FUNDAÇÃO DE ENSINO “EURÍPIDES SOARES DA ROCHA”
CENTRO UNIVERSITÁRIO “EURÍPIDES DE MARÍLIA” - UNIVEM
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

APARECIDO NARDO JUNIOR

APLICAÇÃO DE REDES NEURAIS UTILIZANDO O SOFTWARE MATLAB

MARÍLIA
2005

APARECIDO NARDO JUNIOR

APLICAÇÃO DE REDES NEURAIS UTILIZANDO O SOFTWARE MATLAB

Monografia apresentada ao Curso de Ciência da Computação da Fundação de Ensino "Eurípides Soares da Rocha", Mantenedora do Centro Universitário Eurípides de Marília – UNIVEM, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação

Orientador:
Prof. Dr. José Celso Rocha

MARÍLIA
2005

NARDO JUNIOR, Aparecido

Aplicação de Redes Neurais utilizando o software MATLAB/ Aparecido Nardo Junior; orientador: José Celso Rocha. Marília, SP: [s.n], 2005.

Monografia (Graduação em Ciência da Computação) – Curso de Ciência da Computação, Fundação de Ensino "Eurípides Soares da Rocha", Mantenedora do Centro Universitário Eurípides de Marília – UNIVEM.

1. Redes Neurais 2. Sistemas de Controle

CDD: 006.32

APARECIDO NARDO JUNIOR
R.A. 302181

APLICAÇÃO DE REDES NEURAIS UTILIZANDO O SOFTWARE MATLAB

BANCA EXAMINADORA DA MONOGRAFIA PARA
OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO

CONCEITO FINAL: ______________(________)

ORIENTADOR: ________________________
Prof. Dr. José Celso Rocha

1º EXAMINADOR: ____________________________________

2º EXAMINADOR: _____________________________________

Marília, ____ de _______________________ de 2005.

NARDO JUNIOR, Aparecido. Aplicação de Redes Neurais utilizando o software Matlab. Monografia (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2005.

RESUMO

Este trabalho visa o desenvolvimento de uma aplicação prática de redes neurais utilizando as ferramentas do software Matlab. É abordado conceito básico de redes neurais mostrando como atua um neurônio artificial, topologias de rede e algoritmos de aprendizado para a adaptação dos pesos. Foi implementada e treinada uma rede de múltiplas camadas em diferentes topologias para a base de dados *Zoo*, calculando a soma dos erros quadráticos para o conjunto de treinamento e teste, sendo definida a melhor topologia de acordo com o resultado obtido. O trabalho envolveu um estudo inicial do software Matlab, compreendendo o funcionamento na aplicação de suas funções para posterior uso na implementação do projeto apresentado.

Palavras-chave: redes neurais; sistemas de controle

NARDO JUNIOR, Aparecido. Aplicação de Redes Neurais utilizando o software Matlab. Monografia (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2005.

ABSTRACT

This research pretends develop a Artificial Neural Network practical application using the Matlab tools. It boards Artificial Neural Network basic concepts showing how acts an Artificial Neuron, network topologies and learning algorithms to fit the weights. The network was implemented and trained with multiples layer in many topologies for the Zoo data base, calculating the sum squared error for the training set and defining the best topology according to the results. The research comprise an initial study of Matlab, understanding the operation at function application after use in presented project implementation.

Keywords: Artificial Neural Networks, Control System.

LISTA DE FIGURAS

Figura 1.1 – Neurônio Biológico..12

Figura 1.2 – Neurônio MCP..14

Figura 1.3 – Rede de camada única..15

Figura 1.4 – Rede de múltiplas camadas..16

Figura 1.5 – Aprendizado supervisionado...17

Figura 1.6 – Aprendizado não-supervisionado..18

Figura 1.7 – Aprendizado por reforço..21

Figura 1.8 – Funcionamento de um perceptron..23

Figura 1.9 – Arquitetura de uma rede MLP..27

Figura 2.1 - Gráfico utilizando função plot...38

Figura 3.1 – Gráfico com dez neurônios na camada intermediária...............................50

Figura 3.2 – Gráfico com quinze neurônios na camada intermediária..........................51

Figura 3.3 – Gráfico com vinte neurônios na camada intermediária.............................51

LISTA DE ABREVIATURAS E SIGLAS

MCP: Perceptron de McCulloch e Pitts

MLP: Perceptron Múltiplas Camadas

MSE: Mean Squared Error

RNA: Redes Neurais Artificiais

SSE: Sum Squared Error

SUMÁRIO

INTRODUÇÃO 10

CAPÍTULO 1 – REDES NEURAIS..11

1.1 REDES NEURAIS BIOLÓGICAS...11

1.2 REDES NEURAIS ARTIFICIAIS..12

1.3 HISTÓRICO..12

1.4 MODELO MCP...14

1.5 ARQUITETURAS DE REDES...15

1.6 APRENDIZADO..16

1.6.1 Aprendizado supervisionado...17

1.6.1.1 Aprendizado por correção de erros..18

1.6.2 Aprendizado não-supervisionado..18

1.6.2.1 Aprendizado Hebbiano..19

1.6.2.2 Regra de Oja..20

1.6.2.3 Aprendizado por competição...20

1.6.2.4 Aprendizado por reforço..21

1.7 PERCEPTRON...22

1.7.1 Algorítmo de aprendizado do perceptron..22

1.7.2 Implementação do perceptron no Matlab..25

1.8 REDES MÚLTIPLAS CAMADAS...26

1.8.1 Arquitetura de uma rede MLP...27

1.8.2 Treinamento de redes MLP..28

1.8.3 Algorítmo backpropagation...28

1.8.3.1 Funcionamento do algorítmo backpropagation...29

CAPÍTULO 2 – A FERRAMENTA MATLAB..31

2.1 INTRODUÇÃO AO MATLAB..31

2.2 CARACTERÍSTICAS BÁSICAS DA FERRAMENTA..31

2.2.1 A janela de comandos da ferramenta...32

2.2.2 Variáveis do software Matlab...33

2.3 MATRIZES...34

2.3.1 Gerando vetores...34

2.3.2 Elementos das matrizes...35

2.4 MODO GRÁFICO..38

2.5 CRIAÇÃO DE ARQUIVOS NO MATLAB..38

2.5.1 Arquivos de comando...39

2.5.2 Arquivos gráficos..39

2.6 TOOLBOX DE REDES NEURAIS..40

2.6.1 Funções de criação de rede...40

2.6.2 Funções de transferência..41

2.6.3 Funções de adaptações de pesos e bias...43

2.6.4 Função de treino da rede...44

2.6.5 Medidas de desempenho da rede...45

2.6.6 Variáveis de resposta de uma rede...45

2.6.7 Outras funções...46

CAPÍTULO 3 – APLICAÇÃO PARA A BASE DE DADOS ZOO......................................47

3.1 TREINAMENTO DE UMA REDE MLP PARA A BASE DE DADOS ZOO.................47

3.2 ANÁLISE DE RESULTADO...49

CONCLUSÃO..53

REFERÊNCIAS..54

ANEXOS..55

INTRODUÇÃO

No estudo inicial é apresentada toda a parte conceitual de redes neurais, mostrando as diferentes arquiteturas de redes existentes, o neurônio artificial e sua forma de aprendizado.

Posteriormente, foi estudada e implementada uma rede perceptron de camada única, exercitando aplicações desenvolvidas no software Matlab, e analisando exemplos criados. Para isso, foi necessário a elaboração de problemas para a rede, que através de adaptações nos seus parâmetros, mostrou uma resposta compatível com o que foi proposto.

No segundo capítulo tem-se uma visão sobre o uso do software Matlab envolvendo o seu conhecimento geral no que se refere à forma de utilização das funções na janela de comando com pequenas operações matemáticas.

Toda a parte de ferramenta de redes neurais com suas funções apropriadas foram descritas para a possível utilização correta do recurso oferecido pelo software.

No terceiro capítulo foi desenvolvida uma aplicação que treina e testa uma rede de múltiplas camadas com diferentes configurações para a base de dados Zoo, com seus respectivos resultados e observações finais.

Na aplicação, características de diversos animais foram aplicados para treinamento em uma rede MLP, com algorítmo *backpropagation*, e, como teste de reconhecimento das características aprendidas, outros animais foram colocados à prova.

CAPÍTULO 1 – REDES NEURAIS

1.1 REDES NEURAIS BIOLÓGICAS

Sabe-se que o cérebro humano é composto por aproximadamente 100 bilhões de neurônios, conectados a 10 mil outros (ALVES JUNIOR, 1999), em média, e que todas as funções do corpo de um ser humano é controlado diretamente por estas células.

Esses neurônios são ligados uns aos outros através de sinapses, a qual tem por função transmitirem estímulos por componentes químicos Na+ que corresponde ao Sódio e K+ que é o Potássio, proporcionando uma grande capacidade de armazenamento e processamento de informações.

Quando o impulso é recebido, o neurônio responsável o processa, e atingindo seu limite de excitação, dispara uma resposta por impulso novamente, produzindo uma substância neurotransmissora que passa do corpo celular para o axônio da célula. Esses pulsos são controlados via frequência com que são enviados, alterando a polaridade na membrana pós-sináptica, tendo uma fundamental importância para a resposta do raciocínio do ser humano (BRAGA, 2000).

Em comparação às redes neurais artificiais, temos que seus sinais não possuem polaridade negativa e não são uniformes como as artificiais, apresentando-a somente em algumas partes do organismo. Não podemos definir seus pulsos como sincronizados ou não, pelo fato de não serem contínuos, diferentemente do neurônio artificial.

As principais partes que compõem um neurônio natural são:

- dendritos, cuja finalidade é receber os estímulos provenientes de outros neurônios.

- corpo da célula, também conhecida de soma, cuja finalidade é processar as informações coletadas pelos dendritos.

- axônio, que tem a funcionalidade de transmissão dos estímulos para as células interligadas.

A seguir, é visto a representação biológica de um neurônio natural:

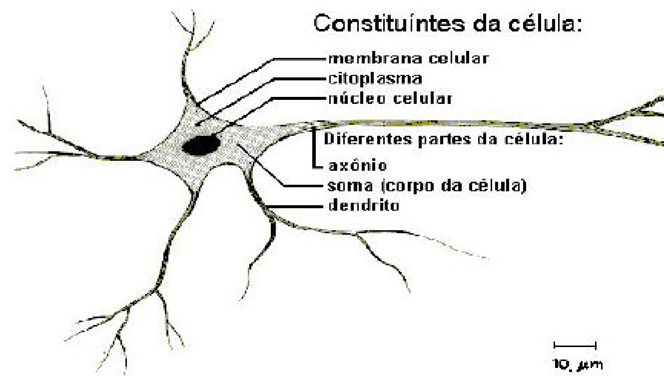


Figura 1.1 – Neurônio Biológico

1.2 REDES NEURAIS ARTIFICIAIS

Uma rede neural artificial é um sistema paralelo distribuído composto por unidades de processamento que executam funções matemáticas, cujo objetivo é solucionar problemas, formando um sistema representativo de um cérebro humano, capaz de aprender através de exemplos e de generalizar a informação aprendida.

Essas unidades, também chamadas de nodos, são dispostas por uma ou mais camadas de rede, interligadas umas às outras através dos pesos sinápticos que armazenam o conhecimento adquirido pela rede.

1.3 HISTÓRICO

No ano de 1943, dois pesquisadores, Warren McCulloch e Walter Pitts, conseguiram

representar o primeiro neurônio artificial utilizando-se de ferramentas matemáticas, se concentrando principalmente em apresentar suas capacidades computacionais e não se preocupando no que se refere às técnicas de aprendizado.

Após alguns anos, o aprendizado das redes artificiais começou a ser estudado com mais importância. No ano de 1949, Donald Hebb apresentou sua teoria de aprendizagem que defendia a tese da variação dos pesos de entrada dos nodos como a forma de se atingir o objetivo proposto, ficando conhecida como a regra de Hebb (apud BRAGA, 2000, p.3).

Em 1951, Minsky construiu o primeiro neurocomputador com capacidade de aprendizado, onde ajustava automaticamente os pesos entre as sinapses.

Frank Rosenblatt, no ano de 1958, criou o modelo chamado de Perceptron, capaz de classificar certos tipos de padrões, incrementando sinapses ajustáveis aos nodos.

Uma nova regra de aprendizado surgiria no ano de 1960, desenvolvida por Widrow e Hoff (apud BRAGA, 2000, p.3), com o mesmo nome de seus desenvolvedores, sendo bastante utilizado ainda nos dias atuais, conhecida também como regra delta. É baseada no método do gradiente para minimização do erro na saída de um neurônio.

Em 1969, Minsky e Papert constataram que o perceptron não conseguia analisar problemas não-linearmente separáveis, ou seja, problemas cuja solução pode ser obtida dividindo-se o espaço de entrada em duas regiões através de uma reta.

Na década de 70, poucas pesquisas foram realizadas no sentido de aperfeiçoamento dos neurônios artificiais e sua capacidade de aprendizagem. Em 1974, Werbos desenvolve bases para o algoritmo de aprendizado *backpropagation*.

Prosseguindo aos estudos e achando soluções para problemas levantados anteriormente, no ano de 1982, John Hopfield publicou um artigo que abordou as propriedades associativas das RNA's. Foi assim que novos interesses de pesquisadores surgiram, acreditando novamente na capacidade antes criticada dos neurônios.

No ano de 1986, Rumelhart, Hinton e Williams (apud BRAGA, 2000, p.4) descrevem o algorítmo de treinamento *backpropagation*, aplicada em redes de múltiplas camadas, o que possibilitou grandes avanços no estudo de uma forma geral.

À partir de então, inúmeros congressos e simpósios internacionais começaram a ser realizados, favorecendo a difusão do conhecimento adquirido.

1.4 MODELO MCP

Assim chamado por representar as iniciais dos nomes de seus criadores, o modelo de McCulloch e Pitts visa simbolizar o funcionamento de um neurônio biológico matematicamente.

O neurônio recebe n terminais de entrada x1, x2, x3, ..., xn e um terminal de saída y, representando os dendritos e axônios, respectivamente, se comparados ao neurônio biológico.

Pesos w1, w2, w3,..., wn são acoplados aos valores de cada entrada, com valores de sinais positivos(excitatórias) ou negativos(inibitórias).

De forma geral, a operação de uma célula da rede se resume em:

1- Sinais são apresentados à entrada

2- Cada sinal é multiplicado por um peso que indica sua influência na saída

3- É feita a soma ponderada dos sinais que produz um nível de atividade

4- Se este nível excede um limite (*threshold*), a unidade produz uma saída.

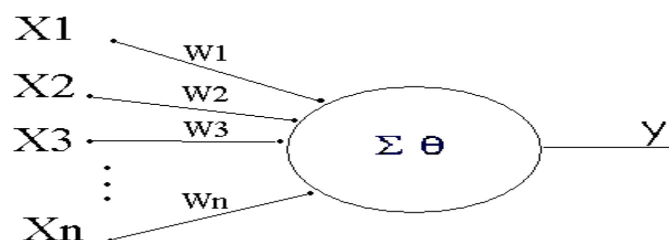


Figura 1.2 – Neurônio MCP

O modelo MCP original apresenta algumas limitações relevantes:

- uma rede implementada com neurônios MCP somente atinge o aprendizado em casos de funções linearmente separáveis.
- os pesos sinápticos são fixos
- pesos com sinais negativos são mais indicados para representar disparos inibidores

1.5 ARQUITETURAS DE REDES

A arquitetura define a disposição dos neurônios e suas ligações sinápticas.

Para a definição da arquitetura de uma rede, os seguintes parâmetros são considerados:

- números de camadas da rede
- número de nodos em cada camada
- tipo de conexão entre os nodos e sua topologia

Primeiramente, são definidas redes quanto ao número de camadas:

- rede de camada única: existe apenas um nó entre a entrada e a saída da rede, conforme a Figura 1.3.

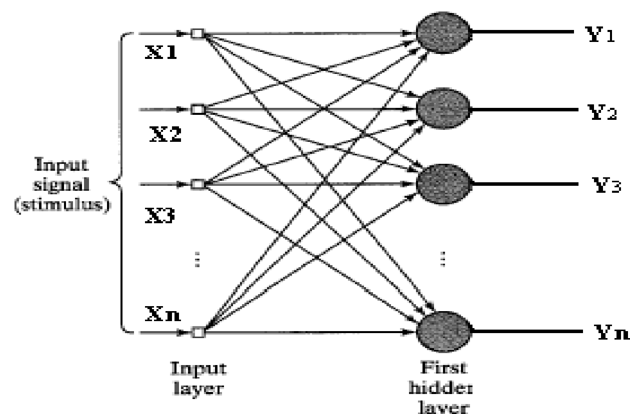


Figura 1.3 - Rede de camada única

- rede de múltiplas camadas: essa rede possui mais de um neurônio entre a entrada e a saída, como pode ser visto na Figura 1.4.

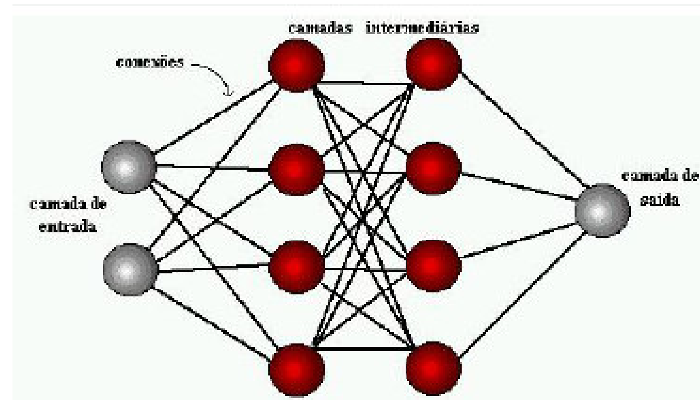


Figura 1.4 – Rede de múltiplas camadas

Quanto à forma de conexão existente em uma rede, define-se:

- feedforward ou acíclica: a saída de um neurônio em uma determinada camada i da rede, não pode alimentar a entrada de um outro em camada inferior ou igual a i.
- feedback ou cíclica: a saída de um neurônio da camada i alimenta a entrada de outros de camada inferior ou igual à i.
- auto-associativa: quando todas as ligações são cíclicas

Finalmente, a classificação quanto à conectividade:

- rede de conexão fraca ou parcial
- rede completamente conectada

1.6 APRENDIZADO

O processo de aprendizagem é a maneira pela qual os parâmetros de uma rede neural são ajustados através de uma sequência contínua de valores apresentados à entrada para que

através de seus algorítmos, adquira o conhecimento necessário na solução de problemas propostos à rede.

Os métodos desenvolvidos de aprendizagem são agrupados em dois paradigmas principais: aprendizado supervisionado e o aprendizado não-supervisionado.

1.6.1 Aprendizado supervisionado

É o método de aprendizado mais comum de treinamento, sendo assim chamado porque a entrada e a saída desejadas para a rede são fornecidas por um supervisor.

O objetivo é encontrar uma ligação entre os valores de entrada e saída, atingindo o menor erro possível, o que resultaria, num valor ideal, tendo o erro igual a zero.

O procedimento realizado resume-se no cálculo da saída da rede após apresentação do padrão de entrada, onde é comparado o valor desejado de saída com o valor gerado, sendo assim calculado o erro, realizando-se a diferença entre esses valores. De acordo com o resultado, os pesos são alterados de forma a ajustar seus valores ao que se deseja na saída.

Tem como desvantagem, caso a ausência do professor venha acontecer, o problema de a rede não aprender novos padrões apresentados para situações em que a rede não foi treinada. Como exemplos de algoritmos de aprendizado supervisionado, tem-se a regra delta e sua generalização em redes de múltiplas camadas usando o algoritmo *backpropagation*.

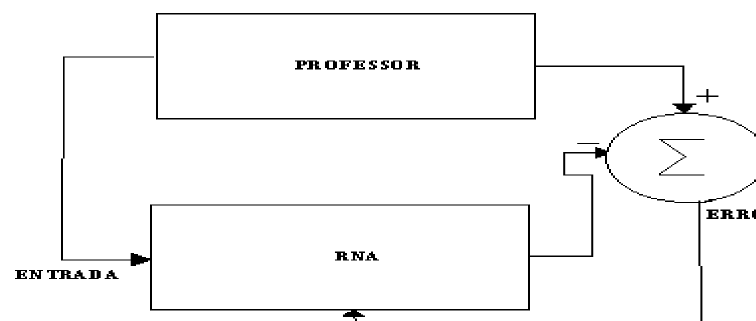


Figura 1.5 – Aprendizado supervisionado

1.6.1.1 Aprendizado por correção de erros

A adaptação por correção de erros procura minimizar a diferença encontrada entre a saída gerada pela rede, obtida pela soma ponderada das entradas com seus respectivos pesos, e a saída desejada.

O erro é simbolizado por e(t), a saída desejada é d(t) e y(t) a saída gerada na rede.

Para o cálculo da correção dos pesos, utiliza-se da fórmula matemática seguinte:

$$Wi(t+1) = Wi(t) + \eta.\varepsilon(t).Xi(t) \tag{1.1}$$

As variáveis Wi(t), η e Xi(t), são respectivamente, valores de pesos correntes, taxa de aprendizado e valores de entrada dos neurônios.

Esse método, também chamado de regra delta de aprendizado, é utilizado para o treinamento do perceptron e na generalização do algoritmo *backpropagation*.

1.6.2 Aprendizado não-supervisionado

No sistema de aprendizado não-supervisionado, não existe um professor para acompanhar o processo de aprendizado, monitorando os erros gerados. Para estes algoritmos, somente os padrões de entrada estão disponíveis para a rede, ao contrário do supervisionado, onde o treinamento tem pares de entrada e saída. Este método está ilustrado na Figura 1.6.

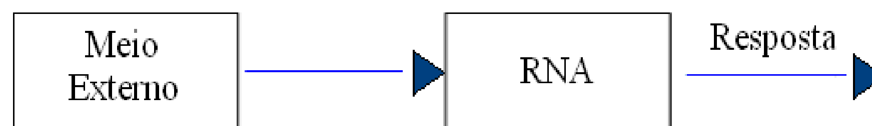


Figura 1.6 - Aprendizado não-supervisionado

Nos seres humanos, os estágios iniciais dos sistemas de visão e audição ocorrem pelo aprendizado não-supervisionado.

Dentre diversos métodos desenvolvidos, destacam-se o aprendizado hebbiano, modelo de Linsker, regra de Oja, regra de Yuille e aprendizado por competição.

1.6.2.1 Aprendizado Hebbiano

Desenvolvido por Hebb em 1949, a regra de aprendizado de Hebb (apud BRAGA, 2000, p.20) propõe que o peso de uma conexão sináptica deve ser ajustado se houver sincronismo entre os níveis de atividade das entradas e saídas.

Se dois neurônios, em lados distintos da sinapse, são ativados sincronamente, tem-se o fortalecimento desta sinapse, caso contrário, será enfraquecida ou até eliminada.

A sinapse hebbiana tem quatro características principais:

- Mecanismo Interativo: qualquer alteração na sinapse depende da interação entre as atividades pré e pós-sinápticas.
- Mecanismo Local: é no mecanismo local que as sinapses efetuam o aprendizado não-supervisionado, por meio da transmissão contínua de sinais.
- Mecanismo dependente do tempo: as modificações nas sinapses dependem do momento em que ocorrem as atividades pré e pós-sináptica.
- Mecanismo conjuncional ou correlacional: a sinapse hebbiana pode ser chamada de conjuncional, visto a ocorrência conjunta das atividades pré e pós-sinápticas ser capaz de provocar uma modificação, e, ainda de correlacional, porque uma correlação entre estas atividades ser suficiente de gerar mudanças.

Matematicamente, o postulado de Hebb é expressado pela equação (1.2):

$$\triangle Wji(t) = \eta . Yi(t).Xj(t) \quad (1.2)$$

As variáveis Y,X e η representam respectivamente o valor gerado na saída da rede, o padrão de entrada e a taxa de aprendizado.

1.6.2.2 Regra de Oja

Visando limitar a saturação dos pesos do vetor w (OJA, 1982 apud BRAGA, 2000, p.23), propôs uma adaptação, utilizando-se da regra de Hebb, como pode ser vista na equação:

$$\triangle W\mathbf{i} = \eta . \mathbf{Y.(Xi - Y.Wi)} \quad (1.3)$$

É assim aplicado à equação 1.3 a taxa de aprendizado(η), o valor aplicado à entrada do neurônio(Xi) com o correspondente peso (Wi), e a saída(Y) do nodo.

1.6.2.3 Aprendizado por competição

No aprendizado por competição temos que, dado um padrão de entrada, fazer as unidades de saída disputarem entre si para serem ativadas, e, consequentemente, seus pesos serem atualizados no treinamento.

Um problema nesse algoritmo é o fato de uma unidade ficar ativa a todo o momento, captando para si todo o espaço de entradas. Como solução, é necessário racionar os pesos, de forma que a soma dos pesos sobre as linhas de entrada de uma unidade tenha limite 1, o que provoca mudanças nos pesos de outras unidades (BRAGA, 2000, p.24).

O algoritmo do aprendizado competitivo se resume em:

1- Apresentar um vetor de entrada

2- Calcular a ativação inicial de cada unidade de saída

3 - Deixar as unidades de saída competirem até que apenas uma fique ativa

4 - Aumentar os pesos nas conexões da unidade de saída e de entrada ativa, proporcionando uma boa probabilidade de estar ativa na próxima repetição do padrão.

1.6.2.4 Aprendizado por reforço

O aprendizado por reforço é uma forma de aprendizado on-line onde de a rede fica em contínuo processo de adaptação de seus parâmetros para obter os resultados esperados.

O sinal de reforço realimentado à rede neural provém de uma resposta na saída da rede, indicando apenas que o valor na saída está ou não correto, não fornecendo à rede a resposta correta para determinado padrão de entrada (BRAGA, 2000, p.25).

A principal diferença entre o algoritmo supervisionado e o algoritmo por reforço é a medida de desempenho usada em cada um dos sistemas. No supervisionado, essa medida é baseada de acordo com um critério de erro conhecido fornecido pelo supervisor, enquanto no algoritmo por reforço esta é baseada em qualquer medida que possa ser fornecida ao sistema.

O aprendizado por reforço está ilustrado na Figura 1.7.

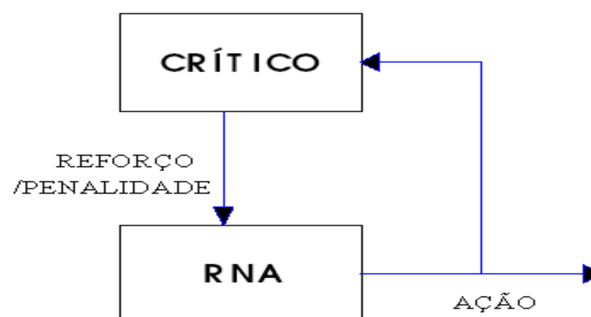


Figura 1.7 – Aprendizado por reforço

1.7 – PERCEPTRON

É um modelo de rede estudado originalmente por McCulloch e Pitts (apud BRAGA, 2000, p.35) e depois desenvolvido por Rosemblatt onde compôs uma estrutura de rede, tendo como unidades básicas, nodos MCP e uma regra de aprendizado.

Apesar da topologia apresentada conter três níveis, ela é conhecida como perceptron de uma única camada, pois somente o nível de saída possui propriedades adaptativas.

Após anos de estudo, foi demonstrado o teorema de convergência do perceptron, onde um nodo MCP treinado com o algoritmo de aprendizado do perceptron sempre resulta a um problema proposto caso seja linearmente separável, ou seja, problemas cuja solução pode ser obtida dividindo-se o espaço de entrada em duas regiões através de uma reta. Esse problema foi solucionado após as descrições da rede de Hopfield em 1982 e do algoritmo *backpropagation* em 1986.

1.7.1 Algoritmo de aprendizado do perceptron

O algoritmo de aprendizado é o responsável pela atualização dos pesos sinápticos do nodo MCP, de forma a adquirir conhecimento por padrões de entrada apresentados e suas respectivas saídas desejadas.

A fórmula geral de atualização de pesos é a mostrada na equação abaixo.

$$Wi(t+1) = Wi(t) + \eta.\varepsilon.Xi, \text{ onde } \varepsilon=(d-y) \quad (1.4)$$

As variáveis W simbolizam os pesos, η é a taxa de aprendizado que define a medida de rapidez com que o vetor de pesos será atualizado, ε é o erro gerado na diferença entre a saída desejada(d) e a que realmente saiu(y), e Xi é o padrão de entrada.

Para demonstrar o funcionamento e o método de aprendizagem do perceptron, considere o exemplo a seguir:

- Treinar uma rede Perceptron com quatro entradas com pesos iniciais de W=[0.2 -0.7 0.8 -0.1], com valor *threshold* de 0.3 e taxa de aprendizado de valor 1.

Tem como padrões de entrada os vetores X1=[1 0 1 0] e X2=[0 1 0 1], tendo como saída desejada os valores 1 e -1 respectivamente.

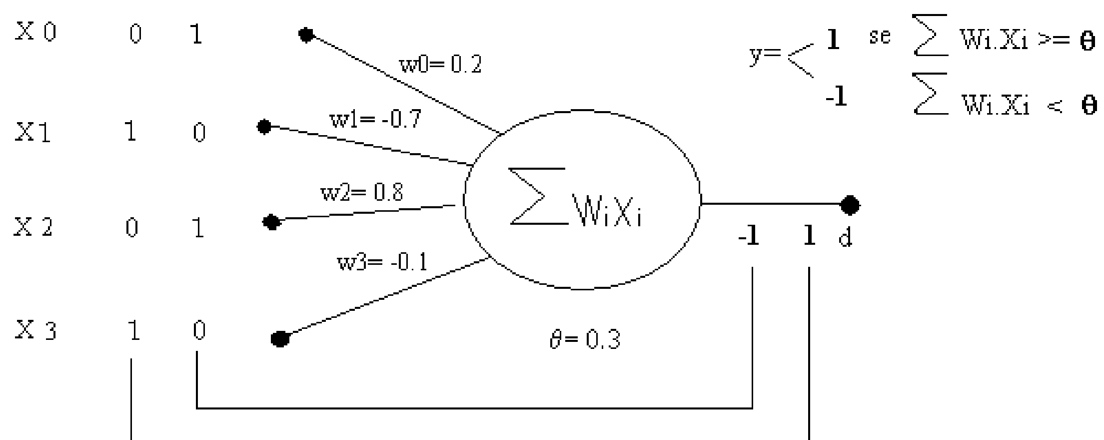


Figura 1.8 - Funcionamento de um perceptron

Primeiramente, é realizada a soma ponderada dos vetores de entrada e os seus pesos.

Para o vetor de entrada X1=[1 0 1 0], com η=1 e $\theta = 0.3$, é calculado:

$\sum$ Wi.Xi = 1 * 0.2 + 0 * 0.7 + 1 * 0.8 + 0 * 0.1 = 1

$\sum$ Wi.Xi = 1 é maior que $\theta = 0.3$, então y = 1

Então o valor gerado na saída sendo 1, diverge com o desejado –1, tendo que aplicar a regra para a atualização dos pesos:

W(t+1) = W(t) + η e Xi , onde e = (di – yi)

e = (-1 – (1)) = -2

W0(1) = W0(0) + η . e . X0 = 0.2 + 1 * –2 * 1 = -1.8

$W1(1) = W1(0) + \eta . e . X1 = -0.7 + 1 * -2 * 0 = -0.7$

$W2(1) = W2(0) + \eta . e . X2 = 0.8 + 1 * -2 * 1 = -1.2$

$W3(1) = W3(0) + \eta . e . X3 = -0.1 + 1 * -2 * 0 = -0.1$

Depois de atualizados os pesos para o primeiro padrão de entrada visto nos cálculos apresentados anteriormente, é calculado para o segundo padrão, onde X2 = [0 1 0 1] e pesos resultantes do padrão anterior W=[-1,8 –0,7 –1,2 –0,1].

$\sum Wi.Xi = 0 * -1.8 + 1 * -0.7 + 0 * -1.2 + 1 * -0.1 = -0.8$

$\sum Wi.Xi = -0.8$ é menor que $\theta = 0.3$, então y= -1

Assim, o valor gerado na saída sendo -1, diverge com o desejado 1, sendo aplicado novamente a regra para a atualização dos pesos.

$e = (d - y) = (1 - (- 1)) = 2$

$W0(2) = W0(1) + \eta . e . X0 = -1.8 + 1 * 2 * 0 = -1.8$

$W1(2) = W1(1) + \eta . e . X1 = -0.7 + 1 * 2 * 1 = 1.3$

$W2(2) = W2(1) + \eta . e . X2 = -1.2 + 1 * 2 * 0 = -1.2$

$W3(2) = W3(1) + \eta . e . X3 = -0.1 + 1 * 2 * 1 = 1.9$

Novamente atualizados, deve-se aplicar o padrão inicial do primeiro passo para X1= [1 0 1 0], utilizando os pesos calculados na etapa anterior, como visto abaixo:

$\sum Wi.Xi = 1 * -1.8 + 0 * 1.3 + 1 * -1.2 + 0 * 1.9 = -1.8 +-1.2 = -3$

$\sum Wi.Xi = -3$ é menor que $\theta = 0.3$, então y= -1

Para y= -1, com o vetor X1, e d = -1, o erro tem valor zero (e = 0), não é necessária atualização dos pesos.

Para confirmação, aplica-se para o segundo padrão, com X2 = [0 1 0 1]:

$\sum$ Wi.Xi = 0 * -1.8 + 1 * 1.3 + 0 * -1.2 + 1 * 1.9 = 1.3 + 1.9 = 3.2

$\sum$ Wi.Xi = 3.2 é maior que $\theta = 0.3$, então y= 1

Como o valor desejado é 1, e o valor que foi produzido foi equivalente, o erro também para esta situação foi zero. Portanto, aqui a rede já está treinada.

1.7.2 Implementação do perceptron no Matlab

Toda a forma de aprendizado do algoritmo já desenvolvido e detalhado vem descrito computacionalmente no programa a seguir:

```
clc;    % limpa a janela de comando
clear   % limpa a memoria
net = newp([0 1; 0 1; 0 1;0 1],1,'hardlims','learnp'); % cria a rede
w = [0.2 -0.7 0.8 -0.1];  % vetor dos pesos das entradas
p = [1 0; 0 1; 1 0; 0 1]; % padroes de entrada para a rede
t = [-1 1]; % valores de saida correspondentes ao seu vetor
net.trainParam.lr = 1; % definindo a taxa de aprendizado
net.IW{1,1} = w;  % definindo os pesos para a rede
net.b{1} = 0.3;   % definindo o valor threshold para a rede

net = train(net,p,t);  % função para treinamento
pesos = net.IW{1,1}    % mostra os valores dos pesos de cada entrada
threshold   = net.b{1} % mostra o valor de limiar de excitaçao
taxa=net.trainParam.lr % mostra o valor da taxa de aprendizado
```

Na janela de comandos do Matlab é apresentado a seguinte resposta:

```
TRAINC, Epoch 0/100
TRAINC, Epoch 2/100
TRAINC, Performance goal met.

pesos = -1.8000  1.3000 -1.2000  1.9000
threshold = 0.3000
taxa = 1
```

É mostrado em ordem de resposta os pesos finais, limiar de excitação e a taxa de aprendizado da rede, atingindo o treinamento após duas épocas.

1.8 REDES MÚLTIPLAS CAMADAS

Redes MLP (Multilayer Perceptron) apresentam um poder computacional muito maior do que aquele apresentado pelas redes com uma única camada. Ao contrário da rede de uma só camada, as redes MLP's podem trabalhar com dados que não são linearmente separáveis, onde a precisão obtida depende do número de nodos utilizados na camada intermediária.

Minsky e Papert (apud AZEVEDO, 2000, p.81) analisaram matematicamente o Perceptron e demonstraram que redes de uma camada não são capazes de solucionar problemas que não sejam linearmente separáveis, como é o caso típico da porta XOR.

Como não acreditavam na possibilidade de se construir um método de treinamento para redes com mais de uma camada, eles concluíram que as redes neurais seriam sempre suscetíveis a essa limitação.

Contudo, o desenvolvimento do algoritmo de treinamento *backpropagation*, por Rumelhart, Hinton e Williams em 1986, precedido por propostas semelhantes ocorridas nos anos 70 e 80, mostrou que é possível treinar eficientemente redes com camadas intermediárias, resultando no modelo de redes neurais artificiais mais utilizado atualmente, as redes Perceptron Multi-Camadas (MLP), treinadas com o algoritmo *backpropagation.*

Se existirem as conexões certas entre as unidades de entrada e um conjunto suficientemente grande de unidades intermediárias, pode-se sempre encontrar a representação que irá produzir o mapeamento correto da entrada para a saída através das unidades intermediárias.

Como provou Cybenko, a partir de extensões do Teorema de Kolmogoroff, são necessárias no máximo, duas camadas intermediárias, com um número suficiente de unidades por camada, para aproximar qualquer função.

As redes MLP têm sido aplicadas com sucesso em uma variedade de áreas, desempenhando tarefas tais como: classificação de padrões (reconhecimento), controle e processamento de sinais.

1.8.1 Arquitetura de uma rede MLP

Uma rede neural do tipo MLP é constituída por um conjunto de nós fonte, os quais formam a camada de entrada da rede (*input layer*), uma ou mais camadas escondidas (*hidden layers*) e uma camada de saída (*output layer*), como pode ser visto na Figura 1.9.

Com exceção da camada de entrada, todas as outras camadas são constituídas por neurônios e, portanto, apresentam capacidade computacional.

Nessas redes, cada camada tem uma função específica. A camada de saída recebe os estímulos da camada intermediária e constrói o padrão que será a resposta.

As camadas intermediárias funcionam como extratoras de características, seus pesos são uma codificação de características apresentadas nos padrões de entrada e permitem que a rede crie sua própria representação, mais rica e complexa, do problema.

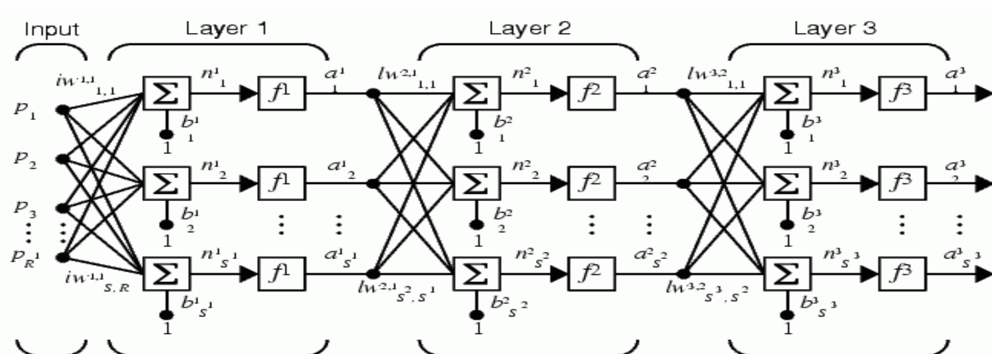


Figura 1.9 – Arquitetura de uma rede MLP

1.8.2 Treinamento de redes MLP

O treinamento consiste na aplicação de algoritmos estáticos ou dinâmicos responsáveis em adaptar uma rede alterando seus parâmetros para que os valores definidos de entrada geram seus correspondentes valores de saída corretamente.

Enquanto os algoritmos estáticos não alteram a estrutura de uma rede ao variar apenas os pesos, os dinâmicos podem aumentar ou diminuir o tamanho da rede quando altera o número de camadas, número de nodos nas camadas intermediárias e número de conexões.

Um dos algoritmos mais conhecidos para realizar este treinamento é o *backpropagation*, sendo um dos principais responsáveis pelo ressurgimento do interesse no estudo de redes neurais.

1.8.3 Algoritmo backpropagation

Em 1974, Werbos apresentou o algoritmo enquanto desenvolvia sua tese de doutorado em estatística e o chamou de "Algoritmo de Realimentação Dinâmica". Em 1982, Parker redescobriu o algoritmo e chamou-o de "Algoritmo de Aprendizado Lógico".

Porém, foi com o trabalho de Rumelhart, Hinton e Williams que em 1986 foi divulgado e se popularizou o uso do *backpropagation* para o aprendizado em redes neurais.

O algorítmo *backpropagation* baseia-se na heurística do aprendizado por correção de erro, em que o erro é retro-propagado da camada de saída para as camadas intermediárias da rede.

1.8.3.1 Funcionamento do algorítmo backpropagation

Durante o treinamento com o algoritmo *backpropagation*, a rede opera em uma sequência de dois passos, chamados de fase forward e fase backward.

Na fase forward, um padrão é apresentado à camada de entrada da rede. A atividade resultante flui através da rede, camada por camada, até que a resposta seja produzida pela camada de saída ficando com os pesos sinápticos todos fixos.

Na fase backward, a saída obtida é comparada à saída desejada para esse padrão particular. Se esta não estiver correta, o erro é calculado. O erro é propagado a partir da camada de saída até a camada de entrada, e os pesos das conexões das unidades das camadas internas vão sendo modificados conforme o erro é retropropagado, no caminho contrário do fluxo de sinal nas conexões sinápticas, daí o nome *backpropagation*.

Os pesos sinápticos são, então, ajustados de forma que a resposta obtida do MLP aproxime-se mais do padrão de resposta desejado.

As redes que utilizam *backpropagation* trabalham com uma variação da regra delta, apropriada para redes multi-camadas: a regra delta generalizada.

No processo de aprendizado, as redes ficam sujeitas aos problemas de procedimentos "hill-climbing", ou seja, ao problema de mínimos locais, que são pontos na superfície de erro que apresentam uma solução estável, embora não seja uma saída correta.

Uma das técnicas mais utilizadas para se evitar os mínimos locais é a adição do termo momentum, sendo simples e efetiva, na fórmula de cálculo de ajuste de pesos dos neurônios(BRAGA,2000,p.67).

Nota-se que a função *threshold* não se enquadra nesse requisito. Uma função de ativação amplamente utilizada, nestes casos, é a função *sigmoid*.

A taxa de aprendizado é uma constante de proporcionalidade no intervalo [0,1], pois este procedimento de aprendizado requer apenas que a mudança no peso seja proporcional à neta. Entretanto, o verdadeiro gradiente descendente requer que sejam tomados passos infinitesimais.

Assim quanto maior for a taxa de aprendizado, maior será a mudança nos pesos, aumentando a sua velocidade, o que pode levar à oscilações nos erros calculados. O ideal seria utilizar a maior taxa de aprendizado possível que não levasse à uma oscilação, resultando em um aprendizado mais rápido.

O treinamento das redes MLP com *backpropagation* pode demandar muitos passos no conjunto de treinamento, resultando um tempo de treinamento consideravelmente longo.

Se for encontrado um mínimo local, o erro do conjunto de treinamento pára de diminuir e estaciona em um valor maior que o aceitável.

Uma maneira de aumentar a taxa de aprendizado sem levar à oscilação é modificar a regra delta generalizada para incluir o termo *momentum*, uma variável com valor entre 0 e 1, influenciando nas mudanças dos pesos.

O termo momentum torna-se útil em espaços de erro que contenham longas gargantas, com curvas acentuadas ou vales com descidas suaves.

CAPÍTULO 2 – A FERRAMENTA MATLAB

2.1 INTRODUÇÃO AO MATLAB

O software Matlab é uma ferramenta de grande poder quando se trata de cálculos numéricos. Engloba uma vasta parte da matemática e capacidades gráficas que, para uma análise do treinamento de uma rede, é fundamental para interpretação posterior.

Grande parte de suas ferramentas encontram-se divididas em toolboxes, ou caixa de ferramentas, o que melhora a visualização das funções existentes quando há necessidade de acesso ao *Help* do programa.

Oferece informações de exemplos aplicados e demonstrados em seu Menu de Ajuda, facilitando compreensões que possivelmente seriam confusas.

Na parte de implementação gráfica possui uma toolbox própria definida por GUI, proporcionando implementações com maiores capacidades de interação entre usuário e sistema.

Através de seus recursos será desenvolvida uma aplicação voltada para a área de redes neurais, visando apresentar soluções para problemas propostos.

Para isso, é necessário um conhecimento básico das funções do programa, o qual será visto alguns dos principais que possivelmente serão usados.

2.2 CARACTERÍSTICAS BÁSICAS DA FERRAMENTA

Ao abrir o aplicativo Matlab, prontamente é aberto uma janela de comando, que representa a principal delas, a qual serão executados e registrados todos os resultados dos comandos e possíveis erros gerados.

Aparece como *prompt* de comando padrão dois sinais de maior (>>), o que fica à espera de comandos para execução.

2.2.1 A janela de comandos da ferramenta

Nela é visualizada respostas de comandos criados no programa e também se criam linhas de comandos com funções e variáveis.

Utilizam-se as teclas seta superior e inferior para maior agilidade e rapidez de repetição de comandos recentemente executados.

Para que isso seja possível, uma memória armazena todos os dados necessários quando da execução nessa janela *Command Window*. Esses dados consistem de variáveis com seus valores e funções.

Abaixo é visto as variedades possíveis do uso das teclas.

↑	retorna a linha anteriormente executada
↓	retorna a linha posteriormente executada
←	move um espaço para a esquerda
→	move um espaço para a direita
Ctrl ←	move o cursor para o início de cada comando da linha à esquerda
Ctrl →	move o cursor para o início de cada comando da linha à direita

Home	move o cursor diretamente para o começo da linha
End	move o cursor diretamente para o final da linha
Del	apaga um caracter à direita
Backspace	apaga um caracter à esquerda

2.2.2 Variáveis do software Matlab

Para a possível criação de variáveis, alguns critérios devem ser seguidos:

- são sensíveis à maiúsculas e minúsculas
- é utilizado somente os primeiros 31 caracteres para formar o nome da variável
- Nomes de variáveis devem começar com uma letra, seguido de letras, dígitos ou underscores(_)

Para a visualização das variáveis criadas e executadas nas linhas de comando, pode se executar o comando *who*, onde aparece apenas o nome ou o comando *whos,* o que resulta em informações mais detalhadas.

Algumas variáveis especiais são definidas no Matlab:

- inf representa o infinito numa definição matemática. É retornado quando ocorre uma divisão por zero

- i e j são unidades imaginárias da matemática

- realmax   retorna o maior número ponto-flutuante representado na computação, que equivale a: realmax é 1.7977e+308

- realmin   retorna o menor número ponto-flutuante representado na computação, que equivale a: realmin é 2.2251e-308

- NaN   retorna para operações numéricas com resultado não definido

- pi   é o valor de pi aproximado: 3.1416

- eps   precisão relativa de ponto-flutuante, equivalente a: 2.2204e-016

- ans   variável utilizada para armazenar valor resultante de uma expressão que não resulta em uma variável declarada

2.3 MATRIZES

O MATLAB permite a manipulação de linhas, colunas, elementos individuais e partes de matrizes.

É um recurso matemático muito importante no desenvolvimento da aplicação de redes neurais, onde se tem toda a parte de dados de entrada e saída da rede representada por matriz.

2.3.1 Gerando vetores

Os dois pontos (:) é um caracter importante no MATLAB.

A declaração **>>** *x = 1 : 5* gera um vetor linha contendo os números de 1 a 5 com incremento unitário, produzindo: *x = 1 2 3 4 5*

Outros incrementos, diferentes de um, podem ser usados.

>> J = 0:0.8:4

J = 0 0.8000 1.6000 2.4000 3.2000 4.0000

>> z = 6 : -l : l

z = 6 5 4 3 2 1

Pode-se, também, gerar vetores usando a função linspace. O exemplo abaixo gera um vetor linear seguindo a seguinte sintaxe: linspace(primeiro_valor,último_valor, número_de _elementos)

>> k = linspace (0, l, 6)

k = 0 0.2000 0.4000 0.6000 0.8000 1.0000

2.3.2 Elementos das matrizes

Um elemento individual da matriz pode ser indicado incluindo os seus subscritos entre parênteses. Por exemplo, dada a matriz N:

N =

1	*2*	*3*
4	*5*	*6*
7	*8*	*9*

A declaração: >> *N(3,3) = N(1,3) + N(3,l)* resulta na matriz abaixo, onde vemos que a posição da matriz da terceira linha com a terceira coluna recebeu o valor resultante da expressão definida.

N =

1	*2*	*3*
4	*5*	*6*
7	*8*	*10*

Temos uma outra forma de manipulação de matrizes. Por exemplo, suponha que D é uma matriz 7x5.

D=

1	*2*	*23*	*18*	*15*
3	*10*	*97*	*14*	*16*
81	*31*	*44*	*20*	*22*
99	*0*	*19*	*21*	*13*
12	*37*	*52*	*62*	*60*
40	*27*	*20*	*4*	*80*
90	*30*	*17*	*15*	*10*

O comando: >> *D(1:3,2),* especifica uma submatriz 3x1, que consiste dos três primeiros elementos da terceira coluna da matriz D.

ans =

2

10

31

Analogamente, **>>** *D(5:7,3:5),* é uma submatriz 3x3, que resulta valores cruzando-se linhas 5 a 7 e colunas 3 a 5.

ans =

52 62 60

20 4 80

17 15 10

É possível também montar uma matriz 3x4 com todos valores unitários com a expressão abaixo:

```
>> J = ones (3:4)          J =  1  1  1  1
                                1  1  1  1
                                1  1  1  1
```

Uma matriz 2x3 com valores zeros é criado abaixo.

```
>> zeros(2:3)
  ans =
       0  0  0
       0  0  0
```

Gera aleatoriamente valores para a dimensão 2 x 3 com o seguinte comando:

```
>> rand(2,3)
  ans =
       0.4860  0.7621  0.0185
       0.8913  0.4565  0.8214
```

Uma matriz identidade formada numa dimensão 4x4 é vista a seguir.

```
>> eye(4,4)
```

ans = *1 0 0 0*

0 1 0 0

0 0 1 0

0 0 0 1

2.4 MODO GRÁFICO

O Matlab disponibiliza funções para representações gráficas, em formas bidimensional e tridimensional.

Uma forma de se desenhar um gráfico 2-D na tela é usando a função plot, onde define-se valores para o esboço.

Exemplo:

```
a=[1:0.1:4] % gera valores compreendidos entre 1 e 4 espaçados entre si de 0,1.
b=sin(a); % para os diversos valores gerados, obtem os senos de cada elemento
plot(b);  %  visualização gráfica
```

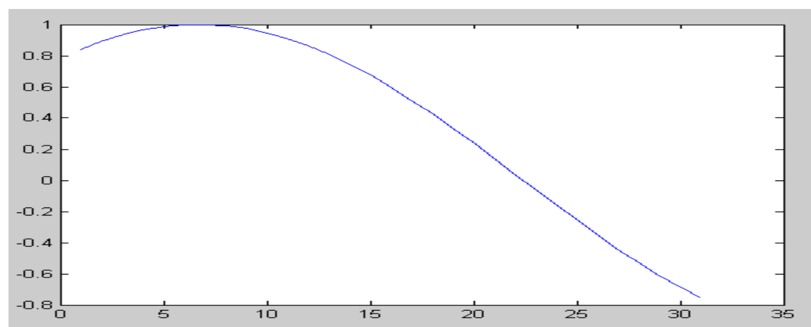


Figura 2.1 – Gráfico utilizando função plot

2.5 CRIAÇÃO DE ARQUIVOS NO MATLAB

Será visto a seguir duas formas diferentes de se armazenar dados no Matlab, cada

qual com suas respectivas características particulares.

2.5.1 Arquivos de comando

O Matlab oferece a opção de armazenamento de programas em formato de arquivos com extensão .m, acessando através do menu: *File->New->M-File*.

É apresentada uma janela para digitação das linhas de comandos, onde assim são acessados os programas desenvolvidos, sem que haja preocupação em nova digitação de linhas de código.

Utiliza a pasta *work*, que se encontra no diretório principal do programa Matlab, como caminho *default* para armazenar o arquivo gerado.

Na janela de texto, apresentam-se várias opções para o usuário, onde se destacam principalmente as de verificação passo-a-passo de erros possíveis existentes no programa(*Debug*), bem como o botão de clique rodar (*Run*), que automaticamente ativa todo o processo de compilação e verificação das linhas de código, apresentando na janela de comandos (*Command Window*) o resultado da execução.

2.5.2 Arquivos gráficos

Como mais um recurso oferecido pelo software, tem-se disponível a biblioteca gráfica através do menu: *File->New->GUI*.

Como ferramentas para utilização, dispõe-se de poucas opções, o que restringe a montagem de uma janela com interface gráfica de maior destaque.

Também faz uso da pasta *work* do diretório principal do programa, como local de armazenamento padrão, sendo assim salvo com extensão .fig.

2.6 TOOLBOX DE REDES NEURAIS

São descritas algumas funções da toolbox de redes neurais do programa Matlab.

2.6.1 Funções de criação de rede

São funções responsáveis em criar toda a arquitetura de uma rede, interligando e criando neurônios em camadas de acordo com os valores configurados para a rede.

newff: Cria uma rede com tipo de conexão feed-forward(acíclica), onde seus neurônios na camada i-ésima não podem ser usados como entrada de nodos em camadas onde seu índice seja menor ou igual a i, ou seja, sempre as saídas dos neurônios irão ser entradas de neurônios de camada superior ao seu.

Sintaxe:

net = newff(PR,[S1 S2...SNl],{TF1 TF2...TFNl},BTF,BLF,PF)

Descrição:

net = newff criará uma nova rede

newff(PR,[S1 S2...SNl],{TF1 TF2...TFNl},BTF,BLF,PF) onde,

PR: é uma matriz de R x 2 onde defino os valores máximos e mínimos para os elementos de entrada.

Si - tamanho das camadas, para N camadas.

Tfi - função de transferência da camada, default = 'tansig', que define o tipo de saída correspondente ao que se deseja na saída do neurônio, variando em –1 até 1.

BTF - função de treinamento de rede *backpropagation*, default = 'traingdx'.

BLF - função de aprendizado supervisionado para o ajuste de pesos usando do algoritmo *backpropagation*, default = 'learngdm'.

PF - função de desempenho/performance, default = 'mse'.

newp: função utilizada para criar rede de camada única.

net = newp(PR,S,TF,LF)

PR - matriz R x 2 de valores mínimo e máximo para R elementos de entrada

S - número de neurônios

TF - função de transferência, default = 'hardlim'.

LF - função de aprendizagem, default = 'learnp'.

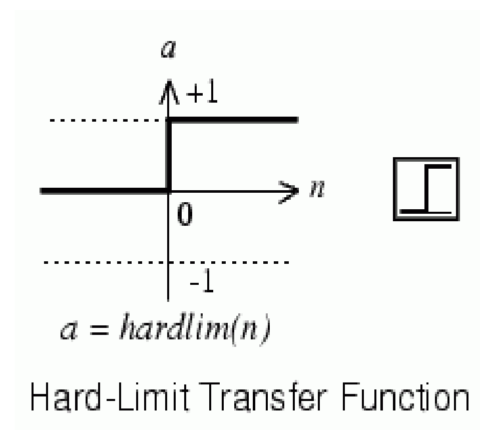
2.6.2 Funções de transferência

Estas funções são definidas para que se obtenha a saída desejada que será apresentada pela rede, de acordo com a necessidade do projeto, onde o valor da soma ponderada da saída do neurônio é aplicado à função selecionada.

À seguir, algumas das principais funções são comentadas.

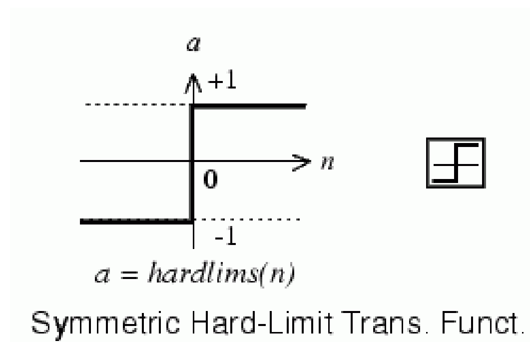
hardlim: gera saída 0 ou 1 dependendo do valor de *threshold*.

Algorítmo: função gera saída 1 se o valor n for maior ou igual a 0 e zero se n for menor que 0. Utilizado na criação de redes via comando newp(perceptron).



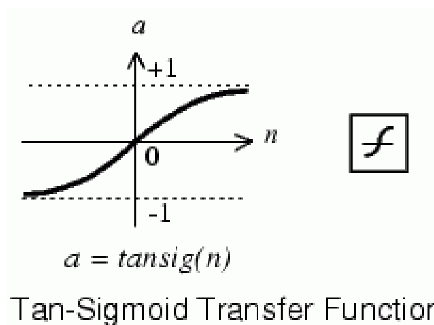
hardlims: gera saída –1 ou 1 dependendo do valor de *threshold*

Algorítmo: Esta função gera saída 1 se o valor n for maior ou igual a 0 e -1 se n for menor que 0. Utilizado na criação de redes via comando newp (perceptron).



Symmetric Hard-Limit Trans. Funct.

tansig: a saída compreenderá valores entre –1 e 1. Calcula de acordo com a fórmula: n = 2/(1+exp(-2*n))-1. Utilizado na criação de redes via comando newff ou newcf.

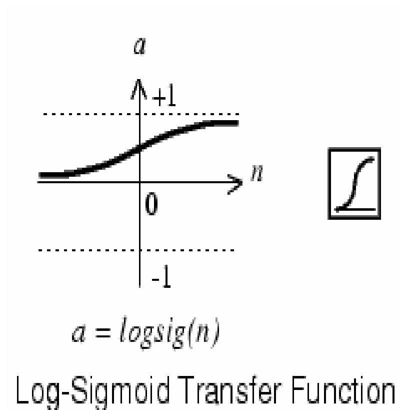


Tan-Sigmoid Transfer Function

logsig: a saída compreende valores no intervalo 0 e 1.

Algorítmo: logsig(n) = 1 / (1 + exp(-n))

Utilizado na criação de redes via comando newff ou newcf.



Log-Sigmoid Transfer Function

2.6.3 Funções de adaptações de pesos e bias

É utilizada para definir o tipo de treinamento da rede à qual será aplicada aos neurônios que fazem parte do sistema, de forma a proporcionar o ajuste dos pesos de entrada para se obter a saída desejada.

traingd: utiliza o algoritmo gradiente descendente *backpropagation*.

Utilizado na criação de redes via comando newff ,newcf e newelm.

Algorítmo: dX = lr * dperf/dX

traingdm: é uma função de treinamento de rede que atualiza os valores dos pesos pelo método do gradiente descendente usando do fator momento representado por mc, e que varia entre 0 e 1.

Utilizado na criação de redes via comando newff, newcf, e newelm.

Algorítmo: dX = mc*dXprev + lr*(1-mc)*dperf/dX

learnp: calcula os pesos alterando-os conforme o algorítmo de variável dW para os neurônios de entrada P e com erro E de acordo com a regra de aprendizado perceptron. É setado como *default* ao usar o comando newp para rede Perceptron.

Utilizado para o comando newp.
dw = 0, if e = 0 Isto pode ser resumido pela fórmula dw = e*p'.
= p', if e = 1
= -p', if e = -1

learnpn: pode resultar em um aprendizado mais veloz que o learnp de acordo com os valores de entradas propostos.

Em testes de comparação com learnp, os valores obtidos tiveram um valor com mais casas decimais e com menores valores de peso de entrada.

Utilizado para o comando newp.

Algorítmo: pn = p / sqrt(1 + p(1)^2 + p(2)^2) + ... + p(R)^2)

dw = 0, if e = 0

= pn', if e = 1 Isto pode ser resumido pela fórmula dw = e*pn'
= -pn', if e = -1

learngd: função de aprendizado para pesos usando o método do gradiente descendente.

Algorítmo: dw = lr*gW.

Aplicação: pode criar uma rede padrão que usa learngd com newff, newcf, ou newelm.

learngdm: outra função de aprendizado gradiente descendente para pesos.

Aplicação: pode criar uma rede padrão que usa learngd com newff, newcf, ou newelm.

Algorítmo: dW = mc*dWprev + (1-mc)*lr*gW

2.6.4 Função de treino da rede

Função responsável em aplicar o treinamento em uma rede neural.

train: Difere de traingd e traingdx, pois essas são funções para adaptações dos pesos da rede, cada uma com seus cálculos matemáticos.

train(NET,P,T,Pi,Ai,VV,TV)

net – rede neural

P – entradas da rede

T – saídas desejadas da rede, default = zeros.

Pi - default = zeros.

Ai - default = zeros.

VV –estrutura de vetores de validação, default = [].

TV – estrutura de vetores de teste, default = [].

Referente à parada de treinamento de uma rede, segue as seguintes condições:

- número máximo de épocas
- limite máximo de tempo
- desempenho atingido no objetivo
- o desempenho do cálculo do gradiente inferior à um_max definido
- desempenho de validação atingiu o valor de falhas máximo (quando este é utilizado).

2.6.5 Medidas de desempenho da rede

SSE – Sum Squared Error

É a soma das diferenças ocorridas na saída gerada comparando-se à entrada, elevando ao quadrado, cada diferença calculada.

$$\sum_{i=1}^{N} (e_i)^2$$

MSE – Mean Squared Error

Esse erro é calculado assim como o SSE, mas no final do resultado, multiplica-se pelo inverso do total de elementos treinados.

$$F = mse = \frac{1}{N}\sum_{i=1}^{N} (e_i)^2 = \frac{1}{N}\sum_{i=1}^{N} (t_i - a_i)^2$$

2.6.6 Variáveis de resposta de uma rede

Define-se variáveis que armazenam os valores de uma rede, seja configurando-os inicialmente ou obtendo o resultado gerado após efetivação do treinamento.

net.IW: armazena os pesos de entrada para obter a saida especificada de acordo com o *threshold*.

net.LW: armazena os pesos sinápticos das camadas intermediárias da rede

net.b: armazena o valor do disparo de saida(*threshold*).

2.6.7 Outras funções

sim: simula uma rede neural. Sintaxe: sim(net, p), onde net é a rede treinada e p as entradas desejadas para aplicar aos neurônios.

init: após treinar uma rede feedforward, os pesos e bias(*threshold*) devem ser inicializados. O comando newff automaticamente inicializa os pesos, mas pode-se reinicializar quando quiser, utilizando assim o comando init.

CAPÍTULO 3 – APLICAÇÃO PARA A BASE DE DADOS ZOO

3.1 TREINAMENTO DE UMA REDE MLP PARA A BASE DE DADOS ZOO

A base de dados conhecida como Zoo Database é de autoria de Richard Forsyth e pode ser encontrada no endereço *fttp* relacionado na referência.

Os dados contêm informações das características de cada animal, como pode ser visto na Tabela 1.

Tabela 1 - Características animais com seus respectivos valores possíveis

Atributo	Valores possíveis
1. cabelo	booleano (0 ou 1)
2. penas	booleano (0 ou 1)
3. ovos	booleano (0 ou 1)
4. leite	booleano (0 ou 1)
5. via aérea	booleano (0 ou 1)
6. aquático	booleano (0 ou 1)
7. predador	booleano (0 ou 1)
8. dentes	booleano (0 ou 1)
9. espinha dorsal	booleano (0 ou 1)
10. respira	booleano (0 ou 1)
11. venenoso	booleano (0 ou 1)
12. nadadeiras	booleano (0 ou 1)
13. pés	numéricos (ajuste dos valores: {0,2,4,5,6,8})
14. cauda	booleano (0 ou 1)
15. doméstico	booleano (0 ou 1)
16. *catsize*	booleano (0 ou 1)
17. tipo	(valores inteiros na escala [1,7])

Essas características formam uma matriz de 17 linhas por 59 colunas onde são apresentados à entrada da rede, cujos valores booleanos 0 ou 1 representam se existe ou não tal característica correspondente (ANEXO E). Uma outra matriz de valores composta de 6 linhas por 59 colunas compõe a saída desejada de resposta da rede definindo assim o animal correspondente de acordo com a característica apresentada (ANEXO F). A definição do animal é representada por combinações de valores numéricos exclusivos para cada animal.

Todos os parâmetros de configuração citados acima, incluindo valores de pesos (ANEXOS G e H) e *threshold*s (ANEXOS I e J) dos neurônios foram armazenados em arquivos constituindo uma matriz, o que foram lidos via comandos específicos do Matlab. Isso evita diferentes respostas de erro de treinamento que seriam encontradas, visto o programa definir seus valores padrões somente na primeira iteração, mudando-os constantemente nas demais.

Após o treinamento, a medida de desempenho da rede, através da função SSE, mostrará o erro resultante do treinamento da rede. Assim será possível analisar qual está sendo a diferença entre os valores gerados para a saída da rede e os que efetivamente deveriam sair.

Como forma de maior interação com as definições de parâmetros da rede e visualização fácil da resposta apresentada, janelas da biblioteca GUI foram implementadas, bastante clicar nas configurações e botões de execução desejados (ANEXOS K e L).

Características gerais da aplicação:

- Número de entradas: 17
- Número de saídas: 6
- Algorítmo de Aprendizado: *backpropagation*
- Função de ativação: sigmóide logística
- Uma camada intermediária
- Camada de saída

Variáveis da aplicação:

- Quantidade de neurônios na camada intermediária: 10, 15, 20
- Taxa de Aprendizado: 0.1, 0.01, 0.001
- Número de épocas: 100, 200, 300.

O código fonte da implementação completo está ao final (ANEXOS A,B,C, D).

3.2 ANÁLISE DE RESULTADO

Para a montagem da tabela foram realizados treinamentos diversos variando valores de taxa de aprendizado, o número de épocas e de neurônios da camada intermediária, tendo como parâmetros de respostas da rede o valor SSE, o tempo gasto de treinamento e os erros gerados no teste da rede.

Serão mostradas três tabelas em sequência com todas as configurações possíveis para cada caso específico.

Tabela 2 – Resultado obtido com dez neurônios na camada intermediária

Configuração	Camada Intermediária	Taxa de Aprendizado	Épocas	SSE Treinamento	SSE Teste	Tempo CPU (segundos)
1	10	0.1	100	3.18429	2.3022	1.182
2	10	0.1	200	2.97749	2.1397	2.073
3	10	0.1	300	2.75911	2.0158	2.995
4	10	0.01	100	3.51871	2.6149	1.151
5	10	0.01	200	3.38613	2.5148	2.164
6	10	0.01	300	3.32371	2.4581	2.974
7	10	0.001	100	6.57707	5.502	1.172
8	10	0.001	200	4.2227	3.3262	2.103
9	10	0.001	300	3.91723	2.9431	3.065

Tabela 3 – Resultado obtido com quinze neurônios na camada intermediária

Configuração	Camada Intermediária	Taxa de Aprendizado	Épocas	SSE Treinamento	SSE Teste	Tempo CPU (segundos)
10	15	0.1	100	3.20211	2.3884	1.232
11	15	0.1	200	2.91342	2.2652	2.233
12	15	0.1	300	2.78628	2.2286	3.224
13	15	0.01	100	3.63879	2.5819	1.222
14	15	0.01	200	3.45277	2.4267	2.284
15	15	0.01	300	3.35288	2.3768	3.194
16	15	0.001	100	8.83251	7.3174	1.201
17	15	0.001	200	4.74000	3.7627	2.193
18	15	0.001	300	4.32059	3.2726	3.244

Tabela 4 – Resultado obtido com vinte neurônios na camada intermediária

Configuração	Camada Intermediária	Taxa de Aprendizado	Épocas	SSE Treinamento	SSE Teste	Tempo CPU (segundos)
19	20	0.1	100	3.02456	2.2069	1.563
20	20	0.1	200	2.70105	2.0693	2.314
21	20	0.1	300	2.43716	2.0016	3.385
22	20	0.01	100	3.95365	2.6942	1.262
23	20	0.01	200	3.51435	2.4737	2.363
24	20	0.01	300	3.34063	2.3782	3.365
25	20	0.001	100	59.6755	46.891	1.351
26	20	0.001	200	10.2319	7.7026	2.384
27	20	0.001	300	6.00911	4.177	3.405

À seguir são apresentados gráficos dos treinamentos com as melhores performances obtidas.

O gráfico da Figura 3.1 corresponde à configuração 3 da Tabela 2 com dez neurônios na camada intermediária, taxa de aprendizado 0.1 e 300 épocas.

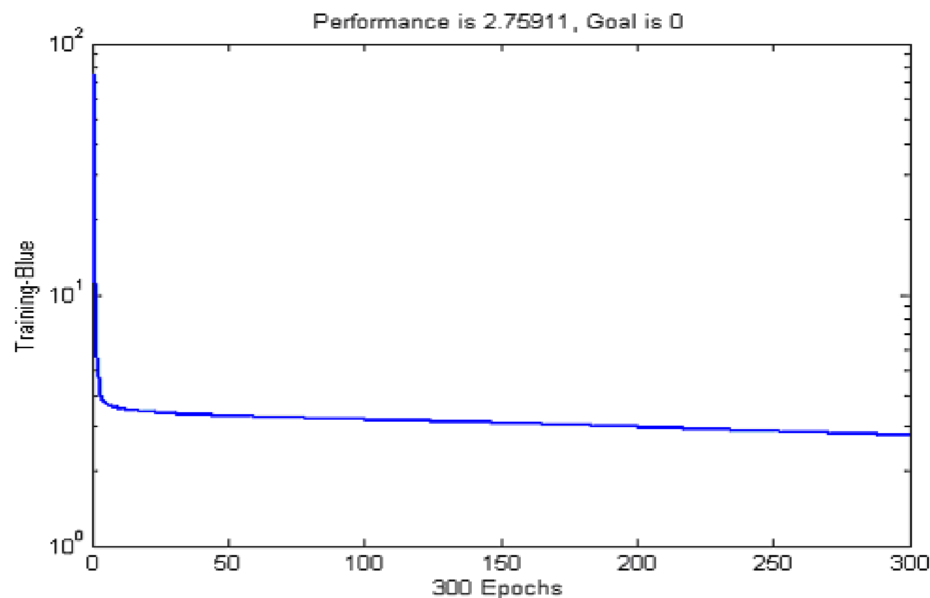


Figura 3.1 – Gráfico com dez neurônios na camada intermediária

O gráfico da Figura 3.2 corresponde à configuração 12 da Tabela 3 com quinze neurônios na camada intermediária, taxa de aprendizado 0.1 e 300 épocas.

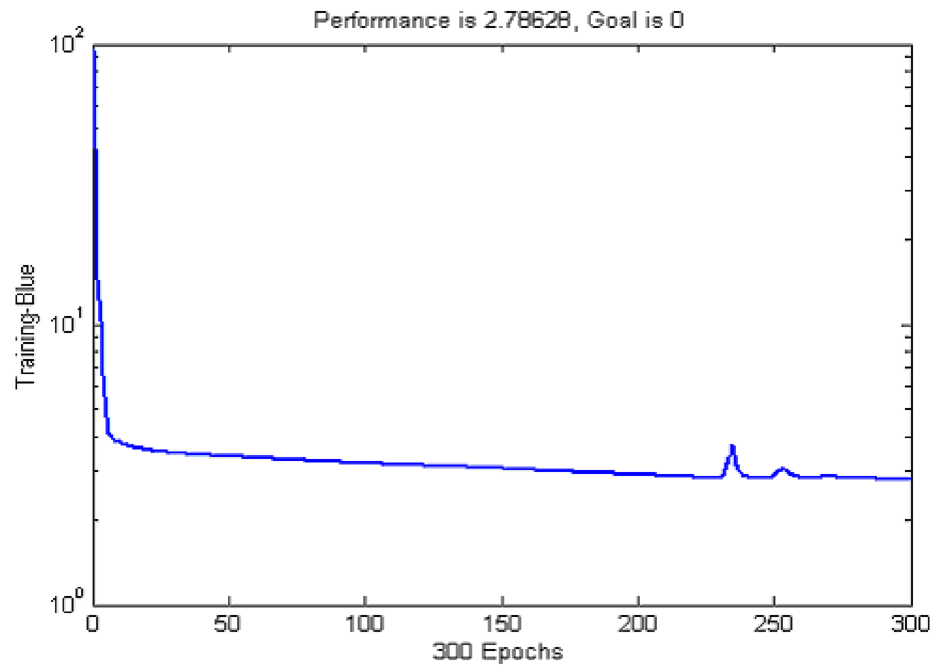


Figura 3.2 – Gráfico com quinze neurônios na camada intermediária

O gráfico da Figura 3.3 corresponde à configuração 21 da Tabela 4, com 20 neurônios na camada intermediária, taxa de aprendizado 0.1 e 300 épocas.

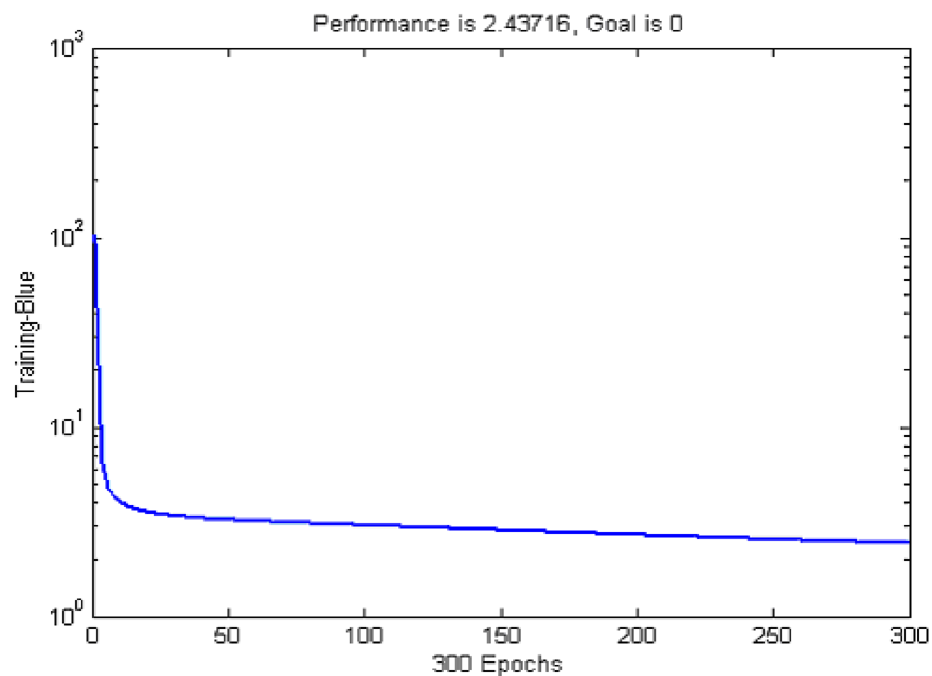


Figura 3.3 – Gráfico com vinte neurônios na camada intermediária

Analisando tabelas e gráficos aqui apresentados, a topologia que melhor desempenho obteve no treinamento foi a de vinte neurônios na camada intermediária, com taxa de aprendizado 0.1 e 300 épocas, onde o SSE atingiu o menor valor entre todos os comparados.

Através dos gráficos constata-se que a rede corresponde ao treinamento solicitado, diminuindo gradativamente o seu valor SSE de erro, e consequentemente ao SSE de teste, o que significa dizer que cada vez em que se aumenta o número de épocas, maior é o conhecimento adquirido pela rede neural, e mais próximo do valor ideal zero estará o indicativo SSE.

Quanto ao tempo gasto para o processamento do treinamento, a rede de maior número de neurônios resultou em maior tempo, visto a necessidade de mais cálculos matemáticos na adaptação dos pesos.

De forma opcional, conforme comenta Nascimento (1999, p.31), não foi apresentado vetor de validação para o treinamento da rede, o que nesta aplicação não se mostrou necessária.

Animais usados como entrada da rede estão relacionados no Anexo O e os de teste no Anexo P.

CONCLUSÃO

O trabalho desenvolvido proporcionou uma visão geral do software Matlab, com a capacidade que tem em oferecer recursos matemáticos aplicados em redes neurais através de suas funções. Foi discutida toda a parte que envolve o desenvolvimento de uma rede neural, desde o funcionamento de um neurônio artificial até a formação gerada com vários deles dispostos em camadas, de forma a adquirir conhecimentos para solução de problemas propostos. Implementado o programa para o treinamento de uma rede definida, foi possível visualizar o processo em gráficos gerados na execução. O treinamento ocorreu com a apresentação de valores padrões de entrada compostos de características animais que o definiam, sendo cada um desses diferentes entre si. Após treinada a rede, outros animais com suas características próprias não utilizados no treinamento foram à teste, resultando à qual animal colocado no treinamento correspondia em igualdades físicas e de ambiente natural. Para esta rede não foram utilizados valores de validação que pudessem informar numericamente a performance do treinamento, mas de forma gráfica, foi possível a constatação da convergência, onde a cada sequência de épocas, o valor SSE diminuía gradativamente. De acordo com a complexidade dos parâmetros configurados, a rede necessitava de mais ou menos iterações para o aprendizado obter uma melhor performance. Para a definição de bom conhecimento adquirido no treinamento, foi analisado os valores SSE’s de treinamento e consequentemente os de teste, onde se procura o menor valor desse parâmetro, que representa a menor diferença existente entre o valor desejado e o que a rede gerou em sua saída.

REFERÊNCIAS

ALVES JUNIOR, Nilton. **Redes neurais**. Disponível em: <http://mesonpi.cat.cbpf.br/naj/redesneurais.pdf >. Acesso em: 08 ago. 2005.

AZEVEDO, F. M. de; BRASIL, L. M.; OLIVEIRA, R. C. L. **Redes neurais com aplicações em controle e em sistemas especialistas**. Florianópolis: Visual Books, 2000.

BRAGA, A. de P.; LUDERMIR, T. B.; CARVALHO, A. C. P.de L. F. **Redes neurais artificiais**. Rio de Janeiro: LTC, 2000.

CARVALHO, A. P. de L. F. **Redes neurais artificiais.** Disponível em: <http://www.icmc.usp.br/~andre>. Acesso em: 10 out. 2005.

CASTRO, M. F. de. **Redes neurais na estimativa da capacidade requerida em comutadores ATM.** Disponível em: <http://www.lia.ufc.br/~miguel/docs/Castro-MCC-UFC-1999.pdf>. Acesso em: 17 set. 2005.

FORSYTH, R. **Banco de dados Zoo**. Disponível em: <fttp://ftp.ics.uci.edu/pub/machine-learning-databases>. Acesso em: 20 jul. 2005.

HANSELMAN D.; LITTLEFIELD B. **Matlab versão do estudante**. Tradução Hércules P. Neves. São Paulo: Makron Books, 1997.

NASCIMENTO, Adriano Sá. **Desenvolvendo agentes inteligentes para a gerência pró-ativa de redes ATM**. Disponível em: <http://www.mcc.ufc.br/disser/AdrianoSa.pdf>. Acesso em: 17 out. 2005.

TATIBANA, C. Y.; KAETSU, D. Y. **Introdução às redes neurais.** Disponível em: <http://www.din.uem.br/ia/neurais>. Acesso em: 20 ago. 2005.

THE MATHWORKS. Disponível em: <http://www.mathworks.com>. Acesso em: 10 set. 2005.

ANEXO A – Código fonte do programa principal

```
function varargout = Work_TCC_GUI(varargin)
% Work_TCC_GUI Application M-file for teste.fig
%  FIG = Work_TCC_GUI launch teste GUI.
%  Work_TCC_GUI('callback_name', ...) invoke the named callback.

% Last Modified by GUIDE v2.0 26-Sep-2005 23:43:32

if nargin == 0  % LAUNCH GUI

	fig = openfig(mfilename,'reuse');

	% Use system color scheme for figure:
	set(fig,'Color',get(0,'defaultUicontrolBackgroundColor'));

	% Generate a structure of handles to pass to callbacks, and store it.
	handles = guihandles(fig);
	guidata(fig, handles);

	if nargout > 0
		varargout{1} = fig;
	end

elseif ischar(varargin{1}) % INVOKE NAMED SUBFUNCTION OR CALLBACK

	try
		if (nargout)
			[varargout{1:nargout}] = feval(varargin{:}); % FEVAL switchyard
		else
			feval(varargin{:}); % FEVAL switchyard
		end
	catch
		disp(lasterr);
	end
end

function varargout = OK(h, eventdata, handles, varargin)

  % definindo variaveis globais
   global  net numneurointerm numneurosaida numentradas taxa epoch
   global pesos_transf_intermediario  pesos_transf_saida  thresh_transf_intermediario  thresh_transf_saida

   clc; % limpa janela de comando

   numentradas= 17;
   numneurosaida= 6;

   %Abrindo arquivos de pesos, threshould, valores de entrada e de saida
   dataentry = fopen('E:\SW_final_TCC\zooprog_2.txt','rt');
   dadoentrada = fscanf(dataentry,'%i',[numentradas 59])  % padroes de entrada

   dataout = fopen('E:\SW_final_TCC\result5_4.txt','rt');
   dadosaida = fscanf(dataout,'%f',[numneurosaida 59]) %padroes de saida

   databias_1 = fopen('E:\SW_final_TCC\bias_1.txt','rt');
   databias_1_1= fscanf(databias_1,'%f',[numneurointerm 1])  % valores thresholds da camada intermediaria
```

```
databias_2 = fopen('E:\SW_final_TCC\bias_2.txt','rt');
databias_2_2= fscanf(databias_2,'%f',[numneurosaida 1]) %valores thresholds da camada de saida

data_pesos_interm = fopen('E:\SW_final_TCC\pesosinterm.txt','rt');
data_pesos_interm_1 = fscanf(data_pesos_interm,'%f',[numneurointerm numentradas]); % pesos da camada
intermediaria

data_pesos_saida = fopen('E:\SW_final_TCC\pesosaida.txt','rt');
data_pesos_saida_1 = fscanf(data_pesos_saida,'%f',[numneurosaida numneurointerm])  %pesos da camada de
saida

%Fechando arquivos
fclose(dataentry);
fclose(dataout);
fclose(databias_1);
fclose(databias_2);
fclose(data_pesos_interm);
fclose(data_pesos_saida);

% criando a rede do tipo backpropagation
net = newff ([0 1 ;0 1 ;0 1 ;0 1 ;0 1 ;0 1 ;0 1 ;0 1 ;0 1 ;0 1 ;0 1 ;0 1 ;0 8 ;0 1 ;0 1 ;0 1 ;1 7],[numneurointerm
numneurosaida],{'logsig','logsig'},'traingd','learngd','sse');
% 17 definiçoes 0 7 ;... indicando as 17 entradas com o min e max de cada uma
p = dadoentrada; % valores de entrada
t = dadosaida;   % valores desejados de saida

%Inicializando valores de threshould e pesos dos neuronios da rede
net.b{1}=databias_1_1;     % inicializando as bias da camada intermediaria
net.b{2}= databias_2_2;    % inicializando as bias da camada de saida
net.IW{1,1}= data_pesos_interm_1;  % inicializando pesos da camada intermediaria
net.LW{2,1}= data_pesos_saida_1;    % inicializando pesos da camada de saida

%mostrando os valores inicializados
threshould_interm = net.b{1}
threshoul_saida   = net.b{2}
pesos_interm      = net.IW{1,1}
pesos_saida       = net.LW{2,1}

%Parametros de configuraçao da rede
net.trainParam.epochs= epoch; %numero de epocas maxima
net.trainParam.lr =     taxa;        % valor da taxa de aprendizado
net.trainParam.min_grad =  0;   % define o criterio de gradiente minimo
net.trainParam.show=      50;    % define a frequencia com que as epocas serao mostradas graficamente
                                 % Ajustando para NaN, nao me mostra o grafico do treinamento

%Treinando a rede
tic; %inicia contagem de tempo
net = train(net,p,t);
toc; % para a contagem e mostra o tempo gasto no treinamento;

%Resultados da Rede, armazenando em variaveis os valores de pesos e thresholds resultantes do treinamento
pesos_final_intermediario   =   net.IW{1,1};
pesos_final_saida = net.LW{2,1};
treshould_final_intermediario =      net.b{1};
treshould_final_saida =      net.b{2};
taxa_de_aprendizado_final= net.trainParam.lr

% simulando a rede para saber se resulta corretamente o esperado
simulation  =      sim(net,p);
```

```
  %transformando numero em string para serem mostrados na janela de pesos_bias
   pesos_transf_intermediario = num2str(pesos_final_intermediario);
   pesos_transf_saida = num2str(pesos_final_saida);
   thresh_transf_intermediario = num2str(treshould_final_intermediario);
   thresh_transf_saida = num2str(treshould_final_saida);

   set(handles.edit_erro,'String',' '); %limpando o campo ERRO toda vez que treina a rede

function varargout = SAIR(h, eventdata, handles, varargin)
   closereq   % se aqui usar quit, fecha ate o Matlab

function varargout = bottom_autores_Callback(h, eventdata, handles, varargin)
   openfig('box_autores','reuse'); % abrindo a janela autor


function varargout = ver_pesos_bias(h, eventdata, handles, varargin)
   global pesos_transf_intermediario  pesos_transf_saida  thresh_transf_intermediario  thresh_transf_saida

   openfig('pesos_bias','reuse'); % abrindo a janela para mostrar pesos e valores thresholds finais

function varargout = bottom_help_Callback(h, eventdata, handles, varargin)
   openfig('Box_Help','reuse'); % abrindo a janela de ajuda de uso do programa

%funçoes abaixo criadas para pegar o valor do item selecionado do numero de neuronios da primeira camada

function varargout = popneuronios(h, eventdata, handles, varargin)
   global numneurointerm  % para definir o valor globamente da variavel, se nao, nao funciona
   val = get(h,'Value')  %esse Value define qual valor de indice do array foi selecionado
   lista_da_string_cell = get(h,'String')
   numneurointerm = str2num(lista_da_string_cell{val}) %pega o valor em string e transforma em numero

function varargout = popaprendizado(h, eventdata, handles, varargin)
   global  taxa % para definir o valor globamente da variavel, se nao, nao funciona
   val = get(h,'Value')  %esse Value define qual valor de indice do array selecionei
   lista_da_string_cell = get(h,'String')
   taxa = str2num(lista_da_string_cell{val})

function varargout = popepoca(h, eventdata, handles, varargin)
   global epoch  % para definir o valor globamente da variavel, se nao, nao funciona
   val = get(h,'Value')  %esse Value define qual valor de indice do array selecionei
   lista_da_string_cell = get(h,'String') %transforma de cell para string
   epoch = str2num(lista_da_string_cell{val}) %transforma de string para numero


%essa funçao abaixo e  inicializado as variaveis com os valores default do menu usando somente
%o CREATEFCN do FRAME principal, o que poderia ser qualquer outro.
%Isso evita com que o usuario, nao clicando direto no treinar, nao de indefiniçao de variaveis e de erro.
function varargout =  values_default(h, eventdata, handles, varargin)
   clc; % limpa a janela de comando
   clear; % limpa a memoria
   global numneurointerm taxa epoch
   numneurointerm=10;
   taxa=0.1;
   epoch=100;

   function varargout = botao_teste(h, eventdata, handles, varargin)
   global net numentradas numneurosaida
   clc;
   num_valores_teste=42
```

```
data_entry_teste = fopen('E:\SW_final_TCC\teste_rna.txt','rt');
dado_entrada_teste = fscanf(data_entry_teste,'%i',[numentradas num_valores_teste])  % padroes de entrada de teste
data_result_teste = fopen('E:\SW_final_TCC\result_teste_rna.txt','rt');
dado_esperado_teste = fscanf(data_result_teste,'%f',[numneurosaida num_valores_teste])  % valores esperados de saida

simulation=sim(net,dado_entrada_teste)  % aplicando entradas a rede e obtendo a saida
erro= dado_esperado_teste-simulation % calculando a diferença do resultado
sse_teste=sse(erro)  %calculando o erro sse
sse_string=num2str(sse_teste)  %transformando numero em string

set(handles.edit_erro,'String',sse_string); %mostrando no campo ERRO da janela principal
```

ANEXO B – Código fonte da janela de amostragem de pesos e bias

```
function varargout = pesos_bias(varargin)
% PESOS_BIAS Application M-file for pesos_bias.fig
%    FIG = PESOS_BIAS launch pesos_bias GUI.
%    PESOS_BIAS('callback_name', ...) invoke the named callback.

global pesos_transf
if nargin == 0  % LAUNCH GUI

	fig = openfig(mfilename,'reuse');

	% Use system color scheme for figure:
	set(fig,'Color',get(0,'defaultUicontrolBackgroundColor'));

	% Generate a structure of handles to pass to callbacks, and store it.
	handles = guihandles(fig);
	guidata(fig, handles);

	if nargout > 0
		varargout{1} = fig;
	end

elseif ischar(varargin{1}) % INVOKE NAMED SUBFUNCTION OR CALLBACK

	try
		if (nargout)
			[varargout{1:nargout}] = feval(varargin{:}); % FEVAL switchyard
		else
			feval(varargin{:}); % FEVAL switchyard
		end
	catch
		disp(lasterr);
	end

end

function varargout = botao_pesos_interm(h, eventdata, handles, varargin)
   global pesos_transf_intermediario
    set(handles.listbox_pesos_bias,'String', pesos_transf_intermediario );

function varargout = botao_pesos_saida(h, eventdata, handles, varargin)
   global pesos_transf_saida
   set(handles.listbox_pesos_bias,'String', pesos_transf_saida );

function varargout = botao_bias_interm(h, eventdata, handles, varargin)
   global thresh_transf_intermediario
   set(handles.listbox_pesos_bias,'String', thresh_transf_intermediario );

function varargout = botao_bias_saida(h, eventdata, handles, varargin)
    global thresh_transf_saida
   set(handles.listbox_pesos_bias,'String', thresh_transf_saida );

function varargout = botao_sair(h, eventdata, handles, varargin)
   closereq;
```

ANEXO C – Código fonte da janela autor

```
function varargout = box_autores(varargin)
% BOX_AUTORES Application M-file for box_autores.fig
%    FIG = BOX_AUTORES launch box_autores GUI.
%    BOX_AUTORES('callback_name', ...) invoke the named callback.


if nargin == 0  % LAUNCH GUI

	fig = openfig(mfilename,'reuse');

	% Use system color scheme for figure:
	set(fig,'Color',get(0,'defaultUicontrolBackgroundColor'));

	% Generate a structure of handles to pass to callbacks, and store it.
	handles = guihandles(fig);
	guidata(fig, handles);

	if nargout > 0
		varargout{1} = fig;
	end

elseif ischar(varargin{1}) % INVOKE NAMED SUBFUNCTION OR CALLBACK

	try
		if (nargout)
			[varargout{1:nargout}] = feval(varargin{:}); % FEVAL switchyard
		else
			feval(varargin{:}); % FEVAL switchyard
		end
	catch
		disp(lasterr);
	end

end
```

ANEXO D – Código fonte da janela Help

```
% BOX_HELP Application M-file for Box_Help.fig

function varargout = Box_Help(varargin)
if nargin == 0  % LAUNCH GUI

	fig = openfig(mfilename,'reuse');

	% Use system color scheme for figure:
	set(fig,'Color',get(0,'defaultUicontrolBackgroundColor'));

	% Generate a structure of handles to pass to callbacks, and store it.
	handles = guihandles(fig);
	guidata(fig, handles);

	if nargout > 0
		varargout{1} = fig;
	end

elseif ischar(varargin{1})

	try
		if (nargout)
			[varargout{1:nargout}] = feval(varargin{:});
		else
			feval(varargin{:});
		end
	catch
		disp(lasterr);
	end
end
```

ANEXO E – Valores de entrada da rede para treinamento

1 0 0 1 0 0 1 1 1 1 0 0 4 0 0 1 1
1 0 0 1 0 0 0 1 1 1 0 0 4 1 0 1 1
0 0 1 0 0 1 1 1 1 0 0 1 0 1 0 0 4
1 0 0 1 0 0 1 1 1 1 0 0 4 1 0 1 1
1 0 0 1 0 0 0 1 1 1 0 0 4 1 1 1 1
0 0 1 0 0 1 0 1 1 0 0 1 0 1 1 0 4
1 0 0 1 0 0 0 1 1 1 0 0 4 0 1 0 1
0 1 1 0 1 0 0 0 1 1 0 0 2 1 1 0 2
0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 7
0 0 1 0 0 1 1 0 0 0 0 0 4 0 0 0 7

0 0 1 0 0 1 1 0 0 0 0 0 6 0 0 0 7
0 1 1 0 1 0 1 0 1 1 0 0 2 1 0 0 2
0 0 1 0 0 1 1 1 1 0 0 1 0 1 0 1 4
0 0 0 1 0 1 1 1 1 1 0 1 0 1 0 1 1
0 1 1 0 1 1 0 0 1 1 0 0 2 1 0 0 2
0 1 1 0 1 0 0 0 1 1 0 0 2 1 0 1 2
0 0 1 0 0 0 0 0 0 1 0 0 6 0 0 0 6
0 0 1 0 0 1 1 1 1 1 0 0 4 0 0 0 5
0 0 1 0 0 1 1 1 1 1 1 0 4 0 0 0 5
1 0 0 1 1 0 0 1 1 1 0 0 2 1 0 0 1

1 0 0 1 0 0 1 1 1 1 0 0 2 0 1 1 1
0 0 1 0 1 0 0 0 0 1 0 0 6 0 0 0 6
1 0 0 1 0 0 0 1 1 1 0 0 2 0 0 1 1
0 1 1 0 1 1 1 0 1 1 0 0 2 1 0 0 2
0 0 1 0 0 1 0 1 1 0 0 1 0 1 0 0 4
1 0 0 1 0 0 0 1 1 1 0 0 4 1 1 0 1
1 0 0 1 0 0 0 1 1 1 0 0 4 1 0 0 1
1 0 1 0 1 0 0 0 0 1 1 0 6 0 1 0 6
1 0 1 0 1 0 0 0 0 1 0 0 6 0 0 0 6
0 1 1 0 0 0 1 0 1 1 0 0 2 1 0 0 2

0 0 1 0 1 0 1 0 0 1 0 0 6 0 0 0 6
0 1 1 0 1 0 0 0 1 1 0 0 2 1 0 0 2
1 0 0 1 0 1 1 1 1 1 0 0 4 1 0 1 1
1 0 0 1 0 0 1 1 1 1 0 0 4 1 0 0 1
0 0 1 0 0 1 1 1 1 1 0 0 4 1 0 0 5
0 0 1 0 0 1 1 0 0 0 0 0 8 0 0 1 7
0 1 1 0 0 0 0 0 1 1 0 0 2 1 0 1 2
0 1 1 0 0 1 1 0 1 1 0 0 2 1 0 1 2
0 0 1 0 0 0 1 1 1 1 1 0 0 1 0 0 3
1 0 1 1 0 1 1 0 1 1 0 0 4 1 0 1 1

1 0 0 1 0 0 1 1 1 1 0 0 4 1 1 1 1
0 1 1 0 0 0 1 0 1 1 0 0 2 1 0 1 2

0 0 0 0 0 0 1 0 0 1 1 0 8 1 0 0 7
1 0 0 1 0 1 1 1 1 1 0 1 0 0 0 1 1
1 0 0 1 0 1 1 1 1 1 0 1 2 1 0 1 1
0 0 0 0 0 1 1 1 1 0 1 0 0 1 0 0 3
0 0 1 0 0 1 1 0 0 0 1 0 0 0 0 0 7
0 0 1 0 0 0 1 1 1 1 0 0 0 1 0 0 3
0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 7
1 0 0 1 0 0 0 1 1 1 0 0 2 1 0 0 1

0 0 1 0 0 1 1 0 0 0 0 0 5 0 0 0 7
0 0 1 0 0 1 1 1 1 0 1 1 0 1 0 1 4
0 1 1 0 1 1 0 0 1 1 0 0 2 1 0 1 2
0 0 1 0 0 1 0 1 1 1 0 0 4 0 0 0 5
0 0 1 0 0 0 0 0 1 1 0 0 4 1 0 1 3
0 0 1 0 0 0 1 1 1 1 0 0 4 1 0 0 3
0 1 1 0 1 0 1 0 1 1 0 0 2 1 0 1 2
1 0 0 1 0 0 0 1 1 1 0 0 2 1 0 1 1
1 0 1 0 1 0 0 0 0 1 1 0 6 0 0 0 6

ANEXO F – Valores de saída desejados para a rede no treinamento

0.4 0.4 0.4 0.2 0.4 0.2
0.4 0.4 0.4 0.2 0.2 0.4
0.4 0.4 0.4 0.2 0.2 0.2
0.4 0.4 0.2 0.4 0.4 0.4
0.4 0.4 0.2 0.4 0.4 0.2
0.4 0.4 0.2 0.4 0.2 0.4
0.4 0.4 0.2 0.4 0.2 0.2
0.4 0.4 0.2 0.2 0.4 0.4
0.4 0.4 0.2 0.2 0.4 0.2
0.4 0.4 0.2 0.2 0.2 0.4

0.4 0.4 0.2 0.2 0.2 0.2
0.4 0.2 0.4 0.4 0.4 0.4
0.4 0.2 0.4 0.4 0.4 0.2
0.4 0.2 0.4 0.4 0.2 0.4
0.4 0.2 0.4 0.4 0.2 0.2
0.4 0.2 0.4 0.2 0.4 0.4
0.4 0.2 0.4 0.2 0.4 0.2
0.4 0.2 0.4 0.2 0.2 0.4
0.4 0.2 0.4 0.2 0.2 0.2
0.4 0.2 0.2 0.4 0.4 0.4

0.4 0.2 0.2 0.4 0.4 0.2
0.4 0.2 0.2 0.4 0.2 0.4
0.4 0.2 0.2 0.4 0.2 0.2
0.4 0.2 0.2 0.2 0.4 0.4
0.4 0.2 0.2 0.2 0.4 0.2
0.4 0.2 0.2 0.2 0.2 0.4
0.4 0.2 0.2 0.2 0.2 0.2
0.2 0.4 0.4 0.4 0.4 0.4
0.2 0.4 0.4 0.4 0.4 0.2
0.2 0.4 0.4 0.4 0.2 0.4

0.2 0.4 0.4 0.4 0.2 0.2
0.2 0.4 0.4 0.2 0.4 0.4
0.2 0.4 0.4 0.2 0.4 0.2
0.2 0.4 0.4 0.2 0.2 0.4
0.2 0.4 0.4 0.2 0.2 0.2
0.2 0.4 0.2 0.4 0.4 0.4
0.2 0.4 0.2 0.4 0.4 0.2
0.2 0.4 0.2 0.4 0.2 0.4
0.2 0.4 0.2 0.4 0.2 0.2
0.2 0.4 0.2 0.2 0.4 0.4

0.2 0.4 0.2 0.2 0.4 0.2
0.2 0.4 0.2 0.2 0.2 0.4

0.2 0.4 0.2 0.2 0.2 0.2
0.2 0.2 0.4 0.4 0.4 0.4
0.2 0.2 0.4 0.4 0.4 0.2
0.2 0.2 0.4 0.4 0.2 0.4
0.2 0.2 0.4 0.4 0.2 0.2
0.2 0.2 0.4 0.2 0.4 0.4
0.2 0.2 0.4 0.2 0.4 0.2
0.2 0.2 0.4 0.2 0.2 0.4

0.2 0.2 0.4 0.2 0.2 0.2
0.2 0.2 0.2 0.4 0.4 0.4
0.2 0.2 0.2 0.4 0.4 0.2
0.2 0.2 0.2 0.4 0.2 0.4
0.2 0.2 0.2 0.4 0.2 0.2
0.2 0.2 0.2 0.2 0.4 0.4
0.2 0.2 0.2 0.2 0.4 0.2
0.2 0.2 0.2 0.2 0.2 0.4
0.2 0.2 0.2 0.2 0.2 0.2

ANEXO G – Pesos iniciais da camada intermediária

0.9798 -0.5396 0.1645 -0.3484 0.1446 -0.5244 0.0652 -0.0573 0.6974 -0.7136
0.9397 -0.9447 0.0193 -0.3716 0.6395 -0.2853 0.7029 -0.1456 0.0771 -0.5488
-0.2812 0.0252 -0.5397 0.3973 -0.3837 0.4907 -0.6677 0.4075 -0.1800 -0.2499
0.0837 -0.1039 0.9921 -0.4975 0.3020 -0.2860 0.7784 -0.6540 0.2089 -0.1913
0.6203 -0.7668 0.3490 -0.4563 0.0178 -0.8632 0.5254 0.3429 -0.5446 -0.8379
0.3393 -0.6298 0.2066 -0.3019 0.4016 -0.6162 0.2989 -0.8588 0.4249 -0.4412
-0.6985 0.6941 -0.0677 0.1850 -0.6277 0.2281 -0.3482 0.5889 -0.0507 -0.5470
0.9894 -0.1084 0.4963 -0.0994 0.5444 -0.5469 0.7721 -0.2870 0.2374 -0.5084
-0.1437 0.3137 -0.8964 0.4045 -0.2155 0.3292 -0.2137 0.5934 -0.1950 -0.0918
0.6953 0.8346 -0.1782 0.4809 -0.0497 0.0389 -0.3352 0.5934 -0.1950 -0.2609
0.8410 0.7072 -0.9473 0.3142 -0.8524 0.8578 0.2568 -0.9156 0.9893 0.2089
0.1473 -0.3390 0.9231 -0.2744 0.8665 -0.6804 -0.6077 0.1637 -0.0374 -0.6277
0.4755 0.6701 -0.3539 0.3105 -0.6317 0.1654 0.3233 -0.3009 0.9892 -0.6554
0.6141 -0.0204 0.0533 -0.3125 0.6345 -0.2974 0.0263 -0.1286 -0.3352 0.5934
0.9012 0.4596 -0.2988 0.2806 -0.8609 0.5498 0.1721 -0.8668 0.4016 -0.6162
0.9892 -0.6554 0.4764 -0.4077 0.3473 -0.1685 -0.5630 0.1024 -0.2812 0.0252
0.3043 0.7733 -0.3338 0.0929 -0.0582 0.7449 0.2489 -0.3502 -0.5446 -0.8379
0.0008 -0.3732 -0.7056 0.7445 0.2732 -0.5049 -0.0527 0.5899 0.7449 0.2489
0.5020 0.1208 -0.6657 0.7282 -0.3689 0.9722 0.9893 -0.2609 -0.6804 -0.6077
-0.2812 0.0252 -0.5397 0.3973 -0.3837 0.4907 -0.6677 0.4075 -0.1800 -0.2496

0.9798 -0.5396 0.1645 -0.3484 0.1446 -0.5244 0.0652
0.9397 -0.9447 0.0193 -0.3716 0.6395 -0.2853 0.7029
-0.2812 0.0252 -0.5397 0.3973 -0.3837 0.4907 -0.6677
0.0837 -0.1039 0.9921 -0.4975 0.3020 -0.2860 0.7784
0.6203 -0.7668 0.3490 -0.4563 0.0178 -0.8632 0.5254
0.3393 -0.6298 0.2066 -0.3019 0.4016 -0.6162 0.2989
-0.6985 0.6941 -0.0677 0.1850 -0.6277 0.2281 -0.3482
0.9894 -0.1084 0.4963 -0.0994 0.5444 -0.5469 0.7721
-0.1437 0.3137 -0.8964 0.4045 -0.2155 0.3292 -0.2137
0.6953 0.8346 -0.1782 0.4809 -0.0497 0.0389 -0.3352
0.8410 0.7072 -0.9473 0.3142 -0.8524 0.8578 0.2568
0.1473 -0.3390 0.9231 -0.2744 0.8665 -0.6804 -0.6077
0.4755 0.6701 -0.3539 0.3105 -0.6317 0.1654 0.3233
0.6141 -0.0204 0.0533 -0.3125 0.6345 -0.2974 0.0263
0.9012 0.4596 -0.2988 0.2806 -0.8609 0.5498 0.1721
0.9892 -0.6554 0.4764 -0.4077 0.3473 -0.1685 -0.5630
0.3043 0.7733 -0.3338 0.0929 -0.0582 0.7449 0.2489
0.0008 -0.3732 -0.7056 0.7445 0.2732 -0.5049 -0.0527
0.5020 0.1208 -0.6657 0.7282 -0.3689 0.9722 0.9893
-0.2812 0.0252 -0.5397 0.3973 -0.3837 0.4907 -0.6677

ANEXO H – Pesos iniciais da camada de saída

-0.3597	0.5410	-0.8897	0.9061	-0.9913	0.1771	-0.6499	0.6318	-0.5605	0.1556
-0.6151	0.4100	-0.4612	0.5746	-0.9209	0.6316	-0.0594	0.0598	-0.5601	0.4934
-0.1205	0.9681	-0.3850	0.9259	-0.0394	0.7523	-0.2874	0.9945	-0.0821	0.8327
-0.2168	0.4268	-0.0609	0.1444	-0.4804	0.9335	-0.6921	0.5652	-0.8867	0.0558
-0.6869	0.1319	-0.2047	0.6466	-0.8448	0.5520	-0.6138	0.9908	-0.0017	0.7306
-0.8639	0.0226	-0.2145	0.9129	-0.5657	0.7692	-0.0594	0.5458	-0.0003	0.4809

-0.3597	0.5410	-0.8897	0.9061	-0.9913	0.1771	-0.6499	0.6318	-0.5605	0.1556
-0.6151	0.4100	-0.4612	0.5746	-0.9209	0.6316	-0.0594	0.0598	-0.5601	0.4934
-0.1205	0.9681	-0.3850	0.9259	-0.0394	0.7523	-0.2874	0.9945	-0.0821	0.8327
-0.2168	0.4268	-0.0609	0.1444	-0.4804	0.9335	-0.6921	0.5652	-0.8867	0.0558
-0.6869	0.1319	-0.2047	0.6466	-0.8448	0.5520	-0.6138	0.9908	-0.0017	0.7306
-0.8639	0.0226	-0.2145	0.9129	-0.5657	0.7692	-0.0594	0.5458	-0.0003	0.4809

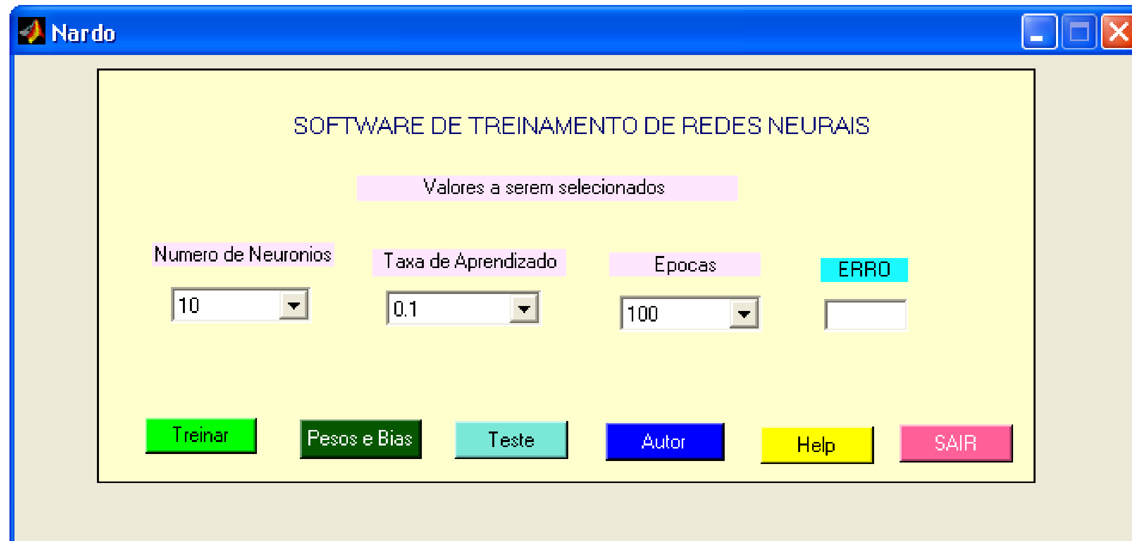
ANEXO I – Thresholds iniciais da camada intermediária

-0.9723
-0.0149
-0.8668
-0.9837
-0.5120
-0.5518
-0.0489
-0.0770
-0.8263
-0.8724
-0.2660
-0.3266
-0.6829
-0.6400
-0.8180
-0.4552
-0.1689
-0.1993
-0.7046
-0.2994

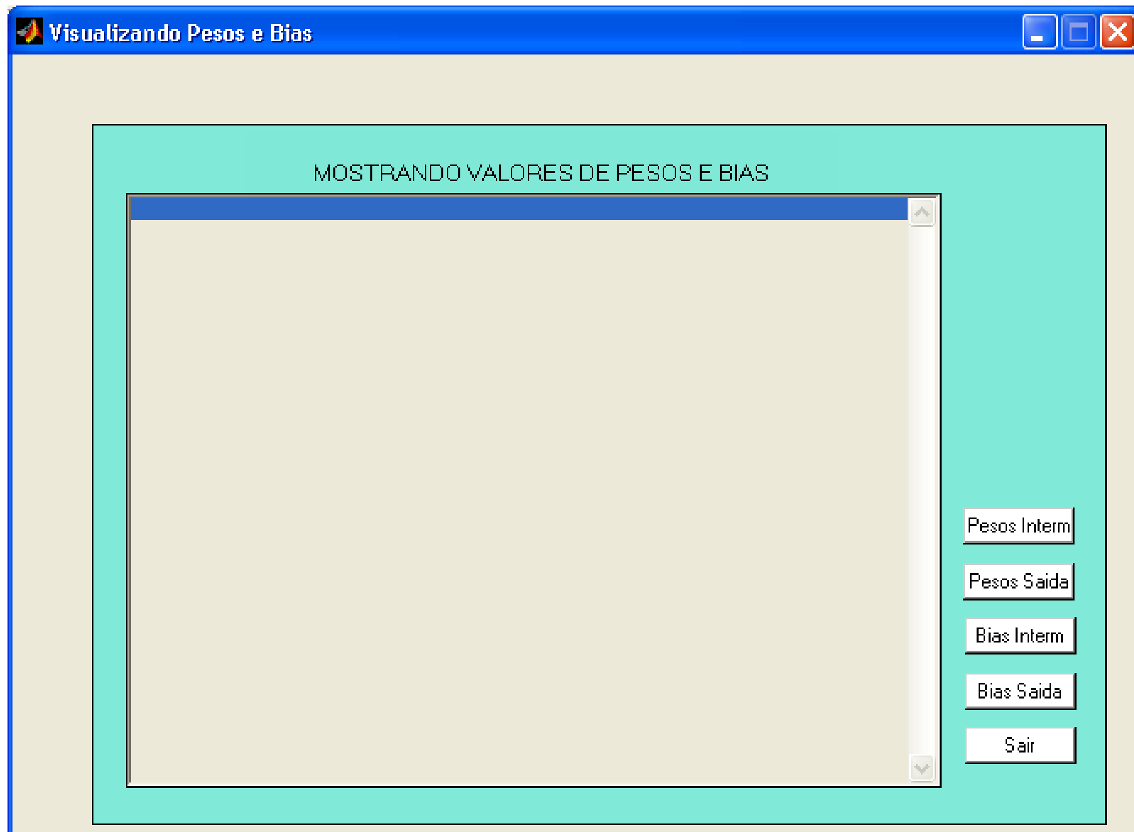
ANEXO J – Thresholds iniciais da camada de saída

-0.3809
-0.2312
-0.2476
-0.4132
-0.2824
-0.7364

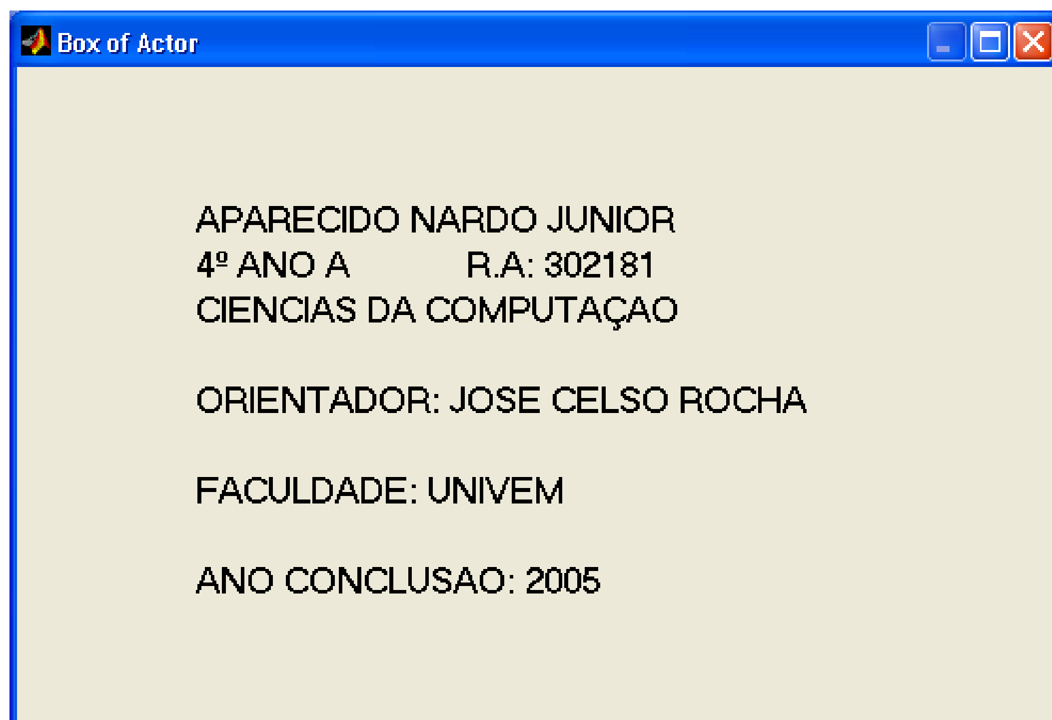
ANEXO K – Janela principal para treinamento da rede



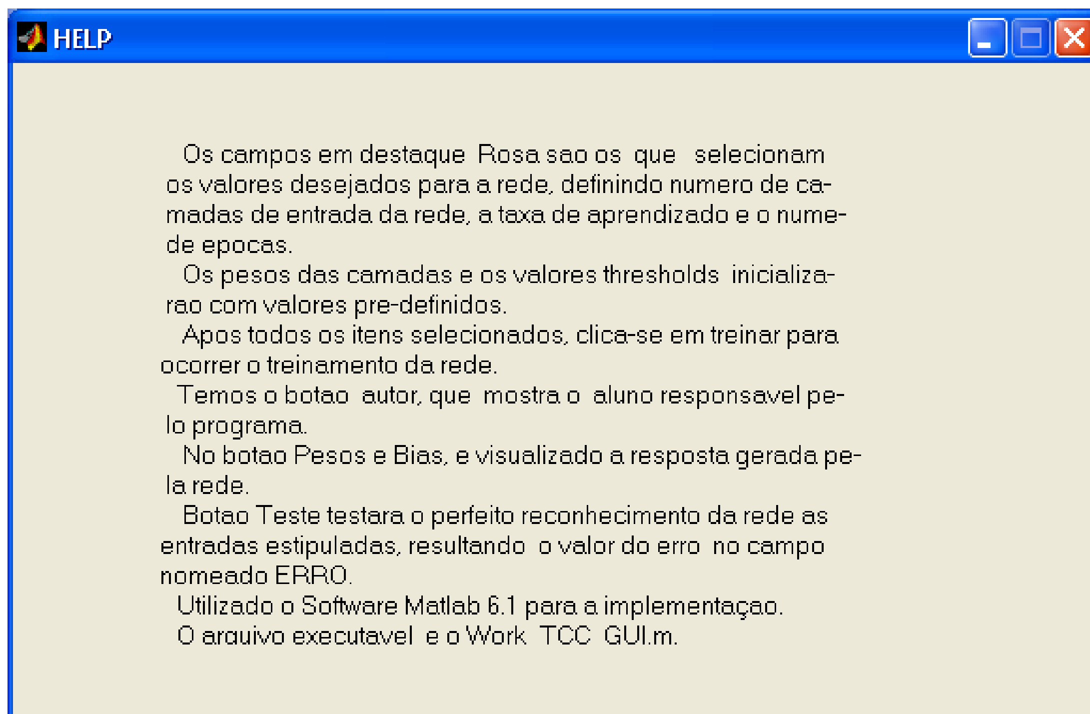
ANEXO L – Janela mostrando valores dos pesos e bias(thresholds) da rede



ANEXO M – Janela de apresentação do autor



ANEXO N – Janela de Ajuda do programa - Help



ANEXO O – Animais colocados em treinamento na rede

*aardvark, antelope, bass, boar, calf, carp, cavy, chicken, clam, crab, crayfish, crow, dogfish, dolphin, duck, flamingo, flea, frog(*não–venenoso*), frog(*venenoso*), fruitbat, girl, gnat, gorilla, gull, haddock, hamster, hare, honeybee, housefly, kiwy, ladybird, lark, mink, mole, newt, octopus, ostrich, penguim, pitviper, platypus, pussycat, rhea, scorpion, seal, sealion, seasnake, seawasp, slowworm, slug, squirrel, starfish, stingray, swan, toad, tortoise, tuatara, vulture, wallaby, wasp.*

ANEXO P – Animais colocados em teste na rede

bear, bufalo, deer, elephant, giraffe, oryx, catfish, chub, herring, piranha, cheetah, leopard, lion, lynx, mongoose, polecat, puma, raccoon, wolf, goat, pony, reindeer, dove, parakeet, lobster, hawk, pike, tuna, porpoise, termite, vampire, skimmer, skua, seahorse, sole, vole, moth, pheasant, sparrow, wren, opossum, worm.