Solução de Problemas de Otimização através de Redes Neurais Multi-Camadas Recorrentes

Luciana Martinez

Orientadora: Roseli Ap. Francelin Romero

DEDALUS - Acervo - ICMSC

303000022428

Dissertação apresentada ao Instituto de Ciências Matemáticas de São Carlos, da Universidade de São Paulo, como parte dos requisitos para obtenção do título de Mestre em Ciências - Área: "Ciência da Computação e Matemática Computacional".

São Carlos - SP

Junho de 1996

Class. -	T
Cutt. -	M379.5
Tombo -	e.1 T578

634012

À Deus, somente a agradecer

Aos meus pais

Agradecimentos

Agradeço aos meus pais, Valdemir e Terezinha, aos meus irmãos, André e Junior, e à minha cunhada Ana, pelo incentivo, apoio e carinho.

Agradeço à professora Dra. Roseli Aparecida Francelin Romero pela orientação, apoio, incentivo e dedicação recebidos durante o desenvolvimento deste trabalho.

Agradeço aos professores Marcos Nereu Arenales e Luíza Fernanda Ribeiro Reis, membros da banca do exame de qualificação, pelas sugestões e críticas dadas no intuito de melhorar a qualidade deste trabalho.

Agradeço aos professores do ICMSC com os quais tive oportunidade de me aperfeiçoar pelas disciplinas cursadas durante o período de graduação e pós graduação, e ao professor Marinho Gomes de Andrade Filho.

Agradeço às secretárias da pós graduação, Beth e Laura, pela disponibilidade e atenção, ao Jacques (do xerox) e a todos os funcionários de maneira geral.

Agradeço às amigas Clébia e Rosângela pela ajuda constante.

Agradeço à amiga Alessandra Fantatto pelo grande incentivo.

Agradeço à Clébia, Rosângela, Célia, Maristela, Sadao, Márcio, Esdras, Patrícia e Verinha pelo convívio e amizade.

Agradeço ao CNPq pelo auxílio financeiro no desenvolvimento deste trabalho.

Enfim, agradeço a todos que de maneira direta ou indireta colaboraram para a realização deste trabalho.

Resumo

Recentemente uma rede neural multi-camadas, baseada no algoritmo *Back-Propagation*, foi proposta para resolver problemas de otimização não lineares. Esta rede tem apresentado bons resultados na solução de problemas não lineares restritos e irrestritos. Este trabalho mostra as facilidades e benefícios da aplicação de técnicas de otimização ao algoritmo de aprendizado desta particular rede neural. O termo *momentum*, o gradiente com busca linear e o método do gradiente conjugado foram incorporados no esquema de aprendizado desta rede neural. Resultados computacionais são apresentados mostrando as vantagens da incorporação destas técnicas nesta rede. Além disso, para verificar a adequabilidade de uso desta particular rede neural, com algumas modificações incorporadas no algoritmo de aprendizado, na solução de problemas de dimensão maior do que os até então testados, uma aplicação é resolvida usando este modelo para otimização de um sistema hidroelétrico de potência. O sistema é constituído de uma usina térmica, uma usina hidroelétrica e com a possibilidade de transferência de energia de sistema vizinho. Resultados são apresentados e mostram a utilidade desta abordagem quando comparada com resultados obtidos por técnicas tradicionais.

Abstract

Recently a neural multi-layer network, based on the *Back-Propagation* algorithm, was proposed for solving nonlinear optimization problems. This network has presented good results in the solution of constrained and unconstrained nonlinear problems. This work shows the feasibility and benefits of applying optimization techniques to learning algorithm in this particular neural network. The *momentum term*, the gradient method and linear search and conjugate gradient method are incorporated into the learning scheme of this neural network. Computation results are presented showing the advantages of incorporating these techniques in this network. Besides that, in order to verify the adequacy of the use of this particular neural network, with some modifications incorporated in the learning algorithm, in solving the problems of bigger dimension than those already tested, an application is solved using this model for optimizing a hydroelectric potential system. The system is constituted of one thermoelectric plant, one hydroelectric plant and with the possibility of transfer of energy from neighbor system. Results are presented and show the usefulness of this approach when compared with the results obtained by traditional techniques.

CONTEÚDO

Lista de Figuras	iii
Lista de Tabelas	iv
Introdução	1
1 Rede Neurais Multi-Camadas e o Algoritmo <i>Back Propagation</i>	4
1.1 Introdução	4
1.2 Redes Neurais Multi-Camadas	4
1.3 Regra Delta Generalizada (RDG)	8
1.4 Algoritmo <i>Back-Propagation</i> (BP)	12
1.5 Aprendizado por Padrão e Aprendizado por Ciclo	14
1.6 Conclusão	15
2 Uma Rede Neural Artificial para Solução de Problemas de Programação Não Linear	16
2.1 Introdução	16
2.2 Problemas de Otimização Irrestritos	17
2.2.1 Rede de 3 Camadas para Solução de Problemas Irrestritos	19
2.2.2 Regra de Atualização dos pesos	20
2.3 Problemas de Otimização Restritos	25
2.3.1 Otimização de Problemas com Restrições de Igualdade	26
2.4 Algoritmo <i>Back Propagation*</i> (BP*)	31
2.5 Conclusão	33

3 Extensões do Algoritmo <i>Back Propagation</i> *	34
3.1 Introdução	34
3.2 Técnicas de Otimização	34
3.2.1 O termo <i>momentum</i>	35
3.2.2 Gradiente Simples com Busca Linear	38
3.2.2.1. Regra de Armijo	40
3.2.2.2. Método da Falsa Posição	40
3.2.3 Gradiente Conjugado	44
3.3 Modificações na Topologia da rede	50
3.3.1 (2n) Neurônios na Camada Intermediária	50
3.3.2 Modificações no Algoritmo de Aprendizado	53
3.3.2.1. Algoritmo BP* Modificado	55
3.4 Problemas sem Solução e Problemas com Várias Soluções	57
3.5 Conclusão	59
 4 Solução de um Sistema Hidroelétrico de Potência	 60
4.1 Introdução	60
4.2 Apresentação do Problema	60
4.3 Solução via Técnicas de Programação Não-Linear	62
4.3.1 Esquema Geral de Solução	65
4.4 Solução via Redes Neurais	66
4.5 Dados e Funções Adotados	70
4.6 Resultados Obtidos	71
4.7 Discussão dos Resultados obtidos	72
4.8 Conclusão	74
 5 Conclusões e Trabalhos Futuros	 75
 Bibliografia	 79

Lista de Figuras

Figura 1.1 Uma rede neural com 4 camadas	6
Figura 2.1 Rede neural para resolver problemas de otimização irrestritos	20
Figura 3.1 Topologia da rede neural para resolver problemas restritos	29
Figura 4.1 Esquema geral de solução para o problema (P)	66
Figura 4.2 Rede neural para resolver o problema (P)	68

Lista de Tabelas

Tabela 2.1 Métodos de otimização e suas respectivas direções	18
Tabela 2.2 Resultados obtidos para o problemas (P-1)	24
Tabela 2.3 Resultados obtidos para o problemas (P-2)	24
Tabela 2.4 Resultados obtidos para o problemas (P-3)	30
Tabela 2.5 Resultados obtidos para o problemas (P-4)	30
Tabela 3.1 Resultados obtidos para o problemas (P-1) com o algoritmo BP* com <i>termo momentum</i> $\mu = 0.9$	36
Tabela 3.2 Resultados obtidos para o problema (P-2) com o algoritmo BP* com <i>termo momentum</i> $\mu = 0.9$	37
Tabela 3.3 Resultados obtidos para o problemas (P-3) com o algoritmo BP* com <i>termo momentum</i> $\mu = 0.9$	37
Tabela 3.4 Resultados obtidos para o problemas (P-4) com o algoritmo BP* com <i>termo momentum</i> $\mu = 0.9$	37
Tabela 3.5 Resultados obtidos para o problemas (P-1) com o algoritmo BP* com busca linear	42
Tabela 3.6 Resultados obtidos para o problemas (P-2) com o algoritmo BP* com busca linear	42
Tabela 3.7 Resultados obtidos para o problemas (P-3) com o algoritmo BP* com busca linear	43
Tabela 3.8 Resultados obtidos para o problemas (P-4) com o algoritmo BP* com busca linear	43
Tabela 3.9 Resultados obtidos para o problemas (P-1) com o algoritmo BP* implementando o método do gradiente conjugado	46
Tabela 3.10 Resultados obtidos para o problemas (P-2) com o algoritmo BP* implementando o método do gradiente conjugado	47
Tabela 3.11 Resultados obtidos para o problemas (P-3) com o algoritmo BP* implementando o método do gradiente conjugado	48

Tabela 3.12 Resultados obtidos para o problemas (P-4) com o algoritmo BP*implementando o método do gradiente conjugado	49
Tabela 3.13 Resultados obtidos para o problemas (P-1) com $2n$ neurônios na camada intermediária	51
Tabela 3.14 Resultados obtidos para o problemas (P-2) com $2n$ neurônios na camada intermediária	51
Tabela 3.15 Resultados obtidos para o problemas (P-3) com $2n$ neurônios na camada intermediária	52
Tabela 3.16 Resultados obtidos para o problemas (P-4) com $2n$ neurônios na camada intermediária	52
Tabela 3.17 Resultados obtidos pelo algoritmo BP* modificado	56
Tabela 3.18 Resultados obtidos para o problema sem solução	57
Tabela 3.19 Resultados obtidos para o problema com mais de uma solução	58
Tabela 4.1 Vazão afluente ao reservatório e demanda de carga em cada período t	70
Tabela 4.2 Resultados obtidos para o problema (P) usando técnicas de Programação Não Linear	71
Tabela 4.3 Resultados obtidos para o problema (P) via rede neural e transformados para o intervalo real do problema	72
Tabela 4.4 Valor da função objetivo	72

INTRODUÇÃO

Nos últimos anos, modelos de Redes Neurais Artificiais vem sendo aplicados a uma ampla variedade de problemas de engenharia e otimização, como por exemplo, no reconhecimento de padrões, no processamento de imagens e sons, em controle ótimo, na previsão de séries temporais ou ainda em problemas de Programação Linear, problemas complexos como o Problema do Caixeiro Viajante (rede de Hopfield) e problemas de Programação Não-Linear.

Estes modelos são compostos de elementos computacionais (neurônios) arranjados em padrões semelhantes a redes neurais biológicas.

Sua estrutura em paralelo, e sua habilidade para aprender, e portanto generalizar, possibilitam o uso de Redes Neurais Artificiais na resolução de problemas complexos (grande escala) que são muitas vezes intratáveis por outras abordagens, uma vez que ao invés de executar um programa de instrução seqüencialmente, estes modelos neurais exploram várias hipóteses competindo simultaneamente em paralelo [5].

Desde o surgimento de algoritmos de aprendizado em redes neurais, uma vasta variedade de melhoramento das técnicas para treinamento tem sido proposta, como é o caso do estudo do efeito do uso de ferramentas da teoria de otimização em algoritmos de redes neurais. Neste caso, resultados favoráveis têm sido obtidos, mostrando que algoritmos de redes neurais incorporando técnicas de otimização podem ser melhorados tanto em tempo de aprendizado (velocidade de convergência) quanto em qualidade de solução.

Jones et al. [7], por exemplo, provaram os benefícios e facilidades do uso da teoria de otimização em um particular algoritmo de redes neurais, o algoritmo de aprendizado *Back-Propagation*, pela incorporação de um método de busca linear ao seu aprendizado. Simulações feitas mostraram que melhores resultados e mais rápida convergência foram obtidos pelo algoritmo *Back-Propagation* com busca linear comparado com o *Back-Propagation* sem a busca. Em [17], Ridmiller apresenta uma seleção das mais elaboradas técnicas (locais e globais) adaptáveis ao aprendizado supervisionado (em particular ao *Back-Propagation*). Dentre as técnicas globais apresentadas, é dado destaque ao algoritmo gradiente conjugado. Noyes [15], por sua vez, apresenta uma análise do desempenho de métodos quasi Newton e o método do gradiente conjugado em tarefas de aprendizado de redes neurais.

Recentemente uma rede neural artificial Multi-Camadas com realimentação foi proposta por Francelin [3] para a solução de problemas de otimização convexos e irrestritos, cujo esquema de atualização de pesos é baseado no algoritmo *Back-Propagation*.

Francelin também propôs uma generalização desta rede, baseada na teoria da dualidade [10], visando a solução de problemas convexos restritos.

A topologia da rede é constituída de três camadas com exatamente n neurônios em cada camada, onde n é a dimensão do problema a ser resolvido. Esta rede, tal como proposta em [3], implementa o método do gradiente simples.

Em virtude do presente trabalho estar baseado neste modelo de rede neural, inicialmente uma descrição desta rede neural é realizada. Em seguida, com o objetivo de investigar o comportamento desta rede neural particular com a incorporação de técnicas de otimização ao seu algoritmo de aprendizado, algumas modificações são incorporadas, tais como: *termo momentum*, gradiente com busca linear, gradiente conjugado, além de modificações em sua estrutura. Resultados computacionais são apresentados.

Além disso, para verificar o desempenho da rede neural apresentada na otimização de problemas mais complexos do que os até então testados, um problema de otimização referente a um sistema hidroelétrico de potência é resolvido. Com base no esquema de solução proposto por Soares [19] para este problema, duas abordagens foram então utilizadas na resolução do mesmo: técnicas clássicas de Programação Não Linear e a abordagem neural apresentada inicialmente. Ambas abordagens foram implementadas neste trabalho e os resultados obtidos são apresentados juntamente com uma análise comparativa referente ao desempenho das mesmas.

Assim sendo, este trabalho é organizado da seguinte forma:

O Capítulo 1 apresenta alguns conceitos gerais sobre redes neurais artificiais Multi-Camadas, tais como: organização, esquema de aprendizado e esquema de treinamento. É realizada também, uma descrição do algoritmo *Back-Propagation* e a derivação da Regra Delta Generalizada baseada em Rumelhart [18].

Em seguida, no Capítulo 2, alguns conceitos e resultados gerais da teoria de otimização são revisados. O modelo de rede neural artificial constituída de três camadas, propriamente dito, para resolver problemas de otimização irrestritos e restritos, convexos e contínuos é apresentado, juntamente com a sua correspondente Regra de Aprendizado.

No Capítulo 3, algumas modificações são discutidas e incorporadas ao esquema de aprendizado da rede neural descrita no Capítulo 2. Alguns resultados computacionais são apresentados a fim de mostrar o desempenho do modelo apresentado com as modificações realizadas.

O Capítulo 4 apresenta um sistema hidroelétrico de potência e duas abordagens utilizadas na sua resolução: técnicas clássicas de Programação Não Linear e a rede neural apresentada no Capítulo 2. Os resultados obtidos são comparados e uma discussão sobre esses resultados é também apresentada.

Finalmente, no Capítulo 5, são feitas conclusões gerais sobre os métodos apresentados nos capítulos anteriores e uma análise do comportamento da rede neural em discussão, na solução do sistema de potência apresentado.

CAPÍTULO 1

Redes Neurais Multi-Camadas e o Algoritmo *Back-Propagation*

1.1 Introdução

Este capítulo apresenta alguns conceitos gerais sobre redes neurais artificiais Multi-Camadas, uma descrição do algoritmo *Back-Propagation* e a derivação da Regra Delta Generalizada.

1.2 Redes Neurais Multi-Camadas

Uma rede neural artificial pode ser definida basicamente como sendo um sistema de interconexão de unidades (chamadas neurônios), baseada no sistema nervoso, que atua após um estímulo externo.

A estrutura de interconexão dos neurônios é denominada de arquitetura da rede neural e apresenta variações quanto ao número de camadas da rede, número de neurônios em cada camada, função de transferência de informação de um neurônio para outro.

Em uma Rede Neural Multi-Camadas (RNMC) a organização das unidades é feita em várias camadas:

- camada de entrada: cujas unidades pertencentes a esta camada são denominadas unidades de entrada.
- camada de saída: cujas unidades pertencentes a esta camada são denominadas unidades de saída.

Podem existir, ainda, inúmeras camadas, entre as camadas de entrada e saída. Estas camadas são denominadas camadas intermediárias. Se a unidade pertence a uma camada intermediária ela é chamada de unidade intermediária.

As unidades são conectadas por arcos e a cada arco está associado um peso (w). A comunicação entre as unidades é feita através desta interconexão dos pesos. Não há, entretanto, comunicação entre unidades de uma mesma camada. As unidades recebem sua entrada da camada inferior à sua e enviam sua saída para a camada superior à sua.

A Figura 1.1 mostra uma rede Multi-Camadas, onde a camada de entrada é a camada inferior da figura e a camada de saída é a camada superior da figura. Neste exemplo a rede possui somente duas camadas intermediárias. Esta rede, denominaremos aqui, como uma rede Multi-Camadas constituída de 4 camadas. Entretanto, alguns autores não consideram a camada de entrada da rede como camada. Para estes autores a rede da Figura 1.1 é considerada uma rede Multi-Camadas de 2 camadas.

Figura 1.1: Uma rede neural com 4 camadas.

A rede Multi-Camadas é usada, geralmente, para estabelecer uma relação entre dois conjuntos de dados. A cada entrada fornecida a rede, uma correspondente saída desejada deve também ser fornecida. Ao conjunto que contém todos os pares de entrada/saída de uma rede dá-se o nome de conjunto de treinamento. Desta forma, o conjunto T de treinamento é dado por:

$$T = \{(x_0, t_0), (x_1, t_1), \dots, (x_n, t_n)\}$$

onde:

x_j : j -ésimo vetor entrada.

t_j : j -ésimo vetor de saída desejada.

Como as saídas desejadas são fornecidas à rede, o aprendizado é do tipo supervisionado [18].

Este tipo de aprendizado consiste basicamente da adaptação iterativa da conexão dos pesos, que transporta informações entre simples unidades de processadores. As variáveis, neste caso, são os pesos (w) correspondentes às interconexões existentes entre as várias camadas. É importante, entretanto, observar que nem todos os pesos precisam ser necessariamente variáveis. Qualquer número de pesos pode ser fixado. Neste caso os pesos fixos não são modificados durante todo o aprendizado.

As unidades de entrada recebem um vetor ativação externo que será propagado através das outras camadas da rede. O esquema de aprendizado da rede consiste em utilizar a entrada fornecida para produzir sua própria saída. A saída produzida pela rede é comparada com a saída desejada fornecida. Se as saídas forem aproximadamente iguais então diz-se que ocorreu o aprendizado. Caso contrário os pesos serão modificados com a finalidade de diminuir esta diferença entre a saída desejada e a saída produzida pela rede. Após o treinamento dos pesos, quando uma entrada for apresentada à rede, sua saída resultante deverá ser igual, ou muito próxima, da saída desejada [3].

Segundo Ridmiller [17], aprendizado supervisionado é provavelmente a mais freqüente técnica usada no campo de redes neurais.

O algoritmo *Back-Propagation*, introduzido em 1986 por Rumelhart e Mc Clelland [18] e muito usado no treinamento de RNMC, tem como base para a atualização dos pesos a Regra Delta Generalizada, a qual implementa o método do gradiente simples.

A seguir temos a derivação da Regra Delta Generalizada baseada em Rumelhart [18].

1.3 Regra Delta Generalizada (RDG)

O sistema de aprendizado da rede consiste em alterar os pesos da rede a fim de diminuir a diferença entre as saídas produzidas pela rede e as saídas fornecidas. O erro quadrático referente a cada par p apresentado à rede pode ser definido como:

$$E_p = \frac{1}{2} \sum_j (t_{pj} - u_{pj})^2 \quad (1.1)$$

onde:

t_{pj} : j -ésima componente do vetor de saída desejada.

u_{pj} : j -ésima componente do vetor saída obtido pela rede.

E , portanto, o erro total do sistema é dado pela soma do erro quadrático de todos os pares apresentados à rede, ou seja:

$$E = \sum_p E_p$$

O problema de otimização associado ao esquema de aprendizado pode ser formulado como:

$$\min_{w_{ij}} E_p = \min_{w_{ij}} \left\{ \frac{1}{2} \sum_j (t_{pj} - u_{pj})^2 \right\} \quad (1.2)$$

Como função de ativação de cada unidade será usada a função não linear conhecida como função logística ou *sigmoid*, que é uma função não decrescente e diferenciável da forma:

$$u_{pj} = g_j(\text{net}_{pj}) = \frac{1}{1 + e^{-\text{net}_{pj}}} \quad (1.3)$$

onde:

u_{pj} : j-ésima componente do p-ésimo vetor de saída.

$\text{net}_{pj} = \sum_i w_{ij} \cdot u_{pi}$: j-ésima componente do p-ésimo vetor de entrada.

Para verificarmos que a RDG implementa o método do gradiente simples, devemos mostrar que a derivada do erro em relação a cada peso é proporcional a mudança de peso ditada pela RDG, com constante de proporcionalidade negativa, isto é, devemos mostrar que:

$$\Delta_p w_{ij} = -\eta \frac{\partial E_p(w)}{\partial w_{ij}}$$

De acordo com Francelin [3], a RGD é dada por:

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

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

$$\delta_{pj} = (t_{pj} - u_{pj}) \cdot u_{pj} \cdot (1 - u_{pj}) \quad (1.4)$$

e, para uma unidade intermediária, é dado por:

$$\delta_{pj} = u_{pj} \cdot (1 - u_{pj}) \cdot \sum_k \delta_{pk} \cdot w_{kj} \quad (1.5)$$

Assim, devemos mostrar que:

$$\delta_{pk} \cdot u_{pi} \sim \frac{\partial E_p(w)}{\partial w_{ij}}$$

Usando a Regra da Cadeia para o cálculo da derivada do erro em relação a cada peso temos:

$$\frac{\partial E_p}{\partial w_{ij}} = \frac{\partial E_p}{\partial \text{net}_{pj}} \cdot \frac{\partial \text{net}_{pj}}{\partial w_{ij}} \quad (1.6)$$

onde:

$$\frac{\partial \text{net}_{pj}}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \left(\sum_k w_{ik} u_{pk} \right) = u_{pj}$$

e aplicando novamente a Regra da Cadeia para o fator $\frac{\partial E_p}{\partial \text{net}_{pj}}$ da equação (1.6):

$$\frac{\partial E_p}{\partial \text{net}_{pj}} = \frac{\partial E_p}{\partial u_{pj}} \cdot \frac{\partial u_{pj}}{\partial \text{net}_{pj}} = \frac{\partial E_p}{\partial u_{pj}} \cdot g'(\text{net}_{pj})$$

$$= \frac{\partial E_p}{\partial u_{pj}} u_{pj} (1 - u_{pj}) \quad (1.7)$$

Para o cálculo do fator $\frac{\partial E_p}{\partial u_{pj}}$ da equação (1.7) dois casos devem ser considerados:

1) u_{pj} pertence a uma unidade de saída da rede.

Neste caso, pela definição de E_p tem-se:

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

Desta forma:

$$\frac{\partial E_p}{\partial \text{net}_{pj}} = -(t_{pj} - u_{pj}) \cdot u_{pj} \cdot (1 - u_{pj}) = -\delta_{pj}$$

e, de acordo com a equação (1.4):

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

2) u_{pj} pertence a uma unidade intermediária da rede.

Tem-se, neste caso, usando novamente a Regra da Cadeia:

$$\begin{aligned} \frac{\partial E_p}{\partial u_{pj}} &= \sum_k \frac{\partial E_p}{\partial \text{net}_{pk}} \cdot \frac{\partial \text{net}_{pk}}{\partial u_{pj}} = \sum_k \frac{\partial E_p}{\partial \text{net}_{pk}} \cdot w_{jk} \\ &= \sum_k -\delta_{pk} w_{jk} \end{aligned}$$

Portanto:

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

e, de acordo com a equação (1.5) tem-se:

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

concluindo, assim, a demonstração.

A aplicação da RDG, portanto, envolve duas fases. Na primeira fase, a entrada é apresentada e propagada para frente (*forward*) através da rede, para calcular a saída u_{pj} de cada unidade. Esta saída é comparada, então, com a saída desejada, resultando em um 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. Desta forma, pode-se dizer que no sentido *forward* a rede avalia a função $E_p(w)$ e no sentido *backward* o seu gradiente [3].

Este esquema de atualização dos pesos juntamente com a aplicação da RDG é conhecido como algoritmo *Back-Propagation* (BP). Redes neurais que usam a RDG são conhecidas como redes Multi-Camadas *feed-forward*.

Taxa 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 denominada taxa de aprendizado ou velocidade de aprendizado. Quanto maior é esta taxa de aprendizado, maiores são as modificações nos pesos.

Rumelhart [18] sugere que se adote o maior valor possível para a taxa de aprendizado, que não provoque oscilações.

1.4 Algoritmo *Back-Propagation* (BP)

O algoritmo BP para uma rede neural artificial de 3-Camadas (apenas 1 camada intermediária) é descrito a seguir.

Sejam:

i: camada de entrada.

h: camada intermediária.

o: camada de saída.

w_{ih} : matriz de pesos entre a camada de entrada e a camada intermediária.

w_{ho} : matriz de pesos entre a camada intermediária e a camada de saída.

$x_k^T = \{x_1, x_2, \dots, x_k\}$: vetor de entrada da camada k.

$y_k^T = \{y_1, y_2, \dots, y_k\}$: vetor de saída da camada k.

$t_o^T = \{t_1, t_2, \dots, t_o\}$: vetor saída desejada.

$g(x) = \frac{1}{1 + e^{-x}}$: função de ativação

Passo Forward:

$$y_i = g(x_i)$$

$$(x_h)^T = (w_{ih})^T \cdot y_i$$

$$y_h = g(x_h)$$

$$(x_o)^T = (w_{ho})^T \cdot y_h$$

$$y_o = g(x_o)$$

Cálculo do erro:

$$\varepsilon = \frac{1}{2} \|t_o - y_o\|$$

Passo Backward:

$$\delta_o = (t_o - y_o) \cdot g'(x_o)$$

$$w_{ho} = w_{ho} + \eta \cdot \delta_o \cdot y_h$$

$$\delta_h = g'(x_h) \cdot (w_{ho})^T \cdot \delta_o$$

$$w_{ih} = w_{ih} + \eta \cdot \delta_h \cdot y_i$$

Algoritmo *Back-Propagation***Início**dado (x^0)**Enquanto** não ocorrer o aprendizado **faça****Início**executar o Passo *Forward*.**Se** $\varepsilon \leq$ limite erro **então**

ocorreu o aprendizado

Senãoexecutar o Passo *Backward*.**Fim do enquanto****Fim****1.5 Aprendizado por Padrão e Aprendizado por Ciclo**

Segundo Ridmiller [17], existem basicamente dois possíveis métodos para se calcular e executar a atualização dos pesos em um algoritmo de aprendizado de uma rede

neural artificial, dependendo de quando a atualização é executada: aprendizado por padrão e aprendizado por ciclo.

O aprendizado por padrão atualiza os pesos da rede depois de cada apresentação do par padrão (entrada/saída) e do cálculo do respectivo gradiente, isto é, a atualização é feita baseada nos gradientes parciais $\left(\frac{\partial E_p}{\partial w}\right)$ de cada padrão. Este tipo de método, também conhecido como “aprendizado *on-line*” ou “aprendizado estocástico”, apresenta melhores resultados para grandes conjuntos padrões, contendo quantias significantes de informações redundantes.

Já o método conhecido como aprendizado por ciclo, ou “aprendizado *batch*“, primeiramente soma informações do gradiente sobre o conjunto padrão total e então executa a atualização dos pesos, ou seja, o aprendizado utiliza-se do gradiente total $\left(\frac{\partial E}{\partial w} = \sum_p \left(\frac{\partial E_p}{\partial w}\right)\right)$ no esquema de atualização dos pesos. Por este motivo, este método contém informações mais seguras com respeito a forma da função erro total.

1.6 Conclusão

Neste capítulo, um modelo de Rede Neural Multi-Camadas foi apresentado. Foi mostrado como ocorre o processo de aprendizado neste modelo e que, na verdade, este processo consiste na solução de um Problema de Otimização através da técnica do gradiente simples.

CAPÍTULO 2

Uma Rede Neural Artificial para Solução de Problemas de Programação Não-Linear

2.1 Introdução

Neste capítulo, um modelo de rede neural artificial Multi-Camadas proposto para resolver Problemas de Programação Não Linear é apresentado juntamente com sua Regra de Aprendizado. O entendimento deste modelo é extremamente importante para o desenvolvimento deste trabalho. Antes, porém, são apresentadas algumas definições necessárias, de acordo com [3].

2.2 Problemas de Otimização Irrestritos

Um problema de otimização pode ser colocado na seguinte forma:

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

onde f é uma função real e o conjunto Ω , denominado conjunto das soluções factíveis, é um subconjunto do $\mathbb{R}^n$. Para $\Omega = \mathbb{R}^n$ o problema é considerado um *problema de otimização irrestrito*.

Existem vários métodos iterativos para resolver o problema (2.1). Todos estes métodos tem como estrutura fundamental:

- seleccionar uma solução inicial x_0 .
- determinar uma direcção de movimento d_k , partindo-se de x_k , que garanta uma diminuição na função.
- determinar o tamanho do passo γ_k , na direcção d_k , a fim de que nesta direcção seja obtida uma diminuição suficiente na função objetivo, ou seja,

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

- atualizar a solução de acordo com

$$x_{k+1} = x_k + p_k \quad \text{onde } p_k = \gamma_k d_k$$

A principal diferença entre estes métodos está na forma pela qual direcções sucessivas de movimento são seleccionadas. A Tabela 2.1 apresenta alguns destes métodos e suas respectivas direcções de descida.

Métodos	Direções
Gradiente Simples	$d_k = -\nabla f(x_k^*)$
Gradiente Conjugado	$d_k = -\nabla f(x_k^*) + b_k d_{k-1}$
Quase-Newton	$d_k = -S(x_k) \nabla f(x_k^*)$
Newton	$d_k = -[H(x_k)]^{-1} \nabla f(x_k^*)$

Tabela 2.1: Métodos de otimização e suas respectivas direções.

onde:

d_k : vetor direção na iteração k ,

$\nabla f(x) = \left(\frac{\partial f(x)}{\partial x_i} \right)_{i=1,2,\dots,n}$: vetor gradiente da função f ,

$b_k = \frac{\nabla f(x_{k+1}) \cdot H \cdot d(x_{k+1})}{d(x_k)^T \cdot H \cdot d(x_k)}$: valor calculado a cada passo k ,

$S(x) \cong [H(x)]^{-1}$: aproximação para a inversa da Hessiana e

$H(x) = \left(\frac{\partial^2 f(x)}{\partial x_i \partial x_j} \right)_{i,j=1,2,\dots,n}$: matriz Hessiana $n \times n$ da função f .

Em geral todos estes métodos para resolução de (2.1) encontram um ponto de mínimo local x^* de maneira que $f(x^*) \leq f(x)$ para todo x em uma vizinhança de x^* . Se x^* é um ponto de mínimo local de $f(x)$ então a norma de $\nabla f(x)$ é zero e $H(x)$ é definida semi-positiva. É possível que existam vários mínimos locais. Somente condições adicionais em $f(x)$, como convexidade, garantem que este mínimo é também global. Informações mais detalhadas sobre esses métodos podem ser encontradas, por exemplo, em [10].

2.2.1 Rede de 3 Camadas para Solução de Problemas Irrestritos

A rede neural artificial proposta por Francelin [3] para encontrar o valor $x^* \in \mathbb{R}^n$ que resolve o problema (2.1) é uma rede constituída de 3 camadas com realimentação da saída. As justificativas para a escolha da mesma se encontram em [3].

A rede apresenta exatamente n processadores (neurônios) em cada camada, onde n é a dimensão do problema a ser resolvido. A primeira camada contém as entradas x da rede e a última camada, as saídas y . A inicialização dos pesos da rede é feita aleatoriamente no intervalo $[-1,1]$ e uma solução inicial $x_0 = (x_1^0, x_2^0, \dots, x_n^0)$ é fornecida como vetor de entrada. Um vetor de saída é então obtido e realimentado para a camada de entrada. Esse processo é repetido até que um determinado teste de otimalidade seja satisfeito.

A Figura 2.1 mostra a topologia de rede considerada. A Camada-0 e a Camada-2 são respectivamente, a camada de entrada e saída da rede. A Camada-1 é a camada intermediária. Os pesos entre a Camada-0 e a Camada-1 são representados por $w_{ij}^{(1)}$ e os pesos entre a Camada-1 e a Camada-2 são representados por $w_{ij}^{(2)}$.

A função de ativação sigmoid, adotada nesse caso, é dada por:

$$u_j^{(k)} = g(\text{net}_j^{(k)}) = \frac{1}{1 + e^{-\text{net}_j^{(k)}}} \quad k=0,1,2 \quad (2.2)$$

onde:

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

$\text{net}_j^{(k)} = \sum_i w_{ij}^{(k)} u_i^{(k-1)}$: j -ésima componente do vetor de entrada da Camada- k .

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

Figura 2.1: Rede neural para resolver problemas de otimização irrestritos.

Desde que as saídas da rede são dadas por (2.2), serão considerados somente problemas de otimização que estejam contidos no intervalo $[0,1]$. Portanto, problemas com solução fora deste intervalo deveram ser escalonados.

2.2.2 Regra de Atualização dos Pesos

Como já visto no Capítulo 1, o esquema de aprendizado em RNMC baseado na RDG permite a solução do problema:

$$\min_{w_{ij}} E_p = \frac{1}{2} \sum_j (t_{pj} - u_{pj})^2 \quad (2.3)$$

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

No caso de problemas gerais de otimização não se conhece, no entanto, a saída desejada, pois esta depende da solução ótima que se quer obter. Portanto, a aplicação direta do algoritmo BP certamente não é apropriada para este tipo de generalização. Desta forma, uma nova função erro $E(x)$ foi definida de acordo com [3]:

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

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

A regra de atualização dos pesos para este modelo foi obtida de forma análoga à RDG, ou seja,

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

O que é equivalente, no caso do problema (2.1), a:

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

Dois casos foram considerados para o cálculo da derivada do erro com relação aos pesos:

(a) Unidades pertencentes à Camada-2:

Pela Regra da Cadeia, obtém-se:

$$\frac{\partial f}{\partial w_{ij}^{(2)}} = \frac{\partial f}{\partial \text{net}_j^{(2)}} \cdot \frac{\partial \text{net}_j^{(2)}}{\partial w_{ij}^{(2)}} = \frac{\partial f}{\partial \text{net}_j^{(2)}} \cdot u_i^{(1)}$$

Colocando

$$\frac{\partial f}{\partial \text{net}_j^{(2)}} = -\delta_j^{(2)}$$

tem-se que:

$$-\delta_j^{(2)} = \frac{\partial f}{\partial \text{net}_j^{(2)}} = \frac{\partial f}{\partial u_j^{(2)}} (u^{(2)}) \cdot \frac{\partial u_j^{(2)}}{\partial \text{net}_j^{(2)}} = \frac{\partial f}{\partial u_j^{(2)}} (u^{(2)}) \cdot g'(\text{net}_j^{(2)})$$

e, portanto:

$$\delta_j^{(2)} = -\frac{\partial f}{\partial u_j^{(2)}} (u^{(2)}) \cdot g'(\text{net}_j^{(2)}) \quad (2.4)$$

(b) Unidades pertencentes à Camada-1:

Novamente pela Regra da Cadeia, obtém-se:

$$\frac{\partial f}{\partial w_{ij}^{(1)}} = \frac{\partial f}{\partial \text{net}_j^{(1)}} \cdot \frac{\partial \text{net}_j^{(1)}}{\partial w_{ij}^{(1)}} = \frac{\partial f}{\partial \text{net}_j^{(1)}} \cdot u_i^{(0)}$$

Analogamente colocando:

$$\frac{\partial f}{\partial \text{net}_j^{(1)}} = -\delta_j^{(1)}$$

tem-se que:

$$\begin{aligned} -\delta_j^{(1)} &= \frac{\partial f}{\partial \text{net}_j^{(1)}} = \frac{\partial f}{\partial u_j^{(1)}} \cdot \frac{\partial u_j^{(1)}}{\partial \text{net}_j^{(1)}} = \left(\sum_k \frac{\partial f}{\partial \text{net}_k^{(2)}} \cdot \frac{\partial \text{net}_k^{(2)}}{\partial u_j^{(1)}} \right) \cdot g'(\text{net}_j^{(1)}) \\ &= \left(\sum_k (-\delta_k^{(2)}) \cdot w_{jk}^{(2)} \right) \cdot g'(\text{net}_j^{(1)}) \end{aligned}$$

e, portanto:

$$\delta_j^{(1)} = \left(\sum_k \delta_k^{(2)} \cdot w_{jk}^{(2)} \right) \cdot g'(\text{net}_j^{(1)}) \quad (2.5)$$

Calculando $\delta_j^{(1)}$ e $\delta_j^{(2)}$ por (2.4) e (2.5) respectivamente, obtém-se:

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

onde η é um valor pequeno e constante no intervalo $[0,1]$.

Critério de Parada: O processo realizado pela rede neural proposta acima converge (estabiliza) quando:

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

Portanto, através deste esquema, as saídas da rede caminham sempre em direção oposta ao gradiente da função f , $-\frac{\partial f}{\partial w_{ij}}$, e os pesos (w) são atualizados de acordo com o método do gradiente simples:

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

e, quando

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

os pesos se estabilizaram, isto é,

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

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

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

Maiores detalhes sobre a obtenção desta Regra de Atualização dos Pesos, podem ser obtidos em [1], [2] e [3].

Os exemplos 1 e 2 ilustram o comportamento da rede da Figura 2.1. Os testes foram realizados num computador 486 DX4, utilizando a linguagem C.

Exemplo 1: Consideremos o problema (P-1):

$$\min (x_1 - 0.1)^2 + (x_2 - 0.2)^2 + (x_3 - 0.3)^2 + (x_4 - 0.4)^2 \quad (\text{P-1})$$

As soluções aproximadas x obtidas pela rede são mostradas na Tabela 2.2, considerando diferentes soluções iniciais para x^0 , ε (limitante para o erro) e η (taxa de aprendizado) e considerando pesos gerados aleatoriamente em $[-1,1]$.

$(x^0)^T$	ε	η	$(x)^T$				tempo em s	iterrações	função objetivo
.2, .15, .45, .6	10^{-2}	0.5	0.104930	0.199712	0.29700	0.399910	0.989	230	0.010007
	10^{-2}	0.75	0.104962	0.199713	0.29970	0.399911	0.659	152	0.010007
	10^{-3}	0.75	0.100498	0.199968	0.299973	0.399992	1.319	289	0.009999
.4, .32, .22, .11	10^{-3}	0.5	0.100494	0.199968	0.299973	0.399992	1.578	437	0.009999
	10^{-3}	0.75	0.100491	0.19969	0.299973	0.399992	1.319	290	0.009999
	10^{-2}	0.3	0.104974	0.199709	0.299695	0.399909	1.758	384	0.010007
	10^{-3}	0.7	0.100491	0.199969	0.29973	0.399992	1.374	311	0.009999

Tabela 2.2: Resultados obtidos para o problema (P-1).

Podemos observar que os resultados obtidos são bons, desde que a solução ótima do problema é $x^* = (0.1, 0.2, 0.3, 0.4)^T$.

Exemplo 2: Consideremos, agora, o problema (P-2):

$$\min (x_1 - 0.5)^4 + (x_1 - 2x_2)^2 \quad (\text{P-2})$$

Neste caso, as soluções aproximadas x obtidas são mostradas na Tabela 2.3, considerando diferentes soluções iniciais para x^0 , ε e η .

$(x^0)^T$	ε	η	$(x)^T$		tempo em s	iterrações	função objetivo
.1, .2	10^{-2}	0.5	0.627910	0.315096	0.330	79	0.000273
	10^{-3}	0.75	0.565265	0.282700	10.934	2604	0.000018
	10^{-3}	0.3	0.565272	0.282703	28.242	6700	0.000018
.2, .4	10^{-3}	0.5	0.565269	0.282701	16.978	3981	0.000018
	10^{-3}	0.75	0.565265	0.282699	10.934	2615	0.000018
	10^{-2}	0.3	0.628248	0.315272	0.549	134	0.000276

Tabela 2.3: Resultados obtidos para o problema (P-2).

Neste caso a solução ótima é dada por $x^* = (0.5, 0.25)^T$.

Em [3], encontra-se também, uma generalização do modelo de rede neural apresentado aqui, para resolver também problemas de otimização restritos. A seguir esta extensão será descrita.

2.3 Problemas de Otimização Restritos

Problemas de otimização restritos, em geral, aparecem na forma:

$$\begin{aligned}
 & \min_x f(x) \\
 & \text{sujeito a : } h_1(x) = 0 \quad g_1(x) \leq 0 \\
 & \quad h_2(x) = 0 \quad g_2(x) \leq 0 \\
 & \quad \vdots \quad \vdots \\
 & \quad h_m(x) = 0 \quad g_p(x) \leq 0 \\
 & \quad x \in \Omega \subseteq \mathbb{R}^n
 \end{aligned} \tag{2.6}$$

onde $m \leq n$ e as funções $f, h_i, i=1,2, \dots, m$ e $g_j, j=1,2,\dots, p$ são contínuas e de classe C^2 .

Problemas desta forma são considerados *problemas de otimização restritos*.

Para simplicidade de notação pode-se denotar $h = (h_1, h_2, \dots, h_m)$ e $g = (g_1, g_2, \dots, g_p)$ e assim rescrever o problema (2.6) da seguinte forma:

$$\begin{aligned}
 & \min_x f(x) \\
 & \text{sujeito a: } h(x)=0 \\
 & \quad g(x) \leq 0 \\
 & \quad x \in \Omega
 \end{aligned} \tag{2.7}$$

Como já foi dito, uma generalização da abordagem anterior (seção 2.2) foi também realizada em [3], com o objetivo de resolver o problema (2.7). Para essa generalização alguns resultados da teoria de otimização foram necessários e, portanto, são descritos a seguir.

2.3.1 Otimização de Problemas com Restrições de Igualdade

Considerando somente as restrições de igualdade do problema (2.7), obtem-se o seguinte problema:

$$\begin{aligned} & \min_x f(x) \\ & \text{sujeito a: } h(x)=0 \end{aligned} \quad (P)$$

que é chamado de problema Primal (P).

Definição [10]: Um ponto x^* satisfazendo as restrições $h(x^*)=0$ é dito ser um ponto regular das restrições se os vetores gradientes

$$\nabla h_1(x^*), \nabla h_2(x^*), \dots, \nabla h_m(x^*)$$

forem linearmente independentes.

Teorema [10]: Seja x^* um ponto regular das restrições $h(x)=0$ e um ponto extremo local (mínimo ou máximo) de $f(x)$ sujeito a essas restrições. Então existe um $\lambda \in \mathbb{R}^m$ tal que:

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

Teorema [10]: Suponha que exista um ponto x^* satisfazendo $h(x^*)=0$ e um $\lambda \in \mathbb{R}^m$ tal que:

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

Suponha também que a matriz $L(x^*) = F(x^*) + \lambda^T H(x^*)$ é definida positiva em $M = \{y: \nabla h(x^*) \cdot y = 0\}$, isto é, para todo $y \in M, y \neq 0$ temos $y^T \cdot L(x^*) \cdot y > 0$. Então x^* é um mínimo local estrito de f sujeito a $h(x) = 0$.

O problema (P) pode ser transformado em um problema de otimização irrestrito através das seguintes considerações:

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

$$L(x, \lambda) = f(x) + \lambda^T h(x)$$

onde λ é um vetor do $\mathbb{R}^m$ e é chamado multiplicador de Lagrange. Seja também a função dual:

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

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

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

$$\max_{\lambda} \phi(\lambda) \quad (D)$$

Pode ser provado 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 $\phi(\lambda)$ com respeito à λ [10].

Desta forma, resolver o problema (P) é equivalente a resolver o problema de otimização irrestrito

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

para cada valor de λ e, em seguida, o valor máximo de $\phi(\lambda)$ deve ser determinado.

Se a função dual for diferenciável em um ponto $\tilde{\lambda}$, então,

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

onde $\tilde{x}$ minimiza a função $L(x, \tilde{\lambda})$ [10].

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

De acordo com a definição de subgradiente, pode ser provado que $h(\tilde{x})$ é um subgradiente da função $\phi(\lambda)$ no ponto $\tilde{\lambda}$ [10].

A partir destes resultados, para resolver o problema (P), os seguintes subproblemas podem ser resolvidos:

(1) Para λ fixado, resolver:

$$\min_x L(x, \lambda) \quad (2.8)$$

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

(2) Para $\tilde{x}$ obtido em (2.8), atualizar λ de acordo com a direção do subgradiente da função ϕ , com um passo η ,

$$\lambda^{k+1} = \lambda^k + \eta \cdot h(x^k), \quad (2.9)$$

uma vez que se deseja resolver:

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

que também é um problema de otimização irrestrito.

De forma análoga a abordagem anterior (minimização irrestrita) a regra de atualização dos pesos para a resolução do problema (D) (maximização irrestrita), de acordo com [3] apresenta a seguinte forma:

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

onde

$$\delta_j^{(2)} = h_j(\tilde{x}) \cdot g'(\text{net}_j^{(2)}) ; j=1,2,\dots,m;$$

$$\delta_j^{(1)} = \sum_k (\delta_j^{(2)} w_{jk}^{(2)}) \cdot g'(\text{net}_j^{(1)}) ; j=1,2, \dots, m$$

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

Critério de parada: Para cada λ obtido em (2.9), resolvemos o problema dado por (2.8). O processo é repetido até que

$$\|\nabla\phi(\lambda)\| \leq \varepsilon$$

onde ε é um valor arbitrário pequeno.

A topologia deste modelo de rede neural é mostrada na Figura 2.2. Maiores detalhes sobre esta generalização podem ser encontrados em [1], [2], [3]. Os exemplos 3 e 4 ilustram o comportamento desta rede neural.

Figura 2.2: Topologia da rede neural para resolver problemas restritos.

Exemplo 3: Consideremos o problema (P-3)

$$\min (x_1 - 0.5)^2 + (x_2 - 0.4)^2 \quad (\text{P-3})$$

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

As soluções aproximadas obtidas (x, λ) para o problema (P-3) são mostradas na Tabela 2.4, considerando diferentes soluções iniciais (x^0, λ^0) , ε e η .

$(x^0, \lambda^0)^T$	ε	η	$(x, \lambda)^T$			tempo em s	iter. (P)	iter. (D)	função objetivo
.1, .2, .3	10^{-2}	0.5	0.445088	0.344944	0.109411	2.363	608	76	0.006046
	10^{-3}	0.75	0.450499	0.350500	0.099045	14.945	2001	1333	0.004901
.3, .1, .2	10^{-3}	0.5	0.450499	0.350500	0.099042	19.835	2678	1678	0.004901
	10^{-2}	0.75	0.445028	0.345002	0.109534	2.582	546	158	0.006047
-.1, .0, .6	10^{-3}	0.5	0.450497	0.350499	0.099055	101.04	12007	9112	0.004901
	10^{-2}	0.75	0.454992	0.354996	0.090454	12.912	1687	1227	0.004051

Tabela 2.4: Resultados obtidos para o problema (P-3).

A solução ótima neste caso é $x^* = (0.45, 0.35)^T$ e $\lambda^* = 0.1$.

Exemplo 4: Consideremos o problema (P-4):

$$\min -x_1 x_2 \quad (\text{P-4})$$

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

Neste caso, as soluções aproximadas obtidas (x, λ) são mostradas na Tabela 2.5, considerando diferentes soluções iniciais (x^0, λ^0) , ε e η .

$(x^0, \lambda^0)^T$	ε	η	$(x, \lambda)^T$			tempo em s	iter. (P)	iter. (D)	função objetivo
.4, .35, .6	10^{-3}	0.5	0.400520	0.200730	1.0000	19.231	3893	18	-0.080397
	10^{-2}	0.75	0.405575	0.207409	1.0000	8.077	1638	14	-0.084120
.2, .1, .4	10^{-2}	0.5	0.405521	0.207338	1.0000	8.626	1734	18	-0.084080
	10^{-3}	0.75	0.400498	0.200729	1.0000	7.088	1441	14	-0.080392
.3, .1, .5	10^{-3}	0.5	0.400509	0.200732	1.0000	14.286	2880	18	-0.080395
	10^{-2}	0.75	0.400498	0.200728	1.0000	7.967	1625	14	-0.080391

Tabela 2.5: Resultados obtidos para o problema (P-4).

Note que os resultados estão próximos da solução ótima $x^* = (0.4, 0.2)^T$ e $\lambda^* = 1.0$.

2.4 Algoritmo *Back-Propagation* * (BP*)

Esta seção apresenta o algoritmo relativo ao processamento da rede neural apresentada nas seções anteriores. Para simplificação de notação denotaremos, aqui, este algoritmo por algoritmo *Back-Propagation* * (BP*).

Sejam:

n : dimensão do problema primal.

m : dimensão do problema dual.

i : camada de entrada.

h : camada intermediária.

o : camada de saída.

w_{ih} : matriz de pesos entre a camada de entrada e a camada intermediária.

w_{ho} : matriz de pesos entre a camada intermediária e a camada de saída.

$x_k^T = (x_k^1, x_k^2)^T$: vetor de entrada $(n+m)$ dimensional da camada k .

$y_k^T = (y_k^1, y_k^2)^T$: vetor de saída $(n+m)$ dimensional da camada k .

$g(x) = \frac{1}{1 + e^{-x}}$: função de ativação.

Inicialização: $y_i^2 = 0$

$$y_h^2 = 0$$

$$\delta_o^2 = 0$$

$$\delta_h^2 = 0$$

Passo Primal Forward: $y_i^1 = g(x_i^1)$

$$(x_h^1)^T = (w_{ih}^1)^T \cdot y_i$$

$$y_h^1 = g(x_h^1)$$

$$(x_o^1)^T = (w_{ho}^1)^T \cdot y_h$$

$$y_o^1 = g(x_o^1)$$

Passo Primal Backward: $\delta_o^1 = -\nabla f(y_o^1) \cdot \nabla g(x_o^1)$

$$w_{ho}^1 = w_{ho}^1 + \eta \cdot \delta_o^1 \cdot y_h^1$$

$$\delta_h^1 = g'(x_h^1) \cdot (w_{ho}^1)^T \cdot \delta_o^1$$

$$w_{ih}^1 = w_{ih}^1 + \eta \cdot \delta_h^1 \cdot y_i^1$$

$$x_i^1 = y_o^1$$

Passo Dual: $y_i^2 = g(x_i^2)$

$$(x_h^2)^T = (w_{ih}^2)^T \cdot y_i$$

$$y_h^2 = g(x_h^2)$$

$$(x_o^2)^T = (w_{ho}^2)^T \cdot y_h$$

$$y_o^2 = g(x_o^2)$$

$$\delta_o^2 = \nabla f(y_o^2) \cdot h(y_o^2)$$

$$w_{ho}^2 = w_{ho}^2 + \eta \cdot \delta_o^2 \cdot y_h^2$$

$$\delta_h^2 = g'(x_h^2) \cdot (w_{ho}^2)^T \cdot \delta_o^2$$

$$w_{ih}^2 = w_{ih}^2 + \eta \cdot \delta_h^2 \cdot y_i^2$$

$$x_i^2 = y_o^2$$

Cálculo do Erro Local: $\varepsilon_i = \|\nabla f(y_o^1) + (x_i^2)^T \cdot \nabla h(y_o^1)\|$

Cálculo do Erro Global: $\varepsilon_g = \|h(y_o^1)\|$

Algoritmo *Back-Propagation* ***Início**dado (x^0, λ^0) **Enquanto** solução não encontrada **faça****Início**executar o Passo Primal *Forward*.**Se** $\varepsilon_l \leq$ limite erro local **então****Se** $\varepsilon_g \leq$ limite erro global **então**

solução foi encontrada

Senão

executar o Passo Dual.

Senãoexecutar o Passo Primal *Backward*.**Fim do enquanto****Fim**

2.5 Conclusão

O modelo de rede neural proposto em [3] para resolver problemas de otimização não lineares foi apresentado e testado neste capítulo, pois sua compreensão é essencial para as modificações que serão realizadas e analisadas no próximo capítulo.

CAPÍTULO 3

Modificações do Algoritmo *Back-Propagation**

3.1 Introdução

Neste capítulo são propostas algumas modificações do algoritmo BP*. A finalidade é investigar o comportamento do algoritmo de aprendizado da rede estruturada anteriormente diante de tais modificações.

3.2 Técnicas de Otimização

É provado que o uso de técnicas de otimização em algoritmos de redes neurais pode melhorar o desempenho das mesmas, tanto em tempo de aprendizagem quanto em termos de qualidade de solução [7], [15].

O algoritmo de aprendizado BP*, na forma em que foi proposto em [3], permite a solução de problemas de otimização convexos implementando um algoritmo de

minimização baseado apenas no método gradiente simples, com taxa de aprendizado constante.

Sendo assim, algumas técnicas de otimização foram incorporadas a este esquema de aprendizado, tais como: *termo momentum*, gradiente com busca linear e o gradiente conjugado, com o propósito de investigar o comportamento da rede modificada em termos de tempo de aprendizagem e qualidade de solução. Estas modificações do algoritmo BP* são apresentadas e discutidas como segue. Alguns resultados computacionais são também mostrados.

3.2.1 O termo *momentum*

A mais popular das variações proposta ao algoritmo BP é a inclusão do *termo momentum* na regra de atualização dos pesos da rede. O uso deste tipo de parâmetro permite incrementar a taxa de aprendizado do algoritmo, sem conduzir o algoritmo para oscilações. A inclusão do *termo momentum*, de acordo com Rumelhart [18], consiste em atualizar os pesos de acordo com:

$$\Delta w_{ij}(k) = -\eta \frac{\partial E(k)}{\partial w_{ij}} + \mu \Delta w_{ij}(k-1) \quad (3.1)$$

onde: $\mu \in [0,1]$

η : taxa de aprendizado e,

k : representa a iteração.

Segundo Rumelhart [18], o parâmetro μ considera o efeito do passo anterior na corrente direção do movimento no espaço dos pesos, evitando assim grandes oscilações na superfície da função erro, ou seja, se o último peso foi mudado em uma determinada direção, o *termo momentum* leva em consideração esta direção para fazer a próxima mudança de peso.

Esta consideração é muito útil em superfícies contendo grandes declives, caracterizada por curvaturas pontiagudas. Estes tipos de curvaturas tendem a causar grandes oscilações na superfície da função. Para evitar este tipo de problema é necessário que sejam considerados passos muito pequenos durante todo o aprendizado, causando um progresso muito lento ao longo da superfície. O *termo momentum* evita as grandes oscilações e desta forma permite que o tamanho do passo seja não necessariamente muito pequeno.

Deve ser notado, entretanto, que embora esta técnica apresente bons resultados em muitas tarefas de aprendizado, ela não pode ser considerada uma técnica para obtenção de estabilidade ou aceleração de convergência [17]. Segundo Ridmiller [17], às vezes, comparáveis ou melhores resultados podem ser obtidos sem o uso do *termo momentum* neste tipo de aprendizado.

Esta modificação foi implementada ao algoritmo original e alguns resultados são descritos a seguir, adotando-se $\mu = 0.9$ e diferentes soluções iniciais para a rede neural apresentada. No exemplo 5, as Tabelas 3.1, 3.2, 3.3 e 3.4 correspondem à resolução dos problemas (P-1), (P-2), (P-3), e (P-4) do Capítulo 2, respectivamente.

Exemplo 5:

5.1) Problema (P-1):

$(x^0)^T$	ε	η	$(x)^T$	tempo em s	iterações	função objetivo
.2, .15, .45, .6	10^{-2}	0.5	0.095379 0.199769 0.300073 0.399374	0.495	102	0.00002
	10^{-2}	0.75	0.096088 0.200227 0.297815 0.401706	0.440	94	0.000024
	10^{-3}	0.75	0.100435 0.200060 0.300036 0.399856	0.695	41	0.0
.4, .32, .22, .11	10^{-3}	0.5	0.099574 0.200277 0.299965 0.400032	0.549	124	0.0
	10^{-3}	0.75	0.100474 0.200055 0.300006 0.399859	0.695	140	0.0
	10^{-2}	0.3	0.095196 0.199876 0.300626 0.400408	0.440	110	0.000024
	10^{-3}	0.7	0.099913 0.200366 0.300155 0.399792	0.495	107	0.0

Tabela 3.1: Resultados obtidos para o problema (P-1) com o algoritmo BP* com *termo momentum* $\mu = 0.9$.

5.2) Problema (P-2):

$(x^0)^T$	ε	η	$(x)^T$	tempo em s	iterrações	função objetivo
.1, .2	10^{-2}	0.5	0.609831 0.306054	0.220	43	0.000151
	10^{-3}	0.75	0.565224 0.282655	6.813	1617	0.000018
	10^{-3}	0.3	0.565311 0.282703	10.220	2410	0.000018
.2, .4	10^{-3}	0.5	0.565257 0.282673	8.242	1947	0.000018
	10^{-3}	0.75	0.565220 0.282652	6.703	2615	0.000018
	10^{-2}	0.3	0.613836 0.307892	0.275	71	0.000172

Tabela 3.2: Resultados obtidos para o problema (P-2) com o algoritmo BP* com termo momentum $\mu = 0.9$.

5.3) Problema (P-3):

$(x^0, \lambda^0)^T$	ε	η	$(x, \lambda)^T$	tempo em s	iter. (P)	iter. (D)	função objetivo
.1, .2, .3	10^{-2}	0.5	0.454951 0.354994 0.090650	4.176	544	323	0.004055
	10^{-3}	0.75	0.450493 0.350498 0.099082	7.308	1135	396	0.004901
.3, .1, .2	10^{-3}	0.5	0.450493 0.350499 0.099079	8.846	1418	469	0.004901
	10^{-2}	0.75	0.454921 0.344966 0.090722	3.681	494	269	0.004060
-.1, .0, .6	10^{-3}	0.5	0.450491 0.350496 0.099089	32.143	3644	2349	0.004902
	10^{-2}	0.75	0.454909 0.354952 0.090797	10.110	1141	868	0.004063

Tabela 3.3: Resultados obtidos para o problema (P-3) com o algoritmo BP* com termo momentum $\mu = 0.9$.

5.4) Problema (P-4):

$(x^0, \lambda^0)^T$	ε	η	$(x, \lambda)^T$	tempo em s	iter. (P)	iter. (D)	função objetivo
.4, .35, .6	10^{-3}	0.5	0.400524 0.200733 1.000000	3.022	685	6	-0.080397
	10^{-2}	0.75	0.402886 0.206249 0.999998	1.648	372	6	-0.084120
.2, .1, .4	10^{-2}	0.5	0.403763 0.206808 0.999768	1.374	303	16	-0.084080
	10^{-3}	0.75	0.400622 0.200240 0.999532	1.319	295	16	-0.080392
.3, .1, .5	10^{-3}	0.5	0.400471 0.200720 1.000000	2.033	452	11	-0.080395
	10^{-3}	0.75	0.39940 0.199560 0.999986	1.264	288	11	-0.080391

Tabela 3.4: Resultados obtidos para o problema (P-4) com o algoritmo BP* com termo momentum $\mu = 0.9$.

Pode-se dizer que a implementação do *termo momentum* ao algoritmo BP* proporcionou um melhor desempenho do mesmo na maioria dos testes feitos.

Comparando-se o algoritmo BP* com *termo momentum* na resolução dos problemas (P-1), (P-2), (P-3) e (P-4), cujos resultados são mostrados no Exemplo 5, com algoritmo BP* sem o *termo momentum* na resolução dos mesmos problemas (Tabelas 2.2, 2.3, 2.4, 2.5) pode-se notar que os resultados obtidos pelo algoritmo BP* com *termo momentum* apresentaram uma diminuição no tempo de aprendizado da rede, um menor número de iterações requeridas para este aprendizado, além de solução mais precisas quando comparados com os resultados obtidos com o BP* sem o *termo momentum*.

É importante notar, ainda, que o uso do parâmetro *momentum* na atualização dos pesos não apresentou problemas quanto à convergência da rede em nenhum dos testes realizados.

3.2.2 Gradiente Simples com Busca Linear

Embora algoritmos baseados no método do gradiente sejam considerados basicamente simples, a dificuldade neste tipo de algoritmo está na escolha de uma taxa de aprendizado apropriada. Uma boa escolha depende do aspecto da função erro, o qual muda com a própria tarefa de aprendizado. No caso de uma função erro sem grandes oscilações, por exemplo, uma pequena taxa de aprendizado resultará em um longo tempo de convergência, enquanto que uma grande taxa de convergência poderá causar oscilações, podendo causar o aumento da função objetivo [17].

Outro problema com o método do gradiente simples é a influência da derivada da função no tamanho do passo. Pequenas derivadas resultam em passos pequenos, enquanto que grandes derivadas, onde cautelosos passos deverão ser tomados, conduzem a um

aumento do tamanho do passo, podendo fazer com que o algoritmo caminhe para regiões diferentes do espaço de pesos [17].

Justificativas simples para o uso de um tamanho de passo constante em algoritmos de aprendizado supervisionado é que a busca linear é dispendiosa em termos de tempo computacional e não necessária, desde que este tipo de algoritmo pode obter bons resultados sem a mesma, e além disso, uma flutuação em toda extensão da função objetivo pode capacitar a rede para escapar de um mínimo local indesejado [7].

Entretanto, decidiu-se incorporar o método de busca linear ao algoritmo original (BP*) para analisar os seus efeitos.

Enquanto a técnica do gradiente simples usada no algoritmo BP* atualiza os pesos através de uma taxa de aprendizado constante η na direção de descida, a busca linear incorporada ao BP* tenta encontrar um parâmetro $\eta(k)$ ótimo em cada iteração. Desta forma, os pesos da rede são atualizados por:

$$\Delta w_{ij}(k) = -\eta(k) \cdot \frac{\partial E}{\partial w_{ij}}(k) \quad (3.2)$$

A determinação deste parâmetro ótimo é um problema de busca linear, ou busca unidimensional, no qual uma pequena taxa inicial é usada e incrementada iterativamente até que a função erro não decresça mais, naquela determinada direção.

Obviamente, este é um procedimento dispendioso em termos de tempo computacional por iteração (*forward*) por causa das numerosas avaliações da função objetivo. O objetivo é, portanto, determinar se a busca unidimensional aplicada ao BP* converge ou não, mais rápido do que o BP* sem a busca.

Entre os métodos de busca unidimensional existentes [10] os seguintes métodos foram implementados e testados aqui neste trabalho: o método da falsa posição e a regra de Armijo.

3.2.2.1 Regra de Armijo [10]:

A Regra de Armijo é uma das mais práticas e populares busca unidimensional. A idéia essencial da regra é garantir que o tamanho do passo α selecionado não seja tão grande nem tão pequeno. Seja a função definida por:

$$\phi(\alpha) = f(x_k + \alpha d_k)$$

A Regra de Armijo é implementada considerando a função $\phi(0) + \varepsilon\phi'(0)\alpha$ para $0 < \varepsilon < 1$. Um valor de α é considerado não ser tão grande se :

$$\phi(\alpha) \leq \phi(0) + \varepsilon\phi'(0)\alpha \quad (3.3)$$

e α é considerado não ser tão pequeno se para um valor selecionado $\eta > 1$

$$\phi(\eta\alpha) \leq \phi(0) + \varepsilon\phi'(0)\eta\alpha \quad (3.4)$$

Desta forma, um α inicial arbitrário é tomado. Se α satisfaz (3.3) ele é incrementado por η (geralmente são usados $\eta = 2$ ou $\eta = 10$ e $\varepsilon = 0.2$) até que (3.3) não mais seja satisfeita e então o penúltimo α é selecionado. Se, no entanto, α original não satisfizer (3.3), ele será dividido repetidamente por η até que o resultante α satisfaça (3.3).

3.2.2.2 Método da Falsa Posição [10]:

O Método da Falsa Posição tem como idéia aproximar a função por uma função quadrática usando a informação de dois pontos. Desta forma, usando:

$$f(x_k), f'(x_k), f'(x_{k-1})$$

é possível ajustar f pela função quadrática:

$$q(x) = f(x_k) + f'(x_k)(x - x_k) + \frac{f'(x_{k-1}) - f'(x_k)}{x_{k-1} - x_k} \cdot \frac{(x - x_k)^2}{2}$$

Uma estimativa para x_{k+1} pode ser determinada encontrando o ponto onde a derivada de q se anula. Desta forma:

$$x_{k+1} = x_k - f'(x_k) \left[\frac{x_{k-1} - x_k}{f'(x_{k-1}) - f'(x_k)} \right] \quad (3.5)$$

Este método, portanto, não depende diretamente dos valores da função f , pode-se dizer que ele é um método para resolver $f'(x) \equiv g(x) = 0$. Por este ponto de vista temos a forma:

$$x_{k+1} = x_k - g(x_k) \left[\frac{x_{k-1} - x_k}{g(x_k) - g(x_{k-1})} \right]$$

Alguns resultados são descritos a seguir para mostrar o desempenho do modelo de rede neural com parâmetro de aprendizado variável. Foram adotadas diferentes soluções iniciais e as duas técnicas diferentes de buscas lineares. Os resultados das Tabelas 3.5, 3.6, 3.7 e 3.8 referem-se aos problemas (P-1), (P-2), (P-3), e (P-4) do Capítulo 2, respectivamente.

Com base na análise dos resultados obtidos, em testes realizados com problemas restritos, a busca linear foi aplicada somente na resolução do subproblema primal, de forma que o subproblema dual permaneceu inalterado. Para os problemas irrestritos (P-1) e (P-2), o parâmetro η das Tabelas 3.5 e 3.6 corresponde ao passo inicial no método de Armijo. No caso dos problemas restritos (P-3) e (P-4) o parâmetro η das Tabelas 3.7 e 3.8 corresponde ao tamanho do passo dual e, ao tamanho do passo inicial, no método de Armijo.

Exemplo 6:**6.1) Problema (P-1):**

Método	$(x^0)^T$	ε	η	$(x)^T$	tempo em s	iter. (D)	função objetivo
Armijo	.2, .15, .45, .6	10^{-2}	0.5	0.104463 0.199706 0.299731 0.399923	0.275	60	0.000020
		10^{-2}	0.75	0.104804 0.199670 0.299707 0.399917	0.275	62	0.000023
		10^{-3}	0.75	0.100449 0.199971 0.299975 0.399993	0.330	75	0.0
	.4, .32, .22, .11	10^{-3}	0.5	0.100480 0.199969 0.299974 0.399992	0.385	79	0.0
		10^{-3}	0.75	0.100450 0.199971 0.299975 0.399993	0.385	75	0.0
		10^{-2}	0.3	0.104893 0.199747 0.299723 0.399921	0.165	36	0.000024
		10^{-3}	0.7	0.100463 0.199970 0.299975 0.399993	0.330	76	0.0
Fal.Pos.	.2, .15, .45, .6	10^{-2}	—	0.104497 0.199731 0.299740 0.399926	0.165	30	0.000020
		10^{-3}	—	0.100498 0.199970 0.299974 0.399992	0.275	47	0.0
	.4, .32, .22, .11	10^{-3}	—	0.104538 0.199732 0.29937 0.399925	0.220	30	0.000021
		10^{-3}	—	0.100443 0.199974 0.299977 0.319993	0.275	88	0.0

Tabela 3.5: Resultados obtidos para o problema (P-1) com o algoritmo BP* com busca linear.

6.2) Problema (P-2):

Método	$(x^0)^T$	ε	η	$(x)^T$	tempo em s	iterrações	função objetivo
Armijo	.1, .2	10^{-2}	0.5	0.627910 0.315096	0.330	79	0.000273
		10^{-3}	0.75	0.564264 0.282184	1.319	307	0.000018
		10^{-3}	0.3	0.565096 0.282594	1.538	359	0.000018
	.2, .4	10^{-3}	0.5	0.564646 0.282393	1.319	306	0.000017
		10^{-3}	0.75	0.565314 0.282717	1.209	283	0.000018
		10^{-2}	0.3	0.628248 0.315272	0.604	134	0.000276
Fal. Pos.	.1, .2	10^{-2}	—	0.628026 0.315034	0.385	88	0.000273
		10^{-3}	—	0.564933 0.282536	0.879	205	0.000018
	.2, .4	10^{-2}	—	0.62817 0.315119	0.385	88	0.000274
		10^{-3}	—	0.564614 0.282386	0.879	212	0.000274

Tabela 3.6: Resultados obtidos para o problema (P-2) com o algoritmo BP* com busca linear.

6.3) Problema (P-3):

Método	$(x^0, \lambda^0)^T$	ε	η	$(x, \lambda)^T$			tempo em s	iter. (P)	iter. (D)	função objetivo
Armijo	.1, .2, .3	10^{-2}	0.5	0.445135	0.344904	0.109260	1.813	452	62	0.006046
		10^{-3}	0.75	0.450499	0.350500	0.099045	17.912	2001	1333	0.004901
	.3, .1, .2	10^{-3}	0.5	0.450498	0.350500	0.099045	19.286	2493	1683	0.004901
		10^{-2}	0.75	0.445028	0.345002	0.109534	3.242	546	158	0.006047
	-.1, .0, .6	10^{-3}	0.5	0.450498	0.350500	0.099054	109.34	12255	10142	0.004901
		10^{-2}	0.75	0.454991	0.354995	0.090460	14.890	1561	1251	0.004051
Fal. Pos.	.1, .2, .3	10^{-2}	-	0.445391	0.344730	0.108311	0.640	123	14	0.006037
		10^{-3}	-	0.450499	0.350500	0.099036	8.249	932	733	0.004901
	.3, .1, .2	10^{-3}	-	0.450499	0.350500	0.099034	7.802	898	686	0.004901
		10^{-2}	-	0.415018	0.344997	0.109596	1.209	196	196	0.006048
	-.1, .0, .6	10^{-3}	-	0.450498	0.350499	0.099047	74.286	6696	6295	0.004901
		10^{-2}	-	0.454985	0.354983	0.090403	9.231	978	868	0.004053

Tabela 3.7: Resultados obtidos para o problema (P-3) com o algoritmo BP* com busca linear.

6.4) Problema (P-4):

Método	$(x^0, \lambda^0)^T$	ε	η	$(x, \lambda)^T$			tempo em s	iter. (P)	iter. (D)	função objetivo
Armijo	.4, .35, .6	10^{-3}	0.5	0.400520	0.200730	1.0000	16.813	3893	18	-0.080397
		10^{-2}	0.75	0.405575	0.207409	1.0000	7.088	1638	14	-0.084120
	.2, .1, .4	10^{-2}	0.5	0.405521	0.207338	1.0000	7.527	1734	18	-0.084080
		10^{-3}	0.75	0.400498	0.200729	1.0000	6.209	1441	14	-0.080392
	.3, .1, .5	10^{-3}	0.5	0.400509	0.200732	1.0000	12.418	2880	18	-0.080395
		10^{-3}	0.75	0.400498	0.200728	1.0000	7.033	1625	14	-0.080391

Tabela 3.8: Resultados obtidos para o problema (P-4) com o algoritmo BP* com busca linear.

Os testes feitos com o problema (P-4) usando o método da falsa posição não convergiram.

De uma maneira geral, o uso da busca linear no algoritmo BP* mostrou resultados considerados favoráveis ao seu desempenho. Os cálculos envolvidos com este tipo de busca foram compensados, neste modelo, por uma convergência mais rápida na maioria dos testes feitos, comparados com a convergência do BP* sem a busca. Porém, em contradição, alguns sérios problemas com convergência foram verificados, como é o caso da aplicação do método da falsa posição para o problema (P-4).

O desempenho de cada busca linear testada aqui, em particular, varia de um problema para outro.

No exemplo 6.1, por exemplo, ambas as buscas (Armijo e falsa posição), proporcionaram melhoras significativas ao algoritmo BP* original, como pode ser observado comparando-se os resultados contidos na Tabela 3.5 com os resultados da Tabela 2.2. Já nos exemplos 6.2 e 6.3, o método da falsa posição obteve melhores resultados em relação a tempo de aprendizagem e número de iterações, comparados com o método de Armijo, no desempenho da mesma tarefa de aprendizado. No exemplo 6.4, no entanto, enquanto o método da falsa posição não convergiu, o método de Armijo apresentou resultados praticamente idênticos ao BP* sem a busca, porém com menor tempo de aprendizado.

3.2.3 Gradiente Conjugado

De acordo com o método do gradiente conjugado, os pesos são atualizados por:

$$\Delta w_{ij}(k+1) = \eta \cdot d(k+1) \quad (3.6)$$

onde, segundo Luenberger [10], a direção $d(k+1)$ é dado por:

$$d(k+1) = -\nabla E(k+1) + \beta \cdot d(k) \quad (3.7)$$

o parâmetro β :

$$\beta = \frac{\nabla E(k+1)^T \cdot H \cdot d(k+1)}{d(k)^T H d(k)} \quad (3.8)$$

e o parâmetro taxa de aprendizado é dado por:

$$\eta = \frac{-\nabla E(k)^T \cdot d(k)}{d(k)^T H \cdot d(k)} \quad (3.9)$$

Observa-se que em (3.8) e (3.9) aparece a necessidade de se calcular a matriz Hessiana da função erro E . No caso de problemas não quadráticos, nos testes realizados neste trabalho, a Hessiana foi calculada a cada iteração e seus valores foram normalizados, a fim de se obter melhores resultados. Testes foram feitos, também, no caso de problemas quadráticos, considerando a matriz Hessiana como uma matriz constante, com valores contidos no intervalo $[0,1]$. Neste caso, em particular, melhores resultados foram obtidos com a matriz Hessiana com elementos constantes com valores iguais a 0.1.

O parâmetro β , segundo Luenberger [10] pode, ainda, ser calculado pelo método de Fletcher-Reeves como :

$$\beta = \frac{\nabla E(k+1)^T \cdot \nabla E(k+1)}{\nabla E(k)^T \cdot \nabla E(k)} \quad (3.10)$$

ou pelo método Polak-Ribiere, sendo dado por:

$$\beta = \frac{(\nabla E(k+1) - \nabla E(k))^T \cdot \nabla E(k+1)}{\nabla E(k)^T \cdot \nabla E(k)} \quad (3.11)$$

Em (3.10) e (3.11) o parâmetro η é dado por uma busca unidimensional na direção $d(k+1)$.

Várias combinações dos parâmetro η e β foram implementadas a fim de se verificar o comportamento do algoritmo BP* diante de tais modificações.

Alguns resultados são apresentados no exemplo 7. Neste exemplo as Tabelas 3.9, 3.10, 3.11, 3.12 contem resultados referentes aos problemas (P-1), (P-2), (P-3) e (P-4) respectivamente, adotando-se diferentes soluções iniciais, diferentes valores para ε e as duas buscas lineares já apresentadas, Armijo e falsa posição, no cálculo do parâmetro η em (3.10) e (3.11).

Exemplo 7:**7.1) Problema (P-1):**

Cálculo de η		$(x^0)^T$			(x^T)	tempo em s	iterrações	função obj.
(3.9)	(3.8)	.2, .15, .45, .6	10^{-2}	—	0.104970 0.199709 0.299687 0.399876	2.033	454	0.0
			10^{-3}	—	0.100498 0.199968 0.299973 0.399994	3.846	863	0.0
		.4, .32, .22, .11	10^{-2}	—	0.104974 0.199710 0.299687 0.399876	2.033	454	0.0
			10^{-3}	—	0.100497 0.199967 0.299972 0.399989	3.846	846	0.0
(3.9)	(3.8)	.2, .15, .45, .6	10^{-2}	—	0.104457 0.200018 0.299945 0.400840	0.055	11	0.0
*	*		10^{-3}	—	0.100423 0.199999 0.299993 0.400077	0.055	21	0.0
		.4, .32, .22, .11	10^{-2}	—	0.104830 0.20007 0.299931 0.400915	0.055	11	0.0
			10^{-3}	—	0.100486 0.199997 0.299991 0.400089	0.055	21	0.0
Armijo	(3.10)	.2, .15, .45, .6	10^{-2}	0.5	0.104416 0.199658 0.299764 0.399923	0.275	63	0.0002
			10^{-2}	0.75	0.104953 0.199707 0.299702 0.399913	0.385	95	0.0002
			10^{-3}	0.75	0.100490 0.199969 0.299973 0.399992	0.824	180	0.0
		.4, .32, .22, .11	10^{-3}	0.5	0.100468 0.199970 0.299974 0.399993	0.330	75	0.0002
			10^{-3}	0.75	0.100491 0.199969 0.299973 0.399992	0.769	180	0.0
			10^{-2}	0.3	0.104566 0.199689 0.299733 0.399923	0.275	52	0.0002
			10^{-3}	0.7	0.100492 0.199969 0.299973 0.399992	0.934	193	0.0
Fal. Pos.	(3.10)	.2, .15, .45, .6	10^{-2}	—	0.104660 0.199692 0.299736 0.399924	0.165	30	0.0002
			10^{-3}	—	0.100453 0.199973 0.299733 0.399993	0.220	48	0.0
		.4, .32, .22, .11	10^{-2}	—	0.104703 0.199692 0.299931 0.399924	0.165	30	0.0002
			10^{-3}	—	0.100459 0.199973 0.299976 0.399993	0.220	48	0.0
Armijo	(3.11)	.2, .15, .45, .6	10^{-2}	0.5	0.104854 0.199699 0.299709 0.399916	0.440	96	0.0002
			10^{-2}	0.75	0.104974 0.199689 0.299707 0.399917	0.330	64	0.0002
			10^{-3}	0.75	0.100494 0.199970 0.299974 0.399992	0.549	119	0.0
		.4, .32, .22, .11	10^{-3}	0.5	0.100497 0.199969 0.299973 0.399992	0.769	180	0.0
			10^{-3}	0.75	0.100496 0.199970 0.299974 0.399992	0.549	119	0.0
			10^{-2}	0.3	0.104927 0.199489 0.299920 0.400058	0.275	58	0.0002
			10^{-3}	0.7	0.100491 0.199970 0.299974 0.399992	0.604	128	0.0
Fal. Pos.	(3.11)	.2, .15, .45, .6	10^{-2}	—	0.104962 0.199624 0.299788 0.399919	0.110	30	0.0002
			10^{-3}	—	0.199971 0.199975 0.299975 0.39993	0.220	48	0.0
		.4, .32, .22, .11	10^{-2}	—	0.104391 0.199686 0.299803 0.399915	0.165	31	0.0002
			10^{-3}	—	0.100487 0.199971 0.299975 0.399993	0.220	48	0.0

Tabela 3.9: Resultados obtidos para o problema (P-1) com o algoritmo BP* implementando o método do gradiente conjugado.

* matriz Hessiana com elementos constante com valores iguais a 0.1.

7.2) Problema (P-2):

Cálculo de η	β	$(x^0)^T$	ε	η	$(x)^T$		tempo em s	iterrações	função objetivo
(3.9)	(3.8)	.1, .2	10^{-2}	—	0.575425	0.288901	3.312	745	0.000038
			10^{-3}	—	0.546342	0.273293	4.341	1017	0.000005
		.2, .4	10^{-2}	—	0.575726	0.289051	3.297	775	0.000039
			10^{-3}	—	0.547725	0.273985	4.396	1013	0.000005
(3.9) *	(3.8) *	.1, .2	10^{-2}	—	0.622009	0.312098	0.220	45	0.000227
			10^{-3}	—	0.540840	0.270430	0.495	111	0.000003
		.2, .4	10^{-2}	—	0.625724	0.313588	0.220	47	0.000252
			10^{-3}	—	0.548056	0.274020	0.385	89	0.000005
Armijo	(3.10)	.1, .2	10^{-2}	0.5	0.628284	0.315289	0.714	168	0.000276
			10^{-3}	0.75	0.564677	0.282367	1.813	432	0.000018
			10^{-3}	0.3	0.559377	0.279802	0.879	200	0.000012
		.2, .4	10^{-3}	0.5	0.564334	0.282220	1.264	301	0.000017
			10^{-3}	0.75	0.565014	0.282565	1.868	436	0.000018
			10^{-2}	0.3	0.628287	0.315244	0.330	84	0.000276
Fal. Pos.	(3.10)	.1, .2	10^{-2}	—	0.628206	0.315164	0.330	88	0.000275
			10^{-3}	—	0.564614	0.282376	0.769	178	0.000017
		.2, .4	10^{-2}	—	0.628305	0.315234	0.385	88	0.000275
			10^{-3}	—	0.565341	0.282730	0.714	167	0.000018
Armijo	(3.11)	.1, .2	10^{-2}	0.5	0.628273	0.315237	1.319	304	0.000276
			10^{-3}	0.75	0.563914	0.282045	1.374	311	0.000017
			10^{-3}	0.3	0.564539	0.282352	0.604	142	0.000017
		.2, .4	10^{-3}	0.5	0.563800	0.281926	1.758	410	0.000017
			10^{-3}	0.75	0.564019	0.282097	1.319	311	0.000017
			10^{-2}	0.3	0.628484	0.315342	0.220	56	0.000277
Fal. Pos.	(3.11)	.1, .2	10^{-2}	—	0.628405	0.315305	0.330	88	0.000277
			10^{-3}	—	0.564232	0.282178	0.659	155	0.000017
		.2, .4	10^{-2}	—	0.628494	0.31569	0.385	88	0.000278
			10^{-3}	—	0.564114	0.282078	0.859	155	0.000017

Tabela 3.10: Resultados obtidos para o problema (P-2) com o algoritmo BP* implementando o método do gradiente conjugado.

* matriz Hessiana normalizada.

7.3) Problema (P-3):

Cálculo de η	β	$(x^0, \lambda^0)^T$	ε	η	$(x, \lambda)^T$			temp. em s	iter. (P)	iter. (D)	função objetivo
(3.9)	(3.8)	.1, .2, .3	10^{-2}	0.5	0.454989	0.355010	0.090749	15.77	1935	989	0.006046
			10^{-3}	0.75	0.450498	0.350501	0.099101	53.35	8880	1954	0.004901
		.3, .1, .2	10^{-2}	0.5	0.454985	0.350501	0.090723	14.40	1944	823	0.006049
			10^{-3}	0.75	0.450499	0.350501	0.099085	36.64	5982	1438	0.004901
		-.1, .0, .6	10^{-2}	0.5	0.454976	0.355008	0.090939	135.4	12304	10976	0.004052
			10^{-3}	0.75	**						
(3.9) *	(3.8) *	.1, .2, .3	10^{-2}	0.5	0.454876	0.354877	0.090689	1.044	128	91	0.00604
			10^{-3}	0.75	0.450484	0.350484	0.099090	2.967	368	250	0.00490
		.3, .1, .2	10^{-2}	0.5	0.454997	0.354990	0.090447	0.879	1141	71	0.00604
			10^{-3}	0.75	0.450481	0.350480	0.099093	2.473	3081	197	0.00490
		-.1, .0, .6	10^{-2}	0.5	0.454846	0.354855	0.090870	23.79	2204	2157	0.00405
			10^{-3}	0.75	**						
Armijo	(3.10)	.1, .2, .3	10^{-2}	0.5	0.445021	0.345002	0.109736	6.484	848	505	0.00604
			10^{-3}	0.75	0.450344	0.349971	0.099115	1.648	383	3	0.00496
		.3, .1, .2	10^{-3}	0.5	0.449502	0.349500	0.100973	10.66	1383	725	0.00510
			10^{-2}	0.75	0.445029	0.345010	0.109719	4.451	593	341	0.00604
		-.1, .0, .6	10^{-3}	0.5	0.450499	0.350500	0.099037	41.37	3961	3401	0.00490
			10^{-2}	0.75	0.454985	0.354990	0.090215	5.055	675	375	0.00405
Fal. Pos.	(3.10)	.1, .2, .3	10^{-2}	0.5	0.445025	0.345010	0.109741	3.187	384	270	0.00604
			10^{-3}	0.75	0.450499	0.350500	0.099021	4.121	522	327	0.00491
		.3, .1, .2	10^{-3}	0.5	0.449501	0.359500	0.100979	3.022	434	212	0.00491
			10^{-2}	0.75	0.445019	0.345002	0.109747	3.022	374	246	0.00604
		-.1, .0, .6	10^{-3}	0.5	0.450499	0.350499	0.099032	26.81	2553	2330	0.00491
			10^{-2}	0.75	0.454992	0.354996	0.090195	4.651	528	407	0.00405
Armijo	(3.11)	.1, .2, .3	10^{-2}	0.5	0.445009	0.344995	0.109843	6.703	799	563	0.00605
			10^{-3}	0.75	0.449501	0.349500	0.100982	10.60	1193	896	0.00510
		.3, .1, .2	10^{-3}	0.5	0.450501	0.354950	0.109835	17.96	1932	1404	0.00510
			10^{-2}	0.75	0.445015	0.345000	0.109831	4.505	544	373	0.00605
		-.1, .0, .6	10^{-3}	0.5	0.450499	0.350499	0.099022	30.11	3018	2416	0.00490
			10^{-2}	0.75	0.445015	0.355006	0.109807	2.198	325	136	0.00605
Fal. Pos.	(3.11)	.1, .2, .3	10^{-2}	0.5	0.445007	0.344944	0.109853	4.506	526	408	0.00605
			10^{-3}	0.75	0.449502	0.349501	0.100984	8.626	991	770	0.00510
		.3, .1, .2	10^{-3}	0.5	0.459501	0.359500	0.100986	10.82	1202	942	0.00510
			10^{-2}	0.75	0.445025	0.345011	0.109811	3.956	473	341	0.00604
		-.1, .0, .6	10^{-3}	0.5	0.450499	0.350499	0.099018	20.66	2033	1792	0.00490
			10^{-2}	0.75	0.447471	0.348642	0.097141	0.495	110	2	0.00540

Tabela 3.11: Resultados obtidos para o problema (P-3) com o algoritmo BP* implementando o método do gradiente conjugado.

* matriz Hessiana com elementos constante com valores iguais a 0.1.

** resultados não convergentes.

7.4) Problema (P-4):

Cálc. de η	β	$(x^0, \lambda^0)^T$	ε	η	$(x, \lambda)^T$			tempo em s	iter. (P)	iter. (D)	função objetivo
(3.9)	(3.8)	.4, .35, .6	10^{-3}	0.5	0.400612	0.200738	1.000000	32.30	6507	4	-.084276
			10^{-2}	0.75	0.406391	0.207345	1.000000	19.34	3907	4	-.084120
		.2, .1, .4	10^{-2}	0.5	0.406522	0.207346	1.000000	14.95	3017	4	-.084291
			10^{-3}	0.75	0.400626	0.200743	1.000000	25.11	5051	4	-.080423
		.3, .1, .5	10^{-3}	0.5	0.400619	0.200740	1.000000	27.64	5567	4	-.080420
			10^{-3}	0.75	0.400621	0.200742	1.000000	27.36	5519	4	-.080418
(3.9) *	(3.8) *	.4, .35, .6	10^{-3}	0.5	0.399402	0.199302	1.000000	0.879	169	4	-.079602
			10^{-2}	0.75	0.401988	0.205374	0.999992	0.879	140	4	-.082558
		.2, .1, .4	10^{-2}	0.5	0.401508	0.205146	0.999906	0.330	60	6	-.082368
			10^{-3}	0.75	0.399436	0.199341	0.999998	0.495	89	6	-.079624
		.3, .1, .5	10^{-3}	0.5	0.399431	0.199293	0.999999	0.549	101	5	-.079604
			10^{-3}	0.75	0.399491	0.199371	0.999995	0.549	101	5	-.079647
Armijo	(3.10)	.4, .35, .6	10^{-3}	0.5	0.400693	0.200960	0.999302	7.857	1449	49	-.080523
			10^{-2}	0.75	0.409546	0.211586	0.987869	3.901	711	45	-.086654
		.2, .1, .4	10^{-2}	0.5	0.405301	0.207317	0.999952	4.165	857	27	-.084026
			10^{-3}	0.75	0.400481	0.200721	0.999997	3.846	745	20	-.080385
		.3, .1, .5	10^{-3}	0.5	0.400490	0.200728	0.999999	6.703	1253	28	-.080390
			10^{-3}	0.75	0.400473	0.200706	0.999997	4.011	780	21	-.080377
Armijo	(3.11)	.4, .35, .6	10^{-3}	0.5	0.400944	0.201285	0.998308	5.495	1022	41	-.080704
			10^{-2}	0.75	0.410558	0.213218	0.983325	2.192	533	33	-.087538
		.2, .1, .4	10^{-2}	0.5	0.405762	0.207982	0.998210	3.077	566	30	-.084391
			10^{-3}	0.75	0.400575	0.200852	0.999575	2.637	491	23	-.080456
		.3, .1, .5	10^{-3}	0.5	0.400541	0.200800	0.999778	4.286	805	32	-.080429
			10^{-3}	0.75	0.400596	0.200873	0.999532	2.802	507	24	-.080469

Tabela 3.12: Resultados obtidos para o problema (P-4) com o algoritmo BP* implementando o método do gradiente conjugado.

* matriz Hessiana com elementos constante com valores iguais a 0.1.

De acordo com os resultados obtidos, os cálculos computacionais envolvidos neste método foram compensados por uma convergência mais rápida do mesmo em relação ao método BP* original. Pode-se notar, pelos resultados apresentados, que a convergência do método foi consideravelmente mais rápida para Hessianas consideradas no intervalo $[0, 1]$, como já era de se esperar, desde que o aprendizado é feito neste intervalo. Alguns problemas com a convergência, no entanto, foram também verificados.

Quanto à eficiência dos métodos em particular, os métodos de Fletcher-Reeves e de Polak-Ribiere não apresentaram problemas quanto a convergência do algoritmo, no entanto, o método do gradiente com, Hessiana normalizada, mostrou-se bem mais rápido. É importante observar que a aplicação deste método esta condicionada a existência da matriz Hessiana.

3.3 Modificações na Topologia da rede

Além das modificações propostas na seção 3.2, algumas modificações também foram feitas na topologia da rede neural da Figura 2.2.

3.31. (2n) Neurônios na Camada Intermediária

A topologia da rede neural descrita no Capítulo 2, e toda as suas extensões apresentadas até aqui, apresentam, como já citado, exatamente n neurônios em cada camada, onde n é a dimensão do problema a ser resolvido.

Uma modificação foi feita nesta estrutura a fim de se analisar o comportamento da mesma. Alguns testes foram feitos alterando-se o número de neurônios na camada intermediária da rede [6]. Assim sendo, testes foram realizados considerando-se a camada

intermediária contendo $2n$ neurônios e as demais camadas, n neurônios. O exemplo 8 mostra alguns dos resultados obtidos na solução dos mesmos problemas (P-1), (P-2), (P-3), (P-4), resolvidos nos exemplos anteriores.

Exemplo 8:

8.1) Problema (P-1):

$(x^0)^T$	ε	η	$(x)^T$				tempo em s	iteração	função objetivo
.2, .15, .45, .6	10^{-2}	0.5	0.104951	0.200176	0.299989	0.399992	0.495	104	0.00025
	10^{-2}	0.75	0.104871	0.200169	0.299989	0.399992	0.330	69	0.00024
	10^{-3}	0.75	0.100483	0.200010	0.299999	0.399999	0.659	134	0.0
.4, .32, .22, .11	10^{-3}	0.5	0.100496	0.200010	0.299999	0.399999	0.934	202	0.0
	10^{-3}	0.75	0.104943	0.200010	0.299999	0.399999	0.659	134	0.0
	10^{-2}	0.3	0.100498	0.200178	0.299989	0.399992	0.824	175	0.00024
	10^{-3}	0.7	0.100498	0.200010	0.299999	0.399999	0.659	143	0.0

Tabela 3.13: Resultados obtidos para o problema (P-1) com $2n$ neurônios na camada intermediária.

8.2) Problema (P-2):

$(x^0)^T$	ε	η	$(x)^T$		tempo em s	iterações	função objetivo
.1, .2	10^{-2}	0.5	0.640743	0.321005	0.440	111	0.000394
	10^{-3}	0.75	0.565158	0.282650	5.714	1295	0.000018
	10^{-3}	0.3	0.565176	0.282659	14.560	3252	0.000018
.2, .4	10^{-3}	0.5	0.565163	0.282652	8.462	1946	0.000018
	10^{-3}	0.75	0.565159	0.282650	5.769	1293	0.000018
	10^{-2}	0.3	0.640693	0.320980	0.769	183	0.000393

Tabela 3.14: Resultados obtidos para o problema (P-2) com $2n$ neurônios na camada intermediária.

8.3) Problema (P-3):

$(x^0, \lambda^0)^T$	ϵ	η	$(x, \lambda)^T$			tempo em s	iter. (P)	iter. (D)	função objetivo
.1, .2, .3	10^{-2}	0.5	0.454979	0.355012	0.090573	11.923	1364	927	0.004051
	10^{-3}	0.75	0.450497	0.350497	0.099076	24.121	2847	1667	0.004901
.3, .1, .2	10^{-3}	0.5	0.450496	0.359502	0.099072	22.308	3187	1162	0.004901
	10^{-2}	0.75	0.454979	0.355013	0.090577	7.692	935	618	0.004051
-.1, .0, .6	10^{-3}	0.5	*						
	10^{-2}	0.75	0.454997	0.354930	0.090710	86.863	7636	7237	0.004057

Tabela 3.15: Resultados obtidos para o problema (P-3) com $2n$ neurônios na camada intermediária.

* resultados não convergentes.

8.4) Problema (P-4):

$(x^0, \lambda^0)^T$	ϵ	η	$(x, \lambda)^T$			tempo em s	iter. (P)	iter. (D)	função objetivo
.4, .35, .6	10^{-3}	0.5	0.400500	0.200732	1.000000	8.187	1560	19	-0.080393
	10^{-2}	0.75	0.405211	0.207271	0.999998	3.846	728	14	-0.083988
.2, .1, .4	10^{-2}	0.5	0.405688	0.207401	0.999999	3.297	610	20	-0.084140
	10^{-3}	0.75	0.400728	0.200551	1.000000	3.022	566	15	-0.080366
.3, .1, .5	10^{-3}	0.5	0.400503	0.200732	1.000000	5.220	993	20	-0.080394
	10^{-3}	0.75	0.400495	0.200722	1.000000	3.681	691	15	-0.080388

Tabela 3.16: Resultados obtidos para o problema (P-4) com $2n$ neurônios na camada intermediária.

Como o cálculo envolvido numa rede com mais nós ($2n$) é bem maior que numa rede contendo apenas n nós, lembrando que o processamento da rede é simulado através de implementação sequencial, era de se esperar que o tempo necessário para a obtenção da solução ótima fosse, em média, maior para este caso, quando comparados com a rede com n nós em cada camada. É o que acontece com o problema (P-3) no exemplo 8.3. Entretanto, os demais problemas apresentaram resultados mais precisos e, a maioria, apresentou menor número de iterações, como pode ser observado nos exemplos 8.1, 8.2 e 8.4. Portanto, o uso de $2n$ neurônios na camada intermediária é uma alternativa válida na busca de melhores resultados, (precisão e velocidade de convergência). No entanto, problemas de convergência ocorreram neste caso.

3.3.2 Modificações no Algoritmo de Aprendizado (BP* Modificado)

Uma modificação foi proposta aqui no algoritmo BP*, a fim de melhorar o desempenho da rede na resolução de problemas restritos, em especial àqueles contendo mais de uma restrição. O objetivo é acelerar a convergência da rede para esta classe de problemas.

De forma análoga a seção 2.3.1, o seguinte problema é considerado:

$$\begin{aligned} &\min f(x) \\ &\text{sujeito a: } h(x)=0 \end{aligned} \quad (P)$$

e as seguintes funções associadas a (P):

$$L(x, \lambda) = f(x) + \lambda^T h(x): \text{função Lagrangeana}$$

$$\phi(\lambda) = \min_x L(x, \lambda): \text{função dual}$$

No algoritmo BP*, tal como apresentado, para se resolver o problema (P), o subproblema primal é resolvido para cada $\tilde{\lambda}$ fixado até atingir a convergência. Em seguida, um novo $\tilde{\lambda}$ é obtido através de um único passo na direção do subgradiente da função dual (passo dual).

Verificou-se, através de testes realizados, que para evitar grandes oscilações do algoritmo na superfície da função erro, na resolução do problema como um todo, uma alternativa é dar um único passo, tanto na solução do subproblema primal quanto na solução do subproblema dual, a cada iteração da rede. Desta forma não há exigências quanto a convergência do subproblema primal para que $\tilde{\lambda}$ seja atualizado. Assim, para resolver o problema (P), os seguintes subproblemas devem ser resolvidos:

(1) Para λ fixado, atualizar x na direção do gradiente da função Lagrangeana L , uma vez que se deseja resolver

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

Desta forma:

$$x^{k+1} = x^k + \eta \cdot (\nabla f(x^k) + \lambda^T \nabla h(x^k)) \quad (3.12)$$

(2) Para x^{k+1} obtido em (3.12), atualizar λ de acordo com o subgradiente da função ϕ :

$$\lambda^{k+1} = \lambda^k + \eta \cdot h(x^{k+1}) \quad (3.13)$$

uma vez que se deseja resolver:

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

Critério de Parada: O processo é repetido até que $\|h(x)\| \leq \varepsilon$, onde ε é um valor arbitrário pequeno.

Também foi observado que, problemas não lineares restritos cuja solução pertencem ao intervalo $[0,1]$ quando são transformados em irrestritos, pela relaxação lagrangeana de suas restrições, podem apresentar Multiplicadores de Lagrange não pertencentes necessariamente ao mesmo intervalo $[0,1]$. Isto é, para alguns problemas restritos cuja solução pertence a $[0,1]$, o subproblema primal tem solução contida em $[0,1]$, mas o subproblema dual pode apresentar solução fora deste intervalo. É o caso, por exemplo, do problema enunciado no exemplo 9, apresentado a seguir.

Na solução destes tipos de problemas, a alternativa encontrada, foi o uso de funções de ativação diferentes no mesmo algoritmo, ou seja:

- Passo Primal:

função de ativação: $g(x) = \frac{1}{1 + e^{-x}}$, cujas soluções estão contidas em $[0,1]$.

- Passo Dual:

função de ativação: $g(x) = \frac{2A}{1 + e^{-Bx}} - A$; onde $A = 1$ e $B = \frac{2}{3}$ e cujas soluções pertencem ao intervalo $[-1,1]$.

O algoritmo BP* Modificado apresenta-se como segue:

3.3.2.1 Algoritmo BP* Modificado

Algoritmo *Back-Propagation** Modificado

Início (x^0, λ^0)

Enquanto solução não encontrada **faça**

Início

executar o Passo Primal *Forward*.

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

solução foi encontrada

Senão

executar o Passo Dual.

executar o Passo Primal *Backward*.

Fim do enquanto

Fim

O exemplo 9 mostra o desempenho da abordagem proposta.

Exemplo 9:

9.1) Consideremos o problema não linear:

$$\begin{aligned} \min \quad & x_1^2 + x_2^2 + x_3^2 \\ \text{sujeito a:} \quad & \begin{cases} 2x_1 - x_2 = 0.3 \\ x_2 + x_3 = 0.8 \end{cases} \end{aligned}$$

A Tabela 3.17 mostra as soluções aproximadas $(x, \lambda)^T$ obtida com o algoritmo BP* modificado na resolução deste problema, considerando diferentes soluções iniciais $(x^0, \lambda^0)^T$, e diferentes valores para ε e η .

$(x^0, \lambda^0)^T$	ε	η	$(x, \lambda)^T$						iterações P/D	função objetivo
.2, .2, .2, .2, .2	10^{-2}	0.5	0.30888	0.31777	0.47222	-0.30889	-0.94446		12121/ 12120	0.4193763
.3, .3, .3, -.4, -.8	10^{-2}	0.5	0.30888	0.31777	0.47222	-0.30889	-0.94446		12107/12106	0.4193763
.3, .3, .3, -.4, -.8	10^{-3}	0.75	0.31088	0.32177	0.47722	-0.31089	-0.95446		29842/29841	0.4279212
.5, .5, .5, -.5, -.5	10^{-2}	0.5	0.30888	0.31777	0.47222	-0.30889	-0.94446		12103/12102	0.4193763
.5, .5, .5, -.5, -.5	10^{-3}	0.75	0.31088	0.32177	0.47722	-0.31089	-0.95446		29834/29833	0.4279212

Tabela 3.17: Resultados obtidos pelo algoritmo BP* modificado

Pode-se observar que a solução obtida esta muito próxima da solução ótima, que neste caso é dada por:

$$x^* = (0.311111, 0.32222, 0.477778)^T, \lambda_1 = -0.322222 \text{ e } \lambda_2 = -0.955556.$$

Nos vários testes realizados com o algoritmo BP* Modificado, os resultados obtidos convergiram para uma solução muito próxima da solução ótima. Não foi constatado, neste caso, problemas quanto a convergência do algoritmo, demonstrando um bom desempenho da abordagem proposta em relação ao algoritmo BP*, para o qual alguns problema de convergência foram verificados.

3.4 Problemas sem Solução e Problemas com Várias Soluções

O exemplo 10 a seguir mostra o desempenho do algoritmo BP* e de algumas de suas modificações já apresentadas, na solução de um problema de otimização sem solução e de um problema de otimização com mais de uma solução ótima respectivamente.

O objetivo é verificar o comportamento da rede na resolução destes tipos de problemas.

Exemplo 10:

10.1) Consideremos o seguinte problema de otimização:

$$\begin{aligned} \min \quad & x_1^2 + x_2^2 \\ \text{sujeito a:} \quad & \begin{cases} x_1 + 2x_2 = 0.5 \\ 2x_1 + 4x_2 = 0.3 \end{cases} \end{aligned}$$

Este problema é um problema de otimização sem solução. Para este caso foram feitos testes com o algoritmo BP* e BP* Modificado, considerando diferentes soluções iniciais $(x^0, \lambda^0)^T$, diferentes valores para ε e η . A Tabela 3.18 mostra os resultados obtidos.

Alg.	$(x^0, \lambda^0)^T$	ε	η	$(x, \lambda)^T$				Erro Local	Erro Global
BP*	.5, .5, .5, .5	10^{-2}	0.5	0.000072	0.000036	0.500000	0.500000	3.35423	-
BP*	.2, .2, -.8, -.8	10^{-2}	0.75	0.999414	0.999915	-0.80000	-0.80000	2.82876	-
BP*M.	.5, .5, .5, .5	10^{-2}	0.5	0.030068	0.059967	-0.99980	0.46994	0.00025	0.34999
BP*M.	.2, .2, -.8, -.8	10^{-2}	0.75	0.029957	0.060021	-0.99978	0.46990	0.00010	0.35199

Tabela 3.18: Resultados obtidos para o problema sem solução

Os resultados mostram que nenhum dos algoritmos conseguiu atingir a convergência, já que não foi possível diminuir o erro global do sistema em nenhum dos testes realizados. No caso do algoritmo BP* a convergência não foi atingida nem mesmo para o subproblema primal (erro local).

10.2) Consideremos o problema:

$$\begin{aligned} \min \quad & x_1^2 + x_2^2 \\ \text{sujeito a:} \quad & \begin{cases} (x_1 - 0.5)^2 + (x_2 - 0.5)^2 = 0.0625 \\ x_1 + x_2 \geq 0.75 \end{cases} \end{aligned}$$

Pela incorporação de variáveis de folga temos:

$$\begin{aligned} \min \quad & x_1^2 + x_2^2 \\ \text{sujeito a:} \quad & \begin{cases} (x_1 - 0.5)^2 + (x_2 - 0.5)^2 = 0.0625 \\ x_1 + x_2 - x_3 = 0.75 \end{cases} \end{aligned}$$

Para este problema foram feitos testes com o algoritmo BP*, BP* Modificado, BP* com termo *momentum* e BP* com busca linear (Armijo e Falsa Posição). Foram consideradas diferentes soluções iniciais para $(x^0, \lambda^0)^T$, diferentes valores para ε e η . A Tabela 3.19 mostra os resultados obtidos.

Alg.	$(x^0)^T$	ε	η	$(x)^T$					Erro Local	Erro Global
BP*	.5, .5, .5, .5, .5	10^{-2}	0.5	0.00121	0.00127	0.99998	0.50000	0.50000	0.50002	-
BP*	.45, .2, .01, -.8, -.8	10^{-2}	0.75	0.00332	0.34214	0.00001	-0.8000	-0.8000	0.80001	-
BP*M.	.5, .5, .5, .5, .5	10^{-2}	0.5	0.34818	0.34978	0.00016	-0.9976	-0.9992	0.9992	0.5728
BP*M.	.45, .2, .01, -.8, -.8	10^{-2}	0.75	0.34914	0.35043	0.00002	-0.9966	-0.9990	0.0533	0.9999
Mom.	.5, .5, .5, .5, .5	10^{-2}	0.5	0.00157	0.00207	0.00191	-0.9286	-0.9999	2.89563	0.8649
Mom.	.45, .2, .01, -.8, -.8	10^{-2}	0.75	0.00566	0.00709	0.00093	-0.9879	-0.9999	2.95001	0.8516
Armijo	.5, .5, .5, .5, .5	10^{-2}	0.5	0.51366	0.00110	0.00001	-0.9999	-1.0000	0.95129	0.30025
	.45, .2, .01, -.8, -.8	10^{-2}	0.75	0.33218	0.31991	0.00002	0.27307	-0.9999	1.18014	0.09794
Fal. .P.	.5, .5, .5, .5, .5	10^{-2}	0.5	0.46608	0.48575	0.00262	-0.9059	-0.9541	0.2759	0.20762
	.45, .2, .01, -.8, -.8	10^{-2}	0.75	0.46093	0.51151	0.00126	-0.9974	-0.9991	1.01305	0.22944

Tabela 3.19: Resultados obtidos para o problema com mais de uma solução

As soluções ótimas neste caso são:

$$x^* = (0.5, 0.25, 0)^T, \lambda^* = (-1, -1)^T \text{ e } x^* = (0.25, 0.5, 0)^T, \lambda^* = (-1, -1)^T.$$

De maneira análoga ao caso sem solução, nenhum dos algoritmos atingiu a convergência, já que não foi possível atingir a estabilidade do sistema em nenhum dos testes realizados.

3.5 Conclusão

Neste capítulo, algumas modificações no algoritmo BP* foram apresentadas e implementadas. Resultados computacionais foram mostrados a fim de verificar seus efeitos.

As técnicas de otimização aplicadas ao algoritmo BP* proporcionaram, de maneira geral, melhoras no seu desempenho, tanto em termos de tempo de aprendizado, quanto em relação a precisão das soluções. No entanto, o uso destas técnicas ocasionou alguns problemas de convergência ao algoritmo.

Algumas modificações foram feitas também na topologia da rede, e os resultados obtidos mostraram um bom desempenho da mesma, diante de tais modificações.

Testes foram realizados, ainda, com o algoritmo BP* e algumas de suas modificações, na resolução de problemas de otimização sem solução e problemas contendo mais de uma solução ótima. Em ambos os casos nenhum dos testes conseguiu atingir a estabilidade e portanto convergência.

CAPÍTULO 4

Solução de um Sistema Hidroelétrico de Potência

4.1 Introdução

Este capítulo apresenta a solução de um sistema hidroelétrico de potência através de duas abordagens: técnicas clássicas de Programação Não-Linear, cujo esquema de solução foi proposto por Soares [19] e o algoritmo BP* apresentado no Capítulo 2.

4.2 Apresentação do Problema

Considere um parque gerador de energia elétrica constituído de uma usina térmica, uma usina hidroelétrica e com a possibilidade de importação de energia de um sistema vizinho.

Uma política de operação de custo mínimo, atendendo a demanda e outras restrições operacionais do sistema, pode ser obtida pela resolução do seguinte problema:

$$\min \sum_{t=1}^T \{ \psi(g^t) + p^t \cdot i^t \} - w \cdot v(x^{T+1}) \quad (4.1)$$

$$\text{sujeito a: } \begin{cases} 0 \leq g^t \leq \bar{g} & (4.2) \\ 0 \leq i^t \leq \bar{i} & (4.3) \\ h^t + g^t + i^t = d^t; \quad t = 1, 2, \dots, T & (4.4) \\ h^t = f(x^t, u^t) & (4.5) \\ x^{t+1} = x^t + \gamma \cdot (y^t - u^t) & (4.6) \\ \underline{u} \leq u^t \leq \bar{u} & (4.7) \\ p^t = a + b i^t & (4.8) \\ x^1 \text{ dado} & (4.9) \end{cases} \quad (P)$$

onde: γ : 2.592 fator de conversão (m^3 / s) em [$10^6 \text{m}^3 / \text{mes}$].

T: número de períodos (intervalos) de tempo.

g^t : geração térmica no período t , [MW^*].

ψ : função custo de geração térmica.

i^t : intercâmbio recebido no período t [MW].

p^t : tarifa de intercâmbio função do intercâmbio.

$\bar{g}$: capacidade da térmica [MW].

$\bar{i}$: capacidade máxima de intercâmbio [MW].

d^t : demanda de carga no período t [MW].

h^t : geração hidráulica no período t [MW].

f : função de produção hidráulica.

x^t : volume do reservatório no início do período t , [10^6m^3].

y^t : vazão afluente ao reservatório no período t [m^3 / s].

u^t : vazão turbinada pela hidroelétrica no período t [m^3 / s].

$\underline{u}$: limite mínimo de turbinagem.

$\bar{u}$: limite máximo de turbinagem.

w : fator de penalização.

v : benefício futuro da água.

MW^* : MW médios no período t .

4.3 Solução via Técnicas de Programação Não-Linear

Um esquema de solução para o problema (P), com base na teoria de Programação Não-Linear, foi proposto por Soares [19]. Segundo Soares, a solução de (P) pode ser obtida por um método dual, onde a função dual $H(\lambda)$ é obtida dualizando-se (P) em relação à restrição (4.4):

$$H(\lambda) = \min_{\{g, i, x, u\}} L(g, i, x, u, \lambda) \quad (4.10)$$

sujeito a: (4.2), (4.3), (4.5) - (4.9)

onde

$$L(g, i, x, u, \lambda) = \sum_{t=1}^T \{ \psi(g^t) + p^t \cdot i^t + \lambda^t (d^t - h^t - g^t - i^t) \} \quad (4.11)$$

Como o Lagrangeano L é separável em relação as variáveis primais, a sua minimização (função dual) pode ser subdividida em 3 subproblemas independentes, para um dado λ (Multiplicador de Lagrange):

$$H(\lambda) = \min_{\{g\}} \sum_{t=1}^T (\psi(g^t) - \lambda^t g^t) + \min_{\{i\}} \sum_{t=1}^T (p^t \cdot i^t - \lambda^t \cdot i^t) \\ + \min_{\{x, u\}} \sum_{t=1}^T \{ -\lambda^t f(x^t, u^t) - w \cdot v(x^{T+1}) \} + \sum_{t=1}^T \lambda^t d^t$$

Desta forma para resolver o problema (P) os seguintes subproblemas deverão ser resolvidos:

P1: Subproblema Térmico (ST), dado por:

$$\begin{cases} \min \sum_{t=1}^T (\psi(g^t) - \lambda^t g^t) \\ \text{s.a.: } 0 \leq g^t \leq \bar{g} \end{cases} \quad (4.12)$$

ao qual, aplicando-se as Condições de Kuhn-Tucker [10], obtém-se como solução:

$$\dot{g}^t = \max(0, \min(\bar{g}, \tilde{g}^t)); \text{ onde: } \left. \frac{d\psi}{dg} \right|_{\tilde{g}^t} = \lambda^t,$$

P2: Subproblema de Intercâmbio (SI) dado por:

$$\begin{cases} \min \sum_{t=1}^T (p^t \cdot i^t - \lambda^t \cdot i^t) \\ \text{s.a.: } p^t = a + b \cdot i^t \\ 0 \leq i^t \leq \bar{i} \end{cases} \quad (4.13)$$

ao qual aplicando-se, novamente, as Condições de Kuhn-Tucker [10], obtém-se como solução:

$$\dot{i}^t = \max(0, \min(\bar{i}, \tilde{i}^t)); \text{ onde } \tilde{i}^t = \frac{\lambda^t - a}{2b} \text{ e}$$

P3: Subproblema Hidroelétrico (SH), dado por:

$$\begin{cases} \min \sum_{t=1}^T -\lambda^t \cdot f(x^t, u^t) - w \cdot v(x^{T+1}) \\ \text{s.a.: } x^{t+1} = x^t + \gamma \cdot (y^t - u^t) \\ \underline{u} \leq u^t \leq \bar{u} \quad t = 1, 2, \dots, T \\ x^1 \text{ dado} \end{cases} \quad (4.14)$$

No caso do subproblema SH, Soares [19] sugere que a solução seja obtida pelo método do Gradiente Reduzido [10], sem troca de base, adotando-se a seguinte partição:

Partição das variáveis do (SH):

x^i : variáveis básicas (dependentes)

u^i : variáveis não-básicas (independentes)

A matriz que representa as restrições de conservação de água no reservatório (4.6), a menos do fator γ , é dada pela tabela:

	x^2	x^3	x^4	...	x^T	x^{T+1}	u^1	u^2	u^3	...	u^{T-1}	u^T
1	1						1					
2	-1	1						1				
3		-1	1						1			
.				.						.		
.					.						.	
.						.						.
T-1					-1	1					1	
T						-1	1					1

$y^1 + x^1$
y^2
y^3
.
.
.
y^{T-1}
y^T

Colocando-se os volumes na base obtém-se a seguinte tabela:

	x^2	x^3	x^4	...	x^T	x^{T+1}	u^1	u^2	u^3	...	u^{T-1}	u^T
1	1						1					
2		1					1	1				
3			1				1	1	1			
.				.			.	.	.			
.					.		.	.	.	.		
.						.	.	.	.	.	.	
T-1					1		1	1	1	...	1	
T						1	1	1	1	...	1	1

$x^1 + y^1$
$x^1 + y^1 + y^2$
$x^1 + y^1 + y^2 + y^3$
.
.
.
.
.
$x^1 + \sum_{t=1}^T y^t$

O Gradiente Reduzido, calculado para as variáveis independentes $\{u^1, u^2, u^3, \dots, u^T\}$, é da forma:

$$\vec{r} = \nabla_N F - \nabla_B F \cdot B^{-1} N$$

onde $B^{-1}N$ é uma matriz triangular inferior $n \times n$.

Neste caso, a componente t do gradiente reduzido tem a forma:

$$r^t = \frac{\partial F}{\partial u^t} - \gamma \sum_{k=t+1}^{T+1} \frac{\partial F}{\partial x^k}$$

ou seja,

$$r^t = -\lambda^t \cdot \frac{\partial f(x^t, u^t)}{\partial u^t} - \gamma \cdot \left(\sum_{k=t+1}^T -\lambda^k \cdot \frac{\partial f(x^k, u^k)}{\partial x^k} - w \cdot \frac{dv(x^{T+1})}{dx^{T+1}} \right)$$

A direção de caminhada d_N das variáveis independentes será o gradiente reduzido projetado:

$$d_N^t = \begin{cases} 0 & \text{se } \{r^t < 0 \text{ e } u^t = \bar{u}\} \text{ ou } \{r^t > 0 \text{ e } u^t = \underline{u}\} \\ -r^t & \text{c. c.} \end{cases}$$

e a direção d_B das variáveis dependentes será obtida por:

$$d_B = -\gamma B^{-1} N d_N$$

e, portanto: $d_B^t = -\gamma \sum_{k=1}^t d_N^k$

4.3.1 Esquema Geral de Solução:

A resolução do problema dual dado por (4.10 e 4.11) é feita através de uma estrutura hierárquica de cálculo em dois níveis: um nível superior (subproblema coordenador (SC)), onde são fixados valores para o parâmetro λ , e um nível inferior, onde são resolvidos os subproblemas (ST), (SI) e (SH).

Desta forma, para cada parâmetro λ fixado no nível superior (SC), cada um dos outros subproblemas é resolvido. A solução ótima destes subproblemas é então retornada ao nível superior para a fixação de novos valores de λ . O novo parâmetro λ obtido é novamente fornecido a cada um dos subproblemas ST, SI, SH e o processo é repetido.

A Figura 4.1 ilustra este esquema de solução, onde * significa solução ótima.

Figura 4.1 Esquema geral de solução para o problema (P).

No nível superior a determinação dos valores do parâmetro λ é feita no sentido de resolver o problema dual, isto é, de maximizar $H(\lambda)$. Neste caso, então, λ caminha na direção do gradiente da função dual, com tamanho de passo η fixo:

$$\lambda^{k+1} = \lambda^k + \eta \nabla H(\lambda^k)$$

onde $\nabla H(\lambda^k) = d - f(x^*, u^*) - i^* - g^*$

Critério de Parada: Este processo iterativo é repetido até que a norma do gradiente da função dual $H(\lambda)$ seja menor que uma dada tolerância, ou seja:

$$\|\nabla H(\lambda)\| \leq \varepsilon.$$

Este esquema foi implementado em Linguagem C e os resultados obtidos são apresentados na seção 4.6.

4.4 Solução via Redes Neurais Artificiais

Com base no esquema de solução, apresentado na seção 4.3, uma nova abordagem foi implementada neste trabalho, visando a otimização do sistema de potência

apresentado. Esta abordagem utiliza a rede neural apresentada no Capítulo 2 com o algoritmo BP* para obtenção da solução ótima para o problema (P).

O mesmo processo de dualização apresentado na seção (4.3) foi utilizado só que agora usando o modelo de rede neural da Figura 2.2 para resolver os subproblemas ST, SI, SH e SC.

O modelo de rede neural utilizado implementa o algoritmo BP* Modificado e apresenta $4T$ neurônios em cada camada, onde cada grupo de T neurônios consecutivos correspondem as T variáveis de cada um dos subproblemas. Desta forma, os T primeiros neurônios da rede correspondem às variáveis g^t do subproblema ST, para $t = 1, 2, \dots, T$. Analogamente, os T neurônios seguintes correspondem às variáveis i^t do subproblema SI, os T próximos neurônios, correspondem às variáveis u^t do subproblema SH e finalmente os T últimos neurônios da rede correspondem às variáveis λ^t do subproblema SC. Esta disposição de variáveis pode ser vista na Figura 4.2.

Não há comunicação entre neurônios dos subproblemas ST, SI e SH. No entanto todos os neurônios são conectados com todos os neurônios correspondentes ao subproblema SC. A Figura 4.2 ilustra estas conexões.

Para a implementação desta rede uma transformação foi realizada nos dados do problema original para que o mesmo apresentasse variáveis pertencentes ao intervalo $[0,1]$, já que as saídas da rede pertencem a este intervalo. Desta forma, os limitantes inferior e superior das restrições de canalização das variáveis passaram a ser 0 e 1 respectivamente e, portanto, tratadas diretamente pela função de ativação sigmoid.

Portanto, as restrições foram eliminadas e os subproblemas passaram a ser considerados como problemas irrestritos, o que demonstra uma das vantagens de se utilizar esta abordagem.

Figura 4.2: Rede neural para resolver o problema (P).

Uma solução inicial $x_0 = (x_0^1, x_0^2, x_0^3, \dots, x_0^4)^T$ é apresentada como um vetor de entrada para a rede da Figura 4.2. Para cada valor do parâmetro λ fornecido pelo subproblema SC, os demais subproblemas tem seus pesos atualizados por intermédio de apenas um passo na direção dos seus respectivos gradientes. A decisão de se caminhar desta forma na busca de solução para cada subproblema foi tomada em função dos testes realizados no Capítulo 3 (algoritmo BP* Modificado). As soluções atualizadas são fornecidas ao subproblema SC onde também apenas um passo é dado no processo de atualização dos respectivos pesos. Assim, para resolver o problema (P) os seguintes subproblemas devem ser resolvidos:

(1) Para cada λ fixado no subproblema SC, atualizar as variáveis g^t , i^t e u^t , para $t = 1, 2, \dots, T$, na direção do gradiente da função Lagrangeana para cada subproblema:

$$g^{k+1} = g^k - \eta \cdot \nabla(\psi(g^t) - \lambda^t \cdot g^t) \quad (4.15)$$

$$i^{k+1} = i^k - \eta \cdot \nabla(p^t \cdot i^t - \lambda^t \cdot i^t) \quad (4.16)$$

$$u^{k+1} = u^k - \eta \cdot \nabla(-\lambda^t \cdot f(x^t, u^t) - w \cdot v(x^{T+1})) \quad (4.17)$$

onde $x^{t+1} = x^1 + \gamma \cdot \sum_{j=1}^{t-1} (y^j - u^j)$ $t = 1, 2, \dots, T$,

$$p^t = a + b \cdot i^t \text{ e}$$

η é um valor no intervalo $[0,1]$ (taxa de aprendizado).

(2) Para g^{k+1} , i^{k+1} e u^{k+1} obtidos em (4.15), (4.16) e (4.17) respectivamente, λ é atualizado de acordo com o gradiente da função H:

$$\lambda^{k+1} = \lambda^k + \eta \cdot (d - f(x^{k+1}, u^{k+1}) - i^{k+1} - g^{k+1}) \quad (4.18)$$

Critério de Parada: A solução geral do problema é encontrada quando os pesos do subproblema SC se estabilizam, ou seja, o processo converge para a solução do sistema quando:

$$\|\nabla H(\lambda)\| \leq \varepsilon.$$

Desta forma, a atualização dos pesos da rede da Figura 4.2 foi realizada executando-se apenas um passo em cada um dos subproblemas, por iteração, sem considerar a convergência de cada um dos subproblemas individuais.

4.5 Dados e Funções Adotados

Ambas as abordagens apresentadas nas seções 4.3 e 4.4 foram implementadas no desenvolvimento deste trabalho. As implementações foram feitas em Linguagem C em microcomputador 486 DX4. Os seguintes dados e funções foram adotados para cada um dos subproblemas:

Subproblema Térmico:

$$\psi(g) = 0.02 \cdot g^2; \bar{g} = 1000 \text{ MW}.$$

Subproblema Intercâmbio:

$$p = a + b \cdot i; a = 10; b = 0.04; \bar{i} = 500 \text{ MW}.$$

Subproblema Hidroelétrico:

$$h = f(x, u) = (k \cdot u - 0.66 \cdot 10^{-4} \cdot u^2);$$

$$k = 0.85 + 0.25 \cdot 10^{-4} \cdot x;$$

$$v(x^{T+1}) = 20 \cdot x^{T+1} - 0.66 \cdot 10^{-3} (x^{T+1})^2 \text{ e}$$

$$\underline{u} = 200 \text{ m}^3 / \text{s}; \bar{u} = 1500 \text{ m}^3 / \text{s}; w = 1; x^1 = 6000 (10^6 \text{ m}^3).$$

tempo (t)	afluência em t $y^t (\text{m}^3 / \text{s})$	demanda em t $d^t (\text{MW})$
1	762	1850
2	882	1880
3	708	2010
4	774	1980
5	949	1920
6	945	1860
7	819	1790
8	1050	1750
9	1376	1740
10	1066	1780
11	853	1820
12	926	1880

Tabela 4.1: Vazão afluente ao reservatório e demanda de carga em cada período t.

Ambas as abordagens implementadas usaram como tolerância para o critério de parada $\varepsilon = 10^{-2}$ e passo constante $\eta = 0.5$.

Na implementação do esquema descrito na seção 4.3 uma busca unidimensional foi realizada na direção $d = (d_B, d_N)^t$. Esta busca, no entanto, foi feita analiticamente, já que a função objetivo considerada é quadrática.

É importante lembrar que para a resolução via rede neural, estes dados foram transformados para que todas as variáveis do problema estivessem contidas em $[0,1]$. A transformação foi feita em função do menor limitante inferior e maior limitante superior das restrições de canalização das variáveis. Após a obtenção da solução fornecida pela rede neural, a transformação inversa foi realizada para que os resultados pudessem, então, ser comparados com os resultados obtidos através do método de Programação Não-Linear utilizado.

4.6 Resultados Obtidos

A Tabela 4.2 apresenta os resultados obtidos para o problema (P) via métodos de Programação Não-linear e a Tabela 4.3 mostra os resultados obtidos através da rede neural mostrada na Figura 4.2, adotando-se $\varepsilon = 10^{-2}$.

tempo (t)	g (MW)	i (MW)	x(10 ⁶ m ³)	u (m ³ / s)
1	1000.000	382.596	6000.000	482.779
2	969.526	359.763	6723.740	561.348
3	926.153	338.076	7554.870	753.979
4	885.207	317.603	7435.695	790.023
5	840.467	295.234	7394.161	798.549
6	788.864	269.432	7784.131	808.798
7	740.331	245.166	8137.167	804.217
8	701.813	225.907	8175.484	822.174
9	656.556	203.278	8766.006	869.956
10	604.178	177.089	10077.674	961.735
11	567.759	158.879	10347.926	1052.050
12	541.436	145.718	9831.990	1171.172
13			9196.502	

Tabela 4.2: Resultados obtidos para o problema (P) usando técnicas de Programação Não-Linear.

tempo (t)	g (MW)	i (MW)	x(10 ⁶ m ³)	u (m ³ / s)
1	1000.05814	375.01184	6000.0000	449.93953
2	961.85579	355.78514	6808.8605	570.72539
3	918.61787	334.37001	7615.6840	762.01227
4	878.26457	314.12999	7475.6841	797.80921
5	834.16423	291.97349	7413.9704	805.63741
6	782.94959	266.44020	7785.5658	815.44187
7	734.63206	242.34420	8121.3805	810.45511
8	696.68774	223.38149	8143.5291	828.28939
9	650.90458	200.62695	8718.2028	873.75739
10	600.62818	175.16235	10020.0154	967.07028
11	562.91431	156.59535	10276.4444	1056.16549
12	537.99949	143.82994	9749.8383	1176.28562
13			9101.0978	

Tabela 4.3: Resultados obtidos para problema (P) via da rede neural e transformados para o intervalo real do problema.

A tabela abaixo mostra o valor mínimo encontrado para a função objetivo em cada uma das implementações.

Abordagem	Função Objetivo
Programação Não-Linear	85563.67187
Rede Neural	83093.07812

Tabela 4.4: Valor da Função Objetivo

4.7 Discussão dos Resultados Obtidos

Pode-se observar, com base nos resultados apresentados nas Tabelas 4.2, 4.3 e 4.4, que os resultados obtidos estão relativamente próximos nas duas implementações, no entanto, a solução obtida para o problema (P) via métodos de Programação Não-Linear satisfaz as restrições do problema de forma mais precisa do que os resultados obtidos com a rede neural da Figura 4.2. Isto se deve ao fato que, no caso do uso da rede neural, a precisão de $\varepsilon = 10^{-2}$ utilizada foi exigida para o problema contido no intervalo $[0,1]$,

enquanto que no caso da implementação do método de programação não Linear esta precisão foi exigida para os dados reais do problema.

Os resultados obtidos por esta abordagem neural na resolução do problema (P) mostra a utilidade e bom desempenho da rede neural apresentada no Capítulo 2, mostrando que esta rede neural pode ser usada como uma abordagem alternativa em relação a técnicas clássicas existentes na solução de problemas de Programação Não-Linear. O seu desempenho comprovou, também, a eficiência do algoritmo BP* modificado proposto no Capítulo 3.

Redes neurais, em particular a rede neural usada neste trabalho, possuem algumas propriedades que permitem que seu uso seja indicado como uma abordagem alternativa em relação a outras técnicas existentes na solução de vários problemas, tais como, as citadas a seguir.

No caso do uso de redes neurais na resolução de problemas programação não lineares deve-se lembrar que um neurônio é basicamente um processador Não-Linear. Como consequência, uma rede neural formada por interconexões de neurônios é, ela própria, não linear.

Além da não linearidade, a rede neural em discussão possui a propriedade de tratar a canalização de variáveis diretamente pela sua função de ativação. Desta forma restrições podem ser eliminadas tornando, assim, mais simples problemas restritos.

A simplicidade está, também, na não complexidade de código. Basicamente, redes neurais são formadas por processadores de informação (neurônios), que são elementos comuns a toda a rede. Desta forma pode-se dizer que a mesma notação pode ser usada em toda a aplicação.

Uma das principais vantagens, no entanto, no uso de redes neurais, está no fato que pela sua própria estrutura, redes neurais podem ser operadas em paralelo, possibilitando que várias informações sejam acessadas ao mesmo tempo.

4.8 Conclusão

Neste capítulo foi apresentado um problema que consiste na otimização de um sistema hidroelétrico de potência. Na resolução deste sistema, duas abordagens foram discutidas e implementadas. Uma comparação dos resultados obtidos, bem como, discussões desses resultados, foram realizadas.

CAPÍTULO 5

Conclusões e Trabalhos Futuros

Neste trabalho, inicialmente, uma rede neural Multi-Camadas, proposta em [3], foi apresentada, visando a solução de problemas de otimização irrestritos, juntamente com uma generalização da mesma, para resolver problemas de otimização restritos, usando a teoria da dualidade [10] e esquemas do tipo subgradiente (algoritmo BP*). A regra de atualização de pesos desta rede tem como base o método do gradiente simples com passo constante.

Em seguida, algumas modificações foram, então, propostas para o esquema de aprendizado desta rede neural particular. Métodos de otimização foram incorporados ao seu algoritmo de aprendizado, tais como, o *termo momentum*, o gradiente simples com busca linear e o método do gradiente conjugado. Alguns resultados foram apresentados mostrando que diante da aplicação de tais técnicas, o algoritmo apresentou melhores

resultados relativos ao tempo de aprendizado e à qualidade de solução, na maioria dos problemas testados. No entanto, verificou-se que com o uso destas técnicas, esta particular rede pode apresentar problemas de convergência.

A incorporação do *termo momentum* na regra de atualização dos pesos do algoritmo BP*, foi a única técnica proposta que não apresentou problemas quanto a convergência da rede. Os resultados obtidos com a aplicação desta regra foram, em média, melhores em relação a tempo de aprendizado quando comparados com o BP* sem o *termo momentum*.

Duas diferentes buscas lineares, ao longo da direção negativa do gradiente da função erro, foram incorporadas ao algoritmo BP*: o método de Armijo e a regra da falsa posição. De forma geral, o uso de busca linear proporcionou melhorias no algoritmo BP* quanto ao tempo de aprendizado. Os cálculos envolvidos com a busca linear foi compensado, neste caso, por uma convergência mais rápida do algoritmo na maioria dos problemas. Porém, problemas de convergência foram verificados durante a aplicação de tais buscas unidimensionais.

Dentre as técnicas de otimização apresentadas, o método do gradiente conjugado foi a que proporcionou mais rápida convergência e soluções mais precisas ao algoritmo BP*, em particular para Hessianas consideradas no intervalo $[0,1]$. Este método também foi implementado através do método de Fletcher-Reeves e do método de Polak-Ribiere, com as duas diferentes buscas lineares apresentadas, para os quais nenhum problema de convergência foi verificado. Vale ressaltar que, para estes dois últimos métodos não há necessidade de se calcular a matriz Hessiana. Na aplicação de tais métodos, a regra da falsa posição foi a busca linear que proporcionou melhores resultados ao algoritmo, porém, estes resultados não foram melhores do que o gradiente conjugado com Hessiana contida em $[0,1]$, em particular, daquelas com elementos constantes com valores iguais a 0.1, para problemas quadráticos, e Hessiana com elementos normalizados, para problemas não quadráticos.

Modificações foram feitas, também, na topologia da rede apresentada: aumento do número de nós na camada intermediária ($2n$) e modificações na abordagem proposta para resolver problemas restritos (BP* Modificado), onde foram consideradas funções de ativação diferentes para neurônios dos subproblemas primal e dual. Diante de tais modificações, a rede apresentou um bom desempenho. Era de se esperar que em uma rede contendo mais nós, e portanto envolvendo mais cálculos, o tempo de aprendizado fosse em média maior. No entanto, muitos dos problemas testados apresentaram, em média, menor tempo de aprendizado e soluções mais precisas se comparados com o BP* com n nós em cada camada. Porém, alguns problemas com convergência foram verificados neste caso.

Já o algoritmo BP* Modificado não apresentou, durante os testes feitos com problemas restritos, nenhum problema quanto a convergência da solução. Problemas restritos com solução pertencente a $[0,1]$ e Multiplicadores de Lagrange não pertencente ao mesmo intervalo, foram resolvidos, com eficiência, por esta abordagem.

Um sistema hidroelétrico de potência foi, então, resolvido para verificar a eficiência do modelo, objeto de estudo neste trabalho. Este sistema visa a minimização do custo da operação em um parque gerador de energia elétrica, atendendo a demanda de energia e outras restrições operacionais.

Duas abordagens diferentes foram implementadas na resolução do problema de otimização referente ao sistema apresentado. Primeiramente, um esquema de solução através de métodos de Programação Não-Linear, proposto em [19], foi apresentado. Este esquema, que tem como base um método dual, foi implementado e os resultados obtidos foram apresentados. Com base em [19], uma nova abordagem foi então proposta para a resolução do sistema. Esta nova abordagem utilizou uma rede neural artificial com 3-camadas recorrente implementando o algoritmo BP* Modificado na busca de solução ótima para o problema. Resultados obtidos foram apresentados e comparados com os

obtidos via Programação Não Linear, comprovando a eficiência da rede neural apresentada.

Uma possibilidade de trabalho futuro diz respeito a incorporação das técnicas de otimização propostas neste trabalho, no esquema de aprendizado da rede neural usada na resolução deste sistema hidroelétrico de potência. Como foi visto, a rede neural utilizada para resolver este problema tem como método de aprendizado, apenas o método do gradiente simples, uma vez que foi utilizado o algoritmo BP* Modificado. Além disto, a realização de uma implementação em paralelo desta rede possibilitará que problemas de grande escala na área de Sistemas de Potência possam ser resolvidos e resultados mais precisos e rápidos sejam obtidos.

Bibliografia

- [1] Francelin, R. A., Gomide, F. A. C. e Loparo, K. (1990). Notes on System Optimization with Artificial Neural Networks. *Internal Report* 016/90. Unicamp.
- [2] Francelin, R. A., Gomide, F. A. C. e Loparo, K. (1991). System Optimization with Artificial Neural Network. *Proc. Int. Joint Conf. on Neural Network - IJCNN'91*, vol. 3, pp. 2639-2644.
- [3] Francelin, R. A. (1993). Otimização de Sistemas Através de Redes Neurais Artificiais. *PhD Thesis*. Unicamp, Campinas, Brasil.
- [4] Francelin, R. A. e Martinez, L. (1996). Solving a Hydroelectric Potential System by Using Recurrent Multi-Layer Perceptron. *VIII CLAIO - Latin - Iberian - American Congress on Operation Research and System Engeneering, XXVIII SBPO - Simpósio Brasileiro de Pesquisa Operacional*, vol. 3, pp. 1313-1318, Rio de Janeiro-RJ.
- [5] Haykin, S. (1995). *Neural Networks A Comprehensive Foudation*. IEEE Computer Society Press. IEEE Press.
- [6] Hush, R. e Horne, B. (1990). An Introduction to Neural Networks. *UNM Internal Report* no. EECE 90-005.
- [7] Jones, K. L., Lustig, I. J. e Kornhauser, A. L. (1990). Optimization Techniques Applied to Neural Network: Linear Search Implementation for Backpropagation. *Proc. Joint Conf. Neural Network*, pp. 939-939, vol. III. San Diego, CA.
- [8] Lippmann, R. P. (1987). An Introduction to Computing with Neural Nets. *IEEE ASSP Magazine*, pp. 4 - 22.
- [9] Lu, B. L. e Ito, K. (1995). Transformation of Nonlinear Programming Problems into Separable Ones Using Multilayer Neural Networks. *Annals of Mathematics and Artificial Intelligence*

- [10] Luenberger, D. C. (1984). *Introduction to Linear and Nonlinear Programming*. Addison-Wesley.
- [11] Marks, R. J., Oh, S. e Atlas, L. E. (1989). Alternating Projection Neural Networks. *IEEE Transaction on Circuits and Systems*, vol. 36, pp. 846-856.
- [12] Martinez, L. e Francelin, R. A. (1996). Optimization Techniques Applied to Multi-Layer Perceptron for Solving Nonlinear Programming Problems. Submetido ao *WCNN'96 World Congress on Neural Networks*. San Diego-USA. Setembro 1996.
- [13] Martinez, L. e Francelin, R. A. (1996). Recurrent Multi-Layer Perceptron in Optimization of a Hydroelectric Potential System. Apresentado ao *ICONE'96 - Second International Conference on Non-Linear Dynamics, Chaos, Control and their Applications in Engineering Sciences*. São Pedro - São Paulo - Brasil, 5 à 8 de agosto, 1996.
- [14] Martinez, L. e Francelin, R. A. (1996). Solving a Hydroelectric Potential System by Neural Network. Aceito para ser apresentado ao *IFAC - International Federation of Automatic, XI Congresso Latino Americano de Automática. AADECA'96*. Buenos Aires - Argentina, 9 à 13 de setembro, 1996.
- [15] Noyes, J. L., Neural Network Optimization Methods. *Department of Mathematics and Computer Science*. Wittenberg University, Springfield. OH 45501.
- [16] Poljak, B. T. (1969). Minimization of Unsmooth Functionals. *USSR Computational Mathematics and Mathematics Physics*, vol 9, No. 3.
- [17] Ridmiller, M. (1994). Advanced Supervised Learning in Multi-Layer Perceptrons - From Backpropagation to Adaptive Learning Algorithms. *Neural Networks* (5).
- [18] Rumelhart, D. E., Hinton, G. E., e Williams, R. J. (1986). Learning international representations by error programming. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Capítulo 8 (D.E.Rumelhart and J.L.McClelland, Eds), Cambridge, Massachusetts: The MIT Press.
- [19] Soares, S., Otimização da operação de sistemas hidroelétricos. *Notas de Aula do Curso de Otimização*. Unicamp.
- [20] Stornetta, W. S. e Huberman, B. A. (1987). An improved Three-Layer, Backpropagation Algorithm. *IEEE First Conferation on Neural Networks*, vol II, pp. 637-643. San Diego.