obter a solução do problema original.

Quando um problema de otimização é formulado como um problema multi-estágio a fim de ser resolvido por PD, é conveniente introduzir variáveis de estado y_n associadas com os estágios individuais n, ($n = 1, 2, ..., N$). O processo de produção, que pode ser visto na Figura 3.1 tem seu início num dado estado inicial onde $y_0 = 12$ horas-máquina são disponíveis: este é o estado relativo ao Estágio 1. Então, o estado após o Estágio 1 - que é também a entrada do Estágio 2 - torna-se $y_1 = y_0 - 4x_1 = 12 - 4x_1$. Produzindo x_2 unidades do segundo produto, a capacidade disponível é reduzida a $y_2 = y_1 - 2x_2 = y_0 - 4x_1 - 2x_2$ que, neste caso, representa o estado final.

Assim, o estado que é uma entrada para o estágio n, y_{n-1}, é transformado num estado correspondente à saída, y_n, através da variável de decisão x_n. As mudanças sucessivas do estado do sistema podem, formalmente, ser descritas por *equações de transformação* da forma:

$$y_n = t_n(y_{n-1}, x_n) \qquad n = 1, 2, ..., N.$$

No exemplo, elas possuem a forma:

$$y_1 = y_0 - 4x_1$$

$$y_2 = y_1 - 2x_2$$

ao lado das restrições $x_1, x_2, y_1, y_2 \geq 0$, que são equivalentes às restrições do problema original 3.1: y_0 igual a 12 e y_2 representando a variável de folga.

As funções de retorno, i.é., as contribuições dos estágios individuais para a função decisão f, no caso geral, dependerão do estado correspondente à entrada e das variáveis de decisão:

$$r_n = r_n(y_{n-1}, x_n) \qquad n = 1, 2, ..., N.$$

No presente exemplo estas funções retorno são dadas por:

$$r_1 = 8x_1$$

$$r_2 = 10x_2.$$

Introduzindo estes símbolos na Figura 3.1, o diagrama de fluxo do problema de 2-estágios passa a ter a forma da Figura 3.2.

A solução pelo esquema *backward* procede como segue. Na primeira etapa dos cálculos - correspondente ao último estágio no sistema de produção. $n = N = 2$, y_1 é considerado como um parâmetro. herdado do estágio anterior do sistema. O estágio é otimizado maximizando sua função decisão, que é dada por:

$$\begin{aligned} \max_{x_2} \quad & f_1 = r_2(x_2) = 10x_2 \\ \text{sujeito a} \quad & 2x_2 \leq y_1 \\ & x_2 \geq 0 \end{aligned} \tag{3.4}$$

A solução é

$$x_2(y_1) = 0.5y_1$$

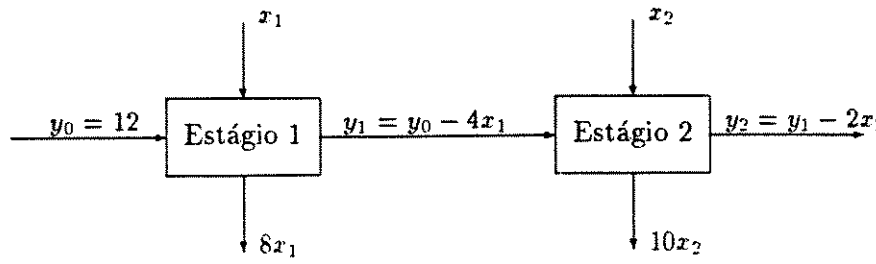


Figura 3.2: Diagrama de Fluxo

$$F_1(y_1) = 5y_1$$

onde F_1 denota o valor máximo da função decisão do estágio,

$$F_1 = \max_{x_2} f_1.$$

Na segunda etapa, a função decisão $f_2 = r_1(x_1 + F_1(y_1))$ será maximizada sujeita a $4x_1 \leq y_0$ (i.é., $y_1 = y_0 - 4x_1 \geq 0$); substituindo a função transformação do estágio $y_1 = y_0 - 4x_1$, f_2 torna-se uma função de x_1 e y_0. Então,

$$\max_{x_1} f_2 = r_1(x_1) + F_1(y_1) = 5y_0 - 12x_1$$

$$\text{sujeito a} \quad 4x_1 \leq y_0 \tag{3.5}$$

$$x_1 \geq 0$$

A solução é

$$x_1(y_0) = 0$$

$$F_2(y_0) = 5y_0$$

onde $F_2 = \max_{x_1} f_2$.

O problema de maximização foi decomposto em dois problemas de otimização de uma única variável, correspondendo a (3.4) e (3.5), respectivamente.

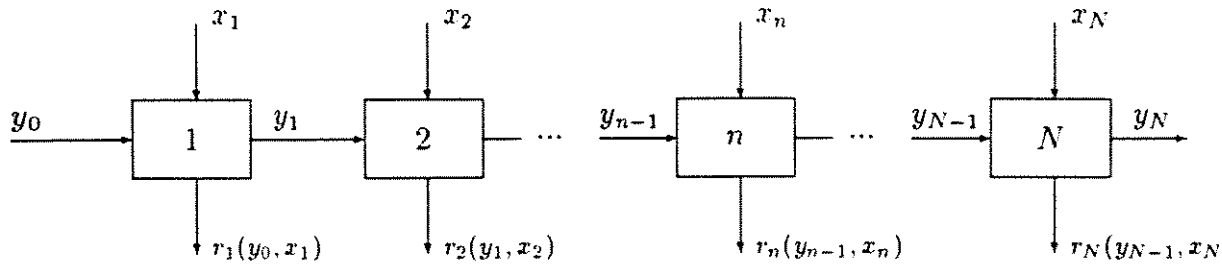


Figura 3.3: Diagrama de Fluxo

$$F_1(y_1) = \max_{x_2} 10x_2 \quad (0 \leq x_2 \leq 0.5y_1)$$

$$F_2(y_0) = \max_{x_1}(8x_1 + F_1(y_1)) \quad (0 \leq x_1 \leq 0.25y_0)$$

onde $max\ f = F_2(y_0)$. Numa formulação mais geral, a decomposição de um problema de dois estágios:

$$max\ f = \max_{x_1, x_2}[r_1(y_0, x_1) + r_2(y_1, x_2)]$$

pode ser expressa pelas **equações recursivas**:

$$F_1(y_1) = \max_{x_2}[r_2(y_1, x_2)]$$

$$F_2(y_0) = \max_{x_1}[r_1(y_0, x_1) + F_1(t_1(y_0, x_1))]$$

Este procedimento de resolver um problema de PD por recursão *backward* pode ser generalizado para qualquer número de variáveis. Um diagrama de fluxo para um sistema com N-estágios é mostrado na Figura 3.3.

As funções decisão de estágios $N, N-1, ..., 2, 1$ são respectivamente

$$f_1 = r_N(y_{N-1}, x_N)$$

$$f_2 = r_{N-1}(y_{N-2}, x_{N-1}) + F_1(y_{N-1}); \qquad y_{N-1} = t_{N-1}(y_{N-2}, x_{N-1})$$

$$\vdots$$

$$f_i = r_{N-(i-1)}(y_{N-i}, x_{N-(i-1)}) + F_{i-1}(y_{N-(i-1)}); \qquad y_{N-(i-1)} = t_{N-(i-1)}(y_{N-i}, x_{N-(i-1)}) \tag{3.6}$$

$$\vdots$$

$$f_N = r_1(y_0, x_1) + F_{N-1}(y_1)$$

onde $y_1 = t_1(y_0, x_1)$ e F_j é o máximo de $f_j, j = 1, 2, ..., N$.

Maximizando as funções decisão de cada estágio com respeito a sua variável de decisão, e tratando o estado de entrada como um parâmetro, obtemos as soluções paramétricas em cada estágio

$$x_n = x_n(y_{n-1}) \quad n = N, N-1, ..., 2, 1.$$

Caso Discreto

O domínio de problemas de PD pode ser contínuo, como foi considerado acima, ou discreto. A ênfase neste capítulo será dada a problemas de PD num domínio discreto. Isto é, as variáveis de decisão de um problema de PD podem receber somente valores discretos, ou as funções retorno e/ou transformação são dadas na forma de tabela de acordo com os valores discretos das variáveis.

Suponha que no Exemplo dado por (3.1), r_1 and r_2 sejam definidas somente para valores inteiros de x_1 e x_2 que satisfazem: $x_1 \leq 3$ e $x_2 \leq 6$:

x_1	0	1	2	3
$r_1(=8x_1)$	0	8	16	24

x_2	0	1	2	3	4	5	6
$r_2(=10x_2)$	0	10	20	30	40	50	60

As funções transformação na forma de tabela são como seguem:

	$y_1(=y_0-4x_1)$			
$y_0 \backslash x_1$	0	1	2	3
12	12	8	4	0

	$y_2(=y_1-2x_2)$						
$y_1 \backslash x_2$	0	1	2	3	4	5	6
0	0						
4	4	2	0				
8	8	6	4	2	0		
12	12	10	8	6	4	2	0

onde os lugares em branco na última tabela, correspondem a combinações de valores de y_1 e x_2 que não são factíveis porque eles implicariam num valor negativo para y_2 (de acordo com a restrição $y_2 = y_1 - 2x_2 \geq 0$).

O procedimento para a solução, usando recursão *backward*, tem a seguinte forma. Para o último estágio ($n = N = 2$) tem-se:

Estágio 2	$f_1 = r_2(x_2) = 10x_2$							$F_1(y_1)$	$x_2(y_1)$	$y_2(y_1)$
$y_1 \backslash x_2$	0	1	2	3	4	5	6			
0	**0**							0	0	0
4	0	10	**20**					20	2	0
8	0	10	20	30	**40**			40	4	0
12	0	10	20	30	40	50	**60**	60	*6*	*0*

onde o valor máximo da função decisão f_1 para cada um dos estados de entrada possíveis (valores de y_1) é mostrado em negrito.

Para o Estágio 1, temos a função decisão $f_2 = r_1(x_1) + F_1(y_1)$ onde através da função transformação obtém-se y_1 para cada valor de x_1; por exemplo, $x_1 = 2$ implica $y_1 = y_0 - 4x_1 = 4$, e a tabela anterior fornece $F_1(y_1) = 20$. Os cálculos desse estágio são apresentados na seguinte tabela:

Estágio 1	$f_2 = r_1(x_1) + F_1(y_1) = 8x_1 + F_1(y_1)$						
$y_0 \backslash x_1$	0	1	2	3	$F_2(y_0)$	$x_1(y_0)$	$y_1(y_0)$
12	**0 + 60**	8 + 40	16 + 20	24 + 0	60	0	12

A solução ótima pode ser encontrada analogamente, partindo-se da última tabela, onde $y_0 = 12$ fornece $x_1 = 0$ (a solução ótima do estágio) que leva a $y_1 = 12$ (pela equação transformação). Da primeira tabela, $y_1 = 12$ fornece o valor ótimo $x_2 = 6$ e o estado de saída $y_2 = 0$. Estes valores são indicados em itálico. O retorno máximo total é $F_2(y_0) = 60$.

3.2.2 Resolução de Problemas de Otimização *Fuzzy*

Problemas de tomada de decisão, particularmente para uma classe de problemas com incerteza não-estocástica, podem ser formulados como modelos *fuzzy* de tomada de decisão.

Não entraremos em detalhes quanto a teoria de conjuntos *fuzzy*, mas apresentaremos os conceitos e propriedades de conjuntos *fuzzy* que forem necessários para a compreensão da revisão brevemente feita aqui. Tanto as definições quanto as propriedades básicas dos conjuntos *fuzzy* podem ser encontradas, por exemplo, em [ZADE1 65,NEGO 85].

Bellman e Zadeh [BEL2 70] sugeriram uma abordagem diferente da otimização tradicional para resolver problemas que possam ser modelados como problemas de otimização *fuzzy*. Nesta abordagem, tanto as metas como as restrições são tratadas exatamente do mesmo modo. Ambas são definidas como conjuntos *fuzzy* no espaço de alternativas e deste modo, elas são tratadas identicamente na formulação de uma decisão.

Para um melhor compreensão deste fato, vamos supor que tenhamos n metas *fuzzy* $G_1, ..., G_n$ e m restrições $C_1, ..., C_m$. A decisão resultante é a intersecção das metas $G_1, ..., G_n$ e das restrições $C_1, ..., C_m$, dadas. Isto é,

$$D = G_1 \cap G_2 \cap ... \cap G_n \cap C_1 \cap C_2 \cap ... \cap C_m \tag{3.7}$$

e a medida de relacionamento μ_D correspondente é

$$\mu_D = \mu_{G_1} \wedge \mu_{G_2} \wedge ... \wedge \mu_{G_n} \wedge \mu_{C_1} \wedge \mu_{C_2} \wedge ... \wedge \mu_{C_m}$$

onde $\wedge$ simboliza o operador min.

A seguir algumas propriedades são apresentadas [BEL2 70].

Propridade 1:

Seja f um mapeamento de $X = \{x\}$ em $Y = \{y\}$ e $G_1, G_2, ..., G_n$ conjuntos *fuzzy* em Y. Dado um conjunto *fuzzy* G_i em Y, pode-se encontrar um conjunto *fuzzy* $\hat{G}_i$ em X que induz G_i em Y. A função relacionamento de $\hat{G}_i$ é dada pela igualdade

$$\mu_{\hat{G}_i} = \mu_{G_i}(f(x)), \quad i = 1, ..., n. \tag{3.8}$$

Propriedade 2:

Se γ é uma constante e f é qualquer função de x, então

$$\max_x(\gamma \wedge f(x)) \;=\; \gamma \;\wedge\; \max_x f(x). \tag{3.9}$$

Consideremos o problema de controle ótimo discreto descrito a seguir, como um exemplo de uma aplicação de problemas de otimização multi-estágio num ambiente *fuzzy*.

Assumiremos que o sistema sob controle é finito, invariante no tempo, e que sua dinâmica satisfaz:

$$x_{t+1} \;=\; f(x_t, u_t) \quad t \;=\; 0, 1, 2, ... \tag{3.10}$$

onde o estado, x_t, no tempo t, assume um dos valores do conjunto finito

$$X = \{\sigma_1, \sigma_2, ..., \sigma_n\};$$

a entrada, u_t, assume um dos valores do conjunto finito

$$U = \{\alpha_1, \alpha_2, ..., \alpha_m\}$$

e f : X x U $\mapsto$ X é uma função não-*fuzzy* e determinística.

Assumiremos, também, que no tempo t, a entrada está sujeita a uma restrição *fuzzy* C_t, que é um conjunto *fuzzy* definido em U e caracterizada pela função de pertinência $\mu_t(u_t)$. Além disso, assumiremos que a meta é um conjunto *fuzzy* G^N definido em X, que por sua vez é caracterizada pela função de pertinência $\mu_{G^N}(x^N)$, onde N denota o tempo no qual o processo termina.

A decisão

$$D \;:\; U \mathbf{x} U \mathbf{x} ... \mathbf{x} U \;\mapsto\; R$$

pode ser escrita da seguinte forma, utilizando (3.7),

$$D \;=\; C^0 \cap C^1 \cap ... \cap C^{N-1} \cap \hat{G}^N$$

onde $\hat{G}^N$ é um conjunto *fuzzy* em U x U x ... x U que induz G^N em X. Mais especificamente, em termos de funções de pertinência, a decisão pode ser dada por:

$$\mu_D(u^0, u^1, ..., u^{N-1}) \;=\; \mu_0(u^0) \wedge \mu_1(u^1) \wedge ... \wedge \mu_{G^N}(x^N)$$

onde x^N, por intermédio de (3.10), pode ser expresso como uma função de $x^0, u^0, ..., u^{N-1}$.

O objetivo principal é o de encontrar uma sequência de entradas $u^0, ..., u^{N-1}$ que

$$\max_{\mu_D}(u^0, u^1, ..., u^{N-1})$$

Para obtermos a solução,

$$u_M^0, u_M^1, ..., u_M^{N-1}$$

pode-se usar a abordagem da PD uma vez que o problema requer uma sequência de decisões interrelacionadas. Para tanto, adotando um esquema recursivo *backward*.

$$\mu_D(u_M^0, u_M^1, \ldots, u_M^{N-1}) =$$

$$= \max_{u^0,\ldots,u^{N-2}} \max_{u^{N-1}} \{\mu_0(u^0) \wedge \ldots \wedge \mu_{N-2}(u^{N-2}) \wedge \mu_{N-1}(u^{N-1}) \wedge \mu_{G^N}(f(x^{N-1}, u^{N-1}))\} \quad (3.11)$$

Esta expressão pode ser reescrita, usando (3.9), como

$$\mu_D(u_M^0, u_M^1, ..., u_M^{N-1}) = \max_{u^0,\ldots,u^{N-2}} \{\mu_0(u^0) \wedge ... \wedge \mu_{N-2}(u^{N-2})\} \wedge \mu_{G^{N-1}}(x^{N-1})) \quad (3.12)$$

onde

$$\mu_{G^{N-1}}(x^{N-1}) = \max_{u_{N-1}} \{\mu_{N-1}(u^{N-1}) \wedge \mu_{G^N}(f(x^{N-1}, u^{N-1}))\}$$

pode ser vista como a função de pertinência de uma meta *fuzzy* no tempo $t = N-1$ que é induzida pela meta dada G^N no tempo $t = N$.

Assim, as seguintes equações de recorrência, que na verdade são análogas as equações dadas em (3.6) e adaptadas para esse problema, fornecem a estratégia de solução desejada:

$$\mu_{G^{N-k}}(x^{N-k}) = \max_{u^{N-k}} \{\mu_{N-k}(u^{N-k}) \wedge \mu_{G^{N-k+1}}(x^{N-k+1})\} \quad (3.13)$$

onde

$$x^{N-k+1} = f(x^{N-k}, u^{N-k}) \; ; \; k = 1, \ldots, N.$$

3.3 Neurônios Max e Min

Neurônios artificiais são modelos de unidades não-lineares com um certo número de entradas e em geral, uma única saída. Estes neurônios possuem uma função não-linear

que transforma as entradas para fornecer uma saída. Exemplos de funções não-lineares, conhecidas também como funções de ativação, incluem a função *hard limiter* [WIDR 88], *threshold* [FUKU 88] e *sigmoid* [RUME 86]. A abordagem desenvolvida por [GOM1 92] difere do modelo não linear usual de neurônio. Ela propõe um modelo generalizado de neurônio, do qual os neurônios Max e Min são casos particulares. Os neurônios max e min são revisados a seguir, pois são essenciais na construção da rede neural DPNN proposta. Neurônios generalizados são introduzidos em [GOM1 92,GOM2 92,ROCH 92].

Geralmente, um neurônio artificial é um elemento computacional que:

- pondera suas n entradas a_i

$$v = \sum_{i=1}^{n} w_i a_i \tag{3.14}$$

de acordo com os pesos sinápticos correspondentes às conexões entre neurônios de entrada (pré-sinápticos) n_i e neurônios de saída (pós-sinápticos) n_k, e

- recodifica $v \in V$ na ativação axônica a_p do neurônio pós-sináptico:

$$a_p = \begin{cases} f(v) & \text{se } v \geq \alpha \\ 0 & \text{caso contrário} \end{cases}$$

onde α é um limiar axônico e f é a função de ativação (ou de transferência).

Se

$$f(v) = 1 \quad \text{para todo} \quad v \geq \alpha$$

o neurônio é um **neurônio binário**, ou um neurônio McCulloch e Pitts. Por outro lado,

$$f : V \mapsto [0, C]$$

onde C é uma constante. Dois limiares axônicos α_1 e α_2 podem também ser definidos, tais que:

$$a_p = \begin{cases} C & \text{se } v \geq \alpha_2 \\ f(v) & \text{se } \alpha_1 \leq v \leq \alpha_2 \\ 0 & \text{caso contrário} \end{cases} \tag{3.15}$$

Os valores de α_1 e α_2 em muitas aplicações são fornecidos por um tipo especial de neurônio, chamado *bias cell*.

Se $C = 1$ então o neurônio é chamado aqui de neurônio *fuzzy*. Outras definições de neurônio *fuzzy* são encontradas em [PEDR 91,GUPTA 92].

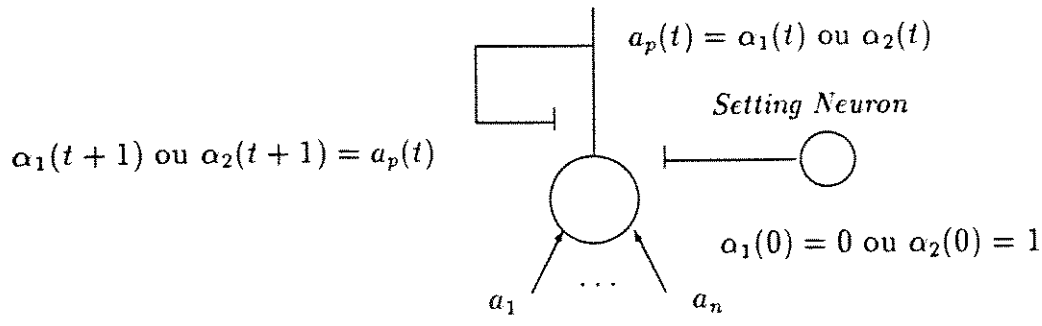


Figura 3.4: Neurônio Recorrente

Uma modificação do neurônio de McCulloch-Pitts relacionada com o controle do limiar axônico foi proposta por Gomide e Rocha [GOM1 92,GOM2 92] para definir um tipo especial de neurônio, chamado neurônio recorrente. A sinapse recorrente é estabelecida se o axônio do neurônio n_j faz contato com dendritos ou com o corpo celular do próprio n_j, Figura 3.4. Se a sinapse recorrente é localizada perto do axônio, então ele pode controlar **o limiar axônico** como uma função da própria atividade de n_j.

Consideremos os seguintes tipos de neurônio definidos em [GOM2 92,ROCH 92]:

- Neurônio de Alto Limiar (HTN): na polarização deste tipo de neurônio é colocada um valor alto, tal que sua saída é mantida igual a zero independente do valor do valor de suas entradas. Em determinados momentos, este limiar é abaixado sob o controle de outro neurônio, chamado aqui por *setting neuron*. Nestes momentos específicos, o valor atual de r em (3.14) é recalculado junto a a_i no axônio n_i e transmitido para a célula HTN pós-sináptica. Neste momento, o neurônio é dito disparar.
- Neurônio de Baixo Limiar (LTN): na polarização deste tipo de neurônio é colocado um valor baixo, tal que sua saída muda de acordo com a modificação temporal de suas entradas. Os neurônios comumente usados em RNA são neurônios LTN.

Vamos supor que exista uma ordenação na apresentação das atividades pré-sinápticas e que o neurônio pós-sináptico seja um neurônio LTN n_p. Isto implica que o neurônio pré-sináptico n_i sempre dispara antes do neurônio n_j se $i \leq j$. Esta ordenação é obtida definindo os neurônios pré-sinápticos como neurônios HTN. Então, a atividade a_i da célula pré-sináptica n_i é mantida igual a zero exceto em $\tau = i$ de seu disparo. Este momento é determinado controlando o *setting neuron*. A atividade a_p da célula pós-sináptica n_p

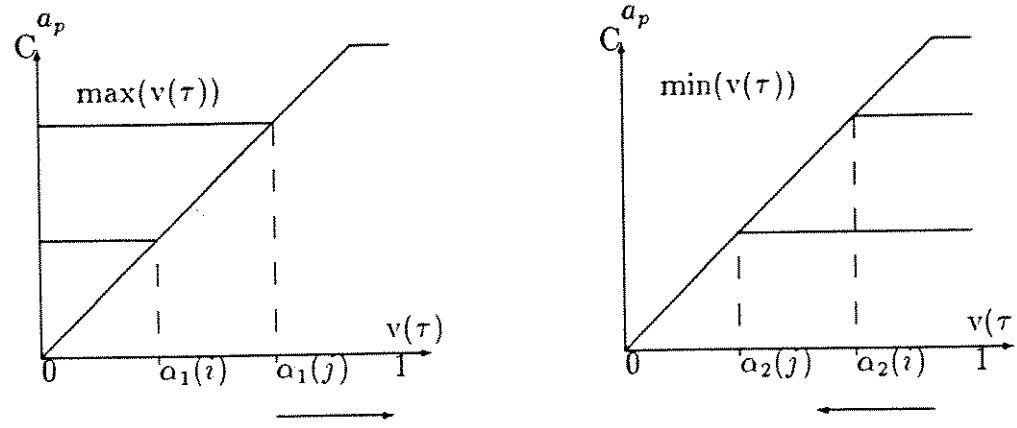


Figura 3.5: Processamento dos neurônios Max e Min

varia com τ. Isto é denotado, aqui, por $a_p(\tau)$. A partir destas condições podemos definir os neurônios Max e Min:

- **Neurônio Max** [Figura 3.5]: é definido se o limiar axônico $\alpha_1(\tau) = 0$ para $\tau = 0$ (pela ação do *setting neuron*), e em τ ele é feito igual ao nível de ativação $a_p(\tau - 1)$ em $\tau - 1$:

$$\alpha_1(\tau) = a_p(\tau - 1) \tag{3.16}$$

Além disso, a saída $a_p(\tau)$ é dada por:

$$a_p(\tau) = \begin{cases} \alpha_1(\tau) & \text{se } v(\tau) \leq \alpha_1(\tau) \\ v(\tau) & \text{caso contrário} \end{cases} \tag{3.17}$$

onde $v(\tau)$ é a ativação pós-sináptica em τ. Nesta condição, a saída $a_p(\tau)$ do neurônio em τ é dada por:

$$a_p(\tau) = \bigvee_{i=1}^{\tau} (w_i a_i)$$

Se $w_i = 1$ para todo i:

$$a_p(\tau) = \bigvee_{i=1}^{\tau} a_i \tag{3.18}$$

onde $\bigvee$ é o operador máximo.

- **Neurônio Min** [Figura 3.5] é definido se o limiar axônico $\alpha_2(\tau) = C$ para $\tau = 0$ (pela ação do *setting neuron*), e em τ é feito igual ao nível de ativação $a_p(\tau - 1)$ em $\tau - 1$:

$$\alpha_2(\tau) = a_p(\tau - 1). \tag{3.19}$$

A saída $a_p(\tau)$ é:

$$a_p(\tau) = \begin{cases} \alpha_2(\tau) & \text{se } v(\tau) \geq \alpha_2(\tau) \\ v(\tau) & \text{caso contrário} \end{cases} \tag{3.20}$$

onde $v(\tau)$ é a ativação pós-sináptica τ.

Nesta condição, a saída $a_p(\tau)$ do neurônio em cada τ é dada por:

$$a_p(\tau) = \bigwedge_{i=1}^{\tau}(w_i a_i)$$

Se $w_i = 1$ para todo i:

$$a_p(\tau) = \bigwedge_{i=1}^{\tau} a_i \tag{3.21}$$

onde $\wedge$ é o operador mínimo.

Instâncias particulares dos neurônios Max e Min, serão úteis na obtenção das redes neurais artificiais propostas nas seções que se seguem.

3.4 DPNN - Rede Neural para Otimização Discreta

3.4.1 Definição da DPNN

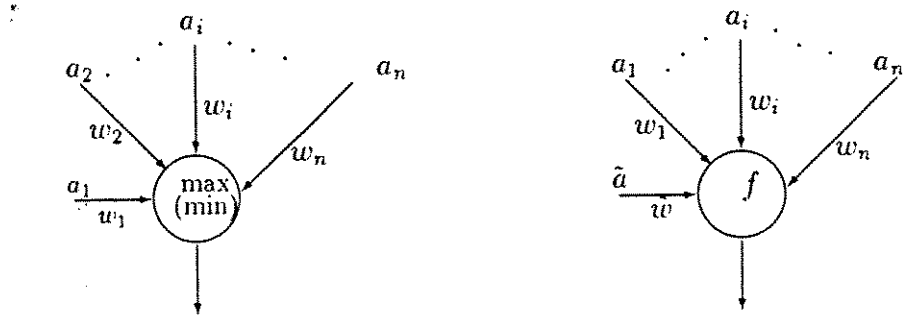
Para a apresentação da DPNN, primeiramente definiremos os neurônios que constituem a DPNN, explicando o significado do porquê da necessidade da existência dos mesmos, de suas entradas, funções e saída. Em seguida, definiremos também as interconexões entre os neurônios definidos.

Definição dos Neurônios da DPNN

A rede DPNN proposta nesta seção é constituída de dois tipos diferentes de neurônios: neurônio-Y e neurônio-X, que serão descritos a seguir.

Assumiremos, sem perda de generalidade, um esquema *backward*, para resolução de problemas de PD e que o problema de otimização a ser resolvido é um problema de maximização. Seja:

- $\mathcal{X}_{N-k}$ o conjunto de valores discretos que a variável de decisão x_{N-k} pode assumir no estágio $N-k$; $k = 0, 1, ..., N-1$;



neurônio-Y neurônio-X

Figura 3.6: Neurônios da DPNN

- $\mathcal{Y}_{N-k}$ o conjunto de valores discretos que a variável de estado y_{N-k-1} pode assumir no estágio $N-k; k=0,1,...,N-1$;

- $$(XY)^j_{N-k} = \{(x^i_{N-k}, y^j_{N-k-1}), x^i_{N-k} \in \mathcal{X}_{N-k}, y^j_{N-k-1} \in \mathcal{Y}_{N-k} \\ / \quad y_{N-k} = t_{N-k}(x^i_{N-k}, y^j_{N-k-1}) \text{ é factível}\} \qquad (3.22)$$

  onde t_{N-k} é a equação transformação dada por (3.6);

- n o número de elementos do conjunto $(XY)^j_k$.

Um **neurônio-Y** [Figura 3.6] é um neurônio Max ou Min, dependendo se o problema de otimização a ser resolvido é um problema de máximo ou de mínimo, respectivamente.

A saída do neurônio-Y, s, é dada pelo máximo ou mínimo de suas entradas ponderadas pelos pesos, i.é.,

$$s = max\{a_1w_1, a_2w_2, ..., a_nw_n\}$$

ou

$$s = min\{a_1w_1, a_2w_2, ..., a_nw_n\}.$$

onde os pesos w_i serão especificados mais adiante e as entradas têm o seguinte significado.

Denotando as n entradas no j-ésimo neurônio-Y, em uma dada camada k por:

$$a_1(Y^j), a_2(Y^j), ..., a_n(Y^j)$$

temos que:

- $a_1(.)$: é a saída do neurônio-X cuja entrada corresponde ao primeiro valor da variável x_{N-k} tal que o par $(x_k^1, y_k^j) \in (XY)_k^j$,

- $a_2(.)$: é a saída do neurônio-X cuja entrada corresponde ao segundo valor da variável x_{N-k} tal que o par $(x_k^2, y_k^j) \in (XY)_k^j$,

$$\vdots$$

- $a_n(.)$: é a saída do neurônio-X cuja entrada corresponde ao n-ésimo valor da variável x_{N-k} tal que o par $(x_k^n, y_k^j) \in (XY)_k^j$.

Um **neurônio-X** [Figura 3.6] é um neurônio que recebe pares de sinais de entrada $(\tilde{a}, a_i)$ com respectivos pesos $\tilde{w}$ e w_i, sincronizados pelo mesmo mecanismo adotado pelo neurônio Max (ou Min), para fornecer uma saída s. A saída s, do neurônio-X é dada por:

$$s = f(\tilde{w}\tilde{a} + w_i a_i)$$

onde

$$f(\theta) = \begin{cases} \theta & \text{se } \theta \geq \alpha, \\ 0 & \text{caso contrário} \end{cases} \tag{3.23}$$

onde α é um limiar do neurônio-X, $\tilde{a}$ um valor fixado para o neurônio-X e os a_i são valores recebidos de outros neurônios, através das conexões *feedback* (na DPNN estes neurônios são os neurônios-Y).

Na DPNN, as entradas do neurônio-X, ponderadas pelos pesos, têm o seguinte significado. Considerando o t-ésimo neurônio-X da camada k e o seu i-ésimo par de entrada, denotado por:

$$(\tilde{w}_k^t \tilde{a}_k^t, w_i a_i)$$

temos que:

- $\tilde{w}_k^t$: corresponde ao coeficiente da variável x_{N-k} na função de decisão r_{N-k} em (3.6);

- $\tilde{a}_k^t$: corresponde ao t^{th} valor da variável x_{N-k};

- a_i: corresponde a i^{th} entrada no t^{th} neurônio-X e w_i é o seu respectivo peso, especificado a seguir.

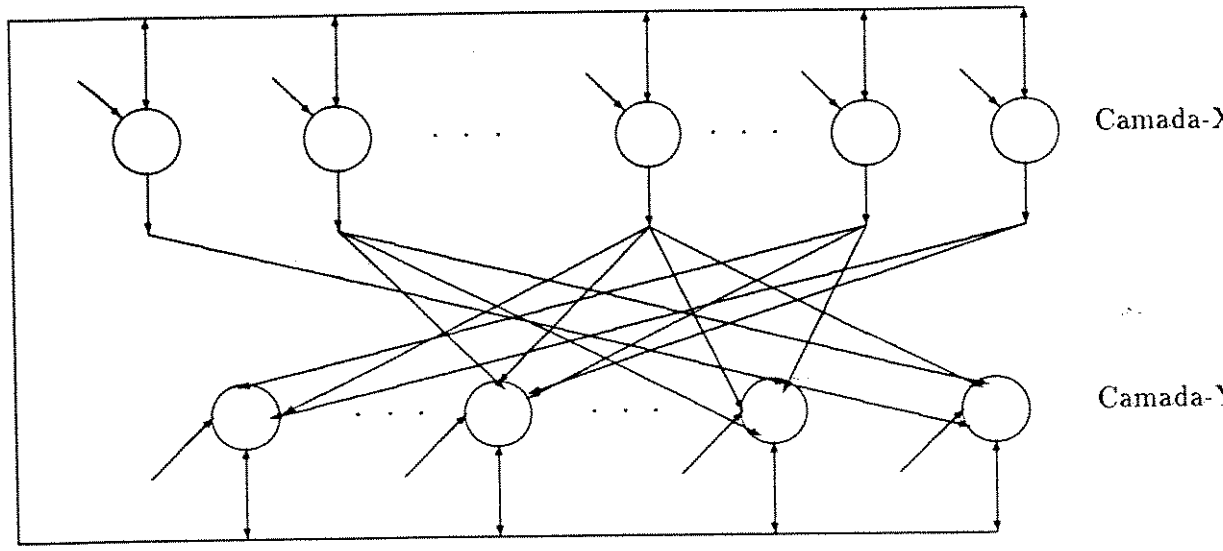


Figura 3.7: Rede Neural para Problemas de PD Discretos - DPNN

A DPNN é constituída de dois tipos diferentes de camadas: camada-Y, que é aquela composta somente de neurônios do tipo Y e camada-X, que é composta somente de neurônios do tipo X.

Na proposta inicial, as camadas da DPNN alternavam entre camadas-Y e camadas-X, sendo que a primeira camada deveria ser uma camada do tipo camada-X. Assim, a DPNN deveria possuir um total de $2*N$ camadas, onde N representa o número de estágios necessários para se obter a solução do problema em questão via PD. Os detalhes dessa proposta inicial estão contidos em [FRA5 92,FRA6 93].

Entretanto, como o número de camadas, neste caso, aumentava com o número de estágios, passamos a analisar a possibilidade da redução do número de camadas. Consequentemente, propõe-se aqui uma rede neural constituída de apenas duas camadas, com realimentação [Figura 3.7]. A primeira camada é uma camada do tipo X e a segunda camada é do tipo Y. As interconexões entre os neurônios dessas duas camadas são definidas a seguir.

Interconexões entre os neurônios

As interconexões (pesos) das camadas-X para camadas-Y, assim como, interconexões das camadas-Y para as camadas-X, são designadas pela própria rede.

Para definição das interconexões entre os neurônios das duas camadas, os seguintes conceitos, descritos e detalhados em [ROCH 92], são importantes.

Sabe-se que durante o processamento químico sináptico, a produção de um transmissor nos terminais axônicos é dependente de ambas as células: pré e pós-sinápticas. As proteínas produzidas pela célula pré-sináptica são chamadas precursores e aquelas liberadas pela pós-sináptica são chamadas controladores. A especificação genética do transmissor a ser produzido é dependente dos sinais fornecidos tanto pelas células pré-sinápticas quanto pelas células pós-sinápticas, e a ativação da leitura dos genes especificados é governada pelo campo produzido pela ativação dos receptores pós-sinápticos. O precursor produzido no corpo da célula pré-sináptica deve ser transportado para os terminais axônicos. O mesmo precursor pode ser usado para produzir diferentes tipos de transmissores, porque a especificação final de um transmissor é dependente dos controladores produzidos pela célula pós-sináptica.

Denotaremos por T um transmissor, R um receptor e C um controlador de um neurônio.

Sabe-se, também que, a ativação da célula pós-sináptica n_j devida ao transmissor T liberado pela célula pré-sináptica n_i ativa moléculas de controle C_j. Isto pode ser representado por

$$T_i \bigoplus R_j >> C_j \tag{3.24}$$

onde $\bigoplus$ e $>>$ denotam operações de concatenação e translação, respectivamente. Cada ligação entre um transmissor pré-sináptico e um receptor pós-sináptico, por outro lado, ativa diferentes tipos de controladores C_j. Estes controladores exercem diferentes tipos de ações entre as células pré e pós-sinápticas.

$$T_i \bigoplus R_j >> C_j \Rightarrow \text{ação}$$

Além disso, a quantidade $q(C)$ de controladores resultante da concatenação $T \bigoplus R$ é calculada como:

$$q(C) = (q(T) \oplus q(R)) \; o \; \mu(T,R) \tag{3.25}$$

onde $q(T)$ é a quantidade de T; $q(R)$ é a quantidade de R. $\mu(T,R)$ é o grau de *matching* entre T e R e as operações $\oplus$ e o são, em geral, normas triangulares tais como, t-normas ou t-conormas [ROCH 92]. O grau de *matching* $\mu(T,R)$ entre o transmissor e o receptor é uma função da quantidade do controlador C :

$$\mu(T,R) = f(q(C)) \tag{3.26}$$

Baseados, então, nesses conceitos, as interconexões entre os neurônios da DPNN são descritas como segue.

Vamos denotar por:

$$T(X^i_{N-k}) \ \epsilon \ R(X^i_{N-k})$$

um transmissor e um receptor do i-ésimo neurônio-X na camada-X, correspondente ao estágio $N-k$, respectivamente, e

$$T(Y^i_{N-k}) \ \epsilon \ R(Y^i_{N-k})$$

o transmissor e o receptor do (i^{th}) neurônio-Y da camada-Y, para o estágio $N-k$. Suponha também que

$$q(T(Y^i_{N-k})) \ = \ q(R(Y^i_{N-k})) \quad \text{para todo i}$$

e

$$q(T(X^i_{N-k})) \ = \ q(C(X^i_{N-k})) \quad \text{para todo i.}$$

Na DPNN, a quantidade $q(T(X^i_{N-k}))$ corresponde a um valor pertencente ao conjunto X_k, e a quantidade $q(T(Y^i_{N-k}))$ corresponde a um valor pertencente ao conjunto Y_k.

Conexões entre os neurônios da camada-X e os neurônios da camada-Y, correspondente ao estágio $N-k$ [Figura 3.8]

Considere o i-ésimo neurônio-X na camada-X, X^i_{N-k}, e a quantidade de seu receptor denominada $q(R(X^i_{N-k}))$. Na DPNN, este neurônio recebe uma quantidade $q(T(Y^j_{N-k}))$ transmitida pelo j-ésimo neurônio-Y da camada-Y, Y^j_{N-k}. A combinação destas quantidades, de acordo com (3.24), ativa o controlador deste neurônio-X.

A função controladora, na DPNN é associada à equação transformação t_{N-k} em (3.6), no estágio $N-k$, i.é.,

$$q(C(X^i_{N-k})) = y_{N-k} = t_{N-k}(q(T(Y^j_{N-k})), q(R(X^i_{N-k})))$$

onde o par $(q(R(X^i_{N-k})), q(T(Y^j_{N-k})))$ corresponde ao par $(x^i_k, y^j_k) \in (XY)^j_{N-k}$.

Se o valor $q(C(X^i_{N-k})) = y_{N-k}$ for factível, então as conexões entre os neurônios da camada-X e os neurônios da camada-Y são estabelecidas da seguinte forma:

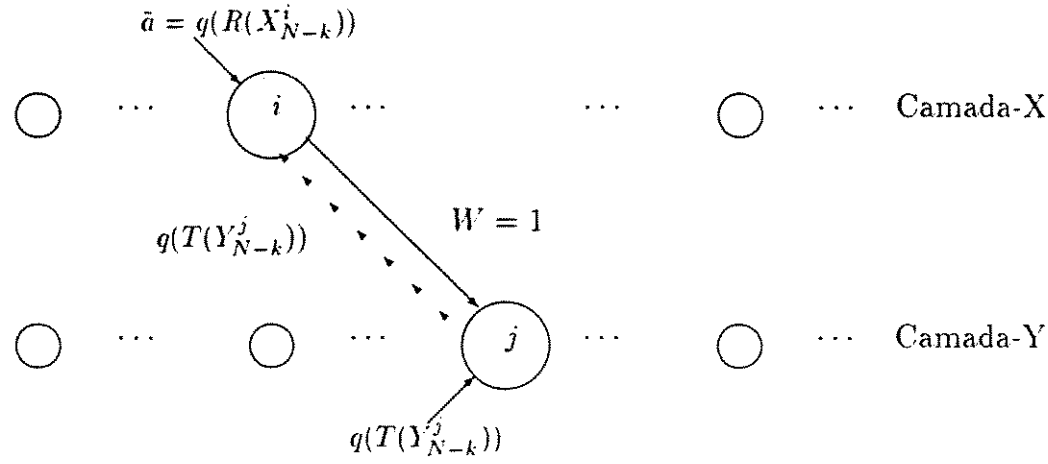


Figura 3.8: Conexões entre neurônio-X e neurônio-Y

$$W(X^i_{N-k}, Y^j_{N-k}) = \begin{cases} 1 & \text{se} \quad q(C(X^i_{N-k})) \geq 0 \\ 0 & \text{caso contrário} \end{cases} \tag{3.27}$$

onde $W(X^i_{N-k}, Y^j_{N-k}) = 1$ denota a conexão entre o i-ésimo neurônio-X da camada-X e o j-ésimo neurônio-Y da camada-Y.

Conexões *feedback* entre os neurônios da camada-Y e os da camada-X [Figura 3.9]

Considere o i-ésimo neurônio-X na camada-X e a quantidade de seu receptor, correspondente ao estágio $N-k$, denominado $q(R(X^i_{N-k}))$. Na DPNN, este neurônio recebe uma quantidade $q(T(Y^j_{N-k}))$ transmitida pelo j-ésimo neurônio-Y da camada-Y. Considere a função controladora definida anteriormente, $q(C(X_{N-k}))$. Analogamente ao caso anterior, a função controladora exerce uma ação sobre o neurônio-X, o qual transmite uma quantidade $q(C(X^i_{N-k}))$ para todos os neurônios-Y da camada-Y, através das conexões *feedback*. Esta ação ativará o neurônio-Y da camada-Y, i.é., o m-ésimo neurônio-Y da camada-Y para o qual

$$q(R(Y^m_{N-k+1})) \;=\; q(T(X^i_{N-k})) \;=\; q(C(X^i_{N-k}))$$

Desta forma, o grau de *matching* entre o transmissor do i-ésimo neuronio-X, no estágio

$N-k$ e o receptor do m-ésimo neurônio-Y, no estágio $N-k+1$,

$$\mu(T(X^i_{N-k}), R(Y^m_{N-k+1}))$$

é uma função da quantidade do controlador, de acordo com (3.26).

Observe que, como estamos nos referindo ao estágio $N-k$ para estabelecer as conexões *feedback*, devemos ressaltar que o neurônio-Y, deverá armazenar além da quantidade do transmissor correspondente ao estágio $N-k$, a quantidade do transmissor relativo ao estágio anterior $N-k+1$. Assim sendo, neurônios do tipo Y deverão conter dois transmissores, um que armazena a quantidade correspondente ao estágio atual e outro que armazena a quantidade correspondente ao estágio anterior.

As conexões *feedback* entre neurônios-Y da camada-Y e os neurônios-X da camada-X, são estabelecidas como segue:

$$W(Y^m_{N-k+1}, X^i_{N-k}) = \begin{cases} 1 & \text{se } q(T(X^i_{N-k})) = q(C(X^i_{N-k})) = q(R(Y^m_{N-k+1})) \\ 0 & \text{caso contrário} \end{cases} \quad (3.28)$$

onde $W(Y^m_{N-k+1}, X^i_{N-k}) = 1$ denota uma conexão entre o m-ésimo neurônio-Y da camada-Y e o i-ésimo neurônio-X da camada-X, i.é., coloca-se peso igual a 1 para uma entrada do neurônio-X.

3.4.2 Dinâmica da DPNN - Um algoritmo

Denotaremos por:

- $\mathcal{X}_{N-k}$ e $\mathcal{Y}_{N-k}$ os conjuntos, como definidos na seção anterior;
- $a_i(Y^j_k)$'s as entradas no j-ésimo neurônio-Y da camada-Y;
- N o número de estágios;
- s a saída de um neurônio da rede.
- $q(R(X^i_{N-k}))$ como a quantidade do receptor do i-ésimo neurônio-X da camada-X, no estágio $N-k$,
- $q(T(X^i_{N-k}))$ como a quantidade do transmissor do i-ésimo neurônio-X da camada-X, no estágio $N-k$.

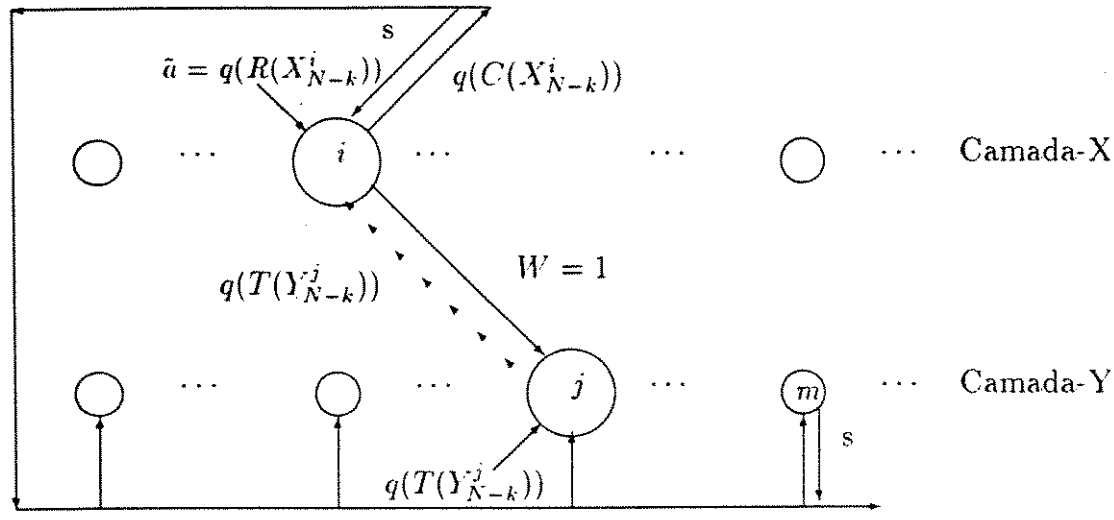


Figura 3.9: Conexões entre neurônio-Y e neurônio-X

- $\tilde{w}_{N-k}$ o coeficiente da variável x_{N-k} na função retorno r_{N-k} no estágio $N-k; k = 0, 1, ..., N-1$.
- t_{N-k} a função transformação; $k = 0, 1, ..., N-1$.

Passo: Inicialização da Rede

Para o i-ésimo neurônio-X na camada-X **faça**

Início

leia($\tilde{w}^i_N$)

leia($q(R(X^i_N))$) = $x^i_N \in \mathcal{X}_N$

$\tilde{a}^i_N = q(R(X^i_N))$

Fim

Para o i-ésimo neurônio-Y na camada-Y **faça**

leia($q(R(Y^i_{N-k}))$) = $y^i_{N-k-1} \in \mathcal{Y}_N$

Estágio N

Para o j-ésimo neurônio-Y na camada-Y **faça**

Para o i-ésimo neurônio-X na camada-Y **faça**

Calcule $q(C(X_N^i)) = t_N(q(T(Y_N^j)), q(R(X_N^i)))$

Se $q(C(X_N^i)) \geq 0$ **então**

Início

$W(X_N^i, Y_N^j) = 1$

$s(X_N^i) = \tilde{w}_N^i * \tilde{a}_N^i$

Fim

senão $W(X_N^i, Y_N^j) = 0$

Saída do Estágio N

Para o t-ésimo neurônio-Y da camada-Y **faça**

Para o i-ésimo neurônio-X da camada-X **faça**

Se $W(X_N^i, Y_N^t) = 1$ **então**

$a_i(Y_N^t) = s(X_N^i)$

Calcular $s(Y_N^t) = \max_l\{a_l(Y_N^t)\}$

Passo - Inicialização do estágio N-k

Para o i-ésimo neurônio-X na camada-X **faça**

Início

leia($\tilde{w}_{N-k}^i$)

leia($q(R(X_{N-k}^i))) = x_{N-k}^i \in \mathcal{X}_{N-k}$

$\tilde{a}_{N-k}^i = q(R(X_{N-k}^i))$

Fim

Para o i-ésimo neurônio-Y na camada-Y **faça**

Início

leia($q(R1(Y^i_{N-k}))$) = $y^i_{N-k-1} \in \mathcal{Y}_{N-k}$

leia($q(R2(Y^i_{N-k}))$) = $y^i_{N-k} \in \mathcal{Y}_{N-(k-1)}$

Fim

Passo Conexões

Criando conexões de neurônio-X para neurônio-Y e vice-versa

Para o j-ésimo neurônio-Y na camada-Y **faça**

Para o i-ésimo neurônio-X na camada-X **faça**

calcule $q(C(X^i_{N-k}))$ = $t_{N-k}(q(T1(Y^j_{N-k})), q(R(X^i_{N-k})))$

Se $q(C(X^i_{N-k})) \geq 0$ **então**

$W(X^i_{N-k}, Y^j_{N-k})$ = 1

Enviar p/ todas as conexões feedback o sinal $q(T(X^i_{N-k}))$ = $q(C(X^i_{N-k}))$

Receber o sinal gerado pelo campo produzido pela ativação dos receptores pós-sinápticos, i.é.,

Para o m-ésimo neurônio-Y da camada Y **faça**

Se $q(R2(Y^m_{N-k}))$ = $q(T(X^i_{N-k}))$ **então**

Receber o sinal de saída do m-ésimo neurônio-Y, $s(Y^m_{N-(k-1)})$ obtido no estágio anterior

Calcular o sinal de saída: $s(X^i_{N-k})$ = $f(\hat{w}^i_{N-k} * \hat{a}^i_{N-k} + 1 * s(Y^m_{N-(k-1)}))$

senão $W(X^i_{N-k}, Y^j_{N-k})$ = 0

Passo: Saída

Para o t-ésimo neurônio-Y da camada-Y faça

Para o i-ésimo neurônio-X da camada-X **faça**

Se $W(X^i_{N-k}, Y^j_{N-k})$ = 1 **então**

$$a_i(Y^t_{N-k}) = s(X^i_{N-k})$$

Calcular $s(Y^t_{N-k}) = \max_l\{a_l(Y^t_{N-k})\}$

Armazenar o valor $q(R(X_{N-k}))$ do neurônio-X correspondente à máxima entrada

Algoritmo Principal

Início

Executar o Passo: Inicialização da Rede

Para $k = 1, 2, ..., N-1$ **faça**

Início

Executar o Passo Inicialização do estágio $N-k$

Executar o Passo Conexões

Executar o Passo Saída

Fim

Fim

A seguir, apresentamos uma equivalência da rede proposta DPNN com o algoritmo tradicional da PD.

3.4.3 Teorema de Equivalência

Teorema: A solução fornecida pela DPNN é equivalente à solução fornecida pela PD.

Prova:

Sem perda de generalidade, vamos assumir que um problema de PD será resolvido usando recursão *backward*. Para mostrar a equivalência, mostraremos que as funções decisão $f_i, i = 1, 2, ..., N$ nos estágios $N-(i-1)$ dadas por (3.6) são também processadas pela DPNN. Lembremos que, para construir a DPNN, as seguintes correspondências foram estabelecidas:

- A cada iteração do processamento da rede corresponde a realização de um determinado estágio da PD,

- Os valores das variáveis discretas x e y correspondem às quantidades dos receptores e transmissores dos neurônios-X e neurônios-Y, respectivamente, i.é.,

$$q(R(X^i_{N-k})) = x^i_{N-k}$$

e

$$q(T(Y^i_{N-k})) = y^i_{N-k-1}$$

para todo i-ésimo neurônio-X e i-ésimo neurônio-Y no estágio $N-k$.

A prova será feita por indução.

a) Primeiro, mostraremos que a ativação da camada-X e camada-Y, no Passo Inicialização da Rede, corresponde ao cálculo realizado no estágio N em (3.6), isto é:

$$f_1 \;=\; r_N(y_{N-1}, x_N) \;\epsilon\; F_1 \;=\; max\{f_1\}$$

são calculados.

De acordo com o algoritmo, as conexões entre a camada-X e camada-Y devem ser estabelecidas. A função controladora

$$q(C(X^i_N)) = t_N(q(T(Y^j_N)), q(R(X^i_N))) = t_N(y^j_{N-1}, x^i_N)$$

é calculada para todo i-ésimo neurônio-X na camada-X. Dependendo do valor obtido, as conexões são ou não estabelecidas. Isto é equivalente a abordagem da PD, que considera somente as combinações das variáveis x e y que tornam t_N factível. Uma vez que as conexões são estabelecidas, o valor de f_1 é calculado.

De fato, de acordo com o Passo Inicialização da Rede, a Saída do Estágio N deve ser processada. Neste caso, cada t-ésimo neurônio-Y recebe n entradas

$$(a_1, a_2, ..., a_l, ..., a_n)$$

onde

$$a_l \;=\; a_l(Y^t_N) \;=\; s(X^i_N) \;=\; \tilde{w}^i_N * \tilde{a}^i_N$$

para $l \leq n$ e algum neurônio-X na camada-X, tal que, existe conexão do i-ésimo neurônio-X, X^i_N, para t-ésimo neurônio-Y, Y^t_N.

Segundo a definição de $\tilde{w}^i_N$ e $\tilde{a}^i_N$ temos que:

$$\tilde{w}^i_N * \tilde{a}^i_N \;=\; r_N(q(R(X^i_N)), q(T(Y^t_N))) \;=\; r_N(x^i_N, y^t_{N-1})$$

Mas, o t-ésimo neurônio-Y, por definição, é um neurônio do tipo Max e assim

$$s(Y_N^t) = max\{a_1, a_2, ..., a_t, ..., a_n\} = max\{r_N(x_N^i, y_{N-1}^i)\} = max\{f_1^i\}$$

e isto é também válido para todo neurônio-Y na camada-Y.

Portanto, $F_1(y_{N-1}^i)$ é calculado no estágio N.

b) Supondo, agora, que no passo k os cálculos desejados no estágio $N-(k-1)$ são computados, devemos mostrar que a iteração $k+1$ também realiza os cálculos corretos correspondentes ao estágio $N-k$. Isto é, devemos mostrar

$$f_{k+1} = r_{N-k}(y_{N-(k+1)}, x_{N-k}) + F_k(y_{N-k}) \tag{3.29}$$

é calculada pela DPNN proposta.

Da hipótese de indução, assumimos que

$$F_k(y_{N-k}) = max\{f_{N-k}^i\}$$

Então, resta-nos mostrar que a primeira parcela de (3.29) é calculada durante a execução da iteração (k+1).

De fato, segundo o Algoritmo Principal, o Passo Conexões deve ser realizado primeiro. Neste passo, as seguintes conexões são ativadas ou estabelecidas:

- Conexões entre os neurônios da camada-X e os da camada-Y,

- Conexões *feedback* entre os neurônios da camada-Y e os da camada-X.

Para estabelecer conexões da camada-X para a camada-Y a função controladora dada por:

$$q(C(X_{N-(k+1)}^i)) = t_{N-k}(q(T(Y_{N-(k+1)}^j)), q(R(X_{N-(k+1)}^i))) = t_{N-k}(y_{N-(k+1)}^j, x_{N-k}^i)$$

é calculada para cada neurônio-Y da camada-Y e todo neurônio-X na camada-X e, dependendo do seu valor, as conexões são ou não estabelecidas. Desta forma, somente as combinações das variáveis x e y que tornam t_{N-k} factível são consideradas.

Com base nesses valores obtidos, $q(C(X_{N-(k+1)}^i))$, as conexões feedback são também ativadas com o objetivo de garantir que os valores de

$$F_{y_{N-k}} = \max_i\{f_{N-k}^i\}$$

obtidos pela rede anteriormente (no passo k), sejam também considerados neste estágio, i.é.,

$$s(X^i_{N-(k+1)}) \;=\; s(Y^m_{N-k}) \;=\; F_k(y^m_{N-k})$$

onde m corresponde ao neurônio-Y tal que:

$$q(R(Y^m_{N-k})) \;=\; q(T(X^i_{N-(k+1)})) \;=\; q(C(X^i_{N-(k+1)})).$$

O próximo passo, é o de determinar o sinal de saída $s(X^i_{N-(k+1)}) = \hat{w}^i_{N-(k+1)} * \tilde{a}^i_{N-(k+1)} + s(Y^m_{N-k})$.

De acordo com a definição, a entrada $\tilde{a}^i_{N-(k+1)}$ no neurônio $X^i_{N-(k+1)}$ juntamente com o seu peso correspondente $\hat{w}^i_{k+1}$ é tal que

$$\hat{w}^i_{N-(k+1)} * \tilde{a}^i_{N-(k+1)} = r_{N-k}(x_{N-k}, y_{N-(k+1)}).$$

Portanto,

$$s(X^i_{N-(k+1)}) \;=\; r_{N-k}(q(R(X^i_{N-(k+1)})), q(T(Y^t_{N-(k+1)}))) \;=\; r_{N-k}(x^i_{N-k}, y^t_{N-(k+1)}).$$

Seguindo o Algoritmo Principal, o Passo Saída desse estágio deve ser calculado. Nesse caso, a saída de cada neurônio-Y da camada-Y é calculada. Supondo que cada neurônio-Y recebe n entradas $(a_1, a_2, \ldots, a_l, \ldots, a_n)$ onde

$$a_l(Y^t_{N-(k+1)}) \;=\; s(X^i_{N-(k+1)})$$

para $l \leq n$ e algum neurônio-X tal que existe uma conexão do neurônio-X, $X^i_{N-(k+1)}$, para neurônio-Y, $Y^t_{N-(k+1)}$ e lembrando que o neurônio-Y, por definição é um neurônio Max, nós temos:

$$s(Y^t_{N-(k+1)}) \;= max\{a_1, a_2, \ldots, a_n\} \;=$$

$$max\{r_{N-k}(x^i_{N-k}, y^t_{N-(k+1)})\} + F_k(y^m_{N-k}) \;=\; max\{f^i_{k+1}\}$$

onde n denota o número de neurônios-X que estão conectados ao neurônio-Y.

Desde que este fato também é válido para todo neurônio-Y na camada-Y, seguramente concluímos que os cálculos desejados para o estágio $N-k$ são executados pela DPNN e assim o teorema está provado.

3.4.4 Exemplo Ilustrativo

Nesta seção, o Algoritmo Principal para a rede neural proposta é desenvolvido para ilustrar como a DPNN é construída. Consideremos para tal o seguinte problema de otimização:

$$\begin{aligned} max \quad & f = 8x_1 + 10x_2 \\ \text{sujeito a} \quad & 4x_1 + 2x_2 \leq 12 \\ & x_1 \, , x_2 \geq 0 \end{aligned} \tag{3.30}$$

A DPNN correspondente é uma rede do tipo da proposta na Figura 3.7. No entanto, os passos do Algoritmo são melhor vizualizados na Figura 3.10.

Esse exemplo pode ser resolvido com $N = 2$ estágios.

Seja:

$\mathcal{X}_2 = \{x_2^i\} = \{0,1,2,3,4,5,6\}$,

$\mathcal{X}_1 = \{x_1^i\} = \{0,1,2,3\}$,

$t_1 = y_0 - 4x_1$ and $t_2 = y_1 - 2x_2$: (funções transformação),

$\mathcal{Y}_2 = \{y_1^i\} = \{0,4,8,12\}$,

$\mathcal{Y}_1 = \{y_0^i\} = \{12\}$,

$\tilde{w}_2^i = 10 \, ; i = 0,1,\ldots,6$. $\tilde{w}_1^j = 8 \, ; j = 0,1,2,3$. $N = 2$.

Durante o Passo de Inicialização da Rede, as seguintes variáveis são inicializadas:

$q(R(X_2^i)) = x_2^i \in \mathcal{X}_2 \quad i = 0,1,2,\ldots,6$

$q(R(X_1^i)) = x_1^i \in \mathcal{X}_1 \quad i = 0,1,2,3$

$q(R(Y_2^i)) = y_1^i \in \mathcal{Y}_1 \quad i = 0,1,2,3$

$q(R(Y_1^0)) = 12$

$\tilde{a}_2^i = q(R(X_2^i)) \quad i = 0,1,\ldots,6$

$\tilde{a}_1^i = q(R(X_1^i)) \quad i = 0,1,2,3.$

Nesse Passo, as conexões entre a camada-X e camada-Y são criadas, correspondentes a realização do Estágio N. Para $j = 0,1,2,3$ e para $i = 0,1,2,\ldots,6$ a quantidade

$$q(C(X_2^i)) = t_2(q(T(Y_2^j)), q(R(X_2^i))) = q(T(Y_2^j)) - 2 * q(R(X_2^i))$$

é calculada e as interconexões são estabelecidas dependendo do seu valor. Por exemplo, para $j = 1$ e $i = 0, 1, 2, ..., 6$ as seguintes conexões são ativadas:

$$W(X_2^i, Y_2^1) = 1 \; ; \; i = 0, 1, 2$$

porque

$$q(C(X_2^i)) = 4 - 2 * x_2^i \geq 0; \; i = 0, 1, 2$$

e

$$W(X_2^i, Y_2^1) = 0 \; ; \; i \geq 3$$

porque

$$q(C(X_2^i)) = 4 - 2 * x_2^i < 0; \; i \geq 3.$$

As saídas $s(X_2^i) = \hat{w}_2^i * a_2^i = 10 * x_2^i; \; i = 0, 1, 2$ são também calculadas. As conexões estabelecidas são mostradas na Figura 3.10, que apresenta o processamento detalhado da rede em cada estágio.

Ainda executando esse mesmo Passo, a saída de cada neurônio-Y na camada-Y, correspondente ao Estágio N, é calculada. Consideremos, então, o neurônio-Y, Y_2^1 e sua saída $s(Y_2^1)$,

$$s(Y_2^1) = max\{a_1, a_2, a_3\}$$

onde

$$a_1 = s(X_2^0)) = 10 * 0 = 0$$
$$a_2 = s(X_2^1)) = 10 * q(R(X_2^1)) = 10 * 1 = 10$$
$$a_3 = s(R(X_2^2)) = 10 * q(R(X_2^2)) = 10 * 2 = 20$$

Portanto,

$$s(Y_2^1) = max\{0, 10, 20\} = 20.$$

Analogamente, as saídas dos neurônios-Y restantes, Y_2^0, Y_2^2, Y_2^3 são calculadas.

A seguir, $k = 1$ e o Passo Conexões é executado. Nesse passo, as conexões *feedback* entre a camada-Y e a camada-X, bem como, entre a camada-X e camada-Y são designadas. Para estabelecer essas conexões, a função controladora:

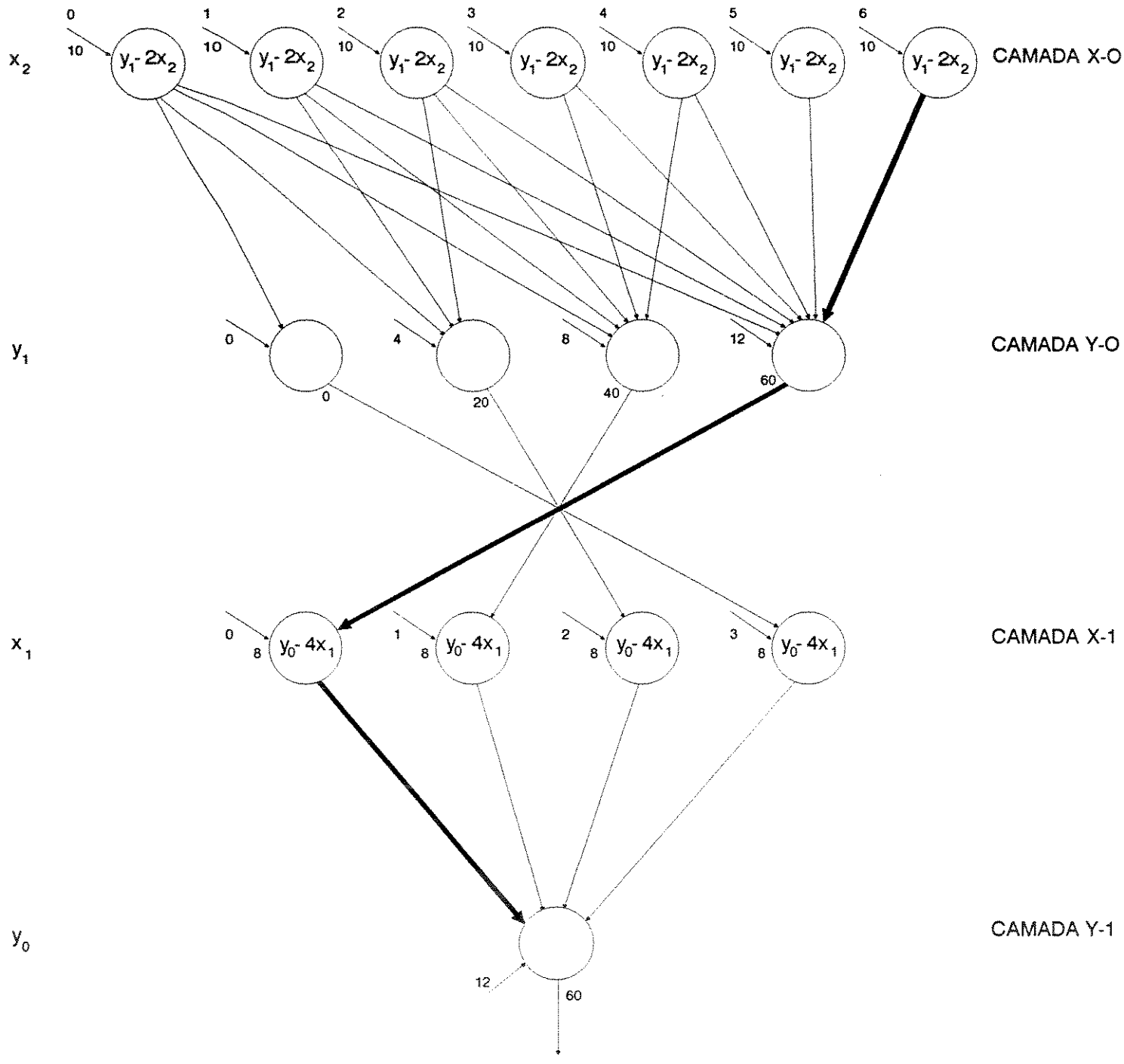


Figura 3.10: DPNN para o Exemplo Ilustrativo

$$q(C(X_1^i)) = t_1(q(R(X_1^i)), q(T(Y_1^0))) = q(T(Y_1^0)) - 4 * q(R(X_1^i))$$

é calculada para $i = 0, 1, 2, 3$. Para ilustrar, considere o neurônio-Y, Y_1^0, e o neurônio-X, X_1^1. Temos então que:

$$q(C(X_1^1)) = 12 - 4 * 1 = 8 > 0$$

Isto implica que $W(X_1^1, Y_1^0) = 1$ e $W(Y_2^2, X_1^1) = 1$ porque o neurônio Y_2^2 é o que satisfaz

$$q(R(Y_2^2)) = q(T(X_1^1)) = q(C(X_1^1)) = 8.$$

O neurônio X_1^1 recebe a saída s do neurônio Y_2^2, $s(Y_2^2) = 40$, e sua saída é dada por:

$$s(X_1^1) = f(\tilde{w}_1^1 * \tilde{a}_1^1 + s(Y_2^2)) = 8 * 1 + 40 = 48.$$

A seguir o Passo Saída é realizado. Nesse Passo, a saída de um único neurônio-Y é calculada do seguinte modo:

$$s(Y_1^0) = max\{a_1, a_2, a_3, a_4\}$$

onde

$$a_1 = s(X_1^0) = 8 * 0 + 60 = 60;$$
$$a_2 = s(X_1^1) = 8 * 1 + 40 = 48;$$
$$a_3 = s(X_1^2))) = 8 * 2 + 20 = 36;$$
$$a_4 = s(R(X_1^3)) = 8 * 3 + 0 = 24;$$

e, portanto,

$$s(Y_1^0) = max\{60, 48, 36, 24\} = 60$$

que corresponde ao valor ótimo da função objetivo (f.o.). A solução ótima pode ser recuperada através das conexões em negrito na Figura 3.10, fornecendo

$$x_1^* = 0; x_2^* = 6; \text{f.o.} = 60.$$

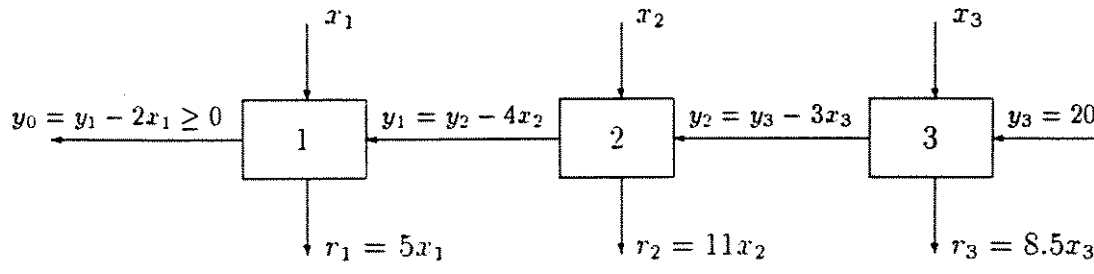


Figura 3.11: Diagrama de Fluxo

3.4.5 Aplicações

O Problema da Mochila

O problema clássico da mochila pode agora ser resolvido pela abordagem proposta:

$$max \quad 5x_1 + 11x_2 + 8.5x_3$$

$$\text{sujeito a} \quad 2x_1 + 4x_2 + 3x_3 \leq 10$$

Resolvendo-o por programação dinâmica com um estágio para cada item ($n = 1, 2, 3$) e com variáveis de decisão x_n, obtém-se o diagrama de fluxo mostrado na Figura 3.11.

A DPNN correspondente para este exemplo é uma rede do tipo da mostrada na Figura 3.7. No entanto, para melhor compreensão a Figura 3.12 mostra o processamento da rede em cada estágio.

Observe que os primeiros neurônios-X da terceira camada da Figura 3.12 recebem várias entradas, cada uma num dado instante de tempo. Estes sinais de entrada são emitidos dinamicamente quando os neurônios da quarta camada são ativados. Um neurônio do tipo X emite vários sinais de saída, mas isto não contradiz a definição de neurônio, uma vez que estes sinais são emitidos cada um a seu tempo. Isto também é válido para os neurônios da quinta camada.

A solução ótima pode ser recuperada através das conexões vistas em negrito na Figura 3.12, obtendo-se:

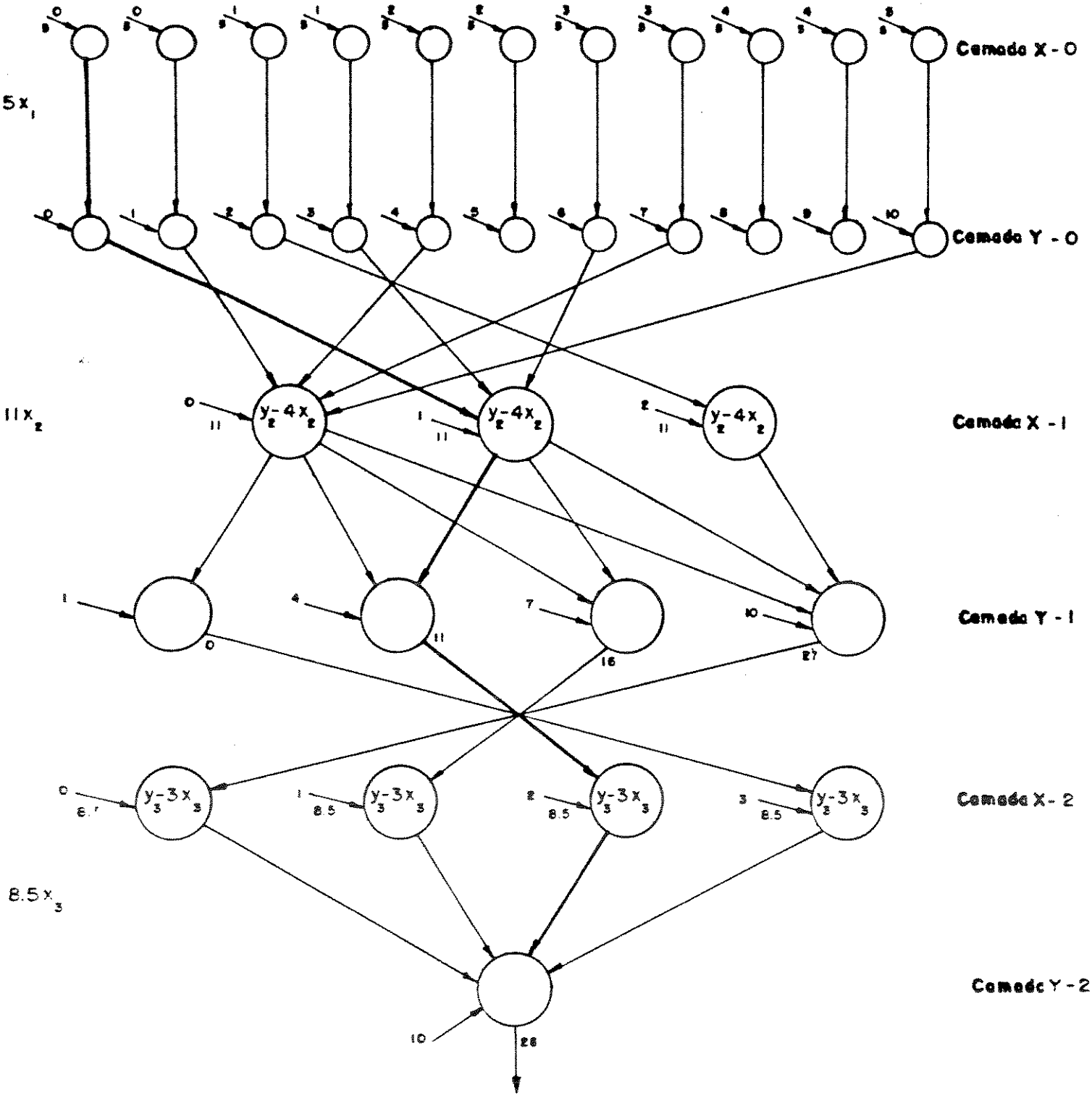


Figura 3.12: Uma DPNN para o problema da mochila

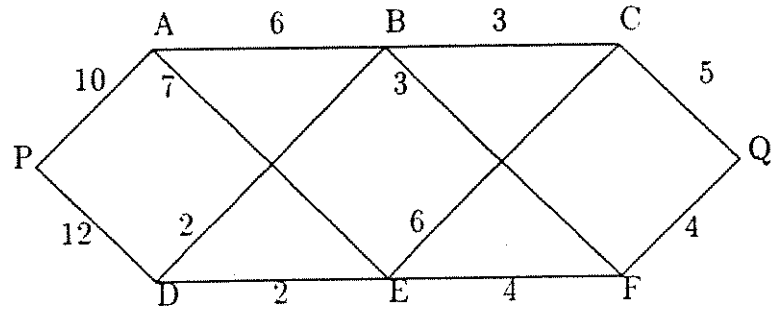


Figura 3.13: Mapa de Caminhos

$$x_1^* = 0; x_2^* = 1; x_3^* = 2; \text{f.o.} = 28.$$

O problema do Caminho Mínimo

Talvez a mais direta aplicação da PD é a determinação do caminho ou da rota mínima em uma rede. Considere o mapa contendo diferentes caminhos na Figura 3.13. Um viajante quer encontrar a rota mínima para sair do ponto P e chegar ao ponto Q. Existem seis pontos intermediários A, B, ..., F. Os comprimentos de cada caminho entre esses pontos são indicados no mapa. Qualquer caminho de P a Q representa uma rota candidata a solução.

Assumindo que a direção de tráfego é sempre da esquerda para à direita, o número de rotas possíveis é finito. O problema pode, portanto, ser resolvido enumerando todas as rotas possíveis e comparando o total de seus comprimentos.

Figura 3.14 mostra uma estrutura de decisão de 4-estágios para o problema. Não existe nenhuma escolha no estágio 4, uma vez que o estado final é o ponto $y_4 = Q$ dado.

Adotando, nesse caso, recursão *forward*, a DPNN correspondente é uma rede do tipo apresentada na Figura 3.7. No entanto, a Figura 3.15 é apresentada com a finalidade de exibir o processamento da rede em cada estágio. Nesta Figura, os pesos não são indicados, porque todos são iguais a 1. A solução pode ser recuperada através das conexões vistas em negrito [Figura 3.15], obtendo-se o caminho mínimo: PDBFQ.

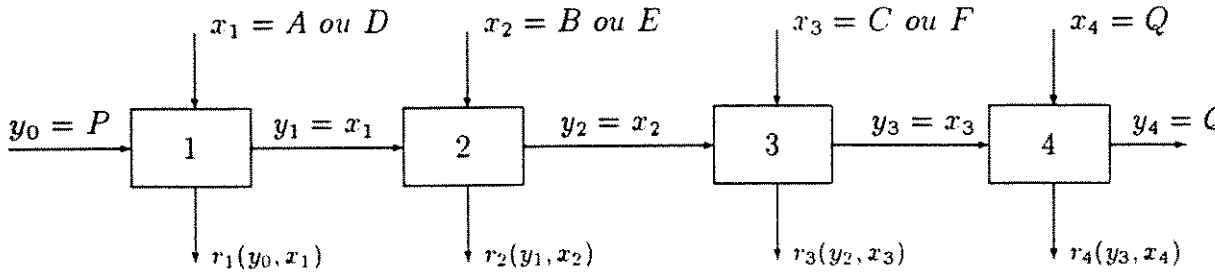


Figura 3.14: Diagrama de Fluxo para o problema do caminho mínimo

Uma questão que necessita de esclarecimentos aqui, refere-se a obtenção da solução de um problema dado, quando a rede termina o seu processamento. De acordo com o algoritmo proposto, o valor da variável x, $q(R(X))$, que correspondente à entrada máxima no neurônio-Y, deve ser armazenado junto ao neurônio-Y, em cada estágio. Assim, baseando-se no modelo matemático de neurônio proposto por Bezdek [BEZD 91], propomos o modelo para o neurônio-Y, mostrado na Figura 3.16. De acordo com este modelo o neurônio-Y deve possuir uma memória para armazenar os correspondentes valores da variável x em cada estágio.

Na Figura 3.16, $s(Y)$ denota a saída do neurônio-Y, que por sua vez é dada por:

$$s(Y) = a_j(Y) = \max_l \{a_l(Y)\}$$

sendo $a_l(Y) = s(X^k)$ para algum k-ésimo neurônio-X na camada-X. Então, se $a_j(Y) = s(X^t)$, o valor $q(X^t)$ deve ser armazenado como o valor ótimo da variável x correspondente a variável de estado y.

3.4.6 Generalização da DPNN

Até agora consideramos problemas nos quais uma única variável por estágio era necessária. Entretanto, a aplicação da PD não se limita a este caso especial.

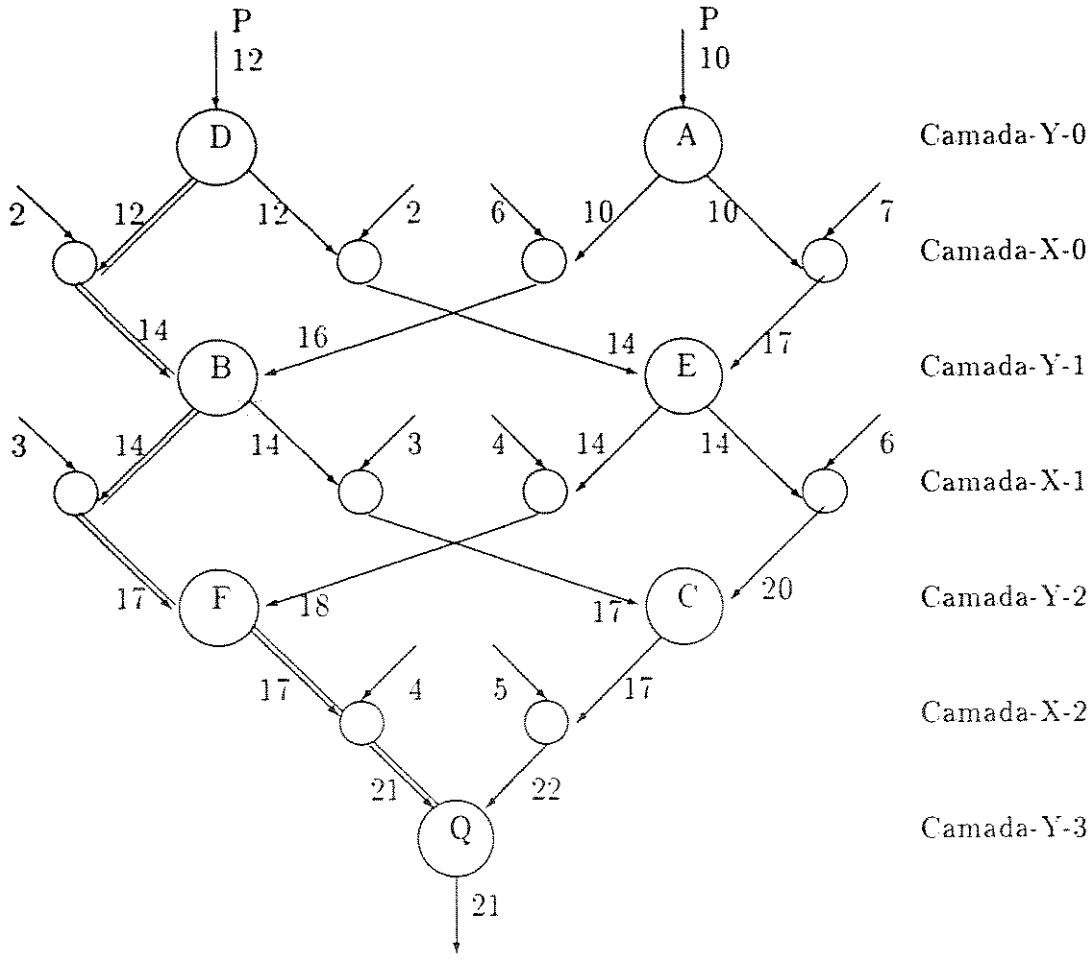


Figura 3.15: DPNN para o problema do caminho mínimo

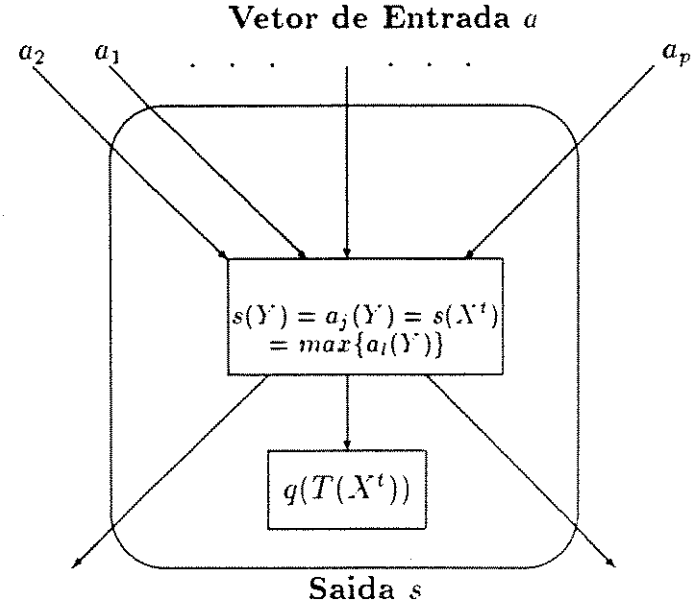


Figura 3.16: Modelo Neurônio-Y

A decomposição em estágios levará, em muitos casos, a problemas de otimização envolvendo várias variáveis de decisão por estágio de modo que x_n não é uma variável simples mas, um vetor. As variáveis de estado y_n podem também ser vetores porque, em princípio, cada restrição requer uma variável de estado.

Consideremos o seguinte problema:

$$max \quad f(x)$$

$$sujeito \ a \quad g(x) \leq 0$$

$$x \geq 0 \quad inteiros$$

onde a função decisão f é uma função separável, $g : R^n \rightarrow R^m$ e $x \in R^n$ é um vetor de inteiros não-negativos.

Para resolver este problema através da DPNN, observemos que as soluções paramétricas em cada estágio são, agora, funções das m variáveis de estado.

$$x_n = x_n(y_n^1, y_n^2, ..., y_n^m) \quad \epsilon \quad F_n = F_n(y_n^1, y_n^2, ..., y_n^m)$$

Então, de acordo com a abordagem proposta, cada neurônio-Y na DPNN deverá ter tantos receptores quantos forem o número de variáveis de estado. Isto implica que o número de neurônios-Y na camada-Y deverá ser igual ao número de combinações possíveis entre os valores das variáveis de estado. A mesma situação ocorre para as funções controladoras. No restante, o processamento da DPNN é exatamente como descrito na seção 3.4.2. As modificações necessárias nos passos descritos na seção 3.4.2 são apresentadas a seguir.

Estágio N

Conexões entre os neurônios da camada-X e os camada-Y, no estágio N

Para o j-ésimo neurônio-Y na camada-Y **faça**

Para o i-ésimo neurônio-X na camada-X **faça**

$p = 1$

Calcule $q(C^1(X_N^i))$

Enquanto $(p \leq (m-1)$ **e** $q(C^p(X_N^i)) \geq 0$) **faça**

$p = p+1$

Calcule $q(C^p(X_N^i)) = t_N(q(T^p(Y_N^j)), q(R(X_N^i)))$

Se $(p = m)$ **então**

$W(X_N^i, Y_N^j) = 1$

senão $W(X_N^i, Y_N^j) = 0$

Passo Conexões

Conexões entre neurônios camada-X e os da camada-Y, no estágio $N-k$

Para o j-ésimo neurônio-Y na camada-Y **faça**

Para o i-ésimo neurônio-X na camada-X **faça**

$p = 1$

Calcule $q(C^1(X_{N-k}^i))$

Enquanto ($p \leq m$ **and** $q(C^p(X_{N-k}^i)) \geq 0$) **faça**

$p = p + 1$

Calcule $q(C^p(X^i_{N-k})) = t_{N-k}(q(T^p(Y^j_{N-k})), q(R(X^i_{N-k})))$

Se $(p = m)$ **então**

Para $p = 1$ to $m - 1$ **faça**

$q(T^p(X^i_{N-k})) = q(C^p(X^i_{N-k}))$

$W(X^i_{N-k}, Y^j_{N-k}) = 1$

senão $W(X^i_{N-k}, Y^j_{N-k}) = 0$

Conexões *feedback* **entre neurônios da camada-Y e os da camada-X**

Para o q-ésimo neurônio-Y da camada-Y **faça**

$p = 1$

Enquanto $(p \leq m - 1)$ **faça**

Se $q(R^p(Y^q_{N-k+1})) = q(T^p(X^q_{N-k}))$ **então**

$p = p + 1$

Se $(p = m)$ **então**

$W(Y^q_{N-k+1}, X^i_{N-k}) = 1$

Receba a saída s do q-ésimo neurônio-Y, $s(Y^q_{N-k+1})$

Calcule o sinal de saída $s(X^i_{N-k}) = f_t(\tilde{w}^i_{N-k} * \tilde{a}^i_{N-k} + s(Y^q_{N-k+1}))$

Para ilustrar esse caso consideremos o seguinte exemplo:

$$\begin{aligned} max \quad f &= 4x_1 + x_2 + x_3 \\ sujeito\ a \quad x_1 + x_2 &\leq 1 \\ 2x_1 \quad + x_3 &\leq 2 \end{aligned}$$

interpretado como um problema de capacidade de utilização ótima com 2 restrições de capacidade. As variáveis x_n, $n = 1, 2, 3$ devem ser inteiras não-negativas. O problema

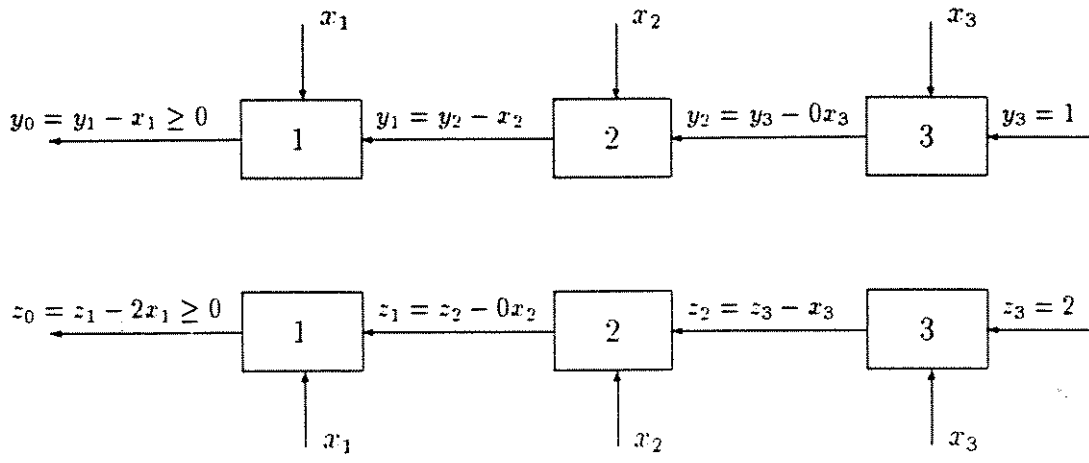


Figura 3.17: Diagrama de Fluxo

pode ser decomposto num problema de três estágios como pode ser visto na Figura 3.17, adotando recursão *forward*.

Para cada um dos dois recursos consumidos sucessivamente, introduzimos as variáveis de estado, y_n e z_n: $n = 0, 1, 2, 3$. O estado final é dado por $y_3 = 1$; $z_3 = 2$; para o estado inicial temos que $y_0 \geq 0, z_0 \geq 0$ (i.é., as variáveis de folga do problema PL são não-negativas).

As variáveis do problema podem tomar os seguintes valores:

variável	valor
x_1	0, 1
x_2	0, 1
x_3	0, 1, 2
y_n	0, 1
z_n	0, 1, 2

A DPNN que resolve este problema é uma rede do tipo da apresentada na Figura 3.7. A Figura 3.18 mostra o processamento detalhado da DPNN correspondente a cada estágio. Observe que existem duas funções controladoras em cada neurônio-X e, cada neurônio-Y tem dois receptores, correspondendo as duas variáveis de estado y_n, z_n.

A solução ótima pode ser recuperada através das conexões em negrito vistas na Figura

3.18, obtendo-se:

$$y_3 = 1;\ z_3^* = 2;\ x_3^* = 0$$

$$y_2 = 1; z_2^* = 2;\ x_2^* = 0$$

$$y_1 = 1; z_1^* = 2\, x_1^* = 1$$

$$y_0 = 0; z_0 = 0.$$

3.4.7 Análise dos Requisitos Computacionais

Consideremos um problema de otimização com N estágios, k_i e j_i valores para as variáveis de estado e decisão, respectivamente no estágio i. $i = 1, 2, ..., N$.

De acordo com [NEMH 66], a solução pela PD requer uma adição para cada combinação (y_i, x_i) nos estágios $2, ..., N$, ou seja

$$\sum_{i=2}^{N} k_i j_i$$

adições. Para cada valor da variável de estado existem, em todos os estágios, $j_i - 1$ comparações e no último estágio existem $k_i - 1$ comparações para se determinar o máximo de $f_N(x_N)$. Um total de

$$\sum_{i=1}^{N} k_i(j_i - 1) + k_i - 1$$

comparações são necessárias.

Vamos assumir que o tempo gasto para uma adição e uma comparação seja t_A e t_C, respectivamente. Então, o tempo total t_{DP} necessário para a solução via PD é

$$t_{DP} = (\sum_{i=2}^{N} k_i j_i) t_A + (\sum_{i=1}^{N} k_i(j_i - 1) + k_i - 1) t_C.$$

Para a DPNN, uma vez que os cálculos relativos aos neurônios-Y são realizados em paralelo, as comparações correspondentes também são processadas em paralelo e assim o tempo total t_{NN} necessário para a solução do problema utilizando a DPNN proposta é

$$t_{NN} = (\sum_{i=2}^{N} k_i j_i) t_A + N t_C$$

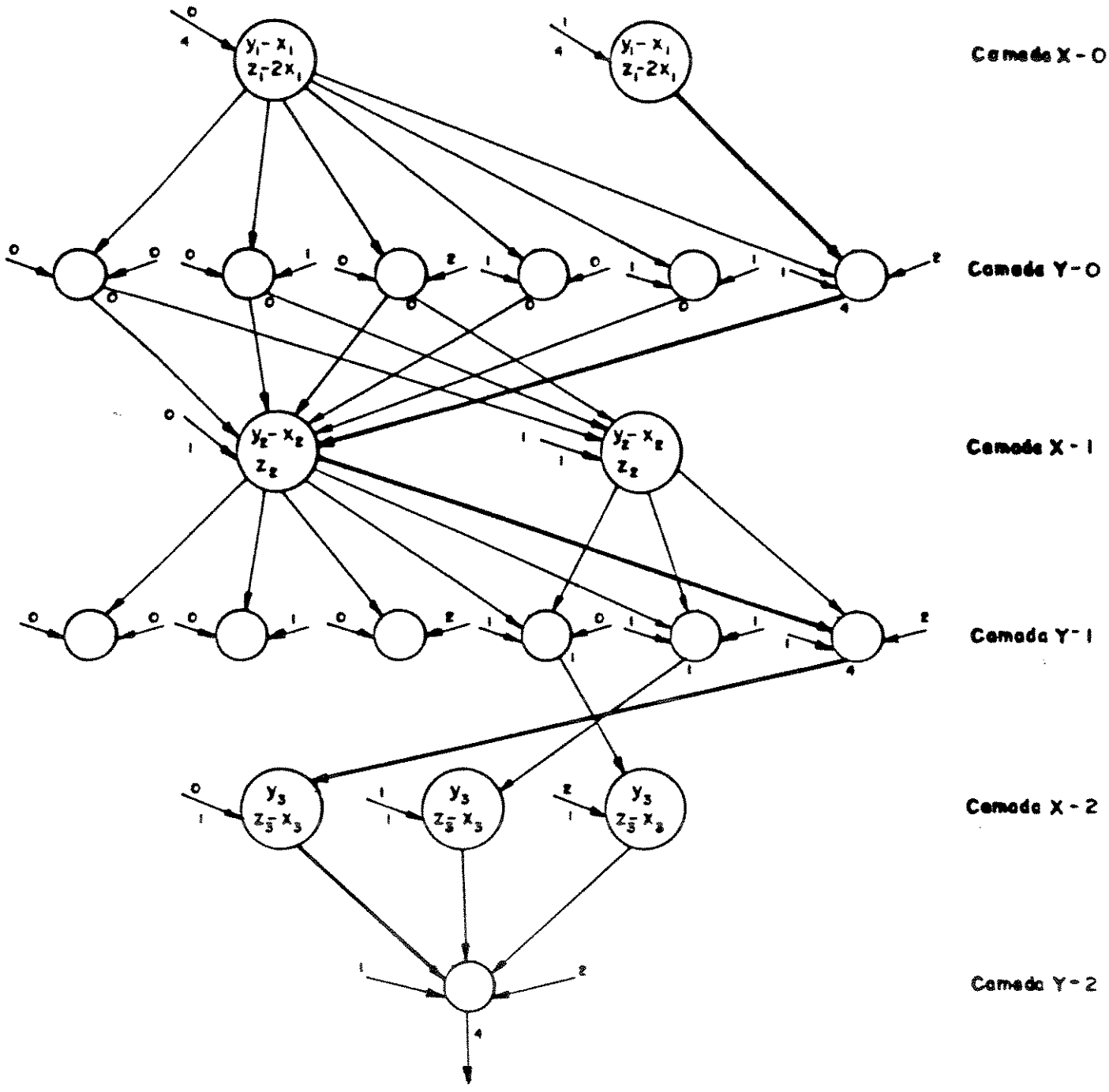


Figura 3.18: Uma DPNN para resolver o problema PL

Claramente,

$$t_{NN} < t_{DP}.$$

Relativamente ao armazenamento necessário até a obtenção da decisão ótima, com referência a um esquema *backward*, Nemhauser [NEMH 66] faz as seguintes considerações.

Primeiramente, observemos que

$$x_k(y_{k-1}) \ , \quad k = N, N-1, ..., 2, 1$$

é armazenado até que o ótimo seja obtido, mas

$$F_k(y_{N-k})$$

é armazenado somente até

$$F_{k+1}(y_{N-(k+1)})$$

ser calculado. Dessa forma, deve haver capacidade de armazenamento suficiente para as N funções $x_k(y_{k-1})$ e para os dois valores consecutivos de F_k.

Observemos também que, para uma variável de decisão de s-componentes existem $s+1$ tabulações de funções ótimas para cada valor da variável $y_{N-(k+1)}$, consistindo de uma única tabulação de $F_k(y_{N-k})$ e s tabulações de $x_k(y_{k-1})$.

Assumindo que existem N estágios, p variáveis de estado, cada uma tendo k valores factíveis e s variáveis de decisão, por estágio, então, o armazenamento total necessário para obtenção do ótimo via PD é

$$N(s + 2)k^p.$$

O fator que tem maior influência sobre esse resultado é o número de variáveis de estado por estágio p.

Para o modelo de rede neural proposto, uma vez que toda informação é armazenada na topologia da rede, temos que é necessário uma rede DPNN de duas camadas com k^p neurônios-Y na camada-Y e s neurônios-X na camada-X para a obtenção da solução ótima. Por exemplo, se o lado direito das duas restrições do problema LP na seção anterior é igual a 10 e 20, respectivamente, a camada-Y deveria ter $11 * 21 = 231$ neurônios-Y (na abordagem de PD, seriam necessárias N tabelas com 231 linhas cada). O número de neurônios-X na camada-X aumenta com o número de valores factíveis para x_n (na abordagem da PD, isto também acontece, e acontece para as N tabelas).

3.5 FDPNN - Rede Neural para Otimização *Fuzzy*

A abordagem desenvolvida na seção anterior é estendida nesta seção para resolver problemas de otimização *fuzzy*. As modificações necessárias, bem como, a rede neural proposta para resolver esta classe de problemas de otimização, são apresentadas.

3.5.1 Definição da FDPNN

Para a apresentação da FDPNN, primeiramente definiremos os neurônios que constituem a FDPNN. Em seguida, definiremos as interconexões entre os neurônios definidos.

Definição dos Neurônios da FDPNN

A rede FDPNN proposta nesta seção é constituída de dois tipos diferentes de neurônios: neurônio-M e neurônio-m, descritos a seguir.

Assumiremos, sem perda de generalidade, um esquema *backward*, para resolução de problemas de PD.

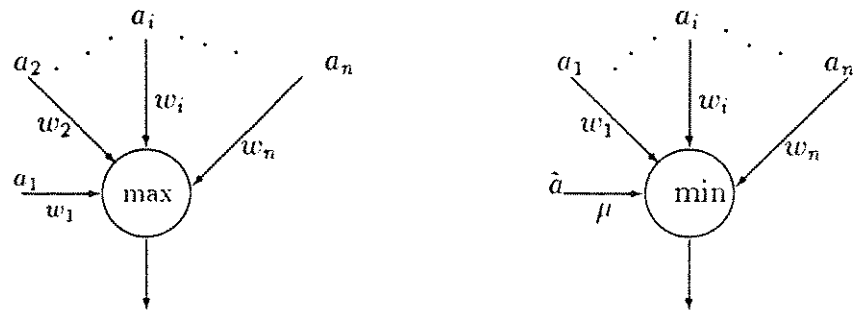
Seja:

- $U_k = \{u_k^1, u_k^2, ..., u_k^p\}$ o conjunto de valores discretos que a variável de decisão u_{N-k} pode assumir no estágio $N-k$; $k = 1, 2, ..., N$;
- $X_k = \{x_k^1, x_k^2, ..., x_k^l\}$ o conjunto de valores discretos que a variável de estado x_{N-k} pode assumir no estágio $N-k$;$k = 1, 2, ..., N$;
- $$(UX)_{N-k}^j = \{(u_k^i, x_k^j),\ u_k^i \in U_k,\ x_k^j \in X_k\ / f_{N-k}(u_k^i, x_k^j) \text{ existe}\},$$

  onde f_{N-k} é uma função de controle não-*fuzzy* como em (3.10);
- n o número de elementos do conjunto $(UX)_{N-k}^j$.

Um **neurônio-M** [Figura 3.19] é um neurônio do tipo Max, como definido na seção 3.3. Um **neurônio-m** [Figura 3.19] é um neurônio do tipo Min, que recebe vários sinais de entrada a_i, sincronizadas pelo mesmo mecanismo adotado pelo neurônio Min (ou neurônio Max) para fornecer uma saída s.

A saída de um neurônio-m, s, é dada por:



neurônio-M **neurônio-m**

Figura 3.19: Neurônios da FDPNN

$$s = \mu(\hat{a}) \wedge w_i a_i$$

onde μ é uma função peso que atua sobre a entrada $\hat{a}$ produzindo $\mu(\hat{a})$ e a_i são entradas recebidas de outros neurônios cujos pesos são w_i. Na FDPNN, $s = \mu(\hat{a}) \wedge a_i$ pois $w_i = 1$ e as entradas nos neurônios-m têm o seguinte significado. Considerando os pares de entrada no neurônio-m:

$$(\mu(\hat{a}), w_i a_i) = (\mu(\hat{a}), a_i)$$

temos que:

- $\mu(\hat{a})$: corresponde ao valor fornecido pela função de pertinência μ, calculada num determinado valor, $\hat{a}$ da variável x_{N-k}.

- a_i são as entradas recebidas de outros neurônios. (Na FDPNN esses neurônios são os neurônios-M).

O conceito de função peso será definido no capítulo 4, onde um neurônio recorrente generalizado será proposto.

A DPNN é constituída de dois tipos diferentes de camadas: camada-M, que é aquela composta somente de neurônios do tipo M e camada-m, que é composta somente de neurônios do tipo m.

Na proposta inicial, as camadas da FDPNN alternavam entre camadas-M e camadas-m, sendo que a primeira camada deveria ser uma camada do tipo camada-M. A rede

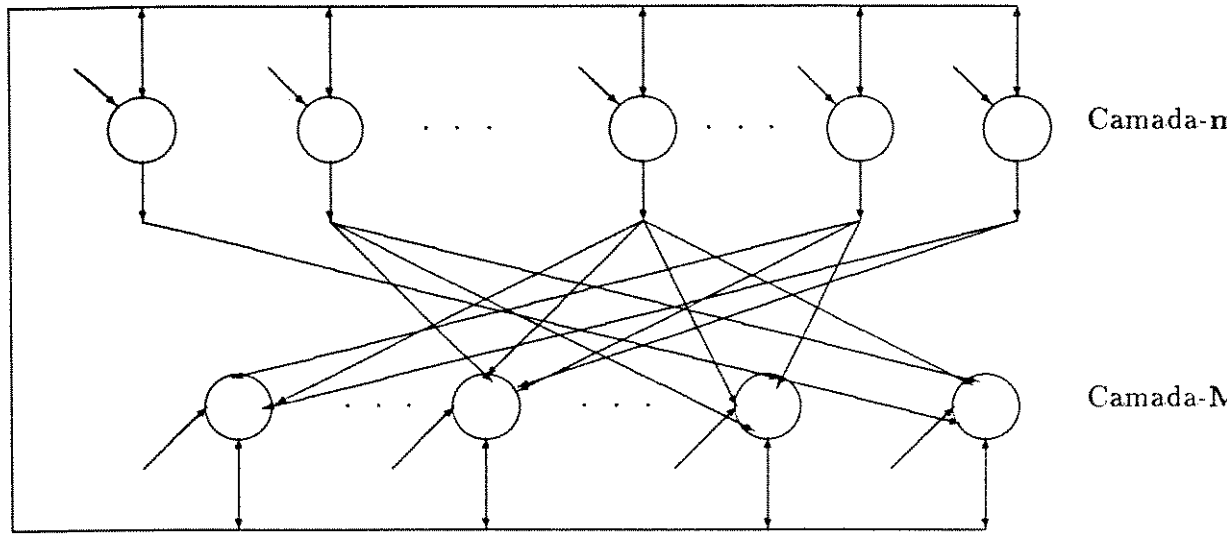


Figura 3.20: Rede Neural para Problemas *Fuzzy* -FDPNN

FDPNN deveria ter um total de $2 * N$ camadas, sendo N o número de estágios necessários para se obter a solução do problema em questão via PD. Os detalhes dessa proposta inicial estão contidos em [FRA7 92,FRAS 93].

Analogamente, ao caso anterior, investigamos a possibilidade de reduzir o número de camadas, uma vez que este número aumentava com o número de estágios. Portanto, propomos aqui uma rede neural constituída de apenas duas camadas, com *feedback* [Figura 3.20], sendo que a primeira camada é uma camada do tipo camada-**m** e a segunda camada é do tipo camada-**M**. As interconexões entre os neurônios dessas duas camadas são definidas a seguir.

Interconexões entre os neurônios

- Conexões entre a camada-**m** e a camada-**M**

  As interconexões (pesos) da camada-**m** para camada-**M** são designadas, a partir de uma tabela de transição de estados dada, como segue:

  $$W(m_k^i, M_k^j) = 1 \quad \text{se} \quad f_{N-k}(q(R(m_k^i)), q(T(M_k))) \quad \text{existe}$$

  onde $W(m_k^i, M_k^j) = 1$ denota a conexão entre o i-ésimo neurônio-m da camada-**m** e o j-ésimo neurônio-M da camada-**M**, no estágio $N - k$.

- Conexões *feedback* entre a camada-**M** e a camada-**m**

Para definição das interconexões entre os neurônios das duas camadas, os conceitos descritos e detalhados em [ROCH 92] são também usados. Adotando a mesma notação anterior, seja:

$$T(m^i_{N-k}) \text{ e } R(m^i_{N-k})$$

um transmissor e um receptor do i-ésimo neurônio-m, no estágio $N-k$, respectivamente e seja

$$T(M^i_{N-l}) \text{ e } R(M^i_{N-l})$$

o transmissor e receptor do i-ésimo neurônio-M, no estágio $N-l$, respectivamente.

Supomos também que

$$q(T(M^i_{N-l})) = q(R(M^i_{N-l})) \text{ para todo i}$$

e

$$q(T(m^i_{N-k})) = q(C(m^i_{N-k})) \text{ para todo i.}$$

A quantidade $q(T(m^i_{N-k}))$ corresponde a um valor pertencente ao conjunto U_k, e a quantidade $q(T(M^i_{N-l}))$ corresponde a um valor pertencente ao conjunto X_l.

Consideremos, então, o i-ésimo neurônio-m na camada-m e a quantidade de seu receptor denominado, $q(R(m^i_{N-k}))$, correspondente ao estágio $N-k$. Na FDPNN, este neurônio recebe uma quantidade, $q(T(M^j_k))$, transmitida pelo (j-ésimo)neurônio-M da camada-M. A combinação destas quantidades ativa o controlador (função controladora) deste neurônio-m.

A função controladora é definida como sendo a equação transformação ou função transição de estados f_{N-k} citada em (3.10), i.é.,

$$q(C(m^i_{N-k})) = y_{N-k} = f_{N-k}(q(R(m^i_{N-k})), q(T(M^j_{N-k})))$$

onde o par $(q(R(m^i_{N-k})), q(T(M^j_{N-k})))$ corresponde ao par $(u^i_k, x^j_k) \in (UX)^j_{N-k}$.

Usando o fato que o controlador pode participar na especificação do transmissor liberado no terminal pré-sináptico, ele exerce uma ação no neurônio-m, o qual transmite uma quantidade $q(C(m^i_k))$ para todos os neurônios-M da camada-M. Esta ação ativará um neurônio-M, por exemplo, o t-ésimo neurônio-M para o qual

$$q(R(M^t_{N-k+1})) = q(T(m^i_{N-k})) = q(C(m^i_{N-k}))$$

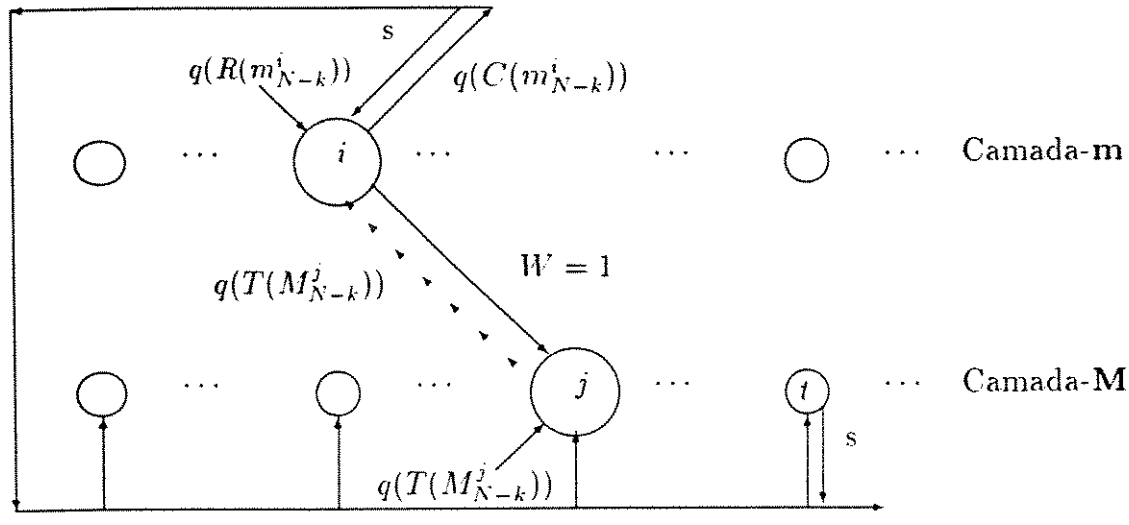


Figura 3.21: Conexões entre os neurônios do tipo **M** e **m**

Da forma como essas interconexões foram definidas, o grau de *matching*

$$\mu(T(m^i_{N-k}), R(M^t_{N-k+1}))$$

entre o transmissor $T(m^i_{N-k})$ e seu $R(M^t_{N-k+1})$ é uma função da quantidade do controlador, concordando com a equação (3.26).

As conexões *feedback* entre neurônios da camada-**M** e os neurônios da camada-**m**, são estabelecidas como segue:

$$W(M^t_{N-k+1}, m^i_{N-k}) = \begin{cases} 1 & \text{if } q(T(m^i_k)) = q(C(m^i_k)) = q(R(M^t_{N-k+1})) \\ 0 & \text{caso contrário} \end{cases} \quad (3.31)$$

onde $W(M^t_{N-k+1}, m^i_{N-k}) = 1$ denota uma conexão entre o t-ésimo neurônio-M da camada-**M** e o i-ésimo neurônio-m da camada-**m**.

3.5.2 Dinâmica da FDPNN - Um algoritmo

Sejam

- U_k e X_k os conjuntos como definidos na seção anterior;
- $a_i(.)$ as entradas para os neurônios-M da camada-M;
- N o número de estágios;
- s a saída de um neurônio na rede,
- $\mu_k = \mu_k(u_k^i)$ a função de pertinência correspondente a restrição do estágio $N-k; k = 1, ..., N-1$,
- $\mu_N = \mu_N(x_N^i)$ a função de pertinência correspondente a meta (estágio N),
- f_{N-k} a função de transição de estado para o estágio $N-k$; $k = 1, ..., N$,

Passo: Inicialização da Rede

Para o i-ésimo neurônio-M na camada-**M faça**

Início

leia$(q(R(M_N^i))) = x_N^i \in X_0$

$s(M_N^i) = \mu_{G^N}(q(R(M_N^i)))$

Fim

Passo - Inicialização do estágio $N-k$

Para cada i-ésimo neurônio-m na camada-**m faça**

leia$(q(R(m_{N-k}^i))) = u_{N-k}^i \in U_k$

Para cada neurônio-M na camada-**M faça**

Início

leia$(q(R1(M_{N-k}^i))) = x_{N-k}^i \in X_k$

leia$(q(R2(M_{N-k}^i))) = x_{N-k-1}^i \in X_{k-1}$

$q(T1(M_{N-k}^i)) = q(R1(M_{N-k}^i))$

$q(T2(M_{N-k}^i)) = q(R2(M_{N-k}^i))$

Fim

Passo Conexões

Criando conexões entre a camada-m e camada-M

Para o j-ésimo neurônio-M **faça**

Para o i-ésimo neurônio-m **faça**

Início

ativar $q(C(m^i_{N-k})) = f_{N-k}(q(T(M^j_{N-k})), q(R(m^i_{N-k})))$

$q(T(m^i_{N-k})) = q(C(m^i_{N-k}))$

$W(m^i_{N-k}, M^j_{N-k}) = 1$

Fim

Criando conexões *feedback* **entre a camada-M e camada-m**

Para o t-ésimo neurônio-M no estágio anterior **faça**

Se $q(R2(M^t_{N-k+1})) = q(T(m^i_{N-k}))$ **então**

Receber o sinal de saída do t-ésimo neurônio-M.

$s(M^t_{N-k+1})$ obtido no estágio anterior

Calcular o sinal de saída: $s(m^i_{N-k}) = \min\{\mu(q(R(m^i_{N-k}))), s(M^t_{N-k+1}))\}$

Calcular $a_i = a_i(M^j_{N-k}) = s(m^i_{N-k})$

Passo Saída

Para o t-ésimo neurônio-M da camada-M **faça**

Calcular $s(M^t_{N-k}) = \max_i\{a_i\}$

Algoritmo Principal

Início

Executar o Passo: Inicialização da Rede

Para $k = 1, 2, ..., N$ **faça**

Início

Executar o Passo Inicialização do estágio $N - k$

Executar o Passo Conexões

Executar o Passo Saída

Fim

Fim

A seguir, apresentamos uma equivalência da rede proposta FDPNN com o algoritmo relativo à estensão da PD no sentido de Bellman e Zadeh.

3.5.3 Teorema de Equivalência

A solução fornecida pela FDPNN é equivalente à fornecida pela abordagem da PD.

Prova:

Assumiremos, sem perda de generalidade, um esquema *backward* para resolução de problemas de PD e que $f_i = f, i = 1, 2, ..., N$ nos estágios $N - i$.

Lembremos que, para a construção da FDPNN as seguintes correspondências foram estabelecidas:

- Para cada variável de decisão $u_{N-(k-1)}$ e variável de estado x_{N-k} existe um correspondente neurônio-m e neurônio-M, respectivamente, no estágio $N - k$.
- Os valores das variáveis discretas u e x correspondem as quantidades dos receptores e transmissores dos neurônio-m e neurônio-M, respectivamente, i.é.,

$$q(R(m_k^i)) = u_k^i$$

e

$$q(T(M_l^i)) = x_l^i$$

para todo i-ésimo neurônio-m e l-ésimo neurônio-M, respectivamente.

A prova será feita por indução.

a) Inicialmente, mostraremos que a ativação da camada-**m** e camada-**M** corresponde aos cálculos realizados no estágio $N-1$ em (3.13), ou seja:

$$\mu_{G^{N-1}}(x^{N-1}) = max_{u^{N-1}}\{\mu(u^{N-1}) \wedge \mu_{G^N}(x^N)\}$$
$$= max_{u^{N-1}}\{\mu(u^{N-1}) \wedge \mu_{G^N}(f(u^{N-1}, x^{N-1}))\}$$

é calculado.

De acordo com o algoritmo, o Passo Inicialização da Rede deve ser inicialmente executado. Nesse passo, os valores de $\mu_{G^N}(x^N)$ são introduzidos à rede, através da execução do comando: $s(M_N^i) = \mu_{G^N}(q(R(M_N^i)))$.

Em seguida, $k = 1$, e os neurônios recebem os valores das variáveis correspondentes ao estágio $N-1$, de acordo com a execução do Passo Inicialização do Estágio $N-k$. Além disso, a função controladora para cada j-ésimo neurônio-M fixado, e para todo i-ésimo neurônio-m,

$$q(C(m_{N-1}^i)) = f(q(T(M_{N-1}^j)), q(R(m_{N-1}^i)))$$

é ativada e as conexões $W'(m_{N-1}^i, M_{N-1}^j) = 1$ são estabelecidas se f existir. (Em geral, f é dada na forma de tabela).

Isto equivale, na abordagem da PD a considerar apenas combinações das variáveis u e x para as quais f existe.

A seguir, a quantidade $q(T(m_{N-1}^i) = q(C(m_{N-1}^i))$ para todo i-ésimo neurônio-m na camada-**m** e as conexões *feedback* entre a camada-**M** e camada-**m** são estabelecidas em função do valor de $q(C(m_{N-1}^i))$ obtido, do seguinte modo:

se o t-ésimo neurônio-M no estágio anterior N é tal que

$$q(R(M_N^t)) = q(T(m_{N-1}^i))$$

então

$$W'(m_{N-1}^i, M_N^t) = 1.$$

Com as conexões estabelecidas, a saída do neurônio-m, m_{N-1}^i.

$$s(m_{N-1}^i) = min\{\mu(q(R(m_{N-1}^i))), s(M_N^t)\} \tag{3.32}$$

e $a_i(M^j_{N-1}) = s(m^i_{N-1})$ são calculadas.

Da definição de $q(R(m^i_{N-1}))$ e $s(M^i_N)$ e lembrando que isto é válido para todo i-ésimo neurônio-m, na camada-**m**, concluímos que o resultado dado por (3.32) é equivalente a:

$$\mu(u^{N-1}) \wedge \mu_{G^N}(f(x^{N-1}, u^{N-1})).$$

Seguindo o Algoritmo Principal, o Passo Saída é agora realizado. Neste caso, lembrando que cada j-ésimo neurônio-M, por definição é um neurônio Max, temos:

$$s(M^j_{N-1}) = \max_i\{a_i(M^j_{N-1})\} = \max_i\{s(m^i_{N-1})\}$$

que é equivalente a

$$max\{\mu(u^{N-1}) \wedge \mu_{G^N}(f(x^{N-1}, u^{N-1}))\} = \mu_{G^{N-1}}.$$

Portanto, o estágio $N-1$ é executado na primeira ativação das camadas:**m** e **M**.

b) Vamos supor agora, que na iteração $(k-1)$, a ativação da camada-**m** e camada-**M**, executam o estágio $N-(k-1)$. Devemos mostrar que na próxima iteração k, os cálculos correspondentes ao estágio $N-k$ são realizados corretamente, pela FDPNN. Em outras palavras, deve-se mostrar que:

$$\mu_{G^{N-k}}(x^{N-k}) = \max_{u^{N-k}}\{\mu(u^{N-k}) \wedge \mu_{G^{N-(k-1)}}(x^{N-(k-1)})\} \tag{3.33}$$

é calculado pela FDPNN proposta.

Da hipótese de indução, temos que:

$$\mu_{G^{N-(k-1)}}(x^{N-(k-1)})$$

já está calculado. Seguiremos o Algoritmo Principal para mostrar que os cálculos restantes em (3.33) são também computados.

De fato, de acordo com o Algoritmo Principal, o Passo Conexões deve ser executado primeiro. Neste passo, as seguintes conexões são estabelecidas:

- Conexões entre a camada-**m** e camada-**M**,

- Conexões *feedback* entre a camada-**M** e camada-**m**.

No primeiro caso, a função controladora dada por

$$q(C(m^i_{N-k})) \;=\; f(q(T(M^j_{N-k})), q(R(m^i_{N-k})))$$

é ativada para todo i-ésimo neurônio-m, na camada-**m** e as conexões $W(m^i_{N-k}, M^j_{N-k}) = 1$ são estabelecidas se f existir no ponto desejado.

Em tais casos, as conexões *feedback* entre a camada-**M** e camada-**m** são determinadas, com o objetivo de garantir que os valores de $\mu_{G^{N-(k-1)}}(x^{N-(k-1)})$ obtidos no estágio anterior, são também considerados neste estágio, i.é., o t-ésimo neurônio-M da camada-**M** é ativado, porque ele é o neurônio que satisfaz o requisito

$$q(R2(M^t_{N-k})) \;=\; q(T(m^i_{N-k})) \;=\; q(C(m^i_{N-k})).$$

Assim sendo, o sinal de saída $s(M^t_{N-(k-1)}) = \mu_{G^{N-(k-1)}}(x_t^{N-(k-1)})$ é recebido pelo neurônio-m, m^i_{N-k} e

$$s(m^i_{N-k}) \;=\; min\{\mu(q(R(m^i_{N-k}))), s(M^t_{N-(k-1)})\}$$

é calculada para todo i-ésimo neurônio-m, na camada-**m**. Portanto,

$$s(m^i_{N-k}) = \mu(u_i^{N-k}) \wedge \mu_{G^{N-k+1}}(x_t^{N-k+1}).$$

A seguir, o Passo Saída é executado. Neste caso, a saída de cada t-ésimo neurônio-M é calculada. Supondo que cada t-ésimo neurônio-M recebe n entradas $(a_1, a_2, ..., a_n)$ onde $a_l = s(m^i_{N-k})$ para $l \leq n$ e algum i tal que existe uma conexão do i-ésimo neurônio-m, m^i_{N-k}, para o t-ésimo neurônio-M, M^t_{N-k}, tem-se:

$$s(M^t_{N-k}) \;= max\{a_1, a_2, ..., a_n\} \;=$$

$$\max_i\{\mu(u_i^{N-k}) \wedge \mu_{G^{N-k+1}}(x_t^{N-k+1})\}$$

Uma vez que isto é válido para todo t-ésimo neurônio-M na camada-**M**, concluímos que os cálculos desejados para o estágio $N-k$

$$\mu_{G^{N-k}}(x^{N-k}) = max\{\mu(u^{N-k}) \wedge \mu_{G^{N-k+1}}(x^{N-k+1})\}$$

são executados pela FDPNN e assim o teorema está provado.

3.5.4 Exemplo Ilustrativo

Como uma ilustração, consideremos o mesmo problema de otimização colocado na seção 3.2.2 proposto por Bellman e Zadeh [BEL2 70], onde o sistema de controle tem 03 estados: x_1, x_2, x_3, duas entradas u_1 e u_2 e o número de estágios é $N = 2$.

Seja a meta *fuzzy* no tempo $t = 2$ definida pela função de pertinência μ_2 cujos valores são:

$$\mu_2(x_1) = 0.3 \; ; \quad \mu_2(x_2) = 1 \; ; \quad \mu_2(x_3) = 0.8.$$

Sejam as restrições *fuzzy* em $t = 1$ e $t = 0$, respectivamente, dadas por:

$$\mu_1(u_1) = 1 \; ; \quad \mu_1(u_2) = 0.6; \mu_0(u_1) = 0.7 \; ; \quad \mu_0(u_2) = 1.$$

Vamos supor que a tabela de transição de estados que define a função f seja dada através da tabela 3.1.

	x_t		
u_t	x_1	x_2	x_3
u_1	x_1	x_3	x_1
u_2	x_2	x_1	x_3

Tabela 3.1: Tabela de Transição

Seguiremos o Algoritmo Principal para a rede proposta afim de ilustrar como a FDPNN é construída. A rede neural FDPNN que resolve este problema pode ser vista na Figura 3.20, considerando dois neurônios na Camada-m e três neurônios na Camada-M. Os passos necessários para obtenção da solução são apresentados na Figura 3.22.

Definiremos:

$U_1 = \{u_1^i\} = \{u_1^0, u_1^1\} = \{u_1, u_2\}$.

$U_0 = \{u_0^i\} = \{u_0^0, u_0^1\} = U_1$.

$f_1 = f_2 = f$: função de transição de estados no estágio k, $k = 1, \ldots, N$.

$X_2 = \{x_2^i\} = \{x_2^0, x_2^1, x_2^2\} = \{x_1, x_2, x_3\}$.

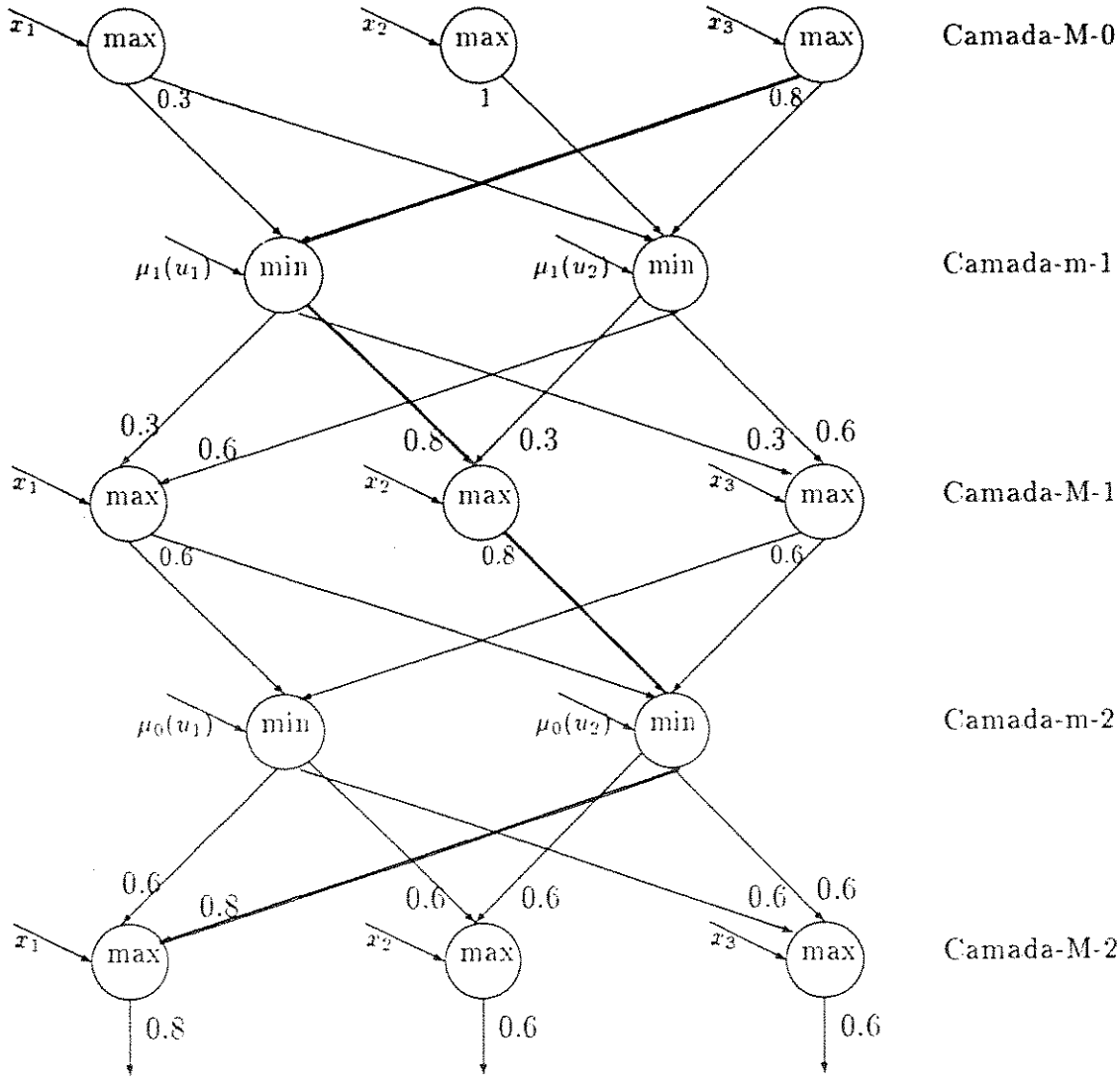


Figura 3.22: FDPNN para o Exemplo Ilustrativo

$$X_1 = \{x_1^i\} = \{x_1^0, x_1^1, x_1^2\} = X_2$$

$$X_0 = \{x_0^i\} = \{x_0^0, x_0^1, x_0^2\} = X_2,$$

$$\mu_2 = \{0.3, 1, 0.8\},$$

$$\mu_1 = \{1, 0.6\},$$

$$\mu_0 = \{0.7, 1\},$$

$$N = 2\,.$$

Durante o Passo de Inicialização da Rede, as saídas dos neurônios da camada-M são inicializadas:

$q(R2(M_2^i)) = x_2^i \in X_2; \quad i = 0,1,2; \quad s(M_2^i) = \mu(q(R(M_2^i)) = \mu_2^i, \quad k = 1$ e o Passo Inicialização do Estágio $N-1$ é executado. Então.

$$q(R(m_1^i)) = u_1^i \in U_1 \quad i = 0,1$$

$$q(R1(M_1^i)) = x_1^i \in X_1 \quad i = 0,1,2$$

$$q(R2(M_1^i)) = x_2^i \in X_2 \quad i = 0,1,2.$$

Em seguida, o Passo Conexões deve ser realizado. Neste passo, as conexões *feedback* entre a camada-m e camada-M são criadas. A função controladora dos neurônios-m, $C(m_1^i)$, $i = 0,1$ é ativada para cada neurônio-M, M_1^j e para todo neurônio-m m_1^i. Dependendo do valor $C(m^i)$, $i = 0,1$ as conexões são estabelecidas.

Por exemplo, consideremos o neurônio-M, M_1^2, e o neurônio-m, m_1^1. A quantidade $C(m_1^1)$ é dada por:

$$q(C(m_1^1)) = f(q(R(m_1^1)).q(T(M_1^2))) = f(u_2, x_3) = x_3$$

o que implica que a conexão *feedback* $W(M_2^2, m_1^1) = 1$ é ativada, porque o neurônio M_2^2 é um neurônio da camada-M que satisfaz

$$q(R2(M_2^2)) = q(T(m_1^1)) = q(C(m_1^1))$$

Então, o neurônio m_1^1 recebe a saída s do neurônio-M, $s(M_2^2) = 0.8$ e sua saída é dada por:

$$s(m_1^1) = \mu_1(u_1^1) \wedge s(M_0^3) = \mu_1(u_2) \wedge s(M_0^3) = 0.6 \wedge 0.8 = 0.6.$$

Seguindo o Algoritmo Principal, o Passo Saída é executado onde as saídas dos neurônios-M são calculadas. Considere o neurônio-M, M_1^2. Temos que sua saída $s(M_1^2)$ é dada por:

$$s(M_1^2) = \max\{a_1, a_2\} = \max\{0.3, 0.6\} = 0.6.$$

As saídas dos neurônios-M restantes, M_1^0, M_1^1 são calculadas de forma análoga.

Novamente, o valor da iteração k é atualizado, $k = 2$ e o Passo Inicialização do Estágio $N - 2$ é executado. Assim, os seguintes valores são atualizados:

$q(R(m_0^i)) = u_0^i \in U_0 \quad i = 0, 1$

$q(R1(M_0^0) = x_0^i \in X_0 \quad i = 0, 1, 2$

$q(R2(M_0^0) = x_1^i \in X_1 \quad i = 0, 1, 2$

Durante a execução do próximo passo, Passo Conexões, a função controladora de cada neurônio-m, m_0^i, $i = 0, 1$, na camada-m é ativada e dependendo do seu valor as conexões entre as camadas: camada-m e camada-M são estabelecidas.

Por exemplo, para $j = 0$ e $i = 0, 1$ as seguintes conexões *feedback* são ativadas:

$$W(M_1^0, m_0^0) = 1$$

e

$$W(M_1^1, m_0^1) = 1$$

porque

$$q(C(m_0^0)) = f(q(R(m_0^0), q(T1(M_0^0))) = f(u_1, x_1) = x_1 = q(T(m_0^0)) = q(R2(M_1^0))$$

e

$$q(C(m_0^1)) = q(T(m_0^1) = q(R2(M_1^1)) = x_2.$$

O neurônio-m, m_0^0, recebe a saída s do neurônio-M, M_1^0, e a sua saída é calculada:

$$s(m_2^0) = \mu_0(u_1^0) \wedge s(M_1^0) = \mu_0(u_1) \wedge s(M_1^0) = 0.7 \wedge 0.6 = 0.6.$$

Analogamente,

$$s(m_0^1) = \mu_0(u_1^1) \wedge s(M_1^1) = \mu_0(u_2) \wedge s(M_1^1) = 1.0 \wedge 0.8 = 0.8.$$

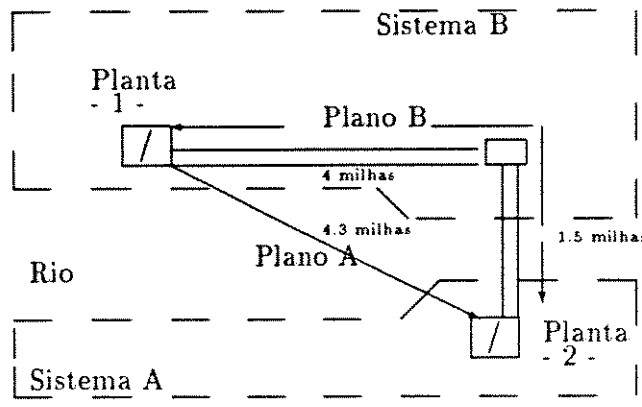


Figura 3.23: Planos de Interconexão

O Passo Saída é executado a seguir. As saídas dos neurônios-M são calculadas como segue:

$$s(M_0^0) = \max\{a_1, a_2\} = \max\{0.6, 0.8\} = 0.8$$

$$s(M_0^1) = \max\{a_1, a_2\} = \max\{0.6, 0.6\} = 0.6$$

$$s(M_0^2) = \max\{a_1, a_2\} = \max\{0.6, 0.6\} = 0.6.$$

A solução ótima pode ser recuperada através das conexões em negrito [Figura 3.22]:

$$u_2 \ , \ u_1$$

e o valor correspondente de μ_2 é 0.8.

3.5.5 Aplicação

Análises de decisões são muito importantes no planejamento de sistemas de potência a longo prazo, pelo fato de envolverem muitos recursos financeiros. Na maioria das decisões no planejamento de sistemas a longo prazo, tanto os objetivos, quanto as restrições e as consequências de possíveis decisões não são precisamente conhecidas.

Um estudo da viabilidade de um projeto de interconexão entre dois sistemas de potência é considerado em [DHAR 79]. A capacidade de transferência de potência é assumida em torno de 2500-3000 MW. As considerações principais na escolha entre HVDC ou HVAC

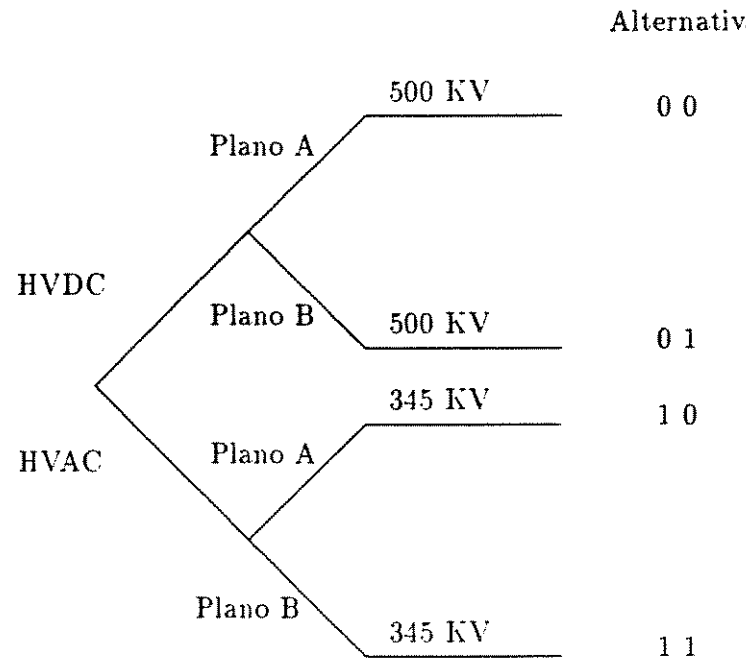


Figura 3.24: Alternativas Viáveis de Interconexões

para uma interconexão são custo, disponibilidades de cabos, benefícios e implicações nos dois sistemas, provocados pela instalação da interconexão proposta.

A Figura 3.23 ilustra dois possíveis planos para as interconexões entre o Sistemas A e Sistema B. As Planta 1 and 2 indicam os lugares escolhidos para geração de potência, para os Sistemas A e B, respectivamente. O projeto de interconexão proposto é válido por até 15 ou 20 anos.

As diferentes alternativas (factíveis) para as interconexões são apresentadas na Figura 3.24.

Os fatores que atuam sobre todas as alternativas, que serão levados em conta nessa aplicação para a análise e seleção da alternativa ótima, são custos de investimento de capital, custos operacionais anuais e segurança.

O sistema de potência, pode estar em um dos 11 estados, cujos graus de pertinência são dados na Tabela 3.2. Os estados do sistema representam todas as condições possíveis do sistema, que podem ter um impacto positivo ou negativo no projeto proposto.

As tabelas de transição de estados que definem as funções f_2, f_1, f_0 relativamente aos fatores citados acima, são mostradas na Tabela 3.3, Tabela 3.4 e Tabela 3.5, respectiva-

mente.

Para construção da FDPNN correspondente, este problema de decisão pode ser formulado como segue. Sejam:

- A = $\{a_1, a_2, a_3, a_4\}$ o conjunto de alternativas,

- A meta *fuzzy* no tempo $t = 3$, o conjunto *fuzzy* cuja função de pertinência é dada pela Tabela 3.2.

- Os diferentes fatores, custos de investimento de capital, custos operacionais anuais e segurança, são as restrições *fuzzy* no tempo $t = 2$, $t = 1$, $t = 0$, cujas funções de pertinência são dadas por:

$$\mu_2(a_1) = 0.6,\ \mu_2(a_2) = 0.6,\ \mu_2(a_3) = 0.79,\ \mu_2(a_4) = 0.89$$

$$\mu_1(a_1) = 0.5,\ \mu_1(a_2) = 0.5,\ \mu_1(a_3) = 0.84,\ \mu_1(a_4) = 0.84$$

$$\mu_0(a_1) = 0.7,\ \mu_0(a_2) = 0.6,\ \mu_0(a_3) = 0.88,\ \mu_0(a_4) = 0.8$$

respectivamente.

A FDPNN é uma rede do tipo da apresentada na Figura 3.20.

No entanto, a Figura 3.25 apresenta duas das cinco soluções possíveis para este problema, indicadas pelas conexões vistas em negrito nessa Figura. Uma das soluções é dada pela sequência de alternativas:

$$a_3\ ,\ a_4\ ,\ a_3$$

enquanto que a outra é dada por:

$$a_4\ ,\ a_4\ ,\ a_3$$

e o valor correspondente de μ_3 é 0.8.

3.5.6 Conclusões

Neste capítulo, uma rede neural DPNN para resolver problemas de otimização discreta foi proposta. O algoritmo para implementação da DPNN e a equivalência entre a DPNN e o procedimento de PD foram apresentados. A DPNN foi testada para algumas aplicações importantes que constam na literatura.

Essa abordagem apresenta algumas vantagens devido ao paralelismo inerente às redes neurais. Os cálculos das funções controladoras em cada neurônio-X, que correspondem aos cálculos das funções transformações na abordagem da PD, são realizados em paralelo,

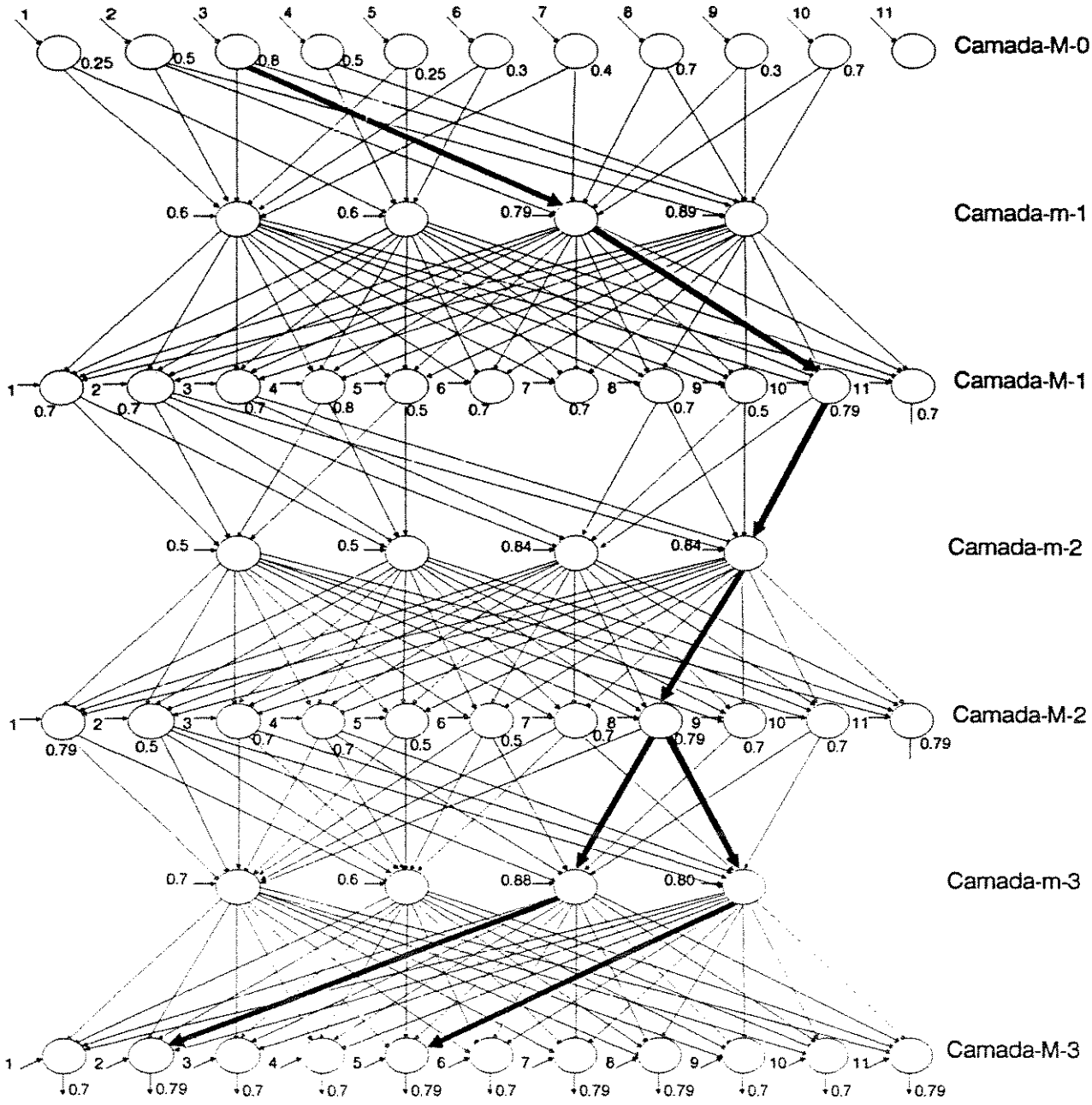


Figura 3.25: FDPNN para o Problema de Planejamento de Interconexão

enquanto que na abordagem de PD, os valores são calculados na forma sequencial e armazenados numa matriz.

Além disso, o cálculo da saída dos neurônios-Y, que é a determinação do valor

$$F_j = max\ f_j \qquad j = 1, 2, ..., N$$

necessário em cada estágio, também é realizado em paralelo. Isto implica numa diminuição no tempo de processamento.

Contudo, a abordagem proposta tal como ela foi apresentada até o momento, só pode ser aplicada a funções separáveis. Uma generalização dos neurônios utilizados na DPNN será proposta no próximo capítulo a partir da qual problemas não-separáveis poderão também ser resolvidos pela DPNN.

Uma rede neural FDPNN para resolver problemas de otimização *fuzzy* também foi desenvolvida. Um algoritmo para a implementação da FDPNN foi proposto e a equivalência entre a FDPNN e a abordagem da PD estabelecida. Além disso, a FDPNN foi testada para algumas aplicações fornecendo resultados satisfatórios.

Novamente, a saída de cada neurônio-M que corresponde à determinação do valor

$$F_j = max\ f_j \qquad j = 1, 2, ..., N$$

necessário em cada estágio, é executada em paralelo, enquanto que na abordagem da PD convencional seria calculada sequencialmente.

Na maioria das redes neurais desenvolvidas na área de otimização, as interconexões são designadas, a priori, e definidas na fase de inicialização da rede. Na abordagem proposta nesta seção, as interconexões *feedback*, da camada-M para a camada-m, são designadas pela própria rede, uma vez fornecidos os valores das variáveis discretas e funções de transição de estados.

Estados	1	2	3	4	5	6	7	8	9	10	11
Graus	0.25	0.50	0.80	0.50	0.25	0.30	0.40	0.70	0.30	0.70	0.30

Tabela 3.2: Estados do sistema p/ o ano horizonte

	Estados do Sistema										
Alternativas	1	2	3	4	5	6	7	8	9	10	11
1	5	4	2	1	1	2	3	6	1	1	7
2	5	4	2	1	1	2	3	6	1	1	7
3	9	8	7	2	1	9	8	10	2	3	10
4	10	9	8	3	2	10	9	10	2	4	10

Tabela 3.3: Custos de Investimento Capital

	Estados do Sistema										
Alternativas	1	2	3	4	5	6	7	8	9	10	11
1	5	4	2	1	1	4	2	5	1	1	5
2	5	4	2	1	1	4	2	5	1	1	5
3	10	9	3	2	1	9	8	10	2	3	10
4	10	9	3	2	1	9	8	10	2	3	10

Tabela 3.4: Custos Operacionais Anuais

	Estados do Sistema										
Alternativas	1	2	3	4	5	6	7	8	9	10	11
1	4	3	2	1	6	4	5	7	1	1	8
2	2	2	2	1	5	3	4	6	1	1	7
3	10	8	6	4	10	9	9	10	2	2	10
4	8	7	4	3	8	7	8	8	2	2	8

Tabela 3.5: Segurança

Capítulo 4

Problemas de Controle Ótimo

4.1 Introdução

Neste capítulo, uma generalização de modelos de neurônios recorrentes é proposta, da qual os neurônios Max e Min são casos particulares. A partir desta generalização, problemas de otimização dinâmica discreta não aditivamente separáveis podem também ser resolvidos. Entre esses, problemas de controle ótimo multi-estágios são focalizados. Uma rede neural para resolver tais problemas é proposta com um método direto para designar seus pesos e incorporar conhecimento sobre o sistema considerado. Diferentes abordagens sobre como incorporar redes neurais em estratégias de controle ótimo foram propostas usando algoritmos do tipo gradiente e *back-propagation*. Uma das vantagens da abordagem proposta é a ausência de treinamento a partir de exemplos. Aplicações são apresentadas para ilustrar o comportamento da rede proposta.

4.2 Sistema de Controle Ótimo e PD

Estamos aqui interessados no problema de se construir um elemento computacional inteligente (um controlador, um *decision maker*) que em N estágios conduz o estado de um sistema dinâmico, de um ponto inicial para algum ponto final, com o objetivo de minimizar (ou maximizar) uma determinada função objetivo.

Suponhamos que o sistema seja dado pela seguinte equação às diferenças não-linear:

$$x(k+1) = f(x(k), u(k)) \quad , \quad k = 0, 1, ..., N-1, \quad x(k) \in R^n, \quad u(k) \in R^m \tag{4.1}$$

Desejamos determinar a lei de controle que minimiza a função objetivo descrita na forma geral:

$$J = h(x(N)) + \sum_{k=0}^{N-1} g(x(k), u(k)) \tag{4.2}$$

onde N indica o estágio final.

A equação às diferenças e o critério de desempenho dados por (4.1) e por (4.2), respectivamente, podem ser uma aproximação discreta para um sistema contínuo, ou podem representar um sistema que é realmente discreto. Consideraremos sistemas do tipo discreto para a obtenção da rede proposta neste capítulo.

Kirk [KIRK 70] mostrou que a aplicação da Programação Dinâmica (PD) para este problema, adotando um procedimento *backward*, conduz à seguinte equação de recorrência:

$$J^*_{N-k,N}(x(N-k)) = \min_{u(N-k)} \{g(x(N-k), u(N-k)) \tag{4.3}$$

$$+ \; J^*_{N-(k-1),N}(f(x(N-k), u(N-k))\} \quad , \quad k = 1, 2, ..., N \tag{4.4}$$

com $J^*(x(N)) = h(x(N))$.

A solução desta equação de recorrência é uma lei de controle ótimo ou uma política ótima,

$$u^*(x(N-k), N-k), \quad k = 1, 2, ..., N,$$

que é obtida através da análise de todos os valores possíveis da variável de controle, para cada estado.

A fim de transformar o processo num procedimento computacional, é necessário quantizar tanto as variáveis de estado, como as variáveis de controle, em um número finito de níveis. Por exemplo, se o sistema é de segunda ordem a Figura 4.1 mostra os valores possíveis para a variável de estado.

Neste exemplo, o número total de valores para a variável de estado em cada estágio k é $s_1 * s_2$, onde s_1 é o número de pontos na direção x_1 e s_2 é o número de pontos na direção x_2. s_1 e s_2 são determinados pela relação

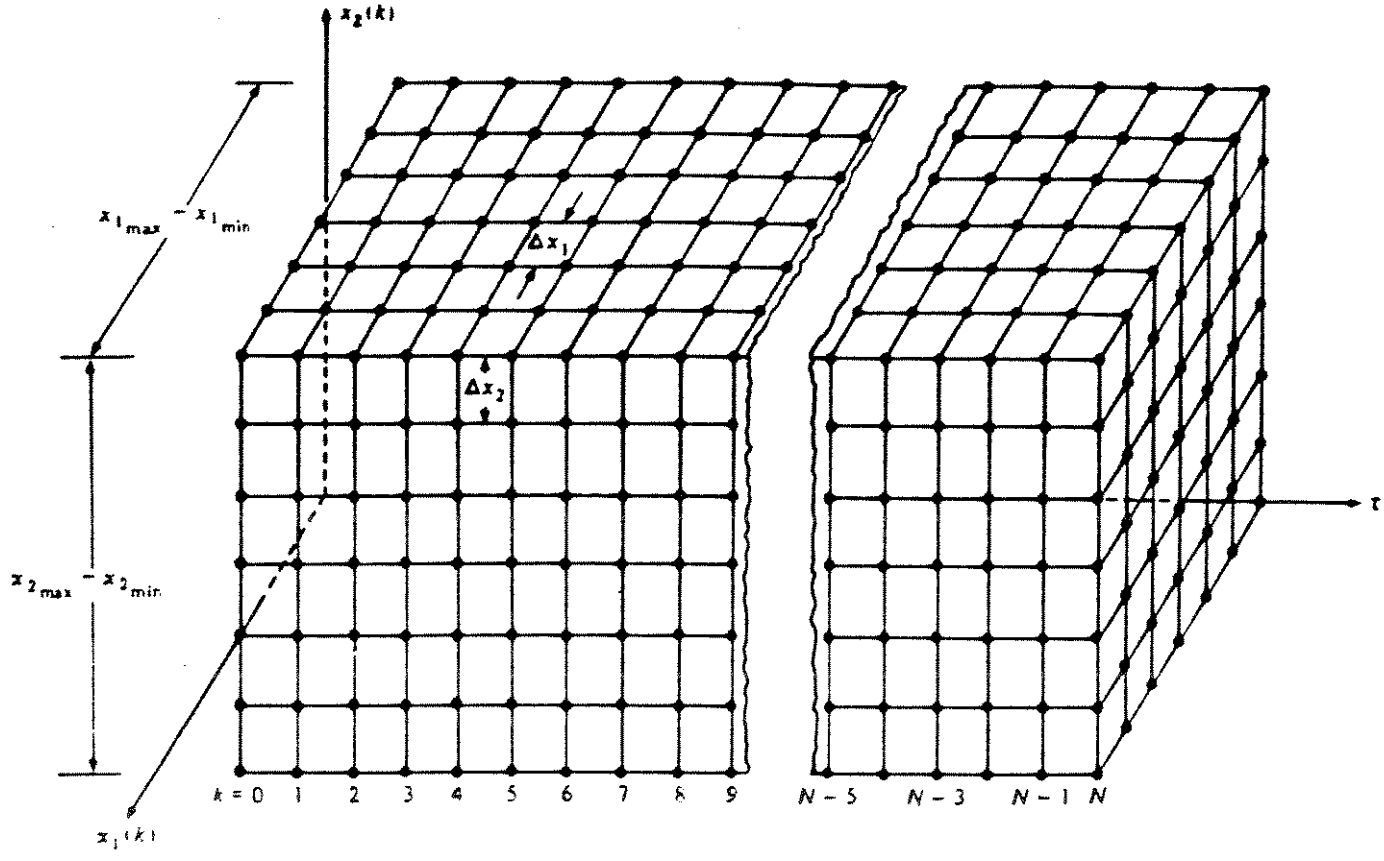


Figura 4.1: Valores quantizados do estado de um sistema de segunda ordem

$$s_i = \frac{x_{imax} - x_{imin}}{\Delta x_i} + 1 \; ; \; i = 1, 2$$

onde Δx_i é escolhido de modo que o intervalo $[x_{imax} - x_{imin}]$ tenha um número inteiro de pontos.

Para um sistema de ordem n, o número total de valores para a variável de estado em cada estágio k é:

$$S = s_1 s_2 .... s_n$$

onde

$$s_i = \frac{x_{imax} - x_{imin}}{\Delta x_i} + 1 \; ; \; i = 1, 2, ... n$$

e a razão $\frac{x_{imax} - x_{imin}}{\Delta x_i}$ é um número inteiro. Os valores da variável de controle são quantizados analogamente; se C é o número total de valores quantizados de $u(k)$, então

$$C = c_1 c_2 ... c_m$$

onde

$$c_i = \frac{u_{imax} - u_{imin}}{\Delta u_i} + 1 \; ; \; i = 1, 2, ... m.$$

No desenvolvimento que se segue, $x^{(i)}(k)(i = 1, 2, ..., S)$ e $u^{(j)}(k)(j = 1, 2, ..., C)$ denotarão os valores quantizados para as variáveis de estado e de controle no estágio k, respectivamente.

4.3 Redes Neurais e Sistemas de Controle

A necessidade de atender a crescente demanda de requisitos de controle, em função do aumento da complexidade de sistemas dinâmicos, tornam Redes Neurais Artificiais (RNA) atraentes, uma vez que possuem capacidade de aprendizado, aproximar funções, classificar padrões, além do potencial para implementações paralelas em *hardware* [ANTS 92].

Em muitas aplicações, RNA são usadas para otimizar determinadas funções objetivo, como por exemplo, Redes de Hopfield [HOPF 85,HOPF 84,HOPF 86] e Redes Multi-Camadas [RUME 86,NGUY 90,PSAL 88] sendo estas últimas, as mais comumente usadas.

Redes Multi-Camadas codificam um mapeamento entre entradas e as saídas, o qual aproxima, sob certas hipóteses, qualquer função dada [FUNA 89]. Entretanto, quando uma rede Multi-Camadas é treinada como **controlador**, a saída desejada não é conhecida. Nesse caso, ela é geralmente induzida através de aproximaçôes das saídas da planta conhecida. Essas aproximaçôes são obtidas baseando-se, ou no modelo matemático da planta ou num modelo de rede neural associado à dinâmica da planta ou associado à dinâmica inversa da planta [ANTS 92].

Diferentes abordagens para treinamento de um controlador foram propostas na literatura. Elas incluem aprendizado com reforço (*reinforcement learning*) [WIDR 83], [BART 83] , [ANDE 89], modelo inverso [WIDR 86,PSAL 88,TEIX 92] e controle ótimo [PSAL 88,NGUY 90] e [FONG 91].

Problemas de controle podem ser divididos em duas grandes classes. A primeira, envolve problemas de regulação e *tracking*, nos quais o objetivo é seguir uma dada trajetória de referência. A segunda, diz respeito a problemas de controle ótimo, nos quais o objetivo é minimizar ou maximizar um funcional que descreve o desempenho do sistema de controle. Quando não se dispõe de um modelo preciso e detalhado do sistema a ser controlado, métodos adaptativos podem ser aplicados. Métodos adaptativos para problemas pertencentes a primeira classe são bem conhecidos, enquanto métodos adaptativos para problemas de controle ótimo têm recebido menor atenção [SUTT 92].

Tanto para problemas de *tracking* quanto para problemas de controle ótimo, métodos adaptativos podem ser separados em métodos diretos e métodos indiretos.

Um método indireto consiste em uma fase de identificação do sistema, seguida da fase de determinação da estratégia de controle. Um exemplo de método indireto pode ser encontrado em [NGUY 90], onde uma rede Multi-Camadas é usada tanto como um emulador para identificar as características do sistema, quanto um controlador para aprender a controlar o emulador. A maioria dos métodos propostos para controle ótimo são métodos indiretos, nos quais os controles são recalculados a partir de um modelo aproximado do sistema, a cada passo. Este procedimento é complexo tornando os métodos diretos mais atrativos [SUTT 92].

Um método direto determina apenas a estratégia de controle a partir do modelo. Em [PSAL 88] encontra-se um método direto, que é uma modificação do *back-propagation* baseado na propagação do erro ocorrido na saída através da planta dada. Podemos citar também, os algoritmos numéricos propostos por [FONG 91] e o esquema proposto por [PARIS 91], no qual uma ampla classe de problemas de controle ótimo não-lineares e não-quadráticos podem ser resolvidos através da conexão de redes Multi-Camadas. Além destes, existem os métodos baseados em técnicas de aprendizado com reforço, que constituem uma área ativa de pesquisa em RNA e em Aprendizado de Máquina [SUTT 92].

Estes métodos fundamentam-se em estudos de aprendizado de animais e no trabalho de controle por aprendizado encontrado em [MEND 70]. Por serem vistos como uma síntese de PD e métodos de aproximação estocástica e por estarem relacionados de certa forma com a rede neural proposta neste capítulo, métodos de aprendizado com reforço, serão revistos a seguir de acordo com [SUTT 92].

Aprendizado com Reforço

Aprendizado com reforço é baseado na idéia que, se uma ação é seguida de estados satisfatórios, ou por uma melhoria no estado, então a tendência para produzir esta ação é aumentada, i.é., reforçada [SUTT 92]. Estendendo esta idéia, ações podem ser selecionadas em função da informação sobre os estados que elas podem produzir, o que introduz aspectos de controle com realimentação.

Resultados empíricos combinando aprendizado com reforço e redes neurais ou outras estruturas de memórias associativas mostraram a eficiência do aprendizado em diversos problemas não-lineares de controle, tais como, [BART 83], [GREF 90], [MILL 91], [WHIT], [ANDE 89] e [SART 92]. De acordo com [BART 91], estes métodos podem ser vistos como métodos para aproximar soluções de problemas de controle ótimo através da PD.

Dado um problema de decisão Markoviano, onde são conhecidas as probabilidades de transição de estados x_k para cada ação a_k e as probabilidades que especificam a utilidade, r_k, é possível encontrar uma estratégia ótima de controle através de métodos de PD. Se o modelo não é conhecido, ele pode ser estimado a partir dos valores observados de r_k e dos estados. A PD pode então ser aplicada ao modelo estimado, o que constitui um método indireto.

Métodos de aprendizado com reforço, tais como, o *Q-learning*, não estimam tal modelo. A idéia básica do *Q-learning*, proposto por Walkins [WATK 89] é estimar uma função real Q, dos estados e ações. $Q(x, a)$ é o valor da utilidade esperada, se a ação a for executada de forma ótima no estado x. A função satisfaz a seguinte relação de recorrência:

$$Q(x,a) = E\{r_k + \gamma \max_b Q(x_{k+1}, b) \ / x_k = x, a_k = a\}$$

A seguinte estratégia de controle pode ser expressa em termos de Q: um controle ótimo para o estado x é qualquer ação a que maximiza $Q(x, a)$.

O algoritmo Q-learning mantém uma estimativa $\hat{Q}$ da função Q. Na passagem do estágio k para $k+1$, um modelo de aprendizado pode observar x_k, a_k, r_k e x_{k+1}. Em função destas observações, $\hat{Q}$, é atualizado no passo $k+1$ como segue:

$\hat{Q}(x, a)$ permanece inalterado para todos os pares$(x, a) \neq (x_k, a_k)$

e

$$\hat{Q}(x_k, a_k) := \hat{Q}(x_k, a_k) + \beta_k[r_k + \gamma \, max_b \hat{Q}(x_{k+1}, b) - \hat{Q}(x_k, a_k)] \qquad (4.5)$$

onde β_k é uma sequência satisfazendo

$$0 < \beta_k < 1; \quad \sum_{k=1}^{\infty} \beta_k = \infty$$

e

$$\sum_{k=1}^{\infty} (\beta_k)^2 < \infty$$

Walkins [WATK 89] mostrou que $\hat{Q}$ converge com probabilidade um, para Q.

Da mesma forma para os métodos de PD, o algoritmo *Q-learning* dado por (4.5), requer memória e tempo de processamento proporcionais ao número de pares de estado/ação. Em problemas de grande porte ou em problemas com estados e ações contínuas, que devem ser discretizados, estes métodos tornam-se muito complexos. Uma abordagem para reduzir a complexidade deste problema é não representar $\hat{Q}$ na forma de uma tabela, mas como uma estrutura parametrizada, como por exemplo, árvores de decisão ou redes neurais.

Correção do Erro versus Aprendizado com Reforço

Este tipo de aprendizado difere do aprendizado supervisionado implementado pelo *Perceptron* [WIDR 60] e *Adaline* [ROSE 59] na classificação de padrões. As diferenças principais entre estes métodos podem ser encontradas em [BART 83].

Métodos com aprendizado supervisionado, também chamados de métodos da Correção do Erro, requerem um conjunto de treinamento constituído de pares de vetores de entrada e saída. Baseado no conhecimento da resposta correta, um sinal correspondente ao erro, dado pela diferença entre a resposta correta e a fornecida pela rede, é retornado ao elemento adaptativo: o elemento faz uso deste sinal para adaptar os valores de seus pesos. Esta sequência é repetida para todos os pares do conjunto de treinamento até que os sinais correspondentes aos erros aproximem-se de zero.

Uma diferença importante entre o método da Correção do Erro e o aprendizado com reforço implementado pelo elemento adaptativo *ASE* [BART 83], é que o último não se baseia exclusivamente nos seus pesos para determinar suas ações. Ele gera ações por um processo aleatório que é polarizado por uma combinação dos valores de seus pesos e de suas entradas. Assim, as ações não são vistas apenas como respostas aos padrões de entrada, mas dependem também, dos estados do sistema. Além disso, se a resposta desejada não é conhecida, a avaliação do desempenho da rede é obtida indiretamente considerando o **efeito** de sua saída no ambiente com o qual a rede interage. Aprendizado com reforço

pode ser aplicado quando este efeito é medido por mudanças num determinado **sinal** (ou *reinforcement* - um termo usado na teoria de aprendizado de animais [ANDE 89]).

Na próxima seção, uma rede neural que implementa um método direto para controle ótimo é proposta. Como o *Q-learning*, o algoritmo é baseado em PD e em função do paralelismo inerente das redes neurais é uma abordagem alternativa para reduzir a complexidade do problema mencionado anteriormente (tempo de processamento e memória). Enquanto o *Q-learning* é uma aproximação do algoritmo convencional da PD, o método direto proposto é equivalente a equação de recorrência da PD. Os valores de x_k, a_k, r_k e x_{k+1}, no caso de um problema de decisão finito, são observados pelo sistema de aprendizado. Na rede proposta, em cada passo $k = 1, 2, \ldots, N$, os espaços de estados e ações são quantizados e o próximo estado x_{k+1} é calculado não só em função da entrada x_k, mas também, do estado anterior da rede, através de uma **função controladora** baseada no modelo do sistema. O processo de programação que implementa o algoritmo equivalente ao da PD é uma alternativa relativa a outros esquemas de aprendizado, pois fundamenta-se na troca de informações que ocorrem entre os neurônios, durante o processamento químico sináptico [ROCH 92].

4.4 DOCNN - Rede Neural para Problemas de Controle Ótimo Discretos

4.4.1 Definição do Neurônio Generalizado

Uma generalização de modelos de neurônios recorrentes é proposta, da qual os neurônios Max e Min são casos particulares. O neurônio é um elemento adaptativo cujo processamento pode ser subdividido em processamento numérico e simbólico. Por processamento numérico entende-se o conjunto de operações aritméticas que o neurônio deve processar, enquanto que por processamento simbólico, o conjunto de operações lógicas processadas durante a troca de mensagens realizada pelos neurônios antes de uma ação.

O neurônio generalizado artificial [Figura 4.2] é um elemento computacional que:

- agrega suas n entradas a_i

$$v = \sum_{i=1}^{n} g_i(a_i) \tag{4.6}$$

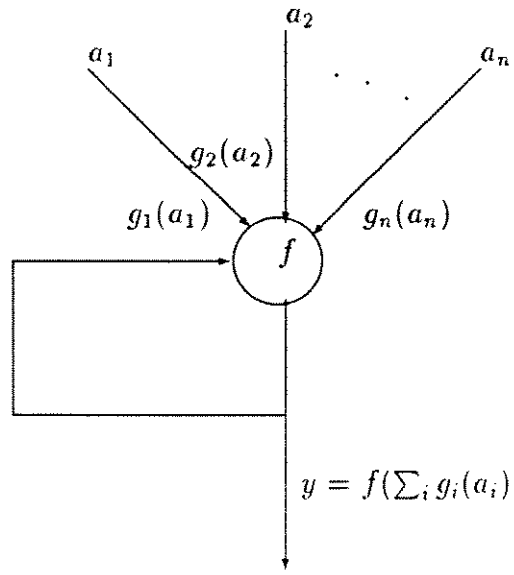


Figura 4.2: Neurônio Generalizado

de acordo com os pesos sinápticos g_i correspondentes às conexões entre os neurônios de entradas (pré-sinápticos) n_i ao neurônio de saída (pós-sináptico) n_k (neste caso, os pesos são funções g_i das entradas a_i).

- recodifica $v \in V$ na ativação a_p do neurônio pós-sináptico:

$$a_p = \begin{cases} f(v) & \text{se } v \geq \alpha \\ 0 & \text{caso contrário} \end{cases}$$

onde α é um limiar e f é uma função de transferência (ou de ativação).

A entrada $a = (a_1, a_2, \ldots, a_n)$ no neurônio n_k é transformada pela função peso G como segue:

$$(a_1, a_2, \ldots, a_n) \overset{G}{\mapsto} (g_1(a_1), g_2(a_2), \ldots, g_n(a_n))$$

onde

$$g_i : R \mapsto R$$

$$a_i \mapsto w_i = g_i(a_i)$$

Assim, a função peso G é uma função $G : R^n \mapsto R^n$ tal que:

$$a = (a_1, a_2, ..., a_n) \mapsto w = (w_1, w_2, ..., w_n) = G(a) = (g_1(a_1), g_2(a_2), ..., g_n(a_n)).$$

O neurônio generalizado pode também possuir uma sinapse recorrente. Esta sinapse recorrente é estabelecida da mesma forma que o neurônio recorrente anterior [Figura 3.4], ou seja, se o axônio do neurônio generalizado faz contato com dendritos ou com o seu próprio corpo celular. Se a sinapse recorrente é localizada perto do axônio, então ele pode controlar o limiar axônico como uma função da sua própria atividade.

Baseados nesta definição, os seguintes neurônios podem ser obtidos:

- Se

$$g_i(a_i) = c_i a_i \text{ onde } c_i \text{ é uma constante} \tag{4.7}$$

e

$$f(v) = \begin{cases} 1 & \text{para todo } v \geq \alpha \\ 0 & \text{caso contrário} \end{cases}$$

então o neurônio é um **neurônio binário**, ou neurônio de McCulloch e Pitts.

- Se a condição dada por (4.7) é válida e

$$f : V \mapsto [0, 1]$$

com

$$a_p = \begin{cases} 1 & \text{se } v \geq \alpha_2 \\ f(v) & \text{se } \alpha_1 \leq v \leq \alpha_2 \\ 0 & \text{caso contrário} \end{cases} \tag{4.8}$$

onde α_1 e α_2 são limiares axônicos (fornecidos em muitas aplicações, por um tipo de neurônio especial chamado *bias cell*), o neurônio obtido é o chamado neurônio **fuzzy** [GOM1 92].

- O neurônio Max [Figura 3.5]: proposto por Gomide e Rocha [GOM1 92,GOM2 92], pode também ser obtido. Se vale (4.7) e o limiar axônico $\alpha_1(\tau) = 0$ para $\tau = 0$ (pela ação do *setting neuron*), e em τ ele é feito igual ao nível de ativação $a_p(\tau - 1)$ em $\tau - 1$:

$$\alpha_1(\tau) = a_p(\tau - 1)$$

e além disso, a saída $a_p(t)$ é dada por:

$$a_p(\tau) = \begin{cases} \alpha_1(\tau) & \text{se } v(\tau) \leq \alpha_1(\tau) \\ v(\tau) & \text{caso contrário} \end{cases}$$

onde $v(t)$ é a ativação pós-sináptica em τ, então a saída $a_p(\tau)$ do neurônio em τ é dada por:

$$a_p(\tau) = \bigvee_{i=1}^{\tau} g_i(a_i) \tag{4.9}$$

Se $g_i = I$ para todo i, obtém-se a mesma relação dada por 3.18, que caracteriza o neurônio Max:

$$a_p(\tau) = \bigvee_{i=1}^{\tau} a_i \tag{4.10}$$

onde $\bigvee$ é operador máximo e I denota a função identidade.

Um neurônio satisfazendo (4.9) será chamado neurônio Max generalizado.

- Analogamente, o neurônio Min [Figura 3.5], proposto por Gomide e Rocha [GOM1 92], [GOM2 92], pode ser obtido e o limiar axônico $a_2(\tau) = 1$ em $\tau = 0$ (pela ação do *setting neuron*), e em τ é feito igual ao nível de ativação $a_p(\tau - 1)$ em $\tau - 1$:

$$a_2(\tau) = a_p(\tau - 1).$$

A saída $a_p(t)$ é dada por:

$$a_p(\tau) = \begin{cases} a_2(\tau) & \text{se } v(\tau) \geq a_2(\tau) \\ v(\tau) & \text{caso contrário} \end{cases}$$

onde $v(\tau)$ é a ativação pós-sináptica em τ, então nestas condições, a saída $a_p(\tau)$ do neurônio em τ é dada por:

$$a_p(\tau) = \bigwedge_{i=1}^{\tau} g_i(a_i). \tag{4.11}$$

Se $g_i = I$ para todo i, obtém-se a mesma relação dada por 3.21, que caracteriza o neurônio Min:

$$a_p(\tau) = \bigwedge_{i=1}^{\tau} a_i \tag{4.12}$$

onde $\bigwedge$ é o operador mínimo.

Um neurônio satisfazendo (4.11) será chamado neurônio Min generalizado.

- Outras operações entre neurônios podem ser obtidas, como por exemplo, a operação produto. Se $g_i(a_i) = g(a_i) = ln(a_i)$ para todo i, então

$$v = \sum_{i=1}^{n} ln(a_i)$$

e se $f(x) = e^x$ tem-se que a ativação axônica pós-sináptica a_p do neurônio é dada por:

$$a_p = \begin{cases} a_1.a_2.\ldots.a_n & \text{se } v \geq a \\ 0 & \text{caso contrário} \end{cases}$$

onde . denota produto algébrico.

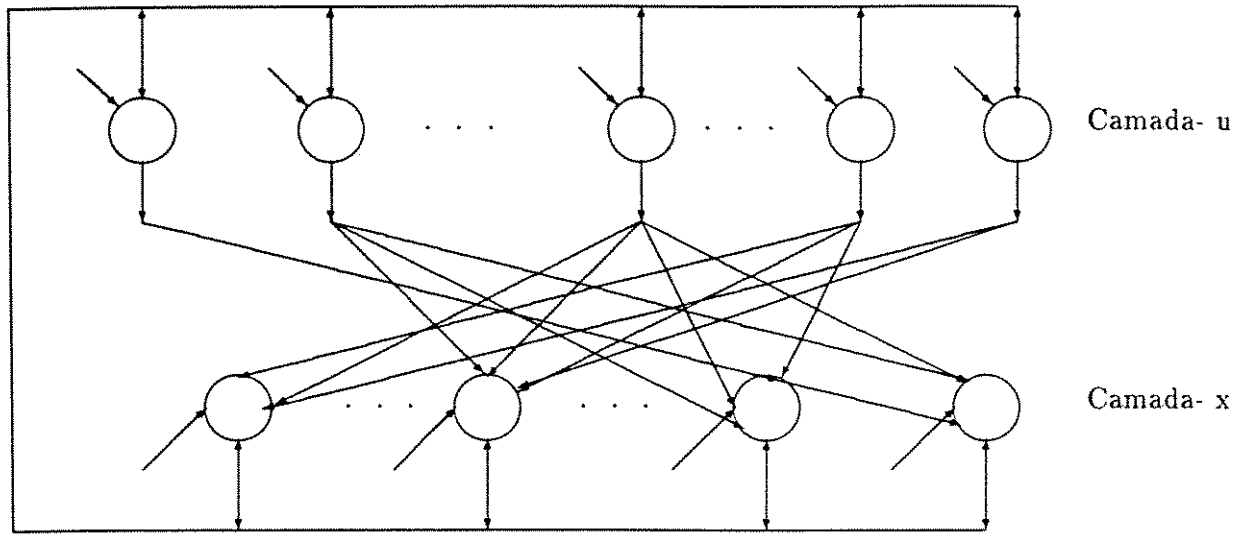


Figura 4.3: Rede Neural para Problemas de Controle Ótimo

A rede neural proposta para problemas de Controle Ótimo Discretos é uma rede com realimentação, constituída de duas camadas [Figura 4.3]. A primeira camada contém neurônios do tipo generalizado, denominados neurônios-u, enquanto que a segunda camada contém neurônios do tipo Max (ou Min), denominados, neurônios-x. A primeira e segunda camadas são denominadas Camada-u e Camada-x, respectivamente.

Uma vez que estamos considerando problemas discretos, as entradas na primeira camada correspondem aos diferentes valores que a variável de controle $u(k)$ pode assumir no estágio $N-k$ e serão denotados por u_i, $i = 1, 2, \ldots$, enquanto que as entradas na segunda camada, correspondem aos diferentes valores que a variável de estado $x(k)$ pode assumir, no estágio-$N-k$, e, por sua vez, serão denotados por x_i, $i = 1, 2, \ldots$.

A ativação das conexões (pesos) entre a Camada-u e Camada-x, a ativação das conexões realimentação entre a Camada-u e Camada-x e a ativação da função controladora são especificadas pela própria rede.

Um **neurônio-u** [Figura 4.4] é um neurônio do tipo generalizado que recebe dois sinais de entrada sincronizados, $a1(u)$ e $a2(u)$, para fornecer uma saída s. Estes sinais $a1(u)$ e $a2(u)$ podem ser vetores de dimensões apropriadas.

Supondo que as entradas $a1(u)$ e $a2(u) \in R^n$ e que as funções peso associadas são

$$G(a1(u)) = (g_1(a1_1(u)), g_2(a1_2(u)), \ldots, g_n(a1_n(u)))$$

e

$$G(a2(u)) = (g_1(a2_1(u)), g_2(a2_2(u)), \ldots, g_n(a2_n(u))).$$

respectivamente, então, a saída s do neurônio-u é dada por:

$$s = f(\sum_{i=1}^{n} g_i(a1_i(u)) + \sum_{i=1}^{n} g_i(a2_i(u)))$$

sendo

$$f(\theta) = \begin{cases} \theta & \text{se } \theta \geq \alpha \\ 0 & \text{caso contrário} \end{cases} \tag{4.13}$$

onde α é um limiar do neurônio-u. Uma função similar é usada pelo elemento *ASE* proposto por [BART 83].

Na rede neural para controle ótimo discreto (DOCNN), as funções peso associadas às entradas do neurônio-u têm o seguinte significado. A função $G(a1(u))$ corresponde a uma parte da função objetivo relacionada com a variável u no estágio $N-k$, sendo portanto, fornecida pelo usuário na fase de inicialização da rede. A função peso $G(a2(u))$ é tal que:

- Para o Estágio-$N-1$, $G(a2(u))$ corresponde a uma parte da função objetivo relacionada com a variável $x(N)$, devendo também ser fornecida na fase de inicialização da rede;
- Para o Estágio-$N-k$, $k=2,\ldots,N$, a função peso $G(a2(u))$ é definida como sendo a função identidade.

Um **neurônio-x** [Figura 4.4] é um neurônio Max generalizado, se problema de otimização a ser resolvido for um problema de Maximização, ou é um neurônio Min generalizado, caso contrário. Ele recebe vários pares de sinais de entrada: $(a1(x), a2(x))$ que são sincronizados para fornecer a saída s.

Supondo que um par de entrada num determinado neurônio-x é $a1(x)$ e $a2(x) \in R^m$ e que as funções peso associadas são

$$G(a1(x)) = (g_1(a1_1(x)), g_2(a1_2(x)), \ldots, g_m(a1_m(x)))$$

e

$$G(a2(x)) = (g_1(a2_1(x))), g_2(a2_2(x)), \ldots, g_n(a2_n(x))),$$

respectivamente, então, a saída s do neurônio-x é dada pelo máximo ou mínimo de suas entradas,

$$s = \max_l \{a_l\}$$

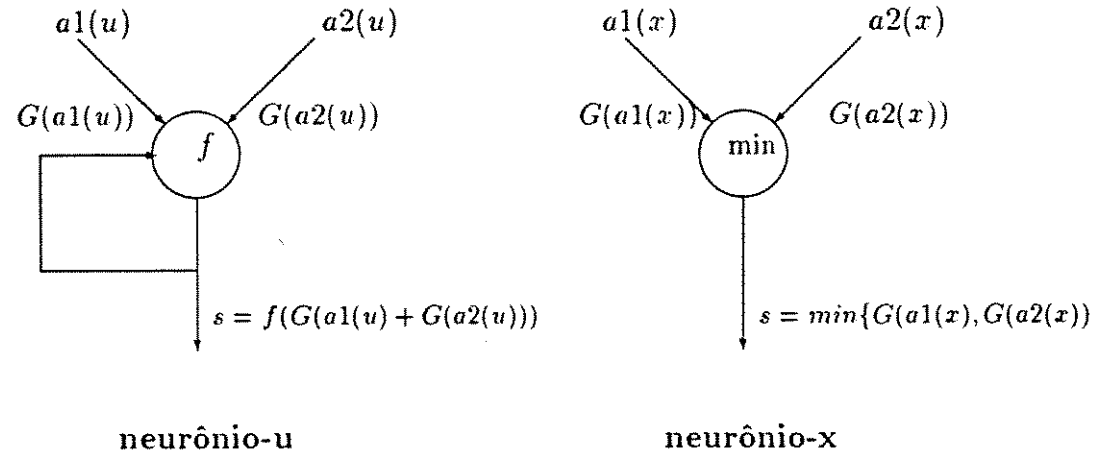


Figura 4.4: Neurônios da DOCNN

ou

$$s = \min_{l}\{a_l\}$$

onde $a_l = G(a1(x)) + G(a2(x))$ e l denota o número de pares de entradas no neurônio-x.

Na DOCNN, as funções pesos associadas às entradas do neurônio-x têm o seguinte significado. O valor de entrada $a1(x)$ é um valor fixado para cada neurônio-x, pois contém um dos valores possíveis que a variável x assume no estágio $N-k$. A função $G(a1(x))$ corresponde a uma parte da função objetivo relacionada com a variável x no estágio $N-k$, sendo portanto, fornecida pelo usuário na fase de inicialização da rede. O vetor de entrada $a2(x)$ tem dimensão 1, uma vez que ele é igual à saída de algum neurônio-u. Então, se $G(a1(x)) = 0$ e $G(a2(x)) = I$, o neurônio-x seria exatamente um neurônio do tipo Max, definido anteriormente.

Na seção Exemplos Ilustrativos mostra-se claramente, através de exemplos, como as funções peso são designadas.

Processamento Simbólico

O processamento simbólico corresponde a fase de programação da rede, pois é durante este processamento que a rede incorpora sistematicamente, conhecimento sobre o sistema dado. Os pesos associados às interconexões da rede são designados nesta fase.

Como no caso das redes propostas DPNN e FDPNN, as conexões entre neurônios são baseadas nos conceitos descritos em [ROCH 92] relacionados com a troca de mensagens

ocorrida entre os neurônios, durante o processamento químico sináptico. Como mencionamos esses conceitos nas seções anteriores, lembramos aqui apenas as relações descritas a seguir, onde T denota um transmissor, R um receptor e C um controlador de um neurônio:

- A ativação da célula pós-sináptica n_j devido ao transmissor T liberada pela célula pré-sináptica n_i ativa moléculas de controle C_j

$$T_i \bigoplus R_j >> C_j \tag{4.14}$$

onde $\bigoplus$ e $>>$ denotam operações de concatenação e translação, respectivamente.

- Cada ligação entre um transmissor pré-sináptico e um receptor pós-sináptico ativa diferentes tipos de controladores C_j. Estes controladores exercem diferentes tipos de ações entre as células pré e pós-sinápticas,

$$T_i \bigoplus R_j >> C_j \Rightarrow \text{ação} \tag{4.15}$$

- a quantidade $q(C)$ de controladores resultante da concatenação $T \bigoplus R$ é calculada como:

$$q(C) = (q(T) \oplus q(R)) \; o \; \mu(T.R) \tag{4.16}$$

onde $q(T)$ é a quantidade de T; $q(R)$ é a quantidade de R e $\mu(T,R)$ é o grau de *matching* entre T e R e as operações $\oplus$ e o são, em geral, t-normas ou t-conormas [ROCH 92].

As relações (4.14) - (4.16), serão interpretadas por:

Se $[(T_i)$ **e** $(R_j)]$ **então** ativar (C_j).
Se $(C_j$ está ativo) **então** ação.

$$q(C) = f(q(R), q(T)).$$

Com estas considerações, o processamento simbólico [Figura 4.5] é definido.

O neurônio-x envia, através de seu transmissor, o sinal $a1(x)$ ao neurônio-u, i.é., $q(T(x)) = a1(x)$. O neurônio-u, através de seu receptor (tal que $q(R(u)) = a1(x)$), recebe este sinal e, juntamente com sua outra entrada $a1(u)$ (tal que $a1(u) = q(T(u))$) ativa sua função controladora, $C(u)$, determinando:

$$q(C(u)) = f(q(T(u)), q(R(u)).$$

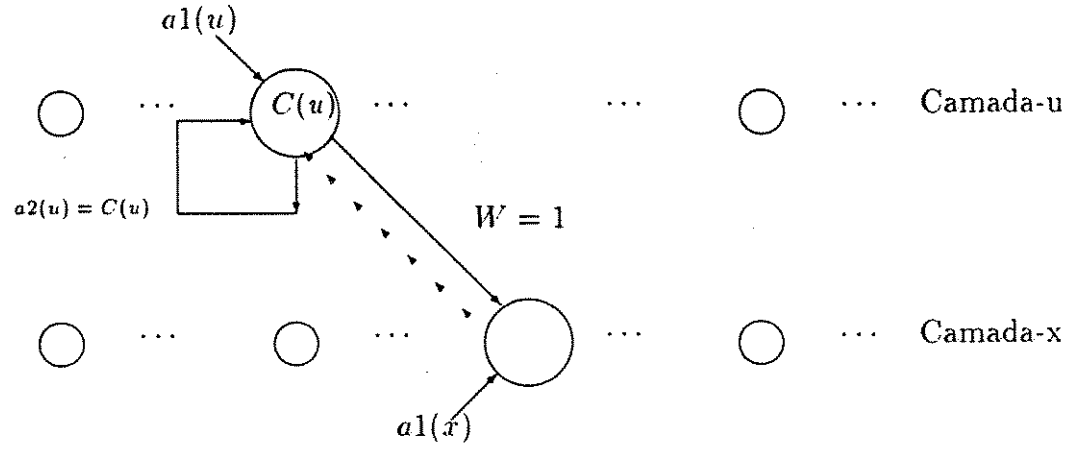


Figura 4.5: Processamento Simbólico entre o neurônio-u e neurônio-x

onde f corresponde ao modelo do sistema. Assim, o valor $C(u)$ é uma função dos sinais de entrada recebidos pelo neurônio-u.

Com o valor da função controladora já calculado, o seguinte teste é realizado:

Se $C(u)$ é factível **então**

uma conexão do neurônio-u para neurônio-x [Figura 4.5] é estabelecida

e o peso correspondente é dado por: W(neurônio-u, neurônio-x) = 1

senão W(neurônio-u, neurônio-x) = 0.

Desta forma, durante o processamento simbólico os valores de $C(u)$ são calculados para todos os pares: (neurônio-u, neurônio-x) e, em função destes valores, ações são selecionadas. As ações selecionadas, neste caso, consistem em ativar ou desativar conexões. Este processo é similar ao que ocorre no sistema *Q-learning*, onde estimativas de $\hat{Q}$ são obtidas para todos os pares de estados/ações e, em função destas estimativas, ações são selecionadas.

Durante o processamento simbólico as seguintes ações são também executadas:

a sinapse recorrente é ativada e o sinal $a2(u) = q(C(u))$ é enviado para si próprio.

ou as conexões de realimentação são ativadas, enviando a $q(T(u)) = C(u)$ para todos os neurônios-x da Camada-x.

Processamento Numérico

- Neurônio-u

O neurônio-u pode realizar dois tipos de atividades. Se sua conexão recorrente for ativada, então o seu sinal de saída, $s(u)$, será emitido em função de suas entradas: $a1(u)$, $a2(u)$ e respectivas funções peso $G(a1(u))$ e $G(a2(u))$,

$$s(u) = f(\sum_{i=1}^{n} g_i(a1_i(u)) + \sum_{i=1}^{n} g_i(a2_i(u)))$$

Por outro lado, se as conexões de realimentação forem ativadas, o grau de *matching* entre cada um dos neurônios-x da Camada-x e o sinal $q(T(u)) = C(u)$ será calculado pelo neurônio-u, i.é., o neurônio-x cuja quantidade do transmissor, $q(R(x))$ é aproximadamente igual a $q(T(u))$, segundo alguma métrica, será identificado e sua saída, $s(x)$, será enviada para o neurônio-u. Aqui a rede utiliza o mesmo processo conhecido como mecanismo competitivo, usado por Kohonen [KOHO 84] em sua rede, através do qual todos os neurônios da rede recebem o sinal de entrada e tentam ser ativados, mas somente o neurônio **vencedor** (aquele que possuir maior grau de *matching* com a entrada) é ativado. Na DOCNN, a métrica usada é a distância euclidiana.

Em seguida o sinal de saída do neurônio-u, $s(u)$, é emitido em função de suas entradas $a1(u)$ e $a2(u) = s(x)$ e respectivas funções peso $G(a1(u))$ e $G(a2(u)) = I$,

$$s(u) = f(\sum_{i=1}^{n} g_i(a1_i(u)) + \sum_{i=1}^{n} g_i(a2_i)).$$

- Neurônio-x

O neurônio-x, através de seu receptor, recebe o sinal de saída $a2(x) = s(u)$, emitido pelo neurônio-u e juntamente com sua entrada, $a1(x) = q(T(x))$ e respectivas funções peso I e G, calcula uma de suas entradas, a_l:

$$a_l = I(s(u)) + \sum_{i=1}^{n} g_i(a1_i(u)).$$

A Figura 4.6 mostra o processamento numérico entre os neurônios x e u, depois que o processamento simbólico foi realizado.

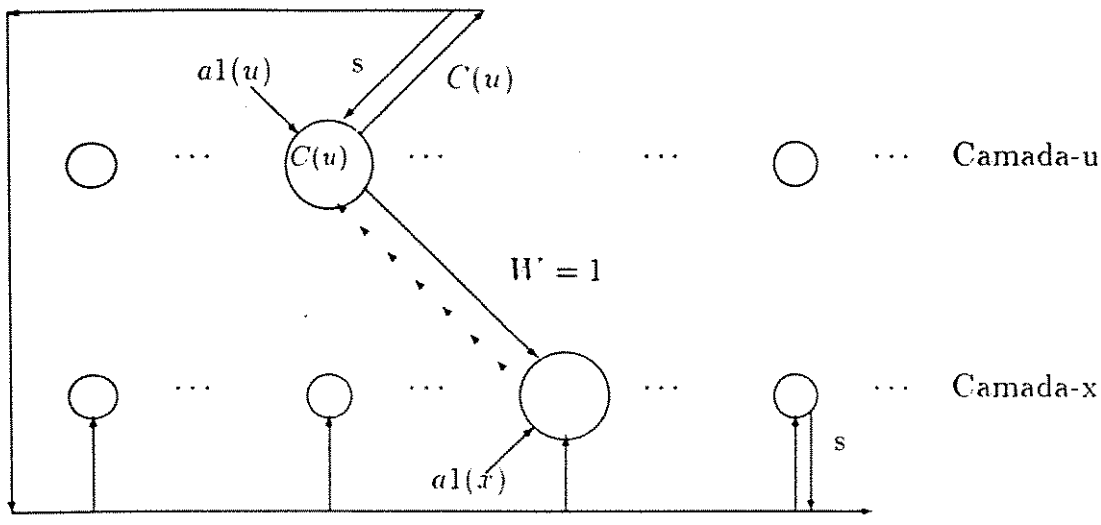


Figura 4.6: Processamento Numérico entre o neurônio-u e neurônio-x

4.4.2 Dinâmica da DOCNN - Algoritmo de Programação

Denotaremos por:

- S - número de neurônios da Camada-u
- C - número de neurônios da Camada-x
- NS - número de estágios
- N - dimensão do vetor de entrada no neurônio-u
- M - dimensão do vetor de entrada no neurônio-x
- u^j - j-ésimo neurônio-u da Camada-u
- $a1(u^j)$ e $a2(u^j)$ - entradas no j-ésimo neurônio-u
- x^j - j-ésimo neurônio-x da Camada-x
- $a1(x^j)$ e $a2(x^j)$ - entradas no j-ésimo neurônio-x
- $q(T(u^i)).q(R(u^i))$ - quantidade do transmissor e receptor do i-ésimo neurônio-u

- $q(T(x^i)), q(R(x^i))$ - quantidade do transmissor e receptor do i-ésimo neurônio-x
- W - matriz de pesos de ordem C x S
- $s(.)$ - saída de um neurônio da rede
- $g(.) = (g_1(.), g_2(.), \ldots . g_n(.))$ - função peso da entrada
- $soma_g(.) = \sum_{k=1} g_k(.)$

Passo - Estágio NS-1

Initialização

Para $j = 1.2, \ldots . S$ **faça**

Para $i = 1.2, \ldots . N$ **faça**

Início

leia$(a1_i(u^j))$

designe $g_i(a1_i(u^j))$

designe $g_i(a2_i(u^j))$

Fim

Para $j = 1.2. \ldots . C$ **faça**

Para $i = 1.2. \ldots . M$ **faça**

Início

leia$(a1_i(x^j))$

designe $g_i(a1_i(x^j))$

Fim

Para $i = 1.2. \ldots . S$ **faça**

$q(T(u^i)) = a1(u^i)$

Para $j = 1.2. \ldots . C$ **faça**

$q(T(x^i)) = a1(x^i)$

Processamento

Para $j = 1, 2, \ldots, C$ **faça**

Início

$l = 1$

Para $i = 1, 2, \ldots, S$ **faça**

Início

Execute o Passo Processamento Simbólico $NS-1$ para o par (u^i, x^j)

Se $W(u^i, x^j) = 1$ **então**

Execute o Passo Processamento Numérico

Fim

Calcular a saída de x^j, $s = \min_l\{a_l\}$

Armazenar o valor de u correspondente ao valor máximo

Fim

Passo - Estágio NS-k

Initialização

Para $j = 1, 2, \ldots, S$ **faça**

Para $i = 1, 2, \ldots, N$ **faça**

Início

leia$(a1_i(u^j))$

designe $g_i(a1_i(u^j))$

Fim

Para $j = 1, 2, \ldots, C$ **faça**

Para $i = 1, 2, \ldots, M$ **faça**

Início

leia$(al_i(x^j))$

designe $g_i(al_i(x^j))$

Fim

Para $i = 1, 2, \ldots, S$ **faça**

$q(T(u^i)) = al(u^i)$

Para $j = 1, 2, \ldots, C$ **faça**

$q(T(x^i)) = al(x^i)$

Processamento

Para $j = 1, 2, \ldots, C$ **faça**

Início

$l = 1$

Para $i = 1, 2, \ldots, S$ **faça**

Início

Execute o Passo Processamento Simbólico $NS-k$ para o par (u^i, x^j)

Se $W(u^i, x^j) = 1$ **então**

Execute o Passo Processamento Numérico

Fim

Calcular a saída de x^j, $s = \min_l\{a_l\}$

Armazenar o valor de u correspondente ao valor máximo

Fim

Passo Processamento Simbólico $NS-1$

Para o i-ésimo neurônio-u, u^i **faça**

Início

Ativar a conexão recorrente, enviando o sinal $q(T(u^i)) = C(u^i)$

para o próprio neurônio u^i, então $a2(u^i) = q(C(u^i))$

Passo Processamento Simbólico $NS-k$

Para o i-ésimo neurônio-u, u^i, **faça**

Início

Receber, através de seu receptor, a quantidade $q(T(x^j)) = x1$ transmitida

pelo neurônio x^j.

Ativar a função controladora $C(u^i) = f(q(T(u^i)), q(R(x^j))$

Se $C(u^i)$ é factível **então**

$W(u^i, x^j) = 1$

senão

$W(u^i, x^j) = 0$

Ativar as conexões de realimentação, enviando o sinal $q(C(u^i))$ para todos

os neurônios-x da Camada-x

Receber a saída $s(x)$, onde x é o neurônio com o maior grau

de *matching* com o neurônio u^i, i.é., $a2(u^i) = s(x)$

Fim

Passo Processamento Numérico

Para o i-ésimo neurônio-u, u^i, **faça**

Calcular a saída

$$s(u^i) = soma_g(a1(u^i)) + soma_g(a2(u^i))$$

Para o j-ésimo neurônio-x, x^j, **faça**

Início

Receber a saída $s(u^i)$, i.é., $a2(x^j) = s(u^i)$

Calcular uma de suas entradas $a_l = s(u^i) + soma_g(x^j)$

$l = l + 1$

Fim

Algoritmo Principal

Início

Executar o Passo Estágio $NS - 1$

Para $k = 2, ..., NS$ **faça**

Executar o Passo Estágio $NS - k$

Fim

4.4.3 Exemplo Ilustrativo

Nesta seção, o Algoritmo Principal para a rede neural proposta é desenvolvido para ilustrar como a DOCNN é construída.

Exemplo 1 [KIRK 70]:

Consideremos o sistema discreto, definido pela equação às diferenças:

$$x(k+1) = [1 + a\Delta t]x(k) + b\Delta t u(k) \quad k = 0, 1, ..., N-1.$$

Deseja-se selecionar valores para $u(0), u(1), \ldots$ de forma a mover o sistema de um estado inicial $x(0)$, para um estado final $x(N)$, com o menor esforço. A função objetivo associada a este problema é:

$$min \quad J = x^2(N) + \lambda\Delta t[u^2(0) + u^2(1) + ... + u^2(N-1)] = x^2(N) + \lambda\Delta t \sum_{k=0}^{N-1} u^2(k)$$

As seguintes restrições são também consideradas:

$$0.0 \leq x(k) \leq 1.5$$

$$-1.0 \leq u(k) \leq 1.0$$

N é o número de estágios e λ é uma constante que permite o ajuste da importância relativa dos dois termos em J.

Para simplificar os cálculos, seja a = 0, b = 1, $\lambda = 2$, N = 2, $\Delta t = 1$. Então, o processo de dois estágios é descrito pela equação às diferenças:

$$x(k+1) = x(k) + u(k) \quad : \quad k = 0, 1$$

e a função objetivo é dada por:

$$J = (x(2))^2 + 2(u(0))^2 + 2(u(1))^2.$$

Para seguir os passos do Algoritmo Principal, vamos supor que a variável u e a variável x são quantizadas nos seguintes valores:

$u(k) = \{-1.0, -0.5, 0., 0.5, 1.0\}$ e $x(k) = \{0., 0.5, 1.0, 1.5\}$.

Temos que: $NS = 2; S = 5; C = 4$.

De acordo com o algoritmo, vamos executar o Passo Estágio $NS - 1 = 1$.

- Inicialização: No problema dado, u e x são variáveis unidimensionais e, portanto, as entradas nestes neurônios são de dimensão 1 e, segundo a discretização, são dadas por:

$a1(u^1) = -1.0; a1(u^2) = -0.5; a1(u^3) = 0.; a1(u^4) = 0.5; a1(u^5) = 1.0$ e

$a1(x^1) = 0.; a1(x^2) = 0.5; a1(x^3) = 1.0; a1(x^4) = 1.5.$

$q(T(u^i)) = a1(u^i);\ i = 1, \ldots, 5;\ q(T(x^i)) = a1(x^i);\ i = 1, \ldots, 4$

As funções peso correspondentes aos neurônios u^i e x^i são, respectivamente $g(a1(u^i))$ e $g(a1(x^i))$, que corresponde ao termo da função objetivo que depende da variável u e a variável x, ambas no estágio 1. Então:

$$g(a1(u^i)) = 2 * [a1(u^i)]^2 \ \epsilon \ g(a1(x^i)) = 0 \text{ (função nula)}$$

A função $g(a2(u^i))$ também deve ser designada, durante a inicialização, e corresponde a uma parte da função objetivo que envolve a variável x no estágio $NS = 2$. Logo:

$$g(a2(u^i)) = [a2(u^i)]^2.$$

Com as variáveis inicializadas, passamos para a fase do processamento do Passo Estágio 1. O passo processamento simbólico 1 deve ser realizado para $j = 1, 2, \ldots, 4$ e $i = 1, \ldots, 5$.

Por exemplo, fixando $j = 1$ e variando i, $i = 1, \ldots, 5$, temos:

- $i = 1$: O neurônio u^1 recebe a $q(T(x^1)) = a1(x^1) = 0$ e juntamente com a quantidade de seu transmissor $q(T(u^1)) = a1(u^1) = -1.0$ ativa a função controladora

$$C(u^1) = f(q(T(u^1), q(R(u^1)) = a1(u^1) + a1(x^1) = -1.0 + 0. = -1.0$$

Como $C(u^1)$ não é factível $W(u^1, x^1) = 0$ e o passo processamento numérico, não é executado.

- $i = 2$: O neurônio u^2 recebe a $q(T(x^1)) = a1(x^1) = 0$ e juntamente com a quantidade de seu transmissor $q(T(u^2)) = a1(u^2) = -0.5$ ativa a função controladora

$$C(u^2) = f(q(T(u^2), q(R(u^1)) = a1(u^2) + a1(x^1) = -0.5 + 0. = -0.5$$

Como $C(u^2)$ não é factível $W(u^2, x^1) = 0$ e o passo processamento numérico taqmbém não é executado.

$i = 3$: O neurônio u^3 recebe a $q(T(x^1)) = a1(x^1) = 0$ e juntamente com a quantidade de seu transmissor $q(T(u^3)) = a1(u^3) = 0.0$ ativa a função controladora

$$C(u^3) = f(q(T(u^3), q(R(u^1)) = a1(u^3) + a1(x^1) = 0. + 0. = 0.$$

Como $C(u^3)$ é factível $W(u^3, x^1) = 1$ e a conexão recorrente do neurônio-u é ativada com $a2(u^3) = C(u^3) = 0.0$. Em seguida, o passo processamento numérico é executado. No processamento numérico e a saída do neurônio u^3 é calculada:

$$s(u^3) = soma_g(a1(u^3)) + soma_g(a2(u^3)) = 2 * (0.0)^2 + (0.0)^2 = 0.0$$

O neurônio-x, x^1, recebe a $s(u^3)$ e calcula a sua primeira entrada:

$$a_1 = s(u^3) + soma_g(a1(x^1)) = 0. + 0. = 0.$$

$i = 4$: O neurônio u^4 recebe a $q(T(x^1)) = a1(x^1) = 0$ e juntamente com a quantidade de seu transmissor $q(T(u^4)) = a1(u^4) = 0.5$ ativa a função controladora

$$C(u^4) = f(q(T(u^4), q(R(u^1)) = a1(u^4) + a1(x^1) = 0.5 + 0. = 0.5$$

Como $C(u^4)$ é factível então $W(u^4, x^1) = 1$ e a conexão recorrente do neurônio-u deve ser ativada com $a2(u^4) = C(u^4) = 0.5$. O passo processamento numérico é executado a seguir. A saída do neurônio u^4 é então:

$$s(u^4) = soma_g(a1(u^4)) + soma_g(a2(u^4)) = 2 * (0.5)^2 + (0.5)^2 = 0.75$$

O neurônio-x, x^1, recebe a $s(u^4)$ e calcula a sua segunda entrada:

$$a_2 = s(u^4) + soma_g(a1(x^1)) = 0.75 + 0. = 0.75$$

e, finalmente, para

$i = 5$: O neurônio u^5 recebe a $q(T(x^1)) = a1(x^1) = 0$ e juntamente com a quantidade de seu transmissor $q(T(u^5)) = a1(u^5) = 1.0$ ativa a função controladora

$$C(u^5) = f(q(T(u^5), q(R(u^1)) = a1(u^5) + a1(x^1) = 1.0 + 0. = 1.0$$

Como $C(u^5)$ é factível $W(u^5, x^1) = 1$ e a conexão recorrente do neurônio-u deve ser ativada com $a2(u^5) = C(u^5) = 1.0$ e o passo processamento numérico 1. é agora executado. Neste passo, a saída do neurônio u^5 é calculada:

$$s(u^5) = soma_g(a1(u^5)) + soma_g(a2(u^5)) = 2 * (1.0)^2 + (1.0)^2 = 3.00$$

O neurônio-x, x^1, recebe a $s(u^5)$ e calcula a sua terceira entrada:

$$a_3 = s(u^5) + soma_g(a1(x^1)) = 3.00 + 0. = 3.00$$

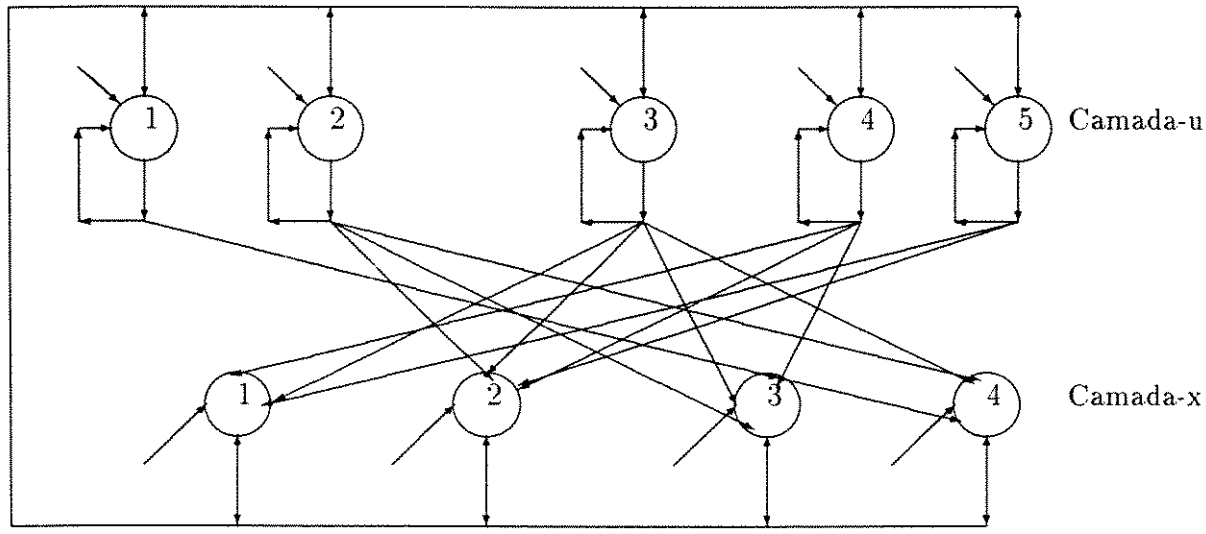


Figura 4.7: DOCNN para o exemplo ilustrativo

A seguir, antes de incrementarmos a variável j, a saída do neurônio x^1 deve ser calculada:

$$s(x^1) = \min\{0.0, 0.75, 3.00\} = 0.0$$

e o valor de $a1(u^3) = 0.0$ deve ser armazenado, como o valor da variável u no estágio 1, que fornece o valor ótimo para o estado $x = 0.0$.

O processo descrito acima é repetido para $j = 2, 3$ e 4 e obtendo-se:

$$s(x^2) = 0.25 \quad \text{com } u = 0.0;$$

$$s(x^3) = 0.75 \quad \text{com } u = -0.5;$$

$$s(x^4) = 1.5 \quad \text{com } u = -0.5.$$

Seguindo o Algoritmo Principal, $k = 2$ e o Passo Estágio 0 é executado. As variáveis neste passo são inicializadas da mesma forma que no Estágio anterior, uma vez que os valores das variáveis u e x foram discretizados do mesmo modo para todos os estágios. Em geral, em problemas de controle os valores discretizados das variáveis u e x são os mesmos para todos os estágios.

Entretanto, as funções peso devem ser atualizadas para o estágio 0. Analisando a função objetivo temos: $g(a1(u^i)) = 2 * [a1(u^i)]^2$ e $g(a1(x^i)) = 0$ uma vez que não aparece termos envolvendo a variável x no estágio 0.

Com as variáveis inicializadas, passamos para a fase de processamento do Passo Estágio 0. O passo processamento simbólico 0 deve ser realizado para $j = 1, 2, \ldots, 4$ e $i = 1, \ldots, 5$.

Por exemplo, considerando $j = 1$ e $i = 3$, obtemos:

Durante o processamento simbólico 0, o neurônio u^3 recebe a $q(T(x^1)) = a1(x^1) = 0$ e juntamente com a quantidade de seu transmissor $q(T(u^3)) = a1(u^3) = 0.0$ ativa a função controladora

$$C(u^3) = f(q(T(u^3), q(R(u^1)) = a1(u^3) + a1(x^1) = 0. + 0. = 0.$$

Como $C(u^3)$ é factível $W(u^3, x^1) = 1$, a conexão de realimentação do neurônio-u é ativada e o sinal $q(T(u^3)) = C(u^3) = 0.0$ é enviado para todos os neurônios-x.

Em seguida, o neurônio-u, u^3 recebe a saída do neurônio-x, que possui maior grau de *matching* com o sinal $C(u^3)$. Neste caso, o neurônio x^1 é o neurônio vencedor, pois a quantidade de seu transmissor, $q(T(x^1)) = q(T(u^3)) = 0.0$.

Então, o passo processamento numérico 0 é executado. O neurônio u^3 recebe $s(x^1) = 0.0$ e a sua saída é calculada:

$$s(u^3) = f(s(x^1) + soma_g(a1(u^3))) = 0.0 + 2 * (0.0)^2 = 0.0$$

O neurônio-x, x^1, recebe a $s(u^3)$ e calcula a sua primeira entrada:

$$a_1 = s(u^3) + soma_g(a1(x^1)) = 0. + 0. = 0.$$

Analogamente, considerando $j = 1$ e $i = 4$, obtemos:

Durante o processamento simbólico 0, o neurônio u^4 recebe a $q(T(x^1)) = a1(x^1) = 0$ e juntamente com a quantidade de seu transmissor $q(T(u^4)) = a1(u^4) = 0.5$ ativa a função controladora

$$C(u^4) = f(q(T(u^4), q(R(u^1)) = a1(u^4) + a1(x^1) = 0.5 + 0. = 0.5$$

Como $C(u^4)$ é factível $W(u^4, x^1) = 1$ e a conexão de realimentação do neurônio-u é ativada e o sinal $q(T(u^4)) = C(u^4) = 0.5$ é enviado para todos os neurônios-x.

Em seguida, o neurônio u^4 recebe a saída do neurônio-x, que possui maior grau de *matching* com o sinal $C(u^4)$. Neste caso, o neurônio x^2 é o neurônio vencedor, pois a quantidade de seu transmissor, $q(T(x^2)) = q(T(u^4)) = 0.5$. A seguir, o processamento numérico deve ser executado. O neurônio u^4 recebe $s(x^2) = 0.25$ e sua saída é calculada:

$$s(u^4) = f(s(x^1) + soma_g(a1(u^4))) = 0.25 + 2 * (0.5)^2 = 0.75$$

O neurônio-x, x^1, recebe a $s(u^4)$ e calcula a sua segunda entrada:

$$a_1 = s(u^3) + soma_g(a1(x^1)) = 0.75 + 0. = 0.75.$$

Continuando o processamento da rede até o final do Estágio 0, obtemos:

$$s(x^1) = 0.0 \text{ com } u = 0.0;$$

$$s(x^2) = 0.25 \text{ com } u = 0.0;$$

$$s(x^3) = 0.75 \text{ com } u = -0.5;$$

$$s(x^4) = 1.5 \text{ com } u = -0.5.$$

A DOCNN correspondente é mostrada na Figura 4.7.

4.4.4 Aplicação

Problemas de Reguladores Lineares Discretos

Nós descrevemos este problema aqui, de acordo com o apresentado em [KIRK 70], cujo sistema discreto é descrito pela equação de estado:

$$x(k+1) = A(k)x(k) + B(k)u(k)$$

onde $u(k)$ e $x(k)$ podem ser variáveis satisfazendo restrições.

O objetivo é encontrar uma política ótima $u^*(x(k), k)$ que minimiza a função objetivo:

$$J = \frac{1}{2}x^T(NS)Hx(NS) + \frac{1}{2}\sum_{k=0}^{NS-1}[x^T(k)Q(k)x(k) + u^T(k)R(k)u(k)]$$

onde H e $Q(k)$ são matrizes nxn, simétricas definidas semi-positiva, $R(k)$ é uma matriz real mxm, definida positiva e NS é um inteiro positivo que denota o número de estágios.

Para simplificar a notação no desenvolvimento que se segue, assumiremos que A, B, R e Q são matrizes constantes.

Para resolver este problema pela abordagem proposta, devemos inicialmente discretizar as variáveis de estado $x \in R^n$ e de controle $u \in R^m$, para os estágios $k = 1, 2, \ldots, NS$.

Por exemplo, se $u = (u_1, u_2, \ldots, u_n)$ onde u_1 pode assumir s_1 valores, u_2 pode assumir s_2 valores e assim sucessivamente, u_n pode assumir s_n valores, no estágio k, temos que a rede DOCNN deverá ter $S = s_1.s_2.\ldots.s_n$ neurônios na Camada-u, onde cada neurônio-u, receberá um vetor de entrada de n componentes, denotado por $a1(u)$, que corresponde a uma das S combinações de valores possíveis para a variável u.

Analogamente, se $x = (x_1, x_2, \ldots, x_m)$ onde x_1 pode assumir c_1 valores, x_2 pode assumir c_2 valores e assim sucessivamente, x_m pode assumir c_m valores, no estágio k, então, a rede DOCNN deverá ter $C = c_1.c_2.\ldots.c_m$ neurônios na Camada-x, onde cada neurônio-x, receberá um vetor de entrada de m componentes, denotado por $a1(x)$, que corresponde a uma das C combinações possíveis para a variável x.

Assumiremos, sem perda de generalidade, que as variáveis de estado $u(k)$ e $x(k)$ são discretizadas da mesma forma para todos os estágios.

As funções peso: $g(a1(u)), g(a1(x))$ devem ser designadas, em cada estágio, para cada neurônio-u e neurônio-x na rede. De acordo com a definição, estas funções correspondem aos termos da função objetivo que dependem das variáveis u e x no estágio considerado. A função peso $g(a2(u))$ corresponde ao termo da função objetivo que depende da variável x no estágio NS, $x(NS)$. Desta forma, temos:

- Para $k = NS-1$

$$g(a1(u)) = \frac{1}{2}u(NS-1)\ R\ u(NS-1)$$

$$g(a2(u)) = \frac{1}{2}x(NS)\ H\ x(NS)$$

$$g(a1(x)) = \frac{1}{2}x(NS-1)\ Q\ x(NS-1).$$

- Para $k = NS-2, NS-3, \ldots, 0$

$$g(a1(u)) = \frac{1}{2}u(k)\ R\ u(k)$$

$$g(a2(u)) = I$$

$$g(a1(x)) = \frac{1}{2}x(k)\ Q\ x(k).$$

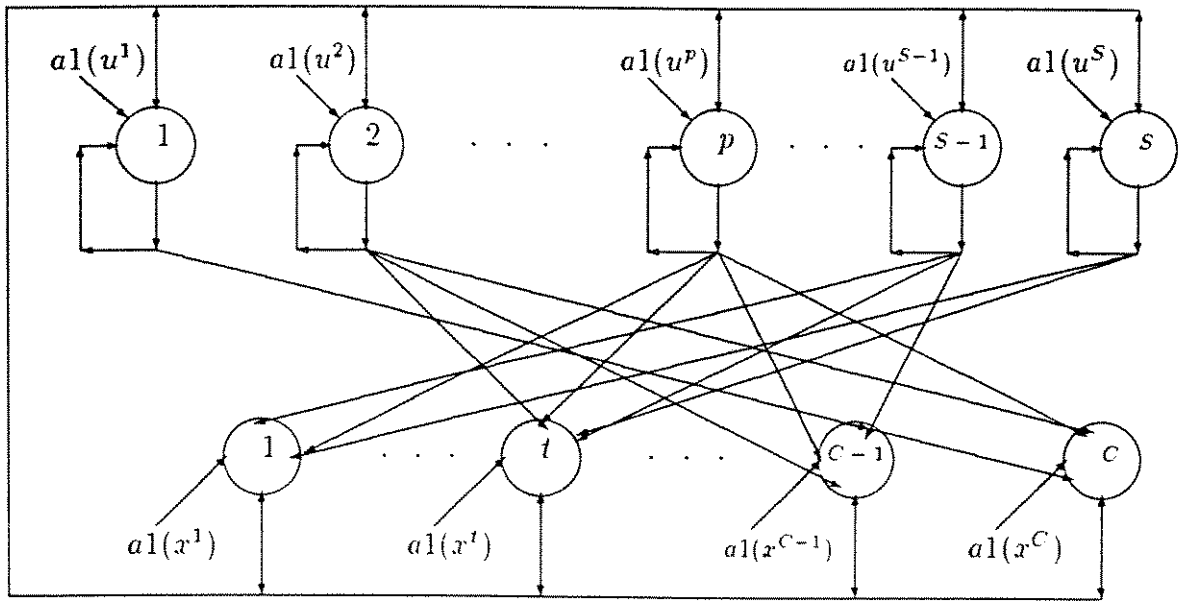


Figura 4.8: DOCNN para o problema de reguladores lineares discretos

O processamento da rede é executado de acordo com o algoritmo proposto na seção 4.4.2. A rede DOCNN correspondente a problemas de reguladores lineares é apresentada na Figura 4.8, para qualquer número de neurônios na Camada-u e qualquer número de neurônios na Camada-x.

Para ilustrar a eficiência da rede DOCNN, um problema de controle ótimo linear quadrático (LQ) é resolvido a seguir. Este problema foi proposto e resolvido por Parisini [PARIS 91], usando redes neurais multi-camadas *feed-forward* encadeadas.

A dinâmica do sistema é dada por:

$$x(k+1) = \begin{bmatrix} 0.65 & -0.19 \\ 0. & 0.83 \end{bmatrix} x(k) + \begin{bmatrix} 7 \\ 7 \end{bmatrix} u(k)$$

onde $x(k) \in R^2$ e $u(k) \in R$. A função objetivo é:

$$J = \sum_{k=0}^{NS-1} u^2(k) + v_{NS} \|x(NS)\|^2$$

com $v_{NS} = 40$ e $NS = 10$.

A variável de estado $x = (x_1, x_2) \in R^2$ deve satisfazer as seguintes restrições:

$$-0.5 \leq x_1(k) \leq 3.5$$

$$-1.0 \leq x_2(k) \leq 1.0$$

enquanto a variável de controle $u \in R$ deve satisfazer:

$$-0.5 \leq u(k) \leq 0.5$$

para todos os estágios $k = 0, 1, \ldots, 9$.

Simulamos a rede DOCNN correspondente a este problema, com $S = 107$ neurônios na Camada-u e $C = 300$ neurônios na Camada-x, implementando o algoritmo proposto na seção 4.4.2 na linguagem C, utilizando uma WorkStation Sparc 2. As trajetórias ótimas obtidas para dois pontos iniciais: $x(0) = (2.5, 1.0)^T$ e $x(0) = (2.5, -1.0)^T$, são mostradas na Figura 4.9. Os resultados obtidos são compatíveis com os apresentados por [PARIS 91]. Estes resultados foram obtidos em 10 iterações, enquanto que na abordagem proposta por Parisini, a simulação envolveu centenas de milhares de iterações, através de esquemas *backpropagation* encadeados.

4.5 Conclusão

Neste capítulo, um modelo generalizado de neurônio recorrente foi proposto, do qual os neurônios Max e Min são também casos particulares. Com base nesta generalização, uma rede neural visando a solução de problemas de controle ótimo multi-estágios foi proposta. A rede implementa um método direto para designar seus pesos e neste sentido a rede possui capacidade de auto-programação. O método direto proposto é equivalente ao algoritmo convencional da Programação Dinâmica e fundamenta-se na troca de informações que ocorrem entre os neurônios, durante o processamento químico sináptico. Em função do paralelismo inerente das redes neurais, a abordagem proposta é uma alternativa tanto em relação a outros esquemas de aprendizado que implementam aproximações do algoritmo de Programação Dinâmica, como por exemplo o *Q-learning*, quanto a outras abordagens de redes neurais que utilizam funções *sigmoidal* e esquemas do tipo *back-propagation*.

Uma análise comparativa sobre o desempenho da rede para resolver problemas mais complexos, bem como, uma implementação paralela da mesma fazem parte de estudo e investigações futuras.

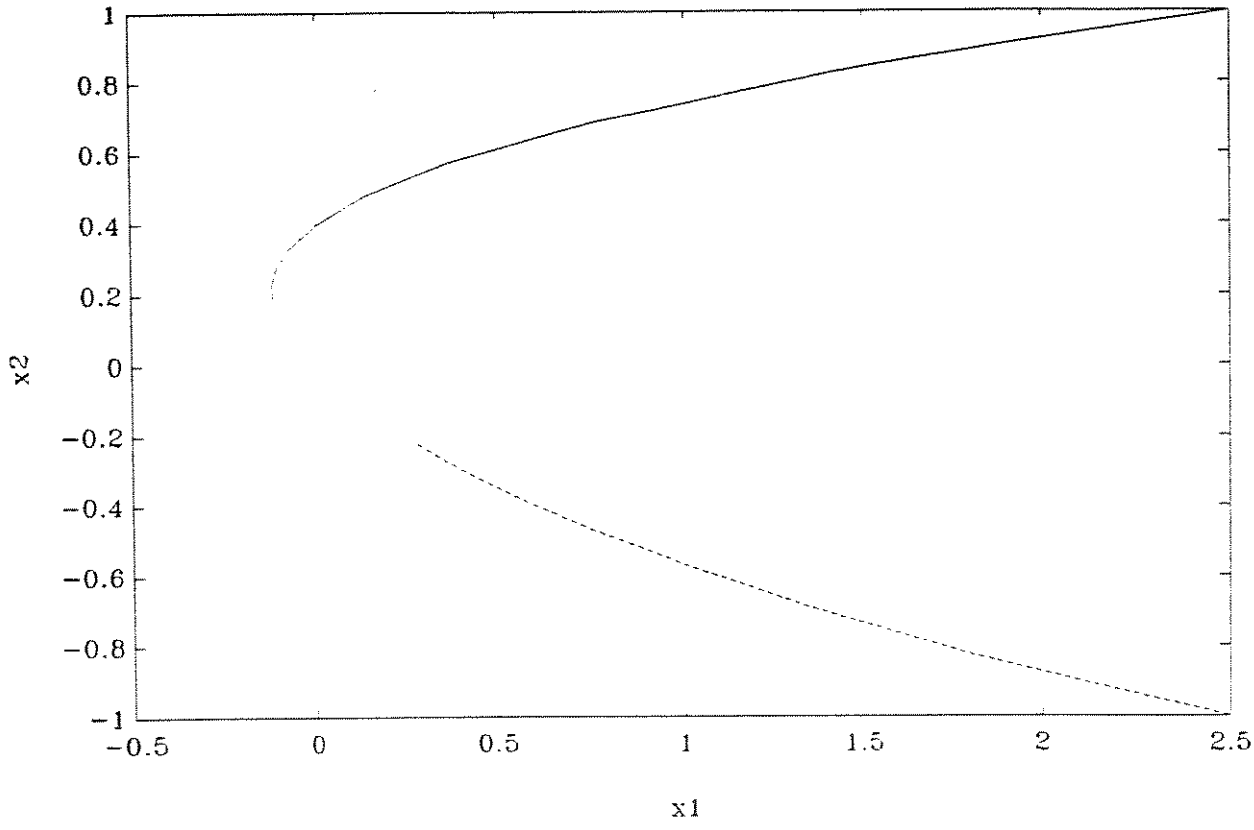


Figura 4.9: Convergência das trajetórias neurais ótimas de x(0) à origem

Capítulo 5

Conclusão

O presente trabalho se concentrou no desenvolvimento de arquiteturas de redes neurais artificiais para otimização de sistemas não-lineares.

Inicialmente, uma rede neural multi-camadas com realimentação foi proposta visando a solução de problemas de otimização convexos estáticos irrestritos, cujo esquema de atualização dos pesos é uma modificação do algoritmo *back-propagation*. Uma generalização da rede foi também apresentada para solução de problemas estáticos convexos com restrições, usando a teoria de dualidade e esquemas do tipo subgradiente. Uma implementação paralela da rede foi realizada usando um sistema constituído de 32 *transputers*. Além disso, uma análise comparativa do comportamento da mesma (considerando uma implementação sequencial) relativamente a métodos tradicionais de otimização e outras abordagens, foi apresentada.

Com base nesta análise e levando em consideração algumas características da rede proposta, tais como, uniformidade dos nós que compõem a rede, mesmo número de nós por camadas, ausência de treinamento através de exemplos, acredita-se que a mesma seja uma abordagem alternativa em relação às redes existentes para resolver problemas de otimização, bem como em relação a métodos tradicionais de otimização, em função do paralelismo presente nas arquiteturas de RNA. Entretanto, é importante lembrar que os testes realizados envolveram apenas funções e/ou restrições lineares, quadráticas ou bi-quadráticas. A resolução de problemas mais complexos faz parte dos trabalhos futuros.

Além disso, uma rede recorrente constituída de apenas duas camadas e baseada em um modelo de neurônio recorrente foi proposta, visando a solução de problemas de otimização envolvendo variáveis discretas de decisão. A rede neural implementa um método direto de programação de seus pesos, o qual incorpora conhecimento sobre a estrutura do problema de otimização associado. Esse método direto fundamenta-se no Princípio de Otimalidade

de Bellmann e é uma alternativa em relação a outros esquemas de aprendizado, pois baseia-se na troca de informações que ocorre entre os neurônios durante o processamento químico sináptico. Tal método foi estendido para incorporar o Princípio de Bellmann-Zadeh, visando a solução de problemas de otimização nebulosos.

Problemas conhecidos de otimização como o problema da mochila, o problema do caminho mínimo, problemas de reguladores lineares e um problema de planejamento de sistemas de potência a longo prazo foram resolvidos para verificar o desempenho e utilização da rede proposta. Uma análise dos requisitos computacionais exigidos foi também realizada, o que comprovou a vantagem da abordagem proposta relativamente ao algoritmo convencional da Programação Dinâmica. Entretanto, deve-se salientar que a eficiência do método depende de uma discretização adequada das variáveis envolvidas e que o número de neurônios cresce com o número de valores discretizados, o que não deve ser um problema num computador neural.

Finalmente uma generalização do modelo de neurônio recorrente foi obtida, possibilitando a utilização da rede proposta na solução de problemas de controle ótimo discretos.

Entre os trabalhos futuros, primeiramente, pretendemos investigar a possibilidade de implementação do algoritmo baseado em PD, *Q-learning*, através da rede neural de duas camadas com realimentação por nós proposta, uma vez que a mesma incorpora um método direto de aprendizado que é equivalente ao algoritmo convencional da PD. Além disso, como a maioria dos resultados obtidos entre os algoritmos de aprendizado com reforço são para controle de processos Markovianos, estudos serão realizados para verificar a adequação da abordagem proposta em ambientes Markovianos. Uma interface mais amigável para a implementação sequencial das duas diferentes arquiteturas de redes neurais propostas, bem como a implementação das mesmas num computador neural farão parte também dos trabalhos futuros.

Bibliografia

[ABU 85] ABU-MOSTAFA, Y. S. and JACQUES, J. St., "Information capacity of the Hopfield model", *IEEE Trans. Inform. Theory*, vol. 31, no. 461-464, 1985.

[ALE 78] ALEKSANDER, I., *The human machine*, ST Saphorin, Switz.: Georgi Publ., 1978.

[ALE 84] ALEKSANDER, I., THOMAS, W. V. and BOWDEN, P. A., "WISARD - a radical step forward in image recognition", *Sensor Review*, 4(3), 120-124, 1984.

[ALE 89] ALEKSANDER, I. (Ed.), *Neural computing architectures: the design of brain-like machines*, The MIT Press, Cambridge, 1989.

[ALLR 90] ALLRED, L. G. and KELLY, G. E., "Supervised learning techniques for back-propagation networks", vol. I, *Proc. IJCNN'90*, vol. I, pp. 721-218, 1990.

[AMAR 72] AMARI, S. I., "Learning patterns and pattern sequences by self-organizing nets of threshold elements", *IEEE Trans. Compt.*, Vol. C21, pp. 1197-1206, 1972.

[AMAR 77] AMARI, S. I., "Neural theory of association and concept-formation", *IEEE Biological Cybernetics*, vol. 26, pp. 175-185, 1977.

[ANDE 89] ANDERSON, C. W., "Learning to control an inverted pendulum using neural networks", *IEEE Contr. Syst. Mag.*, vol. 9, no. 3, April 1989.

[ANTS 92] ANTSAKLIS, P. J., "Neural networks in control systems", *IEEE Control Systems Mag.*, vol 12, no. 2, pp. 8-10, April 1992.

[AIYE 90] AIYER, S. V., NIRANJAN, M. and FALLSIDE, F., "A theoretical investigation into the performance of the Hopfield model", *IEEE Trans. on Neural Networks*, vol. 1, no. 2, June 1990.

[ARRO 58] ARROW, K. J.; HURWICZ, L., and UZAWA, H., *Studies in Linear and Nonlinear Programming*, Stanford University Press, 1958.

[BART 81] BARTO, A. G., SUTTON, R. S., "Goal seeking components for adaptive intelligence: an initial assessment", Air Force Wright Aeronautical Laboratories/Avionics Laboratory Tech. Rep. AFWAL-TR-81-1070, Ohio, 1981.

[BART 83] BARTO, A. G., SUTTON, R. S and ANDERSON C. W., "Neuronlike adaptive elements that can solve difficult learning control problems", *IEEE Trans. Syst., Man and Cybernetics*, vol. SMC-13, no. 5, pp. 834-846, Sept/Oct. 1983.

[BART 91] BARTO, A. G and BRADTKE, S. J., "Real-time learning and control using asynchronous dynamic programming", TR 91-57, Department of Computer Science, Massachusetts, Aug. 1991.

[BAZA 79] BAZARAA, M. S. and SHETTY, C. M., *Nonlinear Programming*, John Wiley & Sons, Inc., 1979.

[BECK 88] BECKER, S. and LE CUN, Y., "Improving the convergence of back-propagation learning wiht second order methods", *Proc. 1988 Connectionist Models Summer School*, Morgan Kaufman, San Mateo, CA, pp. 28-37, 1988.

[BEL1 57] BELLMAN, R., *Dynamic Programming*, Princeton N. J. : Princeton University Press, 1957.

[BEL2 70] BELLMAN, R. E. and ZADEH, L. A. "Decision-making in a fuzzy environment", *Management Science*, vol. 17, no. 4, pp. 141-164, Dec. 1970.

[BEZD 91] BEZDEK, J., *Fuzzy Logic and Computational Neural Networks* , Tutorial Notes, IJCNN'91, Nov. 91.

[BLUM 88] BLUM, A. and RIVEST, R. L., "Training a 3-node neural network is np-complete", *Proc. Computational Learning Theory* (COLT), pp. 9-18, Morgan Kauffman, 1988.

[BRUC 88] BRUCK, J. and GOODMAN, J. W., "On the power of neural networks for solving hard problems", em *Neural Info. Processing Systems*, D. Z. Anderson (Ed.), AIP Conference Proc., pp. 137-143, 1988.

[BURR 91] BURRASCANO, P., "A norm selection criterion for the generalized delta rule", *IEEE Trans. on Neural Networks*, vol. 2, no. 1, Jan. 1991.

[CAIA 75] CAIANELLO, E. and GRIMSON, E., "Synthesis of boolean nets and time behaviour of a general mathematical neuron", *Biol. Cybern.*, 18, pp. 111-117, 1975.

[CAIA 75] CAIANELLO, E. and MARINARO, M., "The inverse problem of neural nets and cellular automata", *In Computer Simulation in Brain Science*, Cambridge: WP, 1987.

[CARP 83] CARPENTER, G. A., GROSSBERG, S., " A massively parallel architecture for a self-organizing neural pattern recognition machine", *Computer Vision, Graphics, and Image Processing*, vol. 37, pp. 4-115. 1983.

[CAUD 87] CAUDILL, M., "Neural networks primer - Part I. Part II. Part III e Part IV", *AI Expert*, Dec/87, Feb/88, Jun/88, Ago/88.

[CAUD 90] CAUDILL, M., "Using neural nets: fuzzy decisions", *AI Expert*, pp. 59-64, April 1990.

[CHES 90] CHESTER, D. L., "Why two hidden layers are better than one", *Proc. Int. Joint. Conf. Neural Networks*, vol. 1, pp. 265-268. 1990.

[CYBE 89] CYBENKO, G., "Approximation by superpositions of a sigmoidal function", *Mathematical Control Signals Systems*, vol. 2. pp. 303-314, 1990.

[COTT 85] COTTRELL, G. W., *A connectionist approach to word sense disambiguation*, Ph.D. Thesis, University of Rochester, Computer Science, 1985.

[COTT 87] COTTRELL, G. W., MUNRO, P., and ZIPSER, D., *Image compression by backpropagation: an example of extensional programming*, ICS Rep. 8702, Univ. of California at San Diego. Feb. 1987.

[DAVI 89] DAVIS. G. W., "Sensitivity analysis in neural net solutions", *IEEE Trans. on Syst., Man, and Cybernetics*, vol. 19, no. 5, Sept/Oct. 1989.

[DHAR 79] DHAR. S. B., "Power system long-range decision analysis under fuzzy environment", *IEEE Trans. on Power Apparatus and Systems*. Vol. PAS-98, no. 2. March/April, 1979.

[DAHL 87] DAHL. E. D., "Neural network algorithm for an NP-complete problem: map and graph coloring", *Proc. 1st. Int. Conf. Neural Networks*, vol. III. San Diego. CA. pp. 113-120. 1987.

[DREY 77] DREYFUS. STUART E., *The Art and Theory of Dynamic Programming*. Academic Press. Inc., 1977.

[FAHL 88] FAHLMAN, S. E., *An empirical study of learning speed in back-propagation networks*, CMU-CS-88-162, 1988.

[FARR 90] FARREL, J. A. and MICHEL. A. N., "A synthesis procedure for Hopfield's continuous time associative memory", *IEEE Trans. Circuits and Systems*, vol.37. pp. 877-884. July 1990.

[FELD 82] FELDMAN. J. A., BALLARD. D. H., "Connectionist models and their properties", *Cognitive Science*, vol. 6. pp. 205-254. 1982.

[FOO 88] FOO, YOON-PIN S. and TAKEFUJI, Y., "Integer linear programming neural networks for job-shop scheduling", *Proc. IEEE Int. Conf. on Neural Networks.* vol. II, pp. 341-348, 1988.

[FONG 91] FONG, K. F. and LOH, P., "Discrete-time optimal control using neural nets", *Int. Joint Conf. Neural Networks*, vol. 2, pp. 1355-1360, Nov. 1991.

[FRA1 90] FRANCELIN, R. A., GOMIDE, F. A. C. and LOPARO, K., *Notes on system optimization with artificial neural networks*, Internal Report 016/90, UNICAMP, Sept. 1990.

[FRA2 91] FRANCELIN, R. A.; RICARTE, I. and GOMIDE, F., *Parallel implementation of neural networks for optimization problems using a transputer system.* Internal Report 021/91, UNICAMP/FEE/DCA, July 1991.

[FRA3 91] FRANCELIN, R.A.; GOMIDE, F. and LOPARO, K., "System optimization with artificial neural networks", *Proc. Int. Joint Conf. on Neural Neworks - IJCNN'91*, vol. 3, pp. 2639-2644, Nov. 1991.

[FRA4 92] FRANCELIN, R. A.; RICARTE, I. and GOMIDE, F., "System optimization with artificial neural networks: parallel implementation using transputers", *Proc. Int. Joint Conf. on Neural Neworks - IJCNN'92*, vol. 4, pp. 630-635, June, 1992.

[FRA5 92] FRANCELIN, R.A. and GOMIDE, F. A. C., *A neural network to solve discrete dynamic programming problems*, DCA/FEE/UNICAMP, Technical Report no. 016, Aug. 1992.

[FRA6 93] FRANCELIN, R.A. and GOMIDE, F. A. C., "A neural network to solve discrete dynamic programming problems", *Proc. IEEE Int. Conf. Neural Networks - ICNN'93*, realizada de 28/03 a 01/04/93, vol. II, pp. 1433-1438, Mar. 1993.

[FRA7 92] FRANCELIN, R.A. and GOMIDE, F. A. C., *A neural network for fuzzy decision making problems*, DCA/FEE/UNICAMP, Technical Report no. 018, Sept. 1992.

[FRA8 93] FRANCELIN, R.A. and GOMIDE, F. A. C., "A neural network for fuzzy decision making problems", *Proc. 2nd IEEE Int. Conf. on Fuzzy Systems*, realizada de 28/03 a 01/04/93, vol. I, pp. 655-660, Mar. 1993.

[FRA9 93] FRANCELIN, R. A.; GOMIDE, F. A. C. and LANÇAS, F. M., "Use of Artificial Neural Networks for the Classification of Vegetable Oils after GC Analysis", *Chromatographia*, vol. 35, pp. 160-166, Feb. 1993.

[FUKU 91] FUKUMI, M. and OMATU, S., "A new backpropagation algoritm with coupled neuron", *IEEE Trans. on Neural Networks*, vol. 2, no. 5, Sept. 1991.

[FUNA 89] FUNAHASHI, K., "On the approximate realization of continuous mappings by neural networks", *Neural Networks*, vol. 2, no. 3, pp. 183-192, 1989.

[FUKU 88] FUKUSHIMA, K.; MIYAKE, S., and ITO, T., "Neocognitron: neural network model for a mechanism of visual pattern", *IEEE Trans. on Syst., Man, and Cybernetics*, SMC-13, pp. 826-834, 1983.

[FUKU 88] FUKUSHIMA, K., "Neocognitron: a hierarchical neural network capable of visual pattern recognition", *Neural Networks*, vol. 1, pp. 119-130, 1988.

[FUKU] FUKUSHIMA, K.,"Cognitron: a self-organizing multilayered neural network", *Biol. Cybern.*, vol. 20, no. 3/4, pp. 121-136, Nov. 1975.

[GIND 91] GINDI, G., MJOLSNESS, E., and ANANDAN, P., "Neural networks for model based recognition", *Neural Networks - Concepts, Applications, and Implementations*, vol. III, Prentice Hall, NJ, 1991.

[GOM1 92] GOMIDE, F. and ROCHA, A. F., "Neurofuzzy controllers", *Proc. 2nd Int. Conf. on Fuzzy Logic and Neural Networks*, IIZUKA, JAPAN, 17-22, July 1992.

[GOM2 92] GOMIDE, F. A. and ROCHA, A. F., "Neurofuzzy components based on threshold", *Proc. IFAC Symposium on Intelligent Components and Instruments for Control Applications*, SICICA'92, Málaga, Spain, 20-22, pp. 425-430, May 1992.

[GORM 88] GORMAN, R. P. and SEYNOWSKI, T.J., "Analysis of hidden units in a layered network trained to classify sonar targets", *Neural Networks*, vol.1 , no. 1, pp. 75-89, 1988.

[GREF 90] GREFENSTETTE, J.J.; RAMSEY, C.L. and SCHULTZ, A.C., "Learning sequencial decision rules using models and competition", *Machine Learning*, vol. 5, pp. 355-382, 1990.

[GROSS 76] GROSSBERG, S.,"Adaptive pattern classification and universal recoding, I: Parallel development and coding of neural feature detectors", *Biolog. Cybernetics*, vol. 23, pp. 121-134, 1976.

[GROSS 76] GROSSBERG, S., *Studies of mind an brain: neural principles of learning, perception, development, cognition, and motor control*, Boston, MA: Reidel, 1982.

[GUPTA 92] GUPTA, M. M. and QI, J., "On fuzzy neuron models", em Fuzzy Logic for the Management of Uncertainty, Eds. Lotfi A. Zadeh, Janusz Kacprzyk, John Wiley & Sons, Inc., 1992.

[HALE 69] HALE, J.K., *Ordinary Differential Equations*. John Wiley & Sons, New York. 1969.

[HAMM 90] HAMMERSTROM, D., "A VLSI architecture for high-performance, low-cost, on-chip learning", *Proc. the Int. Conf. on Neural Networks*, IJCNN'90, vol. II, San Diego, CA, pp. 537-544, June 1990.

[HAMP 90] HAMPSHIRE, J.B. and WAIBEL, A.H.,"A novel objective function for improved phoneme recognition using time-delay neural networks", *IEEE Trans. Neural Networks*, vol. 1, no. 2, pp. 216-228, June 1990.

[HEBB 49] HEBB, D. O., *The organization of behavior*. John Wiley & Sons, New York. 1949.

[HECH 88] HECHT-NIELSEN, R., "Applications of counterpropagation", *Neural Networks*, vol. I, pp. 131-140, 1988.

[HELD 74] HELD, M., WOLFE, P. and CROWDER, H. D., "Validation of subgradient optimization", *Mathematical Programming*, 6, 1974.

[HELS 90] HELLSTROM, B. J. and KANAL, L. N., *The definition of necessary hidden units in neural networks for combination optimization*, UMIACS-TR-90-6 - Univ. of Maryland at College Park, MD, Jan. 1990

[HINT 89] HINTON, G. E., "Connectionist learning procedures", *Artificial Inteligence*, vol. 40, pp. 185-234, 1989.

[HIRS 74] HIRSCH, M. W. and SMALE, S., *Differential Equations, Dynamical Systems and Linear Algebra*, Academic Press, New York, 1974.

[HOLL 89] HOLLER, M; TAM, S.; CASTRO, H.; and BENSON, R., "An electrically trainable artificial neural network (ETANN) with 10240 floating gate synapses", *Proc. Int. Joint Conf. on Neural Network*, vol. II, Washington, DC, pp. 191-196, June 1989.

[HOPF 82] HOPFIELD, J. J. , "Neural networks and physical systems with emergent collective comparabilities", *Proc. Nath. Acad. Sci. USA*, 79, pp. 2554- 2558, 1982.

[HOPF 84] HOPFIELD, J. J., "Neurons with graded response have collective computational properties like those of two-state neurons", *Proc. Nath. Acad. Sci. USA*, vol. 81, pp. 3088-3092, May 1984.

[HOPF 85] HOPFIELD, J. J. and TANK, D. W. , "Neural computation of decisions in optimization problems", *Biol. Cybern.*, vol. 52, pp. 141 - 152, 1985.

[HOPF 86] HOPFIELD, J. J. and TANK, D. W., "Computing with neural circuits: a model". *Science*, vol. 233, pp. 625- 633, Aug. 1986.

[HORN 89] HORNIK, K., STINCHCOMBE, M., and WHITE, H., " Multilayer feedforward networks are universal approximators", *Neural Networks*, vol. 2, pp. 359-366, 1989.

[HORO 78] HOROWITZ, E. and SAHNI, S., *Fundaments of Computer Algorithms*, 1978.

[HUSH 90] HUSH, R. and HORNE, B., *An introduction to neural networks*, UNM Technical Report no. EECE 90-005, Nov. 1990.

[IRIE 88] IRIE, B. and MIYAKE, S., "Capabilities of three-layered perceptrons", *Proc. 2nd Int Conf. Neural Networks*, vol. 1, pp. 641-648, 1988.

[JACO 88] JACOBS, R. A., "Increased rates of convergence through learning rate adaptation", *Neural Networks*, vol. 1, pp. 295-307, 1988.

[JOHA 90] JOHANSSON, E. M., DOWLA, F. U., and GOODMAN, D. M., *Backpropagation learning for multi-layer feed-forward neural networks using the conjugate gradient method*. UCRL-JC-104850 Preprint, Lawrence Livermore National Laboratory, Sept. 1990.

[JOHN 74] JOHNSON, D., "Approximation algorithms for combinatorial problems",*J. of Computer and Systems Sciences*, 9, 1974.

[JONE 90] JONES, K.L., LUSTIG, I.J., and KORNHAUSER, A. L., "Optimization techniques applied to neural networks: linear search implementation for backpropagation", *Proc. Joint Conf. Neural Networks*, vol. III, San Diego, CA, 1990, pp. 933-939.

[KANA 91] KANAL, L. N., *Automatic pattern recognition*, UMIACS-TR-91-11, Jan. 1991.

[KAMG 87] KAMGAR-PARSI, B. and B. KAMGAR-PARSI, " An efficient model of neural networks for optimization", *Proc. IEEE First Intern. Conf. on Neural Networks*, vol. III, pp. 785-790, 1987.

[KENN 88] KENNEDY, M. P. and CHUA, L. O., "Neural networks for nonlinear programming", *IEEE Trans. on Circ. and Syst.*, vol. 35, no. 5, May 1988.

[KEST 58] KESTEN, H., "Accelerated stochastic approximation", *Annals of Mathematical Statistics*, 29, pp. 41-59, 1958.

[KIM 91] KIM, S., AHN, H. and CHO, S., "Variable target value subgradient method",*Mathematical Programming*, 49, 1991.

[KIRK 70] KIRK, D. E., *Optimal Control Theory - An Introduction*, Prentice Hall, Inc., 1970.

[KIRK 83] KIRKPATRICK, S.; GELLAT, C. D. and VECCHI, M. P., "Optimization by simulated annealing", *Science*, vol. 220, no. 4598, pp. 671-680, May 1983.

[KOHO 84] KOHONEN, T., *Self-Organization and Associative Memory*, Springer-Verlag, Berlin, 1984.

[KOHO 88] KOHONEN, T., "An introduction to neural computing", *Neural Networks*, vol. 1, no. 1, pp. 3-16, 1988.

[KOSK 92] KOSKO, B., *Neural Networks and Fuzzy Systems - A Dynamical Systems Approach to Machine Intelligence*, Prentice Hall International, Inc. Editions, 1992.

[KUH 89] KUH, A. and DICKINSON, B. W., "Information capacity of associative memories", *IEEE Trans. on Inform. Theory*, vol. 35, pp. 59-68, Jan. 1989.

[LARS 67] LARSON, R. E., "A survey of dynamic programming computational procedures", *IEEE Trans. on Automatic Control*, pp. 767-774, December 1967.

[LaSAL 68] LaSALLE, J. P., "Stability theory of ordinary differential equations", J. Differential Equations, 4, 57-65, 1968.

[LASD 70] LASDON, L. S., *Optimization Theory for Large Systems*, Macmillan Series in Operations Research, 1970.

[LeCUN 86] Le CUN, Y., "Learning process in an asymmetric threshold network", *Disordered Systems and Biological Organization*, E. Bienenstock, Ed. New York: Springer-Verlag, pp. 233-240, 1986.

[LEE 91] EE, B. W. and SHEU, B. J., " Modified Hopfield neural networks for retrieving the optimal solution", *IEEE Trans. on Neural Networks*, vol. 2, no. 1, Jan. 1991.

[LEON 90] LEONARD, J. and KRAMER, M. A., "Improvement of the backpropagation algorithm for training neural networks", *Computers Chem. Engng*, vol. 14, no. 3, pp. 337 - 341, 1990.

[LIN 91] LIN, W-C., LIAO, F-Y, TSAO, C-K, and LINGUTLA, T., "A hierarchical multiple-view approach to three-dimensional object recognition", *IEEE Trans. Neural Networks*, vol.2 , pp. 84-92, Jan. 1991.

[LIPP 87] LIPPMAN, R. P., " An introduction to computing with neural nets", *IEEE ASSP Mag.*, vol. 4, no. 2, pp. 4-22, April 1987.

[LUEN 73] LUENBERGER, D. C., *Introduction to linear and nonlinear programming.* Addison-Wesley, 1973.

[MARK 89] MARKS, R. J.; OH, SEHO and ATLAS, L. E., "Alternating projection neural networks", *IEEE Trans. on Circ. and Syst.*, vol. 36, no. 6, June 1989.

[McCUL 43] McCULLOCK, W. S., PITTS, W., "A logical calculus of the ideas immanent in nervous activity", *Bulletin of Mathematical Biophysics*, 5, pp. 115-133. 1943.

[MEAD 89] MEAD, C. A., *Analog VLSI and Neural Systems*, Reading, MA: Addison-Wesley, 1989.

[MEAD 90] MEAD, C. A., "Neuromorphic electronic systems", *Proc. IEEE*, vol. 78, pp. 1629-1636, Oct. 1990.

[MEND 70] MENDEL, J.M. and McLAREN, R.W., "Reinforcement learning control and pattern recognition systems", in *Adaptive, Learning and Pattern Recognition Systems: Theory and Applications*, New York Academic, pp. 287-318, 1970.

[McELI 87] McELIECE, R.J. et al., "The capacity of the Hopfield associative memory", *IEEE Trans. Inform. Theory*, vol. IT-33, no. 4, pp. 461-482, July 1987.

[MEHT 90] MEHTA, S. and FULOP, L., "A Neural algorithm to solve the Hamiltonian cycle problem", *Proc. Int. Joint Conf. on Neural Networks*, vol. III, pp. 843-849, 1990.

[MICH 89] MICHEL, A. N.; FARREL, J. A. and POROD, W., "Qualitative analysis of neural networks", *IEEE Trans. on Circ. and Syst.*, vol. 36, no. 2, Feb. 1989.

[MILL 91] MILLINGTON, P.J., *Associative reinforcement learning for optimal control.* M. S. thesis, Massachusetts Institute of Technology, Tech. Rep. CSDL-T-1070, 1991.

[MINS 69] MINSKY, M. and PAPERT, S., *Perceptrons: an introduction to computational geometry*, Cambridge Mass: MIT Press, 1969.

[NARE 90] NARENDRA, K. S. and PARTHASARATHY, K.,"Identification and control of dynamical systems using neural networks", *IEEE Trans. Neural Networks*, vol.1, pp. 4-27, Mar. 1990.

[NARE 91] NARENDRA, K. S. and PARTHASARATHY, K., "Gradient methods for the optimization of dynamical systems containing neural networks", *IEEE Trans. on Neural Networks*, vol. 2, no. 2, March 1991.

[NEGO 85] NEGOITA, C. V., *Expert systems and fuzzy systems*, The Benjamin/Cunnings Publishing Company, Inc., 1985.

[NEMH 66] NEMHAUSER, G., *Introduction to Dynamic Programming*, John Wiley Sons, Inc., 1966.

[NGUY 90] NGUYEN, D. H. and WIDROW, B., *Neural networks for Self-Learning Control Systems*, IEEE Control Systems Mag., vol 10, no 3, pp. 18-23, April 1990.

[PLAT 87] PLATT, J. and AL BARR., "Constrained differential optimization", *Proc. of Neural Information Processing Systems Conference*, 1987.

[PARK 85] PARKER, D. B., *Learning logic: casting the cortex of the human brain in silicon*, TR-47, M.I.T. Center for Computational Research in Economics and Management Science, Cambridge, MA, Feb. 1985.

[PARIS 91] PARISINI, T. and ZOPPOLI, R., *Backpropagation for N-Stage Optimal Control Problems*, IJCNN'91, vol. 2, pp. 1518-1529, Nov. 1991.

[PENG 89] PENG, Y.; REGGIA, J. A and LI, TAO. "A connectionist approach to vertex covering problems", *Proc. Int. Joint Conf. on Neural Networks*, vol. II, p. 590. June 1989.

[PEDR 91] PEDRYCZ, W., "Neurocomputations in Relational Systems", *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 13, no. 03, pp. 289-297, Mar. 1991.

[POLI 87] POLIAC, M. O.; LEE, E. B.; SLAGLE, J. R. and WICK, M. R., "A Crew scheduling problem", *Proc. Int. Joint Conf. Neural Networks*, vol IV, pp. 779-786, 1987.

[POLJ 69] POLJAK, B. T., *Minimization of unsmooth functionals*, USSR Computational Mathematics and Mathematics Physics, vol. 9, no. 3, 1969.

[PSAL 88] PSALTIS, D., SIDERIS, A. and YAMAMURA, A. A., "A Multilayered Neural Network Controller", *IEEE Control Systems Mag.*, vol. 8, no. 2, pp. 17-21, April 1988.

[RAMA 88] RAMANUJAM, J. and SADAYAPPAN, P., "Optimization by neural networks", *Proc. IEEE Int. Conf. on Neural Networks*, vol. II, pp. 325-332, 1988.

[RANG 89] RANGWALA, S. S. and DORNFELD, D. A., "Learning and optimization of machining operations using computing abilities of neural networks", *IEEE Trans. on Syst., Man, and Cybernetics*, vol. 19, no. 2, March/April 1989.

[REGG 89] REGGIA, J., "Methods for deriving competitive activation mechanisms", *Proc. IJCNN'89*, pp. 357-363, 1989.

[ROCH 92] ROCHA, A. F., "Neural nets: a theory for brains and machines", Lecture Notes in Artificial Intelligence, Springer-Verlag, 1992, 393 pages.

[ROSE 59] ROSENBLATT, F., *Principles of Neurodynamics: Perceptrons and the theory of brain mechanisms*, Washington, DC: Spartan Books, 1959.

[RUME 86] RUMELHART,D. E., HINTON, G. E. and WILLIAMS, R. J., *Learning internal representations by error propagation*. Chap 8 (D. E. Rumelhart and J.L. McClelland, Eds), Cambridge, Massachusets : The MIT Press, 1986.

[SAGE 77] SAGE, A. P. and WHITE, C. C., *Optimum Systems Control*, Prentice-Hall. Inc., 1977.

[SARI 70] SARIDIS, G. N., "Learning applied to sucessive approximation algorithms", *IEEE Trans. on Syst. Sci. and Cybernetics*, SSC-6, pp. 97-103, 1970.

[SART 92] SARTORI, M.A and ANTSAKILS, P.J., "Implementations of learning control systems using neural networks", *IEEE Control Systems*, vol. 12, no. 02, pp. 49-57, 1992.

[SEJN 87] SEJNOWSKI, T. J., ROSENBERG, C. R., "Parallel networks that learn to pronounce english text", *Complex Systems*, vol. 1, pp. 145-168, 1987.

[SIMP 90] SIMPSON, P., *Artificial Neural Systems: Foundations, Paradigms, Applications and Implementations*, Pergamon Press Inc., 1990.

[STOR 87] STORNETTA, W. S. and B. A. HUBERMAN, "An improved three-layer back-propagation algorithm", *IEEE First Int. Conf. Neural Networks*, San Diego, pp. II- 637 - 643, 1987.

[SUTT 86] SUTTON, R. S., "Two problems with back-propagation and other steepest-descent learning procedures for networks", *Proc. Eighth Annual Conf. of the Cognitive Sci. Society*, pp. 823-831, 1986.

[SUTT 92] SUTTON, R. S., BARTO, A. G. and WILLIAMS, R. J., "Reinforcement learning is direct adaptative optimal control", *IEEE Control Systems Mag.*, vol. 12, no. 2, pp. 19-22, April 1992.

[SUTT 92] SUTTON, R.S., Ed. Special Issue on Reinforcement Learning, *Machine Learning*, vol. 8, no. 3/4, 1992.

[TAGL 91] TAGLIARINI, G.A.; CHRIST, J. F. and PAGE, E. W., "Optimization using neural networks", *IEEE Trans. on Computers*, vol. 40, no. 12, Dec. 1991.

[TANK1 86] TANK, D. W. and HOPFIELD, J. J., "Simple neural optimization networks: An A \ D converter, signal decision circuit, and a Linear programming circuit", *IEEE Trans. on Circ. And Syst.*, vol. CAS-33, no. 5, May 1986.

[TANK2 86] TANK, W. D. and HOPFIELD, J. J. ,"Simple neural optimization networks: an A/D converter, signal decision circuit, and a linear programming circuit", *IEEE Trans. on Circ. and Syst.*, vol CAS-33, no.5, May 1986.

[TEIX 92] TEIXEIRA, E.P., *Controle de sistemas não lineares através de redes neurais*. Tese de Doutorado, FEE-UNICAMP, Julho 1991.

[VENK 89] VENKATESH, S. S. and PSALTIS, D., "Linear and logarithmic capacities in associative neural networks", *IEEE Trans. Inform. Theory*, vol. 35, pp. 558-568, May 89.

[WAIB 89] WAIBEL, A.; HANAZAWA, T.; HINTON, G.; SHIKANO, K; LANG, K., "Phoneme recognition using time delay neural networks", *IEEE Trans. on Acoustics, Speech and Signal Processing*, ASSP-37, pp. 328-339, Mar. 1989.

[WATK 89] WATKINS, C.J.C.H., *Learning with delayed rewards*, PH.D. Thesis, Cambridge Univ, Phsychology Dept., 1989.

[WERB 74] WERBOS, P., *Beyond regression: new tools for predicting and analysis in the behavioral sciences*, Ph. D. dissertation, Harvard Univ., 1974.

[WHIT] WHITEHEAD, D. and BALLARD, L.C., "Learning to perceive and act trial and error", *Machine Learning*, to be published.

[WIDR 88] WIDROW, B.; WINTER, R. G. and BAXTER, R. A., "Layered neural nets for pattern recognition", *IEEE Trans. Acoust. Speech*, Signal Process., vol 36, pp. 1109-1118, July 1988.

[WIDR 60] WIDROW, B., HOLF, M. E., *Adaptative Switching Circuits*, 1960 IRE WESCON Con. Record, Part 4, 96, 1960.

[WIDR 83] WIDROW, B., GUPTA, N. K. and MAITRA, S., "Punish/Reward: Learning with a critic in adaptive threshold systems", *IEEE Trans. Syst. Man Cybern.*, Sept. 1983.

[WIDR 86] WIDROW, B., *Adaptive Inverse Control*, in Adaptive Systems in Control and Signal Processing 1986, Intern. Federation of Automatic Control, July 1986.

[WILS 88] WILSON, V. and PAWLEY, G. S., "On the stability of the traveling salesman problem algorithm of Hopfield and Tank", *Biolog. Cybern.*, vol. 58, pp 63-70, 1988.

[YANG 90] YANG, H. and GUEST, C.C., "Linear discriminants, logic functions, backpropagation and improved convergence", *Proc. Int. Joint. Conf. Neural Networks*, vol. III, San Diego, CA, 1990, pp. 287-292.

[YEN 90] YEN, W. and McLEAN, D., "Combining heuristics for optimizing a neural net solution to the traveling salesman problem", *Int. Joint Conf. Neural Networks*, vol. I, pp. 259-264, June 1990.

[ZADE1 65] ZADEH, L. A., "Fuzzy Sets", *Information on Control*, no. 8, pp. 338-353, 1965.

[ZADE2 92] ZADEH, L. A. and KACPRZYK, J. (Eds.), *Fuzzy Logic for the Management of Uncertainty*, John Wiley & Sons, 1992.

[ ] Perihelion Software, *The Helios Operating System*, Prentice-Hall, 1989.