

UMA FERRAMENTA DE EXTRAÇÃO DE REGRAS DE REDES NEURAIIS

Cristiano Araújo Maciel Alves

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA CIVIL.

Aprovada por:

Prof. Nelson Francisco Favilla Ebecken, D. Sc.

Prof. José Murilo Feraz Saraiva, D. Sc.

Prof. Antonio César Ferreira Guimarães, D. Sc.

RIO DE JANEIRO, RJ – BRASIL

AGOSTO DE 2001

ALVES, CRISTIANO ARAÚJO MACIEL

Uma Ferramenta de Extração de Regras
de Redes Neurais [Rio de Janeiro] 2001

VIII, 68 p. 29,7 cm (COPPE/UFRJ, M.Sc.,
Engenharia Civil, 2001)

Tese - Universidade Federal do Rio de
Janeiro, COPPE

1. Extração de Regras de Redes Neurais

2. Ferramenta

I. COPPE/UFRJ II. Título (série)

A meu pai,
por ter deixado a maior herança: uma família.

AGRADECIMENTOS

Ao Professor Nelson, pela confiança, incentivo constante e pela orientação motivadora.

Ao Professor Mourão, pelo incentivo ao mestrado.

Ao Professor Marcelo Cabral, por ter me dado as informações necessárias para obter o conhecimento que almejava.

Aos meus amigos, em especial: Otto, Tales, Marcus e Olavo.

À CAPES pela ajuda financeira.

A todos os funcionários da COPPE que, de alguma forma, contribuíram para a realização deste trabalho, em especial: Jony e Thelmo.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M. Sc.)

UMA FERRAMENTA DE EXTRAÇÃO DE REGRAS DE REDES NEURAIIS

Cristiano Araújo Maciel Alves

Agosto/2001

Orientador: Nelson Francisco Favilla Ebecken

Programa: Engenharia Civil

Este trabalho trata de um desenvolvimento de uma ferramenta para extração de regras de redes neurais. A ferramenta é desenvolvida com o auxílio do módulo de redes neurais no ambiente Matlab com o objetivo de treinar a rede; e para extrair as regras utiliza-se a sua linguagem de programação, para implementar o algoritmo RX MODIFICADO.

Esta ferramenta foi desenvolvida com o intuito de extrair regras de redes neurais de uma forma rápida e otimizada para o futuro usuário. A ferramenta proposta, neste trabalho permite que o usuário obtenha regras mais ou menos precisas, e respectivamente mais ou menos complexas, ou seja, de acordo com a necessidade do usuário.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

A TOOL OF RULES EXTRACTION OF NEURAL NETWORKS

Cristiano Araújo Maciel Alves

August/2001

Advisor: Nelson Francisco Favilla Ebecken

Department: Civil Engineering

This work treats of a development of a tool for extraction of rules of neural networks. The tool is developed with the aid of the module of neural networks of Matlab with the objective of training the network; and to extract rules the programming language it is used, to implement the MODIFIED RX algorithm.

This tool was developed with the intention of extracting rules of neural networks in a fast way and optimized for the future user. The proposed tool in this work allows to obtain rules more or less precise, and respectively more or less complex, in agreement with the user's needs.

ÍNDICE

CAPÍTULO 1 - INTRODUÇÃO	1
1.1 Motivação.....	1
1.2 Objetivo.....	1
1.3 Organização.....	1
CAPÍTULO 2 – REDES NEURAIIS	3
2.1 Conceito.....	3
2.2 Tipos de Redes Neurais.....	9
2.2.1 Algoritmo de Retropropagação.....	9
2.3 Aplicações.....	14
CAPÍTULO 3 – EXTRAÇÃO DE REGRAS	15
3.1 Conceito.....	15
3.2 Algoritmos de Extração de Regras.....	16
3.2.1 Algoritmo SUBSET.....	16
3.2.2 Algoritmo Mofn.....	16
3.2.3 Algoritmo LOGIC.....	17
3.2.4 Algoritmo PERCEP.....	17
3.2.5 Algoritmo KT.....	17
3.2.6 Algoritmo TREPAN.....	18
3.2.7 Algoritmo RX.....	19
3.3 Algoritmo de Extração de Regras Implementado.....	20

CAPÍTULO 4 – IMPLEMENTAÇÃO NO MATLAB	24
4.1 Introdução.....	24
4.2 Programa Implementado.....	24
4.3 Funcionamento Otimizado no Matlab.....	26
CAPÍTULO 5 – ESTUDO DOS CASOS	37
5.1 Exemplos: Iris Plants Database, Thyroid Gland Data e Wine Recognition Data.....	37
5.1.1 Resultados.....	38
5.2 Comentários Finais.....	53
CAPÍTULO 6 – CONCLUSÃO	54
BIBLIOGRAFIA	56
APÊNDICE I – IMPLEMENTAÇÃO COMPUTACIONAL	58

CAPÍTULO 1

INTRODUÇÃO

1.1 MOTIVAÇÃO

A necessidade da obtenção de regras de redes neurais de uma forma clara, rápida e em ambiente amigável ao usuário, foi a motivação principal do presente trabalho. As redes neurais são reconhecidas como eficiente recurso de desenvolvimento de modelo de dados, mas quase sempre apresentadas como caixas-pretas. A explicitação do conhecimento apreendido no seu treinamento é hoje em dia uma atividade de pesquisa de grande importância.

1.2 OBJETIVO

Este trabalho tem por objetivo desenvolver uma ferramenta para extração de regras de redes neurais. A ferramenta é formulada com o auxílio do programa Matlab e utiliza no seu contexto o algoritmo RX MODIFICADO. Esta ferramenta foi desenvolvida com o intuito de facilitar enormemente o futuro usuário para extrair regras de redes neurais implementadas pelo Matlab.

1.3 ORGANIZAÇÃO

Com o desenvolvimento de algoritmos de extração de regras de redes neurais, surgiu o interesse em fazer uma otimização entre os programas de redes neurais e programas de extração de regras. O intuito principal é extrair as regras no mesmo programa onde foi treinada a rede neural. Com isso foi desenvolvido um programa de

extração de regras denominado de RX MODIFICADO, no mesmo ambiente que pode ser treinada a rede neural. O ambiente é o sistema Matlab, pois o mesmo tem um módulo de redes neurais e é um programa que propicia na sua linguagem de programação uma ótima interface com algoritmos.

O trabalho está subdividido em capítulos conforme descrito a seguir.

No capítulo 2 é demonstrado o conceito de redes neurais e seus tipos mais difundidos. É explicado detalhadamente o algoritmo de Retropropagação, pois o mesmo será usado para o treinamento da rede neural. Por fim comenta-se as prováveis aplicações de redes neurais.

No capítulo 3 é demonstrado o conceito de extração de regras e os tipos de algoritmos de extração de regras mais difundidos. É explicado detalhadamente o algoritmo implementado denominado de Algoritmo RX MODIFICADO, pois o mesmo será usado para extração de regras.

No capítulo 4 trata da implementação no ambiente Matlab. O Desenvolvimento desta implementação, como também a sua otimização, inicia-se no treinamento da rede neural no respectivo módulo do Matlab até a extração de regras no programa RX MODIFICADO implementado no Matlab.

No capítulo 5 trata dos estudos dos casos utilizando a metodologia implementada.

Por fim, no capítulo 6, são apresentadas as conclusões.

CAPÍTULO 2

REDES NEURAIS

2.1 CONCEITO

O trabalho em redes neurais tem sido motivado desde o começo pelo reconhecimento de que o cérebro humano processa informações de uma forma própria. O cérebro é um computador altamente complexo, não linear e paralelo. Ele tem a capacidade de organizar seus constituintes estruturais, conhecidos por neurônios, de forma a realizar certos processamentos muito mais rapidamente que o mais rápido computador digital hoje existente. Uma representação genérica de um neurônio humano é mostrada na Figura 2.1. Onde os dendritos apicais e basais são zonas receptivas, o corpo celular é onde inicia-se a codificação da saída, os axônios constituem a linha de transmissão e os terminais sinápticos transmitem o aprendizado para outro neurônio.

Uma rede neural é uma máquina que é projetada para modelar a maneira como o cérebro realiza uma tarefa particular ou função de interesse; a rede é normalmente implementada utilizando-se componentes eletrônicos ou é simulada por programação em um computador. Os modelos de redes neurais, realizam a manipulação de informações através da interação de um grande número de unidades básicas de processamento. Essas unidades básicas possuem o nome de neurônios artificiais, e é de fundamental importância para a operação de uma rede neural. Os neurônios em uma rede neural estão organizados em camadas que se interligam, como apresentado na Figura 2.2. As camadas inferiores são compostas de neurônios que se ligam com os neurônios da camada superior, como em uma estrutura de camadas.

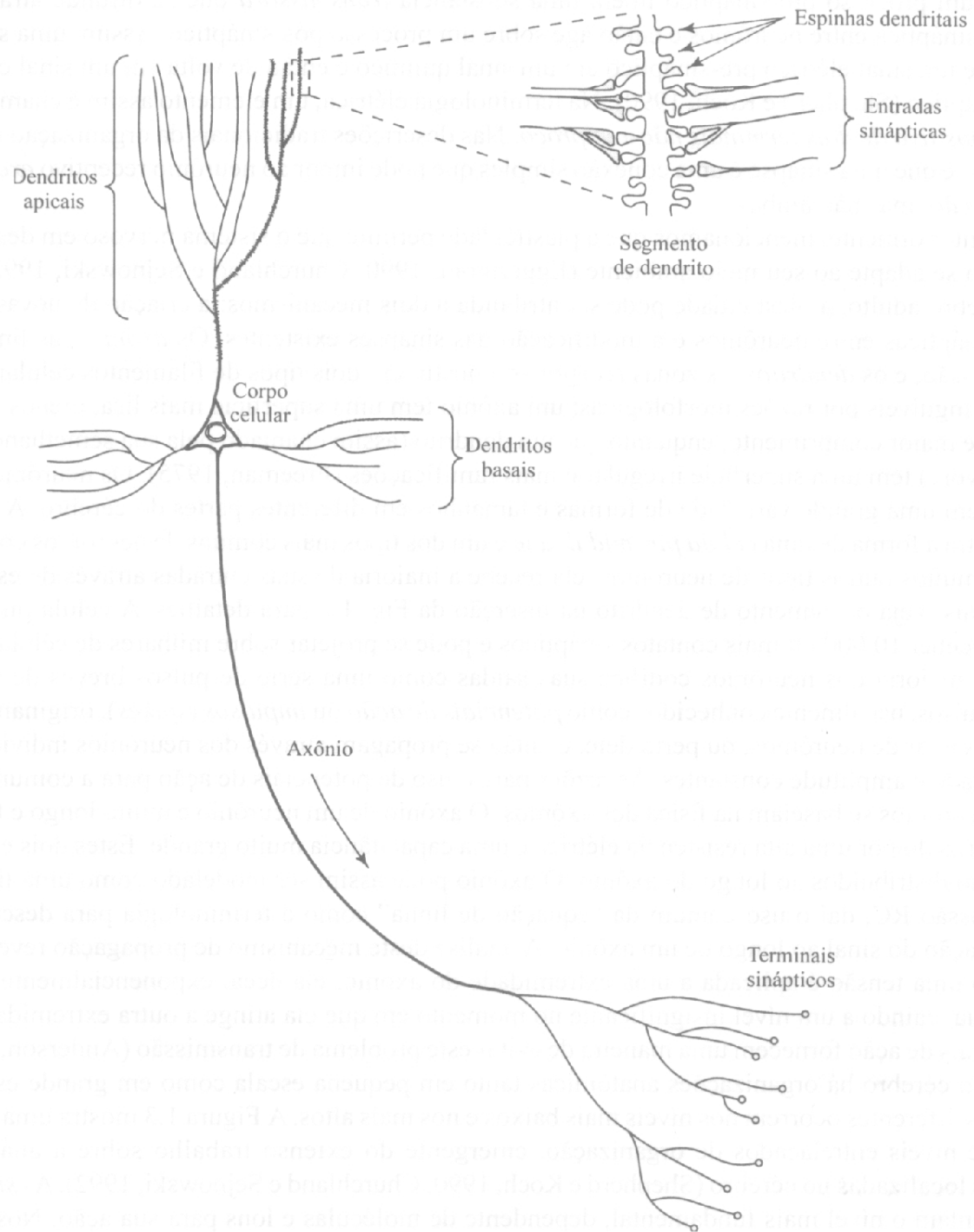


Figura 2.1: Neurônio humano

Cada neurônio possui um estado interno chamado valor de ativação, que é modificado sempre que uma nova entrada é recebida. Essas entradas são combinadas e um novo valor de ativação é calculado através de uma função. A saída

do neurônio também é calculada por uma função a partir do valor de ativação. Esta saída, por sua vez, serve de entrada para o neurônio seguinte e assim por diante.

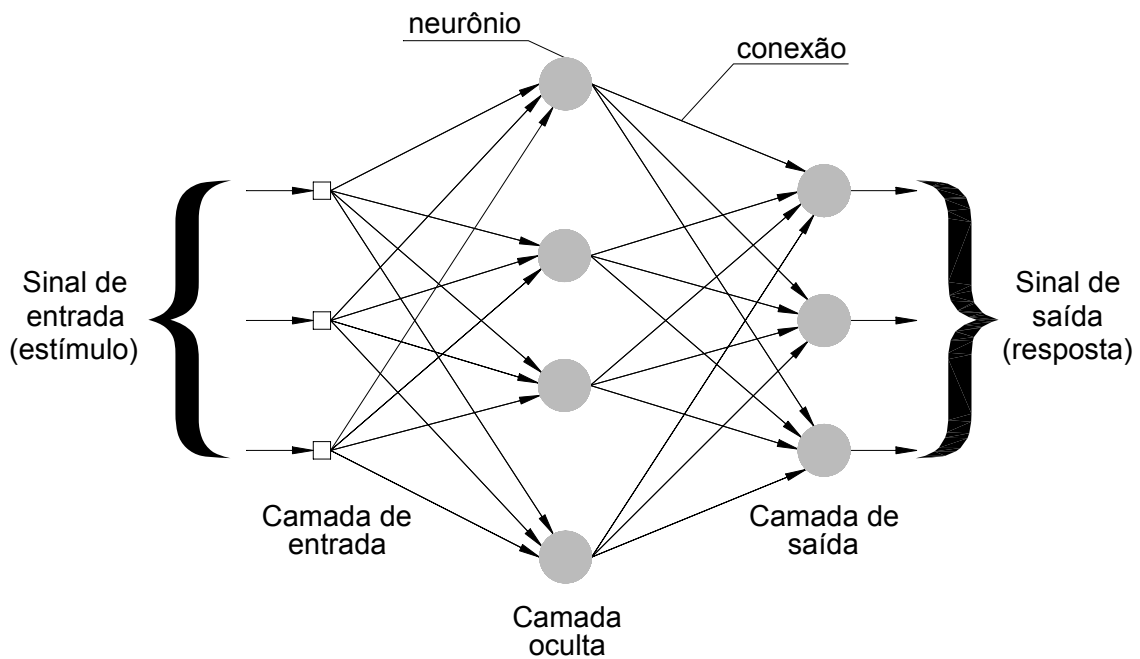


Figura 2.2: Rede Neural

Uma rede neural pode se apresentar em várias configurações, tais como alimentada adiante - *feedforward* ou recorrente como mostra Figura 2.3. A configuração alimentada adiante - *feedforward* se caracteriza pelos neurônios das camadas inferiores se interligarem somente com os neurônios da camada imediatamente superior. A configuração é chamada recorrente quando um neurônio pode receber entradas de qualquer outra camada da rede. A configuração com realimentação acontece quando os neurônios da entrada recebem sinais vindos diretamente dos neurônios da saída.

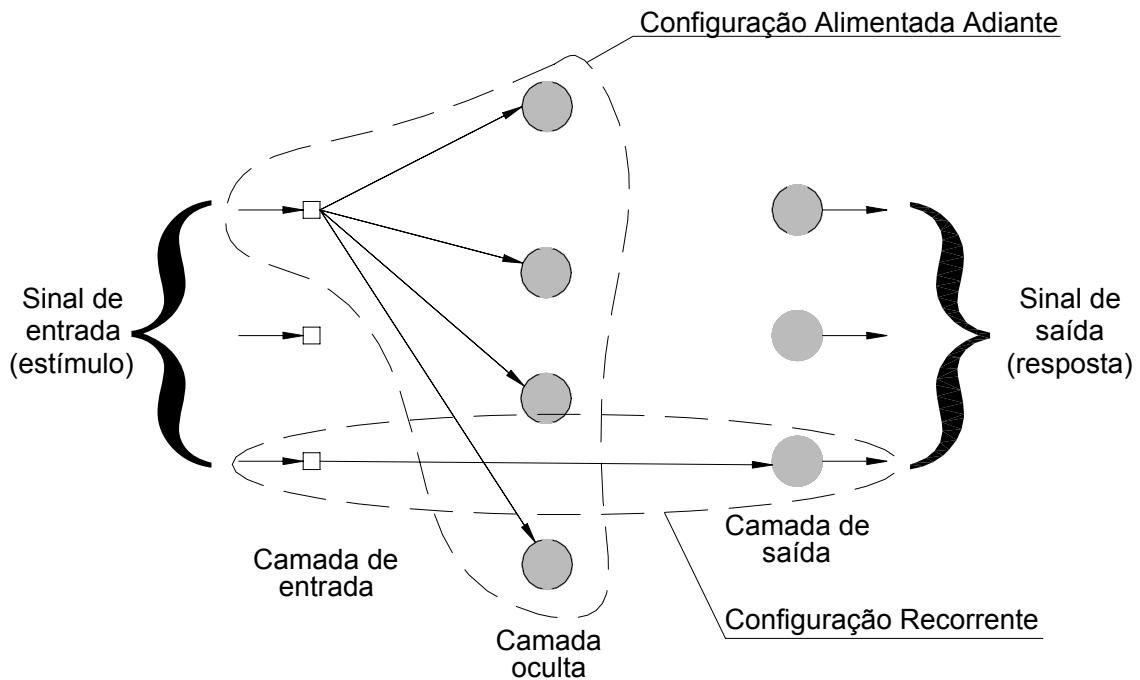


Figura 2.3: Configurações da Rede Neural

A característica principal de uma rede neural é a sua capacidade de aprender. O treinamento de uma rede consiste no ajuste dos parâmetros internos da rede, de maneira que a rede apresente um resultado esperado dada a apresentação de um conjunto de padrões específicos. Os padrões de treinamento da rede contém as informações que se deseja que uma rede aprenda. Os parâmetros a ajustar são os pesos das conexões que interligam os neurônios. Os diversos modelos de redes neurais se caracterizam pela utilização de diferentes técnicas de treinamento. O treinamento genericamente pode ser classificado como supervisionado ou não supervisionado. O treinamento é supervisionado quando o ajuste de parâmetros é feito a partir da apresentação de um conjunto de pares de entradas e saídas padrão. Neste processo uma entrada padrão é apresentada à rede e uma saída é calculada. A diferença existente entre a saída calculada e a saída padrão é o erro produzido, que se quer minimizar. A Figura 2.4 mostra um diagrama em blocos do treinamento supervisionado esquema da rede neural supervisionada.

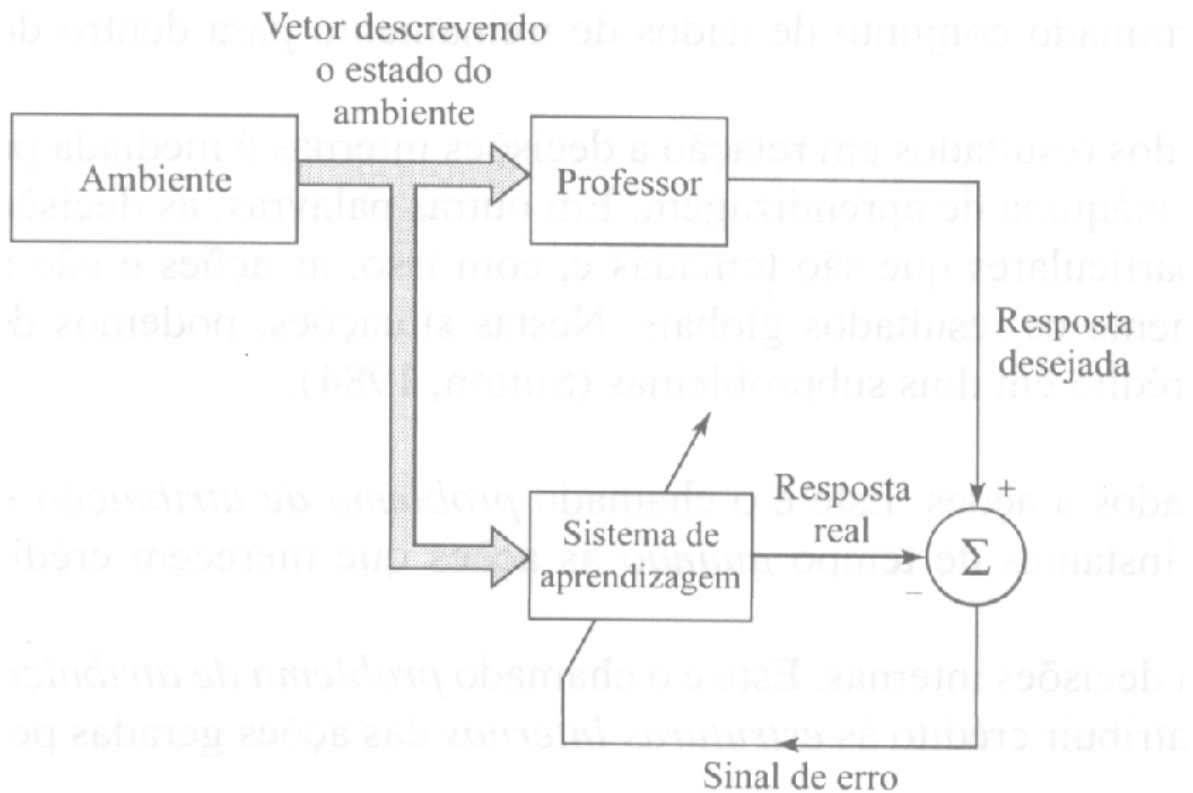


Figura 2.4: Diagrama em blocos do treinamento supervisionado

O treinamento é não supervisionado quando o conjunto de padrões de treinamento possui somente entradas. Neste processo não existe saída padrão, ou seja, não é mostrado a rede neural um alvo para se alcançar. O processo utiliza a comparação entre sinais para a construção de grupos de similaridade.

Depois do treinamento, sendo este bem sucedido, diz-se que a rede treinou segundo o conjunto de padrões de treinamento. Isto quer dizer que seus parâmetros internos, os pesos das conexões, estão ajustados de forma a produzir saídas com um percentual de erro aceitável. O sucesso para um funcionamento adequado da rede neural é a quantidade de camadas e de neurônios que esta possui. O dimensionamento de uma rede neural deve ser iniciado com uma camada oculta e poucos neurônios nesta camada; caso a rede não treine é necessário inicialmente aumentar o número de neurônios, um a um, e refaz-se o treinamento. Em poucas

situações é necessário mais de uma camada oculta e, teoricamente, mais de duas camadas nunca será necessário.

Para a operação de uma rede neural é de fundamental importância o neurônio que é uma unidade de processamento de informação. Uma representação de um neurônio de uma rede neural é mostrada na Figura 2.5. Onde a_i são as entradas na rede neural, w_{ji} são os pesos das conexões, b_j é o bias, net_j é o valor de ativação e y_j é a saída real.

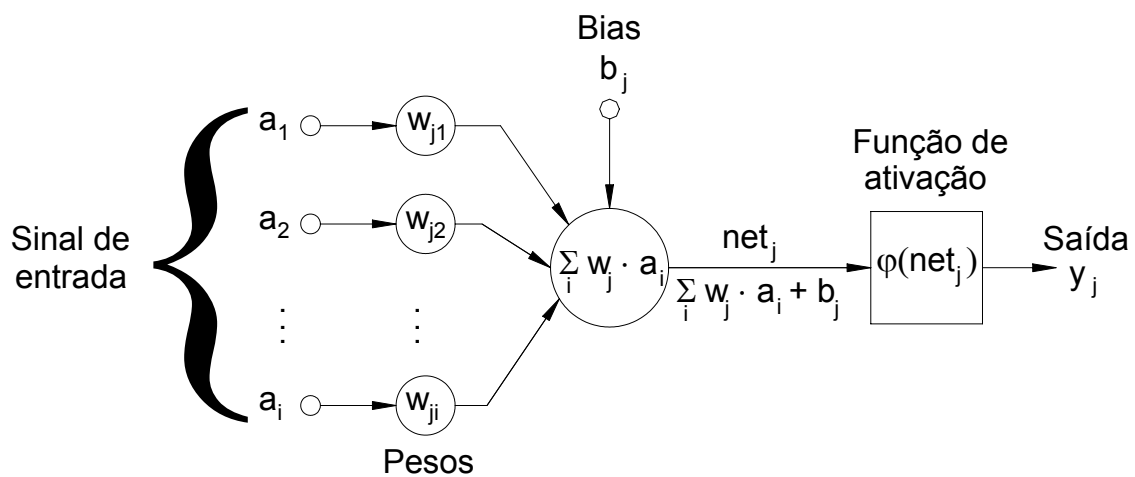


Figura 2.5: Neurônio Artificial

Cada unidade da rede recebe sinais das unidades anteriores. As unidades estão interligadas através de conexões que possuem um peso. O peso define o efeito que a saída de uma unidade exerce sobre a entrada da unidade seguinte. A combinação das entradas, pela soma ponderada das mesmas, gera uma entrada total que serve de base para a modificação do estado interno da unidade, o seu valor de ativação. A saída de uma unidade é uma função do seu valor de ativação. Os pesos das conexões entre as unidades determinam toda a manipulação de valores da rede neural. Estes pesos têm importância fundamental, tanto na fase de treinamento, onde

são ajustados, quanto na fase de produção, em que guardam a memória da rede neural.

2.2 TIPOS DE REDES NEURASIS

As redes neurais podem ser classificadas de acordo com:

- (i) Tipo de Treinamento:
 - Treinamento supervisionado;
 - Treinamento não supervisionado.

- (ii) Algoritmo de Aprendizado:
 - Retropropagação;
 - Kohonen;
 - ART;
 - Redes de base radial e etc.

Estas redes neurais são as mais usuais e solucionam a maioria dos problemas que necessitam de redes neurais; a determinação de qual tipo de rede deve ser adotada é determinada pelo caso presente, ou seja, cada caso é um caso. O ideal é ver qual delas funciona melhor, ou seja, o método utilizado é tentativa e erro. No caso do treinamento supervisionado estes são considerados mais seguros, pelo simples fato de ter uma saída padrão do problema proposto; mesmo assim em certos casos tem que se usar os treinamentos não supervisionados.

2.2.1 ALGORÍTMO DE RETROPROPAGAÇÃO

A configuração da rede neural para o algoritmo de retropropagação é do tipo alimentada adiante – *feedforward*. Ela consiste de um conjunto de unidades sensoriais que constituem a camada de entrada, uma ou mais camadas ocultas de nós computacionais e uma camada de saída de nós computacionais. O sinal de entrada se propaga para frente através da rede, camada por camada e o neurônio em qualquer camada da rede está conectado a todos os neurônios da camada anterior. A Figura 2.6 mostra o grafo arquitetural de uma rede neural alimentada adiante.

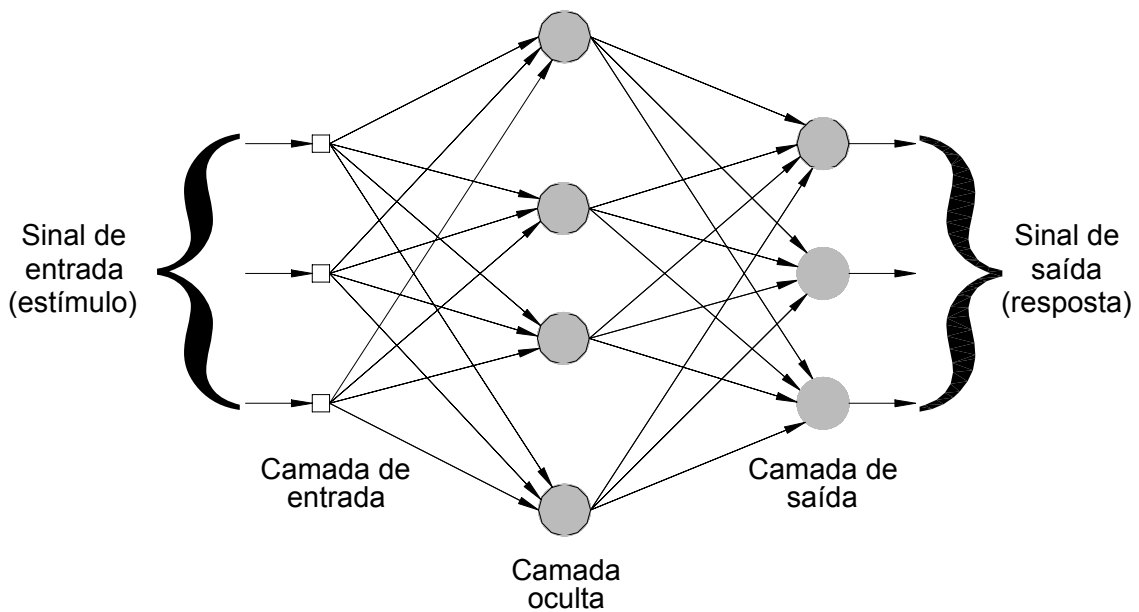


Figura 2.6: Grafo Arquitetural de uma Rede Neural Alimentada Adiante

Neste tipo de rede neural existe dois tipos de sinais: os sinais funcionais e os sinais de erro. O sinal funcional é um sinal de entrada que propaga-se para frente através da rede neural e termina na camada de saída como um sinal de saída. O sinal de erro inicia-se num neurônio de saída e se propaga para trás através da rede neural. A Figura 2.7 mostra uma parte da rede neural com uma configuração alimentada adiante e com os dois tipos de sinais, ou seja, propagando para frente os sinais funcionais e retropropagando os sinais de erro.

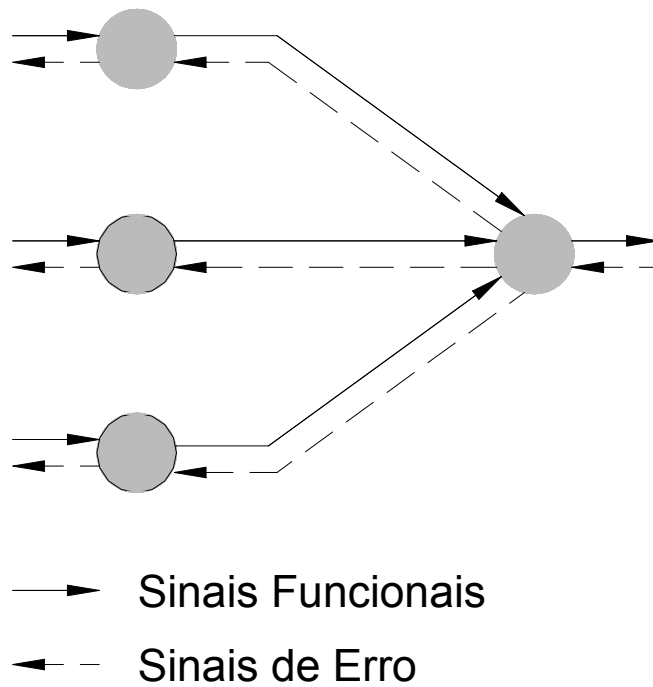


Figura 2.7: Propagação e Retropropagação dos Sinais

O presente trabalho está limitado ao algoritmo de retropropagação, pois o mesmo é muito popular e têm sido aplicado com sucesso para resolver diversos problemas complexos. O algoritmo de retropropagação é um procedimento iterativo que utiliza pares de valores de entrada e saída desejada como padrões para o treinamento. Um padrão é apresentado na entrada da rede, através de seus valores de entrada, e é calculada uma saída da rede. A saída calculada é comparada com a saída desejada do padrão e, havendo diferença, os pesos das conexões entre as unidades são modificados para que esta diferença seja minimizada. A apresentação desses pares é repetida até que as diferenças atinjam um valor mínimo desejado. A Figura 2.8 mostra a representação do algoritmo de retropropagação.

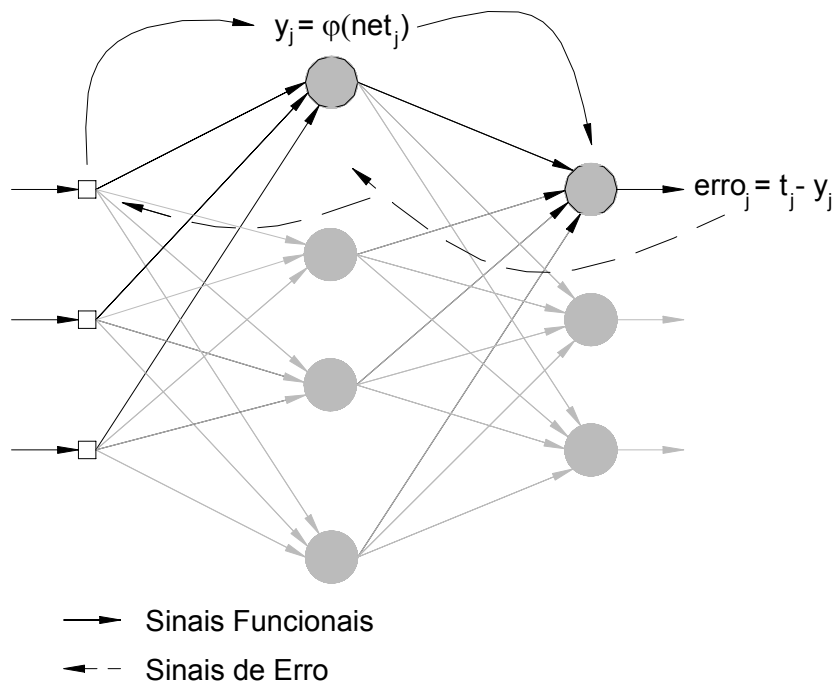


Figura 2.8: Funcionamento do Algoritmo de Retropropagação

O algoritmo de retropropagação é executado em duas fases. A primeira fase corresponde à apresentação da entrada do par padrão de treinamento na entrada da rede neural e a sua propagação até a saída da rede, gerando valores de saída. Os valores de saída são então comparados com os valores da saída do par padrão de treinamento, calculando-se o erro de saída. Na segunda fase este erro é retropropagado pela rede neural, de forma que se faça o cálculo da diferença nos pesos e sua efetiva atualização.

A diferença entre a saída desejada, t_j , dada pelo padrão, e o valor de saída calculado, y_j , é o erro na saída que se quer minimizar, dado por:

$$\text{erro}_j = t_j - y_j$$

O erro médio quadrático por toda a saída é:

$$\text{erro} = \frac{1}{2} \sum_j (\text{erro}_j)^2$$

Um dos critérios mais utilizados para a verificação do término do treinamento da rede neural é a comparação do erro médio quadrático a um valor aceitável. Desta

forma, o resultado do treinamento de uma rede neural é um conjunto de valores de pesos de conexões que fornece uma saída cujo erro máximo é um valor aceitável. As funções utilizadas no algoritmo de retropropagação são a função logística, a função tangente hiperbólica e a função linear. A função logística tem intervalo de variação em [0,1], a função tangente hiperbólica e linear possui intervalo de variação em [-1,1]. A Figura 2.9 mostra o gráfico das três funções.

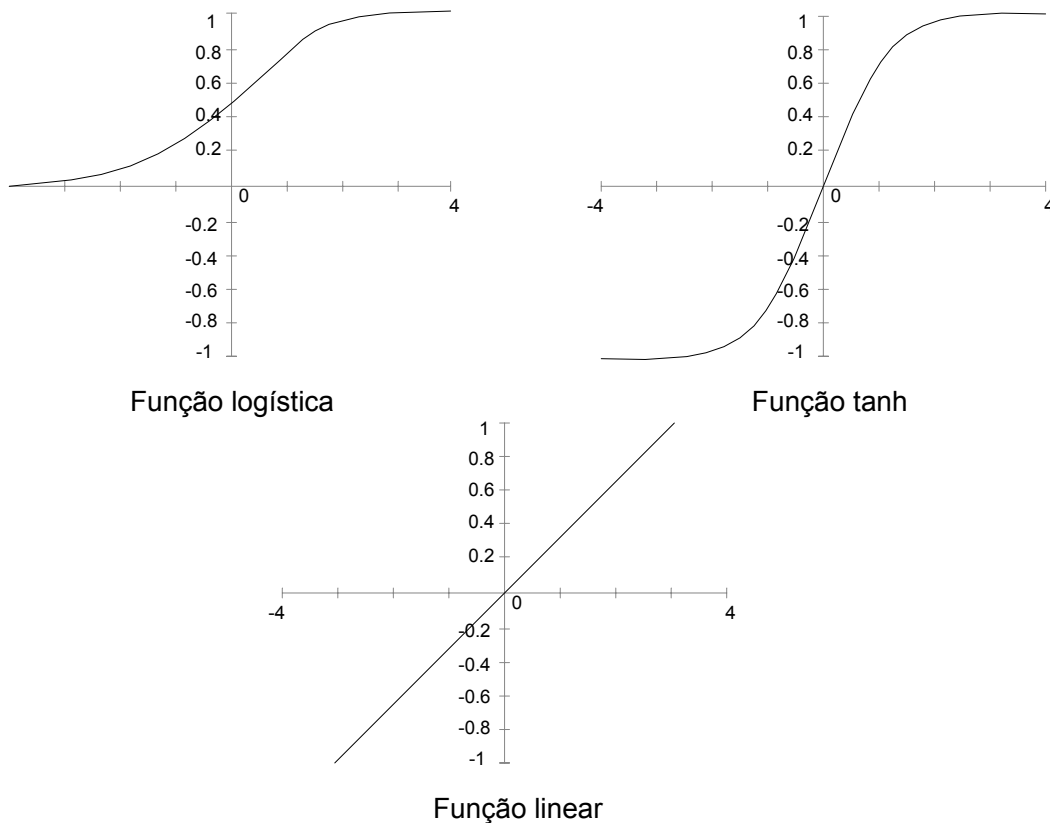


Gráfico 1.1: Funções de Ativação

O valor de ativação é definido por:

$$net_j = \sum_i w_{ji} \cdot a_i + b_j$$

Onde cada b_j é denominado bias, todos os a_i representam as entradas da rede neural e os w_{ji} representam os pesos das conexões. Tanto o bias como os pesos das conexões são ajustados durante o treinamento.

2.3 APLICAÇÕES

As redes neurais encontram aplicações em campos tão diversos, como modelagem, análise de séries temporais, reconhecimento de padrões, processamento de sinais e controle, em virtude de uma importante propriedade: a habilidade de treinar a partir de dados de entrada com ou sem um professor.

CAPÍTULO 3

EXTRAÇÃO DE REGRAS

3.1 CONCEITO

Redes neurais têm sido freqüentemente criticadas devido à não apresentar as razões de suas conclusões. Então surge o interesse em extrair conhecimento compreensível de redes neurais treinadas, aplicando-se para isso algum algoritmo de extração de regras. As regras extraídas são do tipo SE...ENTÃO, onde a parte SE especifica um conjunto de condições sobre valores de atributos previsoires e a parte ENTÃO especifica um valor previsto para o atributo classe. Os atributos previsoires são as premissas da regra que devem ser obedecidas, para assim obter um atributo classe. Em 1988, GALLANT [7] desenvolveu uma abordagem bastante simples no tocante ao entendimento do conhecimento armazenado em redes neurais. Para tal, configura-se a rede neural de modo que cada nó represente uma entidade conceitual. Por exemplo, as unidades de entrada poderiam representar sintomas, as unidades ocultas às doenças e as unidades de saída representariam o tratamento. A tarefa da rede se resume no aprendizado das relações entre estas entidades conceituais. A extração de regras é realizada através da interpretação dos pesos como sendo informações correlacionais entre as entidades conceituais.

Extração de regras é a obtenção de regras do respectivo problema, proposto na rede neural depois do seu aprendizado.

3.2 ALGORITMO DE EXTRAÇÃO DE REGRAS

No tocante ao conhecimento representado por um modelo de rede neural, observa-se que os pesos das conexões codificam o conhecimento adquirido pela rede neural, enquanto que as unidades ocultas são denominadas de detectoras de características. Neste sentido, a aquisição de conhecimento de modelos de redes neurais supervisionadas normalmente envolve a utilização de algoritmos baseados nos valores dos pesos das conexões ou nos valores das ativações das unidades ocultas. Estes algoritmos normalmente são denominados de Algoritmo de Extração de Regras de Redes Neurais. Existem vários algoritmos de extração de regras. Alguns deles são: algoritmo SUBSET, algoritmo Mofn, algoritmo LOGIC, algoritmo PERCEP, algoritmo KT, algoritmo TREPAN e algoritmo RX. Nos demais subitens os algoritmos são comentados resumidamente.

3.2.1 ALGORITMO SUBSET

A metodologia se baseia na análise dos pesos, ou seja, quando a soma ponderada destes é maior do que o valor da tendência – bias – da unidade observa-se que o valor de ativação da unidade está próximo da unidade. Caso contrário, está próximo de zero. Como as ativações das unidades pertencem ao conjunto $\{0,1\}$, observa-se que a busca pelas regras é simplificada, pois as conexões possuem sinais iguais aos dos seus respectivos pesos, ou nenhum sinal.

3.2.2 ALGORITMO Mofn

Este algoritmo é um variante do algoritmo SUBSET, diferenciando-se deste por apresentar regras do tipo:

Se (M dos seguintes N antecedentes são verdadeiros)

Então Conclusão.

Basicamente, o algoritmo Mofn obtém classes de equivalência para os pesos – agrupados por meio de um algoritmo de clustering intitulado de Join Algorithm [8]. Os pesos pertencentes ao agrupamento são então substituídos pela média do agrupamento. Os pesos, cuja influência no cálculo dos conseqüentes é desprezível [9], são eliminados, seguindo-se um processo de otimização – onde os valores dos pesos são fixos – dos valores da tendência – bias – de cada unidade.

3.2.3 ALGORITMO LOGIC

Em 1997, OLIVEIRA et al. [10] desenvolveram um algoritmo para realizar a extração de regras de redes neurais não recorrentes baseado na mesma premissa básica utilizada no algoritmo SUBSET. Neste sentido o algoritmo LOGIC procura determinar subconjuntos de pesos cuja soma ponderada resulta em condição suficiente para a ativação de determinada unidade.

3.2.4 ALGORITMO PERCEP

O algoritmo PERCEP, considera a importância relativa de cada peso para a ativação da unidade, contrariamente ao algoritmo LOGIC, que considera todas as entradas com o mesmo grau de importância.

3.2.5 ALGORITMO KT

As regras geradas pelo método possuem a seguinte forma:

Se (premissas) então (conclusão), ou seja,

Se $A^{1+}, \dots, A_i+, \dots, -A^{1-}, \dots, -A_j-$ então C ou $-C$.

onde A+ são os atributos positivos e os A- são os atributos negativos e C é a conclusão ou conceito.

Basicamente, o algoritmo procura regras heurísticamente, diferenciando os atributos em positivos e negativos.

3.2.6 ALGORITMO TREPAN

O algoritmo se utiliza de questionamentos à rede neural, que assume a função do oráculo, para induzir uma árvore de decisão [11] que aproxima o conceito representado pela rede neural. A função do oráculo é a determinação da classe à qual pertence determinado exemplo do conjunto de treinamento, apresentado por meio de um questionamento. Este pode ser realizado por meio de restrições impostas aos valores de determinado atributo, mantendo-se indefinidos os valores de outros atributos, de modo que se possa verificar a influência de determinado atributo na função representada pela rede neural. Nestes casos, geram-se exemplos completos por meio da seleção randômica dos valores de cada atributo, respeitando-se as restrições pré-estabelecidas. A geração randômica é baseada na modelagem das distribuições marginais de cada atributo. O oráculo – rede neural – é utilizado para:

- Determinar as classes referentes aos exemplos de treinamento;
- Selecionar critérios de divisão – *splits* – para cada nó interno da árvore de classificação;
- Verificar a abrangência de cada nó, ou seja, as classes descritas por cada nó.

O algoritmo se utiliza de expressões booleanas de m of n, ou seja, especifica-se um limiar inteiro m e um conjunto de n condições booleanas, para realizar as divisões. A seleção dos critérios de divisão é realizada por meio do critério da taxa de ganho – *gain ratio criterion* – que avalia as divisões candidatas.

A principal vantagem apresentada pelo algoritmo TREPAN diz respeito à generalidade, não requerendo arquiteturas e/ou algoritmos de treinamento específicos nem tampouco que o modelo seja uma rede neural [6].

3.2.7 ALGORITMO RX

Primeiramente, realiza-se o treinamento da rede neural, de modo que se obtenha a taxa de classificação correta desejada [3]. Eliminam-se, então as conexões redundantes, seguindo-se uma análise dos valores das ativações para a obtenção de regras baseadas nestes valores. Basicamente, pode-se resumir o algoritmo de extração de regras, denominado de RX, nos seguintes passos:

- 1) Processar os eventos – pertencentes ao conjunto de treinamento – através da rede computando-se os valores de ativação das unidades;
- 2) Aplicar um algoritmo de agrupamento – *clustering* – aos valores de ativação das unidades ocultas;
- 3) Enumerar os valores de ativação discretizados e computar os respectivos valores de saída da rede. Gerar regras que descrevam as saídas da rede em termos dos valores de ativação discretizados em 2);
- 4) Para cada unidade oculta, enumerar os valores de entrada que conduzem aos respectivos valores de ativação das unidades ocultas, gerando regras que descrevam os valores discretizados das unidades ocultas em termos das entradas da rede neural;
- 5) Combinar as regras obtidas em 3) e 4) para obter as regras que relacionam as variáveis dependentes às variáveis independentes.

O primeiro passo do algoritmo RX agrupa os valores de ativação das unidades ocultas sem o comprometimento da precisão da rede neural. Após a execução do algoritmo, obtém-se um conjunto de valores de ativação para cada unidade oculta. O

algoritmo de agrupamento – *clustering* – é aplicado em cada unidade oculta separadamente. Primeiramente, classificam-se os valores de ativação em ordem crescente. Segue-se, então, a procura pelos intervalos correspondentes às ativações obtidas para cada classe, de maneira que os agrupamentos não gerem informações redundantes, isto é, intervalos de ativações iguais gerando classes diferentes.

Considerando-se esta metodologia, obtém-se regras da seguinte forma:

Se $(a_1 \theta v_1)$ e $(a_2 \theta v_2)$ e $(a_3 \theta v_3)$ e ... e $(a_n \theta v_n)$ então C;

onde:

a_i = atributos

θ = { = , ≤ , ≥ , < , > , ≠ }

v_i = constantes

C = classes.

3.3 ALGORITMO DE EXTRAÇÃO DE REGRAS IMPLEMENTADO

No que concerne às metodologias utilizadas para a extração de regras dos modelos de redes neurais, cumpre citar a ocorrência de duas tarefas distintas:

- o treinamento da rede neural com o objetivo de se minimizar o erro de classificação;
- a utilização do algoritmo de extração de regras propriamente dito.

O algoritmo utilizado neste trabalho difere do Algoritmo RX no sentido de que a extração de regras é realizada para cada classe separadamente. Para tal, implementou-se um simples algoritmo de clustering, o qual separa o conjunto de treinamento por classes. *Clustering* é o agrupamento de objetos semelhantes.

Para um melhor entendimento do algoritmo de extração de regras implementado neste trabalho, intitulado de algoritmo RX MODIFICADO utiliza-se um exemplo ilustrativo:

Baseando-se no modelo de rede neural abaixo:

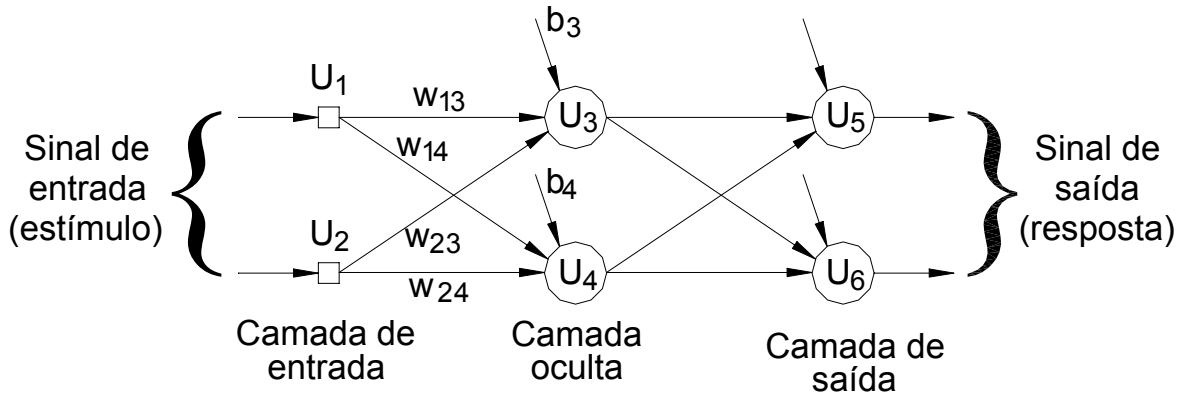


Figura 3.1: Modelo de Rede Neural

onde:

w_{ij} = pesos ótimos obtidos;

b_i = bias obtidos;

$a_3 = w_{13} \cdot U_1 + w_{23} \cdot U_2 + b_3 \Rightarrow$ ativações para a unidade oculta 3;

$a_4 = w_{14} \cdot U_1 + w_{24} \cdot U_2 + b_4 \Rightarrow$ ativações para a unidade oculta 4;

U_1 = unidade de entrada 1;

U_2 = unidade de entrada 2;

$U_3 = \tanh(a_3) \Rightarrow$ saídas para a unidade oculta 3;

$U_4 = \tanh(a_4) \Rightarrow$ saídas para a unidade oculta 4.

O Algoritmo RX MODIFICADO se baseia nos valores das ativações das unidades ocultas. Realizado o treinamento da rede neural, agrupam-se os exemplos de treinamento por classes, a fim de que se possa extrair regras para cada classe individualmente. Feito isto, computa-se os valores das ativações para cada exemplo

pertencente à determinada classe. Assim sendo, obtém-se um conjunto de valores de ativações para cada classe e para cada unidade oculta. A título de ilustração, consideremos os seguintes valores de ativações pertencentes à classe 1 para a unidade oculta 3 (U_3):

$$C_1 = \{ c_{11}, c_{12}, c_{13}, c_{14}, c_{15}, c_{16}, c_{17}, c_{18}, c_{19} \}$$

Para este conjunto de ativações, o algoritmo seguiria os seguintes passos:

- 1) Calcule a média e o desvio padrão do conjunto de valores C_1 ;
- 2) Agrupe os valores c_{ij} cuja Dc_{ij} seja menor ou igual ao desvio padrão multiplicado pelo fator de tolerância MULTI, onde Dc_{ij} é a Distância Euclidiana entre o valor c_{ij} e a média dos valores pertencentes a C_1 ;
- 3) Retire, do conjunto de valores C_1 , os valores agrupados no *cluster*;
- 4) Se $C_1 = \{\phi\}$ então pare. Se não vá para o passo 1).

Supondo-se, então, que o interesse seja agrupar valores distanciados no máximo em duas unidades de desvio padrão em relação à média, o fator de tolerância MULTI seria igual a 2. Se o interesse foi somente no primeiro *cluster* obtido, o algoritmo segue os seguintes passos:

- 1) Calculou a média e o desvio padrão para o conjunto de valores C_1 ;
- 2) Agrupou todos os valores cujas distâncias em relação à média são inferiores a duas unidades de desvio padrão, desconsiderando os demais valores. A distância euclidiana é igual ao módulo da diferença entre dois números.

Obtem-se, para a unidade U_3 , o seguinte *cluster* :

$$\text{Cluster 1} = \{ c_{11}, c_{12}, c_{14}, c_{16}, c_{17}, c_{18} \}.$$

Estes valores representam o conhecimento adquirido pela rede neural, na unidade U_3 , para a classe 1. Para a extração de regras, utilizaremos os valores extremos – máximos e mínimos – deste conjunto. Estes valores são representados pelos seguintes valores:

$$\text{Valor máximo} = c_{12}$$

$$\text{Valor mínimo} = c_{18}$$

Seguindo-se a mesma metodologia, obtem-se os valores máximo e mínimo para a unidade U_4 , por exemplo:

$$\text{Valor máximo} = c_{25}$$

$$\text{Valor mínimo} = c_{29}$$

A partir destes valores, resultaria, por exemplo, a seguinte regra:

SE $\{ c_{18} \leq a_3 \leq c_{12} \}$ e $\{ c_{29} \leq a_4 \leq c_{25} \}$ ENTÃO Classe 1.

CAPÍTULO 4

IMPLEMENTAÇÃO NO MATLAB

4.1 INTRODUÇÃO

O objetivo principal é usar o aplicativo Matlab de uma forma completa tanto com o seu módulo de redes neurais, que serve para treinar a rede, iniciado pelo comando `ntool`, como também usar a sua linguagem de programação para desenvolver um programa de extração de regras de redes neurais, iniciado pelo comando `rxmod`. Com isso, existe uma plena otimização e um desempenho bastante confortável para o futuro usuário. A seguir se ilustra todo o procedimento necessário para obter a extração de regras da forma mais otimizada.

4.2 PROGRAMA IMPLEMENTADO

Com o fluxograma do programa pode-se obter os passos do seu desenvolvimento como também a compreensão do programa, com o respectivo algoritmo RX MODIFICADO. A ilustração do fluxograma do programa `rxmod` se encontra na Figura 4.1.

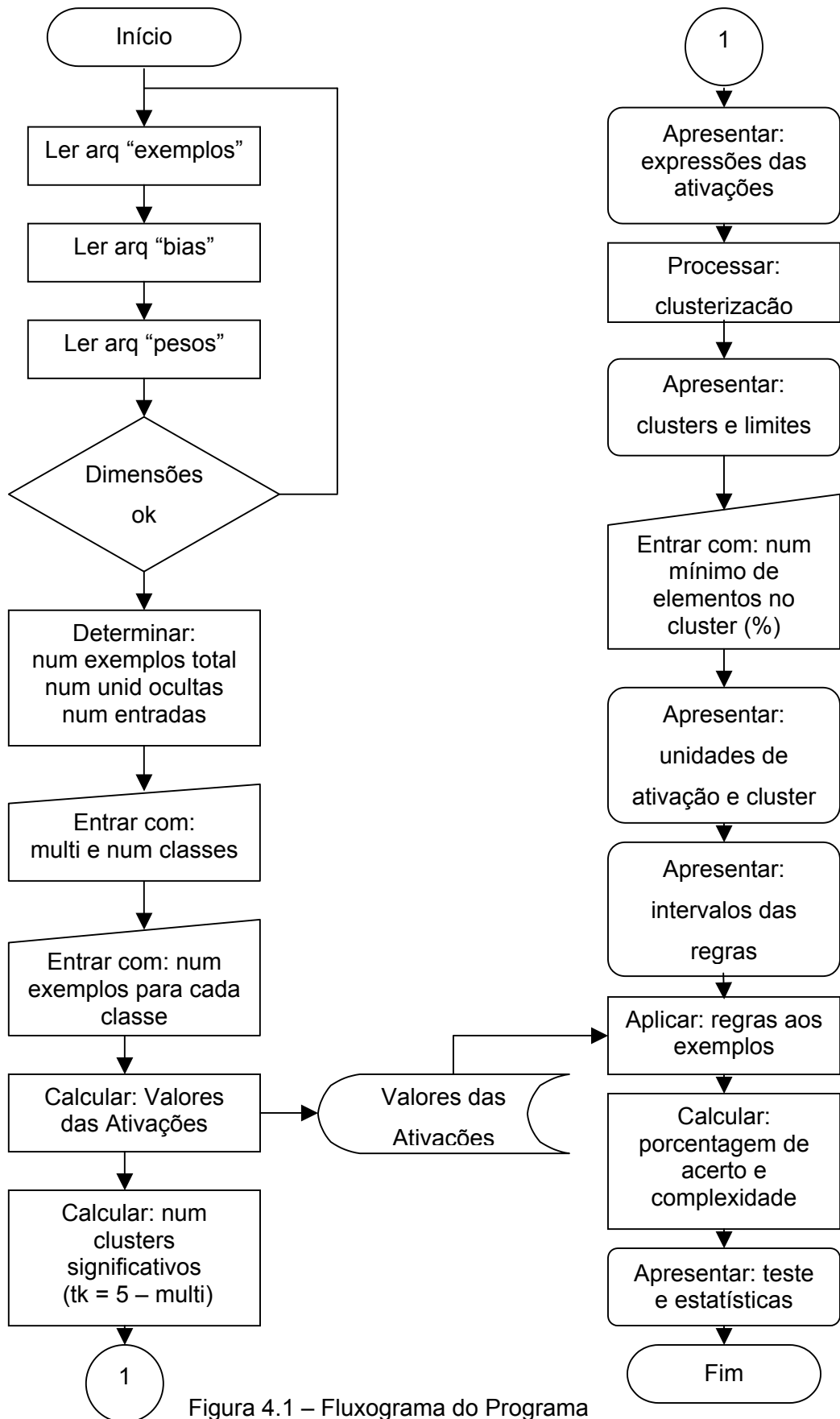


Figura 4.1 – Fluxograma do Programa rxmod

Convém citar que o programa desenvolvido neste trabalho considera os *clusters* significativos:

$TK = (5 - MULTI)$ onde:

TK = número de *clusters* significativos;

MULTI = fator de multiplicação do desvio padrão {1,2,3,4}

O programa define a complexidade do conjunto de regras segundo a seguinte fórmula [13]:

$C = 0,6 \cdot NR + 0,4 \cdot NP$ onde:

C = Complexidade do conjunto de regras;

NR = Número de regras;

NP = Número de premissas do conjunto.

O número de premissas do conjunto é igual à média do número de premissas.

A listagem do código do programa encontra-se no Apêndice I.

4.3 FUNCIONAMENTO OTIMIZADO NO MATLAB

Seguem-se os passos necessários para a obtenção das regras da rede neural no Matlab:

1. Inicialmente, estuda-se o problema proposto para se treinar na rede neural;
2. Separa-se em arquivos(do excel) as entradas - exemplos e o objetivo - *target* da mesma;
3. O arquivo de entrada - exemplos deve ser organizado da seguinte forma: as linhas correspondem ao número de entradas e as colunas ao número de exemplos;

4. O arquivo do Objetivo – *target* deve ser organizado da seguinte forma: a linha corresponde ao número de neurônios da saída e as colunas aos objetivos dos exemplos;
5. Tendo os dois arquivos organizados como segue, inicia-se o programa Matlab e conseqüentemente faz-se a importação dos arquivos para linguagem do programa mestre, conforme ilustra a Figura 4.2. Caso os arquivos estejam organizados de forma trocada entre linhas e colunas é só usar a matriz transposta no Matlab para fazer a devida inversão das linhas por coluna ou vice-versa;
6. Inicia-se o módulo de redes neurais, com o comando *ntool*, conforme ilustra a Figura 4.3, conseqüentemente importa-se para o mesmo, nos respectivos lugares os arquivos de entrada - exemplos e do objetivo – *target*, com o ícone *import*;
7. Cria-se uma nova rede clicando-se no ícone *new network*, conforme ilustra a Figura 4.3, e definem-se os parâmetros da mesma, como: nome da rede, tipo do algoritmo, mínimo e máximo de cada linha da entrada, função de treinamento, função de adaptação da aprendizagem, tipo de erro, número de camadas, camada aparente, número de neurônios nas respectivas camadas e funções de ativação nos neurônios das respectivas camadas, conforme ilustra Figura 4.4. O número de neurônios da camada de saída deve conter o mesmo número de linhas do arquivo *target*;
8. Depois de ter definido e criado a respectiva rede, treina-se a mesma com as respectivas entradas e os objetivos, conforme ilustra a Figura 4.5 e 4.6;
9. Simula-se, usando o ícone *Simulate*, entradas com um arquivo teste ou com o ícone *New Data* para saber a resposta da rede treinada, ou seja, saber se a rede está respondendo bem, as respectivas

entradas, o ícone view do gerenciador de dados serve para visualizar os respectivos dados do gerenciador, conforme ilustra a Figura 4.7 e 4.8, a visualização é limitada na tela quando não se consegue um ótimo desempenho;

10. Depois de ter a confirmação que a rede treinou, exporta-se a rede, conforme ilustra a Figura 4.9 e 4.10, pois nela contém tanto os valores dos pesos das conexões da 1ª camada, como também os valores de bias da 1ª camada. A rede é automaticamente colocada no espaço de trabalho – *Workspace*, conforme ilustra a Figura 4.11;
11. Digita-se na janela de comando – *Command Window* → Rede.iw{1,1}, automaticamente é criado o arquivo “ans” na área de trabalho – *Workspace*, com isso na janela de comando digita-se → pesos=ans, salva-se o arquivo pesos para um arquivo “pesos.mat”, este procedimento é a exportação da matriz peso da 1ª camada da respectiva rede neural;
12. Digita-se na janela de comando – *Command Window* → Rede.b{1}, automaticamente é criado o arquivo “ans” na área de trabalho – *Workspace*, com isso na janela de comando digita-se → bias=ans, salva-se o arquivo bias para um arquivo “bias.mat”, este procedimento é a exportação da matriz bias da 1ª camada da respectiva rede neural;
13. Com esses valores em arquivos, que são necessários para a extração de regras da respectiva rede neural, pode-se iniciar, conforme ilustra a Figura 4.12, o programa implementado para extração de regras de redes neurais. O comando utilizado na janela de comando – *Command Windows* é rxmod, já que o arquivo do programa se encontra localizado na pasta *work* do Matlab;

14. O programa de extração de regras de redes neurais pede três arquivos que são: arquivos com valores de exemplos, arquivos com valores de bias da 1ª camada e arquivos com pesos das conexões da 1ª camada, conforme ilustra Figura 4.13;
15. Posteriormente é feito pelo programa duas perguntas: número do fator multiplicador do desvio padrão e número de classes, conforme ilustra Figura 4.14;
16. O programa pergunta quantos exemplos por cada classe, o número das classes foi definido no passo anterior. Veja a Figura 4.15;
17. O programa gera na janela de comando – *Command Windows*, as expressões das ativações e os clusters com seus respectivos valores máximos e mínimos;
18. O programa pergunta qual o número mínimo de elementos no cluster, isto em porcentagem. Como ilustra a Figura 4.16;
19. Por fim o programa mostra as unidades de ativação e *clusters* significativos, apresenta os intervalos das regras, teste das regras, apresenta o percentual de acerto do conjunto de regras e a complexidade do conjunto de regras.

As figuras 4.2 a 4.16 ilustram as telas correspondentes a implementação efetuada.

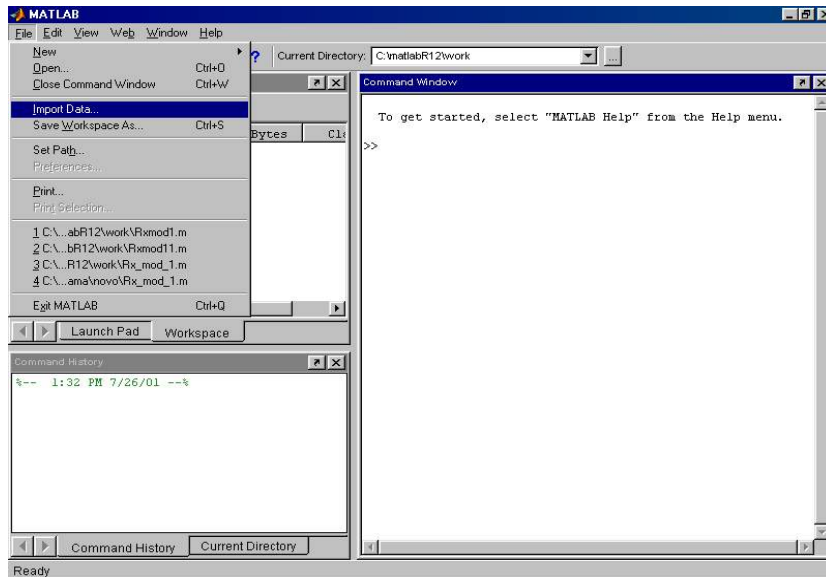


Figura 4.2– Importação dos arquivos de entrada e objetivo

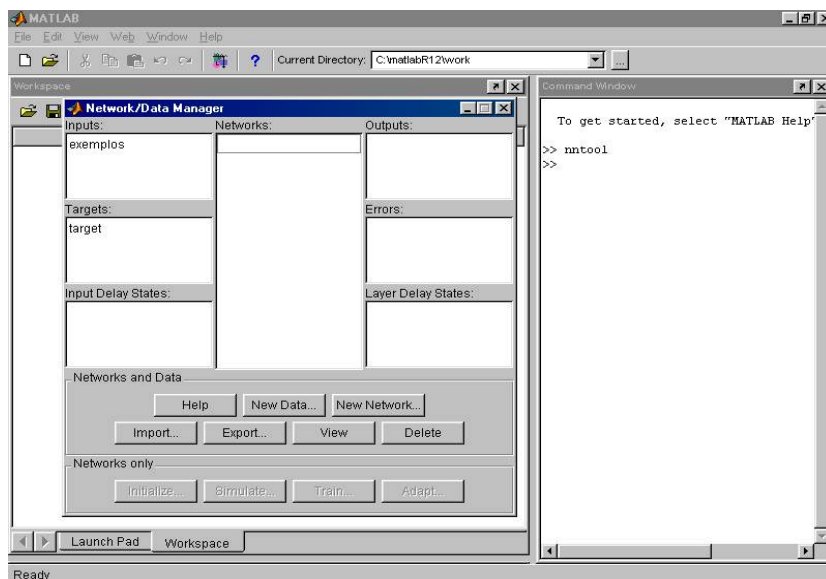


Figura 4.3– Início do módulo de redes neurais utilizando o comando nntool, posteriormente utilizando o ícone *import* para importar os arquivos exemplos e target e finalmente seleciona-se o ícone *New Network* para criar um nova rede

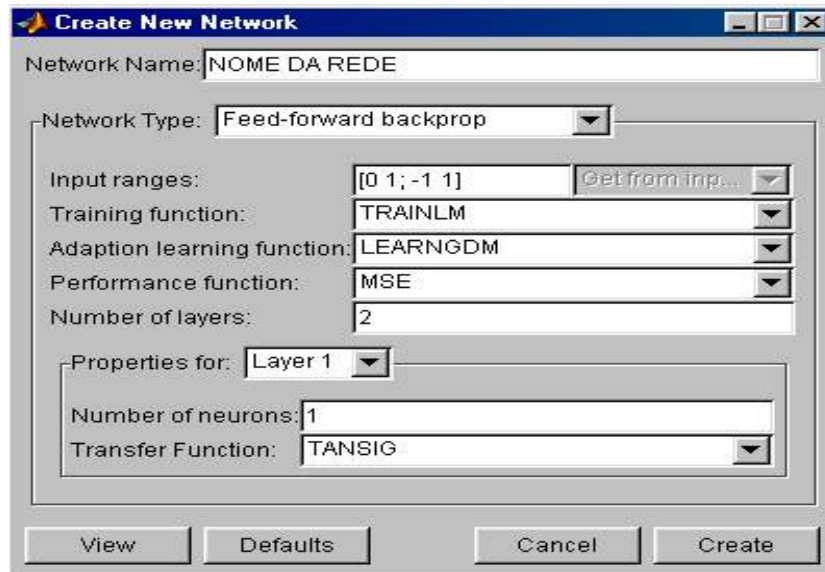


Figura 4.4 – Criando uma nova rede; após definir os parâmetros da rede seleciona-se *create*

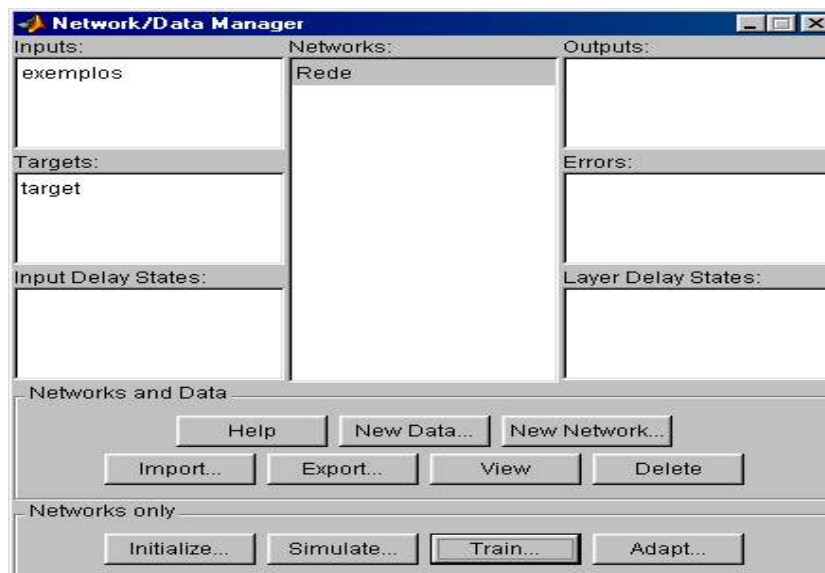


Figura 4.5 – Seleciona-se no nome da rede e posteriormente no ícone *Train...*



Figura 4.6 – Inserir as entradas e objetivos e posteriormente selecionar-se o ícone Train Network, para treinar a rede

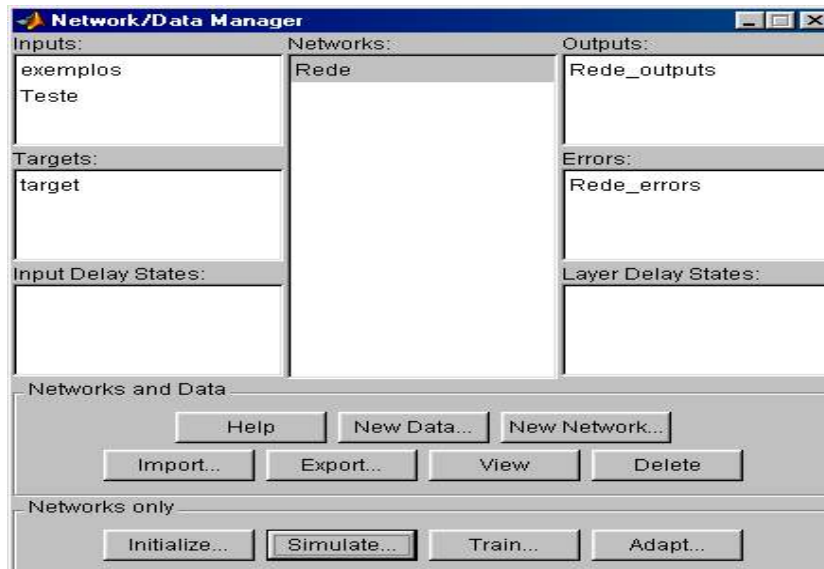


Figura 4.7 – Teste da rede, selecionar-se o ícone Simulate...

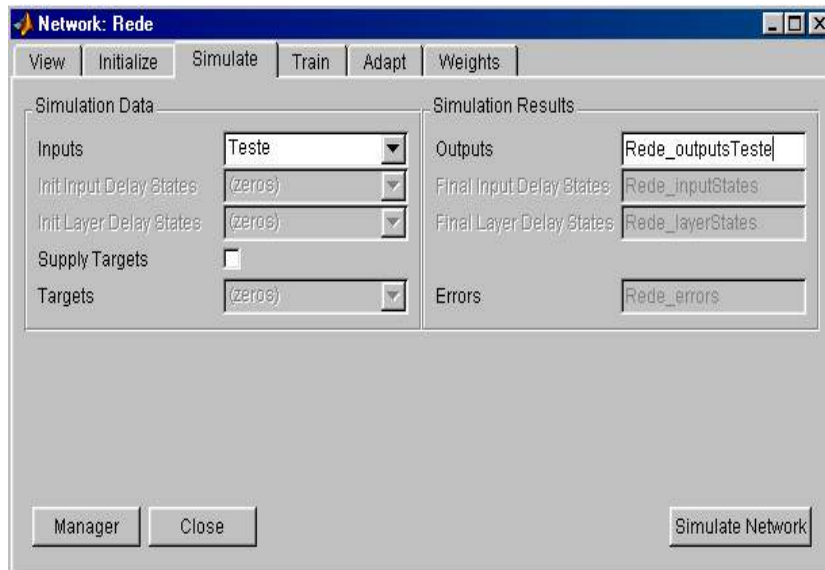


Figura 4.8 – Inserir a entrada teste e definir a Saída teste, posteriormente seleciona-se o ícone Simulate Network

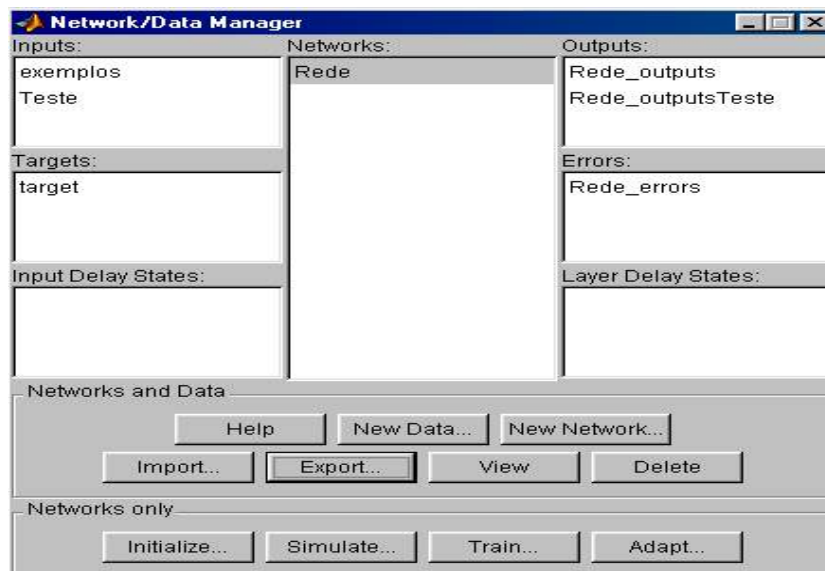


Figura 4.9 – Exportar a rede, seleciona-se o ícone Export...

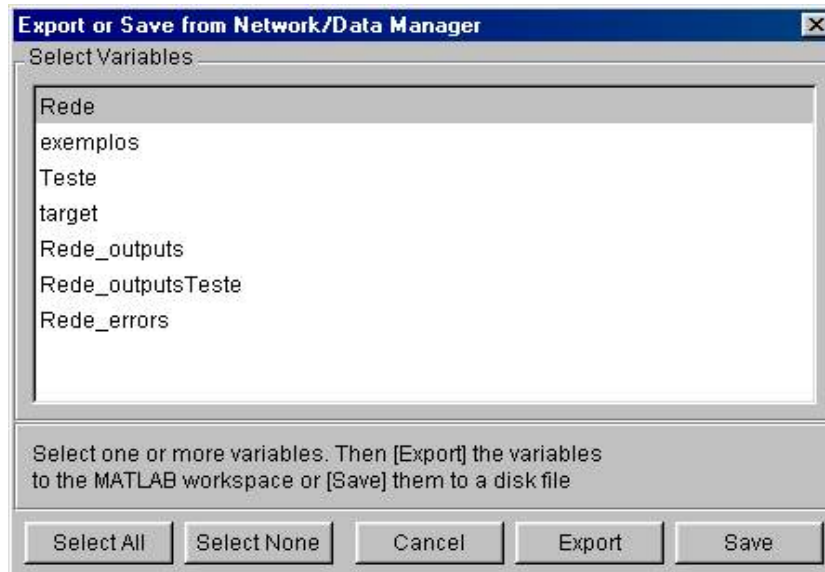


Figura 4.10 – Selecionar a Rede e posteriormente seleciona-se Export

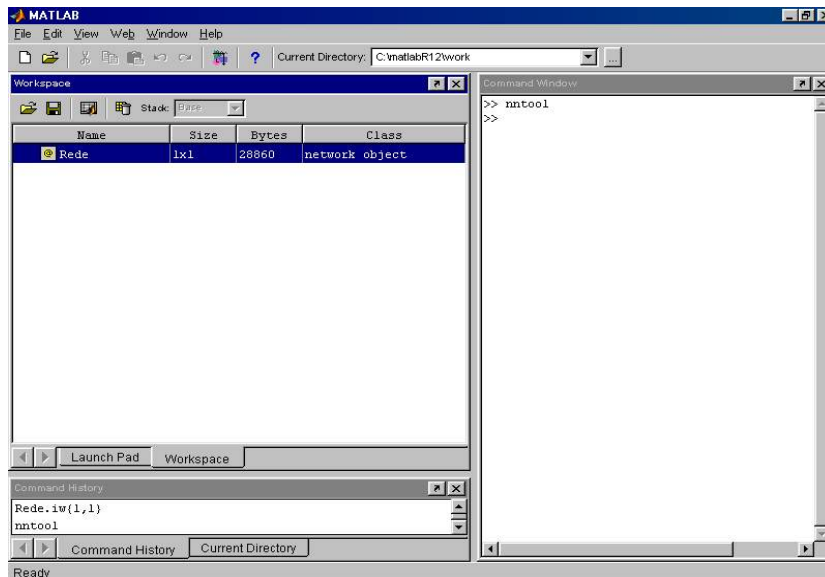


Figura 4.11 – Rede no espaço de trabalho – *Workspace*.

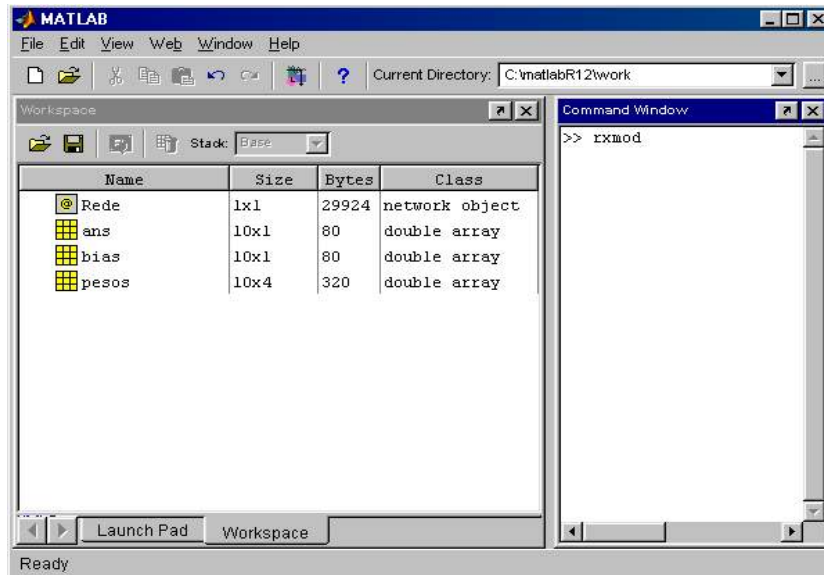


Figura 4.12 – Início do programa de extração de regras de redes neurais, com o comando `rxmod`



Figura 4.13 – Solicitação de arquivos para o programa de extração de regras, `rxmod`

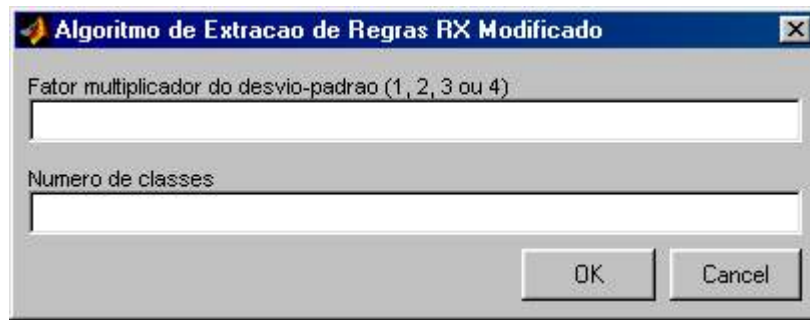


Figura 4.14 – Perguntas do Programa de Extração de Regras

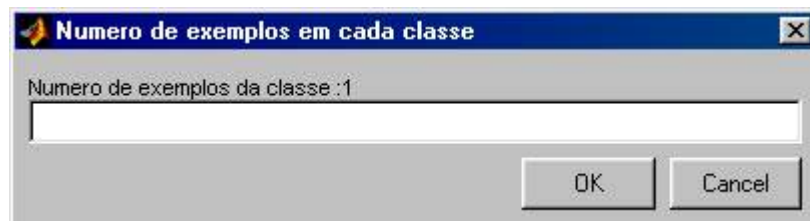


Figura 4.15 – Pergunta sobre o número de exemplos de cada classe

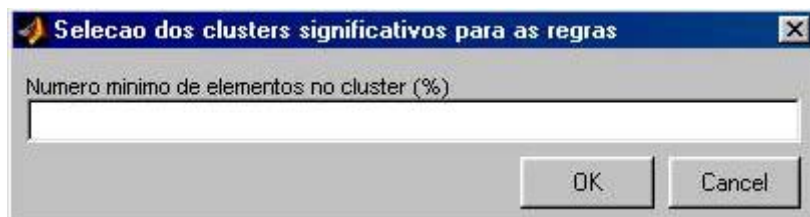


Figura 4.16 – Pergunta sobre o número mínimo de elementos no cluster, isto em porcentagem

CAPÍTULO 5

ESTUDO DOS CASOS

5.1 EXEMPLOS

Utilizou-se, como primeiro exemplo, o banco de dados conhecido como Iris Plants Database, de autoria de R.A. Fisher. Os dados contêm três classes de plantas intituladas de Iris Setosa, Iris Versicolour e Iris Virginica. Cada classe engloba 50 exemplos. A primeira classe é linearmente separável das demais, enquanto que as classes Versicolour e Virginica não são linearmente separáveis entre si. Os quatro atributos envolvidos são: comprimento e largura de sépalas e de pétalas.

Para o segundo exemplo, utilizou-se o Thyroid Gland Data. O doador dos dados é a universidade de *James Cook* na Austrália. O dado contém três classes referentes ao funcionamento da tireóide que são: normal, hipertireoidismo e hipotireoidismo. A primeira classe contém 150 exemplos, a segunda 35 exemplos e a terceira 30 exemplos; respectivamente. Os cinco atributos envolvidos são: teste de elevação de T3 em percentual, a dosagem total de tiroxina no soro, a dosagem total de T3 no soro, o TSH (hormônio-estimulante da tireóide) e a maior diferença absoluta de TSH.

O terceiro e último exemplo é o Wine Recognition Data. O doador dos dados é o *Institute of Pharmaceutical and Food Analysis and Technologies* na Itália. Os dados contêm três classes referentes a qual cultivo o vinho é derivado que são: cultivo 1, cultivo 2 e cultivo 3. A primeira classe contém 59 exemplos, a segunda 71 exemplos e a terceira 48 exemplos; respectivamente. Os treze atributos envolvidos são relativos a quantidade de constituintes no vinho.

Para todos os exemplos considerou-se que o conjunto de dados representa a população para o domínio em questão, optou-se pelo treinamento da rede em todo o conjunto de dados o qual se constitui, pois, no conjunto de treinamento. Assim sendo,

não se utilizou os conjuntos de teste e de validação, nem tampouco se considerou os problemas provenientes de *overtraining*, memorização e generalização.

5.1.1 RESULTADOS

Para o primeiro exemplo, adotou-se a seguinte arquitetura da rede neural, com três camadas; onde a camada de entrada possui 4 sinais de entradas, a camada oculta possui 10 neurônios e a camada de saída possui 3 neurônios. Em se tratando das funções de ativação, utiliza-se da função tanh para a camada oculta e da função linear para a camada de saída, conforme ilustra a Figura 5.1. Utiliza-se da rede tipo: retropropagação, utilizando para a função de treinamento: *Levenberg-Marquardt*, função de adaptação da aprendizagem: o gradiente descendente, para o tipo de erro: erro médio quadrático; conforme ilustra a Figura 5.2.

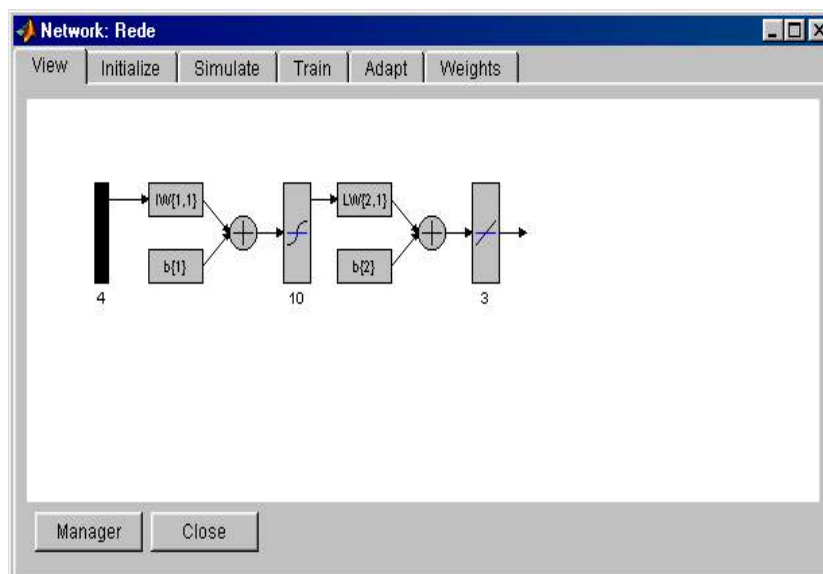


Figura 5.1 – Arquitetura e tipos de funções da rede neural para o exemplo Iris

Plants Database

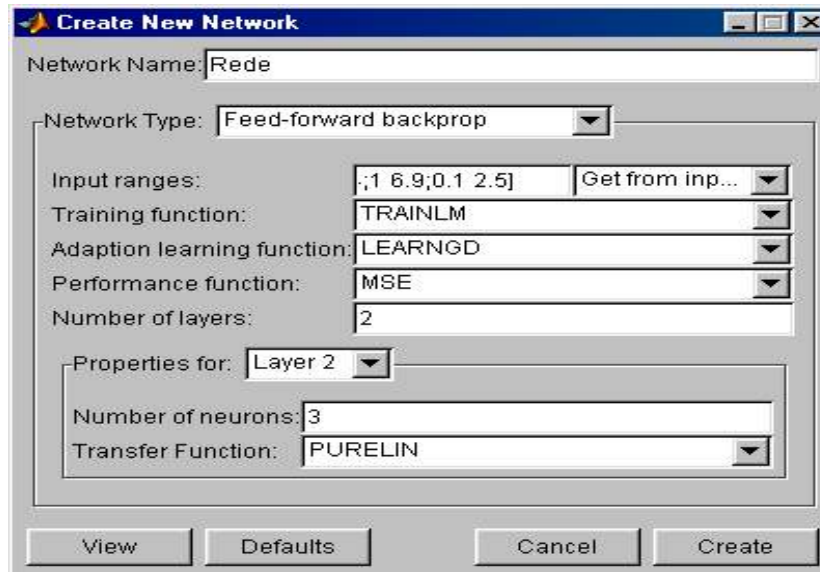


Figura 5.2 – Escolha das variáveis para o determinado tipo de rede neural

Com relação aos parâmetros de treinamento, utilizou-se 1000 épocas, como ilustra a Figura 5.3. O objetivo principal é obter um desempenho muito próximo da meta que é 0, ou seja, um número menor e igual a uma exponencial -12 , como ilustra Figura 4.4. Alcançando-se este objetivo, garantimos as saídas inteiras e idênticas aos objetivos; como também um aprendizado ótimo da respectiva rede neural.

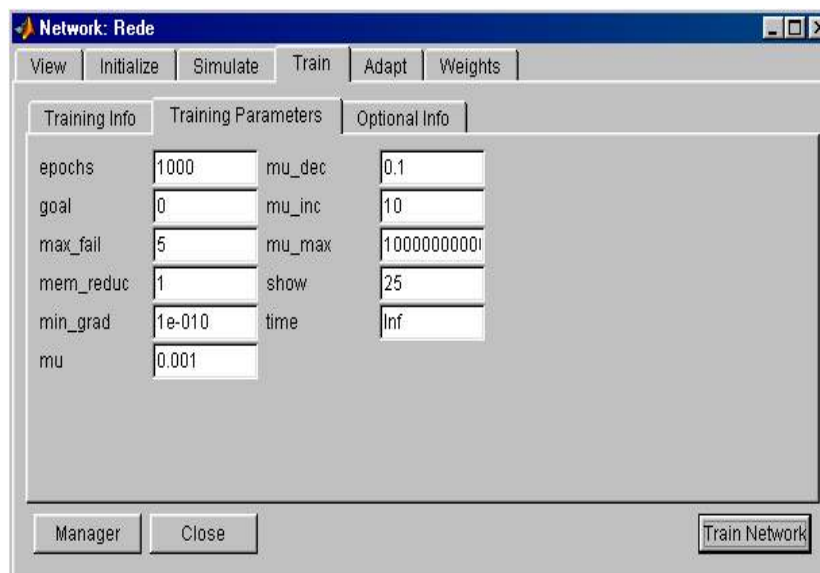


Figura 5.3 – Parâmetros de treinamento utilizado

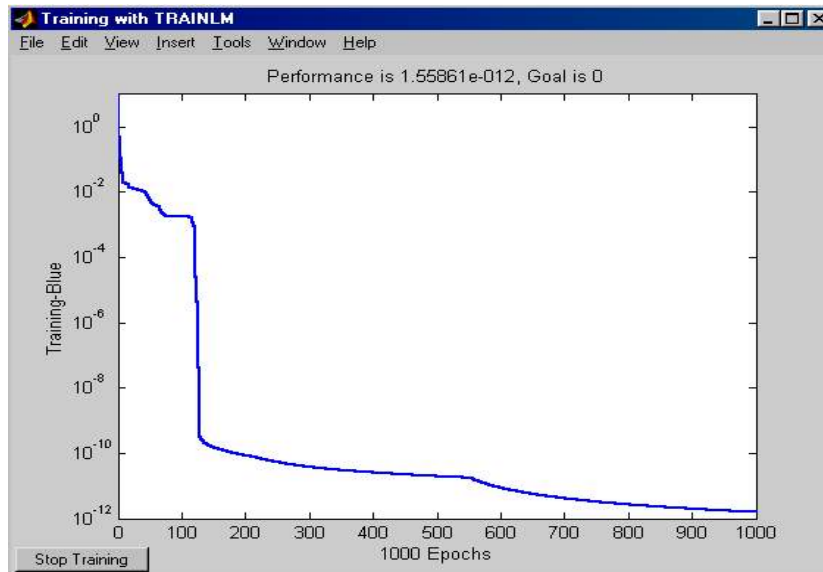


Figura 5.4 – Desempenho ótimo

Pode-se obter um desempenho ótimo, com poucas épocas, desde que no início do treinamento, o mesmo inicie perto do mínimo global e não fique preso num mínimo local por muito tempo, ou seja, cada treinamento é um caso mesmo obedecendo aos mesmos parâmetros.

Após o treinamento da rede neural, testa-se a mesma e posteriormente exporta-se a rede para se obter os arquivos de pesos e bias. Então, inicia-se o programa rxmod, define-se os arquivos para o mesmo de acordo com o exemplo, posteriormente responde: o fator multiplicador do desvio padrão e número de classes; as respostas são respectivamente: 2 e 3. Conforme ilustra a Figura 5.5.

O fator multiplicador do desvio padrão tem por objetivo, determinar o número de *clusters* significativos, ou seja, quanto maior o número do fator multiplicador do desvio padrão menor será o número de *clusters* significativos; com isso proporciona uma regra menos precisa e conseqüentemente menos complexa.

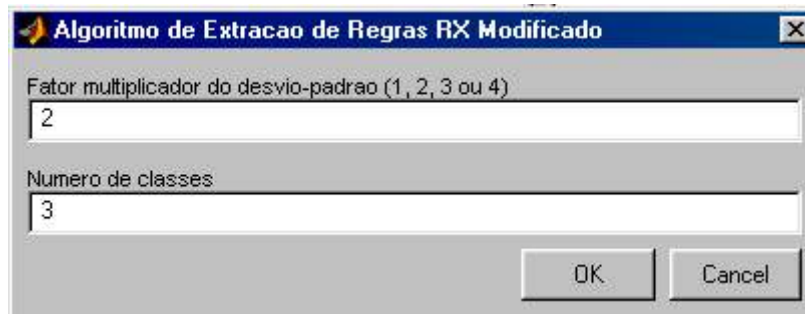


Figura 5.5 – Perguntas do programa de extração de regras

Posteriormente o programa pergunta quantos exemplos por classe, como ilustra a Figura 5.6.

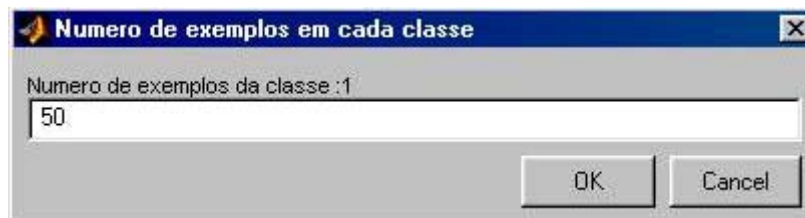


Figura 5.6 – Definição do número de exemplos por classe

Neste exemplo as três classes contêm cada uma 50 exemplos, totalizando 150 exemplos. Depois da confirmação destes dados, o programa gera automaticamente: as expressões das ativações e os *clusters* com seus respectivos valores máximos e mínimos. Após este processamento, pergunta-se o número mínimo de elementos no *cluster*, pois ele é responsável por uma regra mais ou menos refinada, desde que o número do fator multiplicador do desvio padrão não seja elevado. O número de premissas será maior se for considerado um percentual baixo para o número de elementos no *cluster* e vice-versa. O percentual considerado para este exemplo foi de 10%, como ilustra a Figura 5.7.

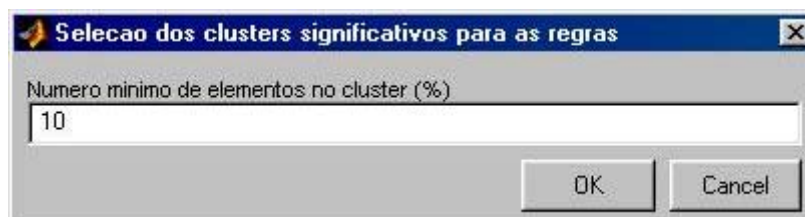


Figura 5.7 – Percentual de elementos no cluster

Após ter confirmado este ultimo dado, automaticamente o programa gera a saída final que são: as regras, teste das regras, o percentual de acerto do conjunto de regras e a complexidade do conjunto de regras. Abaixo mostra-se em ordem: as expressões das ativações, as regras, teste das regras, o percentual de acerto do conjunto de regras e a complexidade do conjunto de regras. As regras são geradas de acordo com a ordem dos exemplos, ou seja, classe 1 é Iris Setosa, classe 2 é Iris Versicolour e classe 3 é Iris Virginica.

Expressões das Ativações:

$$a_1 = -0,096 - 24,891 \cdot x_1 + 12,431 \cdot x_2 - 5,885 \cdot x_3 + 91,048 \cdot x_4$$

$$a_2 = -7,126 + 1,696 \cdot x_1 - 4,185 \cdot x_2 - 0,748 \cdot x_3 + 4,678 \cdot x_4$$

$$a_3 = 9,337 + 1,777 \cdot x_1 + 4,891 \cdot x_2 - 4,257 \cdot x_3 - 8,253 \cdot x_4$$

$$a_4 = 46,739 + 12,133 \cdot x_1 + 12,956 \cdot x_2 - 22,087 \cdot x_3 - 31,271 \cdot x_4$$

$$a_5 = -36,154 + 9,078 \cdot x_1 - 12,121 \cdot x_2 + 2,767 \cdot x_3 - 6,563 \cdot x_4$$

$$a_6 = -15,906 + 9,000 \cdot x_1 - 14,555 \cdot x_2 - 0,418 \cdot x_3 + 9,654 \cdot x_4$$

$$a_7 = 1,718 + 23,908 \cdot x_1 - 12,185 \cdot x_2 + 6,295 \cdot x_3 - 89,902 \cdot x_4$$

$$a_8 = 91,834 + 44,973 \cdot x_1 + 45,513 \cdot x_2 - 76,775 \cdot x_3 - 74,517 \cdot x_4$$

$$a_9 = -15,949 + 9,005 \cdot x_1 - 14,564 \cdot x_2 - 0,419 \cdot x_3 + 9,661 \cdot x_4$$

$$a_{10} = 41,377 - 5,512 \cdot x_1 - 0,880 \cdot x_2 + 0,533 \cdot x_3 + 3,403 \cdot x_4$$

Regras para cada Classe

Regra para a classe: 1

$$Se(-84,04 \leq a_1 \leq -50,50)$$

$$e(-15,12 \leq a_2 \leq -11,18)$$

$$e(23,17 \leq a_3 \leq 31,30)$$

$$e(98,19 \leq a_4 \leq 127,26)$$

$$e(-35,15 \leq a_5 \leq -24,01)$$

$$e(-26,20 \leq a_6 \leq -13,31)$$

$$e(49,19 \leq a_7 \leq 82,11)$$

$$e(299,31 \leq a_8 \leq 389,68)$$

$$e(-26,24 \leq a_9 \leq -13,35)$$

$$e(8,25 \leq a_{10} \leq 16,26)$$

Então CLASSE 1

Regra para a classe: 2

$$Se(-43,98 \leq a_1 \leq 12,87)$$

$$e(-7,54 \leq a_2 \leq -4,10)$$

$$e(0,18 \leq a_3 \leq 8,77)$$

$$e(0,43 \leq a_4 \leq 33,12)$$

$$e(-20,94 \leq a_5 \leq -3,85)$$

$$e(1,53 \leq a_6 \leq 16,83)$$

$$e(-12,26 \leq a_7 \leq 43,39)$$

$$e(0,17 \leq a_8 \leq 102,41)$$

$$e(1,50 \leq a_9 \leq 16,81)$$

$$e(8,33 \leq a_{10} \leq 17,42)$$

Então CLASSE 2

Regra para a classe: 3

$$Se(-25,10 \leq a_1 \leq 78,84)$$

$$e(-5,30 \leq a_2 \leq -1,44)$$

$$e(-9,51 \leq a_3 \leq 0,13)$$

$$e(-34,31 \leq a_4 \leq -0,10)$$

$$e(-21,23 \leq a_5 \leq 1,50)$$

e($6,33 \leq a_6 \leq 26,34$)

e($-77,39 \leq a_7 \leq 25,27$)

e($-97,86 \leq a_8 \leq -0,73$)

e($6,30 \leq a_9 \leq 26,32$)

e($6,65 \leq a_{10} \leq 17,83$)

Então CLASSE 3

Teste das Regras

Para exemplos da CLASSE 1:

86% classificados como CLASSE 1;

0% classificados como CLASSE 2;

0% classificados como CLASSE 3.

Para exemplos da CLASSE 2:

0% classificados como CLASSE 1;

82% classificados como CLASSE 2;

2% classificados como CLASSE 3.

Para exemplos da CLASSE 3:

0% classificados como CLASSE 1;

0% classificados como CLASSE 2;

86% classificados como CLASSE 3.

Para o sistema de regras obtido a porcentagem de acerto do conjunto de regras foi de 84,67% e a complexidade do conjunto de regras 5,8.

Para o segundo exemplo, adotou-se a seguinte arquitetura da rede neural, com três camadas; onde a camada de entrada possui 5 sinais de entradas, a camada oculta possui 10 neurônios e a camada de saída possui 3 neurônios. Em se tratando das funções de ativação, utiliza-se da função tanh para a camada oculta e da função linear para a camada de saída. Utiliza-se da rede tipo: retropropagação, utilizando para a função de treinamento: *Levenberg-Marquardt*, função de adaptação da aprendizagem: o gradiente descendente, para o tipo de erro: erro médio quadrático. Com relação aos parâmetros de treinamento, utilizou-se 100 épocas.

Para a extração de regras os parâmetros do programa rxmod são:

Fator multiplicador do desvio-padrão = 2

Número de classes = 3

Exemplos classe 1 = 150

Exemplos classe 2 = 35

Exemplos classe 3 = 30

Percentual do número mínimo de elementos no cluster = 10%

Abaixo mostra-se em ordem: as expressões das ativações, as regras, teste das regras, o percentual de acerto do conjunto de regras e a complexidade do conjunto de regras. As regras são geradas de acordo com a ordem dos exemplos, ou seja, classe 1 é normal, classe 2 é hipertireoidismo e a classe 3 é hipotireoidismo.

Expressões das Ativações:

$$a_1 = -4,426 + 0,042 \cdot x_1 + 0,017 \cdot x_2 + 0,211 \cdot x_3 - 0,090 \cdot x_4 - 0,144 \cdot x_5$$

$$a_2 = 12,270 - 0,078 \cdot x_1 + 0,099 \cdot x_2 - 0,282 \cdot x_3 + 0,017 \cdot x_4 - 0,011 \cdot x_5$$

$$a_3 = -8,985 + 0,047 \cdot x_1 - 0,294 \cdot x_2 + 0,950 \cdot x_3 - 0,065 \cdot x_4 + 0,385 \cdot x_5$$

$$a_4 = -3,621 + 0,005 \cdot x_1 + 0,039 \cdot x_2 - 1,009 \cdot x_3 - 0,044 \cdot x_4 + 0,309 \cdot x_5$$

$$a_5 = 3,664 - 0,106 \cdot x_1 + 0,157 \cdot x_2 + 0,856 \cdot x_3 + 0,794 \cdot x_4 + 0,629 \cdot x_5$$

$$a_6 = -16,032 + 3,757 \cdot x_1 - 25,617 \cdot x_2 - 21,419 \cdot x_3 - 5,908 \cdot x_4 + 21,696 \cdot x_5$$

$$a_7 = -9,408 + 0,101 \cdot x_1 - 0,269 \cdot x_2 + 0,110 \cdot x_3 - 0,144 \cdot x_4 + 0,244 \cdot x_5$$

$$a_8 = -184,065 + 0,907 \cdot x_1 + 23,123 \cdot x_2 + 6,098 \cdot x_3 + 1,032 \cdot x_4 - 8,576 \cdot x_5$$

$$a_9 = 4,146 - 0,045 \cdot x_1 + 0,029 \cdot x_2 - 0,173 \cdot x_3 - 0,004 \cdot x_4 + 0,267 \cdot x_5$$

$$a_{10} = -0,724 + 0,037 \cdot x_1 - 0,232 \cdot x_2 - 0,203 \cdot x_3 + 0,279 \cdot x_4 - 0,078 \cdot x_5$$

Regras para cada Classe

Regra para a classe: 1

$$\text{Se } (-0,70 \leq a_1 \leq 1,09)$$

$$\text{e(} 2,83 \leq a_2 \leq 5,35)$$

$$\text{e(} -6,36 \leq a_3 \leq -1,89)$$

$$\text{e(} -5,00 \leq a_4 \leq -2,41)$$

$$\text{e(} -5,02 \leq a_5 \leq -0,03)$$

$$\text{e(} 22,66 \leq a_6 \leq 320,78)$$

$$\text{e(} -2,21 \leq a_7 \leq 2,14)$$

$$\text{e(} 16,34 \leq a_8 \leq 225,91)$$

$$\text{e(} -1,25 \leq a_9 \leq 0,82)$$

$$\text{e(} 0,04 \leq a_{10} \leq 2,00)$$

Então CLASSE 1

Regra para a classe: 2

$$\text{Se(} -0,23 \leq a_1 \leq 1,33)$$

$$\text{e(} 4,39 \leq a_2 \leq 8,11)$$

$$e(-9,19 \leq a_3 \leq -2,61)$$

$$e(-10,64 \leq a_4 \leq -3,89)$$

$$e(-4,49 \leq a_5 \leq 8,01)$$

$$e(-547,53 \leq a_6 \leq -6,08)$$

$$e(-8,46 \leq a_7 \leq 0,24)$$

$$e(170,63 \leq a_8 \leq 499,36)$$

$$e(-0,78 \leq a_9 \leq 1,02)$$

$$e(-5,04 \leq a_{10} \leq 0,12)$$

Então CLASSE 2

Regra para a classe: 3

$$Se(-7,59 \leq a_1 \leq 0,38)$$

$$e(1,26 \leq a_2 \leq 4,43)$$

$$e(-3,97 \leq a_3 \leq 11,39) \text{ ou } (14,95 \leq a_3 \leq 16,97)$$

$$e(-4,04 \leq a_4 \leq 8,40) \text{ ou } (10,20 \leq a_4 \leq 12,93)$$

$$e(-4,48 \leq a_5 \leq 38,76)$$

$$e(297,89 \leq a_6 \leq 1120,64) \text{ ou } (1293,13 \leq a_6 \leq 1419,56)$$

$$e(-0,32 \leq a_7 \leq 11,98)$$

$$e(-294,22 \leq a_8 \leq -5,87) \text{ ou } (-405,04 \leq a_8 \leq -391,23)$$

$$e(-1,34 \leq a_9 \leq 9,66) \text{ ou } (11,71 \leq a_9 \leq 13,61)$$

$$e(1,00 \leq a_{10} \leq 12,60)$$

Então CLASSE 3

Teste das Regras

Para exemplos da CLASSE 1 :

77,33% classificados como CLASSE 1;

0% classificados como CLASSE 2;

0% classificados como CLASSE 3.

Para exemplos da CLASSE 2 :

0% classificados como CLASSE 1;

82,85% classificados como CLASSE 2;

0% classificados como CLASSE 3.

Para exemplos da CLASSE 3 :

0% classificados como CLASSE 1;

0% classificados como CLASSE 2;

83,33% classificados como CLASSE 3.

Para o sistema de regras obtido a porcentagem de acerto do conjunto de regras foi de 79,07% e a complexidade do conjunto de regras 6,47.

Para o terceiro exemplo, adotou-se a seguinte arquitetura da rede neural, com três camadas; onde a camada de entrada possui 13 sinais de entradas, a camada oculta possui 10 neurônios e a camada de saída possui 3 neurônios. Em se tratando das funções de ativação, utiliza-se da função tanh para a camada oculta e da função linear para a camada de saída. Utiliza-se da rede tipo: retropropagação, utilizando para a função de treinamento: *Levenberg-Marquardt*, função de adaptação da aprendizagem: o gradiente descendente, para o tipo de erro: erro médio quadrático. Com relação aos parâmetros de treinamento, utiliza-se 100 épocas.

Para a extração de regras os parâmetros do programa rxmod são:

Fator multiplicador do desvio-padrão = 4

Número de classes = 3

Exemplos classe 1 = 59

Exemplos classe 2 = 71

Exemplos classe 3 = 48

Percentual do número mínimo de elementos no cluster = 0%

Abaixo mostra-se em ordem: as expressões das ativações, as regras, teste das regras, o percentual de acerto do conjunto de regras e a complexidade do conjunto de regras. As regras são geradas de acordo com a ordem dos exemplos, ou seja, classe 1 é cultivo 1, classe 2 é cultivo 2 e a classe 3 é cultivo 3.

Expressões das Ativações:

$$a_1 = 6,856 - 0,352 \cdot x_1 + 0,832 \cdot x_2 - 0,733 \cdot x_3 + 0,545 \cdot x_4 - 0,026 \cdot x_5 + 2,382 \cdot x_6 + 0,055 \cdot x_7 + 4,900 \cdot x_8 - 1,106 \cdot x_9 - 0,939 \cdot x_{10} + 1,152 \cdot x_{11} + 0,988 \cdot x_{12} - 0,022 \cdot x_{13}$$

$$a_2 = -2,919 + 1,449 \cdot x_1 + 1,295 \cdot x_2 - 2,553 \cdot x_3 + 0,877 \cdot x_4 - 0,109 \cdot x_5 + 4,139 \cdot x_6 - 2,242 \cdot x_7 - 3,694 \cdot x_8 - 0,589 \cdot x_9 - 1,796 \cdot x_{10} + 2,011 \cdot x_{11} + 1,254 \cdot x_{12} - 0,027 \cdot x_{13}$$

$$a_3 = 9,625 - 1,075 \cdot x_1 - 4,014 \cdot x_2 - 11,272 \cdot x_3 + 0,903 \cdot x_4 - 0,147 \cdot x_5 - 4,612 \cdot x_6 + 14,838 \cdot x_7 + 10,519 \cdot x_8 + 10,059 \cdot x_9 - 2,686 \cdot x_{10} + 27,481 \cdot x_{11} + 11,245 \cdot x_{12} - 0,029 \cdot x_{13}$$

$$a_4 = -1,056 - 0,560 \cdot x_1 + 0,438 \cdot x_2 + 0,188 \cdot x_3 - 0,089 \cdot x_4 - 0,123 \cdot x_5 - 0,033 \cdot x_6 + 0,414 \cdot x_7 + 1,153 \cdot x_8 - 0,892 \cdot x_9 + 0,869 \cdot x_{10} + 2,856 \cdot x_{11} + 1,321 \cdot x_{12} + 0,026 \cdot x_{13}$$

$$a_5 = 1,623 + 0,676 \cdot x_1 + 0,299 \cdot x_2 - 0,833 \cdot x_3 - 0,781 \cdot x_4 + 0,057 \cdot x_5 - 0,793 \cdot x_6 + 1,996 \cdot x_7 + 2,669 \cdot x_8 + 0,861 \cdot x_9 - 0,538 \cdot x_{10} - 1,827 \cdot x_{11} - 0,784 \cdot x_{12} + 0,002 \cdot x_{13}$$

$$a_6 = -1,751 + 0,628 \cdot x_1 - 0,136 \cdot x_2 - 0,371 \cdot x_3 + 0,326 \cdot x_4 + 0,049 \cdot x_5 - 0,059 \cdot x_6 - 0,695 \cdot x_7 - 1,611 \cdot x_8 + 0,086 \cdot x_9 - 0,031 \cdot x_{10} - 0,075 \cdot x_{11} - 0,597 \cdot x_{12} - 0,002 \cdot x_{13}$$

$$a_7 = -70,959 + 1,535 \cdot x_1 + 3,095 \cdot x_2 + 20,435 \cdot x_3 - 2,518 \cdot x_4 + 0,102 \cdot x_5 - 2,604 \cdot x_6 + 10,617 \cdot x_7 + 5,311 \cdot x_8 - 9,075 \cdot x_9 - 1,958 \cdot x_{10} - 17,452 \cdot x_{11} + 5,232 \cdot x_{12} + 0,050 \cdot x_{13}$$

$$a_8 = 7,463 + 0,586 \cdot x_1 + 2,235 \cdot x_2 - 1,379 \cdot x_3 - 0,332 \cdot x_4 - 0,106 \cdot x_5 + 0,494 \cdot x_6 - 2,331 \cdot x_7 - 9,072 \cdot x_8 + 7,147 \cdot x_9 + 3,254 \cdot x_{10} + 0,399 \cdot x_{11} - 2,340 \cdot x_{12} + 0,056 \cdot x_{13}$$

$$a_9 = -1,639 + 0,866 \cdot x_1 + 1,473 \cdot x_2 - 0,451 \cdot x_3 - 0,744 \cdot x_4 + 0,097 \cdot x_5 - 0,993 \cdot x_6 - 0,068 \cdot x_7 - 1,741 \cdot x_8 + 0,986 \cdot x_9 + 0,449 \cdot x_{10} + 0,275 \cdot x_{11} - 0,719 \cdot x_{12} - 0,008 \cdot x_{13}$$

$$a_{10} = 22,265 - 10,722 \cdot x_1 - 0,635 \cdot x_2 + 17,709 \cdot x_3 - 1,188 \cdot x_4 + 0,104 \cdot x_5 - 4,409 \cdot x_6 + 8,572 \cdot x_7 + 12,460 \cdot x_8 - 4,280 \cdot x_9 + 3,263 \cdot x_{10} + 6,682 \cdot x_{11} + 16,543 \cdot x_{12} - 0,012 \cdot x_{13}$$

Regras para cada Classe

Regra para a classe: 1

$$e(-25,26 \leq a_1 \leq 1,94)$$

$$e(-36,94 \leq a_2 \leq 1,53)$$

$$e(8,86 \leq a_3 \leq 48,65)$$

$$e(6,44 \leq a_4 \leq 36,30)$$

$$e(-2,99 \leq a_5 \leq 9,49)$$

$$e(7,57 \leq a_6 \leq 13,17)$$

$$e(7,56 \leq a_7 \leq 54,37)$$

$$e(49,21 \leq a_8 \leq 117,72)$$

$$e(-6,28 \leq a_9 \leq 10,52)$$

$$e(-38,82 \leq a_{10} \leq -7,57)$$

Então CLASSE 1

Regra para a classe: 2

$$\text{Se } (-6,52 \leq a_1 \leq 15,82)$$

$$\text{e } (-12,03 \leq a_2 \leq 20,44)$$

$$\text{e } (0,93 \leq a_3 \leq 82,59)$$

$$\text{e } (-4,71 \leq a_4 \leq 11,19)$$

$$\text{e } (-9,00 \leq a_5 \leq 6,03)$$

$$\text{e } (8,88 \leq a_6 \leq 15,59)$$

$$\text{e } (-46,87 \leq a_7 \leq 5,29)$$

$$\text{e } (18,71 \leq a_8 \leq 63,16)$$

$$\text{e } (-8,87 \leq a_9 \leq 6,12)$$

$$\text{e } (-55,27 \leq a_{10} \leq 0,44)$$

Então CLASSE 2

Regra para a classe: 3

$$\text{Se } (-9,27 \leq a_1 \leq 9,16)$$

$$\text{e } (-16,04 \leq a_2 \leq 10,94)$$

$$\text{e } (-44,01 \leq a_3 \leq -8,30)$$

$$\text{e } (-3,47 \leq a_4 \leq 13,04)$$

$$\text{e } (-9,62 \leq a_5 \leq 0,83)$$

$$\text{e } (11,44 \leq a_6 \leq 16,16)$$

$$\text{e } (-48,67 \leq a_7 \leq -8,56)$$

$$\text{e } (38,28 \leq a_8 \leq 84,77)$$

$$\text{e } (-4,81 \leq a_9 \leq 7,88)$$

$$\text{e } (-68,19 \leq a_{10} \leq -27,45)$$

Então CLASSE 3

Teste das Regras

Para exemplos da CLASSE 1 :

100% classificados como CLASSE 1;

0% classificados como CLASSE 2;

0% classificados como CLASSE 3.

Para exemplos da CLASSE 2 :

0% classificados como CLASSE 1;

98,59% classificados como CLASSE 2;

0% classificados como CLASSE 3.

Para exemplos da CLASSE 3 :

0% classificados como CLASSE 1;

0% classificados como CLASSE 2;

100% classificados como CLASSE 3;

Para o sistema de regras obtido a percentagem de acerto do conjunto de regras foi de 99,44% e a complexidade do conjunto de regras 5,8.

5.2 COMENTÁRIOS FINAIS

Os exemplos analisados, como mostrado, obtiveram excelentes resultados. Deve-se resultar que estes exemplos correspondem a problemas de classificação, com poucos dados, o que é desfavorável para as redes neurais. Por este motivo todos os dados foram utilizados no treinamento. Na realidade o que é recomendável nestes casos é o uso de validação cruzada – *cross validation* [1]. Com relação à validação cruzada múltipla dividi-se o conjunto disponível de N exemplos em K subconjuntos, $K > 1$. O modelo é treinado com todos os subconjuntos, exceto um, e o erro de validação é medido testando-o com este subconjunto deixado de lado no treinamento. Este procedimento é repetido para um total de K tentativas, cada vez usando um subconjunto diferente para a validação. O desempenho do modelo é avaliado pela média do erro quadrado obtido na validação sobre todas as tentativas do experimento.

Entretanto como o objetivo do trabalho era avaliar a implementação rápida e eficiente de uma ferramenta de extração de regras, pode-se concluir que foi atingido a meta estimada.

CAPÍTULO 6

CONCLUSÃO

Uma ferramenta que se proponha treinar e posteriormente extrair as regras de redes neurais de uma forma rápida e otimizada em um ambiente popular como o Matlab foi o objetivo e contribuição deste trabalho. A ferramenta proposta, neste trabalho permite que o usuário obtenha regras mais ou menos precisas, e respectivamente mais ou menos complexas, ou seja, de acordo com a necessidade do usuário.

Procurou-se escolher entre as opções sugeridas na bibliografia uma metodologia que pode-se ter um caráter geral e também oferecer opções para que o usuário decida um balanço entre precisão e complexidade das regras. Por este motivo a opção do Algoritmo RX do Lu et al.[2] pareceu-se bastante adequada. Para tornar este algoritmo mais eficiente restringiu-se ao estudo do problema de classificação de dados, neste caso por ser o aprendizado supervisionado pode-se agrupar os dados para cada classe e obter diretamente as regras de cada classe, eliminando-se a etapa de agregação – *merge*, obrigatório na versão original do algoritmo.

Desta forma o Algoritmo RX modificado, mostrou-se eficaz e eficiente: a eficiência provém do insignificante tempo de processamento exigido pelo algoritmo de agrupamento – *clustering*; a eficácia é constatada qualitativamente e quantitativamente no conjunto de regras. O número de regras é igual ao número de classes. Em relação à qualidade, observa-se a não ocorrência de regras redundantes, o que freqüentemente ocorre nos algoritmos baseados nos valores dos pesos das conexões.

Como continuação do estudo poderiam ser estudados outros algoritmos de clusterização. Além disso, a otimização dos clusters por algoritmos genéticos parece

ser bastante indicada. Como sugestão pode-se também implementar outros tipos de algoritmos de extração de regras e compara-los entre si.

Outros tipos de rede também necessitam ser estudados adequadamente. Na realidade, a atividade de explicitação e análise do conhecimento escondido em redes neurais é uma linha de pesquisa que se mostra em grande desenvolvimento.

BIBLIOGRAFIA

- [1] HAYKIN, S., **Redes Neurais – Princípios e Prática**, 2ª ed., Bookman, Porto Alegre, 2001.
- [2] LU, H., SETIONO, R., LIU, H. “Effective Data Mining Using Neural Networks”, **IEEE Transactions on Knowledge and Data Engineering**, v. 8, n. 6, pp. 957-961, 1996.
- [3] HRUSCHKA, E. R., **Um Estudo sobre a Extração de Regras de Redes Neurais em Aplicações de Data Mining**, Tese de M.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, 1998.
- [4] SANTOS, R. T., NIEVOLA, J. C., FREITAS, A. A., LOPES, H. S. “Extração de Regras de Redes Neurais via Algoritmos Genéticos”, **IV Congresso Brasileiro de Redes Neurais**, pp. 158-163, 1999.
- [5] HRUSCHKA, E. R., EBECKEN, N. F. F. “Extração de Regras de Redes Neurais por meio do Algoritmo RX Modificado: Um Exemplo de Aplicação em Modelagem de Dados Meteorológicos”, **IV Congresso Brasileiro de Redes Neurais**, pp. 047-051, 1999.
- [6] CRAVEN, M. W., SHAVLIK, J. W., “Using Neural Networks for Data Minig”, **Future Generation Computer Systems**, accepted to appear, 1997.
- [7] GALLANT, S. I. “Connectionist Expert Systems”, **Communications of the ACM**, v.31, n.2, pp. 152-169, 1988.

[8] HARTIGAN, J. A., **Clustering Algorithms**, 1ª ed. USA, John Wiley & Sons Inc., Academic Press, Inc., 1973.

[9] TOWELL, G. G., SHAVLIK, J., "Extracting Refined Rules from Knowledge-Based Neural Networks", **Machine Learning**, v.13, pág 71-101, 1993.

[10] OLIVEIRA, J. P., ULIANA, P. B., LIMA, W. C., "Algoritmos de Extração de Regras de Redes Neurais Artificiais". In: Anais do III Congresso Brasileiro de Redes Neurais, pág 173-177, Florianópolis, Julho de 1997.

[11] BREIMAN, L., FRIEDMAN, J. H., OLSHEN, R. A., STONE, C. J., **Classification and Regression Trees**, 1ª ed. Monterey, California, USA, Wadsworth & Brooks, 1984.

[12] Neural Network Toolbox User's Guide, documentação do programa Matlab R12, 2000.

[13] GAINES, B. R. "Transforming Rules and Trees into Comprehensible Knowledge Structures". In: **Advances in Knowledge Discovery and Data Mining**, MIT Press, pp. 205-229, 1996.

[14] MATSUMOTO, É. Y., **Matlab 6: Fundamentos de Programação**, 1ª ed., Érica, São Paulo, 2001.

APÊNDICE I

IMPLEMENTAÇÃO COMPUTACIONAL

Este apêndice contém a listagem do código do programa implementado para a extração de regras de redes neurais. Utilizou-se da linguagem de programação Matlab.

```
%  
% Programa para extracao de regras de redes neurais.  
% Algoritmo RX Modificado  
%  
  
clc; clear;  
  
% Inicializacao de vetores e matrizes para agilizar o programa  
col = []; nec = []; maxt = []; mint = [];  
  
flag = 0; % Enquanto flag=0 os arquivos lidos nao estao com dimensoes  
compativeis  
while flag == 0;  
  
% Leitura do arquivo de exemplos  
[Nome Caminho] = uigetfile({'*.mat'},'Arquivo com Valores de exemplos (num ent x  
num exemp)');  
if ~ischar(Nome)  
    warndlg ('Arquivo nao escolhido.');else  
    arq_ex = fullfile(Caminho,Nome); load(arq_ex);  
end
```

```

% Leitura do arquivo de bias
[Nome Caminho] = uigetfile({'*.mat'},'Arquivo com Valores de bias (num unid ocult x
1)');
if ~ischar(Nome)
    warndlg ('Arquivo nao escolhido.');
```

```

else
    arq_bias = fullfile(Caminho,Nome);    load(arq_bias);
end

% Leitura do arquivo de pesos
[Nome Caminho] = uigetfile({'*.mat'},'Arquivo com pesos (num unid ocult x num ent)');
if ~ischar(Nome)
    warndlg ('Arquivo nao escolhido.');
```

```

else
    arq_pesos = fullfile(Caminho,Nome);    load(arq_pesos);
end

% Verificando as dimensoes dos arquivos lidos
[m_ex n_ex] = size(exemplos);    [m_b n_b] = size(bias);    [m_p n_p] = size(pesos);

if (m_ex ~= n_p) | (m_p ~= m_b)
    warndlg('Dimensoes nao estao de acordo!');    % Arquivos com dimensoes nao
compativeis
else
    flag = 1;    % Arquivos com dimensoes compativeis
end
end

% Definicao do numero de exemplos, do numero de entradas e do numero de
unidades ocultas
nex = n_ex;    nue = n_p;    nu = m_p;

% Leitura do multiplicador de desvio-padrao e do numero de classes
prompt = {'Fator multiplicador do desvio-padrao (1, 2, 3 ou 4)', 'Numero de classes'};
title = 'Algoritmo de Extracao de Regras RX Modificado';
lines= 1;
def = {"",""};

```



```

fprintf('\n*   Algoritmo RX Modificado   *');
fprintf('\n*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*');
fprintf("");

% Apresenta na tela as equacoes de calculo das ativacoes das unidades ocultas
matriz = [bias pesos];
[m_mat n_mat] = size(matriz);

fprintf("");
fprintf('\n\nCalculo das Ativacoes das Unidades Ocultas');
fprintf('\n-----');
fprintf("");

for i=1:m_mat
    fprintf('\na%g =',i);
    for j=0:(n_mat-1)
        if j == 0
            fprintf('%6.3f +',matriz(i,j+1));
        elseif j == n_mat-1
            fprintf('%6.3f x%g',matriz(i,j+1),j);
        else
            fprintf('%6.3f x%g +',matriz(i,j+1),j);
        end
    end
end

% Inicia o processo de clusterizacao
fprintf("");
fprintf('\n\nClusterizacao');
fprintf('\n-----');
fprintf("");

for i=1:nc    % Para cada classe
    fprintf('\n\nClasse associada: %g ',i);
    lim1 = sum(vetor_exe(1:i))+1;
    lim2 = sum(vetor_exe(1:(i+1)));
    for j=1:nu    % Para cada unidade de ativacao

```

```

fprintf('\n\nUnidade de ativacao: %g \n',j);
cluster_in = [ ]; % Recebera os valores de ativacao incluidos no cluster
num_clu
cluster_out = [ ]; % Recebera os valores de ativacao nao incluidos no cluster
num_clu
x = col(lim1:lim2,j);
num_clu = 1;

while num_clu <= tk; % Para cada conjunto de valores de ativacao que resta
for k=1:length(x)
media = mean(x); % Calcula a media e o desvio desse conjunto
desvio = std(x);
if dist(x(k),media) <= multi*desvio % Checa se o valor pertence ao cluster
cluster_in = [cluster_in x(k)]; % Valor selecionado para o cluster
else
cluster_out = [cluster_out x(k)]; % Valor nao selecionado para o cluster
end
end

if isempty(max(cluster_in)) % Checa se o cluster esta vazio e define como
NaN
maxt(i,j,num_clu) = NaN;
else
maxt(i,j,num_clu) = max(cluster_in); % Calcula o valor maximo do cluster
end

if isempty(min(cluster_in)) % Checa se o cluster esta vazio e define como
NaN
mint(i,j,num_clu) = NaN;
else
mint(i,j,num_clu) = min(cluster_in); % Calcula o valor minimo do cluster
end;

nec(i,j,num_clu) = length(cluster_in); % Numero de elementos em cada
cluster

% Apresenta valores para cada cluster na tela

```

```

        fprintf('Cluster: %g \n',num_clu)
        fprintf('No de elementos: %g \t',length(cluster_in))
        fprintf('Valor maximo: %6.2f \t',max(cluster_in))
        fprintf('Valor minimo: %6.2f \r',min(cluster_in))
        num_clu = num_clu + 1;
        x = cluster_out;cluster_in = [ ]; cluster_out = [ ];
    end
end
end

%   Apresentacao na tela do numero do multiplicador de desvio padrao
fprintf("");
fprintf("\n\n");
fprintf('\nMultiplicador de desvio padrao : %g',multi);
fprintf("\n----- -- ----- ---- -");
fprintf("");

%   Escolha do numero de clusters a serem usados nas regras
prompt = {'Numero minimo de elementos no cluster (%)'};
title = 'Selecao dos clusters significativos para as regras';
lines= 1;
def   = {''};
answer = inputdlg(prompt,title,lines,def);
num_min = str2num(answer{1});
lim = round((num_min/100)*(vetor_exe)); %   Numero minimo depende do numero de
exemplos da classe

%   Numero de clusters significativos
fprintf("");
fprintf("\n\n");
fprintf('\nNumero Minimo de Elementos nos Clusters');
fprintf("\n----- ---- -");
fprintf("");
fprintf("\n\nO limite para cada classe e:");
for i=1:nc
    fprintf("\n\n%g elementos para a classe %g.',lim(i+1),i);
end

```

```

% Apresentacao na tela apenas dos clusters significativos - nao vazios
fprintf("");
fprintf("\n\n");
fprintf("\nClusters Significativos");
fprintf("\n-----");
fprintf("");
for i=1:nc
    for j=1:nu
        % Desconsidera unidades de ativacao cujo valor de ativacao varia pouco ao
        longo dos exemplos
        if (std(col(:,j)) < 0.1)
            fprintf("\n\nA unidade %g nao sera considerada por ter ativacao constante.',j);
        else
            for z=1:tk
                % Desconsidera clusters com numero de elementos menor que o minimo
                para aquela classe
                if (nec(i,j,z) < lim(i+1))
                    fprintf("\n\nO cluster %g nao sera considerada por ter poucos elementos.',z);
                else
                    % Imprime os clusters que geram regras
                    fprintf("\n\nRegras para a classe: %g\t',i);
                    fprintf('Unidade de ativacao: %g\n',j);
                    fprintf('Numero do cluster: %g\n',z);
                    fprintf('Numero de elementos: %6.2f \t',nec(i,j,z));
                    fprintf('Limite minimo: %6.2f \t',mint(i,j,z));
                    fprintf('e limite maximo: %6.2f ',maxt(i,j,z));
                end
            end
        end
    end
end

% Apresentacao na tela das regras e contando o numero de premissas
fprintf("\n\n");
fprintf("\nRegras para cada Classe");
fprintf("\n-----");

```

```

fprintf("");

% E repetido o loop anterior mas com a apresentacao na tela das regras
num_prem = 0; % Contador para o numero de premissas a ser utilizado
for i=1:nc
    fprintf("\n\nRegra para a classe: %g\t',i);
    for j=1:nu
        if (std(col(:,j)) < 0.1)
            fprintf("");
        else
            for z=1:tk
                if (nec(i,j,z) < lim(i+1)) | (mint(i,j,z) == NaN) | (maxt(i,j,z) == NaN)
                    fprintf("");
                elseif (z == 2) | (z == 3) | (z == 4)
                    fprintf("\n\tou (%6.2f <= a%g <= %6.2f)',mint(i,j,z),j,maxt(i,j,z));
                    num_prem = num_prem + 1;
                else
                    if (j == 1)
                        fprintf("\nSe(%6.2f <= a%g <= %6.2f)',mint(i,j,z),j,maxt(i,j,z));
                        num_prem = num_prem + 1;
                    else
                        fprintf("\ne(%6.2f <= a%g <= %6.2f)',mint(i,j,z),j,maxt(i,j,z));
                        num_prem = num_prem + 1;
                    end
                end
            end
        end
    end
    fprintf("\nEntao CLASSE\t');
    fprintf(num2str(i));
end

% Teste do conjunto de regras
fprintf("\n\n');
fprintf("\nTeste das Regras e Estatisticas');
fprintf("\n----- --- ----- - -----');
fprintf("");

```

```

% A variavel check tem num de linhas igual ao numero de exemplos e num de
colunas igual ao numero
% de classes. Cada exemplo e testado para cada conjunto de regras de cada classe.
Se for aprovado
% o valor 0 e substituido pelo valor 1.

check = zeros(nexe,nc);
for m=1:nexe
    teste = col(m,:); % Linha com as ativacoes correspondentes ao exemplo testado
    for i=1:nc
        contador = 0*teste;
        % A variavel contador e um vetor nulo com dimensao igual ao num de unidades
de ativacao.
        % Cada vez que uma premissa for satisfeita, a posicao correspondente a essa
unidade e
        % incrementada. Se ao final alguma das regras nao for satisfeita haverá um 1
em alguma
        % posicao. Ex: [1 2 1 3 4 5 1 1 1 3] aprovada ou [1 1 0 1 2 0 2 3 1 1 ] reprovada

    for j=1:nu
        if (std(col(:,j)) < 0.1)
            fprintf("");
            contador(j) = contador(j) + 1;
        else
            for z=1:tk
                if (nec(i,j,z) <= lim(i+1)) | (mint(i,j,z) == NaN) | (maxt(i,j,z) == NaN)
                    fprintf("");
                elseif (z == 2) | (z == 3) | (z == 4)
                    if (mint(i,j,z) <= teste(j)) & (teste(j) <= maxt(i,j,z))
                        contador(j) = contador(j) + 1;
                    end
                else
                    if (j == 1)
                        if (mint(i,j,z) <= teste(j)) & (teste(j) <= maxt(i,j,z))
                            contador(j) = contador(j) + 1;
                        end
                    end
                end
            end
        end
    end
end

```

```

        else
            if (mint(i,j,z) <= teste(j)) & (teste(j) <= maxt(i,j,z))
                contador(j) = contador(j) + 1;
            end
        end
    end
end
end
end
end
end
    if isempty(find(contador == 0)) % Checa o vetor procurando um zero em alguma
posicao
        check(m,i) = 1;
    end
end
end

% Contabilizando as Estatisticas

% estat e uma matriz quadrada na qual cada elemento representa o total de
exemplos de cada classe
% que foi classificado como sendo de cada classe. Notar que alguns elementos sao
classificados em
% todas as classes.
%     Ex: [ 32  1  0 ;
%         3 31  1 ;
%         0  1 28 ]

estat = zeros(nc);
ind = cumsum(vetor_exe);
for i=1:nc
    for j=1:nc
        estat(j,i) = sum(check(ind(j)+1:ind(j+1),i));
    end
end

% Apresentando as Estatisticas

```

```

for i=1:nc
    estat2(:,i) = 100*(estat(:,i)/vetor_exe(i+1));
end

% Como alguns exemplos são classificados como sendo de mais de uma classe, a
% soma das
% porcentagens pode ser maior que 100%

for i=1:nc
    fprintf('\n\nPara exemplos da CLASSE %g :\t\n',i);
    for j=1:nc
        fprintf('\n%g%% classificados como CLASSE %g.',estat2(i,j),j);
    end
end

% Porcentagem dos exemplos classificados corretamente mesmo que também
% tenham sido classificados
% na classe errada
taxa = 100*trace(estat)/nexe;
fprintf('\n\nA taxa de classificação correta é de %4.2f%%',taxa);

num_prem_conj = num_prem/nc;
fprintf('\n\nNúmero de premissas do conjunto é %4.2f.',num_prem_conj);
complexidade = 0.6*nc + 0.4*num_prem_conj;
fprintf('\n\nA complexidade deste conjunto de regras é igual a %4.2f.',complexidade);

```