Centro Universitário de Brasília – UniCEUB
Faculdade de Ciências Exatas e de Tecnologia – FAET
Curso de Engenharia de Computação
Disciplina: Projeto Final
Professor Orientador: Profº. Aderlon Marcelino Queiroz

Projeto do Controle de Um Móvel Utilizando Redes Neurais Artificiais

EDUARDO BRAGA DUTRA ROCHA
REGISTRO ACADÊMICO: 2011466/3

Brasília-DF, Dezembro de 2006

CENTRO UNIVERSITÁRIO DE BRASÍLIA – UniCEUB

Faculdade de Ciências Exatas – FAET

Engenharia de Computação

Projeto do Controle de Um Móvel Utilizando Redes Neurais Artificiais

EDUARDO BRAGA DUTRA ROCHA

REGISTRO ACADÊMICO: 2011466/3

Trabalho apresentado à Banca Examinadora da Faculdade de Ciências Exatas e Tecnologia - UniCEUB, como requisito para conclusão do curso de Engenharia de Computação.

Brasília-DF, Dezembro de 2006

EDUARDO BRAGA DUTRA ROCHA

Projeto do Controle de Um Móvel Utilizando Redes Neurais Artificiais

Monografia apresentada ao Centro Universitário de Brasília – UniCEUB como um dos pré-requisitos para obtenção do título de Bacharel em Engenharia da Computação.

Membros da Banca Examinadora

Membros da Banca	Assinatura
MENÇÃO FINAL:	

Brasília-DF, Dezembro de 2006.

"Não há maior prova de ignorância do que acreditar que o inexplicável é impossível."

S. Bilard

Agradecimentos

Primeiramente, agradeço ao Bom Senhor Deus, pois me acompanha e me guia desde pequenininho. Agradeço a todos que contribuíram para a realização deste trabalho, em especial:

à minha família, paciente e compreensiva: Ari, Daniela, Genaro, Nadir e Socorro, pois sempre me forneceram o carinho e apoio necessário e lutaram sempre por mim. Impossível ter realizado meu trabalho sem eles.

à minha namorada, Vanessa, a pessoa mais carinhosa de todas: me forneceu toda paciência, carinho e amor sincero que se fez necessário. Deu-me, também, condições físicas e psicológicas para terminar meu trabalho.

ao Vasco, que ficou no meu pé esse tempo todo, ajudando-me a concluir o trabalho.

aos amigos que me acompanharam durante o curso: Italo Bruno, Fernando do Prado, Ana Paula e Jonatas de Oliveira. Em muitas ocasiões compartilharam comigo bens valiosíssimos como o conhecimento, alegria, oportunidades e é claro, a amizade. Ana: obrigado pelo caramelo (a borracha) nos momentos necessários. Brunão: sentirei saudades dos lanches no morte-lenta.

ao amigo Cláudio Umpierre: Obrigado pela amizade e pela vontade de vencer. E obrigado por ter me acompanhado no inicio de minha monitoria de suporte.

ao professor Luiz Otávio, que arquitetou a monitoria de Redes e influenciou fortemente meu interesse pela área.

ao professor Gonçalves, companheiro e mestre: obrigado pela confiança e tolerância.

ao professor Luis Palhares, que iluminou as minhas idéias em momento de dificuldade.

ao professor Aderlon: de onde toda essa idéia partiu.

E não poderia me esquecer do Foguinho, personagem de Lázaro Ramos em Cobras e Lagartos, pois acompanhou meus momentos de lazer. “Palavras são sons ao vento, que assumem o sentido que cada um deseja...”

Resumo

Esta monografia aborda o uso de Redes Neurais Artificiais (RNA) para a solução do problema do controle de um elemento móvel, contido em uma área bidimensional e limitada, até o seu objetivo. Neste trabalho é proposto o uso de RNAs de aprendizado supervisionado numa combinação de RNA Perceptron de Múltiplas Camadas e do Algoritmo de Retropropagação de Erro.

Palavras-chaves: Redes Neurais Artificiais, Perceptron de Múltiplas Camadas, MLP, Retropropagação de Erro, Error Backpropagation, Controle de Móvel.

Abstract

This monograph is about using Artificial Neural Networks (ANN) to solve the problem of controlling a small vehicle, which is contained in a two dimensions and limited area, to its goal. This project is intended to use the Learning Supervised ANNs and a combination of Multi-Layer Perceptrons ANN and the Backpropagation Algorithm.

Key Words: Artificial Neural Networks, ANN, Multi Layer Perceptron, Error Backpropagation, Control of Mobile.

Lista de figuras

Lista de Tabelas

Lista de Abreviações

ANN	*Artificial Neural Network* – Rede Neural Artificial
IDE	*Integrated Development Environment* – Ambiente integrado para desenvolvimento de software.
MCP	*McCulloch-Pitts* - Warren McCulloch e Walter Pitts, publicaram o primeiro artigo de Redes Neurais Artificiais da história.
MLP	*Multi Layer Perceptron* – Perceptron de Múltiplas Camadas
RNA	Rede Neural Artificial

1. Introdução

O cérebro é um computador (sistema de processamento de informação) altamente complexo, não-linear e paralelo. Ele tem a capacidade de organizar seus constituintes estruturais, conhecidos por neurônios, de forma a realizar certos processamentos (p. ex., reconhecimento de padrões, percepção e controle motor) muito mais rapidamente que o mais rápido computador digital hoje existente. [HAYKIN, 2001].

As Redes Neurais Artificiais (RNAs) têm seus princípios físicos e lógicos de funcionamento baseados nos estudos feitos sobre a estrutura do cérebro humano para tentar simular sua forma inteligente de processar informação. Estudos da neurofisiologia consideram que a riqueza computacional do cérebro humano está diretamente ligada ao grande número de neurônios, que por sua vez se interconectam através de uma complexa rede de sinapses.

As Redes Neurais Artificiais, de forma geral, são uma imitação do cérebro humano, possuindo neurônios, processamento simplificado, características de aprendizagem, plasticidade e armazenamento de conhecimento.

A Tabela 1.1 apresenta uma comparação das principais diferenças entre os rígidos sistemas computacionais tradicionais e o cérebro humano, e dela pode ser aferida a utilidade da aplicação das Redes Neurais Artificiais.

Tabela 1.1: Diferenças Entre o Computador e o Cérebro Humano, [SIMPSON, 1990].

Fator	Computador	Cérebro Humano
Elementos computacionais	Processadores	Neurônios
Velocidade de processamento	10^{-9} segundos	10^{-3} segundos

Tabela 1.1: Diferenças Entre o Computador e o Cérebro Humano, [SIMPSON, 1990] (cont.)

Fator	Computador	Cérebro Humano
Tipo de processamento	Serial	Paralelo
Confiabilidade dos elementos	Confiável	Não confiável
Tolerância a falhas	Quase nenhuma	Grande
Tipo de Sinal	Preciso, simbólico	Impreciso
Tipo de controle	Centralizado	Distribuído
Armazenamento de informações	Substituível	Adaptado

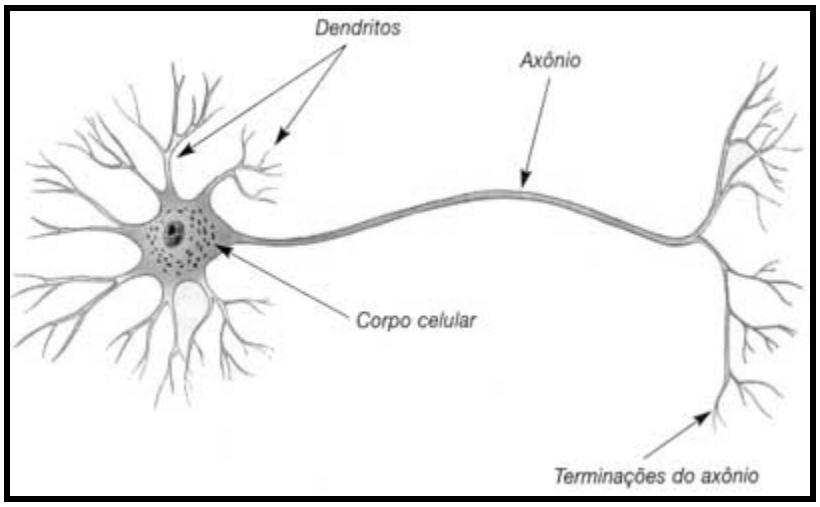


Figura 1.1: Exemplo de neurônio biológico[1].

Os neurônios das Redes Neurais Artificiais possuem características parecidas com as dos neurônios biológicos, e são comumente chamados de neurônios, nós ou células. Cada célula possui apenas uma única saída, denominada de axônio, que pode possuir várias outras ligações colaterais com o mesmo sinal.

[1] Em: http://www.isurp.com.br/aula/ciencia/Marcio/neuronio.jpg. Acesso em 12/12/2006.

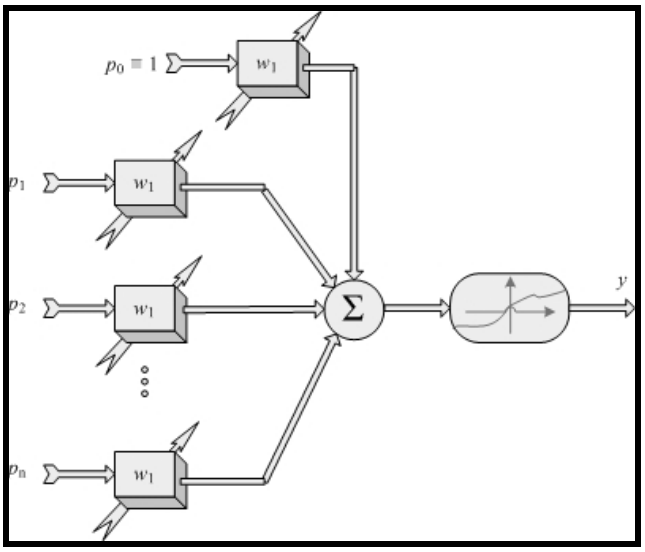


Figura 1.2: Exemplo de neurônio artificial[2].

Podem ser identificados três elementos básicos do modelo neuronal:

- Um conjunto de sinapses (ou elos de conexão), caracterizadas por pesos individuais (multiplicadores);
- Um somador para somar os sinais de entrada;
- Uma função de ativação (ou restritiva) para restringir a amplitude de saída de um neurônio.

[HAYKIN, 2001].

Um grande benefício aferido do uso das Redes Neurais é o uso do aprendizado. Sistemas inteligentes, as Redes Neurais Artificiais são capazes de reter informações e calcular uma melhor rota de destino caso seja necessário.

[2] Em: http://www.ene.unb.br/~adolfo/ISI/Neur%F4nio%20Artificial.jpg. Acesso em 12/12/2006.

Um outro benefício advindo do uso das Redes Neurais Artificiais é a desnecessidade da utilização de uma hipótese inicial sobre as diferenças esperadas no conjunto de dados.

Dessa forma, aplicações complexas em campos como Análise de Padrões, Tolerância a Falhas, Controle de Processos, Monitoração de Motores e Sistemas, Marketing e Análise de Investimentos possuem um novo conceito com relação à tecnologia.

Outra característica interessante das Redes Neurais é a plasticidade. Herdada dos sistemas fisiológicos, a plasticidade beneficia as características mais treinadas ou aquelas que aparecem com maior freqüência durante o processo de aprendizagem.

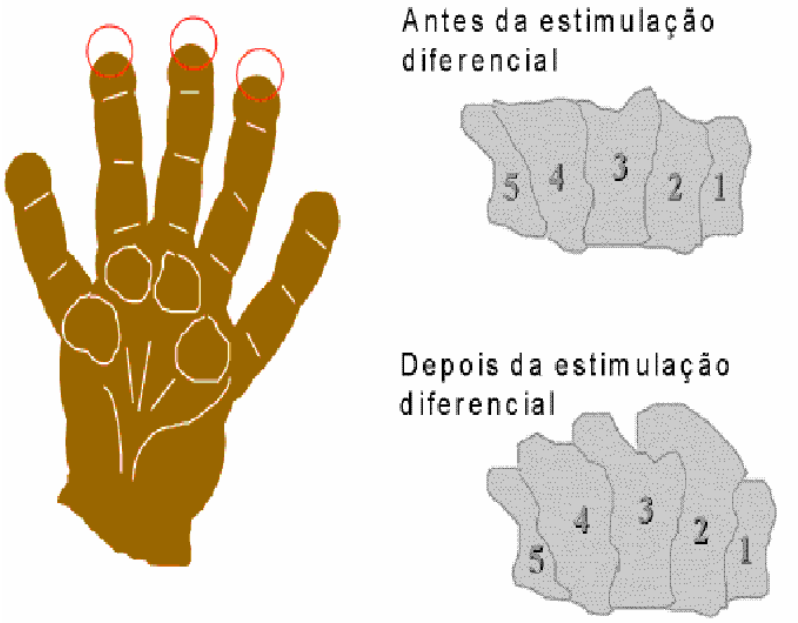


Figura 1.3: Plasticidade de Redes Neurais[3].

[3] Em: http://www.cerebromente.org.br/n05/tecnologia/expansao_pata.gif. Acesso em 12/12/2006.

Uma desvantagem da plasticidade é o "vício". Uma Rede Neural treinada com exemplos repetitivos e parecidos, tende a desenvolver muito determinadas características, como pode ser observado na figura 1.3. Em contrapartida, caso seja necessário obter da Rede Neural alguma característica pouco treinada, provavelmente o resultado não será satisfatório ou a Rede Neural encontrará dificuldade em obter a resposta desejada.

1.1 Motivação

A intenção do presente trabalho busca uma solução para efetuar o controle de um elemento móvel em um ambiente restrito a partir de suas coordenadas e direção atuais, utilizando um ramo da Inteligência Artificial: as Redes Neurais Artificiais. O elemento móvel foi generalizado para que possa ser personalizado em diversos projetos por veículos automotivos, robôs bípedes (humanóides) e quadrúpedes, veículos aquáticos, etc.

Como exemplo de possíveis aplicações baseadas neste projeto podem ser listados:

- Auxílio a deficientes visuais: Um pequeno robô que orienta deficientes visuais em um ambiente ainda desconhecido por eles.
- Carrinhos de supermercados inteligentes: Após serem abandonados em estacionamentos ou locais isolados de um grande supermercado ou hipermercado, os carrinhos de compras retornam com segurança ao lugar apropriado.
- Vigilância de edifícios e residências: Um robô dotado de inteligência de controle pode percorrer os corredores de grandes edifícios como *shoppings centers* ou residências, em missões de vigilância.

- Guia de turistas em hotéis ou museus: Robôs ou maquinários móveis inteligentes podem auxiliar turistas em hotéis, transportando suas bagagens enquanto lhes direcionam ao seu quarto, ou museus, percorrendo os corredores sozinhos como guias enquanto executam gravações informativas.

- Busca de motoristas e passageiros: Os motoristas e passageiros de veículos podem aguardar os seus veículos que se deslocam da garagem ou estacionamento até a saída de um local específico, o que proporciona conforto no caso de congestionamento de veículos ou em casos de chuvas, ou segurança no caso de estacionamentos afastados ou mal iluminados.

Um projeto que serviu de motivação extra para o desenvolvimento deste trabalho foi o Autonomous Land Vehicle In a Neural Network – ALVINN[4]. É um sistema de percepção (visão) para o controle de veículos utilizando Redes Neurais Artificiais que aprende como controlar o veículo observando o motorista dirigindo. O projeto ALVINN está descrito nos trabalhos do instituto *The Robotics Institute*[5].

1.2 Objetivos

Este projeto possui motivação acadêmica e visa proporcionar subsídios para a solução de problemas relativos a controle inteligente de móveis em ambientes restritos, baseado em sua posição e direção no ambiente em que se encontram. O móvel controlado deve ter capacidade suficiente para atingir a saída do ambiente (ou o ponto de objetivo) de forma autônoma.

[4] http://www.ri.cmu.edu/projects/project_160.html. Acesso em 12/12/2006.
[5] http://www.ri.cmu.edu. Acesso em 12/12/2006.

Visa ainda propor um protótipo de Rede Neural Artificial para efetuar o controle do móvel em um ambiente de simulação construído em linguagem de programação, onde será validada a solução encontrada.

A figura 1.4 retrata a idéia proposta.

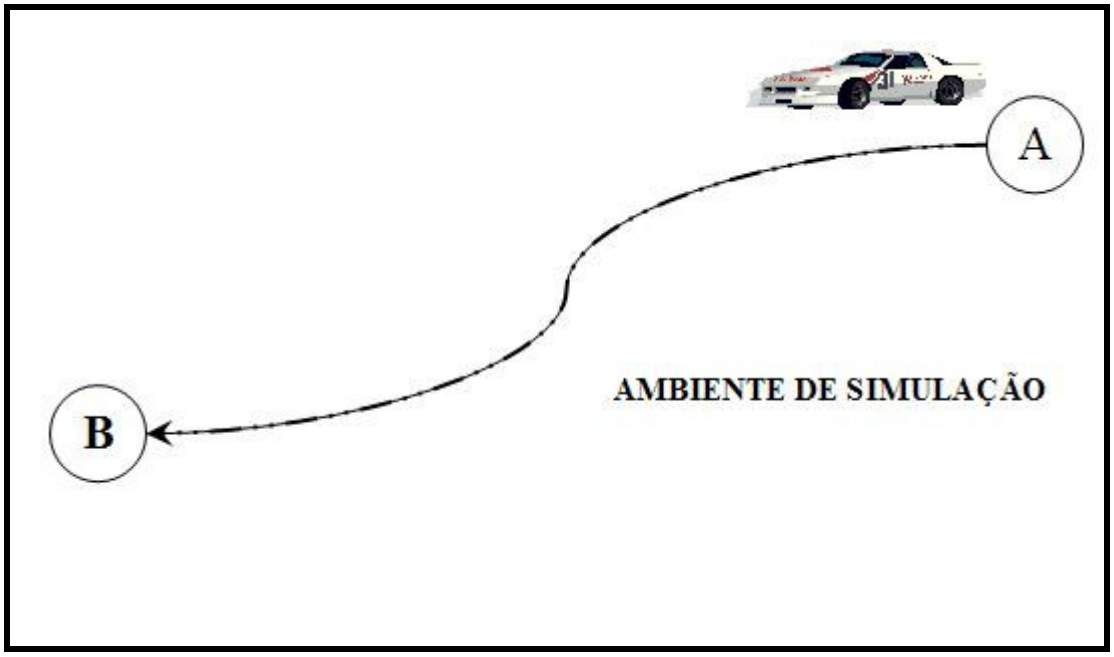


Figura 1.4: Ambiente Virtual de Simulação.

O traçado da rota do veículo da figura 1.4 foi propositalmente generalizado para reforçar a idéia de que o móvel deve traçar o seu caminho até a saída ou objetivo de forma autônoma.

1.3 Estrutura do Trabalho

Esta monografia está organizada em uma estrutura de capítulos, contendo 5 capítulos significativos sobre o tema abordado.

O primeiro capítulo traz a introdução aos sistemas de Redes Neurais e apresenta a motivação e o objetivo do trabalho.

O segundo capítulo aborda Redes Neurais Artificiais e embasa o trabalho sob o aspecto teórico. Aborda os conceitos de Neurônio Artificial, Redes Neurais Artificiais Perceptron de Múltiplas Camadas e o Algoritmo de Retropropagação do erro.

O terceiro capítulo retrata a descrição do projeto do protótipo da Rede Neural Artificial adotada e a modelagem e desenvolvimento do Ambiente de Simulação proposto.

O quarto capítulo relata os resultados obtidos com o uso da Rede Neural Artificial para a simulação do controle de um móvel em um ambiente restrito e faz considerações a respeito dos testes realizados.

O quinto e último capítulo tece conclusões a respeito do projeto realizado, avaliando a aplicabilidade de Redes Neurais Artificiais ao controle de móveis, e oferece sugestões para futuros projetos e estudos.

2. Redes Neurais Artificiais

2.1 O Neurônio de McCulloch-Pitts

2.1.1 Descrição do Neurônio de McCulloch-Pitts (MCP)

Em 1943, o matemático Warren McCulloch e o estatístico Walter Pitts publicaram um artigo no *Bulletin of Mathematical Biophysics* com o título "A Logical Calculus of the Ideas Immanent in Nervous Activity". Esse artigo descreveu o que se tem como o primeiro registro de Redes Neurais Artificiais da história.

"O neurônio descrito por McCulloch e Pitts era ingenuamente simples, talvez mais simples do que deveria ser diante da informação que naquela época já estava disponível sobre o comportamento elétrico da célula nervosa", [Kovács, 2002, p. 28]. Esse neurônio é um dispositivo binário cuja saída pode ser pulso ou não pulso. Às suas entradas são aplicados ganhos arbitrários e individuais e podem ser de entradas de natureza excitatória (positivas) ou inibitória (negativas).

A saída do neurônio é determinada como a soma ponderada das entradas e os fatores de ponderação são os pesos individuais atribuídos a cada entrada.

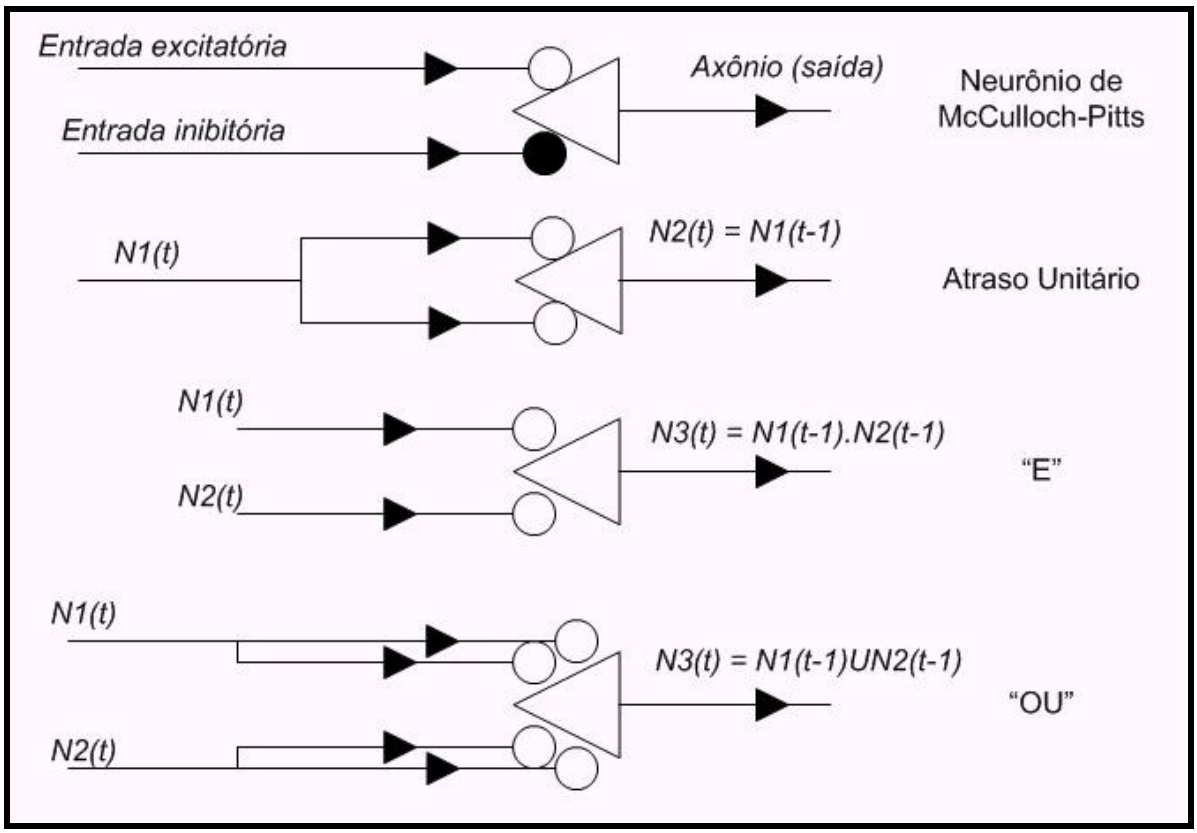


Figura 2.1: Implementações de algumas funções utilizando o neurônio descrito por McCulloch-Pitts.

Fonte: Autor. Baseado em [KOVACS, 1996].

O disparo de um neurônio biológico ocorre quando a soma dos potenciais de ação que ele recebe ultrapassa o seu limiar de excitação.

Para determinar a saída do neurônio observa-se o resultado da soma ponderada: caso seja maior ou igual a um limiar estabelecido, então a saída do neurônio é pulso; caso contrário é não pulso.

Dessa forma, várias funções podem ser implementadas variando-se a quantidade e a forma de conexão de neurônios, os níveis de seus limiares de ativação e os seus pesos

(ganhos). A figura 2.1 reproduz algumas implementações de funções booleanas utilizando ganhos iguais a 1/2 e limiares iguais a 1.

A essência da proposta de McCulloch-Pitts foi a seguinte: "A inteligência é equivalente ao cálculo de predicados que por sua vez pode ser implementado por funções booleanas. Por outro lado, o sistema nervoso é composto de redes de neurônios, que com as devidas simplificações, têm a capacidade básica de implementar estas funções booleanas. Conclusão: a ligação entre inteligência e atividade nervosa fica estabelecida de forma científica", [Kovács, 2002, p. 29].

2.1.2 O Discriminador Linear

O neurônio descrito por McCulloch-Pitts pode ser modelado como um caso particular de discriminador linear com entradas binárias [Kovács, 2002, p. 29]. Genericamente, define-se um discriminador linear com *n* entradas $\{x_1, x_2, ..., x_n\}$ e uma saída *y* por uma das seguintes expressões:

$$y = H\left(\sum_{i=1}^{n} w_i x_i - \Theta\right) = h(w' x - \Theta) \text{ , quando } y \in [0;1]$$

ou

$$y = \text{sgn}\left(\sum_{i=1}^{n} w_i x_i - \Theta\right) = \text{sgn}(w' x - \Theta) \text{ , quando } y \in [-1;1] \qquad \text{Eq.(2.1)}$$

Essas equações afetam diretamente a saída do discriminador e fazem parte da escolha da estrutura da RNA: para resultados y entre 0 e 1, a primeira é a equação mais adequada; para resultados *y* entre -1 e 1, a segunda.

Nas equações descritas em (2.1) os componentes do vetor w, {w1, w2, w3, ..., wn} representam os pesos (ganhos) associados a cada entrada xi e Θ representa o valor do limiar de ativação do neurônio.

Dessa forma, as equações que descrevem o discriminador linear podem ser ilustradas em um diagrama de blocos conforme a figura 2.2.

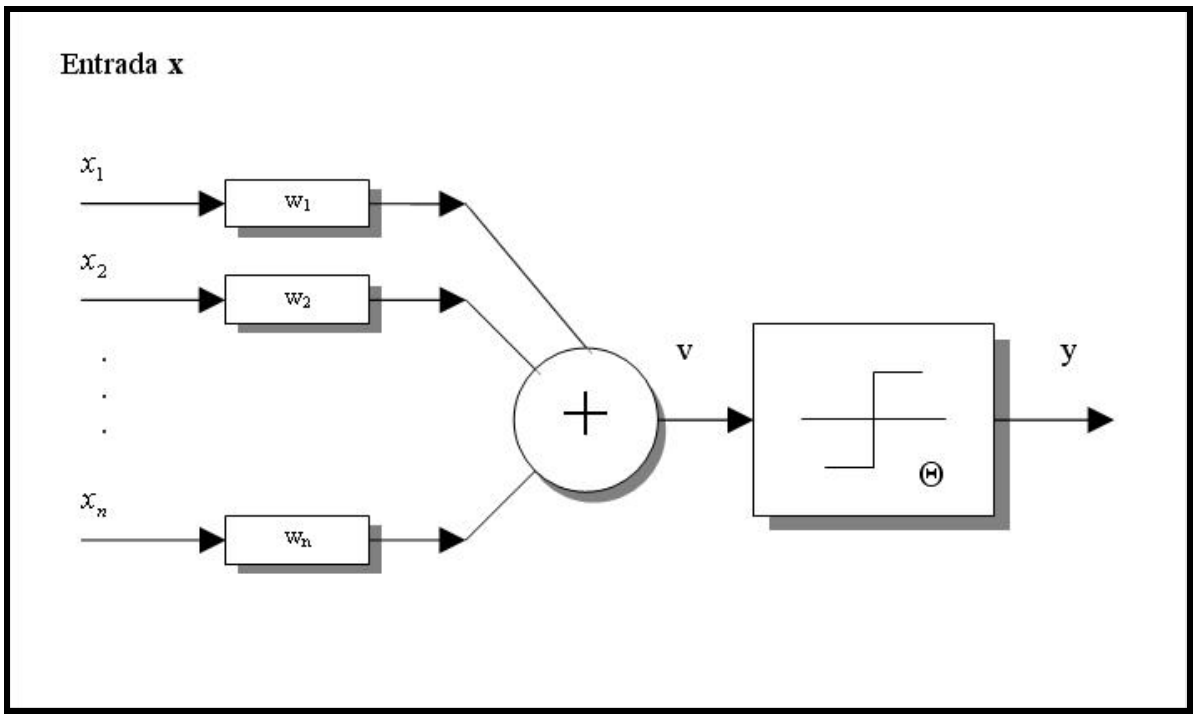


Figura 2.2: Diagrama de blocos do Discriminador Linear.

Fonte: Autor.

As equações descritas em (2.1) “representam um hiperplano que divide o espaço euclidiano $\Re^n$ de dimensão n, em duas regiões A e B”, [Kovács, 2002, p. 30]. Dessa forma é possível definir a posição de um vetor x de componentes de entrada numa das regiões separadas pelo hiperplano das equações de (2.1) se o limiar de ativação for ou não satisfeito, conforme a figura 2.3.

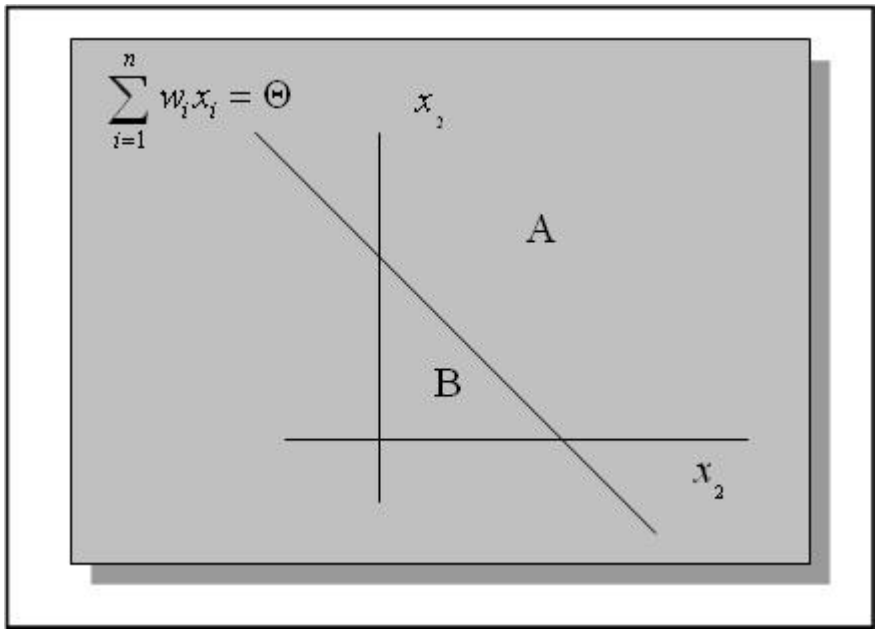


Figura 2.3: O hiperplano separa o espaço euclidiano em duas regiões, acima e abaixo da condição de ativação do neurônio.

Fonte: Autor. Baseado em [BRAGA, 2000].

Esse discriminador só pode ser empregado em situações onde os grupos de elementos pertencentes a A e B formem aglomerados no espaço $\Re^n$ de tal forma que seja possível passar um hiperplano que separe estes dois aglomerados (coleções linearmente separáveis).

Algumas limitações na descrição do modelo MCP original, [Braga, 2000, p. 9]:

1) redes MCP com apenas uma camada só conseguem implementar funções linearmente separáveis;
2) pesos negativos são mais adequados para representar disparos inibidores;
3) o modelo foi proposto com pesos fixos, não ajustáveis.

2.2 Funções de ativação

2.2.1 O neurônio como unidade de processamento

O neurônio artificial apresentado por McCulloch-Pitts apresentava algumas limitações e não era adequado para uso em sistemas de processamentos mais complexos por não ter aplicado em sua estrutura básica um bias, parâmetro responsável pelo efeito de aumentar ou diminuir a entrada líquida da função de ativação.

De forma abrangente, o bias é o parâmetro que faltava ao modelo MCP para a utilização sistêmica do limiar de ativação, em comparação com outros modelos propostos posteriormente.

O modelo de neurônio, como unidade de processamento de informação fundamental para a operação de Redes Neurais Artificiais, foi descrito por [Haykin, 2001, p. 38] como:

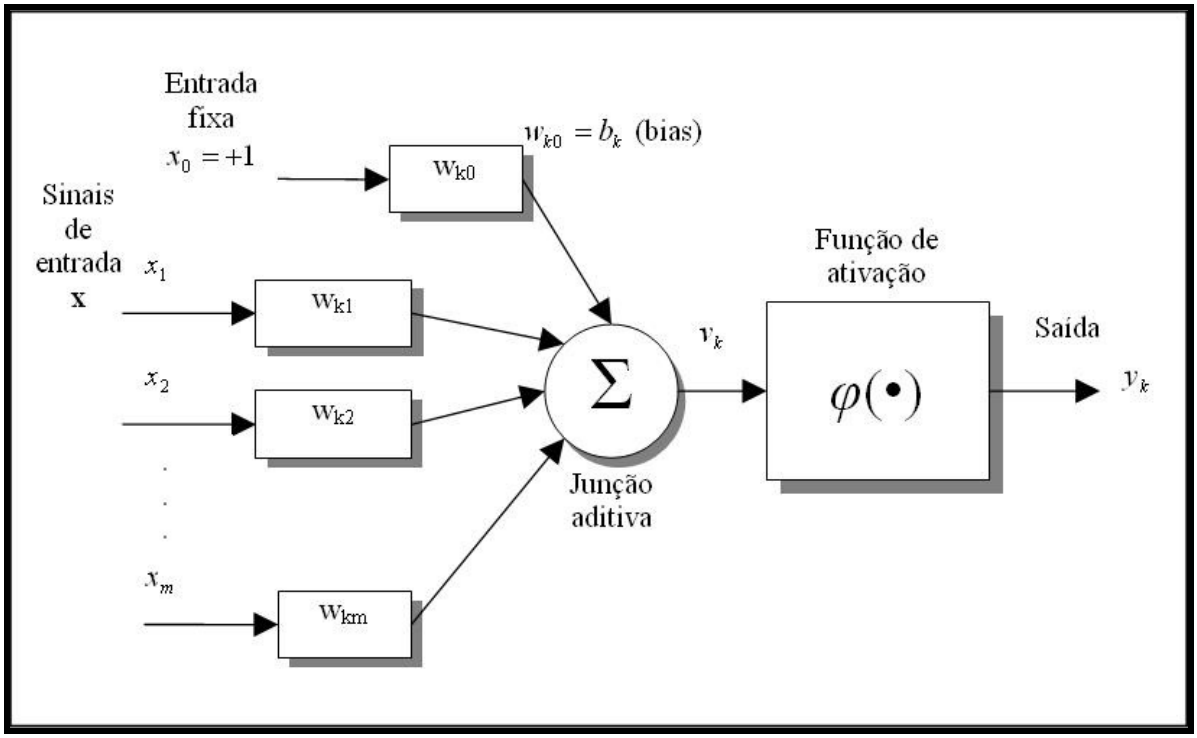


Figura 2.4: Um outro modelo não-linear de um neurônio, detalhado por Haykin.

Em particular, para o diagrama apresentado pela figura 2.4, pode-se expressar em termos matemáticos a descrição da saída y de um neurônio k como:

$$y_k = \varphi(v_k) \qquad \text{Eq.(2.2)}$$

onde

$$v_k = \sum_{j=0}^{m} w_{kj} x_j \qquad \text{Eq.(2.3)}$$

De acordo com a equação 2.3, o bias b_k é sempre incluído no somatório com as demais entradas combinadas com o vetor de entradas **x** e com o vetor de pesos **w** do neurônio (o primeiro termo é $x_0 = +1$, que multiplicado pelo bias b_k contido em w_0 é igual ao próprio bias).

2.2.2 Tipos de funções de ativação

As funções que regem a ativação ou não de um neurônio definem a sua saída em termos da saída do combinador linear associado ao bias (v_k). Essas funções são chamadas de Funções de Ativação e são representadas por $\varphi(\cdot)$. De acordo com [Haykin, 2001, p. 39], são identificados três tipos básicos de funções de ativação:

A) Função de Limiar

Para este tipo de função de ativação, descrito na figura 2.5a:

$$\varphi(v) = \begin{cases} 1, & se \quad v \geq 0 \\ 0, & se \quad v < 0 \end{cases} \qquad \text{Eq.(2.4)}$$

Este é o tipo de função de ativação referido na literatura como o modelo MCP. Nesse modelo, a saída de um neurônio assume o valor 1 se o campo local induzido naquele

neurônio é não negativo, e 0, caso contrário. É um modelo bastante simples, porém não tem larga utilização em sistemas complexos.

B) Função Linear por Partes

Para este tipo de função de ativação, descrito na figura 2.5b:

$$\varphi(v) = \begin{cases} 1, & se \quad v \geq +\frac{1}{2} \\ v, & se \quad -\frac{1}{2} < v < +\frac{1}{2} \\ 0, & se \quad v \leq -\frac{1}{2} \end{cases} \qquad \text{Eq.(2.5)}$$

Esta forma de função de ativação pode ser observada como uma aproximação de um amplificador não-linear. São formas especiais dessa função de ativação:

1) Combinador Linear: A região linear de operação não atinge a saturação;

2) Função de Limiar: O fator de amplificação da região linear é muito grande (a inclinação da reta da região linear passa a ser aproximadamente 90°).

C) Função Sigmóide

A função sigmóide é a função de ativação mais utilizada nos projetos de redes neurais artificiais. Sua curva em forma de *s* é exibida na figura 2.5c. Por assumir um intervalo de valores determinados (como [0,1] ou [-1,1]), a função sigmóide representa uma escolha desejável quando a saída da função de ativação não deve ser necessariamente binária.

Um exemplo de função sigmóide é a função logística, definida por:

$$\varphi(v) = \frac{1}{1 + \exp(-av)} \qquad \text{Eq.(2.6)}$$

Na função logística, o parâmetro *a* indica a inclinação da função sigmóide. Variando-se o parâmetro *a* obtém-se funções sigmóides com diferentes inclinações, de acordo com a figura 2.5c. A função logística também se comporta como função de limiar, quando o parâmetro de inclinação se aproxima do infinito. Outro fator importante da função sigmóide é a capacidade de diferenciabilidade, fato que não ocorre com a função de limiar.

Outro tipo de função sigmóide comumente adotada é a função tangente hiperbólica, definida por:

$$\varphi(v) = \tanh(v) \qquad \text{Eq.(2.7)}$$

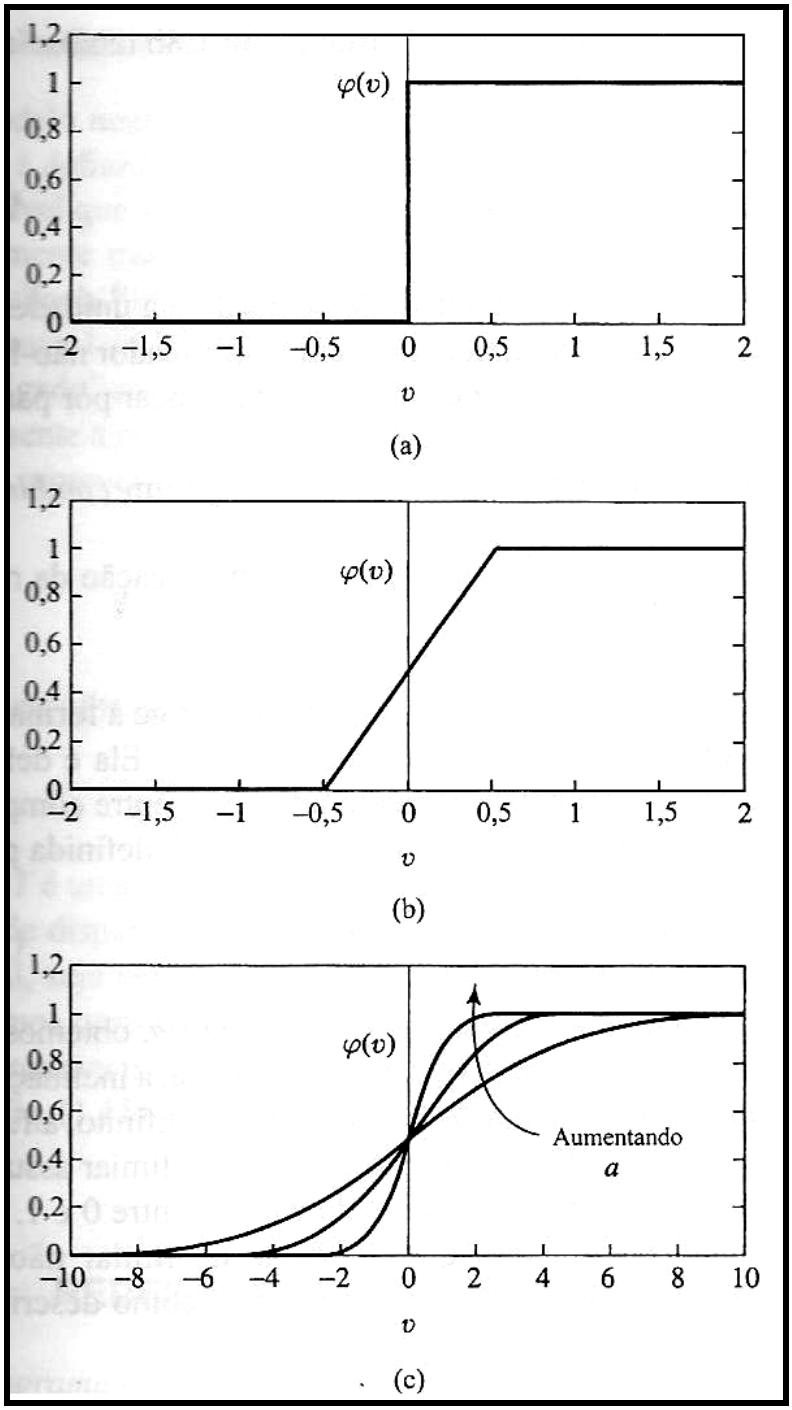


Figura 2.5: (a) Função de limiar. (b) Função linear por partes. (c) Função sigmóide para parâmetro de inclinação a variável.

Fonte: [Haykin, 2001].

2.3 Arquiteturas de Redes Neurais Artificiais

As arquiteturas das RNAs têm a disposição e conexão de seus neurônios de forma a otimizar os problemas tratados por elas. As redes MCP de uma única camada, por exemplo, só conseguem resolver problemas linearmente separáveis, de acordo com o exibido na figura 2.3.

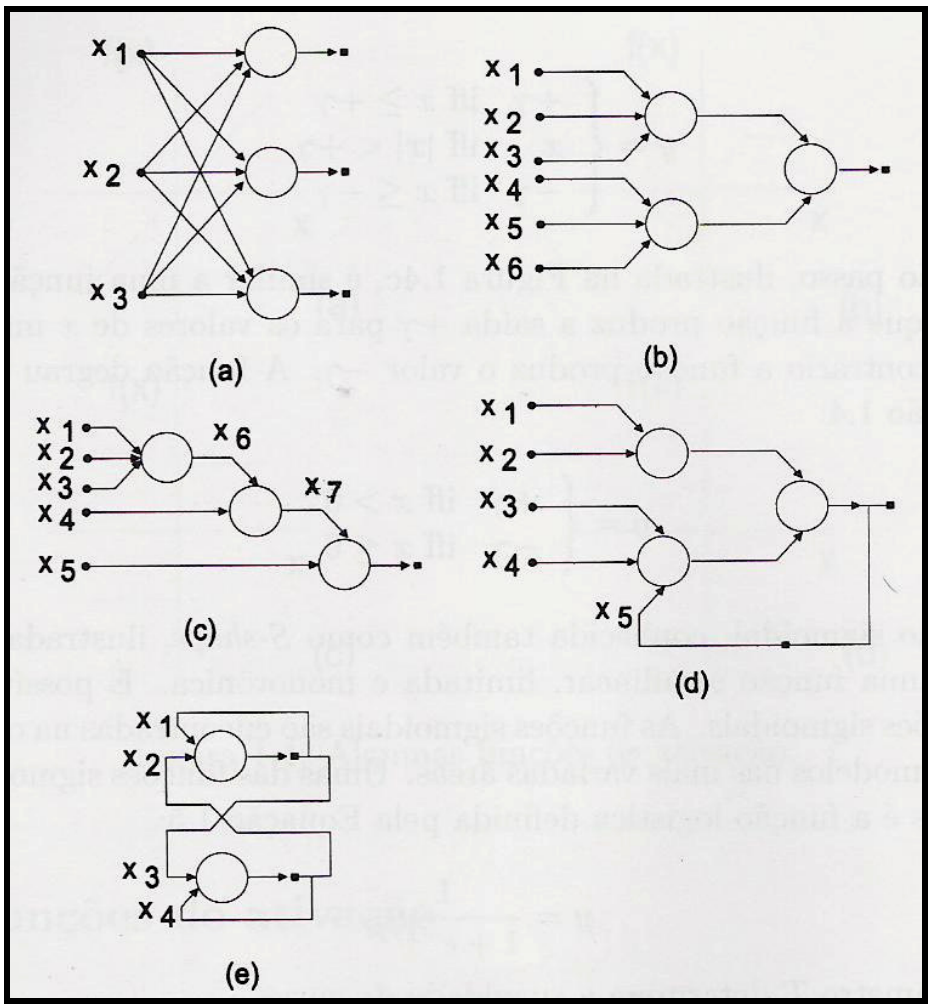


Figura 2.6: Exemplos de arquiteturas de RNAs.

Fonte: [Haykin, 2001].

2.3.1 Arquiteturas de RNAs quanto ao número de camadas

1) Redes de camada única: só existe um nó entre qualquer entrada e qualquer saída da rede (figura 2.6 a,e).
2) Redes de múltiplas camadas: existe mais de um neurônio entre alguma entrada e alguma saída da rede (figura 2.6 b, c, d).

2.3.2 Arquiteturas de RNAs quanto ao tipo de conexões

1) Acíclica, ou *feedforward*: a saída do neurônio na i-ésima camada da rede não pode ser usada como entrada de nodos em camadas de índice menor ou igual a i (figura 2.6 a, b, c).
2) Cíclica, ou *feedback*: a saída de algum neurônio na i-ésima camada da rede é usada como entrada de nodos em camadas de índice menor ou igual a i (figura 2.6 d, e).
    a. Redes cuja saída final (única) é ligada às entradas comportam-se como autômatos reconhecedores de cadeias, onde a saída que é realimentada fornece o estado do autômato (figura 2.6 d).
    b. Se todas as ligações são cíclicas, a rede é denominada auto-associativa. Estas redes associam um padrão de entrada com ele mesmo, e são

particularmente úteis para recuperação ou "regeneração" de um padrão de entrada (figura 2.6 e).

2.3.3 Arquiteturas de RNAs quanto à sua conectividade

1) Rede fracamente (ou parcialmente) conectada (figura 2.6 b, c, d). Como o próprio nome sugere, é uma Rede Neural Artificial onde seus nós não estão conectados de todas as formas possíveis.
2) Rede completamente conectada (figura 2.6 a, e). Essa arquitetura, como o próprio nome sugere, tem em todos os nós todas as conexões possíveis com todos os nós posteriores.

2.4 RNAs e Aprendizagem

De todas as características interessantes das redes neurais artificiais, a habilidade de aprender é a que mais chama atenção. Isso quer dizer que não é necessário especificar todos os detalhes de um processo computacional, basta treinar uma rede para realizar este processo. Partindo desse pressuposto, as redes neurais artificiais podem gerar respostas novas para problemas que nunca foram apresentados anteriormente, inclusive os que possuem descrição complicada em processos computacionais tradicionais.

O objetivo do treinamento de uma rede neural artificial é fazer com que para todos os conjuntos de entradas seja produzido um conjunto de respostas desejadas, ou no mínimo satisfatórias.

Em [Braga, 2000, p. 15], o conceito de aprendizagem é bastante claro:

> "Aprendizagem é o processo pelo qual os parâmetros de uma rede neural são ajustados através de uma forma continuada de estímulo pelo ambiente no qual a rede está operando, sendo o tipo específico de aprendizagem realizada definido pela maneira particular como ocorrem os ajustes realizados nos parâmetros".

Os procedimentos de treinamento que levam as redes neurais artificiais a aprender determinadas tarefas podem ser classificados em duas classes de treinamento: aprendizado supervisionado e aprendizado não-supervisionado.

2.4.1 Processo de aprendizado supervisionado

Este é o método de aprendizado mais comum no treinamento das RNAs, tanto de neurônios com pesos como de neurônios sem pesos. Como as entradas e saídas do sistema neural são fornecidas por um agente externo (um professor), esse tipo de rede é denominada *rede de aprendizado supervisionado*.

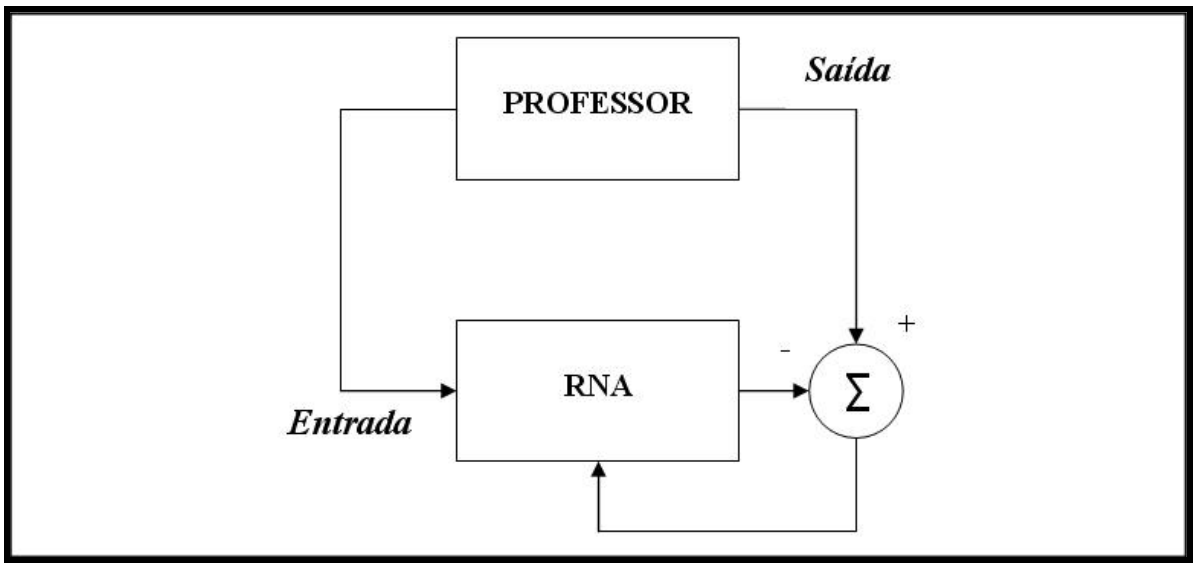


Figura 2.7: Modelo de Redes Neurais Artificiais de Aprendizado Supervisionado.

Fonte: Autor.

O objetivo das RNAs de aprendizado supervisionado é ajustar os parâmetros da rede, de forma a encontrar uma ligação entre os pares de entrada e saída fornecidos.

O procedimento de treinamento, basicamente, funciona da seguinte maneira: o conjunto de informações (vetor) de entrada é aplicado. A saída da rede é calculada e comparada com o conjunto de informações (vetor) de resultados desejados. O erro encontrado é então realimentado através da rede e os pesos são atualizados através de um algoritmo determinado para minimizar esse erro. O processo de treinamento é repetido até que se tenha alcançado valores de erros satisfatórios. A soma dos erros quadráticos de todas as saídas é normalmente utilizada como medida de desempenho da rede e também como função de custos a ser minimizada pelo algoritmo de treinamento.

Um grande ponto fraco observado nas redes neurais artificiais com aprendizado supervisionado é que na ausência do professor, a rede não terá capacidade para aprender

novas estratégias para casos ainda não abordados pelos exemplos fornecidos através dos pares de treinamento (vetores de entrada e resultados desejados).

De acordo com [Braga, 2000, p. 16], os exemplos mais conhecidos de algoritmos para aprendizado supervisionado são a regra delta e a sua generalização para redes de múltiplas camadas, o algoritmo backpropagation.

O aprendizado supervisionado pode ser dos tipos *on-line* e *off-line*. Nos casos de treinamento *off-line*, os dados do conjunto de treinamento não se alteram, e quando é obtida uma solução para a RNA, esta deve se tornar fixa. Em caso de adição de novos dados ao conjunto de treinamento da rede, um novo treinamento será necessário para o adequado funcionamento da RNA, envolvendo, também, os dados anteriores. Nos casos de treinamento *on-lin*e o conjunto de dados está em constante mudança e o processo de adaptação e treinamento é contínuo.

2.4.2 Processo de aprendizado não-supervisionado

Como o próprio nome sugere, no aprendizado não-supervisionado não existe a figura de um supervisor para acompanhar o processo de aprendizado. De forma geral, nesse processo de aprendizagem, somente os padrões de entrada estão disponíveis para a rede, não sendo requerido o vetor de saídas desejadas e obviamente não sendo feitas comparações para determinar a resposta ideal.

"O conjunto de treinamento modifica os pesos da rede de forma a produzir saídas que sejam consistentes, isto é, tanto a apresentação de um dos vetores de treinamento, como

a apresentação de um vetor que é suficientemente similar, produzirão o mesmo padrão nas saídas", [Aurélio, 99, p. 41].

Esse tipo de rede trabalha com extração das propriedades estatística da entrada de dados, o que possibilita o desenvolvimento de uma habilidade de criar representações internas para codificar características do padrão de entrada e criar novas classes ou grupos de forma automática. Para que este tipo de aprendizado se torne viável é necessário que exista a redundância nos dados de entrada, pois sem redundância seria praticamente impossível encontrar quaisquer padrões ou característica padrões nos dados inseridos.

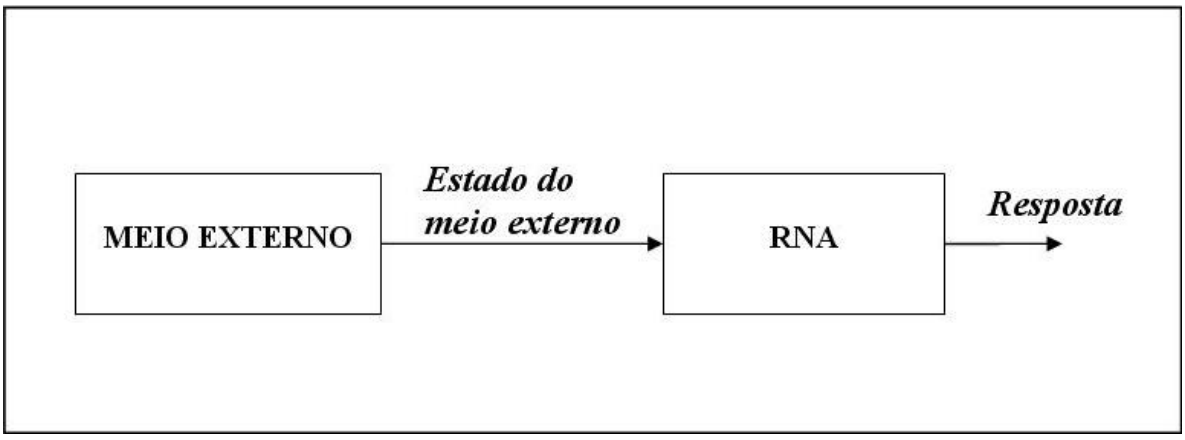


Figura 2.8: Modelo de Aprendizado Não-Supervisionado.

2.4.3. Aprendizagem por correção de erros

Processo de natureza de aprendizagem supervisionada, a aprendizagem por correção de erros busca minimizar a diferença entre a saída calculada pela rede e a saída ótima (desejada), ou seja, o erro da resposta atual da rede.

Observando a figura 2.7, um exemplo típico de uma RNA de aprendizado supervisionado, é fácil verificar que o sinal de erro e(n) pode ser obtido através da soma entre o sinal de saída desejado e o sinal de saída calculado pela RNA (cujo sinal é negativo, conforme figura 2.7). O argumento n representa o instante de tempo discreto, ou mais precisamente, o passo de tempo de um processo iterativo envolvido no ajuste dos pesos sinápticos do neurônio k.

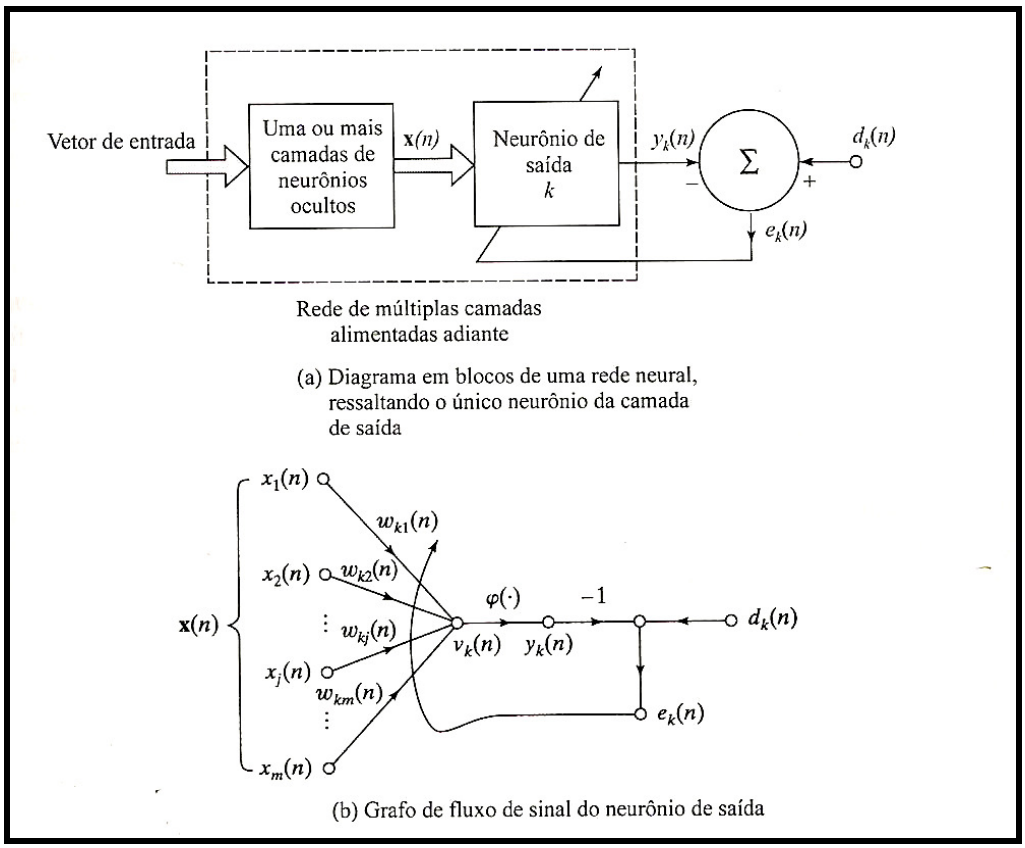


Figura 2.9: Ilustração da aprendizagem por correção de erro.

Fonte: [Haykin, 2001].

Generalizando para um neurônio k de um sistema neural como o apresentado na figura 2.9a [Haykin, 2001, p.77], o sinal de erro pode ser definido como:

$$e_k(n) = d_k(n) - y_k(n) \qquad \text{Eq.(2.8)}$$

onde $d_k(n)$ representa a resposta desejada ou saída-alvo e $y_k(n)$ representa a o sinal de saída dos neurônios.

O sinal de erro $e_k(n)$ aciona um mecanismo de controle, que tem por finalidade a aplicação de uma seqüência de ajustes corretivos aos pesos sinápticos do neurônio k. O objetivo do mecanismo de controle, obviamente, é aproximar, passo a passo, o sinal de saída $y_k(n)$ da resposta desejada $d_k(n)$. Este objetivo é alcançado minimizando-se uma função de custo ou índice de desempenho, $\varepsilon(n)$. Definindo em termos do sinal de erro $e_k(n)$ como:

$$\varepsilon(n) = \frac{1}{2} e_k^2(n) \qquad \text{Eq.(2.9)}$$

Com isso, $\varepsilon(n)$ é o valor instantâneo da energia do erro. Os ajustes passo a passo dos pesos sinápticos do neurônio k continuam até o sistema atingir um estado estável.

A forma genérica para alteração dos pesos por correção de erros é apresentada na equação (2.11). É decorrente da regra delta, uma regra de Widrow-Hoff [Haykin, 2001, p. 77].

Suponha-se que $w_{kj}(n)$ representa o valor do peso sináptico w_{kj} do neurônio k excitado por um elemento $x_j(n)$ do vetor de sinal x(n) no passo de tempo n. De acordo com a regra delta, o ajuste decorrente da diferença de pesos $\Delta w_{kj}(n)$ aplicado ao peso sináptico w_{kj} no passo de tempo n pode ser definido como:

$$\Delta w_{kj}(n) = \eta e_k(n) x_j(n) \qquad \text{Eq.(2.10)}$$

onde η é uma constante positiva determinadora da taxa de aprendizado ao avançar em um passo no processo de aprendizagem.

"O ajuste feito em um peso sináptico de um neurônio é proporcional ao produto do sinal de erro pelo sinal de entrada da sinapse em questão", [Haykin, 2001, p. 78].

Generalizando a equação (2.10) para a obtenção de um valor atualizado do peso sináptico w_{kj}:

$$w_{kj}(n+1) = w_{kj}(n) + \Delta w_{kj}(n) \qquad \text{Eq.(2.11)}$$

Pode-se visualizar os valores $w_{kj}(n+1)$ e $w_{kj}(n)$ como os valores novo e antigo do peso sináptico w_{kj}, respectivamente.

Para a realização da correção de erros através da técnica apresentada, é necessário que a resposta desejada seja fornecida por alguma fonte externa (no aprendizado supervisionado, o professor), que seja diretamente acessível ao neurônio k.

A figura 2.9b exibe uma representação em grafo de fluxo de sinal do processo de aprendizagem por correção de erro, dando destaque à atividade na vizinhança do neurônio k.

2.4.4. Aprendizado Hebbiano

O postulado de aprendizado de Hebb é uma das regras de aprendizagem mais famosas no âmbito das RNAs. Sua regra de aprendizado é aplicada ao processo de aprendizagem não-supervisionada e propõe que o peso de uma conexão sináptica deve ser ajustado se houver sincronismo entre os "níveis de atividade" das entradas e saídas. A sinapse será fortalecida se dois neurônios em lados distintos da sinapse forem ativados sincronamente e será enfraquecida ou mesmo eliminada se forem ativados assincronamente. Em outras palavras, a conexão entre os neurônios pré-sináptico e o pós-sináptico deve ser reforçada se o neurônio pré-sináptico tiver grande influência na ativação do segundo neurônio.

Esse tipo de sinapse é denominado sinapse hebbiana. De forma precisa, pode-se definir uma sinapse hebbiana como uma sinapse que usa um mecanismo dependente do tempo, altamente local e fortemente interativo para aumentar a eficiência sináptica como uma função da correlação entre as atividade pré-sinápticas e pós-sináptica [Haykin, 2001, p. 80].

Características principais da sinapse hebbiana:

1) Mecanismo interativo: a ocorrência de uma modificação em uma sinapse hebbiana depende dos sinais em ambos os lados da sinapse e não se pode analisar as atividades de forma isolada. Qualquer modificação na sinapse hebbiana depende, de

forma determinística ou estatística, da interação entre os sinais pré-sinápticos e os pós-sinápticos. Assim, não se pode fazer uma previsão a partir de apenas uma dessas duas atividades.

2) Mecanismo local: uma sinapse hebbiana é uma transmissão com sinais contínuos que produz modificações sinápticas locais que são entradas específicas. Esses sinais são portadores de informação (representando a atividade incidente nas unidades pré-sináptica e pós-sináptica). É essa informação, localmente disponível, que é utilizada pela sinapse hebbiana para produzir as modificações sinápticas locais específicas para a entrada.

3) Mecanismo dependente do tempo: as modificações em uma sinapse hebbiana dependem do momento exato de ocorrência das atividades pré e pós-sinápticas.

4) Mecanismo correlacional ou conjuncional: a sinapse hebbiana pode ser chamada de sinapse conjuncional pelo fato de a ocorrência conjunta de atividades pré e pós-sinápticas ser suficiente para que exista uma modificação. Além disso, pode também ser chamada de sinapse correlacional porque uma correlação entre estas mesmas atividades também é suficiente para gerar mudanças.

A forma mais simples de aprendizagem hebbiana, ou seja, uma regra para a mudança do peso sináptico, pode ser descrita pelo postulado de Hebb:

$$\Delta w_{ij}(t) = \eta y_i(t) x_j(t) \qquad \text{Eq.(2.12)}$$

onde η representa uma constante positiva que determina a taxa de aprendizado.

A regra de Hebb é classificada como aprendizado não-supervisionado, pois não conta com um supervisor externo para verificar a qualidade da resposta calculada pela rede e realizar os ajustes necessários para minimizar o erro, mas mesmo assim os vetores de entrada e saída ainda são fornecidos (conforme o aprendizado supervisionado). No aprendizado hebbiano o treinamento da rede é feito independentemente da resposta atual da rede, através de um mecanismo local à sinapse.

2.5. Perceptrons

Em 1958, Rosenblatt foi um dos pioneiros das redes neurais artificiais que se sobressaíram por grandes contribuições de peso ao universo da Inteligência Artificial – IA. Rosenblatt propôs o perceptron como o primeiro modelo para aprendizagem supervisionada. O perceptron é a forma mais simples de uma RNA utilizado para a classificação de padrões linearmente separáveis, ou seja, padrões que se encontram em lados diferentes de um hiperplano.

O perceptron é basicamente constituído de um único neurônio com pesos sinápticos ajustáveis e bias. Quando está construído em torno de um único neurônio, é limitado a realizar classificação de padrões com apenas duas classes (hipóteses). Expandindo a camada de saída do perceptron para incluir mais de um neurônio pode-se correspondentemente realizar classificação com mais de duas classes. Porém, todas as classes devem ser linearmente separáveis para que o perceptron funcione adequadamente.

2.5.1. Portas de limiar (threshold gates)

De acordo com a sua importância histórica e facilidade na compreensão correta do perceptron, as portas de limiar (threshold gates) serão detalhadas nessa seção. As threshold pode ser divididas em três tipos: linear, quadrática e polinomial. A função executada por cada uma delas é basicamente a mesma: comparação de uma soma ponderada das entradas com um valor de limiar (threshold). Caso a soma exceda o limite imposto pelo limiar, a saída é ativada, permanecendo desativada em caso contrário. No entanto, estes modelos diferem entre si pela complexidade com que seus pesos são calculados.

1) Portas de limiares lineares

Essas portas de limiares lineares são muito parecidas com os nós MCP e podem ser visualizadas através da figura 2.10.

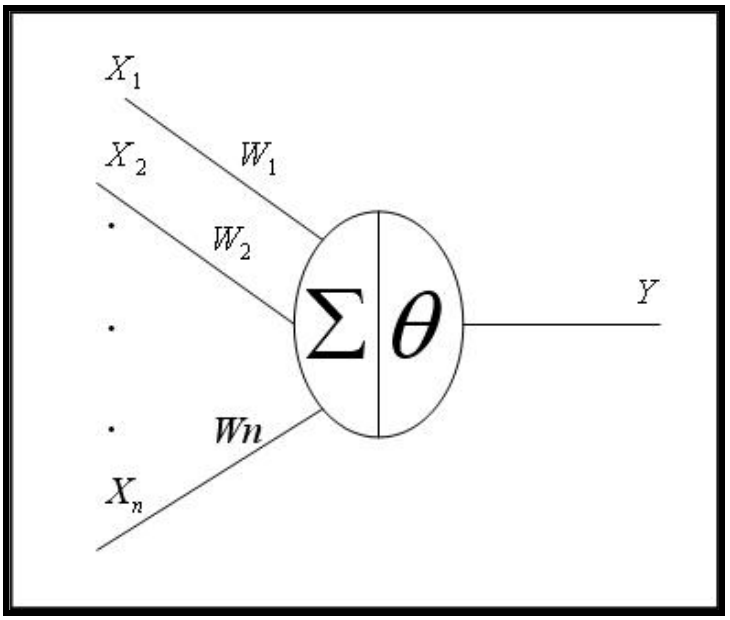


Figura 2.10: Porta de limiar linear. Baseado em [Braga, 2000].

$$y = \begin{cases} 1 & \sum w_i x_i \geq \theta \\ 0 & \sum w_i x_i < \theta \end{cases} \qquad \text{Eq.(2.13)}$$

De acordo com a equação 2.13, as portas de limiar lineares estão restritas à solução de problemas que sejam linearmente separáveis, ou seja, a problemas cuja solução pode ser atingida através da separação de duas regiões por meio de uma reta ou hiperplano (para o caso n-dimensional).

Com uma mesma porta limiar pode-se implementar qualquer uma das funções E, OU, NÃO-E, NÃO-OU, entre outras, bastando alterar os parâmetros da porta. Já à implementação de funções não-linearmente separáveis, como OU EXCLUSIVO, são necessárias pelo menos duas camadas de portas de limiares lineares.

2) Portas de limiar quadráticas

Em casos de grandes valores para n, a relação entre o número de funções linearmente separáveis e o número total de funções booleanas tende a zero, restringindo, assim, a utilização das portas lineares. A solução é aumentar a capacidade computacional das mesmas com a ajuda de uma porta de limiar quadrática.

$$y = \begin{cases} 1 & \sum w_i x_i + \sum w_{ij} x_i x_j \geq \theta \\ 0 & \sum w_i x_i + \sum w_{ij} x_i x_j < \theta \end{cases} \qquad \text{Eq.(2.14)}$$

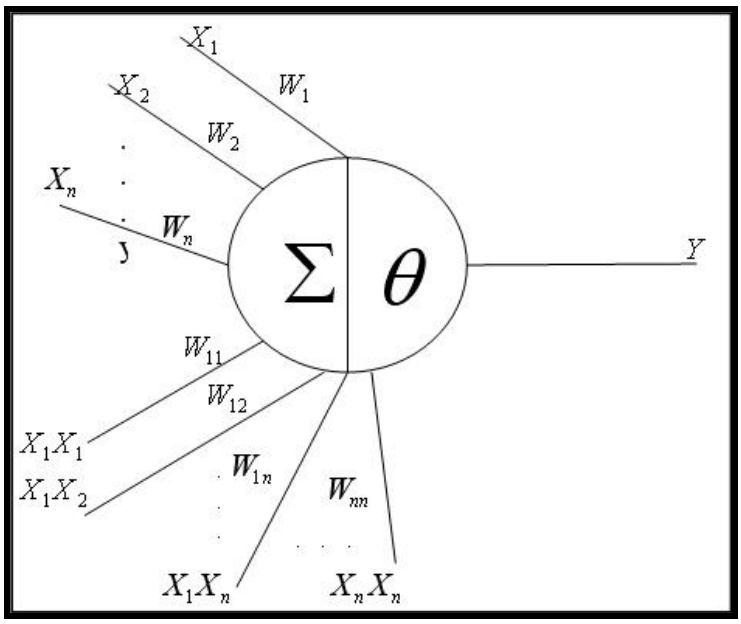


Figura 2.11: Porta de Limiar Quadrática.

Fonte: Autor. Baseado em [Braga, 2000].

O número maior de parâmetros livres ajustáveis, conforme (2.14), confere à porta de limiar quadrática um maior poder computacional. Conforme é facilmente percebido, a equação (2.14) possui termos quadráticos em cada uma das variáveis de entrada, além de termos com produtos cruzados entre cada uma delas. Um exemplo de região de decisão para duas classes utilizando a porta de limiar quadrática é apresentado na figura 2.12. Observe o nível de complexidade dessa região, se comparada com um separador linear comum.

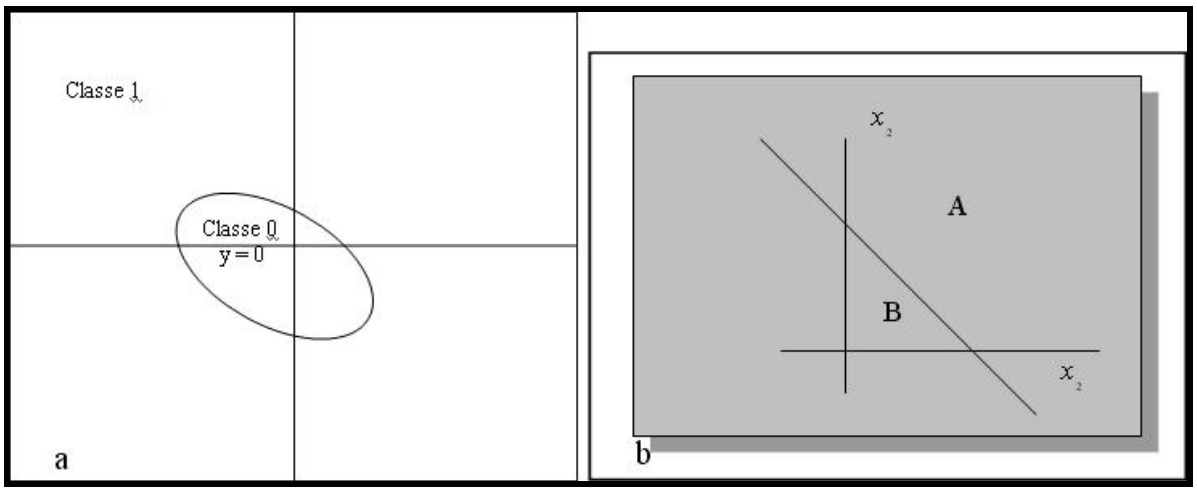


Figura 2.12: a) Exemplo de região de decisão para duas classes para uma porta de limiar quadrática. b) Exemplo de região de decisão para duas classes para um sistema linear.

Fonte: Autor. Baseado em [Braga, 2000]

2.5.2. Perceptrons de camada única

A topologia original do perceptron descrita por Rosenblatt era composta por unidades de entrada, por um nível intermediário formado pelas unidades de associação e por um nível de saída formado pelas unidades de resposta. Essa topologia original, composta por três níveis, forma o perceptron de camada única, visto que apenas o nível de saída possui propriedades adaptativas. O primeiro nível consiste basicamente em unidades sensoras (retina) e as unidades intermediárias de associação são nodos MCP com pesos fixos, definidos antes do início do treinamento.

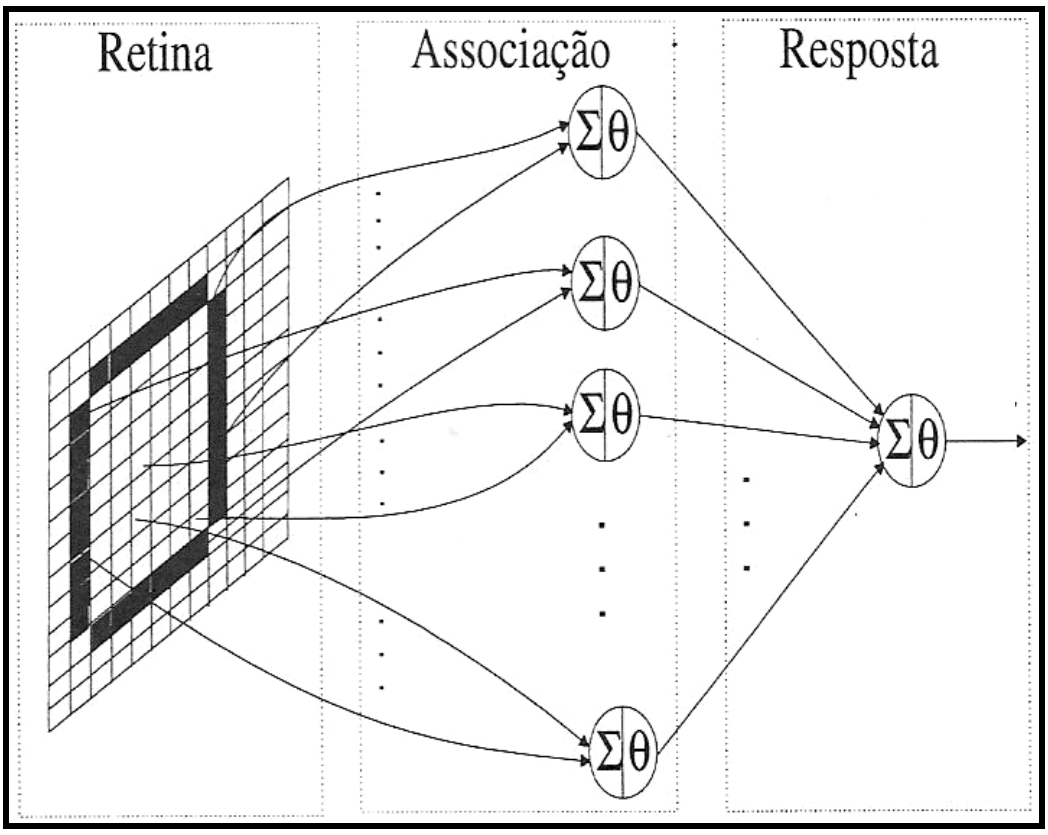


Figura 2.13: Topologia de um perceptron simples com uma única saída.

Fonte: [Braga, 2000].

2.5.2.1. O algoritmo de aprendizado do perceptron

A regra de aprendizado do perceptron permite a adaptação dos seus pesos de forma que a rede execute a sua tarefa. Em outras palavras, é uma regra para a atualização de seus pesos baseada no estado da rede neural artificial em uma posição de tempo discreto *t* (velho) e uma posição de tempo *t+1* (novo, ou vigente).

De forma genérica, o que se visa numa RNA durante seu processo de aprendizagem é a obtenção do incremento Δw a ser aplicado ao vetor de pesos **w** de forma que o seu novo

valor atualizado $w(t+1) = w(t) + \Delta w$ gere saídas mais próximas dos valores desejados para a RNA.

Em [Braga, 2000, p.36] é proposta uma equação específica, que leva a uma regra de atualização dos pesos do perceptron. Essa equação pode ser demonstrada a partir da consideração de um nodo arbitrário da camada de resposta de um perceptron e seus vetores de entrada *x'* e de pesos *w'*.

Como abordado anteriormente, análogo a um neurônio artificial padrão, a ativação do nodo de um perceptron é dada por:

$$\sum_i w'_i \, x'_i = \mathbf{w}' \cdot \mathbf{x}' \qquad \text{Eq.(2.15)}$$

onde $\mathbf{w}' \cdot \mathbf{x}'$ representa o produto interno entre **w'** e **x'**.

A condição para que um disparo de um neurônio ocorra é:

$$\mathbf{w}' \cdot \mathbf{x}' = \theta, \quad \text{ou} \quad \mathbf{w}' \cdot \mathbf{x}' - \theta = 0 \qquad \text{Eq.(2.16)}$$

onde θ representa o limiar de ativação (threshold), ou o bias *b(n)*. De acordo com os tópicos previamente abordados (figura 2.4 e equação 2.3), o neurônio como unidade de processamento pode ter o bias armazenado em seus vetores de entrada **x'** e peso **w'** no índice zero, sem prejuízo para o processamento ou cálculos da saída **y** do neurônio.

Dessa forma, a nova condição crítica de disparo passa a ser:

$$\mathbf{w} \cdot \mathbf{x} = 0 \qquad \text{Eq.(2.17)}$$

onde $\mathbf{w} = \{-\theta, w_1, w_2, ..., w_n\}^T$ e $\mathbf{x} = \{1, x_1, x_2, ..., x_n\}^T$.

Note que as entradas do limiar de ativação –θ e 1, ao serem multiplicadas resultam apenas no limiar de ativação (–θ).

Seja considerado agora o par de treinamento {x,d} para esse nó da rede em questão, onde *x* é o vetor de entrada do nodo, e *d* é a saída desejada para a entrada *x*. A saída da rede poderá ser definida simplesmente como *y*. O erro devido à saída atual pode será definido como:

$$e = d - y \qquad 2.18$$

Para o caso do perceptron cujas saídas estão sempre contidas nos conjuntos $y \in \{0,1\}$ e $d \in \{0,1\}$ só há duas possibilidades possíveis para as quais $e \neq 0$:

d = 1 e *y = 0*,
ou
d = 0 e *y = 1* Eq.(2.19)

Para o primeiro caso de (2.19), onde (d = 1 e y = 0), temos que $e = 1 - 0 \rightarrow e = 1$ e $\mathbf{w} \cdot \mathbf{x} < 0$, já que y = 0. Isso implica em:

$$\|\mathbf{w}\| \cdot \|\mathbf{x}\| \cos(\mathbf{w}, \mathbf{x}) < 0 \qquad \text{Eq.(2.20)}$$

Conseqüentemente, $\cos(\mathbf{w},\mathbf{x}) < 0$ e $\alpha(\mathbf{w},\mathbf{x}) > 90°$, onde α é o ângulo entre os vetores.

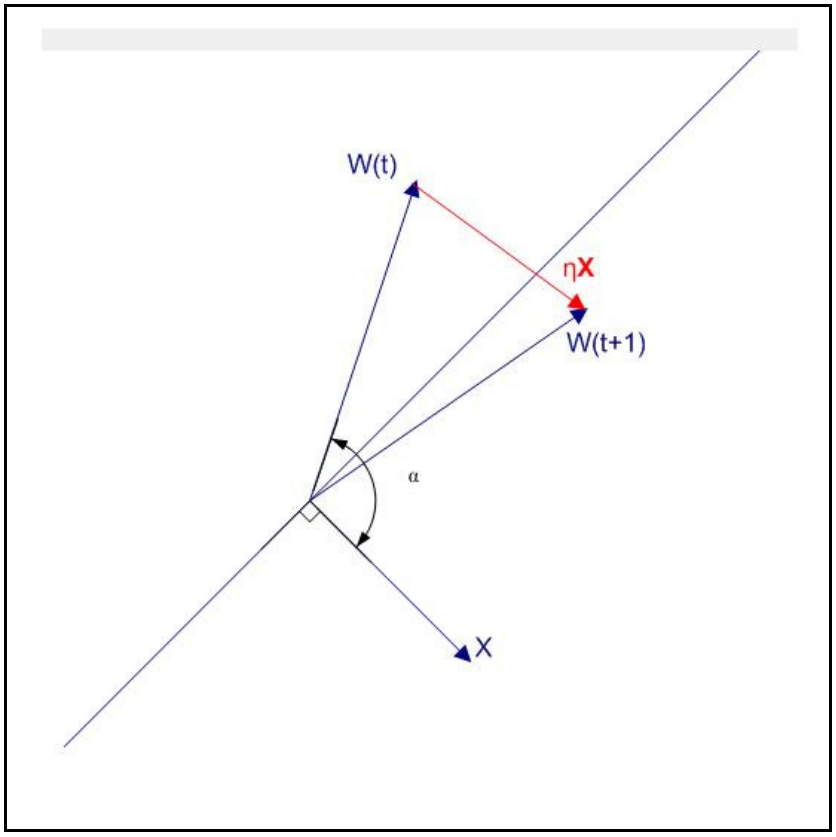


Figura 2.14: Situação relativa dos vetores w e x, em hipótese descrita por (2.20).

Fonte: O autor.

Observando a figura 2.14, percebe-se que uma modificação plausível para o vetor **w** em busca da solução seria somá-lo a um vetor que estivesse na direção de **x**, como o vetor η**x**. No caso, o vetor η**x** funciona como um direcionador para o vetor **w**, apontando sempre para a direção e sentido de **x**. Se η for o inverso do módulo de **x**, o vetor η**x** será um vetor unitário em módulo, servindo apenas para indicar as propriedades de **x** para o vetor de pesos **w** se orientar.

Assim,

$$\Delta \mathbf{w} = \eta \mathbf{x} \qquad \text{Eq.(2.21)}$$

e

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \eta \mathbf{x} \qquad \text{Eq.(2.22)}$$

Para que a equação da regra de atualização de pesos do perceptron fique completa, ainda falta definir o sentido do vetor η**x**, pois, dependendo do caso, ao invés de convergir para um resultado desejado, a saída *y* poderá divergir para o oposto do valor desejado. Uma forma prática de definir o sentido do vetor η**x** é utilizar o erro *e* como orientador: caso seu valor seja diferente de zero, aponta para o sentido negativo ou positivo dependendo de seu resultado. Combinando o erro *e* com a equação 2.22 obtém-se a regra de atualização dos pesos:

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \eta e \mathbf{x}(t), \qquad \text{Eq.(2.23)}$$

onde η funciona como um parâmetro da taxa de aprendizagem.

A regra de atualização é abrangente e pode ser generalizada, sendo utilizada também no segundo caso da equação 2.19.

2.5.2.2. Teorema de convergência do perceptron

A regra de convergência do perceptron, definida em (2.23), tem como objetivo a busca da solução para um único vetor de entrada **x**. Porém, quando mais de um vetor é apresentado à rede e várias atualizações sucessivas são realizadas para cada um deles,

modificações em **w** para um vetor de entrada $\mathbf{x}^i$ qualquer podem afetar a resposta da rede para um outro vetor de entrada $\mathbf{x}^j$. Assim, como o objetivo do perceptron é a busca de um único vetor que seja solução para o problema de classificação dos dados, há dependência entre operações sucessivas de atualização de pesos. O objetivo do teorema de convergência do perceptron é demonstrar que a regra de atualização de pesos do perceptron sempre leva a uma solução em um tempo finito caso os vetores do conjunto de treinamento sejam pertencentes a duas classes linearmente separáveis.

Considere a estrutura exibida na figura 2.15. Nesse modelo o bias *b(n)* é tratado como um peso sináptico, acionado por uma entrada fixa igual a +1 ambos, respectivamente, contidos nos elementos $\boldsymbol{w_0}$ e $\boldsymbol{x_0}$ dos vetores de peso e entrada. Dessa forma os vetores de entrada e pesos podem ser definidos, respectivamente como:

$$\mathbf{x}(n) = [1, x_1(n), x_2(n), ..., x_m(n)]^T$$
e
$$\mathbf{w}(n) = [b(n), w_1(n), w_2(n), ..., w_m(n)]^T \qquad \text{Eqs.(2.24)}$$

onde *n* representa o passo de iteração na aplicação do algoritmo.

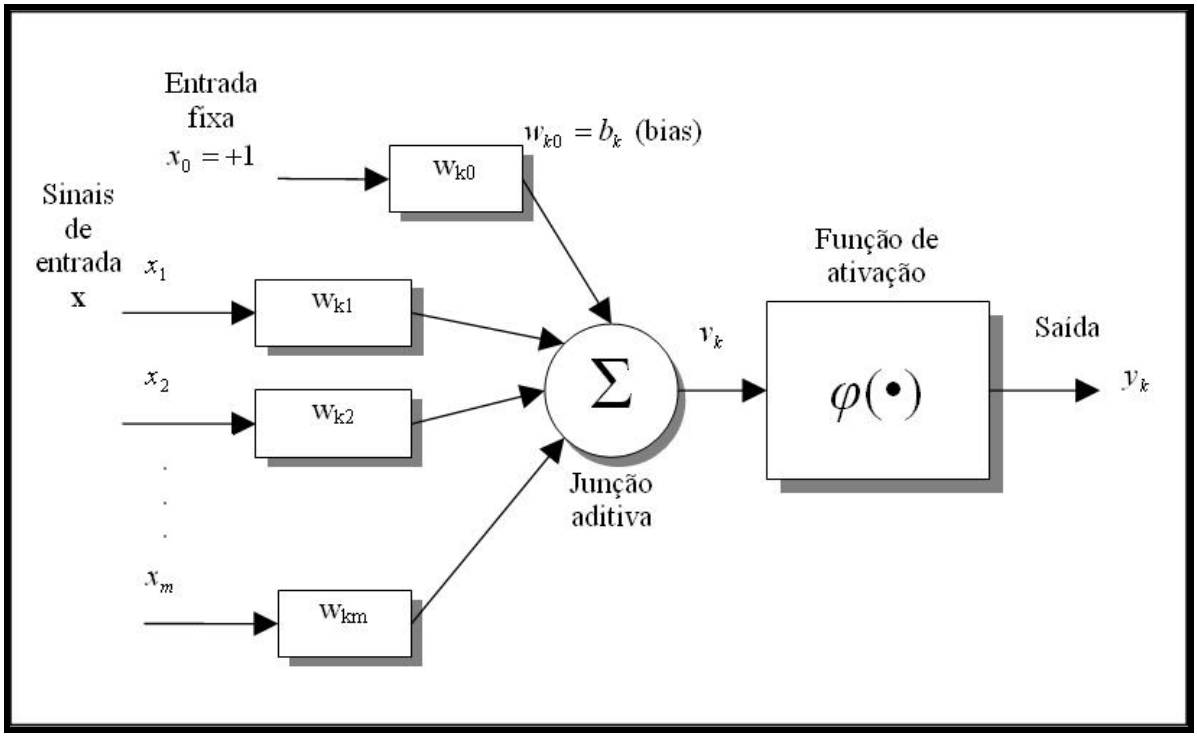


Figura 2.15: Modelo de combinador linear com bias aplicado em sua estrutura, detalhado por Haykin.

De forma correspondente, a saída do combinador linear pode ser escrita na forma compacta:

$$v(n) = \sum_{i=0}^{m} w_i(n)x_i(n) = \mathbf{w}^T(n)x(n) \qquad \text{Eq.(2.25)}$$

Para *n* fixo, a equação $\boldsymbol{w}^T\boldsymbol{x} = 0$, traçada em um espaço m-dimensional (traçada para um bias predeterminado) com coordenadas $x_1, x_2, \ldots x_m$, define um hiperplano como a superfície de decisão entre duas classes diferentes de entrada.

Como condição do funcionamento do perceptron, as duas classes C1 e C2 devem ser linearmente separáveis. Conforme abordado anteriormente, isso significa que os padrões a serem classificados devem estar suficientemente separados entre si no plano euclidiano para que seja possível traçar um hiperplano entre os grupos. A figura 2.16 ilustra os casos onde as classes C1 e C2 são linearmente separáveis (figura 2.16a) e onde as classes C1 e C2 não são linearmente separáveis (figura 2.16b).

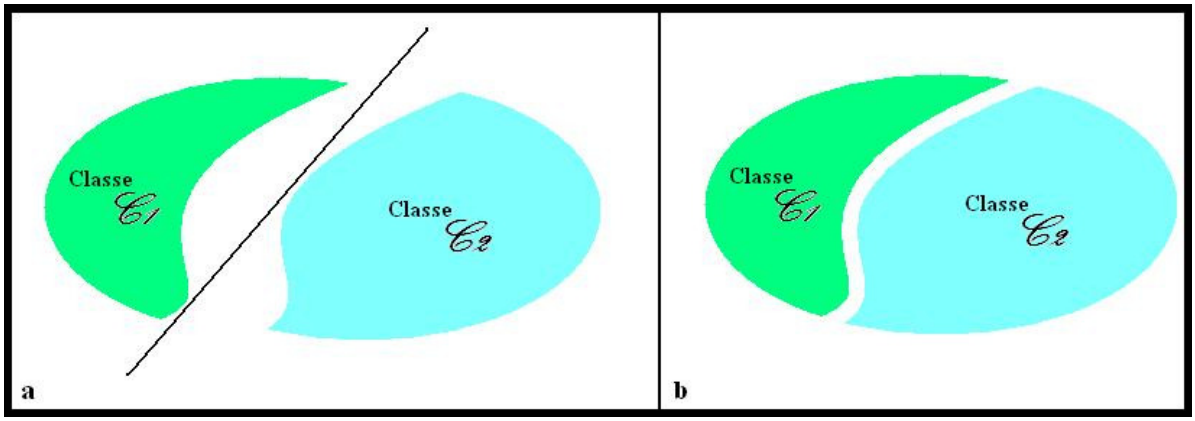


Figura 2.16: a) Um par de grupos linearmente separáveis por um hiperplano. b) Um par de grupos não linearmente separáveis.

Fonte: O autor.

Supondo que as variáveis de entrada do perceptron originem-se de duas classes linearmente separáveis, como 2.16a, define-se os subconjuntos de vetores de treinamento H1 e H2 como sendo, respectivamente, $\mathbf{x}_1(1)$, $\mathbf{x}_1(2)$, ... e $\mathbf{x}_2(1)$, $\mathbf{x}_2(2)$, ..., pertencentes às classes C1 e C2. A união de H1 e H2, define H, o conjunto de treinamento completo. O processo de treinamento envolve o ajuste do vetor de pesos **w** de tal forma que as duas

classes C1 e C2 sejam linearmente separáveis, obedecendo às regras estabelecidas em (2.26):

$\mathbf{w}^T\mathbf{x} > 0$,
para todo vetor de entrada **x** pertencente à classe C1
$\mathbf{w}^T\mathbf{x} \leq 0$,
para todo vetor de entrada **x** pertencente à classe C2 — Eqs.(2.26)

A escolha de $\mathbf{w}^T x = 0$ na segunda linha de (2.26) foi meramente arbitrária. O objetivo do treinamento para o perceptron elementar é, então, encontrar um vetor de pesos **w** ideal que satisfaça às duas expressões em (2.26).

O algoritmo de adaptação dos pesos para a separação das classes C1 e C2 será modificado durante a atualização dos pesos, em decorrência de (2.23), conforme as equações de (2.27):

$\mathbf{w}(t+1) = \mathbf{w}(n) - \eta(n)\mathbf{x}(n)$ se $\mathbf{w}^T\mathbf{x} > 0$ e **x***(n)* pertencer à classe C2
$\mathbf{w}(t+1) = \mathbf{w}(n) + \eta(n)\mathbf{x}(n)$ se $\mathbf{w}^T\mathbf{x} \leq 0$ e **x***(n)* pertencer à classe C1 — Eqs.(2.27)

onde $\eta(n)$ é o parâmetro da taxa de aprendizagem da iteração *n*.

Se $\eta(n) = \eta > 0$, onde η for uma constante independente da iteração *n*, temos uma *regra de adaptação com incremento fixo* para o perceptron.

Primeiramente será comprovada a convergência de uma regra de adaptação com incremento fixo para a qual $\eta = 1$. De forma geral, o valor de η não é importante, desde que

seja positivo. Um valor de $\eta \neq 1$ simplesmente escala os vetores de padrões sem afetar a sua separabilidade, conforme discutido anteriormente (na discussão da figura 2.14).

Suponha-se uma prova apresentada para a condição inicial $\mathbf{w}(0) = \mathbf{0}$. Suponha-se que $\mathbf{w}^T(n)\mathbf{x}(n) < 0$ para $n = 1, 2, ...$, e que o vetor de entrada $\mathbf{x}(n)$ pertença ao subconjunto H1. Isto fere a segunda condição de (2.26), classificando incorretamente os vetores $\mathbf{x}(1), \mathbf{x}(2), ...$

Com a constante $\eta(n) = 1$ pode-se usar a segunda linha de (2.27) para escrever:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mathbf{x}(n) \qquad \text{Eq.(2.28)}$$

para $\mathbf{x}(n)$ pertencente à classe C1.

Com a condição suposta de $\mathbf{w}(0) = \mathbf{0}$, pode-se resolver iterativamente esta equação para $\mathbf{w}(n+1)$ obtendo:

$$\mathbf{w}(n+1) = \mathbf{x}(1) + \mathbf{x}(2) + \ldots + \mathbf{x}(n) \qquad \text{Eq.(2.29)}$$

Como as classes C1 e C2 são assumidas como sendo linearmente separáveis, existe uma solução $\mathbf{w}_0$ para a qual $\mathbf{w}^T\mathbf{x}(n) > 0$ para os vetores $\mathbf{x}(1),..., \mathbf{x}(n)$ pertencentes ao subconjunto H1. Para uma solução fixa, pode-se definir um número α tal que

$$\alpha = \min[\mathbf{w}_0^T \mathbf{x}(n)] \qquad \text{Eq.(2.30)}$$

onde $\mathbf{x}(n) \in$ H1.

Ao multiplicar ambos os lados da equação em (2.29) por w_0^T, obtém-se:

$$\mathbf{w}_0^T\mathbf{w}(n+1) = \mathbf{w}_0^T\mathbf{x}(1) + \mathbf{w}_0^T\mathbf{x}(2) + \ldots + \mathbf{w}_0^T\mathbf{x}(n) \qquad \text{Eq.(2.31)}$$

Conseqüentemente, baseado em (2.30), tem-se:

$$\mathbf{w}_o^T\mathbf{w}(n+1) \geq n\alpha \qquad \text{Eq.(2.32)}$$

Pode-se concluir também, pela inequação de Cauchy-Schwarz [Braga, 2000, p. 40] e [Haykin, 2001, p. 166] que dados dois vetores $\mathbf{w}_0$ e $\mathbf{w}(n+1)$, a expressão de (2.32) se torna:

$$\|\mathbf{w}_0\|^2 \|\mathbf{w}(n+1)\|^2 \geq [\mathbf{w}_0^T\mathbf{w}(n+1)]^2 \qquad \text{Eq.(2.33)}$$

Sendo assim, a inequação (2.32) combinada com (2.33) se tornam:

$$\|\mathbf{w}_0\|^2 \|\mathbf{w}(n+1)\|^2 \geq n^2\alpha^2$$
$$\|\mathbf{w}(n+1)\|^2 \geq \frac{n^2\alpha^2}{\|\mathbf{w}_0\|^2} \qquad \text{Eq.(2.34)}$$

A tradução de (2.34) é que a norma ao quadrado do vetor de pesos varia pelo menos de forma quadrática com o número de iterações *n*. Para haver convergência, é necessário

que a norma euclidiana de $\mathbf{w}(n+1)$ seja limitada, o que é demonstrado seguindo outra linha de raciocínio.

Como definido para o teorema de convergência do perceptron, $\eta = 1$. Assim, a equação de (2.23), a regra de atualização dos pesos do perceptron, pode ser descrita como:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + e\mathbf{x}(n) \qquad \text{Eq.(2.35)}$$

Obtendo-se a norma euclidiana e elevando ambos os lados ao quadrado em (2.35), obtém-se que:

$$\begin{aligned} \|\mathbf{w}(n+1)\|^2 &= \|\mathbf{w}(n) + e\mathbf{x}(n)\|^2 \\ &= \|\mathbf{w}(n)\|^2 + e\|\mathbf{x}(n)\|^2 + 2e\mathbf{w}^T(n)\mathbf{x}(n) \end{aligned} \qquad \text{Eq.(2.36)}$$

Conforme abordado nas expressões de (2.19), o erro e pode assumir dois valores possíveis, $e = 1$ ou $e = -1$. Como foi suposto para o teorema de convergência do perceptron, $\mathbf{w}^T(n)\mathbf{x}(n) < 0$ implicando em $e = 1$. Assim:

$$\|\mathbf{w}(n)\|^2 + \|\mathbf{x}(n)\|^2 \geq \|\mathbf{w}(n)\|^2 + e\|\mathbf{x}(n)\|^2 + 2e\mathbf{w}^T(n)\mathbf{x}(n) \qquad \text{Eq.(2.37)}$$

o que também é verdade para a situação em que $e = -1$.

Assim, podem-se concluir as relações descritas na equação (2.38):

$$\|\mathbf{w}(n+1)\|^2 \leq \|\mathbf{w}(n)\|^2 + \|\mathbf{x}(n)\|^2$$
$$\|\mathbf{w}(n+1)\|^2 - \|\mathbf{w}(n)\|^2 \leq \|\mathbf{x}(n)\|^2 \qquad \text{Eq.(2.38)}$$

Somando os *n* primeiros termos em ambos os lados da segunda relação descrita em (2.38):

$$\|\mathbf{w}(n+1)\|^2 \leq \sum_{i=1}^{n} \|\mathbf{x}(i)\|^2 \qquad \text{Eq.(2.39)}$$

De acordo com [Haykin, 2001, p. 167] e [Braga, 2000, p. 41], define-se β como:

$$\beta = \max_{i=1,2,\ldots,n} \|\mathbf{x}(i)\|^2 \qquad \text{Eq.(2.40)}$$

Pode-se escrever uma segunda condição para limite para o vetor $\|\mathbf{w}(n+1)\|$ através da relação:

$$\|\mathbf{w}(n+1)\|^2 \leq \beta n \qquad \text{Eq.(2.41)}$$

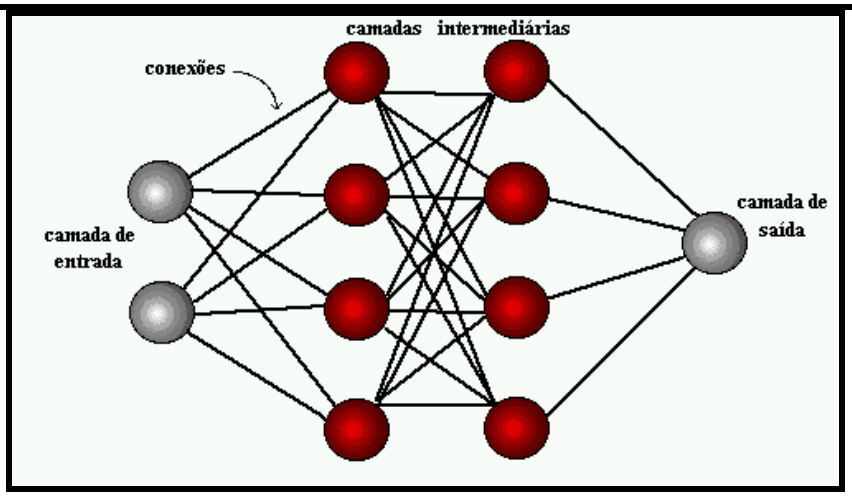
As duas condições para $\|\mathbf{w}(n+1)\|$ descritas pelas relações de (2.34) e (2.41) na verdade implicam a existência de um limite *t* para as operações de ajuste, já que a solução destas duas inequações resulta em

$$0 \leq n \leq \frac{\beta \|\mathbf{w}_0\|^2}{\alpha^2} \qquad 2.42$$

"*Portanto, pode-se concluir que o número de iterações é limitado por* $t = \frac{\beta \|\mathbf{w}_0\|^2}{\alpha^2}$ *o que implica finalmente que o algoritmo sempre converge em um tempo finito, como se queria demonstrar*" [Braga, 2000, p. 41].

2.5.3. Perceptrons de Múltiplas Camadas – MLP

Tipicamente, uma rede neural Perceptron que possui múltiplas camadas alimentadas adiante se consiste em uma rede Perceptron de Múltiplas Camadas (MPL – MultiLayer Perceptron) quando seus neurônios estão dispostos em conjuntos chamados camadas, da seguinte maneira:



[camada de entrada] → [camada intermediária ou oculta] → [camada de saída]

Figura 2.17: Exemplo da Estrutura de uma Rede MPL[6].

[6] Fonte: http://www.icmc.usp.br/~andre/research/neural/MLP.htm. Acesso em 12/12/2006.

Para que o erro seja levado em consideração em uma rede MLP, uma das formas é utilizar o algoritmo *error backpropagation* (retropropagação de erro), onde o erro é propagado para trás, camada a camada, informando aos neurônios de cada camada o desvio de sua resposta em relação à desejada.

Em [HAYKIN, 2001], está definido que uma das características das funções de ativação dos neurônios das redes MLP, para que se possa aplicar o algoritmo error backpropagation, é que devem possuir não linearidades suaves (serem diferenciáveis em qualquer ponto), o que é o caso da função sigmóide mostrada na figura 2.5c.

A adaptação da resposta de uma rede MLP está relacionada ao problema de atribuição de crédito, ou seja, como descobrir o quanto o neurônio de uma determinada camada é responsável pelas saídas certas ou erradas da rede [BRAGA, 2005]. Mais especificamente, deseja-se saber qual a parcela de "culpa" que cada neurônio tem pelos erros da rede [HAYKIN, 2001].

A rede MLP exemplificada na figura 2.17 é uma generalização. Dependendo do problema a ser resolvido, a rede pode ter uma ou mais camadas ocultas, e a quantidade de neurônios em cada uma das camadas também é definida conforme a descrição do problema abordado sugere. Esse fator, aliado à capacidade de aprendizagem proporcionado pelo algoritmo error backpropagation, proporciona um maciço poder computacional à rede MLP o que explica a sua larga utilização para a resolução de diferentes problemas em diferentes áreas.

A rede MLP funciona como um aproximador universal, ou seja, qualquer função ou problema, seja linearmente separável ou não, pode ser solucionado com a utilização de um Perceptron de multicamadas [ROCHA, 2006].

2.5.3.1 O algoritmo de retropropagação de erro

Basicamente, a aprendizagem com o algoritmo de retropropagação de erro, error backpropagation, consiste de dois passos através da rede: um para frente e um para trás. Primeiro, ao ser aplicado o vetor de entrada, um resultado é produzido (tendo seu efeito propagado através da rede, para frente). Nessa primeira fase os pesos sinápticos ainda são todos fixos. Depois, há o ajuste dos pesos sinápticos com base no erro detectado na saída da rede (tendo seu efeito propagado através da rede, em sentido contrário às conexões sinápticas).

O algoritmo de retropropagação de erro ajusta os erros das sinapses da penúltima camada de forma relacionada aos erros das sinapses da última. Os da antepenúltima relacionados aos da penúltima [ROCHA, 2006].

O objetivo geral do algoritmo de retropropagação de erro é fazer com que os resultados gerados pela rede tornem-se mais próximos da resposta desejada.

A taxa de aprendizagem, η, é um componente intrínseco ao algoritmo de retropropagação de erro. Ela determina a velocidade com a qual a rede convergirá ao mínimo erro quadrático. Se for uma taxa pequena, corre-se o risco de a convergência demorar muito ou de a rede ficar presa em um mínimo local. Se for uma taxa grande, a

oscilação provocada durante a convergência fará com que a rede tenha dificuldade em estabilizar-se.

Portanto, verifica-se que para que a RNA consiga trabalhar de forma eficiente, deve-se obter um valor adequado para a taxa de aprendizagem η .[7]

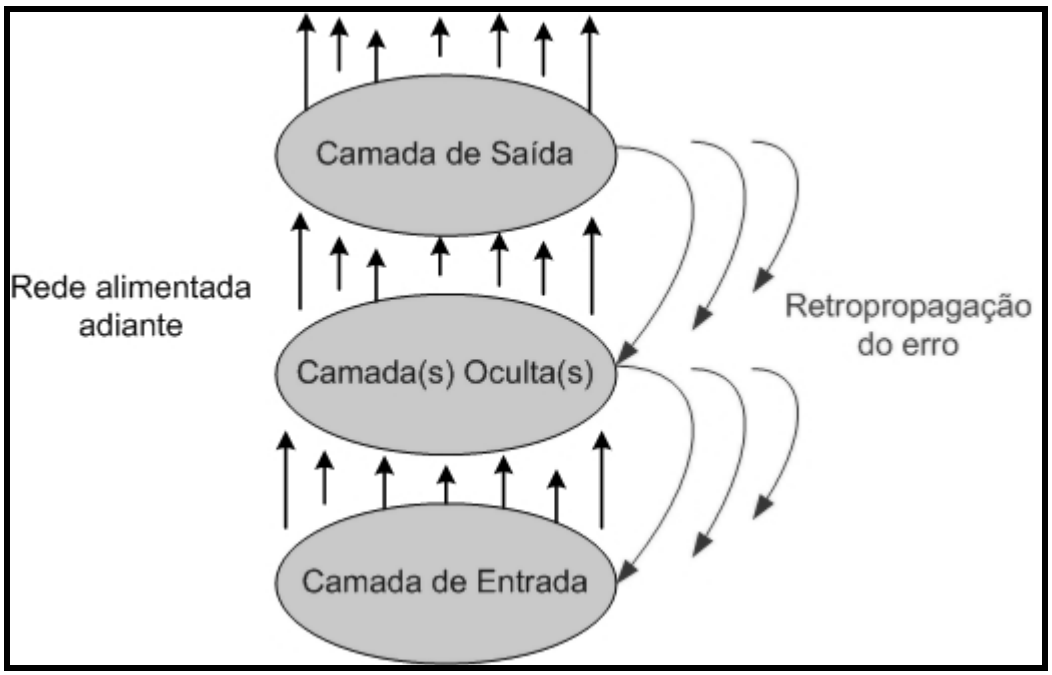


Figura 2.18: Esquema do funcionamento do algoritmo de retropropagação do erro.

Fonte: Autor.

[7] Na verdade, para que uma RNA MLP com *error backpropagation* funcione adequadamente deve-se ter ainda, um valor adequado também para o *bias*, conforme abordado anteriormente.

3. O Projeto: Controlando um elemento móvel utilizando Redes Perceptron Multicamadas

3.1 Descrição e objetivos do projeto

A finalidade do projeto é criar um ambiente de simulação baseado em programação para verificar a viabilidade da aplicação de Redes Neurais Artificiais para controlar um elemento móvel em ambiente bidimensional restrito e livre de barreiras, conforme a figura 3.1 abaixo.

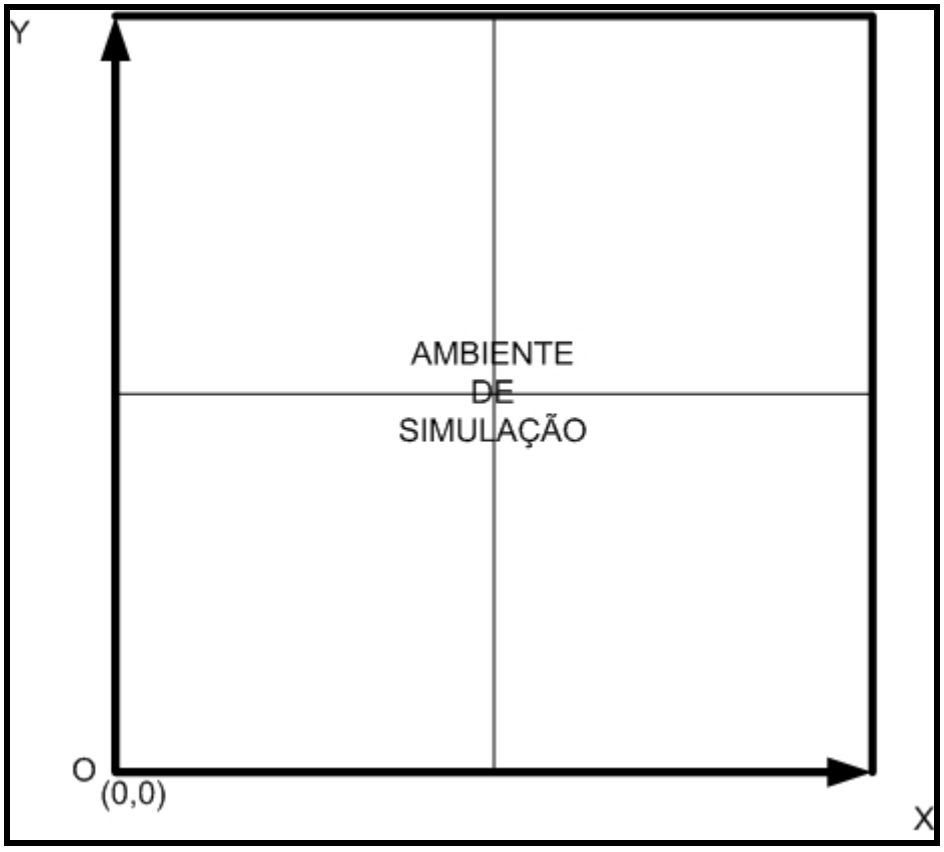


Figura 3.1: Ambiente de simulação proposto.

Fonte: Autor.

O elemento móvel deve possuir a capacidade de se movimentar até a saída do ambiente de forma independente, e sua margem de erro deve ser consideravelmente pequena, de forma que o objetivo ainda possa ser alcançado mesmo na presença do erro.

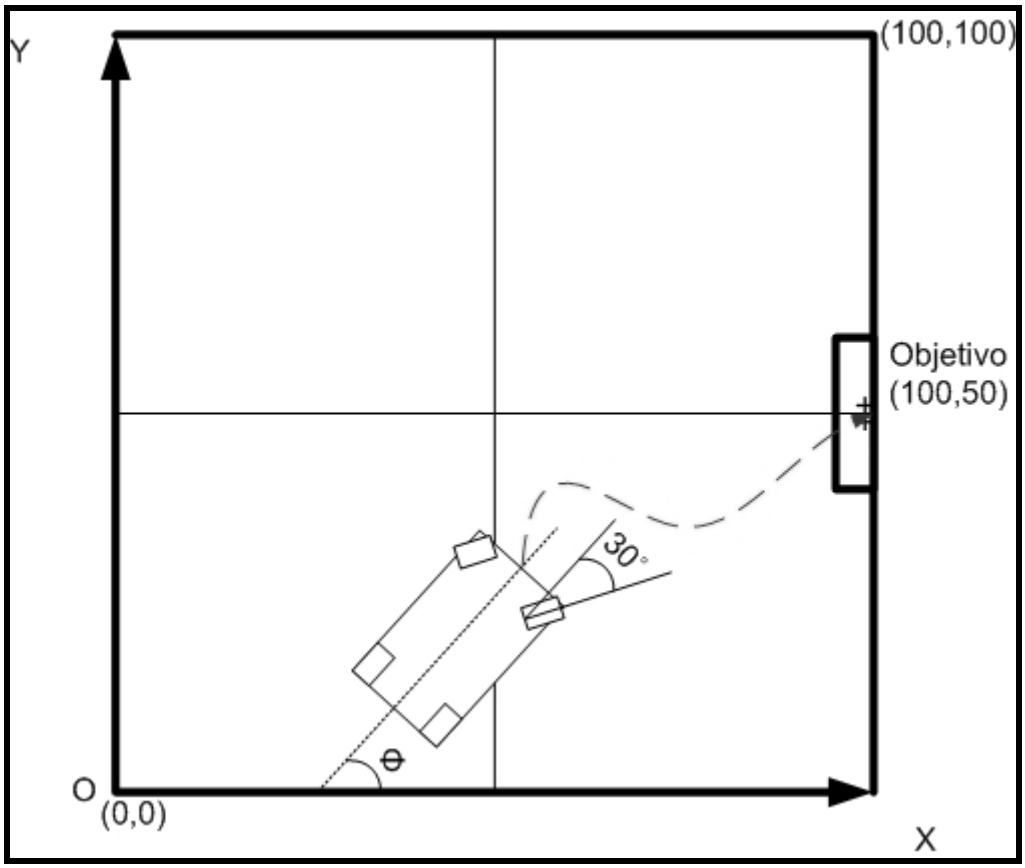


Figura 3.2: O elemento móvel e o seu objetivo na simulação: a saída do ambiente.

Fonte: Autor.

Obviamente, após identificar a provável direção da saída do ambiente, o móvel se deslocará até ela, sendo recalculada, após cada DELTA-S de distância percorrida, a nova direção que o móvel deverá seguir.

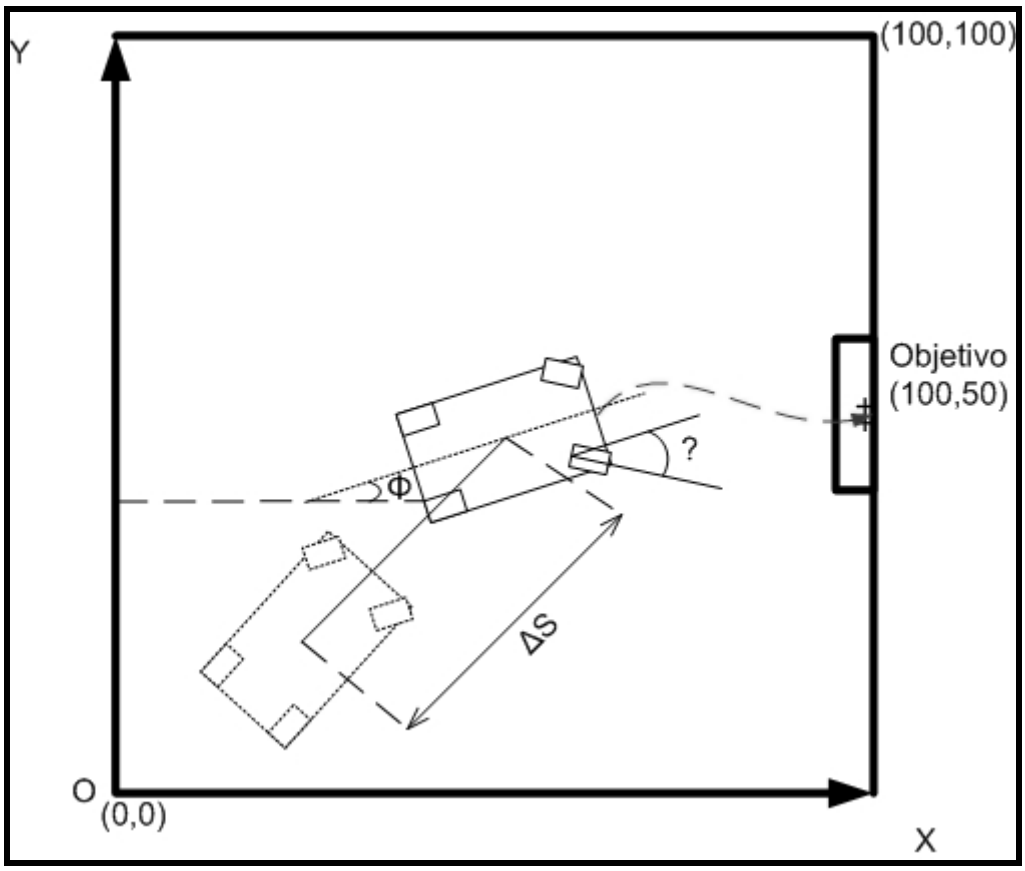


Figura 3.3: O elemento móvel consulta a nova direção à RNA a cada DELTA-S percorrido.

Fonte: Autor.

Da mesma forma como acontece na natureza, o sistema verifica, a cada passo percorrido, se a direção tomada está realmente correta, fazendo as alterações necessárias na direção tomada em tempo real para que o elemento móvel tenha condições de alcançar o seu objetivo.

Para facilitar a programação do ambiente de simulação e da RNA, definiu-se a área do ambiente simulado como sendo um quadrado de lado igual a 100 unidades de medida, e

a saída deverá estar sempre no ponto (50,100) com uma margem de aceitação de erro de ±4 unidades[8], conforme ilustrado pela figura 3.4.

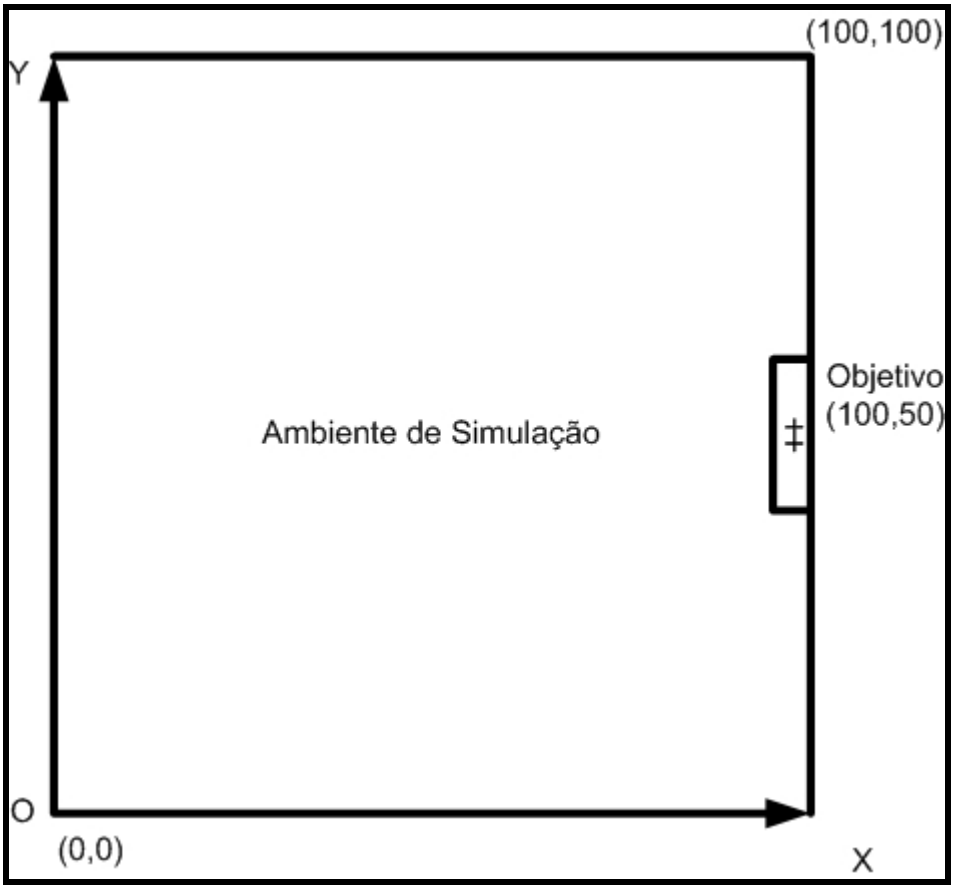


Figura 3.4: A saída do ambiente RNA.

Fonte: Autor.

[8] A margem de ±4 unidades se deve ao fato de o ponto (50,100) ser considerado como o centro de uma porta de largura igual a 8 unidades. Embora a saída do ambiente possua essa margem de aceitação, tal margem não será utilizada pela RNA no cômputo da direção a ser tomada pelo móvel, conforme será demonstrado adiante. Durante o cálculo da nova direção a ser tomada, será sempre utilizado o ponto exato da saída, (50,100).

3.2 Ferramentas e métodos utilizados para a elaboração do projeto

A linguagem selecionada para implementação do sistema de redes neurais foi a linguagem C++, auxiliada pela IDE C++ Builder.

O tipo de Rede Neural Artificial escolhida para controlar o elemento móvel foi a Rede de Perceptrons de Múltiplas Camadas (MultiLayer Perceptron – MLP) em regime de aprendizado supervisionado *off-line*, associados ao algoritmo de retropropagação de erro (error backpropagation), todos detalhados no capítulo 2. A escolha do tipo de RNA foi baseada na necessidade apresentada pelo problema: para a resolução desse problema de controle se faz necessário o uso de uma arquitetura de Rede Neural Artificial supervisionada recorrente, e a rede MLP com retropropagação de erro é um modelo simples e amplamente utilizado pela comunidade acadêmica que atende aos requisitos propostos.

3.2.1 O cálculo da saída desejada

Para treinar a rede neural é necessário fornecer exemplos, que consistem em uma matriz com entradas de posições (X,Y) e ângulos (PHI) aleatórios e seus respectivos valores de saídas desejados, conforme ilustrado na figura 3.5. Os valores devem ser aleatórios para que seja evitado o vício, efeito colateral da plasticidade das Redes Neurais Artificiais.

Conforme observado na figura 3.5, os pares de entrada são compostos por coordenadas e ângulos, que indicam a posição do móvel e a sua inclinação relativa ao eixo

das abscissas e à origem do ambiente, o ponto (0,0). As saídas desejadas são apenas valores de ângulos, indicativos da direção a ser tomada para alcançar a saída do ambiente.

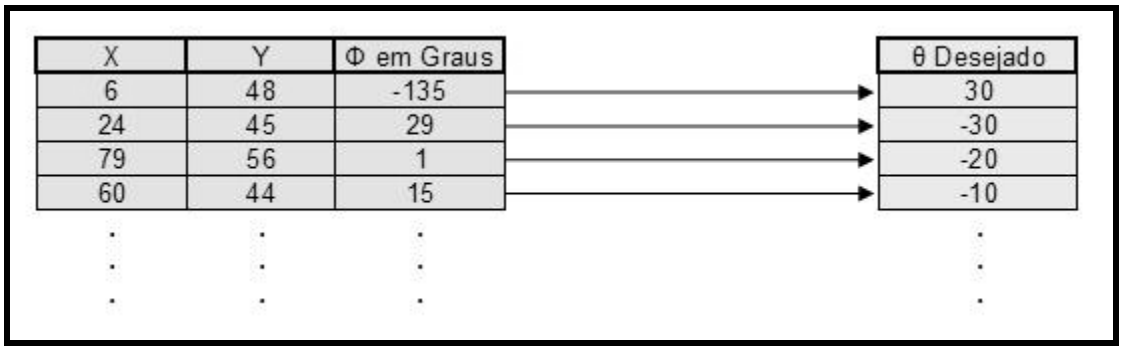


Figura 3.5: Matriz de associação: cada par de entrada [posição-ângulo] possui uma saída desejada. Os exemplos são aleatórios.

Fonte: Autor.

A cada iteração, isto é, a cada DELTA-S deslocado pelo móvel, a posição e a inclinação atualizadas do móvel são informadas à RNA para que uma nova direção seja calculada, garantindo o aumento da tolerância a falhas da rede.

Para criar uma matriz de exemplos para a RNA foi utilizada uma planilha do Microsoft Excel para gerar valores aleatórios para as entradas e calcular as saídas desejadas, utilizando os princípios básicos de trigonometria. A equação utilizada para calcular a nova direção θ a ser tomada, em graus, está descrita em 3.1:

$$\theta = ArcTg\left(\frac{50-Y}{100-X}\right) - \phi\,, \qquad \text{Eq. (3.1)}$$

onde θ representa a nova direção a ser tomada (em graus), Y representa a posição atual do elemento móvel em relação ao eixo das ordenadas, X representa a posição atual do

elemento móvel em relação ao eixo das abscissas e Φ representa o ângulo de inclinação do elemento móvel com relação à origem do plano cartesiano e ao eixo das abscissas.

Para o treinamento da RNA, a nova direção θ a ser tomada como base de correção da direção do móvel foi limitada a ±30˚, de acordo com a movimentação máxima das rodas de um veículo automotivo comum. Esse valor máximo foi adotado para que exista a possibilidade da Rede Neural Artificial treinada ser aplicada ao controle de veículos e outros elementos móveis com ângulo de movimentação dos elementos de direção maior que ±30˚. Observando outros autores de projetos semelhantes, verifica-se que o ângulo de movimento normalmente também está limitado a ±30˚, como o exposto em [CARDENAS, 2004]. A figura 3.6 exibe a limitação das rodas do elemento móvel proposto para o projeto.

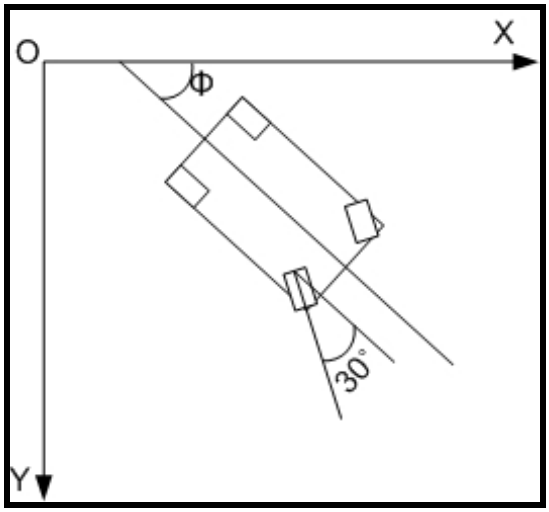


Figura 3.6: Elemento móvel: a movimentação das rodas está limitada a ±30˚.

Fonte: Autor.

A figura 3.7 exibe a medição do ângulo de movimento de uma roda de um veículo automotivo Celta, realizada para a verificação do ângulo de movimentação das rodas dianteiras de um veículo automotor. O resultado da medição foi de 57,79˚, o que representa uma variação de ±32,21˚ no ângulo de movimentação das rodas dianteiras. Levando-se em

consideração o erro do processo de medida, conclui-se que a aproximação utilizada de ±30˚ é adequada para a solução do problema.

Figura 3.7: Medição do ângulo de inclinação da roda dianteira de um veículo automotor[9].

3.2.2 A estrutura da RNA

Para definir a estrutura da RNA, isto é, o número de neurônios, o número de camadas, os valores para os pesos, os valores do bias, etc, foram efetuados testes exaustivos e a Rede Neural Artificial foi idealizada com 304 neurônios, sendo 3 na camada de entrada da rede, 300 nas camadas ocultas e 1 na camada de saída, conforme o seguinte esquema:

[9] A figura do carro superior foi obtida em: http://www2.uol.com.br/bestcars/carros/gm/celta/ac_2.jpg. Acesso em novembro de 2006.

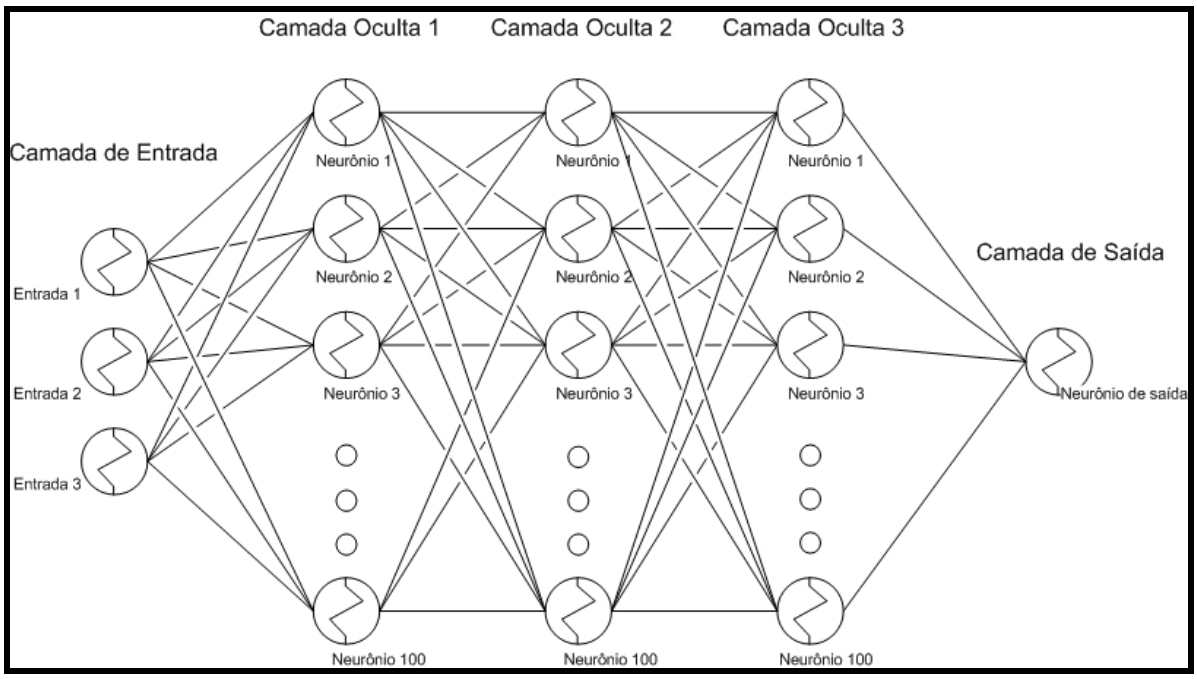


Figura 3.8: Topologia da RNA adotada: 304 neurônios de forma totalmente conectada.

Fonte: Autor.

A camada de entrada da rede possui três neurônios, que representam sensores de localização/navegação. Um neurônio é responsável por receber parâmetros relativos à posição X do móvel, outro para a posição Y e o último para o ângulo ϕ, que definem a posição e inclinação do elemento móvel em relação à origem do ambiente de simulação.

Os três neurônios de entrada são, na realidade, receptores para os valores de entrada indicativos da posição (X,Y) do móvel e seu ângulo em graus (ϕ) de entrada, relativo à origem do ambiente. Para que a rede trabalhe com valores escalares menores nessa camada, os neurônios recebem a posição e o ângulo em escala dez vezes menor.

As camadas ocultas da rede possuem 100 neurônios cada, totalizando 300 neurônios distribuídos em 3 camadas ocultas. A função das camadas ocultas é de fato executar

cálculos e aprender como controlar o móvel, a partir das informações fornecidas pela camada de entrada e dos resultados auferidos através da camada de saída.

A saída da rede neural fornece um ângulo de ajuste θ codificado, contido no intervalo [0,1]. Os valores possíveis para a decodificação do ângulo estão contidos no conjunto $\{-30°, -20°, -10°, 0°, 10°, 20°, 30°\}$, e os valores do conjunto estão associados a valores do intervalo da saída da rede neural. A responsabilidade da decodificação dos valores do ângulo pertence ao utilizador da RNA, que inclusive pode ou não optar pela utilização do conjunto de decodificação do ângulo proposto.

Intervalo	Ângulo associado
[0,0.1[	-30°
[0.1,0.275[	-20°
[0.275,0.425[	-10°
[0.425,0.625]	0°
]0.625,0.725]	10°
]0.725,0.9]	20°
]0.9,1]	30°

Figura 3.9: Associação da saída da RNA a um conjunto numérico.

Fonte: Autor.

O intervalo [0,1] como saída da rede neural foi escolhido pelo fato de a função sigmoidal trabalhar melhor com valores entre 0 e 1 (como é o caso da função logística).

3.2.3 O treinamento da Rede Neural Artificial

Para a construção de um conjunto de treinamento adequado para a rede, os seguintes fatores foram levados em consideração:

- **Tipo de Rede Neural adotada**: Rede MLP com retropropagação de erro;

- **Topologia da Rede Neural Artificial**: 304 neurônios artificiais dispostos em matrizes de [1x3], [3x100] e [1x1][10] neurônios nas camadas de entrada, ocultas e de saída, respectivamente, totalmente conectados;
- **Ângulo máximo de movimento das rodas do veículo**: ±30º;
- **Possibilidades diferentes de posicionamento do veículo no ambiente de simulação**: [360ºx100unidadesx100unidades] = 3.600.000 possibilidades de posicionamento.

Diversos treinamentos foram realizados, e chegou-se à conclusão de que um conjunto de 1.000 exemplos das 3.600.000 possibilidades é suficiente para executar o treinamento da Rede Neural Artificial.

Outros conjuntos com diferentes números de exemplos foram testados, mas se mostraram ineficientes, no caso de conjuntos com número de exemplos menor que 1.000, ou demandaram uma capacidade computacional elevada em que a relação [custo-computacional]/[benefício] não é vantajosa, no caso de conjuntos com número de exemplos maior que 1.000.

Outro ponto crítico avaliado para a construção do conjunto de exemplos foi a inserção de diversos exemplos não aleatórios no conjunto de treinamento, estrategicamente selecionados para garantir uma plasticidade mais uniforme à Rede Neural Artificial.

[10] Matrizes da RNA dispostas em Colunas e Linhas, no esquema de nomenclatura: [Coluna x Linha].

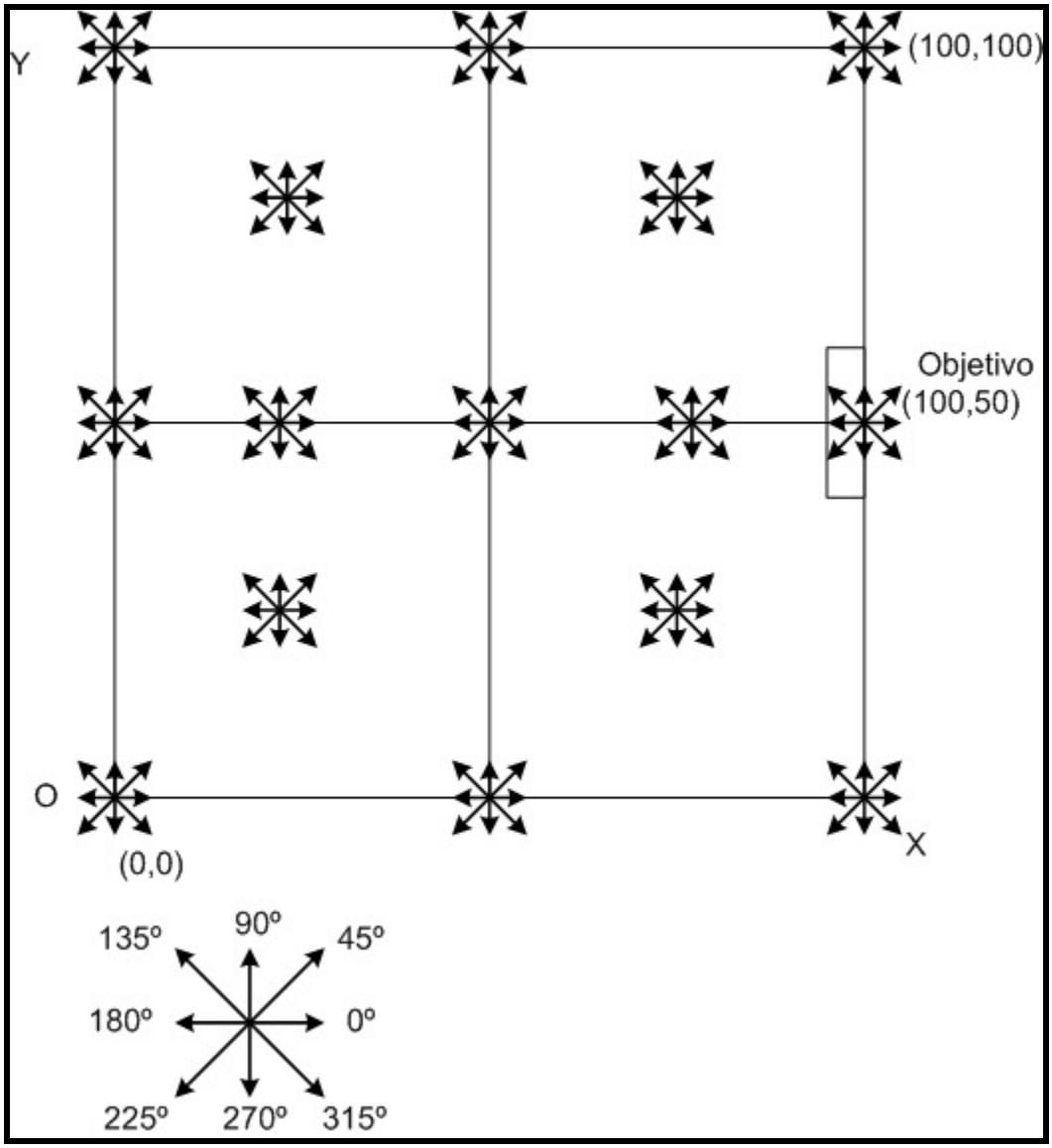


Figura 3.10: Pontos estratégicos selecionados no ambiente de simulação para compor parte do conjunto de exemplos de treinamento da RNA.

Fonte: Autor.

A figura 3.10 exibe os pontos estratégicos selecionados para o conjunto de treinamento e a tabela 3.1 mostra o conjunto de exemplos estratégicos utilizados para o treinamento. O conjunto completo de treinamento pode ser consultado no Apêndice A do presente trabalho.

Tabela 3.1: Conjunto de exemplos estratégicos utilizados para o treinamento da Rede Neural Artificial.

X	Y	Φ em Graus	Saída da Rede	θ Associado
0	0	0	1	30
0	0	45	0,2	-20
0	0	90	0	-30
0	0	135	0	-30
0	0	180	0	-30
0	0	225	1	30
0	0	270	1	30
0	0	315	1	30
0	0	360	1	30
50	0	0	1	30
50	0	45	0,5	0
50	0	90	0	-30
50	0	135	0	-30
50	0	180	0	-30
50	0	225	0	-30
50	0	270	1	30
50	0	315	1	30
50	0	360	1	30
99	0	0	1	30
99	0	45	1	30
99	0	90	0,5	0
99	0	135	0	-30
99	0	180	0	-30
99	0	225	0	-30
99	0	270	1	30
99	0	315	1	30
99	0	360	1	30
20	20	0	0,8	20
20	20	45	0,2	-20
20	20	90	0	-30
20	20	135	0	-30
20	20	180	0	-30
20	20	225	1	30
20	20	270	1	30
20	20	315	1	30
20	20	360	0,8	20
70	20	0	1	30
70	20	45	0,5	0
70	20	90	0	-30
70	20	135	0	-30
70	20	180	0	-30
70	20	225	0	-30
70	20	270	1	30
70	20	315	1	30
70	20	360	1	30
0	50	0	0,5	0
0	50	45	0	-30
0	50	90	0	-30
0	50	135	0	-30
0	50	180	0	-30
0	50	225	1	30
0	50	270	1	30
0	50	315	1	30
0	50	360	0,5	0
10	50	0	0,5	0
10	50	45	0	-30
10	50	90	0	-30
10	50	135	0	-30
10	50	180	0	-30
10	50	225	1	30
10	50	270	1	30
10	50	315	1	30
10	50	360	0,5	0
50	50	0	0,5	0
50	50	45	0	-30
50	50	90	0	-30
50	50	135	0	-30
50	50	180	0	-30
50	50	225	1	30
50	50	270	1	30
50	50	315	1	30
50	50	360	0,5	0
80	50	0	0,5	0
80	50	45	0	-30
80	50	90	0	-30
80	50	135	0	-30
80	50	180	0	-30
80	50	225	1	30
80	50	270	1	30
80	50	315	1	30
80	50	360	0,5	0
99	50	0	0,5	0
99	50	45	0	-30

Tabela 3.1: Conjunto de exemplos estratégicos utilizados para o treinamento da Rede Neural Artificial (cont.).

X	Y	Φ em Graus	Saída da Rede	θ Associado
99	50	90	0	-30
99	50	135	0	-30
99	50	180	0	-30
99	50	225	1	30
99	50	270	1	30
99	50	315	1	30
99	50	360	0,5	0
20	70	0	0,35	-10
20	70	45	0	-30
20	70	90	0	-30
20	70	135	0	-30
20	70	180	1	30
20	70	225	1	30
20	70	270	1	30
20	70	315	1	30
20	70	360	0,35	-10
70	70	0	0	-30
70	70	45	0	-30
70	70	90	0	-30
70	70	135	0	-30
70	70	180	1	30
70	70	225	1	30
70	70	270	1	30
70	70	315	0,65	10
70	70	360	0	-30
0	99	0	0	-30
0	99	45	0	-30
0	99	90	0	-30
0	99	135	0	-30
0	99	180	1	30
0	99	225	1	30
0	99	270	1	30
0	99	315	0,8	20
0	99	360	0	-30
50	99	0	0	-30
50	99	45	0	-30
50	99	90	0	-30
50	99	135	0	-30
50	99	180	1	30
50	99	225	1	30
50	99	270	1	30
50	99	315	0,5	0
50	99	360	0	-30
99	99	0	0	-30
99	99	45	0	-30
99	99	90	0	-30
99	99	135	1	30
99	99	180	1	30
99	99	225	1	30
99	99	270	0,5	0
99	99	315	0	-30
99	99	360	0	-30

O resultado do treinamento da Rede Neural, isto é, sua inteligência adquirida, deve ser armazenado em arquivos de texto formatados, para que o treinamento possa ser interrompido ou salvo quando desejado. Os arquivos de texto armazenam informações relativas aos pesos das sinapses e parâmetros de otimização do aprendizado da rede como o bias.

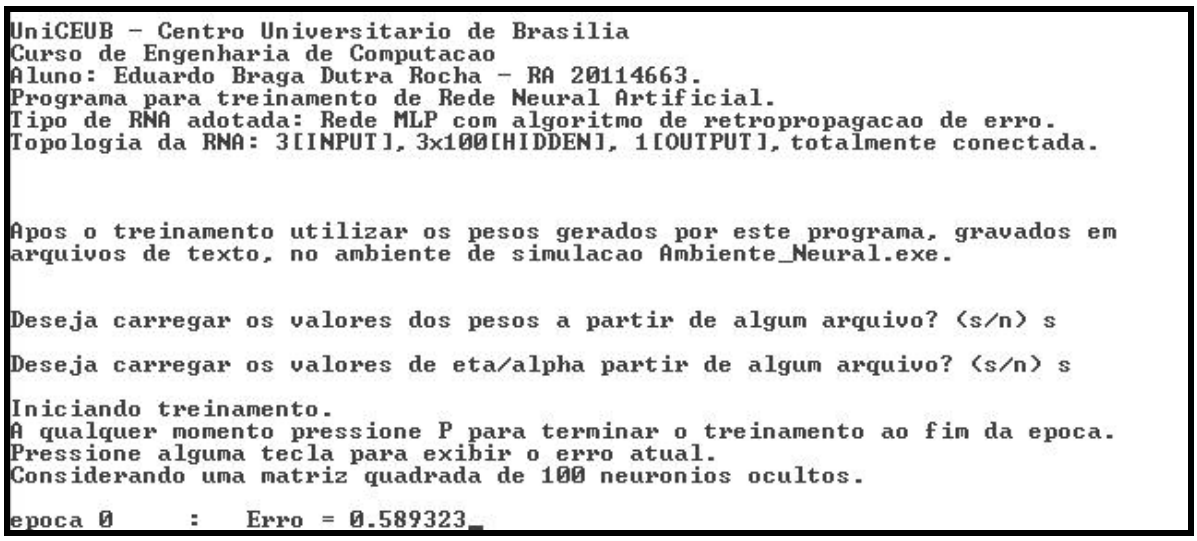


Figura 3.11: Tela do programa de treinamento da RNA.

De acordo com o especificado anteriormente foi elaborado um programa para o treinamento da RNA, cujo objetivo principal é o armazenamento do conhecimento em arquivos, para que possa ser carregado e utilizado no ambiente de simulação. A figura 3.11 exibe detalhe da tela do programa elaborado.

O programa de treinamento da RNA foi escrito em linguagem C e seu código fonte encontra-se detalhado no Apêndice B do presente trabalho.

3.2.4 Elaboração do ambiente de simulação

3.2.4.1 Composição do ambiente

A composição do ambiente de simulação tem como principais itens os descritos abaixo:

a) Elemento móvel: o móvel que será controlado pelo ambiente. Baseado em consultas periódicas à Rede Neural Artificial, adquire ciência sobre o ângulo necessário para aplicar nas rodas dianteiras. Sua velocidade é constante e mede:

$$[VelocidadeDoMovel] = \frac{1 \cdot R}{It}, \qquad \text{Eq.(3.1)}$$

onde R representa o Raio do veículo, igual a duas unidades de medida, e It significa Iteração.

No ambiente de simulação o elemento móvel é representado pela figura 3.12.

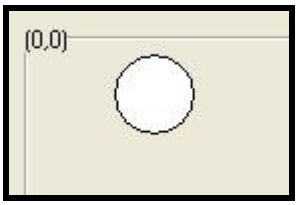


Figura 3.12: Elemento móvel.

b) *Grid*: representa o limite físico onde o elemento móvel pode se movimentar. É um quadrado de lado igual a 100 unidades de medida. O plano cartesiano tem origem no vértice superior esquerdo do *grid*, onde o eixo das abscissas tem seus valores positivos para a direita e o eixo das ordenadas tem seus valores positivos para baixo.

A saída do *grid* está centrada no ponto (100,50) e é representada por retângulo de 4 unidades de medida de comprimento por 2 unidades de medida de largura.

Por motivos estéticos e para aumentar a visibilidade, o *grid* foi dimensionado com escala 5 vezes maior que o limite físico utilizado no treinamento da RNA, medindo 500 unidades de lado. A escala aplicada ao *grid* não influencia na aplicação da RNA pois todos os valores relativos ao *grid* que possuem escala maior são divididos por 5 antes de serem repassados à rede.

No ambiente de simulação o *grid* é representado pela figura 3.13.

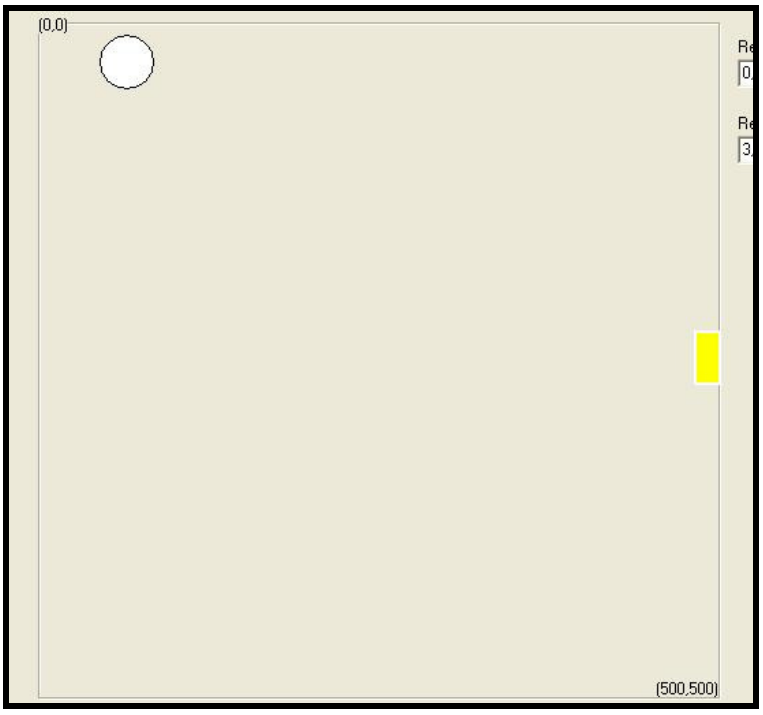


Figura 3.13: *Grid* do ambiente de simulação.

c) Caixas de texto indicativas da posição e direção do móvel: armazenam a posição atual (X,Y) do elemento móvel no *grid*, bem como sua atual inclinação Φ relativa ao eixo das abscissas. É possível utilizá-las para a inserção manual de posições.

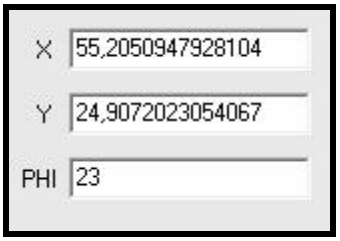


Figura 3.14: Caixas de texto indicativas da posição e direção do móvel.

d) Menu principal: sua maior utilidade é fornecer a opção de carregamento dos pesos (base de conhecimento) de um treinamento realizado com a Rede Neural Artificial, gravados em arquivos de texto.

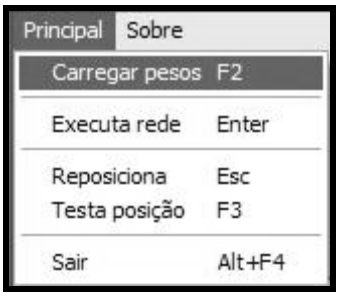


Figura 3.15: Menu principal e opção de carregamento dos pesos.

e) Opções de detecção de colisão e aproximação de resultados: a opção de detecção de colisão analisa a posição (X,Y) do móvel no *grid* a cada iteração de movimento do elemento móvel. Caso o móvel atinja alguma aresta do *grid*, é retornada uma mensagem de colisão e o móvel é reposicionado aleatoriamente no *grid*.

A opção de aproximação de resultados aumenta a área do objetivo do móvel, que se torna um retângulo de 4 unidades de medida de comprimento por 2 unidades de medida de largura.

A justificativa para utilização da aproximação de resultados é baseada no fato de que a posição do elemento móvel é medida no seu centro, desconsiderando o comprimento do raio do móvel. Assim, é possível utilizar a RNA treinada para atingir objetivos diferentes de uma saída levando em consideração a dimensão do móvel.

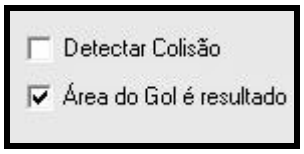


Figura 3.16: Opções de detecção de colisão e aproximação de resultados.

3.2.4.2 Esquema de funcionamento do elemento móvel

Durante a simulação, o elemento móvel possui velocidade constante e possui movimentos baseados em iterações. Conforme ilustrado na figura 3.3, o carro percorre um ΔS a cada iteração da rede.

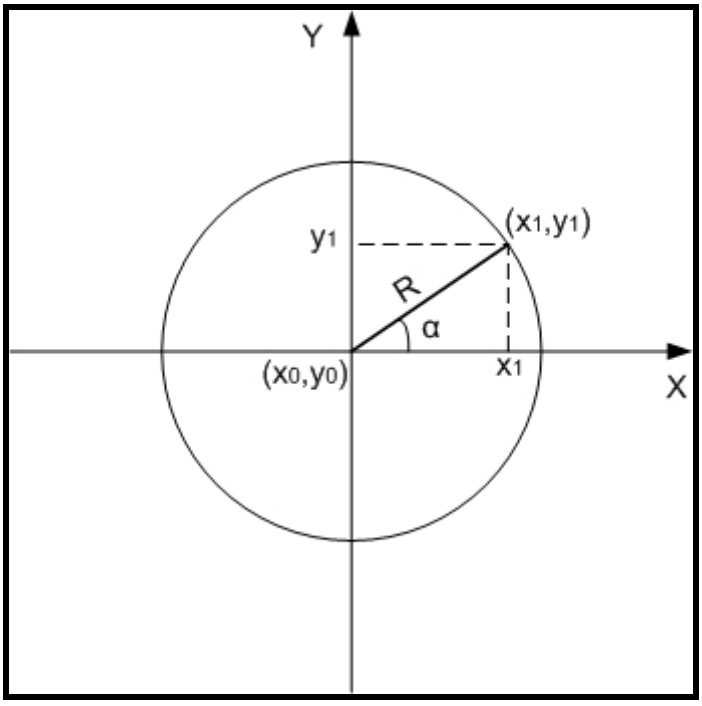


Figura 3.17: Uma circunferência de Raio R, utilizada como base para o esquema de movimento do móvel.

Se o carro for considerado com a forma de uma circunferência de raio R, conforme a figura 3.17 exibe, é possível calcular a sua nova posição (x_1,y_1) através do uso das seguintes equações:

$$x_1 = \cos\left(\alpha \cdot \frac{\pi}{180}\right) \cdot R + x_0 \qquad \text{Eq.(3.2)}$$

e

$$y_1 = sen\left(\alpha \cdot \frac{\pi}{180}\right) \cdot R + y_0 \qquad \text{Eq.(3.3)}$$

onde R representa o raio do veículo, que será utilizado também como incremento ou ΔS. Note que os ângulos não estão definidos em graus pois a biblioteca *math.c* da IDE de programação só trabalha com ângulos medidos em radianos.

3.2.4.3 Esquema de funcionamento do ambiente

O ambiente de simulação possui o seguinte esquema de funcionamento:

1) O Carregamento do Conhecimento

1.1) Os pesos são carregados a partir de arquivos de texto, provenientes de um treinamento anterior realizado através do programa de treinamento da RNA.

1.2) Os pesos alimentam uma estrutura de Rede Neural Artificial semelhante à usada para o treinamento. Essa RNA não possui retropropagação de erro, pois a ela não compete mais o aprendizado: seu objetivo é apenas a utilização do conhecimento armazenado nos pesos de um treinamento prévio para produzir uma saída relativa ao conhecimento carregado.

2) A Execução da Rede Neural Artificial

2.1) Os valores contidos nas caixas de texto indicativas da posição e direção do móvel são enviadas como parâmetro de entrada para a RNA, já respeitada a escala de 5:1 utilizada no *grid*.

2.2) A RNA produz a saída contida no intervalo [0,1].

2.3) Em seguida é efetuada a decodificação para o valor de ângulo associado, conforme a figura 3.8 ilustra.

2.4) o novo valor do ângulo θ (reajuste do ângulo das rodas) é somado ao Φ (ângulo atual) do móvel.

2.5) O móvel desloca-se no sentido da nova posição, conforme definido na seção anterior.

3) Os Valores são atualizados

3.1) Os valores contidos nas caixas de texto indicativas da posição e direção do móvel são atualizados.

3.2) A saída ideal para a rede é calculada paralelamente, com o intuito de informar qual a margem de erro da presente iteração.

4) O móvel verifica se o objetivo foi ou não atingido.

5) O processo é reiniciado a partir do passo 2.

A figura 3.18 exibe um diagrama do funcionamento do Ambiente de Simulação desenvolvido.

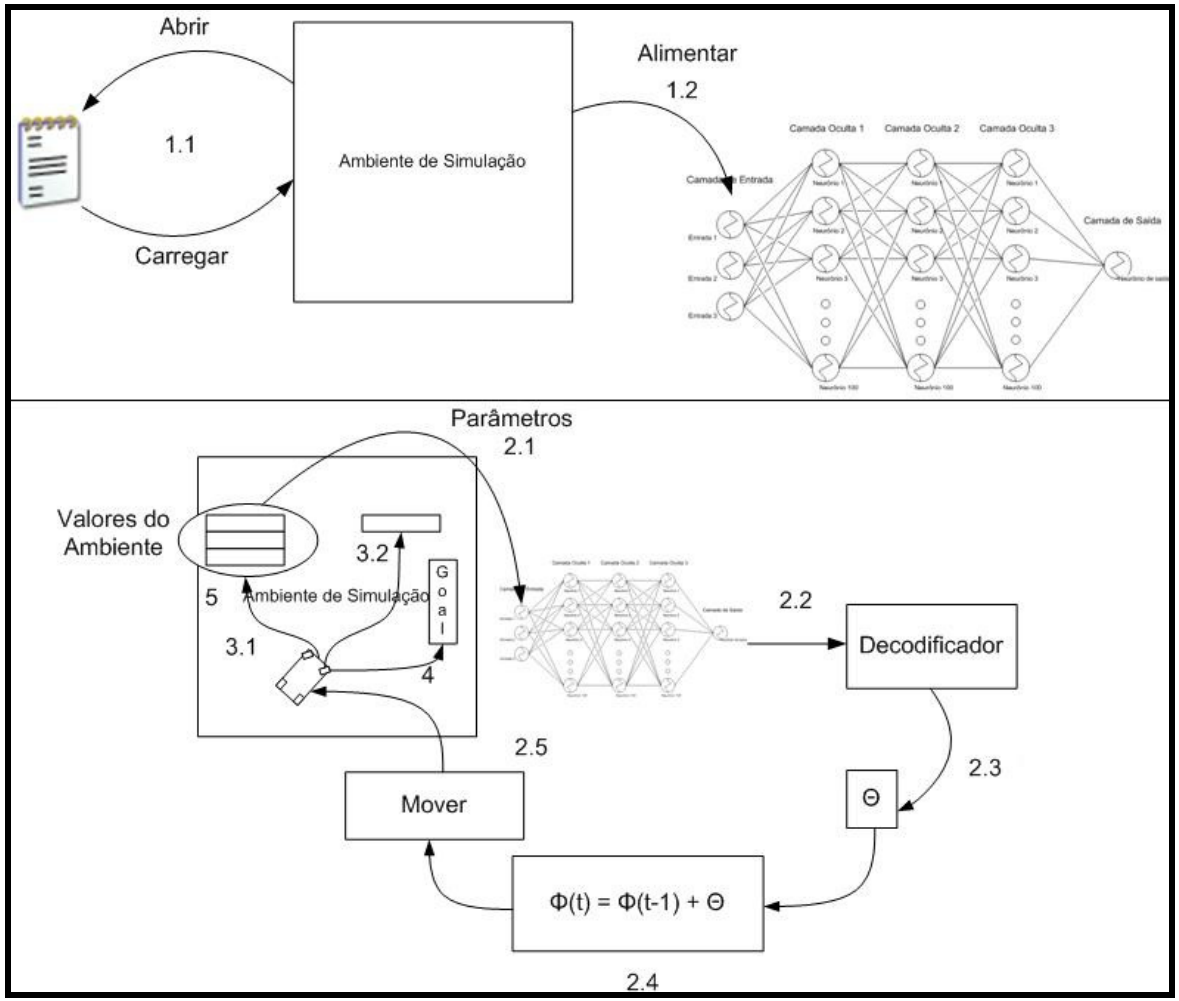


Figura 3.18: Diagrama do esquema de funcionamento do ambiente.

4. Resultados do Projeto

4.1 Resultados de treinamentos obtidos com a estrutura de RNA escolhida

Durante a execução do programa para o treinamento da RNA (detalhado no Apêndice B do presente trabalho) observou-se que, para um conjunto de 1.000 exemplos utilizados, conforme descrito na seção 3.2.3, a Rede Neural Artificial apresentou uma taxa de erro médio que, na maior parte do tempo, manteve-se no intervalo adimensional compreendido entre [0.2,0.9].

O treinamento durou cerca de 6 horas em um microcomputador dotado de processador INTEL Centrino Duo 1,66 GHz e foi interrompido por ter atingido resultado satisfatoriamente estável.

A figura 4.1 exibe a tela do programa de treinamento durante um dos treinamentos realizados com a rede.

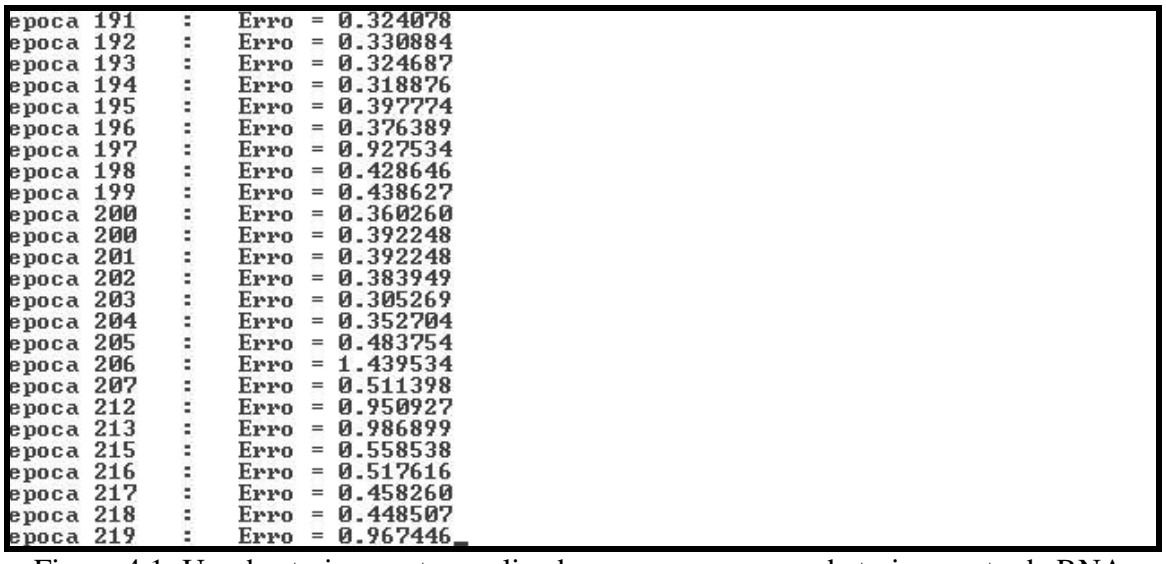


Figura 4.1: Um dos treinamentos realizados com o programa de treinamento da RNA.

4.2 Resultados obtidos no ambiente de simulação

O ambiente de simulação obteve desempenho satisfatório durante sua execução. Para avaliar os resultados obtidos com a aplicação da RNA desenvolvida ao ambiente de simulação foram efetuadas duas categorias de experimentos: a primeira, objetivando verificar a taxa de acerto da saída pelo móvel; a segunda, objetivando verificar a média do erro de direção tomada pelo móvel durante seu trajeto até a saída.

A primeira parte dos testes da primeira categoria de experimentos, da taxa de acerto da saída pelo móvel, foi realizada sob a situação mais próxima possível da realidade para a simulação de um veículo real: com a habilitação das opções de detecção de colisão e aproximação de resultados, descritas na seção 3.2.4.1. A segunda parte foi realizada de forma a obter o máximo desempenho possível da Rede Neural Artificial: ignorando colisões ocorridas com as arestas do *grid* do ambiente de simulação.

A justificativa para a divisão da categoria de experimentos em duas seções é a possibilidade do uso da Rede Neural Artificial para o controle de veículos, onde a colisão é significativa, ou a possibilidade do uso da Rede Neural Artificial para o controle de objetos cuja movimentação não é restrita, como um robô bípede ou quadrúpede, onde a colisão não influi no desenvolvimento de sua trajetória.

4.2.1 Testes da taxa de acerto da saída do ambiente pelo móvel

4.2.1.1 Testes realizados com detecção de colisão ativada

No total, foram realizadas duas baterias de testes para verificar a taxa de acerto da saída do ambiente pelo móvel, a primeira composta por 100 testes e a segunda por 200 testes. O objetivo de cada teste é verificar se o móvel atinge o seu objetivo, ao ser aleatoriamente posicionado no *grid* e utilizada a Rede Neural Artificial para movimentá-lo, levando em consideração os limites físicos do *grid* como parâmetros de falha dos testes. Em outras palavras, caso o elemento móvel atinja uma aresta do *grid*, o teste é avaliado como falho, e, caso o elemento móvel atinja o seu objetivo sem se deparar com nenhuma aresta do *grid*, o teste é avaliado como sucessivo. As opções de detecção de colisão e aproximação de resultados foram ativadas para a realização dos testes.

Na primeira bateria de 100 testes realizados, 85 testes obtiveram resultado positivo, isto é, o elemento móvel conseguiu encontrar a saída do ambiente sem sofrer colisão. Nos outros 15 testes houve colisão do móvel com as arestas do ambiente, onde 5 dessas colisões foram ocasionadas por erro de generalização da RNA e 10 foram ocasionadas pelo fato de ter sido gerada uma posição aleatória muito próxima a alguma aresta do *grid* do ambiente.

A segunda bateria de experimentos foi realizada com 200 testes realizados em novas posições e direções aleatórias. Do total, 174 testes obtiveram resultado positivo, onde o móvel conseguiu encontrar o seu objetivo final sem sofrer colisão. Em 26 testes houve colisão do móvel com as arestas do *grid*, onde 17 dessas colisões foram ocasionadas por erro de generalização da RNA e 9 foram ocasionadas pelo fato de ter sido gerada uma posição aleatória muito próxima a alguma aresta do *grid* do ambiente.

A tabela 4.1 exibe os resultados obtidos nas duas primeiras baterias de testes.

Tabela 4.1: Resultados obtidos nas baterias de testes com a opção de detecção de colisão ativada.

	Total de testes realizados	Total de acertos	Total de erros	Erros ocasionados por erro da RNA	Erros ocasionados por outros motivos
Bateria de Testes 1	100	85 (85%)	15 (15%)	5 (5%)	10 (10%)
Bateria de Testes 2	200	174 (87%)	26 (13%)	17 (8,5%)	9 (4,5%)
Total	300	259 (86,33%)	46 (13,67%)	22 (7,33%)	19 (6,34%)

Do total de testes realizados com a Rede Neural Artificial para controlar o móvel até a saída, 86,33% dos testes obtiveram sucesso o que representa uma boa margem de acertos. Caso a opção de detecção de colisão fosse desligada durante os testes, o elemento móvel sempre encontraria o objetivo por não haver nenhum obstáculo em seu caminho, porém, estaria totalmente inconsistente com a realidade de um veículo dotado de rodas, visto que um veículo real pararia após uma colisão.

4.2.1.2 Testes realizados sem a ativação de detecção de colisão

Para esta seção de testes da verificação da taxa de acerto da saída do ambiente pelo móvel sem a detecção de colisão, também foram realizadas duas baterias de experimentos: a primeira com 100 e a segunda com 200 testes. O objetivo de cada teste é verificar se o móvel atinge o seu objetivo, ao ser aleatoriamente posicionado no *grid* e utilizada a Rede Neural Artificial para movimentá-lo, desconsiderando os limites físicos do *grid* como parâmetros de falha dos testes. Em outras palavras, caso o elemento móvel atinja o seu

objetivo mesmo tendo se deparado com alguma aresta do *grid* durante o itinerário, o teste é avaliado como sucessivo. As opções de detecção de colisão e aproximação de resultados foram desativadas para a realização dos testes: dessa forma, embora não seja possível ultrapassar os limites do *grid* do ambiente de simulação, o elemento móvel não interromperá o teste caso uma colisão aconteça.

Na primeira bateria de 100 testes realizados, 100 testes obtiveram resultado positivo, isto é, o elemento móvel conseguiu encontrar a saída do ambiente independentemente de ter se deparado com uma aresta do *grid* durante seu itinerário.

A segunda bateria de experimentos, com 200 testes realizados, revelou que 200 testes obtiveram resultados positivos, isto é, o elemento móvel conseguiu encontrar a saída do ambiente independentemente de ter se deparado com uma aresta do *grid* durante seu itinerário.

A tabela 4.2 exibe os resultados obtidos nas duas últimas baterias de testes.

Tabela 4.2: Resultados obtidos nas baterias de testes com a opção de detecção de colisão desativada.

	Total de testes realizados	Total de acertos	Total de erros	Erros ocasionados por erro da RNA	Erros ocasionados por outros motivos
Bateria de Testes 1	100	100 (100%)	0 (0%)	0 (0%)	0 (0%)
Bateria de Testes 2	200	200 (100%)	0 (0%)	0 (0%)	0 (0%)
Total	300	300 (100%)	0 (0%)	0 (0%)	0 (0%)

Do total de testes realizados com a Rede Neural Artificial para controlar o móvel até a saída, 100% dos testes obtiveram sucesso o que representa uma excelente margem de

acertos. Deve ser levado em consideração o fato de que o elemento móvel não poderia ser um veículo dotado de rodas nesta hipótese, pois, caso houvesse uma colisão, o veículo não teria capacidade de continuar o seu itinerário a menos que fosse implementado um sistema de marcha a ré (o que evade do escopo deste trabalho).

Com base nesses valores de resultados dos testes é possível afirmar que um elemento móvel sem restrições de colisão, como um robô bípede ou quadrúpede ou um veículo aquático, alcançará sempre seu objetivo caso utilize a Rede Neural Artificial proposta neste trabalho para guiar sua movimentação.

4.2.2 Teste do erro de direção tomada pelo móvel durante o seu trajeto

Com o intuito de prosseguir com a avaliação dos resultados da RNA, duas novas baterias de experimentos foram realizadas, visando analisar o erro de direção do móvel durante seu trajeto até o objetivo final, realizado a partir de um ponto aleatório.

A figura 4.2 exibe o itinerário traçado pelo móvel no ambiente de simulação durante o primeiro teste, realizado a partir de um ponto arbitrário do *grid*, (109.71, 203.4) e de uma direção também arbitrária, (162.48˚).

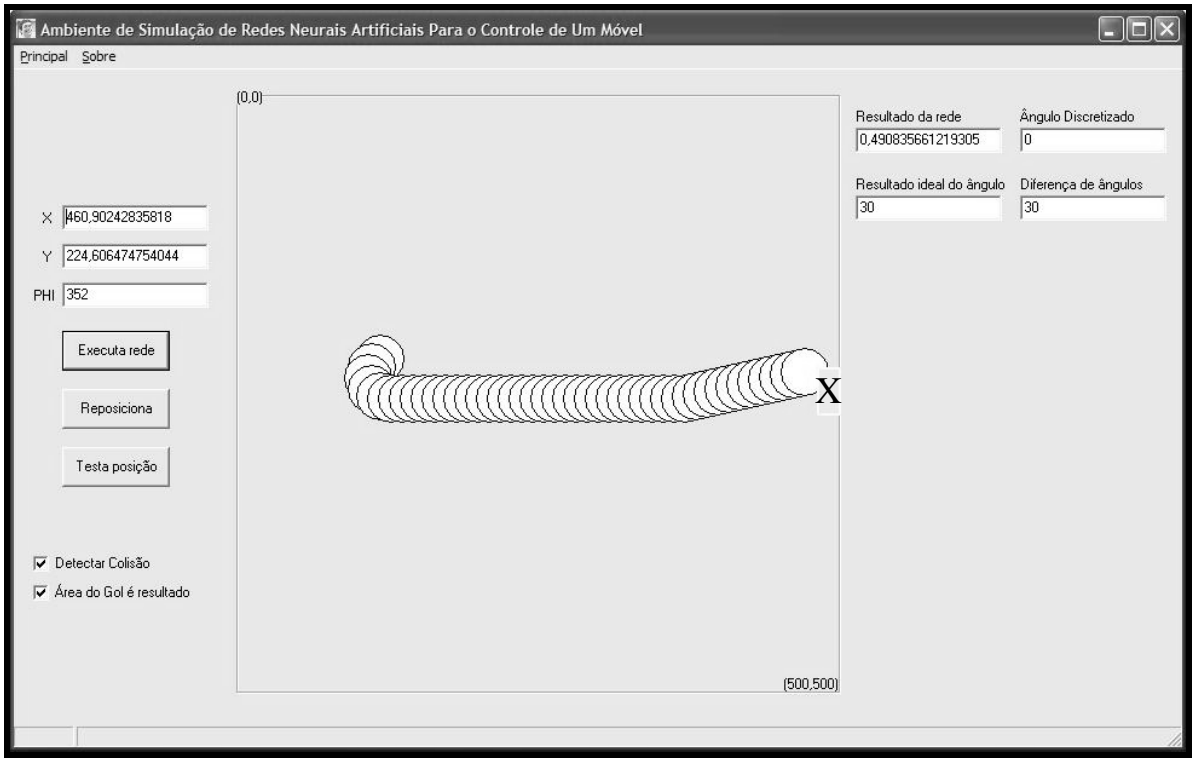


Figura 4.2: Trajetória do móvel no primeiro teste de análise do erro.

A tabela 4.3 lista os resultados numéricos de posição/direção (X,Y/PHI) do móvel, a saída gerada pela rede (TETA), o valor ótimo (desejado) e o erro de direção tomada, medido em graus.

Tabela 4.3: Resultados numéricos do primeiro teste de medição de erro realizado.

X	Y	PHI	TETA	IDEAL	ERRO	|ERRO|
109,71	203,4	162,48	0	0	0	0
102,309	220,43	132	-30	-30	0	0
99,921	229,78	102	-30	-30	0	0
102,09	238,51	72	-30	-30	0	0
109,43	244,69	42	-30	-30	0	0
118,78	246,07	12	-30	-30	0	0
127,99	246,34	2	-10	-11,4	-1,4	1,4
136,99	246,34	2	0	-1,38	-1,38	1,38
145,99	246,34	2	0	-1,37	-1,37	1,37
154,99	246,34	2	0	-1,35	-1,35	1,35

Tabela 4.3: Resultados numéricos do primeiro teste de medição de erro realizado (cont.).

X	Y	PHI	TETA	IDEAL	ERRO	|ERRO|
163,99	246,34	2	0	-1,33	-1,33	1,33
172,99	246,34	2	0	-1,31	-1,31	1,31
181,99	246,34	2	0	-1,3	-1,3	1,3
190,99	246,34	2	0	-1,28	-1,28	1,28
199,99	246,34	2	0	-1,26	-1,26	1,26
208,99	246,34	2	0	-1,23	-1,23	1,23
217,99	246,34	2	0	-1,21	-1,21	1,21
226,99	246,34	2	0	-1,19	-1,19	1,19
235,99	246,34	2	0	-1,16	-1,16	1,16
244,99	246,34	2	0	-1,13	-1,13	1,13
253,99	246,34	2	0	-1,1	-1,1	1,1
262,99	246,34	2	0	-1,07	-1,07	1,07
271,99	246,34	2	0	-1,03	-1,03	1,03
280,99	246,34	2	0	-0,99	-0,99	0,99
289,99	246,34	2	0	-0,95	-0,95	0,95
298,99	246,34	2	0	-0,91	-0,91	0,91
307,99	246,34	2	0	-0,86	-0,86	0,86
316,99	246,34	2	0	-0,81	-0,81	0,81
325,99	246,34	2	0	-0,75	-0,75	0,75
334,99	246,34	2	0	-0,69	-0,69	0,69
343,99	246,34	2	0	-0,61	-0,61	0,61
352,99	246,34	2	0	-0,54	-0,54	0,54
361,99	246,34	2	0	-0,45	-0,45	0,45
370,9	244,6	-8	-10	-0,35	9,64	9,64
379,9	242,6	352	0	10,64	10,64	10,64
388,9	240,6	352	0	11,78	11,78	11,78
397,9	238,6	352	0	13,1	13,1	13,1
406,9	236,6	352	0	14,64	14,64	14,64
415,9	234,6	352	0	16,47	16,47	16,47
424,9	232,6	352	0	18,66	18,66	18,66
433,9	230,6	352	0	21,32	21,32	21,32
442,9	228,6	352	0	24,62	24,62	24,62
451,9	226,6	352	0	28,77	28,77	28,77
460,9	224,6	352	0	30	30	30
Média de erro						5,309302

Conforme observa-se na tabela 4.3, a média do erro do ângulo da direção do móvel foi de ±5,309˚. Isso significa que durante a execução de seu trajeto, o móvel obteve um desvio de direção menor que 6 graus, na média, o que é particularmente bom se for considerado que a Rede Neural foi treinada com um conjunto de 1.000 exemplos diante de

uma possibilidade de 3.600.000 exemplos admissíveis. Tal proeza se deve à capacidade atribuída às redes Perceptron de Múltiplas Camadas de construírem uma representação interna dos dados.

Para reforçar a conclusão tomada no parágrafo anterior, foi realizado um segundo teste para a medição do erro de direção do móvel em uma nova trajetória originada em um ponto aleatório qualquer.

A figura 4.3 exibe o novo percurso tomado pelo móvel, a partir das coordenadas arbitrárias (132.2,89.9) e direção também arbitrária (-27.61˚) adotadas.

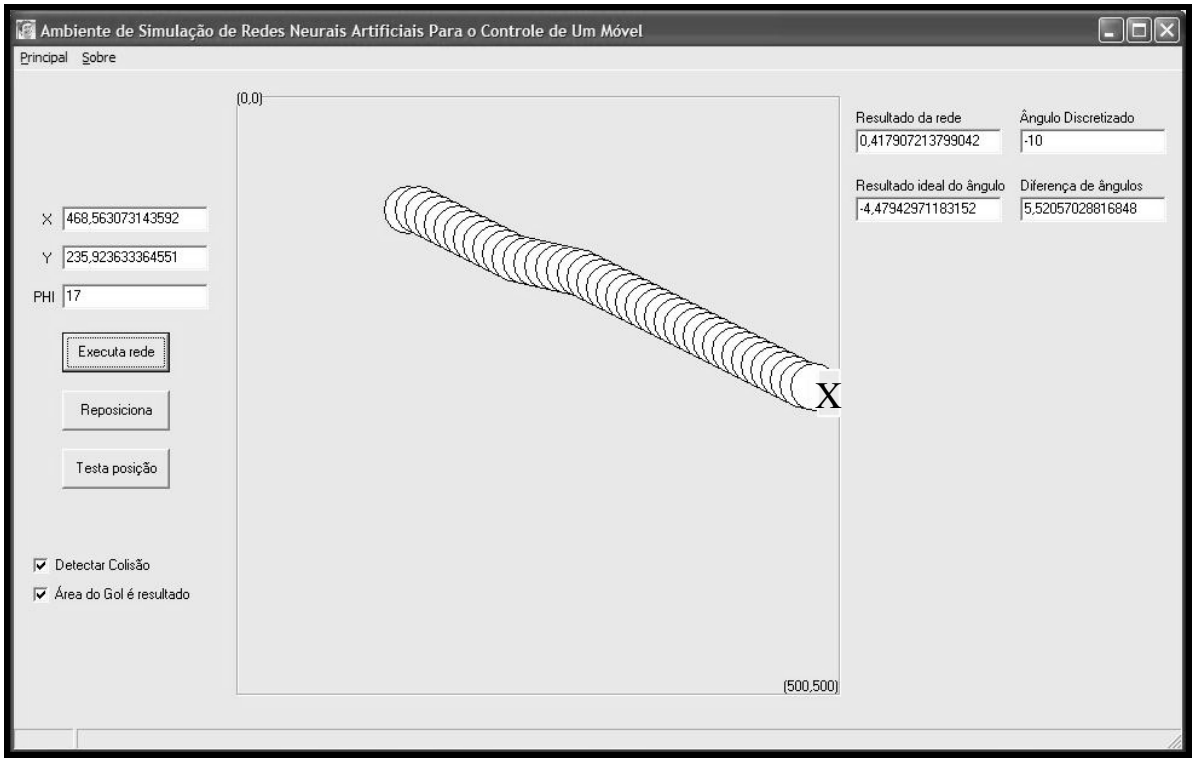


Figura 4.3: Trajetória do móvel no segundo teste de análise do erro.

A tabela 4.4 lista todas as decisões tomadas pela rede ao movimentar o móvel.

Tabela 4.4: Resultados numéricos do segundo teste de medição de erro realizado

X	Y	PHI	SAÍDA	IDEAL	ERRO	|ERRO|
132,2	89,9	-27,61	0	30	30	30
141,98	89,52	363	30	30	0	0
149,91	93,53	27	30	27,15	-2,84	2,84
157,91	97,53	27	0	-2,9	-2,9	2,9
165,91	101,53	27	0	-2,96	-2,96	2,96
173,91	105,53	27	0	-3,02	-3,02	3,02
181,91	109,53	27	0	-3,08	-3,08	3,08
189,91	113,53	27	0	-3,15	-3,15	3,15
197,91	117,53	27	0	-3,22	-3,22	3,22
205,91	121,53	27	0	-3,3	-3,3	3,3
213,91	125,53	27	0	-3,38	-3,38	3,38
221,91	129,53	27	0	-3,46	-3,46	3,46
230,56	131,92	17	-10	-3,55	6,44	6,44
239,56	133,92	17	0	6,78	6,78	6,78
248,56	135,92	17	0	7,14	7,14	7,14
257,56	137,92	17	0	7,52	7,52	7,52
266,56	139,92	17	0	7,93	7,93	7,93
275,56	141,92	17	0	8,37	8,37	8,37
283,91	145,53	27	10	8,84	-1,15	1,15
291,91	149,53	27	0	-1,17	-1,17	1,17
299,91	153,53	27	0	-1,2	-1,2	1,2
307,91	157,53	27	0	-1,23	-1,23	1,23
315,91	161,53	27	0	-1,27	-1,27	1,27
323,91	165,53	27	0	-1,3	-1,3	1,3
331,91	169,53	27	0	-1,34	-1,34	1,34
339,91	173,53	27	0	-1,39	-1,39	1,39
347,91	177,53	27	0	-1,44	-1,44	1,44
355,91	181,53	27	0	-1,49	-1,49	1,49
363,91	185,53	27	0	-1,55	-1,55	1,55
371,91	189,53	27	0	-1,61	-1,61	1,61
379,91	193,53	27	0	-1,69	-1,69	1,69
387,91	197,53	27	0	-1,77	-1,77	1,77
395,91	201,53	27	0	-1,87	-1,87	1,87
403,91	205,53	27	0	-1,98	-1,98	1,98
411,91	209,53	27	0	-2,11	-2,11	2,11
419,91	213,53	27	0	-2,26	-2,26	2,26
427,91	217,53	27	0	-2,44	-2,44	2,44
435,91	221,53	27	0	-2,67	-2,67	2,67
443,91	225,53	27	0	-2,95	-2,95	2,95
451,91	229,53	27	0	-3,31	-3,31	3,31
459,91	233,53	27	0	-3,8	-3,8	3,8
468,56	235,92	17	-10	-4,47	5,52	5,52
Média de erro						3,02439

Conforme observa-se na tabela 4.4, a média de erro durante o trajeto do veículo foi menor no segundo teste, o que significa que o erro médio do ângulo da direção do móvel durante o percurso foi de, aproximadamente, ±3˚.

5. Conclusões

5.1 Conclusões do projeto desenvolvido

De acordo com o exposto no presente trabalho conclui-se que o uso de Redes Neurais Artificiais Perceptron de Múltiplas Camadas associadas ao Algoritmo de Retropropagação de Erro (RNA-MLP com Error Backpropagation) é suficiente para efetuar o controle do elemento móvel no ambiente de simulação proposto.

Como pôde ser percebido pelo resultado dos testes de taxa de acerto e média de erro, descritos nas seções 4.2.1 e 4.2.2 respectivamente, o controle do elemento móvel obteve resultados satisfatórios e suficientes para uma aplicação acadêmica.

O número de possibilidades de posicionamento diferentes do móvel no *grid* é de, no mínimo, [360˚x100x100] = 3.600.000 posições diferentes admissíveis, sem levar em consideração a parte fracionária existente. Para o treinamento da rede foram utilizados 1.000 exemplos (dos 3.600.000 admissíveis), o que representa 0,03% do total de possibilidades. Esses números dão uma idéia do poder de generalização da Rede Neural Artificial para os pontos ainda desconhecidos e reforçam a conclusão de sucesso do trabalho e do uso de RNA-MLP com Error Backpropagation para o controle do móvel no ambiente de simulação proposto.

Além dos testes realizados, outro fator importante a ser levado em consideração é que em todos os exemplos utilizados no treinamento da RNA foram utilizados números inteiros como parâmetros de entrada. Durante a execução da simulação, praticamente todas as posições e direções utilizadas como parâmetros de entrada para a execução da RNA

treinada pertenciam ao universo dos Números Reais. Para que se tenha uma idéia do número possível de posições do móvel no g*rid* utilizando números reais: caso entre um inteiro e seu sucessor existisse apenas o "meio" como parte fracionária, o número de possibilidades no *grid* aumentaria para [(360˚x2)x(100x2)x(100x2)] = 28.800.000 posições diferentes. Esse fator é uma nova contribuição para o reforço da conclusão de sucesso do trabalho e do uso de RNA-MLP com Error Backpropagation para o controle do móvel no ambiente de simulação proposto.

Embora a Rede Neural Artificial desenvolvida tenha uma aprendizagem e generalização do conhecimento adquirido satisfatórios para o controle do móvel, existe ainda uma margem de erro considerável para os pontos desconhecidos, isto é, para os pontos que não foram utilizados no treinamento da RNA. Uma solução para diminuir essa margem de erro está no aumento do número de exemplos do conjunto de treinamento da Rede Neural Artificial. Outra possível solução para a redução do erro seria a associação de conceitos de Lógica Difusa (Lógica Fuzzy) à Rede Neural Artificial, o que resultaria em uma Rede Nebulosa.

5.2 Sugestões para futuros projetos

Em projetos futuros poderiam ser adicionadas barreiras ao ambiente de movimentação do elemento móvel, que seria controlado por RNA ou Redes Nebulosas.

Uma nova sugestão para projetos futuros é o controle de um móvel por RNA ou Rede Nebulosa em um ambiente onde a saída pudesse ser posicionada pelo usuário no momento da aplicação.

Unindo as duas sugestões anteriores, uma nova sugestão para projetos futuros seria controlar um móvel em uma residência ou ambiente com vários cômodos, o que seria bastante útil caso fosse aplicado para facilitar várias tarefas do dia-a-dia, como a entrega de documentos em escritórios através do móvel, segurança noturna em *shoppings centers* e grandes lojas, coleta de lixo, localização de pessoas ou objetos, etc.

6. Referências

[AURÉLIO, VELLASCO, LOPES, 1999] Aurélio, Marco; Vellasco, Marley; Lopes, Carlos Henrique. Descoberta de Conhecimentos e Mineração de Dados – Apostila. ICA – Laboratório de Inteligência Computacional Aplicada, Departamento de Engenharia Elétrica, PUC–Rio, 1999.

[BITTENCOURT, 2002] Bittencourt, João Ricardo. Ambiente para simulação de múltiplos agentes autônomos, cooperativos e competitivos.

[BRAGA, 2000] Braga, Antônio P.; LUDERMIR, Teresa B.; CARVALHO, André C. P. L. F. Redes Neurais Artificiais Teoria e Aplicações, Ed. LTC, 2000.

[BRAGA, 2004] Braga, Vasco Roberto Marinho. Construindo modelos de perfil de clientes em uma instituição financeira com KDD. Universidade Católica de Brasília, 2004.

[BRAGA, 2005] Braga, André Luiz Sordi, VANNGEN – Uma ferramenta CAD flexível para a implementação de Redes Neurais Artificiais em Hardware. Publicação: ENM.DM-03A/05 – UnB.

[CARDENAS, 2004] Cardenas, Antonio Moran. Intelligent Car Parking Using Fuzzy-Neural Networks. Universidad Peruana de Ciencias Aplicadas UPC, 2004.

[FERREIRA, 2004] Ferreira, Aurélio Buarque De Holanda. Novo Dicionário da Língua Portuguesa. Ed. Positivo, 2004

[HAYKIN, 2001] Haykin, Simon, Redes Neurais Princípios e prática. Ed. Bookman, 2001.

[HEINEN, 2002] Heinen, Milton Roberto. Autenticação On-line de assinaturas utilizando Redes Neurais Artificiais. Universidade do Vale do Rio dos Sinos, 2002.

[KOVACS, 1996] Redes Neurais Artificiais: Fundamentos e Aplicações; Um texto básico. Livraria da Física, 1996.

[MATEUS, 2000] Mateus, César Augusto. C++ Builder 5: Guia Prático. Ed. Erica, 2000.

[MIZRAHI, 1990] Mizrahi, Victorine Viviane. Treinamento em Linguagem C, Módulo 1, 2 e Profissional. Ed. Makron Books, 1990.

[MURRAY, TARASSENKO, 1994] Murray, Alan; Tarassenko, Lionel. Analogue Neural VLSI; A pulse stream approach. Ed. Chapman & Hall, 1994.

[OLIVEIRA JÚNIOR, HIME AGUIAR, 1999] Oliveira Júnior, Hime Aguiar e. Lógica Difusa Aspectos práticos e aplicações, Ed. Interciência, 1999.

[OSÓRIO, VIEIRA, 1999] Osório, Fernando Santos; Vieira, Renata. Tutorial Sistemas Híbridos Inteligentes. Tutorial da Unisinos, Universidade do Vale do Rio dos Sinos, 1999.

[ROCHA, 2006] Rocha, Daniel Lyra. Utilização de um ambiente de Honeynet no treinamento de Redes Neurais Artificiais para detecção de intrusão. Publicação: PPGENE.DM – 259/06.

[SILVA, PORTUGAL, CECHIN, 2001] Silva, A. B. M.; Portugal, M. S.; Cechin, A. L.. Redes Neurais Artificiais e Análise de Sensibilidade: Uma aplicação à demanda de importações brasileira. Revecap vol. 5 n. 4, 2001.

[SIMPSON, 1990] Simpson, P. Artificial Neural Systems: Foundations, Paradigms, Applications and Implementations. Pergamon Press Inc. 1990

http://cilib.sourceforge.net/#License – Acesso em 12/12/2006.

http://en.wikipedia.org/wiki/Artificial_neural_network – Acesso em 12/12/2006.

http://en.wikipedia.org/wiki/Cybernetics – Acesso em 12/12/2006.

http://en.wikipedia.org/wiki/Neural_network_software – Acesso em 12/12/2006.

http://en.wikipedia.org/wiki/Neural_networks – Acesso em 12/12/2006.

http://equipe.nce.ufrj.br/thome/grad/nn/curso/mdidatico.htm – Acesso em 12/12/2006.

http://www.ai-junkie.com/ann/evolved/nnt1.html – Acesso em 12/12/2006.

http://www.cerebromente.org.br/n05/tecnologia/plasticidade2.html – Acesso em 12/12/2006.

http://www.cs.bham.ac.uk/~jxb/NN/nn.html – Acesso em 12/12/2006.

http://www.din.uem.br/ia/neurais/#artificial – Acesso em 12/12/2006.

http://www.ece.utep.edu/research/webfuzzy/docs/kk-thesis/kk-thesis-html/ – Acesso em 12/12/2006.

http://www.ene.unb.br/~adolfo/ ISI – Acesso em 12/12/2006.

http://www.faqs.org/faqs/ai-faq/neural-nets/ – Acesso em 12/12/2006.

http://www.functionx.com/bcb/howto/showhint.htm – Acesso em 12/12/2006.

http://www.ibrtses.com/delphi/neuralnets.html – Acesso em 12/12/2006.

http://www.inf.ufsc.br/~awangenh/CG/opengl.html – Acesso em 12/12/2006.

http://www.inf.unisinos.br/~osorio/ – Acesso em 12/12/2006.

http://www.inf.unisinos.br/~osorio/neural.html – Acesso em 12/12/2006.

http://www.infowester.com/redesneurais.php – Acesso em 12/12/2006.

http://www.kcl.ac.uk/depsta/rel/neuronet/old/NEuroNet1/software/software.html - Não se encontra mais disponível na Internet.[11]

http://www.linhadecodigo.com.br/ – Acesso em 12/12/2006.

http://www.lncc.br/verao/verao05/arquivos/MiniCursoDMLNCC050203N.pdf - Não se encontra mais disponível na Internet[12]

http://www.mare-net.com/mscardi/work/nn/nnintro.htm – Acesso em 12/12/2006.

http://www.msd-brazil.com/msd43/m_manual/mm_sec6_59.htm – Acesso em 12/12/2006.

http://www.nd.com/definitions/backprop.htm – Acesso em 12/12/2006.

http://www.neural-networks-at-your-fingertips.com/bpn.html – Acesso em 12/12/2006.

[11] Acesso no segundo semestre de 2005.
[12] Acesso no segundo semestre de 2005.

http://www.ppgia.pucpr.br/~nievola/DadosWEB/XII-ERI-RNA.ppt – Acesso em 12/12/2006.

http://www.programmersheaven.com – Acesso em 12/12/2006.

http://www.ri.cmu.edu – Acesso em 12/12/2006.

http://www.ri.cmu.edu/projects/project_160.htm – Acesso em 12/12/2006.l

http://www.sbfisica.org.br/rfai/Vol17/Num1/v17_12.pdf – Acesso em 12/12/2006.

http://www.sbrn.org.br/ – Acesso em 12/12/2006.

http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0101-74382004000200005 – Acesso em 12/12/2006.

http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0103-17592003000300010 – Acesso em 12/12/2006.

http://www.torch.ch/ – Acesso em 12/12/2006.

http://www.unidev.com.br/ – Acesso em 12/12/2006.

7. Apêndices

Apêndice A – Conjunto de treinamento utilizado para o treinamento da Rede Neural Artificial

Linha	X	Y	Φ em Graus	θ positivo em binário
1	0	0	0	1
2	0	0	4,5	0,2
3	0	0	9	0
4	0	0	13,5	0
5	0	0	18	0
6	0	0	22,5	1
7	0	0	27	1
8	0	0	31,5	1
9	0	0	36	1
10	5	0	0	1
11	5	0	4,5	0,5
12	5	0	9	0
13	5	0	13,5	0
14	5	0	18	0
15	5	0	22,5	0
16	5	0	27	1
17	5	0	31,5	1
18	5	0	36	1
19	9,9	0	0	1
20	9,9	0	4,5	1
21	9,9	0	9	0,5
22	9,9	0	13,5	0
23	9,9	0	18	0
24	9,9	0	22,5	0
25	9,9	0	27	1
26	9,9	0	31,5	1
27	9,9	0	36	1
28	2	2	0	0,8
29	2	2	4,5	0,2
30	2	2	9	0
31	2	2	13,5	0
32	2	2	18	0
33	2	2	22,5	1
34	2	2	27	1
35	2	2	31,5	1
36	2	2	36	0,8
37	7	2	0	1
38	7	2	4,5	0,5
39	7	2	9	0

Linha	X	Y	Φ em Graus	θ positivo em binário
40	7	2	13,5	0
41	7	2	18	0
42	7	2	22,5	0
43	7	2	27	1
44	7	2	31,5	1
45	7	2	36	1
46	0	5	0	0,5
47	0	5	4,5	0
48	0	5	9	0
49	0	5	13,5	0
50	0	5	18	0
51	0	5	22,5	1
52	0	5	27	1
53	0	5	31,5	1
54	0	5	36	0,5
55	1	5	0	0,5
56	1	5	4,5	0
57	1	5	9	0
58	1	5	13,5	0
59	1	5	18	0
60	1	5	22,5	1
61	1	5	27	1
62	1	5	31,5	1
63	1	5	36	0,5
64	5	5	0	0,5
65	5	5	4,5	0
66	5	5	9	0
67	5	5	13,5	0
68	5	5	18	0
69	5	5	22,5	1
70	5	5	27	1
71	5	5	31,5	1
72	5	5	36	0,5
73	8	5	0	0,5
74	8	5	4,5	0
75	8	5	9	0
76	8	5	13,5	0
77	8	5	18	0
78	8	5	22,5	1

Linha	X	Y	Φ em Graus	θ positivo em binário
79	8	5	27	1
80	8	5	31,5	1
81	8	5	36	0,5
82	9,9	5	0	0,5
83	9,9	5	4,5	0
84	9,9	5	9	0
85	9,9	5	13,5	0
86	9,9	5	18	0
87	9,9	5	22,5	1
88	9,9	5	27	1
89	9,9	5	31,5	1
90	9,9	5	36	0,5
91	2	7	0	0,35
92	2	7	4,5	0
93	2	7	9	0
94	2	7	13,5	0
95	2	7	18	1
96	2	7	22,5	1
97	2	7	27	1
98	2	7	31,5	1
99	2	7	36	0,35
100	7	7	0	0
101	7	7	4,5	0
102	7	7	9	0
103	7	7	13,5	0
104	7	7	18	1
105	7	7	22,5	1
106	7	7	27	1
107	7	7	31,5	0,65
108	7	7	36	0
109	0	9,9	0	0
110	0	9,9	4,5	0
111	0	9,9	9	0
112	0	9,9	13,5	0
113	0	9,9	18	1
114	0	9,9	22,5	1
115	0	9,9	27	1
116	0	9,9	31,5	0,8
117	0	9,9	36	0
118	5	9,9	0	0
119	5	9,9	4,5	0
120	5	9,9	9	0
121	5	9,9	13,5	0
122	5	9,9	18	1
123	5	9,9	22,5	1
124	5	9,9	27	1

Linha	X	Y	Φ em Graus	θ positivo em binário
125	5	9,9	31,5	0,5
126	5	9,9	36	0
127	9,9	9,9	0	0
128	9,9	9,9	4,5	0
129	9,9	9,9	9	0
130	9,9	9,9	13,5	1
131	9,9	9,9	18	1
132	9,9	9,9	22,5	1
133	9,9	9,9	27	0,5
134	9,9	9,9	31,5	0
135	9,9	9,9	36	0
136	1,6	7,3	5,3	0
137	6,7	5,7	-6,5	1
138	6,6	7,6	2,9	0
139	8,2	3	17,1	0
140	0,6	9,1	17,8	1
141	1,9	4,3	4,2	0
142	7,8	3,6	-3,9	1
143	4,2	7,4	9,4	0
144	8,2	8,4	9,6	0
145	0,4	3	-2,3	1
146	7,2	0,9	-15,6	0
147	3,3	5,3	-9,3	1
148	8,5	3,4	-8,7	1
149	4	0,1	-7,1	1
150	4,5	4,6	3,3	0
151	8	8,2	-10,7	1
152	3,7	9,2	-2,4	0,35
153	4,4	7,5	-16,2	1
154	7,4	4,9	-8,3	1
155	7,5	9,9	9,8	0
156	6,5	0,2	6	0,35
157	4	7,2	10,8	0
158	2,5	9,8	-7,2	1
159	4,6	5,1	11,8	0
160	8,4	0,9	-4,7	1
161	5,5	9,5	6,5	0
162	6,9	4,3	-13,9	1
163	2,9	7	11,9	0
164	6,8	6,7	-12,2	1
165	5,5	7,8	-0,4	0
166	8,1	1,8	-10,8	1
167	3,8	8,4	-0,7	0,2
168	2,2	6,8	-13,7	1
169	4,3	0	-17,5	0
170	9,7	4,8	16,6	0

Linha	X	Y	Φ em Graus	θ positivo em binário
171	5	7,2	-4,6	0,8
172	3,4	4,3	-7,9	1
173	8,9	3,3	-0,8	1
174	1,4	8,5	4,3	0
175	1,7	1	-0,2	1
176	8,2	9,3	16,3	1
177	9,8	1,8	9,7	0,35
178	6	1,6	5,5	0,35
179	0,6	5,4	10,5	0
180	4,5	3,1	-4,2	1
181	3,5	4,1	16,5	0
182	1,1	8,1	15,7	0
183	0,4	7	12,8	0
184	6,4	6,5	8,5	0
185	2	2,3	-2,6	1
186	5	5	8,5	0
187	2,2	9,6	15,3	1
188	0,9	8,5	5,6	0
189	6,5	8,1	-10,9	1
190	0,1	4,8	-8,5	1
191	2,8	2,6	-4,4	1
192	3	6	7,7	0
193	8,5	1,4	-14,8	0
194	8,9	2	-5,9	1
195	1,5	5,7	-13,7	1
196	2,1	8	-0,3	0,2
197	6,8	4,7	-17,1	1
198	0,9	5,6	-11,2	1
199	9	4,7	-17,3	0
200	7,9	4,9	3,9	0
201	2,2	6,8	3,9	0
202	2	8,6	7,1	0
203	4,6	2,4	-4,1	1
204	5,6	3,2	6	0
205	3,7	0,4	6,4	0
206	9,4	4,3	-6,9	1
207	9,7	5,1	-17,4	1
208	5,8	1,9	8,1	0
209	8,3	3,6	-2,2	1
210	8,4	9,2	-7,5	0,65
211	5	1,2	16,3	0
212	2,7	1,2	-15,8	0
213	9,8	7,9	3,4	0
214	1,9	8,3	-2,3	0,5
215	0,2	9,2	-0,6	0,2
216	2,5	9	6,4	0

Linha	X	Y	Φ em Graus	θ positivo em binário
217	8,7	7,8	-16,1	1
218	0,8	7,7	-8,1	1
219	0,4	4,6	10,3	0
220	3,5	7,5	-12,7	1
221	8,7	7,8	7,4	0
222	4,9	2,5	-12,3	1
223	8,4	4	0	1
224	2,8	0,2	-2,3	1
225	6,3	1	8,1	0
226	4,8	2,5	16,7	0
227	0,5	4	-3,4	1
228	5,5	1,6	6,9	0
229	3,5	6,6	3,7	0
230	5,3	7,5	7,5	0
231	9,5	5,4	-2,7	0,35
232	9,3	7,1	-6,5	0,35
233	7,7	7,7	14,8	1
234	6,8	4,4	-9	1
235	9,8	3,6	15	0
236	6,5	5,6	14,3	0
237	0,8	2,8	0,7	0,65
238	0,8	5,5	-13,5	1
239	0,1	3,3	-7,8	1
240	5	6,6	5,8	0
241	8,7	3,6	-10,3	1
242	4,4	1	-0,2	1
243	3	5,5	-13,3	1
244	4,8	1,1	6,1	0,2
245	1,5	0,7	3,2	0,35
246	5,4	5,6	4	0
247	2,6	1,7	14	0
248	9,4	1,4	14	0
249	3,4	7,5	-15,5	1
250	6,9	3,8	-13	1
251	5,8	6,3	-4,1	0,8
252	1,4	9	-3	0,65
253	1,4	7,5	15,7	0
254	6	8,2	-16,6	1
255	0,4	7,7	6,6	0
256	6,2	2	13,6	0
257	6,3	9,1	-1,6	0
258	6,8	5,2	13,2	0
259	7,3	4,4	-4,6	1
260	3,1	3	-15,3	1
261	6,9	1,5	11,3	0
262	6,2	1,3	-10,4	1

Linha	X	Y	Φ em Graus	θ positivo em binário
263	4,5	0,9	-16,3	0
264	5,2	5	-13,1	1
265	3,5	4,8	-14,5	1
266	9,1	4,1	16,8	0
267	9,3	1,3	8,4	0,5
268	3,2	3,7	-8,8	1
269	9,1	3,7	14,3	0
270	3,7	0,7	-15,4	0
271	9,1	6,7	5	0
272	5,6	6,8	7	0
273	7,1	8,2	1,2	0
274	3	9,2	-7,4	1
275	8	7,7	-10	1
276	4,6	5,5	3,7	0
277	3	3,6	16,3	0
278	7	6,3	16,4	1
279	9,3	4,5	14,4	0
280	8,7	0,9	-2,1	1
281	4,7	5,3	-6,9	1
282	3,5	5,5	-13,4	1
283	8,9	1,5	2,8	1
284	3,6	4,5	5,7	0
285	2,8	5,6	9,3	0
286	5,3	8	-15,2	1
287	8,8	6	-3,1	0,35
288	9,6	0,5	-7,9	1
289	1	8	-11,1	1
290	1,4	8,9	8,7	0
291	9,4	9,6	5,7	0
292	0,4	1,6	-15,5	1
293	8,2	0,8	16,6	0
294	1,6	2,4	10	0
295	6,9	0,2	7	0,35
296	3,2	9,7	7,3	0
297	0,6	2,5	-2,7	1
298	1,2	7,5	-1,1	0,5
299	0,6	2,9	-12,6	1
300	0	9,1	4	0
301	6,6	5,9	15,8	0
302	3,5	8,9	1,6	0
303	0,3	1	3,8	0,2
304	8,1	0,6	-8,2	1
305	4	0,1	-4,5	1
306	7,1	0,7	11,8	0
307	5,6	2,1	-17,8	0
308	3,8	5,5	-11,1	1

Linha	X	Y	Φ em Graus	θ positivo em binário
309	3,7	5	14,9	0
310	9,3	7,2	5,2	0
311	4,2	7,4	-14,8	1
312	2,2	1,1	10	0
313	3,5	2	-8,2	1
314	7,9	3,2	-9,9	1
315	7,9	8,3	6,3	0
316	0,9	2,7	6,3	0
317	7,2	9,5	8,1	0
318	3,3	2,5	-6,2	1
319	9,1	9,2	-11,1	1
320	3,9	2	7,1	0
321	5,4	7,2	-0,4	0,2
322	5,9	9,5	-11,3	1
323	1,2	1,9	-7	1
324	2	6,2	-6,3	1
325	9	8,6	-4,8	0
326	1,6	4,6	8,2	0
327	0,1	4,2	-1,7	0,8
328	4,7	7	2,7	0
329	7,8	5,1	18	1
330	4,7	8,2	13,9	0
331	8,1	6	14,1	0
332	1,9	8,8	8,9	0
333	3,4	0,3	-6,9	1
334	2,5	3,2	15,4	0
335	5,3	0,5	-6,5	1
336	5,2	1,8	13,1	0
337	2,9	3,5	5,6	0
338	9,9	7,4	0,4	0
339	7	7,5	10,7	0
340	5,9	1,3	-14,4	0
341	7,6	3,6	11,2	0
342	9,8	5	-1,1	0,65
343	2,8	0,9	-0,7	1
344	2,9	3	15,1	0
345	4,3	8,4	10,3	0
346	8,2	1,5	-0,9	1
347	5	7,1	-2,4	0,5
348	0,6	2,8	-7,5	1
349	5	5,1	13	0
350	8,1	2,5	-1,7	1
351	4,8	5,9	11,7	0
352	4,5	5,3	2,6	0
353	8,4	2,3	-2,4	1
354	1,2	7,5	-5,5	1

Linha	X	Y	Φ em Graus	θ positivo em binário
355	6,3	9,8	-14,2	1
356	6,4	0,9	-4,1	1
357	8	6,3	-2,8	0,35
358	5,9	6	17,8	1
359	9,2	7,4	-14,5	1
360	9,6	7	4,6	0
361	3,6	6,4	8,6	0
362	9,3	5,5	-9,8	1
363	4,9	0,2	-14,4	0
364	1,6	1	-3,6	1
365	3,6	5,1	15	0
366	2,5	1,2	0,5	0,8
367	9,7	6,9	11,1	1
368	5,3	2,9	12,3	0
369	8,9	8,6	17	1
370	9	1,6	10,6	0
371	7,1	3,4	-1,7	1
372	4,2	2,3	12,1	0
373	7,2	4,2	17,4	0
374	5,8	9,7	2,9	0
375	1,3	2,6	5,8	0
376	6,8	7,9	-11,6	1
377	9,4	3,1	-15,1	0
378	8,3	1,7	1,8	1
379	6,7	5	-12,7	1
380	1,4	8,9	-15,9	1
381	7,5	8,6	17,9	1
382	7,7	1,3	-8,7	1
383	0,2	9,2	3,1	0
384	3,6	8,8	-8	1
385	7,8	2,8	4,1	0,5
386	1,6	0,8	-15,1	1
387	5,5	7,1	-15,1	1
388	4,3	3	-5,7	1
389	8,8	8,9	17,3	1
390	2,3	6,8	3,2	0
391	0	1,8	2	0,5
392	2,8	9,1	-1,4	0,2
393	5,3	4	0,4	0,65
394	5,4	4	-13,3	1
395	3,2	4	-14,5	1
396	4,4	2,7	9	0
397	6,9	9,3	-1,1	0
398	7,7	7,2	-8,2	1
399	1,3	2,1	11,9	0
400	7,8	7,1	-3,9	0,5

Linha	X	Y	Φ em Graus	θ positivo em binário
401	3,9	0,1	13,4	0
402	3	9,4	14,1	0
403	6,8	6,8	12,4	0
404	5,9	7	-8,9	1
405	1	7,1	9,5	0
406	4,5	1	5	0,35
407	5,7	0,9	-12,7	1
408	7,7	8,7	3,2	0
409	2,7	9,8	-17,8	1
410	0,2	4,6	-1,4	0,8
411	0,7	1	-2,7	1
412	9	9,4	-4,7	0
413	4,9	2,5	-9,3	1
414	6,4	8,5	5	0
415	7	2,2	9,7	0
416	8,6	0,3	5,5	0,8
417	5,6	9,7	3,9	0
418	8,4	2,6	-4,2	1
419	2,1	6,7	15	0
420	2,5	9,8	14,2	0
421	8,3	0,9	-7,7	1
422	2,3	7,1	-10,7	1
423	7,3	2,9	-4	1
424	1,5	6,8	-12,8	1
425	2	7,6	-10,1	1
426	7,8	1,2	-13,5	0
427	4,4	8,2	16,5	1
428	7,4	6,3	11	0
429	9,5	3,5	-1,1	1
430	0,2	6,6	-2	0,65
431	0,2	9,9	-11,8	1
432	5	7,8	-9,7	1
433	4,9	3,3	-7,7	1
434	3,4	1,9	12,7	0
435	6,3	9,8	-5,6	0,5
436	7,1	9	11,7	0
437	0,8	0,3	-14,3	1
438	0	1,9	-7,9	1
439	9,5	0,9	9,2	0,35
440	8,1	1,7	6,7	0,35
441	7,2	0,8	-4	1
442	0,5	3,1	-7,8	1
443	0,9	0,9	12,5	0
444	7,3	8,6	-1,5	0
445	4,6	3,3	-14,6	1
446	5,1	5	14,8	0

Linha	X	Y	Φ em Graus	θ positivo em binário
447	8,9	1,7	6,6	0,65
448	0,1	7,3	-4,9	1
449	6,9	4	-7,1	1
450	9,7	3,1	3,6	1
451	1,5	8,5	-1,4	0,35
452	8,2	3,3	-3,1	1
453	1,6	9,2	14,2	0
454	5,7	1,2	-12,8	1
455	2,3	3,8	9,5	0
456	9,1	0,1	-10,1	0
457	1,5	8,2	-11,3	1
458	9,2	1,4	9,9	0,2
459	4,6	8	14,5	0
460	0,8	2,3	17,3	0
461	5,7	6,7	4,8	0
462	3,2	9	5,3	0
463	1,8	9,1	6,9	0
464	6,6	2,1	-17,4	0
465	1,2	1,1	17,8	0
466	9,2	3,5	-1,9	1
467	8,3	1,7	-1,5	1
468	9,6	0,5	11,8	0
469	1,6	4,5	-10	1
470	8,3	3,3	-16,5	0
471	5	6,4	3,4	0
472	8,2	9,5	16,1	1
473	0,6	1,7	-1,7	1
474	0,8	7,7	-1,8	0,5
475	6,2	6,5	16,2	1
476	7,2	5,5	7,5	0
477	4,3	7,3	-9,2	1
478	1	9,8	-3,2	0,5
479	6	4,1	0,2	0,65
480	2,7	4,4	-2,9	1
481	0,4	7,3	-14,2	1
482	4,6	7,9	9,9	0
483	9,4	9,6	-3,2	0
484	4,1	1,7	-17,2	0
485	3,3	4,7	7,6	0
486	0,4	3,4	-15,9	1
487	9,5	4,2	12,2	0
488	8,6	5	16,2	0
489	5,8	7,4	6,6	0
490	5	8	4,9	0
491	2,6	4,6	-0,2	0,65
492	1,8	4,8	5,5	0

Linha	X	Y	Φ em Graus	θ positivo em binário
493	2,2	3,7	-4,4	1
494	6,5	3,3	-3,2	1
495	8,1	4,8	9,8	0
496	9,2	3,7	7,4	0,2
497	7,5	1,7	-16,6	0
498	2,2	0,5	6,4	0
499	7,3	2,6	-0,8	1
500	5,2	7,6	2,4	0
501	2,1	6	-15,7	1
502	3	4,5	-16,1	1
503	3,2	8,8	-6,8	1
504	6,2	4,5	17,7	0
505	6,1	9,3	-4,7	0,5
506	3,5	2,5	-9,2	1
507	1	7,2	-16,3	1
508	6,4	8,4	-6,2	0,8
509	4,7	2,1	9,6	0
510	7,6	5,9	-17,9	1
511	0,9	9,1	3,5	0
512	3,8	1,3	-15,4	0
513	9,4	0,5	17,1	0
514	0,9	0,8	8,7	0
515	8,3	9,8	-6,6	0,5
516	5,6	6,1	-7,7	1
517	3,3	0,2	16,7	0
518	7,9	9	17,6	1
519	6,7	6,3	1,1	0
520	0,3	3,4	2,5	0,2
521	8,9	1,7	10	0
522	9,6	5	10,9	0
523	7,5	2,9	-13,9	1
524	9,2	5,8	15,1	1
525	3	4,4	11,6	0
526	3,9	4,7	-8,8	1
527	7	4,9	-4,3	1
528	7,6	2,2	-9,2	1
529	1,7	9,8	-8,4	1
530	5,3	0,9	-6	1
531	8	0,6	14,9	0
532	5,1	3,9	-3,5	1
533	4,7	6,8	9,7	0
534	8,3	1	6	0,65
535	8,1	7,6	-8,3	1
536	9,1	8,2	4,2	0
537	4,3	6,5	-10	1
538	2,9	6,6	2,1	0

Linha	X	Y	Φ em Graus	θ positivo em binário
539	2,2	2,8	-15,7	1
540	2,6	7,5	-10	1
541	0,1	2,3	-15,7	1
542	8,6	1,2	-6,5	1
543	3,8	9,4	-8,2	1
544	2,6	3,1	17,7	0
545	2,8	6,5	-9,8	1
546	2,1	0,9	-3,3	1
547	4,2	8,7	17,2	1
548	3,5	3,1	11,8	0
549	5,4	5,9	-15,4	1
550	9,5	8	17	1
551	1	7,5	10	0
552	8,4	4,5	5,1	0
553	1,4	4,3	4,6	0
554	9,1	7,5	8,6	0
555	1,3	6,8	11,5	0
556	0,7	7,7	11,4	0
557	6,7	9,7	-8,6	1
558	3,3	2,9	4,3	0
559	1,2	9,2	-5,4	1
560	1,2	5,6	-2,8	0,8
561	6,2	6,5	3,6	0
562	8	7,4	-4,8	0,5
563	2,6	8	-2,6	0,5
564	1,1	2,4	12,8	0
565	8,4	6,4	1,6	0
566	0,7	8	5,5	0
567	8,5	8,2	1,8	0
568	5	3,7	3,2	0,2
569	4,7	9,1	-9,6	1
570	2,5	7,7	16,7	1
571	5,2	2,9	10,4	0
572	8,7	3,9	7,3	0
573	7,1	9,2	-16,8	1
574	5,4	4,8	10	0
575	6,7	4	10,2	0
576	9,3	7,7	17,8	1
577	1,9	6	-14	1
578	9,5	9	13,8	1
579	7,2	0,5	-2	1
580	1,8	3,8	14,6	0
581	5,6	9,3	11,2	0
582	5	8,3	-2,6	0,35
583	0,3	7,4	-7,6	1
584	2,7	2,8	16,4	0

Linha	X	Y	Φ em Graus	θ positivo em binário
585	2,8	9,9	16,1	1
586	1,5	4,8	17,4	0
587	9,7	9,5	1,8	0
588	7,8	9,9	11,4	0
589	9,6	0,6	11,4	0
590	2,3	7,8	0	0,2
591	8,5	5	-16,9	1
592	1,4	5,7	-4,4	1
593	4,6	3,3	1,2	0,65
594	8,3	7,9	2	0
595	9,2	2,2	-0,2	1
596	2,5	1,9	-3,2	1
597	6,6	4,9	-11,7	1
598	8,3	7,4	15,5	1
599	5,1	6,5	6,5	0
600	6,9	9,6	-0,4	0
601	4,1	9,3	-2,7	0,35
602	1,6	6,2	-1,4	0,65
603	7,6	1,7	15,9	0
604	0,4	7	-3,1	0,8
605	3,2	5,3	-7,2	1
606	1,1	2,5	8,7	0
607	3,2	0,2	-10,4	1
608	7,7	3,5	-12	1
609	3,6	8,2	-6	1
610	2,7	2,6	-9,2	1
611	8	9,2	-15,4	1
612	4,9	3,2	13,8	0
613	5,1	6,5	17,5	1
614	7,4	8,5	0,7	0
615	7,3	3,6	-9,8	1
616	2,4	8	6,9	0
617	1,5	9,7	-2,2	0,35
618	1,5	4,9	-5,7	1
619	8,1	0,2	15,7	0
620	1,5	8,3	8,9	0
621	4,1	4,3	1,9	0,35
622	3	1,1	-2,7	1
623	0,4	1,4	2,1	0,5
624	1,7	1	-3,4	1
625	5,6	5,8	-0,7	0,5
626	2,7	3,9	16,4	0
627	8,4	8,3	5,1	0
628	8,6	6,7	0	0
629	5,9	3,8	-3,2	1
630	7,7	6,4	12,3	0

Linha	X	Y	Φ em Graus	θ positivo em binário
631	8,2	1,3	-6,8	1
632	8,1	0,6	-0,5	1
633	5,4	1	4,1	0,5
634	0,5	5,8	0,6	0,35
635	6,5	6,1	16,8	1
636	5,2	3,5	5,9	0
637	8,6	6	6,3	0
638	5,3	5,4	-0,3	0,5
639	5,3	2,4	-3,7	1
640	3	1,3	3,2	0,5
641	6,5	1,5	6,1	0,2
642	6,1	9,2	-12,6	1
643	8,1	0	-1,6	1
644	7	0,4	15,6	0
645	7,8	2,5	-12,8	1
646	5,4	0,3	5,3	0,35
647	0,3	2,2	-12,6	1
648	8,7	4,3	12,8	0
649	7,5	2	8,6	0
650	8,1	1,7	-9,4	1
651	5,7	8,1	-10,2	1
652	5,6	9,4	-5	0,5
653	6,4	7,9	16,8	1
654	7,1	3,2	8,7	0
655	9	3,8	-3,5	1
656	5,3	3,6	-15,3	1
657	2	2	4,8	0
658	9,8	8,2	-12,6	1
659	7,2	3,4	-6,8	1
660	6,2	4,7	-13,6	1
661	6,2	0,2	11,5	0
662	8,8	8,9	16	1
663	7,8	3,9	-11,9	1
664	7,4	7,9	16	1
665	4,9	7,7	-12,4	1
666	3,1	1	-8,2	1
667	9,5	8,1	-8,9	0,65
668	7,2	3,5	3,6	0,35
669	8,1	5,6	13,6	0
670	2,3	8	-12	1
671	4,9	1,7	8,8	0
672	7,2	1,2	13	0
673	9,1	4,2	14,8	0
674	6	2,9	-10,2	1
675	7	1,8	-13,1	1
676	3,7	8,5	5,7	0

Linha	X	Y	Φ em Graus	θ positivo em binário
677	9,1	3,4	-10,7	1
678	4,7	5,8	-13	1
679	0,5	6,2	17	0
680	8	3,3	-6,1	1
681	9,7	8,6	13,7	1
682	6,2	4,1	-14,5	1
683	7,2	2,9	14,2	0
684	5,8	9,6	-4,9	0,5
685	2,8	7	16,5	1
686	0,3	0,1	4,7	0,2
687	5,9	1,7	-7	1
688	4,7	1,5	15,9	0
689	4,9	9,7	14,4	1
690	5,7	2,3	-11,8	1
691	9,1	1,9	12,3	0
692	3,6	5,6	-15,8	1
693	6,7	8,4	-15	1
694	6,4	7,6	11,3	0
695	6,9	6,4	-5,5	1
696	8,4	6,4	10,8	0
697	1,7	0,7	-14,9	1
698	4,6	2,2	-4,1	1
699	5,1	0,6	-9,4	1
700	9,1	4,7	-9,2	1
701	2	4,1	-8,1	1
702	8,8	2,4	12,9	0
703	9,2	6,9	-15,3	1
704	3,3	3,1	17,4	0
705	8,1	3,9	7,9	0
706	4,3	6,2	-8,1	1
707	1,8	0,2	13,2	0
708	3,6	8,8	-9,5	1
709	0,9	5,4	5,1	0
710	6,4	0,7	-0,9	1
711	3,1	9	-9,2	1
712	7,9	1,5	2,8	1
713	3	9,1	-11,2	1
714	7,9	6,9	-17,4	1
715	8,5	0,2	7	0,5
716	4,7	7,6	-2,9	0,5
717	3,7	4,5	-16	1
718	6,1	5,3	-10,9	1
719	0,6	2,2	-12,1	1
720	1,9	1,3	7,1	0
721	9,9	8,8	15,4	1
722	3,2	9,5	1,7	0

Linha	X	Y	Φ em Graus	θ positivo em binário
723	8,4	2,9	-12,8	0
724	1,4	4,9	-8,1	1
725	8,7	0,9	-16,8	0
726	4,8	4,4	-10,7	1
727	4,9	9,9	2,9	0
728	3,4	4,3	-14,6	1
729	3,7	9,4	13,4	0
730	1,8	1,2	-11,4	1
731	5,7	2,5	11,5	0
732	3,4	4,4	-3,5	1
733	6,4	3,1	-6,7	1
734	5,2	1,3	-7,7	1
735	1,4	2,4	4,7	0
736	3,3	8,1	-11,9	1
737	1,7	7,1	9,3	0
738	3,4	2,8	-7,6	1
739	3,2	9,8	2,4	0
740	5,3	7,5	-3,7	0,65
741	4,2	5	10,1	0
742	6,6	3,8	9,2	0
743	9,6	4,5	-5,8	1
744	6,6	7,4	-14,2	1
745	2,4	7,8	-17,9	1
746	7,2	2,9	-1,2	1
747	8,9	9,5	3,1	0
748	5,5	1,3	2,8	0,65
749	7,3	3,2	17,9	0
750	7,2	9,3	-6,6	0,65
751	8,7	5,1	-9,6	1
752	5,6	9,6	8	0
753	0,2	8	2,8	0
754	9,2	5,7	-0,1	0
755	3,3	3,3	-14,9	1
756	7,8	6,4	17,4	1
757	8,3	0	-10,3	1
758	7,4	2,8	-6,8	1
759	2,7	8,7	4,3	0
760	1,3	6,2	-4,4	1
761	5,6	9,9	-12,3	1
762	1,6	3,5	-2	1
763	8,2	1	15,6	0
764	7,9	0,3	-13,5	0
765	9,3	7,1	-2,3	0
766	5,4	0,5	-3	1
767	4,7	5,6	11,9	0
768	9,6	1,6	-12,5	0

Linha	X	Y	Φ em Graus	θ positivo em binário
769	3,3	2,8	-3,7	1
770	9	1,3	-1,2	1
771	0,4	9,8	11,2	0
772	5,8	5,5	10,7	0
773	6,2	2,4	-9,6	1
774	6,7	1,8	10,3	0
775	6,8	8,7	-9,5	1
776	6,5	8,9	-5,5	0,65
777	9,4	7,9	-3,8	0
778	8,3	2,2	-0,7	1
779	5,4	8	-14,3	1
780	8,5	9,3	16,3	1
781	3,1	5	-2,5	0,8
782	2,8	4,9	3	0
783	7	6,2	-17,4	1
784	6,9	4,6	3,9	0
785	7,9	4,8	-11,1	1
786	1,3	4,7	-3,2	1
787	9,5	0,6	11,8	0
788	8,1	5,7	6,6	0
789	8,6	6,8	1,3	0
790	7,5	9,7	16	1
791	5,9	4	17,1	0
792	9,7	8,7	1,8	0
793	3,9	2,2	-12,4	1
794	4,7	3,2	2,7	0,35
795	3,4	7,3	0	0,2
796	2,3	3,2	6,9	0
797	1,3	4,1	-10,9	1
798	7,4	2,5	5,1	0,35
799	2,7	4,5	13,1	0
800	5,4	6,2	0,7	0,2
801	6	4,9	2	0,2
802	5,8	0,8	-9,7	1
803	9,3	9,1	-15,9	1
804	0,5	5,3	-12,7	1
805	8,4	4,8	-17	1
806	7,5	3,3	-5,3	1
807	5,5	7,4	7,7	0
808	9,7	6,2	1,2	0
809	9,6	2,5	10,1	0,2
810	1,4	3,9	0,5	0,5
811	8,9	1,7	-8,2	1
812	4,4	6	12,8	0
813	2,5	4,1	-17,1	1
814	5,9	1,5	-2,7	1

Linha	X	Y	Φ em Graus	θ positivo em binário
815	6,4	2,6	-1,4	1
816	3,7	1,1	13,8	0
817	0,9	5	1,3	0,35
818	6,2	3,8	2,2	0,5
819	1	9,5	9,1	0
820	7,6	4,7	7,3	0
821	8,9	8,9	10,1	0
822	3,6	8,4	15,5	1
823	4,8	1,1	6	0,2
824	2,7	6,9	-15,8	1
825	3,4	9,1	9,9	0
826	1,6	4,3	11,5	0
827	3,9	4,5	-0,9	0,65
828	7,7	4,7	4,6	0
829	5,4	4,4	17,9	0
830	4,2	7,6	1	0
831	8,7	1,8	12,4	0
832	4,6	2,7	-17,4	0
833	5,7	2,6	11,8	0
834	1,4	6,4	7,9	0
835	5,5	6,5	-13,5	1
836	7,3	6,3	-15,7	1
837	1,7	5,8	7,3	0
838	9,1	5,8	7,7	0
839	3,5	3,4	-0,5	0,8
840	6,4	3,4	-4,1	1
841	2,7	1,2	-2,2	1
842	8,7	3	2,5	1
843	4,9	0,3	5	0,35
844	3,9	8	-3,8	0,65
845	9,9	7,6	-10,9	0,8
846	7,5	4,1	14,4	0
847	9,4	0,3	14,6	0
848	7,9	6	2,3	0
849	3	8,9	14	0
850	2,5	8,4	6,4	0
851	4,4	0,7	9,8	0
852	9	3,9	-14,7	0
853	4,8	1,1	11	0
854	3,5	4,6	-5,9	1
855	6,2	3,3	10,6	0
856	2,3	9,7	1,9	0
857	1,8	1,1	16,7	0
858	7,8	8,3	17,5	1
859	3,7	3,6	15,3	0
860	4,1	0,5	8,4	0

Linha	X	Y	Φ em Graus	θ positivo em binário
861	3,6	3	-15,6	1
862	5	4,3	-5,7	1
863	2,2	7,9	-14,5	1
864	1,9	4,2	-13,1	1
865	9	8,3	1,9	0
866	9,7	1,2	-17,9	0
867	9,5	5,3	-3,1	0,5
868	1,8	7,4	-12,9	1
869	0,4	5,8	6,1	0
870	9,7	5,2	-17,4	1
871	2,7	0,9	-0,3	1
872	4,1	9,7	17,2	1
873	5,3	2,8	-12	1
874	1,3	4,5	-3,4	1
875	2,6	5,5	-13,4	1
876	6,2	7,6	-12	1
877	1,4	5,8	0	0,35
878	9,9	0,8	1,9	1
879	7,6	4,4	-17	0
880	0,3	9,4	-3,3	0,65
881	3	3,7	0,5	0,65
882	4,9	5,4	1,3	0,2
883	8,3	1,9	-16	0
884	7,9	5,1	-10,9	1
885	8,9	5,5	4,9	0
886	3,2	5,9	9,9	0
887	7,7	8,1	-2,2	0
888	8,6	7,3	6	0
889	7,2	1,6	-4,1	1
890	5,8	9,6	2,7	0
891	8,7	0,4	-3,7	1
892	6,1	7	6,3	0
893	4,5	6,3	9,8	0
894	6,7	9,8	-4,6	0,35
895	3,4	5,9	0,3	0,35
896	4,6	1,8	-8,9	1
897	5,8	0,3	13,6	0
898	2,4	3,4	17,7	0
899	7,3	2,4	1,8	1
900	2,7	6,9	-16,9	1
901	4,5	5,6	-5,8	1
902	9,8	4,4	7	0,5
903	3,1	1,8	-12,3	1
904	7,3	3,5	17	0
905	7,8	3,7	-8,7	1
906	9,3	6,4	-13	1

Linha	X	Y	Φ em Graus	θ positivo em binário
907	3,9	7	-8,8	1
908	5,4	1,6	-6,9	1
909	7,9	9	-4,3	0,2
910	5,7	8	-5	0,8
911	2,5	5,2	13,2	0
912	0,7	7,3	-0,9	0,5
913	4,1	1,9	7,8	0
914	3,2	9,3	2,4	0
915	8,1	7,5	1,6	0
916	9,7	1,8	-9,6	0
917	4,8	5,2	-1,1	0,65
918	8,7	5,7	8,1	0
919	0,6	7,1	8,7	0
920	2,9	5,1	-9,2	1
921	0	7,5	-11,2	1
922	1,8	2,5	14,5	0
923	3,9	3,8	5	0
924	3,1	6	-9,7	1
925	4,1	5,2	7,4	0
926	4,7	9,7	13,7	0
927	8,3	9	-5,8	0,35
928	6	4,1	11,5	0
929	7,8	0,5	-11,6	1
930	3,5	8,9	-14,4	1
931	1,3	8,6	-17,2	1
932	7,6	9,1	17,3	1
933	9,7	1,9	17,7	0
934	5,4	3,8	-9	1
935	3,3	0,1	11,8	0
936	5,9	6,9	12,6	0
937	6,7	6,9	6,9	0
938	9	4,5	14	0
939	9,4	8,2	6,2	0
940	6,2	1,9	13,8	0
941	3,7	0,2	17	0
942	4,9	6,2	13,4	0
943	6,3	1,4	2,8	0,8
944	4,4	8,3	-13,6	1
945	8,9	6,8	-12,8	1
946	8,1	7,8	-16,6	1
947	9,2	7,1	-12,1	1
948	3,7	0	-5,3	1
949	2,9	7,3	8,7	0
950	9,5	1,6	16,4	0
951	1	5,4	-7,4	1
952	8,6	5,1	-7,8	1

Linha	X	Y	Φ em Graus	θ positivo em binário
953	5,4	5,3	-0,7	0,5
954	3,6	0,8	8	0
955	8,7	8,1	-4,7	0,2
956	5,7	4,4	-6,7	1
957	5,3	0,9	4,1	0,5
958	9,1	7,5	-16,3	1
959	5,6	4,9	-2,2	0,8
960	7,3	9,7	-7	0,65
961	6,3	3,4	2,4	0,5
962	2,1	0,9	-2,7	1
963	1,9	8,1	11,1	0
964	3,8	3,4	-7	1
965	6,1	8,8	13,5	0
966	4,6	1,4	-10,7	1
967	6,7	3	8,1	0
968	2,4	2,3	-2,7	1
969	1,6	8,2	-13,7	1
970	0,9	1,6	8,5	0
971	0	2,7	17,8	0
972	8,7	9,4	-11,1	1
973	4,9	0,9	15	0
974	6,3	5,7	-9,7	1
975	1,1	3,6	-4,2	1
976	6	9,1	-14,5	1
977	8,2	9,7	14	1
978	2,6	9,8	-5,6	0,8
979	5,9	4,1	-5,9	1
980	8	7,1	-15,9	1
981	3,1	6,4	-3,3	0,8
982	6,9	4	-3,7	1
983	4,8	1,1	-8,3	1
984	1,5	4,3	16	0
985	8,1	5	1,7	0,2
986	6,5	3,8	-13,2	1
987	7,6	0,4	-11,7	1
988	7,3	4,4	2,7	0,35
989	7,5	7,5	-14,7	1
990	0,6	2,1	9	0
991	7,2	5,5	1,6	0
992	2,3	8	3,3	0
993	9,4	5,7	-11,4	1
994	6,7	4,4	-6,2	1
995	8,2	3,4	1,1	1
996	9,3	6,9	3,9	0
997	0,9	8,3	7,2	0
998	1,3	7,9	-8,3	1

Linha	X	Y	Φ em Graus	θ positivo em binário
999	6,8	6,6	-11,1	1

Linha	X	Y	Φ em Graus	θ positivo em binário
1000	6,5	0,8	-7,2	1

Apêndice B – Código fonte do programa de treinamento da Rede Neural Artificial

```
/*
UniCEUB - Centro Universitario de Brasilia
FAET - Faculdade de Exatas e Tecnologia
Curso de Engenharia de Computacao
70790-075 - Brasilia - DF, Brazil.



ALUNO: EDUARDO BRAGA DUTRA ROCHA


PROGRAMA DE TREINAMENTO DE UMA REDE NEURAL ARTIFICIAL PERCEPTRON DE MULTIPLAS
CAMADAS COM ALGOTITMO DE RETROPROPAGACAO DE ERRO


[ROCHA, 2006]


O uso desse codigo fonte objetiva a criação e armazenamento de pesos para o
AMBIENTE DE SIMULACAO DE REDES NEURAIS ARTIFICIAIS PARA O CONTROLE DE UM MOVEL.
Armazene os arquivos de texto gerados com esse programa no mesmo diretorio do
executavel Ambiente_Neural.exe
*/


/*************************************************************************
* REDE NEURAL ARTIFICIAL -  Baseado no algoritmo de JOHN BULLINARIA,  2004     *
*                                                                               *
*       POR                                             *
*          EDUARDO BRAGA DUTRA ROCHA                                *
*               ENGENHARIA DE COMPUTAÇÃO - UNICEUB - DF - BRASIL - 2006     *
*                                                 *
*************************************************************************/


/* Notes before compiling:
```

```
This project was originally compiled using C++ Builder 5. Before running the
project be sure to adjust the linker parameters in the project options.
Assuming that there is a project created for this source code, press the
Shift+Ctrl+F11 to open the Project Options. Inside the Linker border, change
the Max stack size, Max heap size and Image base values to 0x01000000,
0x01000000 and 0x04000000 respectively.
Now, the project is prepared to be run.

*/

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <fcntl.h>
#include <conio.h>
//Defines the Number of Patterns (examples) used to train the ANN.
#define NUMPAT 1000
//The number of artificial neurons in the INPUT LAYER
#define NUMIN  3
//The number of the artificial neurons in each HIDDEN LAYER
#define NUMHID 100
//The number of artificial neurons in the OUTPUT LAYER.
#define NUMOUT 1
//This is a random function defined by John. Returns a random number in [0,1]
//interval.
#define rando() ((double)rand()/RAND_MAX)

main() {
//Variables for recording weights in file-------------------------------------
FILE *record, *read;
double store;
int option;
```

```
   int    i, j, k, p, np, op, ranpat[NUMPAT+1], epoca;
   int    NumPattern = NUMPAT, NumInput = NUMIN, NumHidden = NUMHID, NumOutput = NUMOUT;


//Here is the association of the examples to the input matrix----------
   double Input[NUMPAT+1][NUMIN+1] =
{
//This is the input training set. Notice that its included a line of zeros and
//a collumn of zeros. This is the reason for the '+1' in maxtrix declaration.--
//Note: this training set is described before, in appendix A.
//End of input training set--------------------------------------------------
};
//---------------------------------------------------------------------------


//This is the target training set. Notice that its included a line of zeros and
//a collumn of zeros. This is the reason for the '+1' in maxtrix declaration.--
   double Target[NUMPAT+1][NUMOUT+1] =
{
//This is the the desired training set. To each input there is a desired
//result. For example: for the Input[1][] (the 0,0,0) is desired the
//Target[1][] (the 0); for the Input[1000][] (the 6.5,0.8,-7.2) is desired the
//Target[1000][] (the 1).---------------------------------------------------
//Note: this training set is described before, in appendix A.
//End of target training set------------------------------------------------
};
//---------------------------------------------------------------------------


//These matrixes define the ANN structure. The above Input[][] tells how many
//input's neurons the network will have. The above Target is related to
//Output[][] below, so they must have the same size. The SumHX[][] below will
//store the somatory of the related neuron value, where X is related to the
//number of the layer. The WeightXX[][] below stores the weight value relative
//to the layer described by XX. DeltaWeightXX[][] stores the difference value
//needed to correct the weights values in error backpropagation action. The
//alpha and eta accelerates the learning, like explained in the monograph.-----
//---------------------------------------------------------------------------
```

```
   double                    SumH1[NUMPAT+1][NUMHID+1],                    WeightIH1[NUMIN+1][NUMHID+1],
Hidden1[NUMPAT+1][NUMHID+1],SumH2[NUMPAT+1][NUMHID+1],                    WeightH1H2[NUMHID+1][NUMHID+1],
Hidden2[NUMPAT+1][NUMHID+1],SumH3[NUMPAT+1][NUMHID+1],                    WeightH2H3[NUMHID+1][NUMHID+1],
Hidden3[NUMPAT+1][NUMHID+1];
   double SumO[NUMPAT+1][NUMOUT+1], WeightH3O[NUMHID+1][NUMOUT+1], Output[NUMPAT+1][NUMOUT+1];
   double DeltaO[NUMOUT+1], SumDOW[NUMHID+1], DeltaH1[NUMHID+1], DeltaH2[NUMHID+1], DeltaH3[NUMHID+1];
   double                DeltaWeightIH1[NUMIN+1][NUMHID+1],                DeltaWeightH1H2[NUMHID+1][NUMHID+1],
DeltaWeightH2H3[NUMHID+1][NUMHID+1], DeltaWeightH3O[NUMHID+1][NUMOUT+1];
   double Erro, eta = 0.08, alpha = 0.15 , smallwt = 0.5;


fprintf(stdout, "\n\nUniCEUB - Centro Universitario de Brasilia\n") ;
fprintf(stdout, "Curso de Engenharia de Computacao\n") ;
fprintf(stdout, "Aluno: Eduardo Braga Dutra Rocha - RA 20114663.\n") ;
fprintf(stdout, "Programa para treinamento de Rede Neural Artificial.\n") ;
fprintf(stdout, "Tipo de RNA adotada: Rede MLP com algoritmo de retropropagacao de erro.\n") ;
fprintf(stdout, "Topologia da RNA: 3[INPUT], 3x100[HIDDEN], 1[OUTPUT], totalmente conectada.\n") ;
fprintf(stdout, "\n\n\nApos o treinamento utilizar os pesos gerados por este programa, gravados em \narquivos de texto, no ambiente de
simulacao Ambiente_Neural.exe.\n") ;

//Inicialization of deltaweight and weight-----------------------------------


   fprintf(stdout, "\n\nDeseja carregar os valores dos pesos a partir de algum arquivo? (s/n) ") ;
   option=getche();
 if (option == 's' || option == 'S')
       {


      /*WeightIH1[NUMIN+1][NUMHID+1]*/
      read = fopen("02_WeightIH1.txt","r");
      for(i=0;i<NUMIN+1;i++)
            for(j=0;j<NUMHID+1;j++)
                  fscanf(read,"%lf, ",&WeightIH1[i][j]);
      fclose(read);
      /* End of WeightIH1[NUMIN+1][NUMHID+1]*/
```

```
/*DeltaWeightIH1[NUMIN+1][NUMHID+1]*/
read = fopen("02_DeltaWeightIH1.txt","r");
for(i=0;i<NUMIN+1;i++)
        for(j=0;j<NUMHID+1;j++)
                fscanf(read,"%lf, ",&DeltaWeightIH1[i][j]);
fclose(read);
/* End of DeltaWeightIH1[NUMIN+1][NUMHID+1]*/

/*WeightH1H2[NUMHID+1][NUMHID+1]*/
read = fopen("02_WeightH1H2.txt","r");
for(i=0;i<NUMHID+1;i++)
        for(j=0;j<NUMHID+1;j++)
                fscanf(read,"%lf, ",&WeightH1H2[i][j]);
fclose(read);
/*End of WeightH1H2[NUMHID+1][NUMHID+1]*/

/*DeltaWeightH1H2[NUMHID+1][NUMHID+1]*/
read = fopen("02_DeltaWeightH1H2.txt","r");
for(i=0;i<NUMHID+1;i++)
        for(j=0;j<NUMHID+1;j++)
                fscanf(read,"%lf, ",&DeltaWeightH1H2[i][j]);
fclose(read);
/*End of DeltaWeightH1H2[NUMHID+1][NUMHID+1]*/

/*WeightH2H3[NUMHID+1][NUMHID+1]*/
read = fopen("02_WeightH2H3.txt","r");
for(i=0;i<NUMHID+1;i++)
        for(j=0;j<NUMHID+1;j++)
                fscanf(read,"%lf, ",&WeightH2H3[i][j]);
fclose(read);
/*End of WeightH2H3[NUMHID+1][NUMHID+1]*/

/*DeltaWeightH2H3[NUMHID+1][NUMHID+1]*/
read = fopen("02_DeltaWeightH2H3.txt","r");
for(i=0;i<NUMHID+1;i++)
```

```
            for(j=0;j<NUMHID+1;j++)
                fscanf(read,"%lf, ",&DeltaWeightH2H3[i][j]);
        fclose(read);
        /*End of DeltaWeightH2H3[NUMHID+1][NUMHID+1]*/

        /*WeightH3O[NUMHID+1][NUMOUT+1]*/
        read = fopen("02_WeightH3O.txt","r");
        for(i=0;i<NUMHID+1;i++)
            for(j=0;j<NUMOUT+1;j++)
                fscanf(read,"%lf, ",&WeightH3O[i][j]);
        fclose(read);
        /*End of WeightH3O[NUMHID+1][NUMOUT+1]*/

        /*DeltaWeightH3O[NUMHID+1][NUMOUT+1]*/
        read = fopen("02_DeltaWeightH3O.txt","r");
        for(i=0;i<NUMHID+1;i++)
            for(j=0;j<NUMOUT+1;j++)
                fscanf(read,"%lf, ",&DeltaWeightH3O[i][j]);
        fclose(read);
        /*End of DeltaWeightH3O[NUMHID+1][NUMOUT+1]*/

      } /*End if from option*/

//Else, if user doesnt want to load the weights values, put some random values
//to initialize the weights.----------------------------------------------------
   else
   {
   for( j = 1 ; j <= NumHidden ; j++ ) {    /* initializes WeightIH1 and DeltaWeightIH1 */
      for( i = 0 ; i <= NumInput ; i++ ) {
         DeltaWeightIH1[i][j] = 0.0 ;
         WeightIH1[i][j] = 2.0 * ( rando() - 0.5 ) * smallwt ;
      }
   }
   for( j = 1 ; j <= NumHidden ; j++ ) {    /* initializes WeightH1H2 and DeltaWeightH1H2 */
      for( i = 0 ; i <= NumHidden ; i++ ) {
```

```
            DeltaWeightH1H2[i][j] = 0.0 ;
            WeightH1H2[i][j] = 2.0 * ( rando() - 0.5 ) * smallwt ;
        }
    }
    for( j = 1 ; j <= NumHidden ; j++ ) {    /* initializes WeightH2H3 and DeltaWeightH2H3 */
        for( i = 0 ; i <= NumHidden ; i++ ) {
            DeltaWeightH2H3[i][j] = 0.0 ;
            WeightH2H3[i][j] = 2.0 * ( rando() - 0.5 ) * smallwt ;
        }
    }
    for( k = 1 ; k <= NumOutput ; k ++ ) {    /* initializes WeightHO and DeltaWeightHO */
        for( j = 0 ; j <= NumHidden ; j++ ) {
            DeltaWeightH3O[j][k] = 0.0 ;
            WeightH3O[j][k] = 2.0 * ( rando() - 0.5 ) * smallwt ;
        }
    }
    }/*End of 'else' from if that asks about loading weights*/


//Asks about loading the eta/alpha values-------------------------------------
    fprintf(stdout, "\n\nDeseja carregar os valores de eta/alpha partir de algum arquivo? (s/n) ") ;
    option=getche();
if(option == 's' || option == 'S')
{
        /* eta */
        read = fopen("02_eta.txt","r");
        fscanf(read,"%lf, ",&eta);
        fclose(read);
        /* End of eta */


        /* alpha */
        read = fopen("02_alpha.txt","r");
        fprintf(read,"%lf, ",&alpha);
        fclose(read);
        /* End of alpha */
```

```
}




puts("\n\nIniciando treinamento.\nA qualquer momento pressione P para terminar o treinamento ao fim da epoca.");
puts("Pressione alguma tecla para exibir o erro atual.");
printf("Considerando uma matriz quadrada de 3x%d neuronios ocultos.\n",NUMHID);



// Epochs start now-------------------------------------------------------------
//Iterates the wheights updates-----------------------------------------------
   for( epoca = 0 ; epoca < 100000000 ; epoca++) {


      //Get out the loop if P is pressed.-----------------------------------
      if(kbhit())
            if (getch()=='P') break;
                                    else fprintf(stdout, "\nepoca %-5d :   Erro = %f", epoca, Erro) ;
//-------------------------------------------------------------------------


//Utilize this for randomizing the orders of individuals----------------------
      for( p = 1 ; p <= NumPattern ; p++ ) {
         ranpat[p] = p ;
      }
      for( p = 1 ; p <= NumPattern ; p++) {
         np = p + rando() * ( NumPattern + 1 - p ) ;
         op = ranpat[p] ; ranpat[p] = ranpat[np] ; ranpat[np] = op ;
      }
      Erro = 0.0 ;


//For the Pattern X, do: repeat it for all the training patterns.--------------
      for( np = 1 ; np <= NumPattern ; np++ ) {
         p = ranpat[np];


//Hidden 1 unit activations
```

```
        for( j = 1 ; j <= NumHidden ; j++ ) {
          SumH1[p][j] = WeightIH1[0][j] ;  //This is the BIAS!!
          for( i = 1 ; i <= NumInput ; i++ ) {
            SumH1[p][j] += Input[p][i] * WeightIH1[i][j] ;
          }
          if(SumH1[p][j] > -14.0 && SumH1[p][j] < 14.0)
              Hidden1[p][j] = 1.0/(1.0 + exp(-SumH1[p][j])) ;
          else Hidden1[p][j] = (SumH1[p][j] < -14.0)? 0 : 1;
        }
//Hidden 2 unit activations
        for( j = 1 ; j <= NumHidden ; j++ ) {
          SumH2[p][j] = WeightH1H2[0][j] ;
          for( i = 1 ; i <= NumHidden ; i++ ) {
            if((SumH2[p][j] + (Hidden1[p][i] * WeightH1H2[i][j])) > -1.69e+308 && (SumH2[p][j] + (Hidden1[p][i] *
WeightH1H2[i][j])) < 1.69e+308)
              SumH2[p][j] += Hidden1[p][i] * WeightH1H2[i][j] ;
            else SumH3[p][j] = ((SumH2[p][j] + (Hidden1[p][i] * WeightH1H2[i][j])) < -1.69e+308)? -1.69e+308 : 1.69e+308;
           }
          if(SumH2[p][j] > -14.0 && SumH2[p][j] < 14.0)
              Hidden2[p][j] = 1.0/(1.0 + expl(-SumH2[p][j])) ;
          else Hidden2[p][j] = (SumH2[p][j] < -14.0)? 0 : 1;



        }
//Hidden 3 unit activations
        for( j = 1 ; j <= NumHidden ; j++ ) {
          SumH3[p][j] = WeightH2H3[0][j] ;
          for( i = 1 ; i <= NumHidden ; i++ ) {
            if((SumH3[p][j] + (Hidden2[p][i] * WeightH2H3[i][j])) > -1.69e+308 && (SumH3[p][j] + (Hidden2[p][i] *
WeightH2H3[i][j])) < 1.69e+308)
            SumH3[p][j] += Hidden2[p][i] * WeightH2H3[i][j] ;
            else SumH3[p][j] = ((SumH3[p][j] + (Hidden2[p][i] * WeightH2H3[i][j])) < -1.69e+308)? -1.69e+308 : 1.69e+308;
          }
          if(SumH3[p][j] > -14.0 && SumH3[p][j] < 14.0)
              Hidden3[p][j] = 1.0/(1.0 + exp(-SumH3[p][j])) ;
```

```
            else Hidden3[p][j] = (SumH3[p][j] < -14.0)? 0 : 1;

         }
//Output unit activations
         for( k = 1 ; k <= NumOutput ; k++ ) {    /* compute output unit activations and errors */
            SumO[p][k] = WeightH3O[0][k] ;
            for( j = 1 ; j <= NumHidden ; j++ ) {
               SumO[p][k] += Hidden3[p][j] * WeightH3O[j][k] ;
            }
            Output[p][k] = 1.0/(1.0 + exp(-SumO[p][k])) ;   /* Sigmoidal Outputs */
//Yet inside Output FOR, computing the first error based on the Output Layer
            Erro += 0.5 * (Target[p][k] - Output[p][k]) * (Target[p][k] - Output[p][k]) ;   /* SSE */
            /*              Erro -= ( Target[p][k] * log( Output[p][k] ) + ( 1.0 - Target[p][k] ) * log( 1.0 - Output[p][k] ) ) ;     Cross-Entropy
Error */
            DeltaO[k] = (Target[p][k] - Output[p][k]) * Output[p][k] * (1.0 - Output[p][k]) ;   /* Sigmoidal Outputs, SSE */
            /*            DeltaO[k] = Target[p][k] - Output[p][k];     Sigmoidal Outputs, Cross-Entropy Error */
            /*            DeltaO[k] = Target[p][k] - Output[p][k];     Linear Outputs, SSE */
         }
//Beginning of Backpropagation to Hidden Layer 3
         for( j = 1 ; j <= NumHidden ; j++ ) {
            SumDOW[j] = 0.0 ;
            for( k = 1 ; k <= NumOutput ; k++ ) {
               SumDOW[j] += WeightH3O[j][k] * DeltaO[k] ;
            }
            DeltaH3[j] = SumDOW[j] * Hidden3[p][j] * (1.0 - Hidden3[p][j]) ;
         }
//Beginning of Backpropagation to Hidden Layer 2
         for( j = 1 ; j <= NumHidden ; j++ ) {
            SumDOW[j] = 0.0 ;
            for( k = 1 ; k <= NumHidden ; k++ ) {
               SumDOW[j] += WeightH2H3[j][k] * DeltaH3[k] ;
            }
            DeltaH2[j] = SumDOW[j] * Hidden2[p][j] * (1.0 - Hidden2[p][j]) ;
         }
//Beginning of Backpropagation to Hidden Layer 1
         for( j = 1 ; j <= NumHidden ; j++ ) {
```

```
            SumDOW[j] = 0.0 ;
            for( k = 1 ; k <= NumHidden ; k++ ) {
               SumDOW[j] += WeightH1H2[j][k] * DeltaH2[k] ;
            }
            DeltaH1[j] = SumDOW[j] * Hidden1[p][j] * (1.0 - Hidden1[p][j]) ;
         }
//Update of Weights between Layer Input and Layer Hidden 1
         for( j = 1 ; j <= NumHidden ; j++ ) {
            DeltaWeightIH1[0][j] = eta * DeltaH1[j] + alpha * DeltaWeightIH1[0][j] ;
            WeightIH1[0][j] += DeltaWeightIH1[0][j] ;
            for( i = 1 ; i <= NumInput ; i++ ) {
               DeltaWeightIH1[i][j] = eta * Input[p][i] * DeltaH1[j] + alpha * DeltaWeightIH1[i][j];
               WeightIH1[i][j] += DeltaWeightIH1[i][j] ;
            }
         }


//Update of weights between Layer Hidden 2 and Hidden 1
         for( k = 1 ; k <= NumHidden ; k ++ ) {
            DeltaWeightH1H2[0][k] = eta * DeltaH2[k] + alpha * DeltaWeightH1H2[0][k] ;
            WeightH1H2[0][k] += DeltaWeightH1H2[0][k] ;
            for( j = 1 ; j <= NumHidden ; j++ ) {
               DeltaWeightH1H2[j][k] = eta * Hidden1[p][j] * DeltaH2[k] + alpha * DeltaWeightH1H2[j][k] ;
               WeightH1H2[j][k] += DeltaWeightH1H2[j][k] ;
            }
         }
//Update of weights between Layer Hidden 3 and Hidden 2
         for( k = 1 ; k <= NumHidden ; k ++ ) {
            DeltaWeightH2H3[0][k] = eta * DeltaH3[k] + alpha * DeltaWeightH2H3[0][k] ;
            WeightH2H3[0][k] += DeltaWeightH2H3[0][k] ;
            for( j = 1 ; j <= NumHidden ; j++ ) {
               DeltaWeightH2H3[j][k] = eta * Hidden2[p][j] * DeltaH3[k] + alpha * DeltaWeightH2H3[j][k] ;
               WeightH2H3[j][k] += DeltaWeightH2H3[j][k] ;
            }
         }
//Update of weights between Layer Hidden 3 and Output
```

```
            for( k = 1 ; k <= NumOutput ; k ++ ) {
               DeltaWeightH3O[0][k] = eta * DeltaO[k] + alpha * DeltaWeightH3O[0][k] ;
               WeightH3O[0][k] += DeltaWeightH3O[0][k] ;
               for( j = 1 ; j <= NumHidden ; j++ ) {
                  DeltaWeightH3O[j][k] = eta * Hidden3[p][j] * DeltaO[k] + alpha * DeltaWeightH3O[j][k] ;
                  WeightH3O[j][k] += DeltaWeightH3O[j][k] ;
               }
            }
         } //Next Pattern!

         if( epoca%100 == 0 ) fprintf(stdout, "\nepoca %-5d :   Erro = %f", epoca, Erro) ;
//Stops the learning process if error is very small.
         if( Erro < 0.0004 ) break ;
      }

      fprintf(stdout, "\n\nInformacoes da Rede Neural Artificial - EPOCA FINAL: %d\n\nPat\t", epoca) ;   /* print network outputs */
      for( i = 1 ; i <= NumInput ; i++ ) {
         fprintf(stdout, "Entrada%-4d\t", i) ;
      }
      for( k = 1 ; k <= NumOutput ; k++ ) {
         fprintf(stdout, "Desejado%-4d\tSaida%-4d\t", k, k) ;
      }
      for( p = 1 ; p <= NumPattern ; p++ ) {
      fprintf(stdout, "\n%d\t", p) ;
         for( i = 1 ; i <= NumInput ; i++ ) {
            fprintf(stdout, "%f\t", Input[p][i]) ;
         }
         for( k = 1 ; k <= NumOutput ; k++ ) {
            fprintf(stdout, "%f\t%f\t", Target[p][k], Output[p][k]) ;
         }
      }

/* Procedure for recording the values of the weights in a text file */
      fprintf(stdout, "\n\nDeseja gravar em arquivo os valores dos pesos? (s/n)\n\n") ;
      option=getch();
```

```
if (option == 's' || option == 'S' || option == 'P')
    {
    /*WeightIH1[NUMIN+1][NUMHID+1]*/
    record = fopen("02_WeightIH1.txt","w");
    for(i=0;i<NUMIN+1;i++)
          for(j=0;j<NUMHID+1;j++)
                fprintf(record,"%lf, ",WeightIH1[i][j]);
    fclose(record);
    /* End of WeightIH1[NUMIN+1][NUMHID+1]*/

    /*DeltaWeightIH1[NUMIN+1][NUMHID+1]*/
    record = fopen("02_DeltaWeightIH1.txt","w");
    for(i=0;i<NUMIN+1;i++)
          for(j=0;j<NUMHID+1;j++)
                fprintf(record,"%lf, ",DeltaWeightIH1[i][j]);
    fclose(record);
    /* End of DeltaWeightIH1[NUMIN+1][NUMHID+1]*/

    /*WeightH1H2[NUMHID+1][NUMHID+1]*/
    record = fopen("02_WeightH1H2.txt","w");
    for(i=0;i<NUMHID+1;i++)
          for(j=0;j<NUMHID+1;j++)
                fprintf(record,"%lf, ",WeightH1H2[i][j]);
    fclose(record);
    /*End of WeightH1H2[NUMHID+1][NUMHID+1]*/

    /*DeltaWeightH1H2[NUMHID+1][NUMHID+1]*/
    record = fopen("02_DeltaWeightH1H2.txt","w");
    for(i=0;i<NUMHID+1;i++)
          for(j=0;j<NUMHID+1;j++)
                fprintf(record,"%lf, ",DeltaWeightH1H2[i][j]);
    fclose(record);
    /*End of DeltaWeightH1H2[NUMHID+1][NUMHID+1]*/

    /*WeightH2H3[NUMHID+1][NUMHID+1]*/
```

```
record = fopen("02_WeightH2H3.txt","w");
for(i=0;i<NUMHID+1;i++)
        for(j=0;j<NUMHID+1;j++)
                fprintf(record,"%lf, ",WeightH2H3[i][j]);
fclose(record);
/*End of WeightH2H3[NUMHID+1][NUMHID+1]*/

/*DeltaWeightH2H3[NUMHID+1][NUMHID+1]*/
record = fopen("02_DeltaWeightH2H3.txt","w");
for(i=0;i<NUMHID+1;i++)
        for(j=0;j<NUMHID+1;j++)
                fprintf(record,"%lf, ",DeltaWeightH2H3[i][j]);
fclose(record);
/*End of DeltaWeightH2H3[NUMHID+1][NUMHID+1]*/

/*WeightH3O[NUMHID+1][NUMOUT+1]*/
record = fopen("02_WeightH3O.txt","w");
for(i=0;i<NUMHID+1;i++)
        for(j=0;j<NUMOUT+1;j++)
                fprintf(record,"%lf, ",WeightH3O[i][j]);
fclose(record);
/*End of WeightH3O[NUMHID+1][NUMOUT+1]*/

/*DeltaWeightH3O[NUMHID+1][NUMOUT+1]*/
record = fopen("02_DeltaWeightH3O.txt","w");
for(i=0;i<NUMHID+1;i++)
        for(j=0;j<NUMOUT+1;j++)
                fprintf(record,"%lf, ",DeltaWeightH3O[i][j]);
fclose(record);
/*End of DeltaWeightH3O[NUMHID+1][NUMOUT+1]*/

/* eta */
record = fopen("02_eta.txt","w");
fprintf(record,"%lf, ",eta);
fclose(record);
```

```
        /* End of eta */


        /* alpha */
        record = fopen("02_alpha.txt","w");
        fprintf(record,"%lf, ",alpha);
        fclose(record);
        /* End of alpha */


        }


puts("Pressione alguma tecla para sair.");
getch();
   return 1 ;
}


/*************************************************************************/
```

Apêndice C – Código fonte do programa do ambiente de simulação

C.1) Código do arquivo Ambiente_Neural_Codigo.cpp

```
/*
UniCEUB - Centro Universitario de Brasilia
FAET - Faculdade de Exatas e Tecnologia
Curso de Engenharia de Computacao
70790-075 - Brasilia - DF, Brazil.



ALUNO: EDUARDO BRAGA DUTRA ROCHA


AMBIENTE DE SIMULACAO DE REDES NEURAIS ARTIFICIAIS PARA O CONTROLE DE UM MOVEL
[ROCHA, 2006]

O uso desse codigo fonte pressupoe um treinamento anterior, feito com a
utilizacao do programa desenvolvido para o treinamento da RNA desenhada.


*/
//---------------------------------------------------------------------------

#include <vcl.h>
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<dos.h>
#pragma hdrstop
#include <math.h>
#include "Ambiente_Neural_Codigo.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"

//This is the max angle of the car's wheels.
#define MAXWHEELANGLE 30
//This is the grid scale. It means that the grid is 5 times greater than the
//grid used to train the ANN
#define GRIDSCALE 5
//This is the car's ray
#define CARRAY 10
#define PI 3.141592
//Defines the Number of Patterns (examples) used to train the ANN.
#define NUMPAT 1
//The number of artificial neurons in the INPUT LAYER
#define NUMIN  3
//The number of the artificial neurons in the squared HIDDEN LAYER (it means
//that the HIDDEN LAYER will be a layer of NUMHID^2 neurons).
#define NUMHID 100
//The number of artificial neurons in the OUTPUT LAYER.
#define NUMOUT 1


//This is a random function defined by John. Returns a random number in [0,1]
//interval.
#define rando() ((double)rand()/RAND_MAX)


TEnvironment *Environment;

//---------------------------------------------------------------------------
__fastcall TEnvironment::TEnvironment(TComponent* Owner)
```

```
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------

//Declaring Functions ------------------------------------------------------

//This function collects data from the environment and activates the neurocalc
//function.----------------------------------------------------------------
void executarRedeNeural(void);
//---------------------------------------------------------------------------

//This is a function used to move the car without activating the RNA code.----
void movecar(void);
//---------------------------------------------------------------------------

//This is the function responsible to execute the neural code. Its a part of
//the ANN training program.
double neurocalc(double X, double Y, double Phi);
//---------------------------------------------------------------------------

//This function decodes the neural network's output. It's a number between 0
//and 1.-------------------------------------------------------------------
double discretizaangulo(double angulo);
//---------------------------------------------------------------------------

//This function places the mobile in a random place in grid.----------------
void randomplace();
//---------------------------------------------------------------------------

//This function is responsible to calculate the ideal value for the angle that
//the mobile would need.---------------------------------------------------
double idealvalue(double X, double Y, double Phi);
//---------------------------------------------------------------------------

//This function is responsible to limit the angle between 0 and 180.----------
double limit(double angle);
//---------------------------------------------------------------------------

//Responsible to limit the wheel's angle between the -30 and 30 degrees range.
double maxangle(double angle);
//---------------------------------------------------------------------------

//Show the About screen.----------------------------------------------------
void showAbout(void);
//---------------------------------------------------------------------------

//Hide the About screen.----------------------------------------------------
void hideAbout(void);
//---------------------------------------------------------------------------


//Global variables----------------------------------------------------------

   int    NumInput = NUMIN, NumHidden = NUMHID, NumOutput = NUMOUT;

   double NInput[NUMPAT+1][NUMIN+1]=
   {0,0,0,0,
    0,0,0,0};

   double SumH1[NUMPAT+1][NUMHID+1], WeightIH1[NUMIN+1][NUMHID+1],
Hidden1[NUMPAT+1][NUMHID+1],SumH2[NUMPAT+1][NUMHID+1], WeightH1H2[NUMHID+1][NUMHID+1],
Hidden2[NUMPAT+1][NUMHID+1],SumH3[NUMPAT+1][NUMHID+1], WeightH2H3[NUMHID+1][NUMHID+1],
Hidden3[NUMPAT+1][NUMHID+1];
   double SumO[NUMPAT+1][NUMOUT+1], WeightH3O[NUMHID+1][NUMOUT+1],NOutput[NUMPAT+1][NUMOUT+1];
   double eta, alpha;

//---------------------------------------------------------------------------
```

```
//---------------------------------------------------------------------------
void __fastcall TEnvironment::DisplayHint(TObject *Sender)
{
   StatusBar1->Panels->Items[1]->Text =
      GetLongHint(Application->Hint);
}
//---------------------------------------------------------------------------

void __fastcall TEnvironment::Button1Click(TObject *Sender)
{
movecar();
}
//---------------------------------------------------------------------------

//This is a function used to move the car without activating the RNA code.----
void movecar()
{

double x, y, angulo;

//Receives the X, Y and angle values from the Text Boxes.----------------------
x = atoi(Environment->Edit1->Text.c_str());
y = atoi(Environment->Edit2->Text.c_str());
angulo = atoi(Environment->Edit3->Text.c_str());
//---------------------------------------------------------------------------

//Verifying if x, y and angle values are acceptable.---------------------------
x = (x<0)?0:x;
y = (y<0)?0:y;
angulo = (angulo<0)?(360-pow(pow(angulo,2),0.5)):angulo;
x = (x>500)?500:x;
y = (y>500)?500:y;
angulo = (angulo>360)?(360-angulo):angulo;
//---------------------------------------------------------------------------

//Verifying if the objective is already concluded.-----------------------------
if(x>=460 && (y>=210 && y<=290))
MessageBox(0,"Objetivo concluído.","Fim", MB_ICONINFORMATION + MB_OK);
//---------------------------------------------------------------------------

//Verifying if there is a collision.-------------------------------------------
else if(x>=500)
{
 MessageBox(0,"Houve colisão.\rMovimentando o móvel para um ponto aleatório.", "Colisão detectada", MB_ICONEXCLAMATION +
MB_OK);
 randomplace();
}
//---------------------------------------------------------------------------

//Assuming that the car can move...--------------------------------------------
else
      {
//Calculates the new position for the car, using trigonometry.-----------------
      x = cos(angulo*3.1415/180)*CARRAY + x;
      y = sin(angulo*3.1415/180)*CARRAY + y;
//---------------------------------------------------------------------------

//Validating the position calculated.------------------------------------------
      x = (x<0)?0:x;
      y = (y<0)?0:y;
      x = (x>500)?500:x;
      y = (y>500)?500:y;
//---------------------------------------------------------------------------

//Moves the car, considering the car's ray.------------------------------------
      Environment->Car->Left = x-CARRAY;
      Environment->Car->Top  = y-CARRAY;

      Environment->Edit1->Text = x;
```

```
        Environment->Edit2->Text = y;
        }// End of last 'else'
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
}
//---------------------------------------------------------------------------

//This is the function responsible for loading the weights values in their
//respective matrixes.-------------------------------------------------------
void __fastcall TEnvironment::Carregarpesos1Click(TObject *Sender)
{
  MessageBox(0, "Bem Vindo.\rIniciando carregamento dos pesos.", "^^!", MB_OK);
/* Variables for recording weights in file*/

//Just a procedure for loading the weights into matrixes.----------------------
FILE *read;
   int    i, j, k ;
       Environment->StatusBar1->Panels->Items[0]->Text = "0%";
      /*WeightIH1[NUMIN+1][NUMHID+1]*/
                read = fopen("02_WeightIH1.txt","r");
       for(i=0;i<NUMIN+1;i++)
             for(j=0;j<NUMHID+1;j++)
                   fscanf(read,"%lf, ",&WeightIH1[i][j]);
       fclose(read);
       /* End of WeightIH1[NUMIN+1][NUMHID+1]*/
       Environment->StatusBar1->Panels->Items[0]->Text = "15%";
       /*WeightH1H2[NUMHID+1][NUMHID+1]*/

       read = fopen("02_WeightH1H2.txt","r");
       for(i=0;i<NUMHID+1;i++)
             for(j=0;j<NUMHID+1;j++)
                   fscanf(read,"%lf, ",&WeightH1H2[i][j]);
       fclose(read);
       /*End of WeightH1H2[NUMHID+1][NUMHID+1]*/
       Environment->StatusBar1->Panels->Items[0]->Text = "30%";

       /*WeightH2H3[NUMHID+1][NUMHID+1]*/
       read = fopen("02_WeightH2H3.txt","r");
       for(i=0;i<NUMHID+1;i++)
             for(j=0;j<NUMHID+1;j++)
                   fscanf(read,"%lf, ",&WeightH2H3[i][j]);
       fclose(read);
       /*End of WeightH2H3[NUMHID+1][NUMHID+1]*/
       Environment->StatusBar1->Panels->Items[0]->Text = "45%";

       /*WeightH3O[NUMHID+1][NUMOUT+1]*/
       read = fopen("02_WeightH3O.txt","r");
       for(i=0;i<NUMHID+1;i++)
             for(j=0;j<NUMOUT+1;j++)
                   fscanf(read,"%lf, ",&WeightH3O[i][j]);
       fclose(read);
       /*End of WeightH3O[NUMHID+1][NUMOUT+1]*/
       Environment->StatusBar1->Panels->Items[0]->Text = "60%";

       /* eta */

       read = fopen("02_eta.txt","r");
       fscanf(read,"%lf, ",&eta);
       fclose(read);
       /* End of eta */

       Environment->StatusBar1->Panels->Items[0]->Text = "75%";
       /* alpha */
       read = fopen("02_alpha.txt","r");
       fprintf(read,"%lf, ",&alpha);
       fclose(read);
       /* End of alpha */
       _fcloseall();
       Environment->StatusBar1->Panels->Items[0]->Text = "100%";
```

```
   MessageBox(0, "Fim do carregamento.", "\'^^!", MB_OK);
   Environment->StatusBar1->Panels->Items[0]->Text = "";
//---------------------------------------------------------------------------
}
//---------------------------------------------------------------------------

//This is the function responsible to execute the neural code. Its a part of
//the ANN training program. The difference is not implementing the error
//backpropagation code.
double neurocalc(double X, double Y, double Phi) {

double value;
int    i, j, k ;

NInput[1][1] = X/10;
NInput[1][2] = Y/10;
NInput[1][3] = Phi/10;


//Hidden 1 unit activations
        for( j = 1 ; j <= NumHidden ; j++ ) {    /* compute hidden unit activations */
           SumH1[1][j] = WeightIH1[0][j] ;  //This is the BIAS!!
           for( i = 1 ; i <= NumInput ; i++ ) {
              SumH1[1][j] += NInput[1][i] * WeightIH1[i][j] ;
           }
           //It's a clause to block the expl() function from blowing up
           if(SumH1[1][j] > -14.0 && SumH1[1][j] < 14.0)
                Hidden1[1][j] = 1.0/(1.0 + exp(-SumH1[1][j])) ;
           else Hidden1[1][j] = (SumH1[1][j] < -14.0)? 0 : 1;
        }
//Hidden 2 unit activations
        for( j = 1 ; j <= NumHidden ; j++ ) {    /* compute hidden unit activations */
           SumH2[1][j] = WeightH1H2[0][j] ;
           for( i = 1 ; i <= NumHidden ; i++ ) {
              if((SumH2[1][j] + (Hidden1[1][i] * WeightH1H2[i][j])) > -1.69e+308 && (SumH2[1][j] + (Hidden1[1][i] *
WeightH1H2[i][j])) < 1.69e+308)
                 SumH2[1][j] += Hidden1[1][i] * WeightH1H2[i][j] ;
              else SumH3[1][j] = ((SumH2[1][j] + (Hidden1[1][i] * WeightH1H2[i][j])) < -1.69e+308)? -1.69e+308 : 1.69e+308;
              }
           //It's a clause to block the expl() function from blowing up
           if(SumH2[1][j] > -14.0 && SumH2[1][j] < 14.0)
                Hidden2[1][j] = 1.0/(1.0 + expl(-SumH2[1][j])) ;
           else Hidden2[1][j] = (SumH2[1][j] < -14.0)? 0 : 1;


        }
//Hidden 3 unit activations
        for( j = 1 ; j <= NumHidden ; j++ ) {    /* compute hidden unit activations */
           SumH3[1][j] = WeightH2H3[0][j] ;
           for( i = 1 ; i <= NumHidden ; i++ ) {
              if((SumH3[1][j] + (Hidden2[1][i] * WeightH2H3[i][j])) > -1.69e+308 && (SumH3[1][j] + (Hidden2[1][i] *
WeightH2H3[i][j])) < 1.69e+308)
                 SumH3[1][j] += Hidden2[1][i] * WeightH2H3[i][j] ;
                 else SumH3[1][j] = ((SumH3[1][j] + (Hidden2[1][i] * WeightH2H3[i][j])) < -1.69e+308)? -1.69e+308 : 1.69e+308;
           }
           //It's a clause to block the expl() function from blowing up
           if(SumH3[1][j] > -14.0 && SumH3[1][j] < 14.0)
                Hidden3[1][j] = 1.0/(1.0 + exp(-SumH3[1][j])) ;
           else Hidden3[1][j] = (SumH3[1][j] < -14.0)? 0 : 1;
        }
//Output unit activations
        for( k = 1 ; k <= NumOutput ; k++ ) {    /* compute output unit activations and errors */
           SumO[1][k] = WeightH3O[0][k] ;
           for( j = 1 ; j <= NumHidden ; j++ ) {
              SumO[1][k] += Hidden3[1][j] * WeightH3O[j][k] ;
           }
           NOutput[1][k] = 1.0/(1.0 + exp(-SumO[1][k])) ;   /* Sigmoidal Outputs */
        }
//Updates the screen text box
Environment->t1->Text=NOutput[1][1];
```

```
value=NOutput[1][1];

return(value);
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
void __fastcall TEnvironment::FormCreate(TObject *Sender)
{
randomplace();
Environment->About->Top = 0;
Environment->About->Left = 0;
Application->OnHint = DisplayHint;

}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
void __fastcall TEnvironment::ExecutaRedeClick(TObject *Sender)
{
executarRedeNeural();
}
//---------------------------------------------------------------------------

//This function collects data from the environment and activates the neurocalc
//function.----------------------------------------------------------------
void executarRedeNeural(void)
{
double x, y, angulo, angulonovo, Xo, Yo, Yp, ideal, t=0;

do{
//Receives the X, Y and angle values from the Text Boxes.----------------------
x = atoi(Environment->Edit1->Text.c_str());
y = atoi(Environment->Edit2->Text.c_str());
angulo = atoi(Environment->Edit3->Text.c_str());
//---------------------------------------------------------------------------

//Verifying if x, y and angle values are acceptable.---------------------------
x = (x<0)?0:x;
y = (y<0)?0:y;
angulo = (angulo<0)?(360-pow(pow(angulo,2),0.5)):angulo;
x = (x>500)?500:x;
y = (y>500)?500:y;
angulo = (angulo>360)?(360-angulo):angulo;
//---------------------------------------------------------------------------

//Defining standard values for the objectives.
Xo = 460;
Yo = 210;
Yp = 270;
//---------------------------------------------------------------------------

//If the objective must be simplified, change the default values for the
//objectives----------------------------------------------------------------
if(Environment->AreaGol->Checked == False)
{
Xo = 500;
Yo = 240;
Yp = 260;
}
//---------------------------------------------------------------------------

//Verifying if the objective is already concluded.-----------------------------
if(x>=Xo && (y>=Yo && y<=Yp))
{MessageBox(0,"Objetivo concluído.","Fim", MB_ICONINFORMATION + MB_OK);
 break;
 }
//Verifying if there is a collision--------------------------------------------
if(Environment->DetectaColisao->Checked == True)
```

```
if(x>=500 || x<=0 || y>=500 ||y<=0)
{
 MessageBox(0,"Houve colisão.\rMovimentando o móvel para um ponto aleatório.", "Colisão detectada", MB_ICONEXCLAMATION +
MB_OK);
 randomplace();
 break;
//---------------------------------------------------------------------------

}
//Assuming that the car can move...-----------------------------------------

//Calculating the ideal value before----------------------------------------
      ideal = idealvalue(x/GRIDSCALE,y/GRIDSCALE,angulo);

//Asks for the ANN the angle correction and decodes the output---------------
      angulonovo = neurocalc(x/GRIDSCALE,y/GRIDSCALE,angulo);
      angulonovo = discretizaangulo(angulonovo);
//---------------------------------------------------------------------------

//Prints the angle correction in screen-------------------------------------
      Environment->t2->Text = angulonovo;
//---------------------------------------------------------------------------

//Calculates the new angle to the car be moved------------------------------
      angulo = angulo+angulonovo;
//---------------------------------------------------------------------------

//Calculates the new position, using trigonometry---------------------------
      x = cos(angulo*3.1415/180)*CARRAY + x;
      y = sin(angulo*3.1415/180)*CARRAY + y;
//---------------------------------------------------------------------------

//Validating some values...-------------------------------------------------
      x = (x<0)?0:x;
      y = (y<0)?0:y;
      x = (x>500)?500:x;
      y = (y>500)?500:y;
//---------------------------------------------------------------------------

//Moving the car in the grid------------------------------------------------
      Environment->Car->Left = x-CARRAY;
      Environment->Car->Top  = y-CARRAY;
//---------------------------------------------------------------------------

//Updating the values in environment----------------------------------------
      Environment->Edit1->Text = x;
      Environment->Edit2->Text = y;
      Environment->Edit3->Text = angulo;
      Environment->Edit4->Text = ideal;
      Environment->Edit5->Text = ideal-angulonovo;
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
}while (t<0);

}
//---------------------------------------------------------------------------

//This function decodes the neural network's output. It's a number between 0
//and 1.-------------------------------------------------------------------
double discretizaangulo(double angulo)
{
if(angulo<0)
{
char msg[100];
      AnsiString buf[30];
      strcpy(msg,"Saída da rede não planejada.\r");
      Environment->Buffer->Text = angulo;
      strcat(msg,Environment->Buffer->Text.c_str());
      strcat(msg," foi o valor de saída da rede.\rÂngulo será -30.");
      MessageBox(0, msg, "Saída não planejada", MB_ICONERROR + MB_OK);
```

```
        angulo = -30;
}
else if(angulo>=0 && angulo<0.1)
        angulo = -30;
else if(angulo>=0.1 && angulo<0.275)
        angulo = -20;
else if(angulo>=0.275 && angulo<0.425)
        angulo = -10;
else if(angulo>=0.425 && angulo<=0.625)
        angulo = 0;
else if(angulo>0.625 && angulo<=0.725)
        angulo = 10;
else if(angulo>0.725 && angulo<=0.9)
        angulo = 20;
else if(angulo>0.9 && angulo<=1)
        angulo = 30;
else if(angulo>1)
        {
        char msg[100];
        AnsiString buf[30];
        strcpy(msg,"Saída da rede não planejada.\r");
        Environment->Buffer->Text = angulo;
        strcat(msg,Environment->Buffer->Text.c_str());
        strcat(msg," foi o valor de saída da rede.\rÂngulo será 30.");
        MessageBox(0, msg, "Saída não planejada", MB_ICONERROR + MB_OK);
        angulo = 30;
        }


return angulo;
}
//---------------------------------------------------------------------------



//This function places the mobile in a random place in grid.--------------------
void randomplace()
{

Environment->Edit1->Text = 500*(rando());
Environment->Edit2->Text = 500*(rando());
Environment->Edit3->Text = 359*(rando()-0.5);
Environment->Car->Left = atoi(Environment->Edit1->Text.c_str())-CARRAY;
Environment->Car->Top = atoi(Environment->Edit2->Text.c_str())-CARRAY;

}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
void __fastcall TEnvironment::Button2Click(TObject *Sender)
{
randomplace();
}
//---------------------------------------------------------------------------

//This function is responsible to calculate the ideal value for the angle that
//the mobile would need.-----------------------------------------------------
double idealvalue(double X, double Y, double Phi)
{

double value ;
Y = (Y<0)?0:(Y>=50)?49.9:0;
X = (X<0)?0:(X>=100)?99.9:0;
value = (atan((50-Y)/(100-X))*(180/PI))-Phi;
value = limit(value);
value = maxangle(value);
return(value);
}
//---------------------------------------------------------------------------
```

```
//This function is responsible to limit the angle between 0 and 180.----------
double limit(double angle)
{

return (angle>180)?-(360-angle):((angle<-180)?360-pow(pow(angle,2),0.5):angle);

}

double maxangle(double angle)
{
return ((angle<-MAXWHEELANGLE)?-MAXWHEELANGLE:((angle>MAXWHEELANGLE)?MAXWHEELANGLE:angle));
}
//---------------------------------------------------------------------------


//Show the About screen.----------------------------------------------------
void showAbout(void)
{
Environment->About->Visible = True;
}
//---------------------------------------------------------------------------

//Hide the About screen.----------------------------------------------------
void hideAbout(void)
{
Environment->About->Visible = False;
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
void __fastcall TEnvironment::SobreoNeuroSada1Click(TObject *Sender)
{
if(SobreoNeuroSada1->Checked == False)
{showAbout();
SobreoNeuroSada1->Checked = True;}
else
{
hideAbout();
SobreoNeuroSada1->Checked = False;
}

}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
void __fastcall TEnvironment::Button3Click(TObject *Sender)
{
hideAbout();
SobreoNeuroSada1->Checked = False;
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
void __fastcall TEnvironment::Button4Click(TObject *Sender)
{
if(ImgTopologia->Visible == False)
ImgTopologia->Visible = True;
else
ImgTopologia->Visible = False;
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
void __fastcall TEnvironment::TabControl1Change(TObject *Sender)
{
if(TabControl1->TabIndex == 0)
{NetPanel->Visible = True;
AuthorPanel->Visible = False;}

if(TabControl1->TabIndex == 1)
```

```
AuthorPanel->Visible = True;

}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
void __fastcall TEnvironment::Sair1Click(TObject *Sender)
{
exit(0);
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
void __fastcall TEnvironment::Executarede1Click(TObject *Sender)
{
executarRedeNeural();
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
void __fastcall TEnvironment::Reposiciona1Click(TObject *Sender)
{
randomplace();
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
void __fastcall TEnvironment::Testaposio1Click(TObject *Sender)
{
movecar();
}
//---------------------------------------------------------------------------
```

C.2) Código do arquivo Ambiente_Neural_Codigo.h

```
//---------------------------------------------------------------------------

#ifndef Ambiente_Neural_CodigoH
#define Ambiente_Neural_CodigoH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ExtCtrls.hpp>
#include <Menus.hpp>
#include <ComCtrls.hpp>
#include <jpeg.hpp>
//---------------------------------------------------------------------------
class TEnvironment : public TForm
{
__published:            // IDE-managed Components
        TEdit *Edit1;
        TEdit *Edit2;
        TGroupBox *Grid;
        TEdit *Edit3;
        TButton *Button1;
        TShape *Car;
        TMainMenu *MainMenu1;
        TMenuItem *Principal1;
        TMenuItem *Carregarpesos1;
        TMenuItem *Sair1;
        TMenuItem *Sobre1;
        TMenuItem *SobreoNeuroSada1;
        TStatusBar *StatusBar1;
        TEdit *t1;
        TLabel *Label1;
        TButton *ExecutaRede;
        TEdit *Buffer;
        TEdit *t2;
        TLabel *Label2;
        TShape *Saida;
        TLabel *Label3;
        TLabel *Label4;
        TLabel *Label5;
        TButton *Button2;
        TCheckBox *DetectaColisao;
        TCheckBox *AreaGol;
        TPanel *About;
        TTabControl *TabControl1;
        TButton *Button3;
        TPanel *NetPanel;
        TImage *ImgTopologia;
        TLabel *Label10;
        TLabel *Label11;
        TLabel *Label9;
        TLabel *Label8;
        TLabel *Label12;
        TLabel *Label7;
        TButton *Button4;
        TLabel *Label6;
        TPanel *AuthorPanel;
        TLabel *Label13;
        TLabel *Label14;
        TImage *Image1;
        TLabel *Label15;
        TMemo *Memo1;
        TLabel *Label16;
        TMenuItem *Executarede1;
        TMenuItem *Reposiciona1;
        TMenuItem *Testaposio1;
        TEdit *Edit4;
```

```
        TLabel *Label17;
        TEdit *Edit5;
        TLabel *Label18;
        TMenuItem *N1;
        TMenuItem *N2;
        TMenuItem *N3;
        TLabel *Label19;
        TLabel *Label20;
        void __fastcall DisplayHint(TObject *Sender);
        void __fastcall Button1Click(TObject *Sender);
        void __fastcall Carregarpesos1Click(TObject *Sender);
        void __fastcall FormCreate(TObject *Sender);
        void __fastcall ExecutaRedeClick(TObject *Sender);
        void __fastcall Button2Click(TObject *Sender);
        void __fastcall SobreoNeuroSada1Click(TObject *Sender);
        void __fastcall Button3Click(TObject *Sender);
        void __fastcall Button4Click(TObject *Sender);
        void __fastcall TabControl1Change(TObject *Sender);
        void __fastcall Sair1Click(TObject *Sender);
        void __fastcall Executarede1Click(TObject *Sender);
        void __fastcall Reposiciona1Click(TObject *Sender);
        void __fastcall Testaposio1Click(TObject *Sender);
private:     // User declarations
public:                 // User declarations
        __fastcall TEnvironment(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TEnvironment *Environment;
//---------------------------------------------------------------------------
#endif
```

Eduardo Braga Dutra Rocha, 2006.
edwardrock@gmail.com
eduardo@eduardorocha.com.br