Universidade Estadual de Campinas
Faculdade de Engenharia Agrícola

APLICAÇÃO DE REDES NEURAIS ARTIFICIAIS MULTICAMADAS ESTÁTICAS NO PROCESSO DE SELEÇÃO DE FRUTOS

JEAN PAULO SILVA RAMOS

Orientador: Prof. Dr. INÁCIO MARIA Dal FABBRO

Tese de Doutorado apresentada como parte dos requisitos exigidos para a obtenção do título de Doutor em Engenharia Agrícola.
Área de concentração: Máquinas Agrícolas

CAMPINAS, SP Brasil
2001

Dedicatória

A Deus que me deu a luz, meus pais que sempre estiveram ao meu lado e a Marqueniz pelo apoio e incansável paciência.

Agradecimentos

Ao grande amigo e orientador Inácio Maria Dal Fabbro
Ao amigo Afonso Peche pelo orientação inicial.
Aos meus gerentes Marcos Pinedo, Paulo Henrique Jammal e André Magno Pimentel pela compreensão e apoio nas liberações eventuais para término dos créditos.
Ao companheiro Eduardo Longano pelos inúmeros incentivos para a conclusão da tese.
A Aninha e Marta pelo apoio e paciência nestes longos anos.
Ao corpo docente da Faculdade de Engenharia Agrícola que proveu o conhecimento necessário para a conclusão desta obra.

SUMÁRIO

Página

RESUMO

Neste trabalho, um novo enfoque de classificação de frutos é apresentado. A classificação de frutos depende do reconhecimento de padrões naturais ou artificiais de acordo com categorias pré-definidas. Os atributos de um fruto que está sendo classificado, deve ser comparado com um padrão de referência. Após a comparação a decisão da categoria adequada é tomada. A maior parte da classificação de frutos é baseada na habilidade humana. O objetivo geral deste trabalho é desenvolver um modelo artificial de classificação de frutos que considere simultaneamente várias propriedades físicas durante o processo de classificação do fruto.

Aqui, são utilizadas três redes neurais artificiais para a classificação e análise de diferentes atributos de frutos. A primeira rede neural artificial utiliza uma única propriedade física para a classificação de fruto, propriedade física Peso. A segunda rede neural artificial utiliza vetores com informações e classes de duas propriedades do fruto, propriedades físicas Peso e Diâmetro. A terceira rede neural artificial utiliza vetores com informações e classes de três propriedades físicas: Peso, Diâmetro e Firmeza. Todas as redes neurais utilizam o modelo Perceptron de múltiplas camadas e algoritmo de retro propagação do erro (“backpropagation”). Qualquer vetor de atributos de fruto, pertencente ao espaço amostral de treinamento, apresentado ao modelo é classificado segundo a classe desejada.

Palavras Chaves: Redes neurais artificiais, Perceptron de múltiplas camadas, algoritmo de retro-propagação, classificação de frutos.

ABSTRACT

In this work, a new approach of classification of fruits is presented. The classification of fruits depends on the recognition of natural or artificial standards according to pre-defined categories. The attributes of a fruit that is being classified, must be compared with a reference standard. After the comparison, the decision of the adequate category is taken. Most of the classification of fruits is based on the human being decision. The general objective of this work is to develop an artificial model of classification of fruits that simultaneously considers more than ones attributes of physical properties during the process of classification of the fruit.

Three artificial neural networks were used to the classification and analysis of different attributes of fruits. The first artificial neural network uses only one attribute for the classification of fruit, physical property Weight. The second artificial neural network uses vectors with information and categories of two attributes, physical properties Weight and Diameter. The third artificial neural network uses vectors with information and categories of three attributes, physical properties Weight, Diameter and Firmness. All the neural networks use the Perceptron model of multiple layers and error algorithm called backpropagation. Any vector of fruit attributes, contained in the training sample data, presented to the model is classified in the desired category.

Key Words: Artificial neural nets, Multilayer Perceptron, backpropagation algorithm, fruits sorting, classifiers, pattern recognition.

Capítulo 1

1 Introdução

Neste capítulo são apresentados os principais problemas relacionados ao processo de decisão na agricultura e a abrangência deste trabalho. Muitas são as pesquisas envolvidas na descoberta de ferramentas que auxiliem a decisão humana. Neste capítulo será abordada a característica humana de inferir sobre fatos e o processo de classificação de frutos. A somatória das contribuições individuais de cada pesquisa na área de classificação de frutos conduzirá a um avanço geral do processo de automatização do processamento de frutos.

1.1 O Problema de Classificação

O aparecimento de novas tecnologias na área de computação provocou um aumento significativo nas pesquisas de novos algoritmos computacionais que tentam simular o comportamento humano em tarefas cotidianas, em destaque a estas pesquisas surgem as redes neurais artificiais.

Redes neurais têm seus princípios fundamentais assentados nos trabalhos de modelagem biológica de processos neurofisiológicos, cognitivos e comportamentais, freqüentemente identificada como uma sub especialidade da Inteligência Artificial, e outras vezes, como uma classe de modelos matemáticos para problemas de classificação e reconhecimento de padrões, e outras ainda, como uma parte da teoria conexionista dos processos mentais e finalmente como uma categoria de modelos em ciência de cognição (**KOVACS,** 1996).

As redes neurais de uma forma simplista se propõem a simular o pensamento humano através de modelos matemáticos. Muitos são os problemas reais que podem ser simulados e resolvidos através do uso dessa técnica. Uma das características básicas do pensamento humano é sua capacidade de fazer inferências baseadas em fatos apresentados. A classificação natural de frutos é um bom exemplo desta habilidade, através de atributos simples tais como Peso, Tamanho (raio ou diâmetro), Firmeza ("Firmness"), Cor, Aroma, permitem que uma dona de casa, em suas

compras no supermercado, decida pela compra de um fruto exposto na prateleira, indicando desta forma se um dado fruto satisfaz nossos padrões de qualidade.

Para o processo de classificação artificial de frutos é necessário o desenvolvimento de modelos que permitam estabelecer o relacionamento entre a entrada de padrões de classificação (Entradas), análise/processamento desta informação e convergência para uma saída de classificação definida (Saída). A rede neural deve aprender a reconhecer padrões de entrada, que contém valores de propriedades físicas e definir a classe adequada de saída da rede, ou seja, dado a entrada de um determinado vetor de padrão de qualidade, escolher em que categoria/classe de saída este produto será melhor enquadrado. A rede neural deverá considerar a interatividade de todos os componentes vetoriais de entrada, sendo que cada componente individualmente representa uma propriedade física, no processo de classificação.

1.2 Objetivos

O objetivo geral deste trabalho é desenvolver um modelo artificial de classificação de frutos que considere simultaneamente vários atributos de propriedades físicas durante o processo de classificação do fruto. Através do uso das redes neurais artificiais, espera-se conseguir modelos capazes de classificar frutos utilizando-se da entrada simultânea de vetores de padrões. Os vetores de padrões contém os valores das propriedades físicas Peso, Diâmetro e Firmeza ("Firmness"), podendo conter valores de qualquer outra propriedade física. As redes neurais artificiais deverão aprender a relação entre os padrões de entrada , que representam os valores das propriedades físicas, e as classes de classificação desejadas na saída.

Em trabalhos futuros, através da aquisição de sinais elétricos dos vários atributos do fruto e da conversão do sinal analógico em digital, será possível compor e apresentar vetores de padrões de entrada à rede neural e classificar estes vetores segundo as classes desejadas. Este trabalho utiliza de mapas de padrões pré-definidos, hipotéticos, capazes de representar valores reais. A aquisição dos sinais para a criação do mapa de padrões pode ser feita automaticamente utilizando-se de transdutores, tais como, transdutores de Peso, Diâmetro, Cor, Firmeza ("Firmness") e outros.

A figura 1.1 apresenta as fases necessárias para a implementação de um mecanismo real de classificação e separação de frutos.

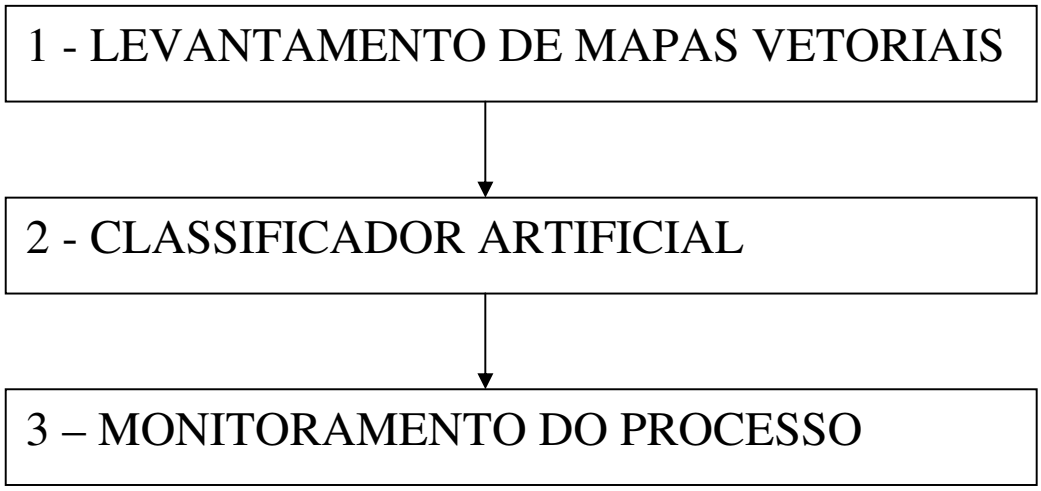

Figura 1.1 Diagrama do Processo de Classificação/Monitoramento

O levantamento de mapas vetoriais, que descrevem o comportamento de propriedades físicas e classes comerciais de frutos, é uma premissa para o classificador artificial, pois esses mapas são as entradas do classificador artificial. Uma vez que os processos 1 e 2 para um dado tipo de fruto sejam concluídos, é possível a implementação de componentes eletromecânicos para monitoramento em máquinas comerciais de classificação e separação em classes. A abrangência deste deste trabalho restringe-se ao desenvolvimento de modelos de classificação artificial de frutos que permita a implementação real de uma solução de mercado.

Especificamente, pretende-se:

? Inicialmente, modelar o problema dos cenários unidimensional, bidimensional, tridimensional;

? Determinar a topologia da rede neural para todos os cenários;

? Testar a escalabidade do cenário unidimensional através do aumento de neurônios na camada de entrada;

? Verificar a convergência das redes neurais para os cenários unidimensional, bidimensional e tridimensional.

Capítulo 2

2 Classificação de Frutos

2.1 Introdução

Muitas são as aplicações em engenharia agrícola que envolve decisões. Normalmente essas decisões são realizadas com a interferência da razão humana. Recentemente muitos trabalhos tem sido realizados no sentido de automatizar o processo decisório em algumas atividades agrícolas. Neste capítulo são apresentadas algumas das pesquisas realizadas em engenharia agrícola utilizando-se de redes neurais artificiais.

O processo classificatório representa uma das atividades de maior necessidade da decisão humana. Os tópicos ,apresentados a seguir, citam algumas das pesquisas realizadas nos últimos anos relacionadas à automatização do processo decisório para a classificação de frutos. Neste capítulo são introduzidos os conceito básicos para este trabalho.

2.2 Redes Neurais e Engenharia Agrícola

Nos últimos anos inúmeros trabalhos foram realizados na busca da automatização de processos agrícolas, bem como inúmeras ferramentas têm sido pesquisadas no sentido de descobrir como modelar problemas existentes no mundo real, automatizar processos tais como a operação de tratores, classificação e outras tarefas repetitivas, bastante adequadas a automatização. Várias ferramentas de apoio podem ser citadas, tais como micro-processadores, eletrônica embarcada, lógica "*fuzzy*" e outras. Entre essas ferramentas se destacam aquelas que se utilizam de redes neurais artificiais. As pesquisas nesta área vão desde o uso de redes neurais em irrigação até na automatização de tratores por comandos de voz. Alguns dos principais trabalhados são:

SATO e SALOKHE (1993) realizaram pesquisas na área de automação de tratores baseadas em comando de voz reconhecidos por redes neurais. A capacidade reduzida da rede neural utilizada fez com que os comandos ficassem limitados a somente as vogais a, e, i, o, u. Neste trabalho, inicialmente, a rede neural foi treinada para reconhecer entrada de voz e ruído, e,

durante o reconhecimento, comparar os sinais aprendidos com os sinais testados para distinguir entre o ruído do trator e a voz do operador.

Segundo SAYEED et al. (1995) a manutenção da qualidade de produtos alimentícios tais como petiscos/aperitivos, é um problema desafiador durante seu processamento. Nos dias atuais, é típico das companhias de alimentos determinar os padrões de qualidade através de uma equipe de profissionais treinados para tal. Este processo é subjetivo e susceptível a erros de classificação. O objetivo dos autores era investigar a qualidade de produtos desta categoria utilizando-se de redes neurais, através da captura de imagens de textura e forma. Uma rede neural tipo "*backpropagation*" foi treinada com um grande número de amostras de textura e forma como entrada e atributos sensoriais na saída. A rede demonstrou predizer os atributos sensoriais da qualidade deste tipo de alimento com razoável grau de precisão. As análises foram validadas através da comparação dos atributos sensoriais previstos obtidos com uso da rede neural com os valores obtidos de amostras não treinadas pela rede neural..

De acordo com ELIZONDO et al. (1994) é muito importante para os agricultores reconhecer o estágio de desenvolvimento de diferentes tipos de planta para decidir sobre a época mais adequada de colheita. Apesar do desenvolvimento de modelos de simulação computadorizados para prever o crescimento das plantas sobre condições ambientes, os autores desenvolveram pesquisas de um modelo de rede neural para predizer a florada e maturidade fisiológica de soja. As variáveis de entrada da rede neural foram: temperatura máxima e mínima do ar, fotoperíodo, dias decorridos do plantio e dias decorridos da florada.

DOWELL (1993) usou diferentes taxas de aprendizado, momentos e números de amostras de aprendizagem, para melhorar a precisão na medida da qualidade de grãos de amendoins. O autor encontrou em seu estudo que a taxa de aprendizado, por si só, não tinha efeito significativo na classificação de grãos amendoins danificados. Entretanto, diferentes combinações da taxa de aprendizado, números de elementos de processamento, ciclos de treinamento, e momento, produziram variação nos resultados. FREEMAN (1993) reconheceu que o valor da taxa de aprendizagem afeta o desempenho da rede. Ele sugeriu que um aumento neste valor, como procedimento de aprendizagem e redução de erros, acelera a convergência, entretanto, se o valor da taxa de aprendizagem for muito grande a rede poderá perder a estabilidade. A faixa, dentro da qual FREEMAN referenciou, para a variação da taxa de aprendizagem, foi 0.05 e 0.25.

Recentemente, pesquisadores do "College of Agricultural and Environmental Sciences" usaram redes neurais para resolver um problema não tão complexo, na realidade tão fácil para as

pessoas que se torna enfadonho, repetitivo mas muito adequado para um computador. Esta tarefa repetitiva se chama "*egg candling*" ou seja, o processo de rastreamento de defeitos e fraturas em ovos para efeito de classificação. A classificação pode ser definida em duas categorias diferentes: a categoria de ovos aptos para consumo humano e a categoria de ovos que devem ser descartados. Para cada ovo que você compra em um supermercado, foi necessário que um "par de olhos" tenha assegurado, no processamento dentro da granja, que este ovo esteja sem nenhum defeito ou quebrado. Para encontrar defeitos, cada ovo é colocado em frente de uma luz, a qual é usada como uma vela para descobrir fraturas, manchas de sujeira e pontos de sangue.

Dois profissionais chamados estranhamente de "homens-vela" podem inspecionar 240.000 ovos por dia. Para melhorar a eficiência deste processo foi ensinada uma rede neural a como encontrar fraturas em uma imagem computadorizada de um determinado ovo analisado e foi descoberto que esta rede após testes conseguiu aprender a classificar melhor que um classificador novato (McCLENDON et al, 1994). Baseado no sucesso inicial desta rede neural novas redes foram desenvolvidas para outras aplicações, tais como detecção de manchas de sujeiras e pontos de sangue. A rede neural de detecção de fraturas em ovos leva aproximadamente uma hora para ser treinada, usando um conjunto de 90 ovos bons e 90 ovos com fraturas. A máquina depois de preparada consegue examinar todos os 180 ovos em apenas um segundo, desta forma em uma hora a máquina consegue observar 650.000 ovos, aumentando significativamente o desempenho desta atividade e principalmente garantindo a uniformidade da classificação dos ovos.

Uma vez a rede treinada, esta necessita de uma fração de segundo para computar seu resultado. A máquina consegue ser treinada com 180 ovos e consegue aprender a distinguir centenas de milhares de ovos por dia. Assim como as pessoas humanas a máquina não consegue memorizar todos estes ovos, consegue por tentativa e erro aprender a julgar se um ovo está fraturado aprendendo como uma fratura se parece no geral. A rede neural consegue aprender a generalizar a partir destes 180 ovos analisados para qualquer ovo apresentado para a rede classificar.

Dos estudos na área de pecuária pode-se citar outro trabalho onde uma rede neural artificial do tipo retro-propagação ("*backpropagation"*) foi usada para detectar mastitis clínicas através de um arquivo de 460.474 registros de teste diários. Dois arquivos de dados foram criados para treinar a rede, contendo dados relativos de idade, estágio de lactação, contagem de células somáticas e componentes do leite. Resultados das análises das características operacionais indicaram que a rede pesquisada conseguia descriminar estados mastíticos com precisão global de

86%. O treinamento com um número significativo de casos de mastitis aumentou a habilidade da rede na descriminação dos casos positivos dos negativos. Variáveis adicionais tiveram pouco efeito na predição da precisão da rede (YANG et al. 1999).

YANG et al. (1999) usaram em seu trabalho um pacote de programas de simulação de redes neurais artificiais (NEURALWARE, 1993) o qual facilitou a manipulação dos parâmetros de configuração, tais como tipo de rede, taxa de aprendizado, momento e curva de aprendizado, e no caso da arquitetura da rede, parâmetros tais como o número de camadas escondidas e elementos de processamento em cada camada escondida criada. Várias arquiteturas de redes neurais, assim como parâmetros do algoritmo de retro-propagação, foram testados. (As redes descritas a seguir foram utilizadas devido à obtenção dos melhores resultados). Redes neurais de 3 camadas foram construídas utilizando 10 elementos de processamento na camada escondida. Em um primeiro caso, duas redes foram utilizadas com 6 elementos de processamento na camada de entrada correspondendo às variáveis tradicionais e treinadas com os arquivos de dados contendo os dados de incidência de mastitis clínica.

YANG et al. (1999) treinou as redes com a regra de aprendizado regra delta ("*normalized cumulative delta-rule*"). A função de ativação, responsável por transferir o sinal de entrada aplicado nos elementos de processamento, foi uma função tangente hiperbólica.

Uma tabela de análise foi utilizada para efeito comparativo e para avaliar o poder de uma rede na detecção de mastitis clínicas. Na tabela 2.1, os símbolos A,B,C e D mostram o número real para cada saída observada. A probabilidade condicional para a resposta verdadeira: positiva (VP), é estimada através da divisão do número correto de estados mastíticos previstos (A) pelo número de incidências reais (A+C) (SWETS e PICKETT, 1982). Esta probabilidade condicional, geralmente referida como sensitividade, é expressa como:

$$P(VP)=A/(A+C) \times 100 \qquad (2.2.1)$$

As outras três probabilidades condicionais denominadas respostas verdadeiro: negativo (VN) (geralmente referida como espeficidade), falso: positivo (FP), falso: negativo (FN), geralmente são obtidas da seguinte forma:

$$P(VN) = D / (B + D) \times 100 \qquad (2.2.2)$$

Tabela 2.1 – 2 x 2 para determinação da habilidade de uma rede

Predição de Mastitis	Observação de Mastitis		Total Predito
	Sim	Não	
Sim	A	B	A + B
Não	C	D	C+ D
Total Observado	A + C	B + D	A + B + C + D

$$P(FP) = B / (B + D) \times 100 \quad (2.2.3)$$

$$P(FN) = C / (A + C) \times 100 \quad (2.2.4)$$

Estas 4 probabilidades condicionais medem diferentes visões da capacidade de uma rede para identificar estados mastíticos.

Da tabela 2.1, duas probabilidades posterior de resposta do verdadeiro-positivo e verdadeiro-negativo podem ser calculadas:

$$P\ (PSTP) = A/(A+B) \times 100 \quad (2.2.5)$$

$$P\ (PSTN) = D/(C+D) \times 100 \quad (2.2.6)$$

$$P\ (TTCR) = (A+D)/(A+B+C+D) \times 100 \quad (2.2.7)$$

As equações 2.2.1, 2.2.2, 2.2.5 e 2.2.7 foram usadas para comparar os resultados das 4 redes e seus respectivos conjuntos de dados.

SALESHI et al. (1998) desenvolveu um trabalho para analisar o efeito de quatro diferentes parâmetros de aprendizagem assim como o método de apresentação dos dados com relação à performance de uma rede neural tipo retro-propagação na predição do rendimento do leite. Os parâmetros avaliados foram (1) taxa de aprendizagem das camadas escondida e saída, (2) momento, (3) número de iterações (época) e (4) apresentação de dados na forma binária.

O objetivo deste estudo foi investigar o efeito de diferentes valores de taxa de aprendizagem na camada escondida e camada de saída, momento e tamanho da época na predição da produtividade de leite, usando redes neurais tipo retro-propagação ("*backpropagation*").

2.3 Redes Neurais e Classificação de Frutos

Um exemplo dos problemas de classificação de frutos e vegetais pode ser citado através de um trabalho realizado na Nova Zelândia. Este trabalho cita a subjetividade do mercado de cogumelos, onde não existem especificações detalhadas do processo de classificação de cogumelos, sendo que estas classificações são feitas baseadas nas experiências dos produtores (especialistas). O objetivo desse estudo era o desenvolvimento de um conjunto quantitativo de descritores, os quais podem descrever os critérios de classificação por qualidade e disseminar os mesmos critérios para os consumidores. A comparação entre esses dois conjuntos descritores de classificação permitiu verificar a diferença entre a interpretação de qualidade entre produtores e consumidores (BOLLEN et al, 2000).

Geralmente os consumidores consideram somente os atributos visuais. Durante este estudo ocorreu um certo desacordo relacionado à classificação entre os consumidores e os produtores. Os consumidores indicaram que os produtores adotaram um padrão de qualidade que consideravelmente excedia a expectativa dos consumidores estudados.

Outro exemplo de classificação de frutos e aplicação de redes neurais está relacionado a maturidade de frutos. Se uma rede neural pode imitar exatamente quão maduro os compradores gostam de comprar seus tomates, pode-se então, associar este aspecto com conjuntos de modelos matemáticos que predizem quanto tempo será necessário até que os tomates cheguem a este ponto de maturidade, desta forma as quitandas poderão comprar tomates com a cor ideal de venda segundo a visão de seus compradores (THAI e SHEWFELT, 1991).

THAI e SHEWFELT (1991) afirmam que a inspeção das frutas e verduras é um procedimento importantíssimo na comercialização, armazenamento e processamento dos alimentos. A cor dentro do processo de inspeção, dependendo do produto, é um dos critérios mais importantes. A cor está relacionada à qualidade dos alimentos indicando maturação ou defeitos. A decisão na qualidade dos produtos varia temporalmente e entre inspetores. Neste trabalho foram mapeados os valores medidos das cores do tomate com os valores subjetivos da cor de preferência dos consumidores.

A qualidade dos produtos é definida como a composição daquelas características que diferenciam unidades individuais de um produto que afetam o grau de aceitação pelo comprador. Características de qualidade na compra são aquelas que podem ser percebidas pelos sensos de tato, aroma e/ou aparência visual, sem ingerir o produto, são muito importantes na diferenciação dos produtos na compra e preparação dos alimentos. Firmeza ("*Firmness*") e cor são as características primárias na decisão de compra de pêssegos frescos e tomates (THAI e SHEWFELT, 1991).

A classificação de laranjas prevê três tipos de defeitos causados por danos mecânicos:

1. Esmagamentos - são causados pelas ações mecânicas do granizo, vento, transporte e por outros meios;
2. Ferimentos e cortes - são causados, também, pelas ações mecânicas do granizo, vento, transporte e por outros meios;
3. Oleocelose rasa - é uma lesão mecânica com mais de um centímetro ou no máximo três manchas, cuja soma das superfícies ultrapasse a um centímetro de diâmetro.

Existem outros defeitos que devem resultar no descarte do fruto:

1. Podridão
2. Manchas de pragas
3. Manchas de doenças
4. Leprose
5. Frutos murchos
6. Frutos passados

A classificação visual é feita com base nestas características e defeitos, onde os frutos são separados manualmente de acordo com seus aspectos fisiológicos, ou seja, são descartados quando o selecionador julgar que o fruto não possui o padrão necessário para a comercialização. Esse sistema é sujeito à falhas e requer muita concentração do selecionador, além disso, é necessário grande número de selecionadores para agilizar o processo de classificação. De acordo com PELEG (1993) parte desta classificação está sendo feita atualmente por computador, da seguinte forma: os frutos são submetidos à inspeção visual um a um (por exemplo, através de esteira rolante), onde são fortemente iluminados e filmados. As imagens são comparadas em um computador com padrões do fruto existentes na memória do computador, desta forma então são classificados. O computador poderá descartar frutos fora do padrão de forma, tamanho, cor e manchas na superfície. Com o uso deste processo consegue-se uma classificação muito mais rápida e segura do que quando feita por processo manual, porém o processo tem suas limitações visto que é baseado apenas no aspecto visual.

Outro processo de classificação de frutos muito utilizado é a separação por peso. Processos mecânicos permitem uma rápida classificação de frutos de acordo com seu peso, da seguinte forma: em uma roda giratória são presos diversos suportes onde são colocados os frutos,

à medida que a roda gira, os frutos são retirados do suporte de acordo com seu peso. Os frutos mais pesados serão retirados primeiro, enquanto os mais leves são retirados por último (PELEG, 1993).

De acordo com PELEG (1993) há também a classificação por densidade. Nesse tipo de classificação, os frutos são imersos em um líquido com densidade conhecida e separados em vários níveis de acordo com suas densidades.

Na determinação da qualidade dos frutos, geralmente duas características são consideradas:

1 - "Ripeness" - maturidade
2 - Aparência Visual da fruta

Maturidade ("Ripeness") é o estado geral do estágio de maturação de uma fruta e é julgado pelos seguintes parâmetros: peso, comprimento, diâmetro, Firmeza ("*Firmness*") da fruta fresca, cor, Sólido Solúveis Totais, acidez, aroma, e parâmetros específicos da fruta (MICCOLIS e MIKAL, 1991), citado por OZER **et al.** (1995).

Muitos métodos de classificação tem sido baseados na extração da informação de um único sensor e na correlação de algumas características de qualidade, tais como Firmeza ("*Firmness*") (DELWICHE **et al**. 1987; ABBOTT, 1994; SHMULEVICH **et al**. 1993), forma e textura da superfície, (RIGNEY **et al**. 1992; MILLER E DELWICHE, 1991); cor (TIMM **et al**., 1993; THAI e SHEWFELT, 1991) citados por OZER **et al.** (1995).

Várias técnicas de classificação tem sido usadas tais como o classificador de Bayes e redes neurais artificiais usando um único sensor ou uma fonte única de informação. Classificação com classificador de Bayes e redes neurais baseadas em processamento artificial de imagens tem sido recentemente explorada OZER **et al.** (1995).

Infelizmente a realidade brasileira não acompanha a realidade de outros países e a seleção de frutos é feita de forma manual, onde a pessoa humana é usada como classificador, sendo vulnerável a critérios humanos, os quais podem variar temporalmente e de classificador para classificador.

Capítulo 3

3 Redes Neurais Artificiais

3.1 Introdução

Neste capítulo serão apresentados os conceitos teóricos dos modelos neurais naturais e artificiais, e apresentada uma comparação entre o funcionamento do cérebro humano, neurônio biológico e neurônio artificial. São apresentadas também o histórico das pesquisas de redes neurais artificiais, as características principais dos modelos existentes de redes, assim como o processamento de uma rede neural básica, o Perceptron.

3.2 Redes Neurais Biológicas

O neurônio biológico é basicamente o dispositivo computacional elementar do sistema nervoso que possui muitas entradas e uma saída. As entradas ocorrem através das conexões sinápticas, que conectam a árvore dendrital aos axônios de outras células nervosas. Os sinais que chegam por estes axônios são pulsos elétricos conhecidos como impulsos nervosos ou potenciais de ação, constituindo a informação que o neurônio processará de alguma forma para produzir como saída um impulso nervoso no seu axônio (KOVACS, 1996).

O cérebro humano é considerado o mais fascinante processador, sendo composto por aproximadamente 10 bilhões de neurônios. Todas as funções e movimentos do organismo estão relacionados ao funcionamento destas pequenas células. Os neurônios estão conectados uns aos outros através de sinapses, e juntos formam uma grande rede, chamada rede neural. As sinapses transmitem estímulos através de diferentes concentrações de Na+ (Sódio) e K+ (Potássio), o resultado disto pode ser extendido por todo o corpo humano. Esta grande rede neural proporciona uma fabulosa capacidade de processamento e armazenamento de informação (KOVACS, 1996).

O sistema nervoso é formado por um conjunto extremamente complexo de neurônios. Nos neurônios a comunicação é realizada através de impulsos, quando um impulso é recebido, o neurônio o processa, e passado um limite de ação, dispara um segundo impulso que produz uma

substância neurotransmissora o qual flui do corpo celular para o axônio que por sua vez pode estar ou não conectado a um dendrito de outra célula. O neurônio que transmite o pulso pode controlar a freqüência de pulsos aumentando ou diminuindo a polaridade na membrana pós sináptica. Eles têm um papel essencial na determinação do funcionamento, comportamento e do raciocínio do ser humano. Ao contrário das redes neurais artificiais, redes neurais naturais não transmitem sinais negativos, sua ativação é medida pela freqüência com que emite pulsos, freqüência esta de pulsos contínuos e positivos. As redes naturais não são uniformes como as redes artificiais e apresentam uniformidade apenas em alguns pontos do organismo. Seus pulsos não são síncronos ou assíncronos, devido ao fato de não serem contínuos, o que as difere de redes neurais artificiais.

Baseada na célula nervosa básica chamada neurônio, como qualquer célula biológica, o neurônio é delimitado por uma fina membrana celular que além da sua função biológica normal, possui propriedades que permitem a transmissão de sinais elétricos.

O neurônio é dividido em três partes principais:

- Os dendritos, que tem por função, receber os estímulos transmitidos por outros neurônios;
- O corpo de neurônio, também chamado de soma, é responsável por coletar e combinar informações vindas de outros neurônios. É o centro dos processos metabólicos da célula nervosa;
- O axônio, responsável pela condução do impulso nervoso na saída do neurônio.

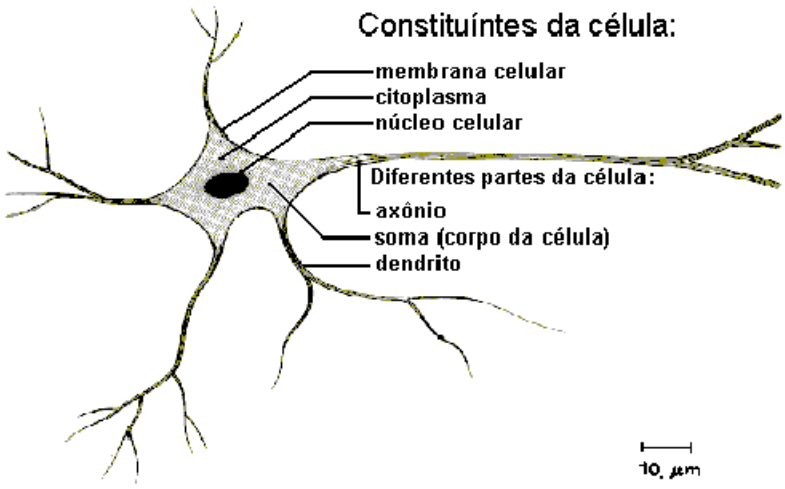

Figura 3.2.1 Neurônio Biológico (KOVACS, 1996)

3.3 Histórico

As redes neurais artificiais têm sua evolução baseada na contribuição individual de inúmeros pesquisadores destacando-se entre eles as publicações de: McCULLOCH e PITTS (1943), HEBB (1949) e ROSEMBLATT (1957). Estes trabalhos introduziram o primeiro modelo de redes neurais simulando "máquinas", o modelo básico de rede de Auto-organização, e o modelo Perceptron de aprendizado supervisionado, respectivamente.

Alguns pesquisadores da área de redes neurais entendem que ocorreram poucas pesquisas na área durante os anos 60 e 70. Um novo progresso significativo reiniciou-se com a publicação dos trabalhos de HOPFIELD (1982) relatando a utilização de redes simétricas para otimização e de RUMELHART et al. (1986) que introduziram o poderoso algoritmo de retro-propagação ("*Backpropagation*").

A seguir segue cronologicamente alguns dos principais contribuidores na pesquisa de redes neurais artificiais:

1943 - McCULLOCH e PITTS - Apresentam o primeiro modelo de redes neurais simulando "máquinas".

1949 - Donald HEBB - Apresenta o modelo básico de rede de Auto-organização.

1958 - John Von NEUMAIN – "*The computer and The brain*", modelos de neurônios usando relés e válvulas.

1956 - Albert UTLEY - "*Informons*", separador linear que ajusta seus parâmetros de entrada.

1958 - Frank ROSENBLATT (Cornell University) - "Perceptrons", generalização dos modelos de McCULLOCH/PITTS.

1959 - Bernard WIDROW e Marcian HOFF (Stanford University) - "Adaline and Madaline" - Perceptrons melhorados.

1969 - Marvin MINSKY e PAPERT - livro "Perceptrons".

As redes neurais apareceram para modelar como o cérebro funciona. Verdadeiramente tais estudos se originaram na psicologia. As teorias de Freud, William James e de outros psicólogos do século dezenove serviram de base para os posteriores trabalhos que originaram o estudo das redes neurais de hoje.

McCULLOCH e PITTS (1943) formularam o primeiro modelo de rede neural caracterizando os neurônios, mas não a habilidade de aprendizado. HEBB (1949) introduziu a idéia de aprendizado de HEBB, detalhado em "*Organization of behaviour*", declarando que mudanças nas forças sinápticas (conexões entre neurônios) são proporcionais às ativações dos neurônios. Esta foi a teoria básica para a criação de redes neurais com a habilidade de aprender.

A teoria de aprendizado de HEBB introduziu uma regra de atualização de força entre redes neurais de duas camadas, criando a habilidade destas redes de aprenderem. Frank ROSENBLATT incorporou esta idéia de aprendizado dentro de uma rede de duas camadas, chamando o resultado de Perceptron (ROSENBLATT, 1957). ROSENBLATT formulou uma regra de aprendizado baseada nos pesos ajustados em proporção ao erro entre neurônios de saída e saídas esperadas. Seu teorema de convergência de Perceptron converge para um grupo de pesos desejados. ROSENBLATT também formulou um Perceptron de três camadas e seu aprendizado. Entretanto, ROSEMBLATT foi incapaz de encontrar um método de treinamento de pesos entre os neurônios de entrada e a camada escondida de neurônios.

Muitos eram os problemas que não podiam ser resolvidos com rede de duas camadas. A falta de um procedimento matemático mais exato que permitia o aprendizado em uma rede multicamadas foi a barreira mais forte no desenvolvimento de soluções com uso de redes neurais. Em 1969 MINSKY e PAPERT mencionaram esse problema em Perceptrons (MINSKY, 1969). MINSKY apontou corretamente as limitações dos Perceptrons para vários problemas e reconheceu a possibilidade das redes de múltiplas camadas vencerem limitações. Porém ele não encontrou nenhuma forma de permitir o aprendizado dentro de tais redes e concluiu que as redes tipo múltiplas camadas poderiam ter poucos avanços em pesquisas futuras.

Alguns trabalhos interessantes continuaram a ser realizados dentro dos limites de redes de duas camadas. KOHONEN usou uma rede de duas camadas para construir o conteúdo de uma memória endereçável KOHONEN (1984), na qual nenhum índice separado deve ser dado para recuperar um item particular; o item por si só já é seu índice. Isto é, ao invés de fazer uma referência à memória a [i], onde **a** é a matriz e **i** é o índice, para obter o conteúdo de **c**, deve-se referir a localidade da memória com a [c]. KOHONEN referiu-se a este modelo como memória associativa. Memória associativa é baseada em um algoritmo de aprendizado não supervisionado, onde os pesos são ajustados puramente na base dos padrões apresentados, sem se preocupar com as saídas desejadas.

KOHONEN desenvolveu outro modelo baseado em aprendizado não supervisionado conhecido como "*learning vector quantizer*" (LVQ). O LVQ usa o aprendizado competitivo que permite neurônios de entrada ativar um e somente um neurônio de saída. (KOHONEN, 1984).

Desde meados de 1960 Stephen GROSSBERG tem desenvolvido modelos matemáticos de funções cerebrais (GROSSBERG, 1976). Apesar de serem estudos biológicos e psicológicos, conseguiu-se chegar a vários modelos de redes neurais, os quais são caracterizados pela habilidade de treinamento em tempo real, capacidade de auto-organização e habilidade de formar representações de fenômenos complexos. Também, o treinamento em tempo real permitiu a muitos modelos de GROSSBERG manusear dados que se modificam com o tempo.

Apesar do grande trabalho desenvolvido com sistemas, o grande passo no avanço das redes neurais, desde o Perceptron de duas camadas ROSENBLATT, foi a descoberta do algoritmo de "Retro-Propagação", permitindo o treinamento de redes de múltiplas camadas. O obstáculo original que impedia o progresso das redes de múltiplas camadas devido à incapacidade de treinamento foi removido.

RUMELHART et al. exploraram a retro-propagação em 1986 em seus trabalhos, simulando processos cognitivos (RUMELHART et al., 1986). Seus trabalhos foram bem difundidos pelo mundo cientifico através do processamento distribuído paralelo de McCLELLAND e RUMELHART, (RUMELHART, 1986).

Desde então a retro-propagação tem sido explorada em várias áreas não relacionadas a estudos e simulações de processos cognitivos, sendo uma ferramenta poderosa e prática na solução de problemas que seriam difíceis utilizando a técnica computacional convencional. Estes problemas abrangem desde o processamento de imagens, reconhecimento de voz e caracteres, à previsão e otimização.

Outros modelos tem sido desenvolvidos desde então, oferecendo vantagens em algumas aplicações sobre os modelos de retro-propagação canônicos. Bart KOSKO introduziu as memórias associativas bidirecionais, uma extensão das tentativas de KOHONEN nas memórias de conteúdos endereçáveis que permitem que dois diferentes tipos de padrões possam ser correlacionados, opostamente ao auto-associador mais simples, onde todos os padrões são do mesmo tipo (KOSKO, 1992).

Muitos destes modelos começaram a ser implementados em Integração à Larga Escala (VLSI). Muitos benefícios através do uso destes modelos nos problemas do mundo real não podem ser percebidos sem o uso de computação paralela.

3.4 Mapeamento do cérebro e da Máquina

O dispositivo computacional mais poderoso conhecido pelo homem é o cérebro humano. Uma criança de três anos pode executar inúmeras tarefas com extrema facilidade que foge em

muito as potencialidades dos computadores mais sofisticados, consegue reconhecer dúzias de rostos e centenas de objetos dos mais diferentes ângulos, em condições diferentes de luz, manipular um ambiente complexo; compreendendo e usando um vocabulário complexo incluindo gestos. Enormes esforços em pesquisas, com pouco sucesso tem sido gasto na tentativa de reproduzir em computadores versões limitadas de algumas destas potencialidades que o cérebro realiza. Em contrapartida os computadores conseguem realizar inúmeras atividades com extrema rapidez que levaria muito tempo para a maioria dos humanos, tais como complexos cálculos aritméticos com baixa probabilidade de erro.

Existe um grande antagonismo entre a mente humana e os computadores, visto que a capacidade de rápido processamento aritmético é muito difícil para os humanos e tão fácil para as máquinas, em contra partida o simples reconhecimento facial é extremamente simples para humanos e tão complexo para as máquinas.

Se duas cabeças pensam melhor do que uma, um cientista gostaria muito de ter uma mente extra para trabalhar. Esta mente extra o possibilitaria continuar o trabalho quando fosse descansar.

De todos os dispositivos dentro da caixa de ferramentas disponíveis para um pesquisador até agora, o computador é o que mais se assemelha a uma mente podendo processar milhões de dados instantaneamente. Porém a mente humana é capaz de fazer julgamentos, o que ainda não é tão simples para os computadores, tudo que os computadores podem fazer é o que são programados pelo homem.

Classificação de padrões é um exemplo de tarefa que os computadores tradicionais não conseguem fazer a menos que bem programados. Ao contrário dos humanos, a maioria dos computadores não pode lidar com situações que não tenham sido explicitamente programados para executar. Quando algo não se enquadra no programa, o computador simplesmente “não processa”.

Redes neurais são técnicas que fazem mais do que apenas processar milhares de números. Estas ferramentas computacionais permitem o uso da experiência, quase que como humanos, permitindo trabalhar com novas situações.

Até agora, cientistas encontraram pelo menos duas formas onde as redes neurais artificiais e a mente humana são similares: :

· Ambas trabalham surpreendentemente bem em tarefas diversas que são problemas para técnicas computacionais tradicionais.

· Ninguém sabe precisamente como ambas funcionam.

As redes neurais artificiais aparecem como modelos matemáticos simplificados do sistema nervoso central, resultante de três componentes básicos derivados do estudo do cérebro: neurônios, forças sinápticas e mecanismos de aprendizado. Os dois primeiros componentes das redes neurais artificiais são a base construtora que determina a estrutura, tamanho e capacidade da rede.

O terceiro componente, os mecanismos de aprendizado, determinam como a rede se comportará. Neurônios e forças sinápticas em redes neurais são análogas, porém não são as mesmas estruturas encontradas no cérebro. Os trabalhos reais da parte biológica são também muito complexos para serem simulados.

Os modelos neurais procuram aproximar o processamento dos computadores ao cérebro. As redes neurais artificiais possuem um grau de interconexão similar a estrutura do cérebro e um moderno computador convencional, a informação é transferida em tempos específicos dentro de um relacionamento com um sinal para sincronização.

Na Tabela 3.4.1 é apresentada de forma comparativa as relações existentes entre o cérebro humano e o computador:

Tabela 3.4.1 - Comparação entre cérebro humano e o computador

Parâmetro	Cérebro	Computador
Material	Orgânico	Metal e silício
Velocidade	Milisegundos	Nanosegundos
Tipo de Processamento	Paralelo	Seqüencial
Armazenamento	Adaptativo	Estático
Controle de Processos	Distribuído	Centralizado
Número de elementos processados	10 E11 à 10 E14	10E5 à 10E6
Ligações entre elementos processados	10.000	<10

O mesmo paralelo pode ser traçado comparando o computador com as redes neurais. Para tanto, a comparação não se dará com um computador específico encontrado no mercado, mas sim com o paradigma predominante nos computadores atuais.

Tabela 3.4.2 - Quadro comparativo entre computadores e neurocomputadores.

Computadores	Neurocomputadores
Executa programas	Aprende
Executa operações lógicas	Executam operações não lógicas, transformações, comparações.
Depende do modelo ou do programador	Descobre as relações ou regras dos dados e exemplos
Testa uma hipótese por vez	Testa todas as possibilidades em paralelo

3.5 Introdução às Redes Neurais Artificiais

Uma *rede neural artificial* (RNA) é um sistema de processamento de informação que possui algumas características de desempenho em comum com as redes neurais biológicas. Os modelos neurais artificiais têm como principal fonte de inspiração as redes neurais biológicas (SILVA, 1998).

As redes neurais artificiais consistem em um método de solucionar problemas, construindo um sistema que tenha circuitos que simulem o cérebro humano, inclusive seu comportamento, ou seja, aprendendo, errando e fazendo descobertas. São mais que isso, são técnicas computacionais que apresentam um modelo inspirado na estrutura neural de organismos inteligentes e que adquirem conhecimento através da experiência. Uma grande rede neural artificial pode ter centenas ou milhares de unidades de processamento, enquanto que o cérebro de mamíferos pode ter muitos bilhões de neurônios.

BUFO (2000) define uma rede neural como uma técnica matemática, realizada dentro de um fluxograma seqüencial de cálculo projetado, para obter resultados a partir de entradas de dados, independendo da lei que rege estes resultados. As redes neurais artificiais oferecem recursos quando outros meios matemáticos podem ser impotentes, tais como:

- Permite ajuste simultâneo de mais de uma variável de entrada e saída.
- Permite um acesso rápido a resultados onde a percepção e capacidade de identificação humana e computação tradicional são impotentes.

- Em situações em que os dados reais coletados de entrada na rede neural não estão em controle estatístico, a rede neural pode gerar um critério de relacionamento entre entrada e saída, independente da qualidade destes dados. Isto mostra a versatilidade da rede neural. Os dados a serem usados em redes neurais à priori devem ser certificados de sua origem e qualidade.
- No caso de difícil construção de modelos matemáticos entre uma ou mais variáveis interagindo de forma não linear, dando origem a um ou mais resultados, a rede neural consegue estabelecer uma relação entre os dados de entrada e o resultado, possibilitando aproximações, o que vem facilitar as simulações de processo de produção.

Redes Neurais Artificiais (RNA) podem ser definidas como "modelos matemáticos que, inspirados em estruturas cerebrais, são capazes de processar informação" (OLIVEIRA et al. 1996).

Os modelos de redes neurais artificiais são concebidos como uma resposta aos problemas que envolvem raciocínio de bom senso e processamento de informações incompletas, imprecisas e/ou vagas. A maior parte destes problemas, geralmente, está relacionada aos sistemas que necessitam automatizar processos que envolvam classificação de padrões e tomadas de decisão.

Em muitas aplicações, é importante que a rede forneça suas próprias classificações dos dados de treinamento. Para isso, é necessário fazer duas suposições básicas sobre a rede: a primeira é que seja observado o conceito de classe, definido como padrões de entrada que compartilham aspectos comuns; a segunda é que a rede seja capaz de identificar aspectos comuns entre os padrões de entrada.

Uma rede neural típica é constituída de uma camada de entrada de dados, uma ou mais camadas intermediárias de processamento, também chamada camada oculta ou escondida e uma camada de saída que fornece os resultados, como ilustra a figura 3.5.1. Cada camada é constituída de elementos chamados de neurônios, onde ocorre a modificação dos dados segundo uma função de ativação com o campo de variação entre 0 e 1. Essa função de ativação ou transferência pode ser uma função sigmoidal, podendo até ser uma função linear, exemplificado na figura 3.5.2. A camada de entrada recebe as variáveis, podendo ser um neurônio para cada variável, sendo que cada variável assume diversos valores. Podem possuir várias camadas intermediárias, mas na maioria das aplicações uma camada é suficiente para representar o relacionamento entre dados de entrada e saída. A camada de saída tem tantos neurônios quanto forem os diferentes dados

objetivos. Todos os neurônios de uma camada estão interligados com todos neurônios da camada subseqüente. Cada uma dessas interligações possui um peso $\mathbf{w}_{i,j}$ que multiplica a saída do neurônio anterior para gerar a entrada do neurônio subseqüente, bem como cada neurônio pode receber um valor constante chamado *bias* **B**, cuja finalidade é acelerar o processo de cálculo da minimização do erro entre o valor fornecido pela rede neural e o valor real (BUFO, 2000).

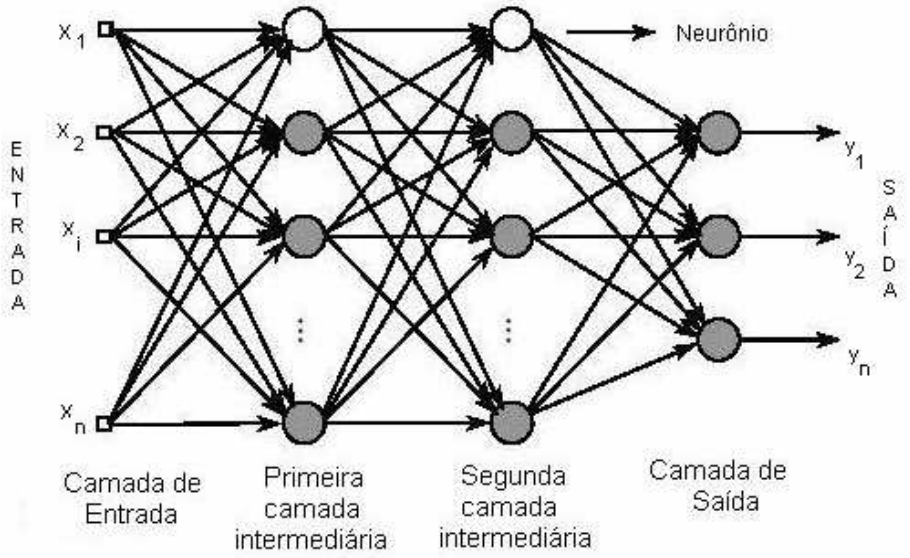

Figura 3.5.1 Rede Neural de Múltiplas Camadas

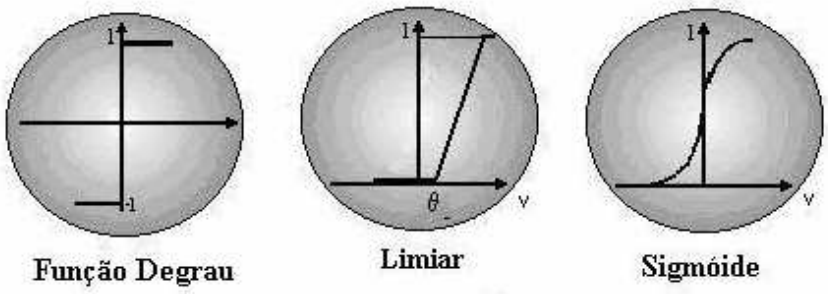

Figura 3.5.2 Funções de Ativação

3.5.1 Características Principais

Segundo SILVA (1998), as redes neurais artificiais têm sido desenvolvidas como generalizações de modelos matemáticos da cognição humana ou biologia neural, assumindo que:

- O processamento da informação ocorre em vários elementos chamados *neurônios*;

• Os sinais são propagados de um elemento a outro através de *conexões*;

• Cada conexão possui um *peso* associado, que em uma rede neural típica, pondera o sinal transmitido;

• Cada neurônio (ou unidade) aplica uma *função de ativação* (geralmente não-linear) à sua entrada de rede (soma ponderada dos sinais de entrada) para determinar sua saída.

Uma rede neural pode ser caracterizada por três aspectos principais: (1) o padrão de conexões entre as unidades (*arquitetura*), (2) o método de determinação dos pesos das conexões (*algoritmo de treinamento ou aprendizado*) e (3) sua *função de ativação*.

Os modelos neurais artificiais oferecem um paradigma atrativo, pois "aprendem" a resolver problemas através de exemplos. Segundo GHAZANFARI et al (1996), durante o processo de aprendizado, a rede, através do ajuste de suas conexões, tenta mapear seus padrões de entrada aos requeridos padrões de saída.

As redes neurais simplificam o processamento dos modelos biológicos. Um neurônio, ou nó, dentro de uma rede neural refere-se a um modelo que simula o comportamento de um grupo biológico de neurônios. O neurônio possui três funções: (1) aceitar as entradas das forças sinápticas e provavelmente somá-las; (2) aplicar uma função de ativação à entrada; e (3) passar a saída, via sinapses, para outras unidades. A função de ativação usada por um neurônio freqüentemente determina a capacidade da rede.

De forma geral, a operação de um neurônio da rede se resume em:

- Sinais são apresentados à entrada;
- Cada sinal é multiplicado por um peso que indica sua influência na saída da unidade;
- É feita a soma ponderada dos sinais que produz um nível de atividade;
- Se este nível excede um limite (“limiar”) a unidade produz uma saída;

Duas funções de ativação muito comuns são “degrau” [y = 1 se t > 0, senão y = 0 ou -1] e sigmóide [y = 1 / (1 + exp(-t))] ilustradas na figura 3.5.2. Funções “degrau” são usadas em redes de Hamming e Hopfield e outras redes com entradas e processamento digitais. A função sigmóide é muito poderosa devido a sua não linearidade. A função sigmóide domina as aplicações que usam entradas contínuas e na implementação de redes de retro-propagação.

Forças sinápticas são as conexões entre os neurônios. Associados às sinápses existem os pesos, um valor que é multiplicado pela força sináptica. O comportamento das redes é alterado através das alterações dos valores dos pesos. Os mecanismos de aprendizado são associados com estas modificações.

A operação das sinapses, por convenção, é similar à multiplicação de matrizes. Normalmente, redes neurais artificiais são construídas de camadas de neurônios, com cada neurônio em cada camada conectada por forças sinápticas a cada neurônio na camada anterior.

Por exemplo, considerando uma rede com quatro neurônios na primeira camada e três neurônios em sua segunda camada, se o vetor de saída da primeira camada é Y_1, então o vetor de entrada de cada neurônio na segunda camada será a soma dos produtos de cada elemento em Y_1 multiplicado por suas respectivas forças sinápticas. Se X_2 é o vetor de entrada da segunda camada, então $X_2 = WY_1$ onde W é uma matriz de 3 por 4 contendo os pesos associados com as forças sinápticas apropriadas. O vetor Y_2 é formado pela aplicação da função de ativação em cada elemento individual do vetor X_2.

Redes Neurais são apropriadas para problemas que requerem a interpretação de grande quantidade de dados. Além disto as redes neurais podem ser usadas para resolver problemas em que as entradas e correspondentes valores de saídas são conhecidos, porém o relacionamento entre estas entradas e saídas não é muito bem entendida.

De acordo com GHAZANFARI et al (1996), redes neurais são treinadas para fazer uma tarefa específica. Treinamento é o processo de alteração sistemática dos pesos de uma rede de forma a fazer os mapeamentos adequados entre um conjunto de entrada e respectivas saídas desejadas. Os pesos das conexões devem ser ajustados da camada de saída em direção às entradas ,retro-propagação dos erros. Quando todos os padrões de entrada são seqüencialmente introduzidos dentro da rede e os pesos são adequadamente ajustados, o ciclo de treinamento é completado. Ciclos de treinamento são repetidos até que a rede encontre um valor de erro pré-determinado ou até que um número especificado de ciclos de treinamento seja completado, este ciclo iterativo é interrompido segundo este critério de parada.

Segundo SALESHI et al. (1998), o treinamento redes neurais de múltiplas camadas do tipo retro-propagação ("*backpropagation"*) consiste em encontrar um conjunto de pesos que minimizam o erro global da rede, baseado na comparação do valor de entrada e saída desejada. Para o ajuste de pesos da rede, a regra delta de aprendizagem é freqüentemente usada. Com esta regra, o termos de correção aplicado para cada peso na rede neural é uma função da entrada com

seu elemento processante, termo de rede e taxa de aprendizagem. SALESHI, em seu trabalho treinou as redes com a regra-delta cumulativa e função de ativação tangente-hiperbólica para uso nos neurônios da camada escondida. O conjunto de dados de 62526 registros foi separado em dois grupos, um para treinamento e outro para teste.

GHAZANFARI et al. (1996), em seu trabalho de classificação de nozes trabalhou com 4 classes, consistida cada uma com 150 nozes aleatoriamente selecionadas, dividida em 2 conjuntos diferentes, um conjunto para teste e um segundo para treinamento.

Saturação refere-se ao estado onde os valores da soma dos neurônios (a soma das entradas multiplicada por seus respectivos pesos) se enquadram dentro da faixa onde a função de ativação se aproxima de seu valor máximo finito ou mínimo assintoticamente. Isto causa que a derivada da função de ativação seja 0, e conseqüentemente leve a 0 alterações nos pesos dos neurônios durante a retro-propagação (SALESHI et al. 1998).

Segundo GHAZANFARI et al. (1996), classificadores usando redes neurais podem chegar a melhores precisões na classificação através do aumento do número de neurônios na camada intermediária, entretanto esta decisão pode diminuir a capacidade de generalização da rede.

GHAZANFARI et al. (1996), define que embora várias camadas escondidas possam ser usadas em redes neurais artificiais, duas camadas escondidas são suficientes para gerar arbitrariamente complexas regiões de decisão.

3.6 Classificação das Redes Neurais

Existem várias maneiras de classificar os diferentes tipos de redes neurais. A forma mais comum é através do método de aprendizado, se este aprendizado é dinâmico ou estático, supervisionado ou não supervisionado. Os termos estáticos ou dinâmicos referem-se como a rede altera o seu comportamento como função do tempo. Redes estáticas têm seu comportamento congelado, isto é, os pesos da rede são determinados na implementação e não se alteram à medida que novos valores são introduzidos (SILVA, 1998). Dois exemplos deste tipo são as redes neurais de Hamming e Hopfield. A rede de Hamming consiste de duas sub redes, a sub-rede inferior, a qual aceita entradas de um padrão binário codificado e a sub-rede superior, a qual determina qual o padrão aprendido possui a menor distância de Hamming em relação ao padrão de entrada. Os

padrões da rede de Hamming são pré-definidos e não podem ser modificados a não ser por uma intervenção manual.

A rede de Hopfield binária se comporta de maneira similar à rede de Hamming. Dada uma entrada, a rede escolhe o padrão armazenado que possui a menor distância de Hamming em relação a esse padrão de entrada. A rede de Hopfield trabalha através da conexão dos nós de saída com todos os outros nós de entrada da rede e possui somente uma camada de nós que age tanto como nó de entrada como nó de processamento. A rede de Hopfield processa até que seja encontrado um padrão que não se altere mais.

Ambas as redes são simples de se projetar e implementar, devido a sua simplicidade, são as redes preferidas na implementação prática.

Redes dinâmicas mudam seu comportamento temporalmente e provavelmente são as redes mais usadas. Estas redes também podem ser classificadas como redes não supervisionadas ou supervisionadas, dependendo de como são condicionadas. Uma comparação pode ser feita nos métodos de aprendizado dinâmico com condicionamento em psicologia. Aprendizado não supervisionado é análogo ao condicionamento em psicologia. O condicionamento reforça o estímulo ao relacionamento aprendido. É fornecida uma entrada a quem está aprendendo e que reage com uma saída. Este aprendizado é repetido até que seja decidido que o elemento que está aprendendo tenha aprendido o relacionamento. Pensa-se então que o elemento consegue associar as entradas e decidir qual será a saída apropriada, eventualmente uma nova entrada dará a mesma saída (KOVACS, 1996).

Aprendizado supervisionado é a forma mais usada de aprendizado em redes neurais, normalmente feito pelo algoritmo de retro-propagação. Aprendizado supervisionado é análogo ao condicionamento operante em psicologia, no qual as respostas são reforçadas e não os estímulos. Para se ter o aprendizado supervisionado, são necessários amostras conhecidas. Saídas são calculadas às entradas fornecidas à rede. O supervisor compara a saída da rede à saída desejada e determina em que nível a rede deve ser “excitada” ou “inibida”. Então a rede modifica seu comportamento de acordo com a estratégia de supervisionamento (KOVACS, 1996).

O algoritmo de retro-propagação usa este conceito na forma de um gradiente regressivo. O algoritmo move a rede de uma superfície de erro desconhecido para um mínimo local ou global.

Qual deverá ser o gradiente e em qual direção, é determinado por cálculo com uma função de erro. Este erro é retro-propagado através da rede para determinar como os pesos devem ser ajustados para um melhor comportamento. O nome retro-propagação deve-se a este algoritmo.

Na prática, as redes não ficam aprendendo todo o tempo. Elas normalmente são desenvolvidas, treinadas e uma vez validadas, tem seu treinamento interrompido e o comportamento congelado com relação ao tempo.

Através da construção da rede e seleção dos seus pesos, quase qualquer função multidimensional ou associação entre entradas e saídas da rede pode ser aprendida pela mesma, (BULLOCK et al., 1992).

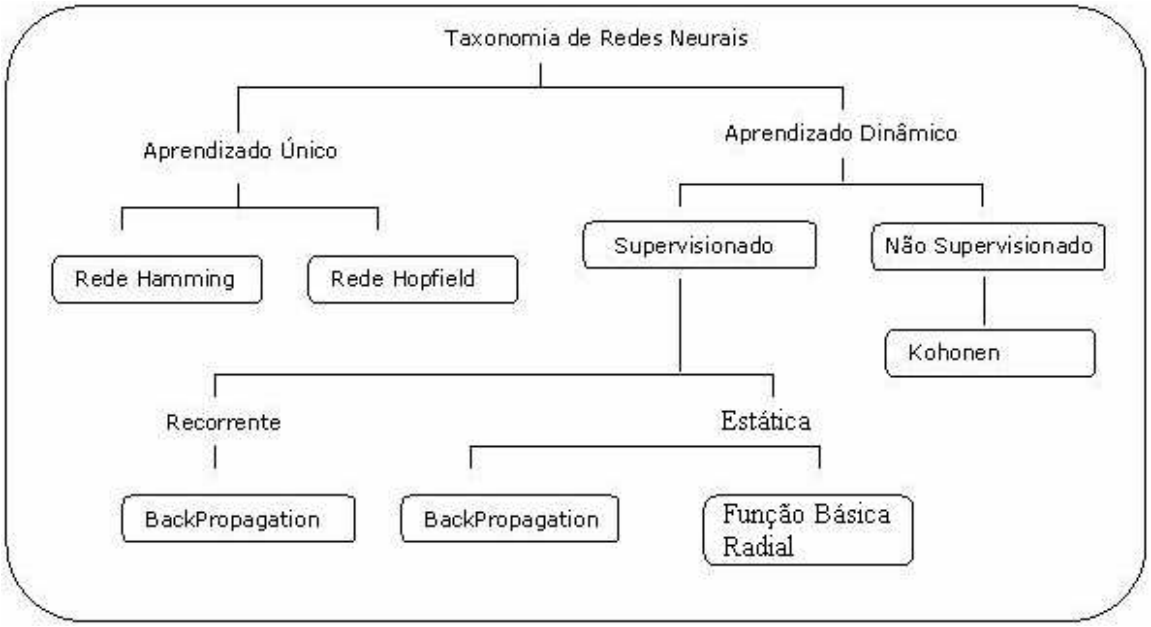

Figura 3.61 Taxonomia das Redes Neurais Artificiais

A propriedade mais importante das redes neurais é a habilidade de aprender com seu ambiente e com isso melhorar seu desempenho. O aprendizado da rede (treinamento) é feito através de um processo iterativo de ajustes de seus pesos. O aprendizado ocorre quando a rede neural atinge uma solução generalizada para uma classe de problemas.

Denomina-se algoritmo de aprendizado a um conjunto de regras bem definidas para a solução de um problema de aprendizado. Existem muitos tipos de algoritmos de aprendizado específicos para determinados modelos de redes neurais, estes algoritmos diferem entre si principalmente pelo modo como os pesos são modificados (SILVA, 1998).

A rede neural se baseia nos dados para extrair um modelo geral. Portanto, a fase de aprendizado deve ser rigorosa e verdadeira, a fim de se evitar modelos espúrios. Todo o

conhecimento de uma rede neural está armazenado nas sinapses, ou seja, nos pesos atribuídos às conexões entre os neurônios. De 50 a 90% do total de dados devem ser separados para o treinamento da rede neural, dados estes escolhidos aleatoriamente, a fim de que a rede "aprenda" as regras e não "decore" exemplos. O restante dos dados só é apresentado à rede neural na fase de testes a fim de que ela possa "deduzir" corretamente o inter-relacionamento entre os dados.

Outro fator importante é a maneira pela qual uma rede neural se relaciona com o ambiente (KOVACS, 1996). Nesse contexto existem os seguintes paradigmas de aprendizado:

1. Por independência de quem aprende

As Redes Neurais Artificiais aprendem por memorização, contato, exemplos, por analogia, por exploração e também por descoberta.

2. Por retroação do mundo

Diz respeito à ausência ou presença de realimentação explícita do mundo exterior, ou seja, que em certos intervalos de tempo um agente assinala acertos e erros.

2.1 - Aprendizado Supervisionado: utiliza um agente externo que indica à rede um comportamento bom ou ruim de acordo com o padrão de entrada.

2.2 - Aprendizado Não Supervisionado (Auto-organização): não utiliza um agente externo indicando a resposta desejada para os padrões de entrada, utiliza-se, entretanto, exemplos de coisas semelhantes para que a rede responda de maneira semelhante.

3. Por Finalidade do Aprendizado

3.1 - Auto-associador: é apresentado à rede uma coleção de exemplos para que ela memorize. Quando se apresenta um dos elementos da coleção de exemplos, mas de modo errôneo, a rede deve mostrar o exemplo original, funcionando desta forma como um filtro.

3.2 - Hetero-associador: é uma variação do Auto-associador, mas que memoriza um conjunto de pares. O sistema aprende a reproduzir o segundo elemento do par mesmo que o primeiro esteja pouco modificado, funcionando desta maneira como um reconhecedor de padrões.

É necessário também que exista um Detector de Regularidades, que nada mais é que um reconhecedor de padrões em que o sistema deve se auto-organizar e criar padrões possíveis.

Segundo SNNS (1995), pode-se denominar ciclo como sendo uma apresentação de todos os *N* pares (entrada e saída) do conjunto de treinamento no processo de aprendizado. A correção dos pesos num ciclo pode ser executado de dois modos:

1. Modo Padrão: A correção dos pesos acontece a cada apresentação à rede de um exemplo do conjunto de treinamento. Cada correção de pesos baseia-se somente no erro do exemplo apresentado naquela iteração. Assim, em cada ciclo ocorrem *N* correções.
2. Modo em Lote ("*Batch"*): Apenas uma correção é feita por ciclo. Todos os exemplos do conjunto de treinamento são apresentados à rede, seu erro médio é calculado e a partir deste erro fazem-se as correções dos pesos.

3.7 Perceptron

No final da década de 1950, Rosenblatt na Universidade de Cornell, deu prosseguimento as idéias de Mcculloch. Ele criou uma genuína rede de múltiplos neurônios do tipo descriminadores lineares e chamou esta rede de Perceptron. Um Perceptron é uma rede com os neurônios dispostos em várias camadas. Os neurônios que recebem diretamente as entradas da rede constituem o que se chama de camada de entrada (ROSENBLATT, 1957).

Os neurônios que recebem como entradas as saídas daqueles da camada de entrada constituem a segunda camada e assim sucessivamente até a camada final que é a camada de saída. As camadas internas que não são nem a de entrada e nem a de saída são geralmente referidas como camadas intermediárias ou escondidas.

A operação de uma unidade de processamento, proposta por McCULLOCH e PITTS em 1943, pode ser resumida da seguinte maneira:

- Sinais são apresentados à entrada;
- Cada sinal é multiplicado por um número, ou peso, que indica a sua influência na saída da unidade;
- É feita a soma ponderada dos sinais que produz um nível de atividade;
- Se este nível de atividade exceder um certo limite ("limiar") a unidade produz uma determinada resposta de saída (McCULLOCH e PITTS, 1943).

A figura 3.7.1 apresenta o neurônio proposto com as entradas X, função de ativação f(a) e saída y.

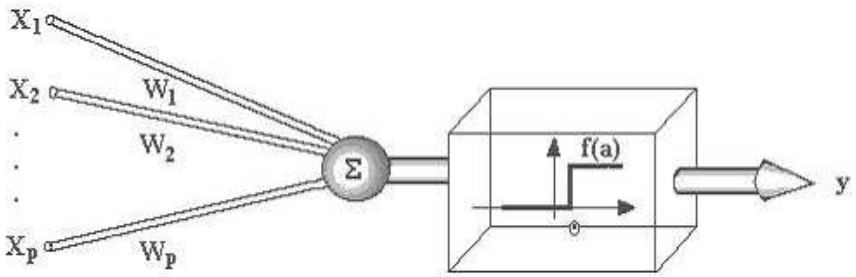

Figura 3.7.1 Neurônio de McCULLOCH

Com referência à figura 3.7.2, uma rede neural com K camadas, terá como entrada um vetor X_o de dimensão J_o de componentes $X_{o,j}$, j =1, 2,...J_o. Estas entradas são conectadas às entradas dos J_i neurônios da camada de entrada às saídas $X_{i,j,}$ j = 1, 2,... $J_{1.}$ Estes formam as componentes de um novo vetor X_1 de dimensão J_i que serão as entradas dos J_2 neurônios da camada seguinte e assim sucessivamente até a última camada que consistirá de J_k neurônios fornecendo como saída da rede um vetor Y= XK de dimensão J_k. Portanto $X_{k,j}$ denota a saída do *j*-ésimo neurônio na *k*-ésima camada, sendo que para k=0 representa a *j*-ésima entrada da rede.

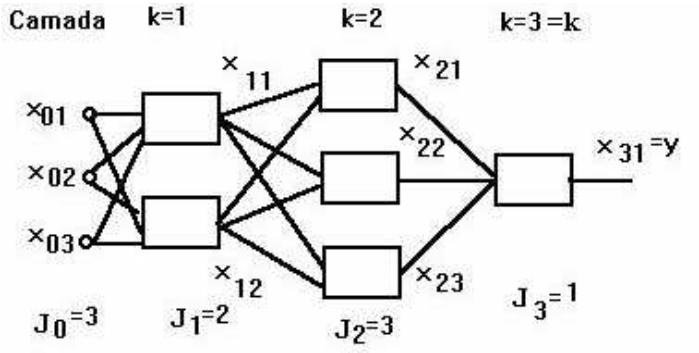

Figura 3.7.2 Representação esquemática de uma rede neural de três camadas, três entradas e uma saída (KOVACS, 1996) .

Muitos autores preferem referir-se ao vetor de entrada **X** como a camada de entrada, constituído por neurônios de ligação direta, isto é, que nada fazem além de conectar os estímulos à rede. Neste caso, a rede da figura 3.7.2 seria considerada como uma rede de quatro camadas cada uma de dimensão J_o, J_1, J_2 e J_3. Mantida a referência explicita às dimensões mencionadas,

passa a ser irrelevante qual é a convenção adotada quanto ao número de camadas na rede (KOVACS, 1996).

Sistemas nervosos biológicos possuem a propriedade essencial de serem capazes de "aprender" uma função. Poderia-se, portanto imaginar uma maneira de "ensinar" a rede artificial até que esta "aprendesse" a função desejada. Define-se o tipo de treinamento mais intuitivo possível: O "treinamento através de exemplos". Por este método, são apresentados exemplos de comportamento à rede, isto é para um estímulo $y^d_l = f(x^d_l)$ para todos os $l = 1, 2, ..., L$ exemplos.

Por simplicidade, pode-se considerar um Perceptron de n entradas formado por um único descriminador linear. A função deste Perceptron será de classificar corretamente vetores que lhe são aplicados à entrada, representativos de padrões, em duas categorias: y=1 correspondendo a aqueles que pertencem a categoria A e y=0 aos que pertencem a categoria "não A". ROSENBLATT efetivamente construiu um Perceptron deste tipo, batizado de Mark1, de 400 fotocélulas arranjadas em uma matriz de 20 x 20 Pixels, que constituíam as componentes do vetor de entrada e que servia para o reconhecimento de caracteres grafados nesta matriz. Assim por exemplo, se a função do Perceptron fosse reconhecer o símbolo A, a sua resposta deveria ser y=1 sempre que alguma variante deste padrão fosse apresentado na sua entrada e y=0 em caso contrário. Supondo que efetivamente estas duas classes de padrões {A} e {não A} fossem linearmente separáveis, resta a questão de como escolher os ganhos de ponderação das entradas w_i e o liminar Θ, parâmetros que definem unicamente um descriminador linear de n entrada $\{x_1, x_2, ..., x_n\}$ e uma saída y, conforme estabelecido na expressão:

$$y = H\left(\sum_{i=1}^{n} w_i x_i - \Theta \right) = H\left(w^t x - \Theta \right) \Rightarrow y \in [0;1] \qquad (3.7.1)$$

$$y = \mathrm{sgn}\left(\sum_{i=1}^{n} w_i x_i - \Theta \right) = \mathrm{sgn}\left(w^t x - \Theta \right) \Rightarrow y \in [-1;1] \qquad (3.7.2)$$

onde os componentes do vetor **w** , $\{ w_1, w_2, ..., w_n\}$ são os ganhos associados (pesos) às entradas x_i , Θ é o valor do liminar , ***H*** (ν) é a função degrau unitário e *sgn*(ν) o operador de sinal. A figura 3.7.3 apresenta um diagrama de blocos do descriminador linear.

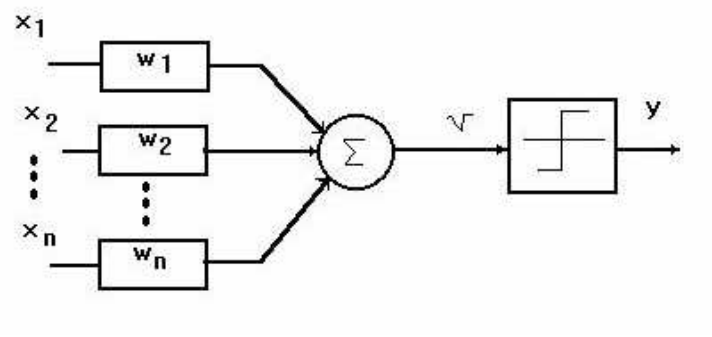

Figura 3.7.3 Diagrama de blocos do descriminador linear.

É possível formalizar melhor o método de treinamento através de exemplos, definindo inicialmente um conjunto Ψ com L exemplos de treinamento. Cada exemplo é um par do tipo (x_l^d , y_l^d), onde os x_l^d são entradas e as correspondentes saídas y_l^d . Estas saídas y_l^d serão +1 ou -1 conforme o vetor x_l^d , pertença a classe Φ_1 ou Φ_2 que se pretende separar. Portanto o conjunto de treinamento é:

$$\Psi = \left\{\left(x_l^d, y_l^d\right)\right\}_{l=1}^{L} \qquad (3.7.3)$$

Quando o descriminador linear exibe o comportamento expresso acima, diz se que está treinado. Resta determinar algum algoritmo de ajuste dos parâmetros que assegure esta convergência, se possível da maneiras mais rápida possível (KOVACS, 1986).

O descriminador linear é inicializado com parâmetros arbitrários { w_0 , Θ_0 } e os vetores x_l^d dos exemplos de treinamento são aplicados seqüencialmente à sua entrada. Utilizando-se de algum algoritmo para ajuste dos parâmetros {w, Θ} procura-se convergir para valores {w^*, Θ^*} tais que a saídas do descriminador coincidam com as saídas especificadas no conjunto de treinamento para todos os exemplos em Ψ. Ou seja:

$$y = \mathrm{sgn}(w^{*t} x_{i,l}^d - \Theta^*) = y_l^d \quad \text{para } l = 1, 2, \ldots L \quad (3.7.4)$$

3.7.1 Separabilidade Linear

Quando Redes Neurais Artificiais (RNA) de uma só camada são utilizadas, os padrões de treinamento apresentados à entrada são mapeados diretamente em um conjunto de padrões de saída da rede, ou seja, não é possível a formação de uma representação interna. Neste caso, a

codificação proveniente do mundo exterior deve ser suficiente para implementar esse mapeamento.

Tal restrição implica que padrões de entrada similares resultem em padrões de saída similares, o que leva o sistema à incapacidade de aprender importantes mapeamentos. Como resultado, padrões de entrada com estruturas similares, fornecidos do mundo externo, que levem a saídas diferentes não são possíveis de serem mapeados por redes sem representações internas, isto é, sem camadas intermediárias. Um exemplo clássico deste caso é a função ou-exclusivo (XOR).

MINSKY e PAPERT (1969) analisaram matematicamente o Perceptron e demonstraram que redes de uma camada não são capazes de solucionar problemas que não sejam linearmente separáveis. Como não acreditavam na possibilidade de se construir um método de treinamento para redes com mais de uma camada, eles concluíram que as redes neurais seriam sempre suscetíveis a essa limitação.

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

O Perceptron possui uma característica que restringe sua utilização para a maioria dos problemas reais, este modelo não consegue expressar decisões não-lineamente separáveis. O Perceptron é basicamente um dispositivo linear que retorna um dado valor, 1 por exemplo, se o produto ponto do vetor da entrada com o vetor peso associado mais a polarização excede um dado valor inicial, e um outro valor, -1 por exemplo, se o ponto inicial não for alcançado.

O produto ponto do vetor da entrada e do vetor peso associado mais a polarização $f(x_1, x_2,..,x_n)=w_1x_1+w_2x_2+ \ldots w_nx_n+w_b=$ Threshold, é representado graficamente em $x_1,x_2...,x_n$ de forma linear. Esta função separa este espaço em duas categorias. Todos os vetores da entrada que dão o valor de a $(f(x_1, x_2,..., x_n)=w_1x_1+w_2x_2+... +w_nx_n+wb)$ maior do que o ponto inicial é separado em um espaço, e aqueles que não atingem este valor são separados em outro. A figura 3.7.1 apresenta dois exemplos, um linearmente separável e outro não separável.

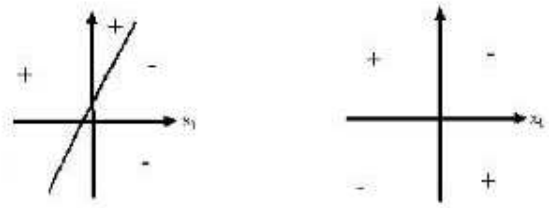

Figura 3.7.1 Problema linearmente separável (Esquerda) e não linearmente separável (Direita).

Considere-se o espaço de padrões da figura 3.7.2, com duas classes distintas **A** e **B**.

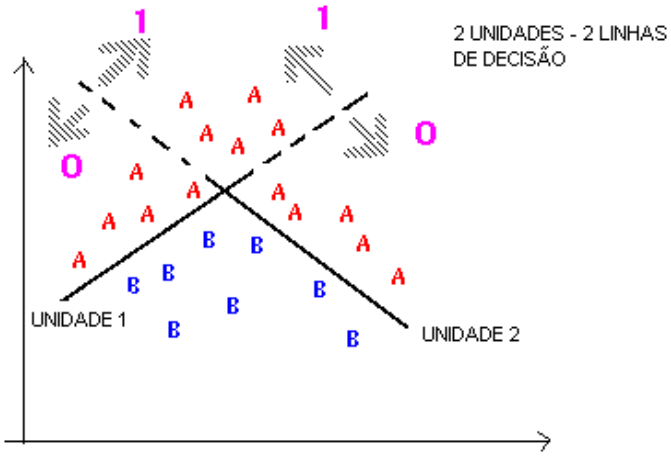

Figura 3.7.2: Classes não linearmente separáveis – 2 planos

As classes A e B não podem ser separadas por um único hiperplano. Em geral necessita-se de uma superfície de decisão formada arbitrariamente. Na figura 3.7.2, tem-se esta superfície aproximada por dois planos. Para resolver este problema é necessário construir uma rede de duas camadas. Cada plano é determinado por um dos pares de um nó escondido.

Para superfícies mais complexas são necessárias mais unidades escondidas, como pode ser observado na figura 3.7.3.

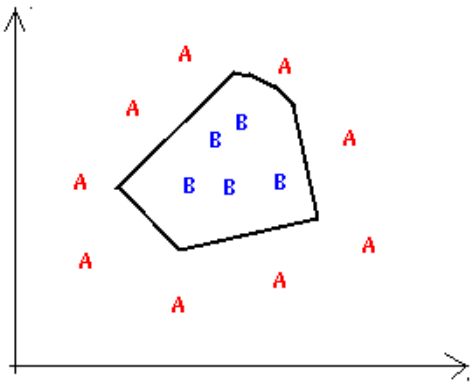

Figura 3.7.3 Superfícies de decisão mais complexas

Para resumir a força das ferramentas que tem sido desenvolvidas, é possível treinar uma rede de múltiplas camadas para categorizar um número arbitrário de classes e com uma superfície de decisão arbitrária. Tudo que é necessário é que exista um conjunto de entradas e saídas desejadas, que seja determinado um conjunto de unidades intermediárias que vão ser usadas e que o gradiente (descendente) do erro com o algoritmo de retro-propagação seja executado.

Uma das falhas clássicas do Perceptron em aprender um dos mais clássicos problemas é o caso XOR. A solução do caso XOR de um Perceptron apareceu com o desenvolvimento dos modelos de neurônios que aplicaram uma função sigmóide à somatória dos pesos ($w_1x_1+w_2x_2+ ... +w_nx_n+wb$) para fazer a ativação da não linearidade. Um exemplo de uma função sigmóide geralmente usada é a função logística dada por O(y) = 1 / (1+e^(-y)), onde y =$w_1x_1+w_2x_2+ ... + w_nx_n$ + wb. Quando estas "unidades sigmóides" são arranjadas camada por camada, com cada camada sendo chamada pela outra, agindo como vetores de entrada, tem-se a criação então da rede de múltiplas camadas.

As redes neurais de múltiplas camadas consistem normalmente de três ou quatro camadas. Existe sempre uma camada de entrada, uma camada de saída e geralmente uma camada intermediária, embora alguns problemas de classificação necessitem duas camadas intermediárias. Os neurônios da camada da entrada não são afetados pela função sigmóide, os valores brutos destes neurônios são alimentados na camada seguinte, a camada intermediária. Os neurônios da camada intermediária uma vez computados possibilitam que as ativações destes sejam alimentados na camada seguinte, até que todas as ativações alcancem eventualmente a camada da saída, onde cada neurônio da camada da saída está associado com uma categoria específica de classificação. Em uma rede de múltiplas camadas, cada neurônio em uma camada é

conectado por um peso a cada neurônio na camada anterior a esta. Uma polarização é associada também a cada uma destas somatórias de pesos. Desta forma para calcular o valor de cada neurônio na camada intermediária e de saída, verificando assim a ativação do neurônio, basta aplicar a f(somatória) à soma da somatória dos pesos e polarização. Este processo será explicado melhor no tópico de Perceptron de Múltiplas Camadas.

3.7.2 Caso XOR

De acordo com SILVA (1998), o caso XOR é um problema artificial bastante conhecido na literatura. É caracterizado por um alto grau de não linearidade e por não preservar a idéia de vizinhança, ou seja, mudando apenas um *bit* de cada vetor de entrada, a saída é invertida.

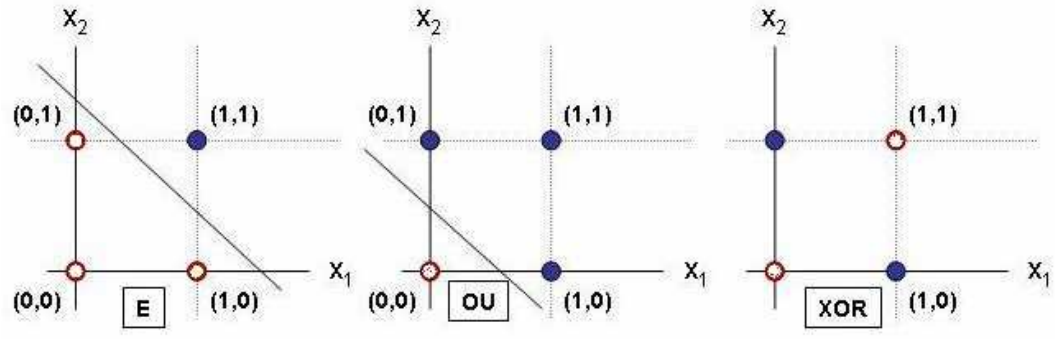

Figura 3.7.2.1 Funções booleana E, OU, XOR.

A figura 3.7.2.1 apresenta 3 funções booleanas **E**, **OU**, **XOR** de duas variáveis. É possível verificar que as funções **E** e **OU** são linearmente separáveis, existem retas que separam os vetores de entrada em regiões tal que a saída reflete corretamente o resultado das funções booleanas (KOVACS, 1996). A função **XOR**, o chamado OU exclusivo, não é linearmente separável, como pode ser observado na figura 3.7.2.1, Seriam necessárias duas retas descriminatórias para separar os vetores {1,1} e {0,0} pertencem à classe vermelha e os vetores {0,1} e {1,0} pertencem a classe azul. Ao contrário das portas lógicas E e OU, apenas uma única reta não consegue fazer a separação linear entre as duas classes no caso XOR.

Para as 16 funções booleanas de duas variáveis, somente duas (o XOR e o seu complemento) não são linearmente separáveis. Com a dimensão crescente do vetor de entrada x, as funções booleanas não separáveis linearmente passam a formar um conjunto denso, isto é, passam a ser a grande maioria.

A tabela 3.7.1.1 mostra a tabela verdade do Caso XOR, com pode ser observado existe uma simetria oposta entre os valores de entradas iguais e diferentes.

Tabela 3.7.1.1 Tabela Verdade XOR

	x_1	x_2	Saída
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	0

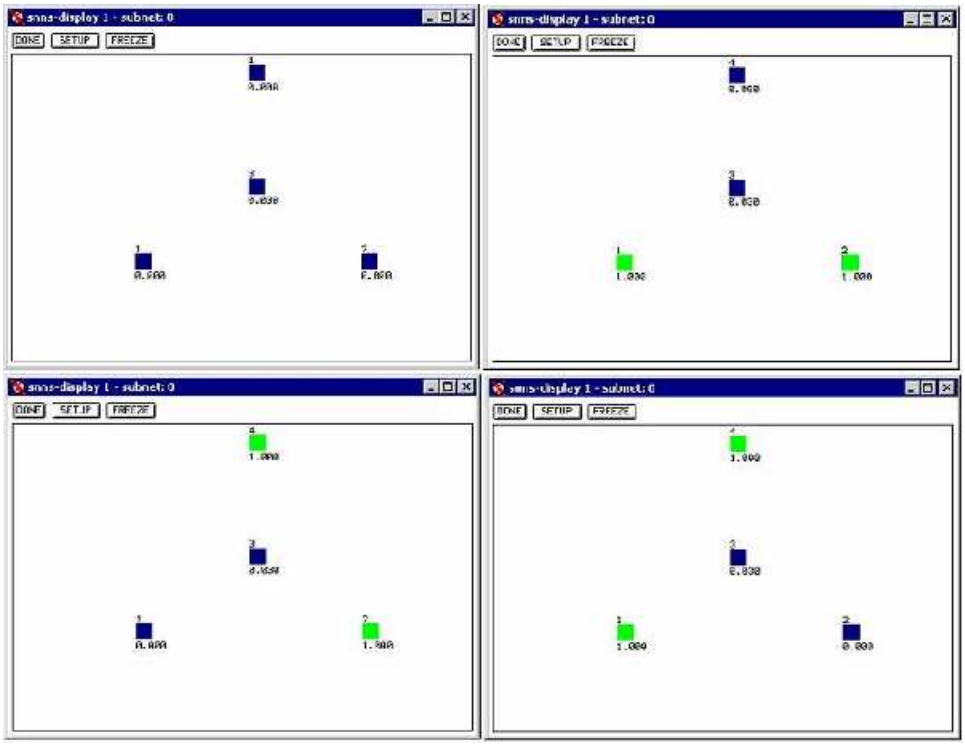

Figura 3.7.2.2 Rede Neural XOR – 3 Camadas

Os anexos IV e V trazem dois programas de implementação do caso XOR. Os programas foram desenvolvidos utilizando-se da linguagem Visual C++ 6.0 e Visual Basic 6.0, ambos da fabricante de Software Microsoft.

Foram utilizados com os mesmos pesos iniciais e número de iterações e não demonstraram nenhuma diferença significativa nas respostas. Ambas as redes fizeram o treinamento e classificaram os valores de saída correto para o caso XOR. Como já era esperado, a

convergência da rede neural XOR para o programa desenvolvido em Visual C++ teve tempo bem inferior à mesma implementação feita em Visual Basic.

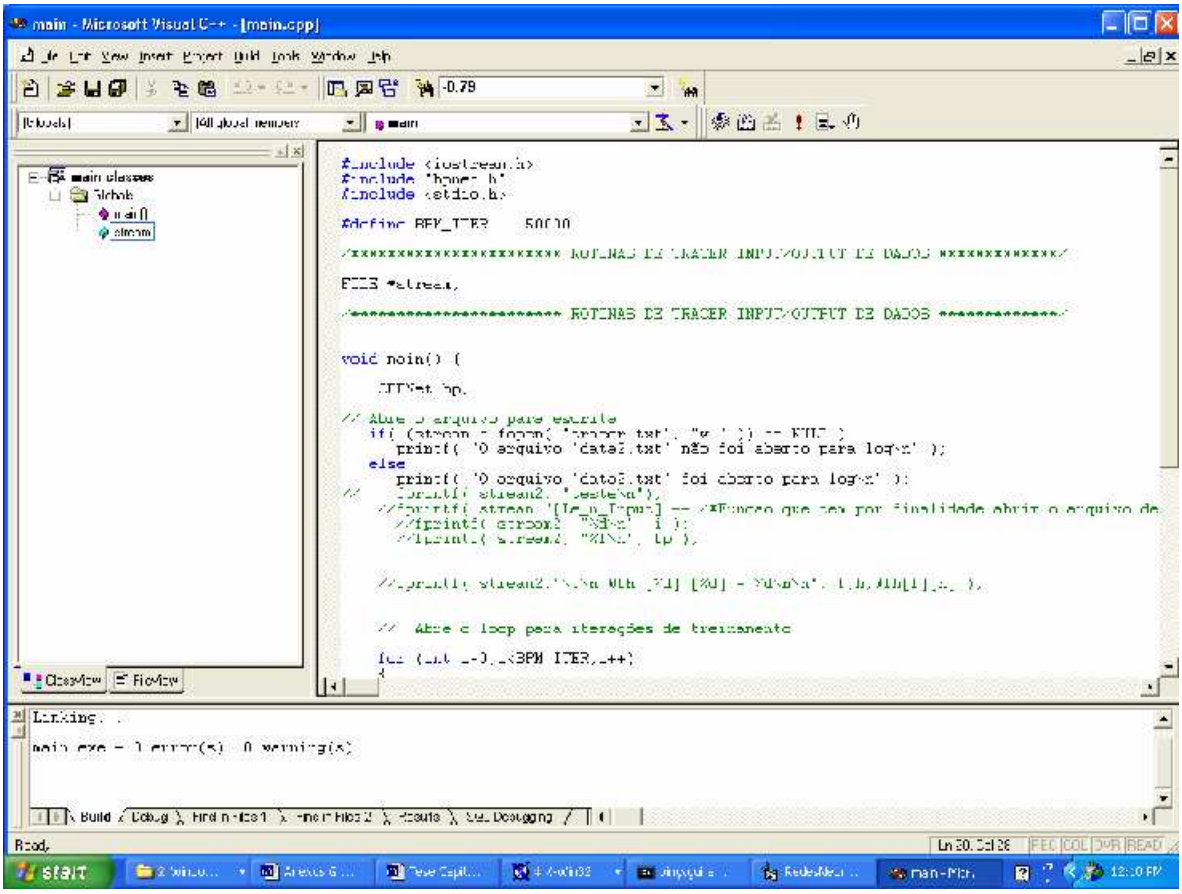

Figura 3.7.2.3 Implementação caso XOR em Visual C++

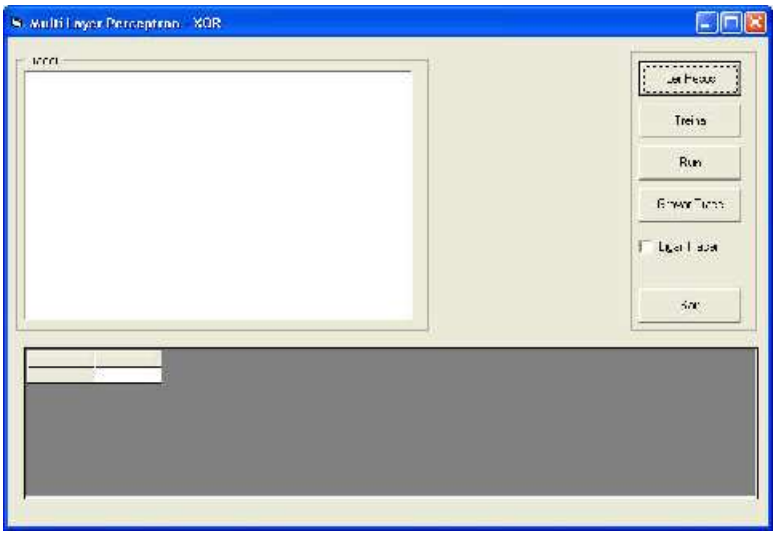

Figura 3.7.2.4 Implementação caso XOR em Visual Basic

3.8 Perceptron Múltiplas Camadas – MLP

3.8.1 Introdução

RUMELHART et al. (1986) através de suas pesquisas demonstraram que é possível treinar redes neurais com camadas intermediárias, resultando no modelo de redes neurais artificiais mais utilizado atualmente, as redes Perceptron de múltiplas camadas (MLP), treinadas com o algoritmo retro-propagação ("*backpropagation*"). Este tipo de rede pode possuir uma ou mais camadas intermediárias, também comumente chamada de camada escondida ou oculta, cada camada possui uma função específica. A camada de entrada é responsável pela recepção dos padrões de entrada, a camada de entrada se liga à camada intermediária através de conexões. As conexões guardam os pesos que serão futuramente multiplicados pelas entradas, garantindo o conhecimento da rede. A camada de saída recebe os valores da camada intermediária fornecendo a resposta da rede.

A Figura 3.8.1.1 apresenta uma arquitetura do tipo MLP com duas camadas intermediárias. A rede apresentada como exemplo possui todas as conexões, o que significa que um neurônio em qualquer camada da rede está conectado a todas as outras unidades (neurônios) na camada anterior. O fluxo de sinais através da rede é feito positivamente, da esquerda para a direita, camada a camada.

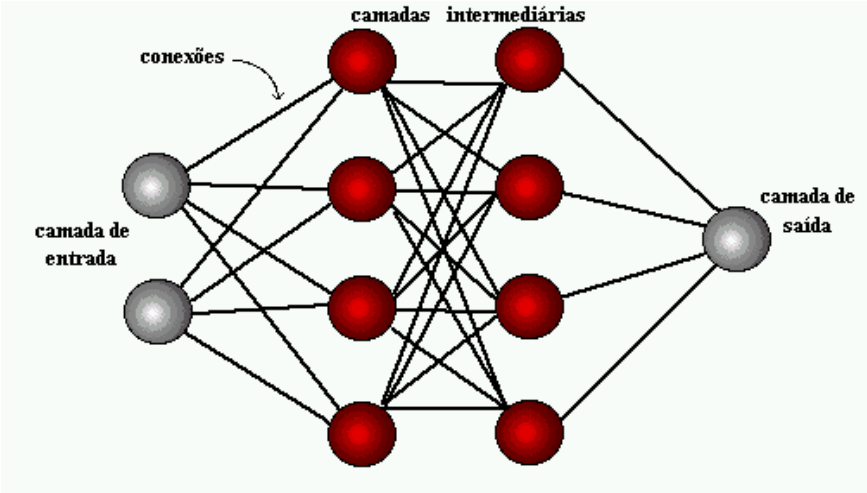

Figura 3.8.1.1. Rede Neural tipo MLP.

Segundo SILVA (1998), as arquiteturas do tipo Perceptron de múltiplas camadas (MLP) constituem os modelos neurais artificiais mais utilizados e conhecidos. Tipicamente, esta

arquitetura consiste de um conjunto de unidades sensoriais que formam uma camada de entrada, uma ou mais camadas intermediárias (ou escondidas) de unidades computacionais e uma camada de saída. Os sinais de entrada são propagados camada a camada pela rede em uma direção positiva, ou seja, da entrada para a saída. Esta arquitetura representa uma generalização do Perceptron apresentado anteriormente. As redes do tipo MLP tem sido utilizadas com sucesso para a solução de vários problemas envolvendo altos graus de não-linearidade. Seu treinamento é do tipo supervisionado e utiliza um algoritmo muito popular chamado retro-propagação do erro ("*error backpropagation"*). Este algoritmo é baseado numa regra de aprendizagem que "corrige" o erro durante o treinamento (HAYKIN, 1994).

A rede bem definida, com número correto de camadas, neurônios e pesos corretos , permite que a relação entre um conjunto de padrões de entrada e saídas seja processada por uma rede neural treinada e estabilizada.

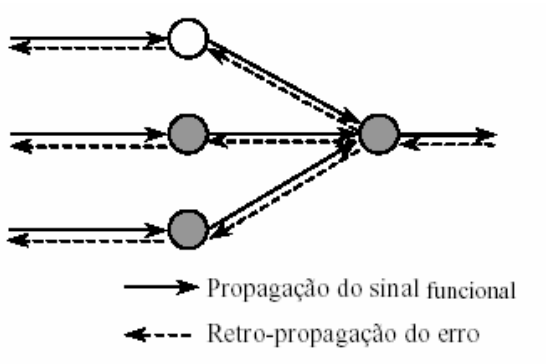

Figura 3.8.1.2. Representação das direções de propagação do sinal funcional e do erro.

A Figura 3.8.1.2. ilustra alguns neurônios e suas conexões, apresentando apenas uma parte da rede. Nesta rede, dois tipos de sinais podem ser identificados:

• **sinal funcional**: um sinal funcional é um sinal de entrada (estímulo) que chega na entrada e é propagado positivamente (neurônio a neurônio) através da rede, e aparece na saída como um sinal de saída.

• **sinal de erro:** os sinais de erro originam-se nas saídas e são retro-propagados (neurônio a neurônio) através da rede.

A camada de entrada geralmente é composta por neurônios sensoriais, ou seja, unidades que não modificam os sinais externos, apenas os distribuem para a primeira camada intermediária. As unidades de saída constituem a camada de saída da rede, e as demais unidades

constituem as camadas intermediárias. As camadas intermediárias são todas aquelas que não fazem parte da entrada e nem da saída.

Cada unidade intermediária ou de saída é responsável por duas tarefas:

• calcular o sinal na saída da unidade, que geralmente é expresso como uma função não-linear do sinal de entrada e pesos sinápticos associados e

• calcular uma estimativa instantânea do vetor gradiente, que é necessário para a retro-propagação do erro através da rede (SILVA, 1998).

3.8.2 Treinamento de uma rede Perceptron Múltiplas Camadas – MLP (Algoritmo)

Durante o treinamento com o algoritmo retro-propagação, a rede opera em uma seqüência de alguns passos. Primeiro, um padrão é apresentado à camada de entrada da rede. A atividade resultante flui através da rede, camada por camada, até que a resposta seja produzida pela camada de saída. No segundo passo, a saída obtida é comparada à saída desejada para esse padrão particular. Se a saída obtida comparada com a saída desejada não estiver correta, o erro é então calculado. O erro é propagado a partir da camada de saída até a camada de entrada, e os pesos das conexões das unidades das camadas intermediárias vão sendo modificados conforme o erro é retro-propagado (BUFO, 2000).

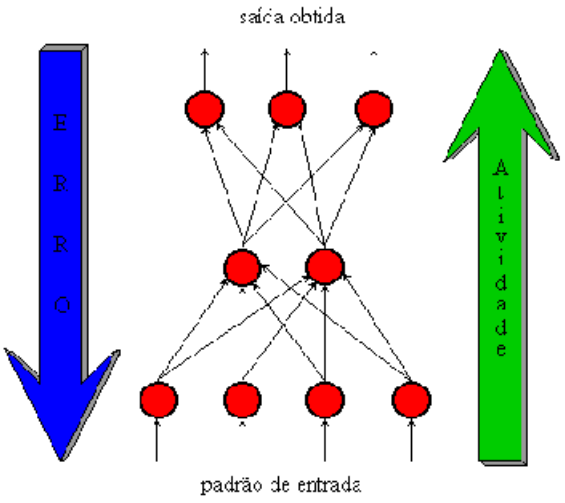

Figura 3.8.2.1. Atividades da rede múltiplas camadas.

SILVA (1998), define que o treinamento de uma rede neural de múltiplas camadas, usando o algoritmo de retro-propagação, é feito basicamente com os passos abaixo:

1. O padrão é apresentado à camada de entrada da rede.

2. O padrão de entrada é multiplicado por um conjunto de pesos e propagado até a última camada, chamada camada de saída.
3. O valor de saída é então comparado ao valor da saída desejada para o padrão em análise.
4. Caso o valor de saída obtido seja diferente do valor desejado, então o erro é calculado.
5. O erro é propagado à partir da camada de saída até a camada de entrada, os pesos das conexões das camadas intermediárias vão sendo modificados à medida que o erro é propagado.

BUFO (2000), afirma que antes de usar qualquer algoritmo, é indispensável a definição da topologia da rede neural e a consolidação/normalização do conjunto de dados de treinamento, ou seja, a variável $\mathbf{v}_i$ com os seus valores de entrada e da variável $\mathbf{d}_i$ com os valores que são os resultados desejados coletados.

A seguir é descrito o algoritmo de retro-propagação ("*backpropagation*"), segundo BUFO (2000):

1. Normalizar os valores da variável $\mathbf{v}_i$ e os valores da variável $\mathbf{d}_{i,}$ tal que estejam contidos entre 0 e 1. Este procedimento é imperativo para o bom desempenho da rede neural.
2. Inicializar todos os *bias* adotando pequenos valores aleatórios: **B**
3. Inicializar todos os pesos adotando pequenos valores aleatórios: **w**
4. Calcular as saídas da camada intermediária.
5. Calcular as saídas da camada de saída.
6. Calcular o erro local $\mathbf{e_o}$ da camada de saída e atualizar o *bias* **B**.
7. Atualizar as conexões **w** entre as camadas oculta e camada de saída.
8. Calcular o erro local $\mathbf{e}_i$ da camada oculta e atualizar os *bias* **B** dessas camadas.
9. Atualizar as conexões **w** entre as camadas oculta e camada de entrada.
10. Testar a convergência: Se Erro Global < Erro permitido, então pare. Senão volte ao passo 4.

Após o treinamento, a rede deve ser validada. A fase de validação compreende a aplicação deste algoritmo para um novo conjunto de dados de entrada e saída obtidos no processo em estudo e dentro do mesmo intervalo de variação dos dados utilizados no treinamento, usando

os pesos **w**'s e *bias* **B**'s encontrados na fase de treinamento, a fim de que a rede neural forneça valores de saída.

As redes que utilizam o algoritmo de retro-propagação trabalham com uma variação da regra delta, apropriada para redes de múltiplas camadas: a regra delta generalizada. A regra delta padrão essencialmente implementa um gradiente descendente no quadrado da soma do erro para funções de ativação lineares. As redes sem camadas intermediárias podem resolver problemas onde a superfície de erro tem a forma de um parabolóide com apenas um mínimo.

De acordo com SILVA (1998), basicamente, o processo de retro-propagação do erro é constituído de duas fases: uma fase de propagação do sinal funcional ("*feed-forward"*) e uma de retro-propagação do erro ("*backpropagation"*). Na fase positiva, os vetores de dados são aplicados às unidades de entrada, e seu efeito se propaga pela rede, camada a camada. Finalmente, um conjunto de saídas é produzido como resposta da rede. Durante a fase positiva, os pesos das conexões são mantidos fixos. Na retro-propagação do erro, por outro lado, os pesos são ajustados de acordo com uma regra de correção do erro. Especificamente, a resposta da rede em um instante de tempo é subtraída da saída desejada ("*target"*) para produzir um sinal de erro. Este sinal de erro é propagado da saída para a entrada, camada a camada, originando o nome "retro-propagação do erro". Os pesos são ajustados de forma que a distância entre a resposta da rede e a resposta desejada seja reduzida.

Muitas vezes, a superfície do erro pode ter relevos mais complicados onde suas derivadas podem ser mais difíceis de serem calculadas. Nos casos onde ocorrem relevos mais complexos devem ser utilizadas redes com camadas intermediárias. Mesmo com várias camadas alguns casos, onde os pesos iniciais não são adequados podem ocorrer problemas com mínimos locais, impossibilitando a convergência da rede.

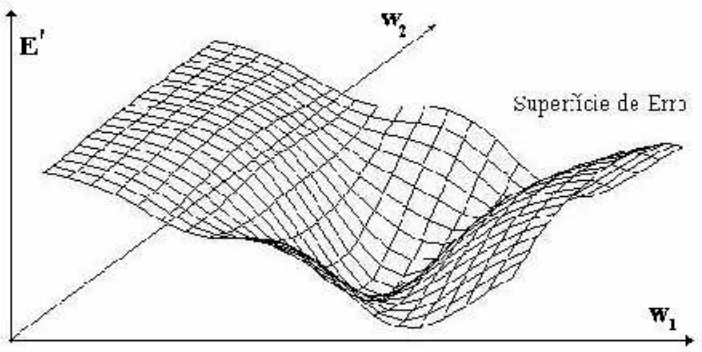

Figura 3.8.2.2. Exemplo de superfície de erro.

A regra delta generalizada funciona quando são utilizadas na rede unidades com uma função de ativação semilinear, que é uma função diferençável e não decrescente. Uma função de ativação amplamente utilizada, nestes casos, é a função sigmóide.

A taxa de aprendizado é uma constante de proporcionalidade no intervalo [0,1]. O procedimento de aprendizado requer apenas que a mudança no peso seja proporcional à reta (SILVA, 1998).

SILVA (1998) afirma que o verdadeiro gradiente descendente requer que sejam tomados passos infinitesimais. Assim quanto maior for essa constante, maior será a mudança nos pesos, aumentando a velocidade do aprendizado, o que pode levar a uma oscilação do modelo na superfície de erro. O ideal é utilizar a maior taxa de aprendizado possível que não leve a uma oscilação, resultando em um aprendizado mais rápido.

GHAZANFARI et al. (1996) afirma que a escolha de um tamanho adequado para uma aplicação específica é dependente do problema e geralmente consome um certo tempo e esforço. Em geral, grandes redes possuem grande habilidade em realizar uma tarefa mas perdem no poder de generalização. Pelo outro lado, redes pequenas podem nunca convergir para uma dada aplicação.

A velocidade de treinamento das redes do tipo MLP depende principalmente:

- da condição inicial dos pesos e limiares;
- da determinação adequada das funções de ativação;
- da arquitetura da rede;
- do conjunto de dados e
- do algoritmo de treinamento.

3.8.3 Mínimo Local

O treinamento das redes MLP com retro-propagação pode demandar muitos passos no conjunto de treinamento, resultando em um tempo de treinamento consideravelmente longo. Se for encontrado um mínimo local, o erro para o conjunto de treinamento pára de diminuir e estaciona em um valor maior que o aceitável como pode ser observado na figura 3.8.3.1. Uma maneira de aumentar a taxa de aprendizado sem levar à oscilação é modificar a regra delta generalizada para incluir o termo momentum, uma constante que determina o efeito das mudanças passadas dos pesos na direção atual do movimento no espaço de pesos.

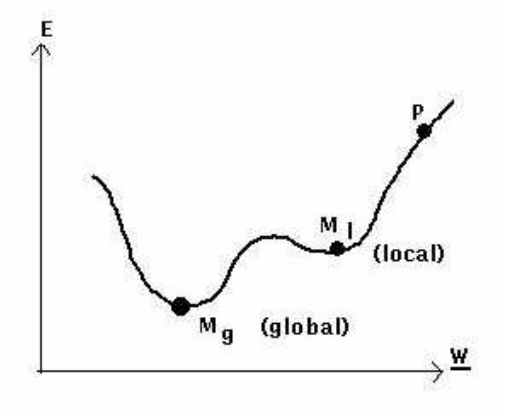

Figura 3.8.3.1 Mínimo local e global

Desta forma, o termo momentum leva em consideração o efeito de mudanças anteriores de pesos na direção do movimento atual no espaço de pesos. O termo momentum torna-se útil em espaços de erros que contenham longas gargantas, com curvas acentuadas ou vales com descidas suaves, como o apresentado na figura 3.8.2.2.

SILVA (1998), define que uma rede do tipo MLP possui três características distintas:

- função de ativação;
- número de camadas e unidades intermediárias e
- forma das conexões.

3.8.4 Função de Ativação

O modelo de cada unidade da rede pode incluir uma não linearidade na sua saída. É importante enfatizar que a não-linearidade deve ser suave, ou seja, diferençável, diferentemente da função sinal utilizada pelo Perceptron original.

A função de ativação representa o efeito que a entrada interna e o estado atual de ativação exercem na definição do próximo estado de ativação da unidade. Geralmente, esta função é monotonicamente não-descrescente e apresenta um tipo de não-linearidade associada ao efeito da saturação. A seguir são descritos alguns tipos de função de ativação empregados na literatura para as arquiteturas do tipo MLP (KOSKO, 1992; HAYKIN, 1994).

Função Linear

Este tipo de função de ativação é muito utilizado nas unidades que compõem a camada de saída das arquiteturas MLP.

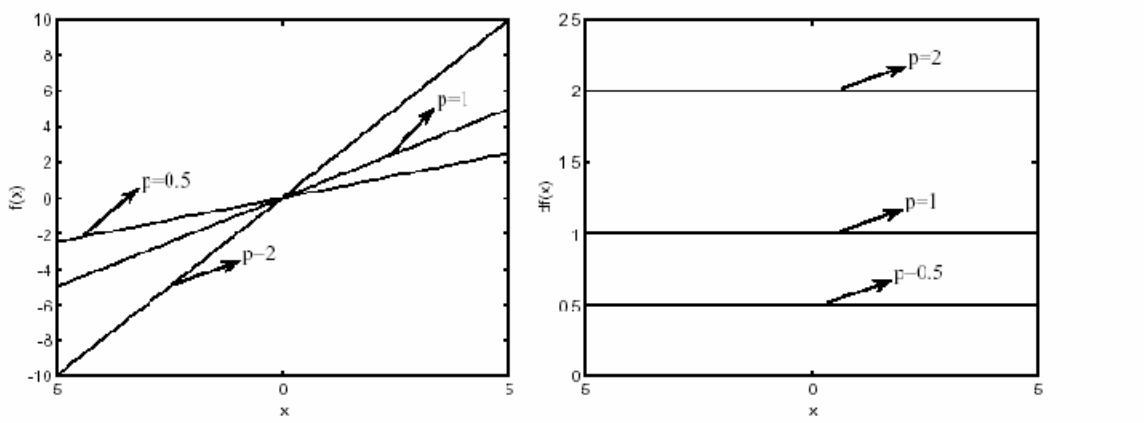

Figura 3.8.4.1 (a) Função linear. (b) Sua derivada em relação à entrada interna.

A saída linear com $p = 1$ simplesmente repete o sinal que entra no neurônio na sua saída. A Figura 3.8.4.1 apresenta saídas lineares e suas derivadas.

A expressão para esta função de ativação e sua derivada é:

$$f(x) = p.x, f'(x) = p \qquad (3.8.1)$$

Função Logística

A Figura 3.8.4.2 (a) mostra que a função logística possui intervalo de variação entre 0 e 1.

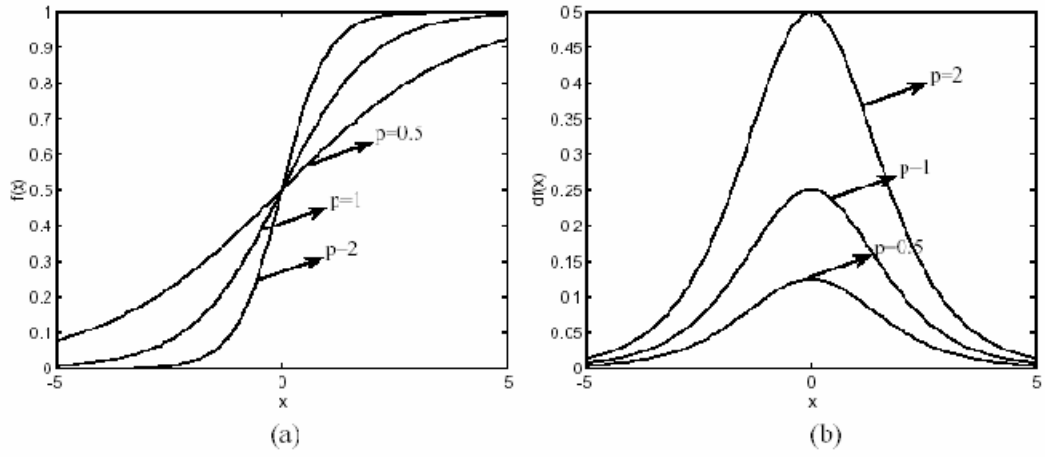

Figura 3.8.4.2 (a) Função logística. (b) Sua derivada em relação à entrada interna.

$$f(x) = \frac{e^{px}}{1+e^{px}} = \frac{1}{1+e^{-px}}, \qquad f'(x) = p\,f(x).\left(1 - f(x)\right) \tag{3.8.2}$$

A origem deste tipo de função está vinculada à preocupação em limitar o intervalo de variação da derivada da função, pela inclusão de um efeito de saturação. Sua derivada também é uma função contínua (ver Figura 3.8.4.2).

Função Tangente Hiperbólica

Pelo fato da função logística apresentar valores de ativação apenas no intervalo (0, 1), em muitos casos ela é substituída pela função tangente hiperbólica, que preserva a forma sigmoidal da função logística, mas assume valores positivos e negativos ($f(x) \in (-1, 1)$). A função tangente hiperbólica e sua derivada (ver Figura 3.8.4.3) são dadas pelas expressões:

$$f(x) = \frac{e^{px} - e^{-px}}{e^{px} + e^{-px}} = \tanh(px), \qquad f'(x) = p\left(1 - f(x)^2\right) \tag{3.8.4}$$

A função tangente hiperbólica pode ser transladada para o intervalo (0, 1), assim como a função logística pode ser transladada para o intervalo (-1, 1).

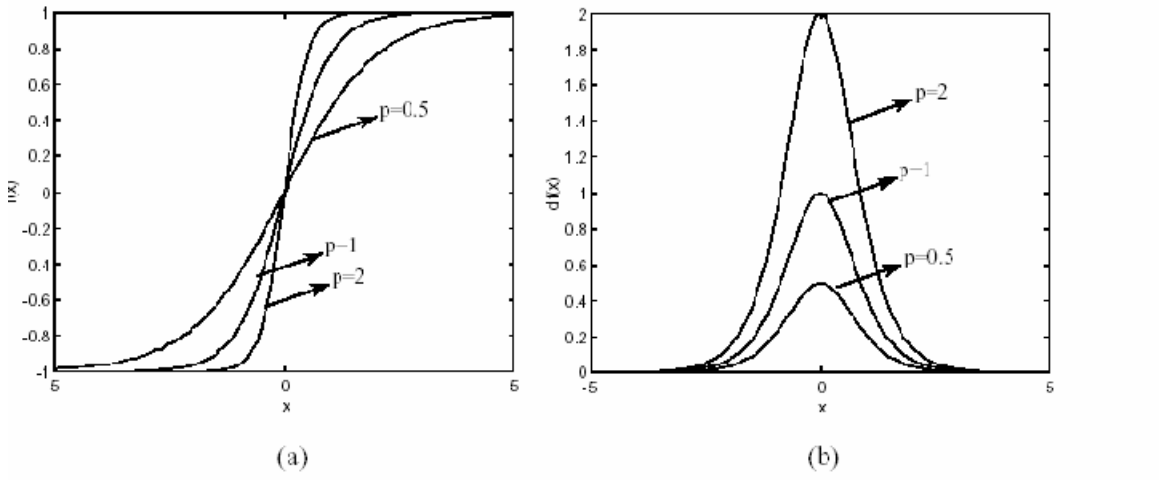

Figura 3.8.4.3 (a) Função tangente hiperbólica. (b) Sua derivada em relação à entrada interna.

Função Arco-Tangente

Esta função possui valores de ativação no intervalo ($-\pi/2$, $\pi/2$), e pode ser apresentada como uma alternativa à função tangente hiperbólica para a implementação computacional, pois

requer menos cálculos para sua elaboração. Comparações sucintas entre o tempo de processamento das funções tanh(.), atan(.), e logística mostram que a função atan(.) possui o menor tempo de processamento, seguida da função logística e por último a função tangente hiperbólica.

A função arco tangente é dada pela expressão abaixo:

$$f(x) = \text{atan}(px) \qquad f'(x) = p.\frac{1}{1+x^2} \tag{3.8.5}$$

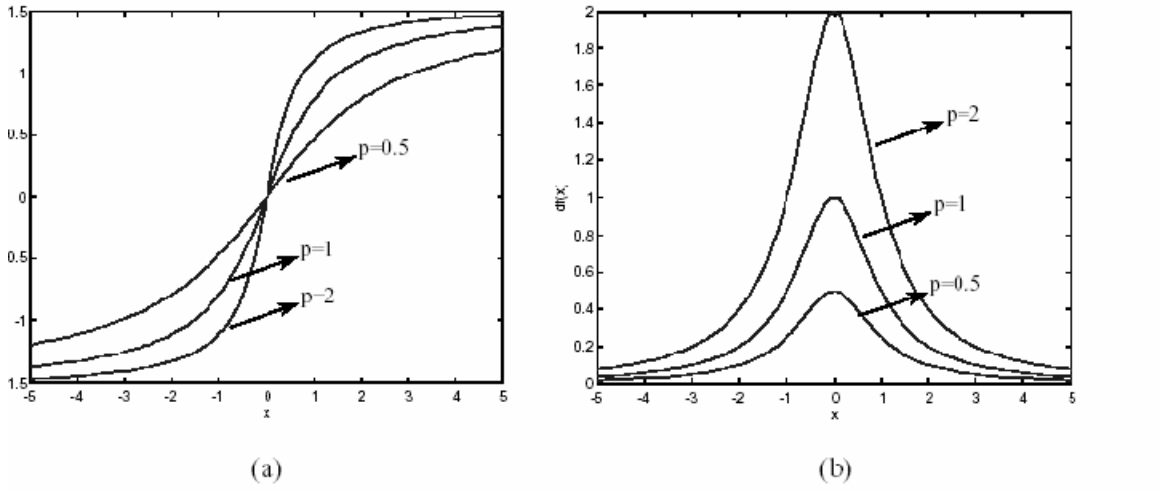

Figura 3.8.4.4 (a) Função arco-tangente (atan). (b) Sua derivada em relação à entrada interna.

Os limites dos intervalos da função apresentada acima podem ser transladados para o intervalo (-1,1) comumente utilizado (SILVA, 1998).

3.8.5 Processamento de uma rede Perceptron Múltiplas Camadas – MLP

A figura 3.8.5.1 apresenta a topologia típica de uma rede neural Perceptron de múltiplas Camadas (MLP). Todos os neurônios de uma camada estão interligados com todos neurônios da camada subseqüente. Cada uma dessas interligações tem um peso $\mathbf{w}_{i,j}$ que multiplica a saída do neurônio anterior para gerar a entrada do neurônio subseqüente.

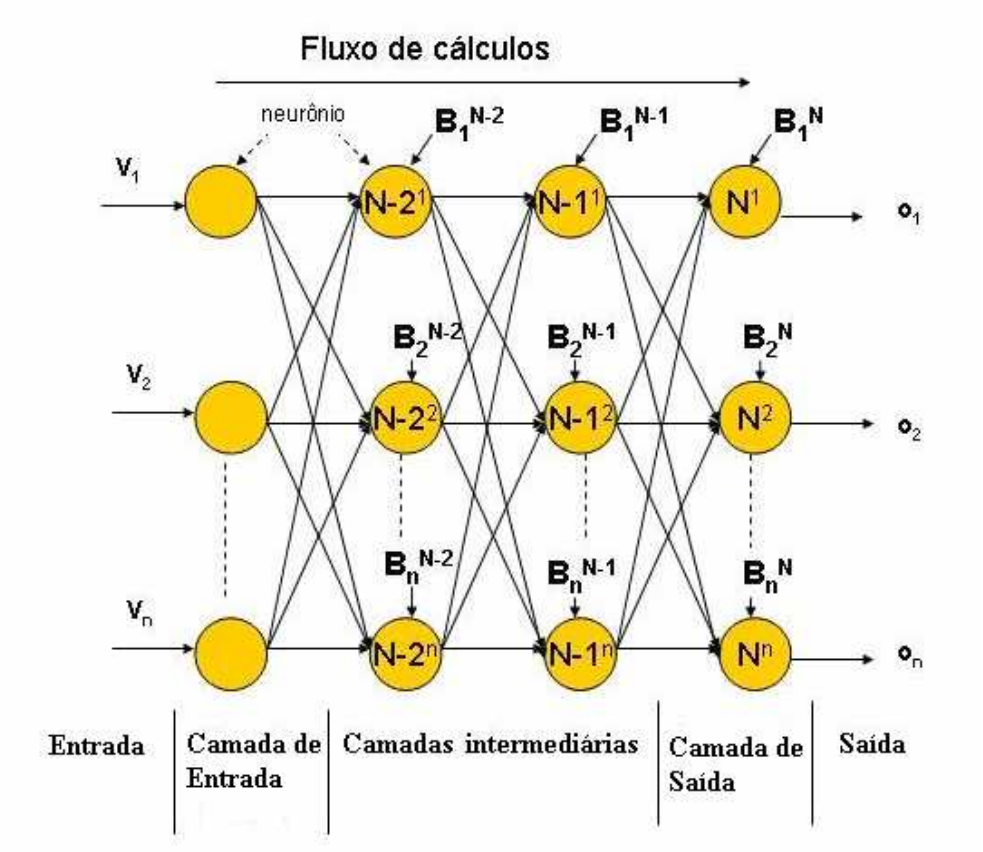

Figura 3.8.5.1 Topogia típica de uma Rede MLP.

A figura 3.8.5.2 apresenta um neurônio genérico **j** da camada intermediária. As entradas deste neurônio **j** consistem de um vetor **x** com n componentes, que são as n entradas no neurônio **j** e um *bias*, cujo valor é 1.

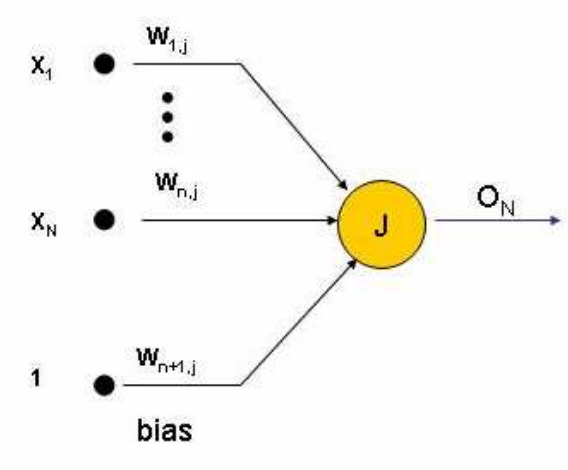

Figura 3.8.5.2 Neurônio genérico da camada escondida.

A dedução à seguir, apresentada por BUFO (2000), define que cada uma das entradas tem um peso $\mathbf{w}_{ij}$ associado a ela. O cálculo dentro do neurônio **j** consiste do somatório das entradas multiplicado pelos seus pesos correspondente, mais o *bias*:

$$I_j = \sum_{i=1}^{n} x_i * w_{i,j} + w_{n+1,j} \qquad (3.8.5.1)$$

Onde:

x = vetor de entradas

w = pesos

I_j = somatório das entradas da camada oculta.

A seguir, a saída do neurônio **j** é calculada a partir de uma função sigmóide (logística):

$$x_j = \frac{1}{1+e^{-I_j}} \qquad (3.8.5.2)$$

Uma vez que as saídas da camada oculta ("*hidden"*) são calculadas, elas são passadas para a camada de saída ("*output"*). Para a camada de saída, os cálculos são análogos às equações (3.8.5.1) e (3.8.5.2).

O cálculo dentro do neurônio de saída é **j** é a somatória conforme a equação (3.8.5.3)

$$I_N = \sum_{j=1}^{n} x_j * w_{j,N} + w_{n+1,N} \qquad (3.8.5.3)$$

e a saída da rede neural é o valor procurado conforme a equação (3.8.5.4):

$$o_N = \frac{1}{1+e^{-I_N}} \qquad (3.8.5.4)$$

A rede neural é treinada para minimizar a função de erro. A equação 3.5.5.5 apresenta uma das várias funções de erro (SNNS, 1995):

$$EG = \sum_{m=1}^{R} \sum_{k=1}^{P} (d_k^m - o_k^m)^2 \qquad (3.8.5.5)$$

Onde:

EG = Erro Global

d = Valor real

o = Valor Predito pela Rede

R = Número de pares de dados de entrada e saída por treinamento

P = Número de neurônios da camada de saída

Vários métodos de otimização podem ser usados para minimizar o erro global expresso pela equação (3.8.5.5). O método mais tradicional é o método do gradiente descendente, o qual foi utilizado por RUMELHART no desenvolvimento da rede (“*backpropagation”)* (RUMELHART et al., 1986).

O peso $w_{i,j}^k$ que é a conexão do neurônio **j** da camada **k-1** para o neurônio **i** da camada **k**, e o *bias* $\mathbf{B_j}$ devem ser incrementados ou decrescidos, visando a minimização do erro global. Estas correções do peso e *bias* podem ser obtidas de (NEURALWARE, 1996).

$$\Delta w_{i,j}^k = -\alpha * \frac{\partial EG}{\partial w_{i,j}^k} \qquad (3.8.5.6)$$

$$\Delta B_j = -\alpha * \frac{\partial EG}{\partial B_j} \qquad (3.8.5.7)$$

O peso α é um coeficiente de aprendizagem. As equações (3.8.5.6) e (3.8.5.7) são resolvidas pela regra da cadeia (LONA, 1996), o que resulta:

$$\Delta w_{i,j}^k = \alpha * \sum_k e_i^k * x_j^{k-1} \qquad (3.8.5.8)$$

$$\Delta B_j = \alpha * \sum_k e_i^k \qquad (3.8.5.9)$$

onde $e_{i,j}^k$ é o erro local do neurônio **i** da camada **k,** que é calculado pela equação (NEURALWARE, 1996):

$$e_i^k = -\frac{\partial EG}{\partial I_i^k} \qquad (3.8.5.10)$$

A equação (3.8.5.10) é resolvida usando a regra da cadeia duas vezes, para obter o erro local da camada **k** devido a todos os erros locais da camada **k+1**, conforme a equação:

$$e_i^k = x_i^k * (1 - x_i^k) * \sum_j (e_j^{k+1} * w_{j,m}^{k+1}) \qquad (3.8.5.11)$$

Esta equação (3.8.5.11) retroage o erro para todos os neurônios da rede que não estão na camada de saída. Para os neurônios da camada de saída o erro local é dado pela equação (12):

$$e_k^N = -\frac{\partial EG}{\partial I_k^N}$$

$$e_k^N = -\frac{\partial EG}{\partial o_k} * \frac{\partial o_k}{\partial I_k^N}$$

$$e_k^N = o_k * (1 - o_k) * (d_k - o_k) \qquad (3.8.5.12)$$

Onde N é o neurônio da camada de saída e **k** é o índice da variável em estudo que assume diversos valores.

No ajuste dos pesos e *bias*; em primeiro os pesos e *bias* entre as camadas ocultas e camadas de saídas são ajustados usando as seguintes equações:

Ajuste dos pesos: $w_{i,j,novo}^N = w_{i,j,velho}^N + \alpha * \sum_{k=1}^{n} e_k^n \pm \beta \qquad (3.8.5.13)$

Ajuste dos bias: $B_{j,novo} = B_{j,velho} + \alpha * \sum_{k=1}^{n} e_k^n \pm \beta \qquad (3.8.5.14)$

O ajuste dos demais pesos e *bias* de todas as camadas que antecedem aquela primeira camada oculta que antecede a camada de saída é feito usando as seguintes equações:

$$\text{Ajuste dos pesos: } w^{K}_{i,j,novo} = w^{K}_{i,j,velho} + \alpha * \sum_{k=1}^{n} e^{k}_{i} * x^{k-1}_{j} \pm \beta \qquad (3.8.5.15)$$

$$\text{Ajuste dos bias: } B_{k,novo} = B_{k,velho} + \alpha * \sum_{k=1}^{n} e^{k}_{i} \pm \beta \qquad (3.8.5.16)$$

(BUFO, 2000).

O fator β nas equações (3.8.5.13),(3.8.5.14),(3.8.5.15) e (3.8.5.16) pode ser utilizado com a finalidade de conseguir obter a convergência do erro EG com menor número de iterações.

Após a topologia da rede neural já consolidada, para que elas sejam operacionalizadas, é necessário estabelecer os seguintes critérios de cálculos:

A – Um conjunto de dados de entrada e o seu correspondente de saída com intervalo de variação definido, obtidos numa situação real.

B – Um conjunto de valores de pesos e *bias* fixados aleatoriamente.

C – A função exponencial de ativação localizada em cada neurônio.

D – estabelecer a função do erro global entre o valor predito pela rede neural e o valor real objetivo.

E – Estabelecer o método de otimização para a minimização do erro global, o qual é retropropagado proporcionalmente para todas as entradas de todos os neurônios de forma a alterar o valor dos pesos e *bias*, até encontrar o conjunto de pesos e *bias* que gerem um erro global mínimo fixado.

Estabelecidos os critérios é então colocada a rede neural em funcionamento com valores reais da variável de entrada e valores reais do resultado objetivo, dentro de um intervalo de variação definido. A isso se chama fase de treinamento, isto é, encontrar os valores dos pesos e *bias* que gerem um erro global mínimo fixado.

Após o treinamento da rede neural, esta é novamente colocada em funcionamento com os pesos e *bias* encontrados na fase de aprendizagem e com um conjunto de dados entrada-saída não utilizados na fase de treinamento, também conhecido como dados de teste, a fim de validar a rede neural.

3.8.6 Utilização de uma Rede Perceptron Múltiplas Camadas – MLP

Após o treinamento de uma rede de múltiplas camadas, ou seja, quando o erro estiver em um nível satisfatório, esta rede neural poderá ser utilizada como uma ferramenta para classificação de dados. Para isto, a rede deverá ser utilizada apenas no modo progressivo ("*feed-forward*"). No caso de uma rede adequadamente treinada, novas entradas são apresentadas à camada de entrada, são processadas nas camadas intermediárias e os resultados são apresentados na camada de saída, como no treinamento, mas sem a retro-propagação do erro, visto que a rede já está estabilizada e representa o correto relacionamento entre valores de entrada e saída. A saída apresentada é o modelo dos dados, na interpretação da rede. Como pode ser observado na figura 3.8.5.1, o fluxo dos dados ocorre somente da camada de entrada para a camada de saída, sem nenhum fluxo de informação no sentido contrário.

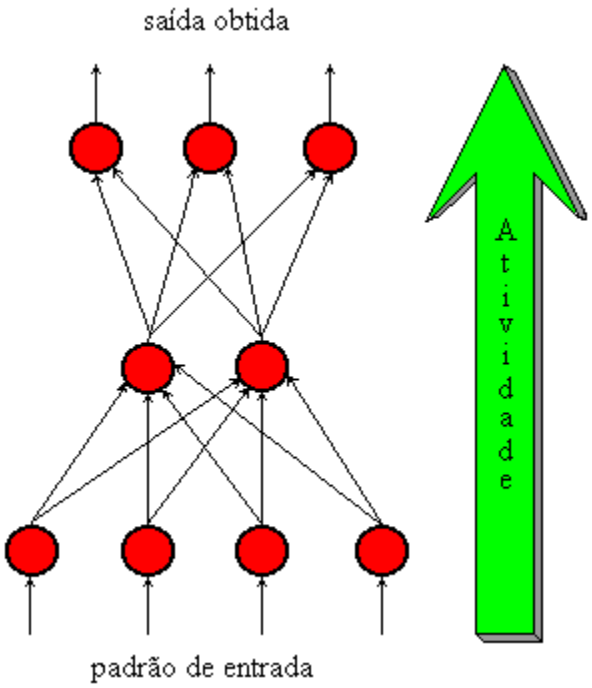

Figura 3.8.6.1 Utilização da rede MLP, após a estabilização.

Capítulo 4

4 Material e Métodos

4.1 Introdução

Neste capítulo serão apresentados três modelos de classificação de frutos baseados na interatividade do número de propriedades físicas. Os modelos de máquinas de classificação de frutos comercialmente disponíveis consideram principalmente uma única propriedade física, como Peso ou Diâmetro, na separação em classes. Os modelos teóricos propostos neste capítulo apresentam a classificação em termos vetoriais e propõem a consideração da interatividade das propriedades físicas nos modelos comerciais.

Três diferentes modelos vetoriais são apresentados, considerando o número de propriedades físicas, associadas para a classificação:

1. Cenário Unidimensional: considera a classificação baseada em apenas uma propriedade física.
2. Cenário Bidimensional: considera a classificação baseada na interação de duas propriedades físicas.
3. Cenário Tridimensional: considera a classificação baseada na interação de três propriedades físicas.

4.2 Classificador Artificial de Frutos

Nos capítulos anteriores foram apresentadas as principais características das redes neurais e os problemas relacionados à classificação de frutos. Neste capítulo serão apresentados

conjuntos de cenários, que permitirão criar um modelo de classificação artificial para qualquer fruto e número de parâmetros de classificação.

A classificação natural de frutos, através de classificadores humanos, possui algumas características peculiares, tais como, a classificação para um mesmo fruto, pode variar entre diferentes classificadores, ou seja, diferentes pessoas possuem diferentes percepções de classificação. Um dado fruto pode estar apto ou não para um determinado propósito, dependendo do nível de tolerância individual de cada classificador. Além desta variação entre diferentes classificadores existe a variabilidade temporal implícita para o mesmo classificador humano, isto é, uma mesma pessoa classificando, pode variar entre janela temporais, o nível de tolerância de descarte para um mesmo fruto.

A classificação natural consegue empiricamente selecionar um dado fruto baseado em inúmeros parâmetros simultâneos, diferentemente das máquinas classificadoras artificiais. Analisando o caso de uma dona de casa que vai a uma feira livre para comprar um dado fruto, como exemplo uma banana, esta classificadora humana, de forma inconsciente estará analisando este fruto através de inúmeros canais naturais de análise.

A tabela 4.2.1 apresenta alguns dos principais canais naturais, assim como ações de disparo, habilitados em um classificador, aptos a entrada de dados referentes a análise sensorial de um fruto.

Tabela 4.2.1 Canais Humanos de Análise.

	Canais Humanos de Análise		
	Canal/Ação	**Propriedade**	**Inferência/Análise**
1	Toque	Firmeza (“Firmness”)	A fruta está muito dura ou macia demais para consumo
2	Toque	Peso	O peso pode dar a indicação da palatabilidade do fruto
3	Visual	Manchas Pragas / Doenças	A presença de qualquer mancha sobre o fruto pode dar a indicação de podridão interna ou inaptidão de consumo.
4	Visual	Cor	O fruto pode estar verde indicando que este ainda não está maduro para consumo.
5	Olfato	Aroma	No caso da banana não é um dos canais mais apropriados para análise da aptidão do fruto para consumo, porém pode ser usado como um canal adicional.

De forma simplista a classificação humana de um fruto é apenas a análise de aptidão de consumo. Em uma análise mais profunda pode-se entender que todos estes canais de comunicação entre fruto e classificador são uma combinação de propriedades físicas, apresentadas simultaneamente em paralelo, permitindo que todos estes parâmetros sejam analisados em conjunto, classificando o fruto como apto ou inapto para um dado propósito. Pode parecer estranho este tipo de análise, mas fisicamente o que acontece é que a dona de casa habilita inúmeros transdutores naturais que transformam as propriedades físicas dos frutos em impulsos elétricos que são levados a rede neuronal do cérebro dela (classificador), ou seja, implicitamente ela usa o nariz como transdutor para sentir o aroma, a mão como transdutor de firmeza ("firmness"), peso e inúmeras outras inferências relacionadas às propriedades físicas e mecânicas do fruto em estudo. A tabela 4.2.2 mapeia alguns dos transdutores naturais de uma pessoa humana:

Tabela 4.2.2 Transdutores Naturais de Classificação

	Transdutores Naturais de Classificação		
	Transdutor	**Ação**	**Propriedade**
1	Mão	Toque	Firmeza ("Firmness")
2	Mão	Toque	Peso
3	Olhos	Visual	Manchas Pragas / Doenças
4	Olhos	Visual	Cor
5	Nariz	Olfato	Aroma

Para a análise das condições de um determinado fruto, em um certo instante, é necessário proceder-se à observação sistemática das propriedades do mesmo fruto. Os dados relativos a estas propriedades são obtidos via transdutores.

Na prática os transdutores são equipamentos eletro-mecânicos que transformam grandezas física em outras. Existem inúmeros transdutores que detectam luz, som, pressão, peso e outros, convertendo alterações de grandezas físicas não elétricas (ex. pressão ou temperatura) em quantidades elétricas (ex. resistência).

No caso da classificação artificial de frutos é necessário obter o sinal destes diferentes canais de entrada para processamento em paralelo. Para a obtenção dos valores destes canais deve-se ter um transdutor para cada canal, o qual fará a conversão desta grandeza física em um sinal elétrico, capaz de ser recebido por um microcomputador. Este processo faz exatamente a conversão de um sinal analógico, contínuo, em um sinal discreto digital. Para realizar este

processo de conversão é necessário que os transdutores estejam ligados a circuitos de conversão chamados de conversores de sinais Analógico/Digital, no caso da recepção para processamento, e no caso de ação pós-processamento deverá ser usado o caso contrário, conversores de sinais Digital/Analógico.

4.2.1 Conversores D/A e A/D

Conversores Analógico/Digital e Digital/Analógico são dispositivos eletrônicos responsáveis pela obtenção de sinais e monitoramento de processos. Estes equipamentos permitem o interfaceamento de medição/controle de propriedades físicas do mundo analógico com o mundo digital, permitindo que sinais de variáveis físicas sejam alimentados a um programa de computador, processados e se necessário sinais são enviados através de componentes servo mecânicos para controle do processo. Desta forma existe um fluxo iterativo entre os transdutores que passam as informações para os circuitos de interfaceamento Analógico/Digital, permitindo o processamento do computador e interação final com os controladores de processo. A figura 4.2.1 apresenta de forma simplista a interação entre estas interfaces e o computador.

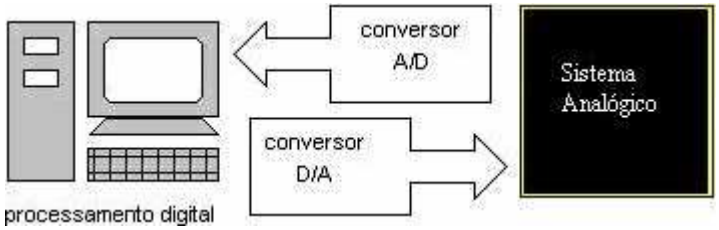

Figura 4.2.1 Ilustração de um computador "conversando" com o ambiente externo.

O processo de aquisição de dados via transdutores específicos permite que as variáveis físicas tais como Peso, Diâmetro, cor e outras possam ser obtidas para a criação de mapas de padrões do tipo daqueles, que serão apresentados nos próximos tópicos. Vários destes transdutores estão disponíveis comercialmente, já disponibilizando interfaces A/D. Neste trabalho, é tido como premissa a existência de tais equipamentos, assim como os mapas de padrões. À seguir será apresentado a titulo ilustrativo o funcionamento de um conversor analógico básico.

Um dos tipos mais utilizados de conversores de sinais Analógico em Digitais, é o tipo por aproximação sucessiva ("*Sucessive Approximation A/D Converter*"), são os mais comuns

entre os conversores A/D, permitem uma conversão rápida, proporcionando uma gama de 100.000 ou mais conversões por segundo (ZAMBALDE, 1991).

Na técnica de aproximação sucessiva, é utilizado um algoritmo para converter a entrada analógica em digital. Este algoritmo consiste em ajustar o bit mais significativo (MSB) para 1 e todos os outros bits para 0. O comparador compara a saída do conversor D/A (Vd) com o sinal da entrada analógica (Ve). Se Vd > Ve, o 1 é removido do MSB e *enviado* para o próximo bit mais significativo. Se Ve > Vd, o MSB permanece como 1 e o próximo bit mais significativo também recebe 1. Assim o 1 é deslocado e testado em cada bit do decodificador D/A até o final do processo, para obter o valor binário equivalente.

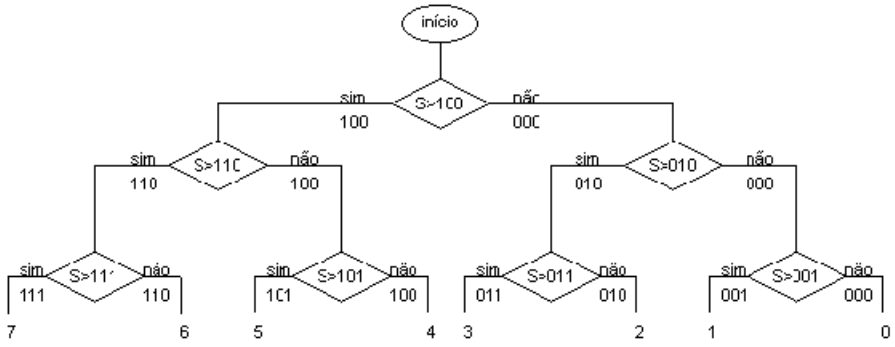

Figura 4.2.2 Fluxograma para conversão de um número de 3 bits

Um circuito comparador compara a entrada analógica com a saída de um conversor D/A controlado pela lógica conhecido como SAR ("*Sucessive Approximation register*"), que é basicamente um registrador de deslocamento. Sob o comando do relógio ("*clock*") o SAR é inicialmente colocado em zero. Assumindo uma entrada analógica (Ve) positiva, o registrador de deslocamento liga o primeiro bit (MSB). Se o comparador detecta que a saída D/A é menor que a entrada, este bit é deslocado, caso contrário é desligado. Assim, sucessivamente o próximo bit é ligado, a palavra é comparada e mantida ou modificada de acordo com o resultado da comparação do resultado. A seqüência continua até que o último bit significativo (LSB) seja comparado e ajustado, após isto, o sinal convertido é validado e o dispositivo que o espera pode recebê-lo.

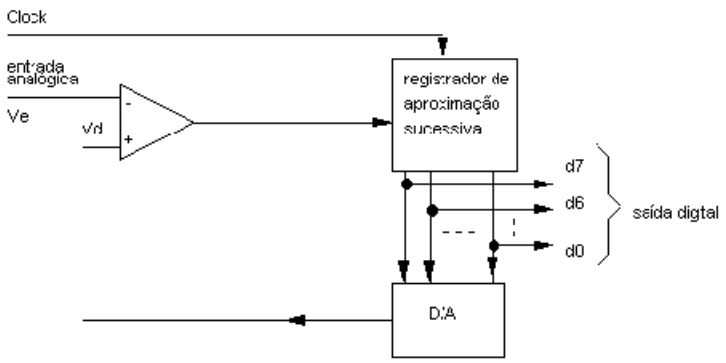

Figura 4.2.3 Conversor A/D aproximação sucessiva

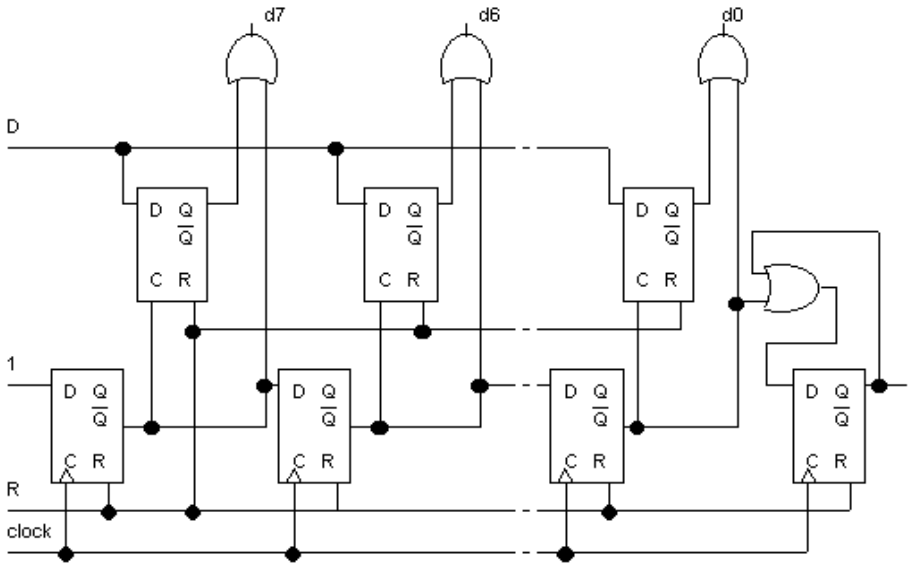

Figura 4.2.4. Exemplo para a realização de um registro de aproximação sucessiva.

Diversos são os conversores análogico/digitais que podem ser utilizados para implementar o sistema. A escolha do tipo de conversor a ser adotado é feita baseada no custo, exatidão, velocidade e disponibilidade de componentes.

A figura 4.2.5 apresenta um modelo de sistema de Aquisição de dados e conversão do sinal da propriedade física em um sinal digital apropriado para ser trabalhado dentro de um computador. Como pode ser observado o sinal analógico da propriedade física é fornecido à um

transdutor especifico, calibrado e apropriado para este tipo de grandeza física. Este transdutor após receber o sinal da propriedade física trabalha o sinal amplificando/condicionando, assim como eliminando ruídos. Através do uso de alguns componentes eletrônicos de condicionamento o sinal chega finalmente ao conversor analógico/digital que irá transformar o sinal em um sinal discreto, formado por "0" e "1", sinal binário. Depois de convertido o sinal e devidamente recebido através dos "*ports*" do programa, este valor convertido da propriedade física, pode ser trabalhado pela rede neural possibilitando a classificação deste sinal. Uma vez que exista uma definição classificatória, é possível através do processo inverso, utilizando-se de um circuito digital/analógico, fazer o monitoramento desta propriedade física. Como exemplo é possível que o sinal saído do "*port*" do computador, faça a ativação de um motor de passo que ativa um mecanismo especifico de uma esteira de seleção.

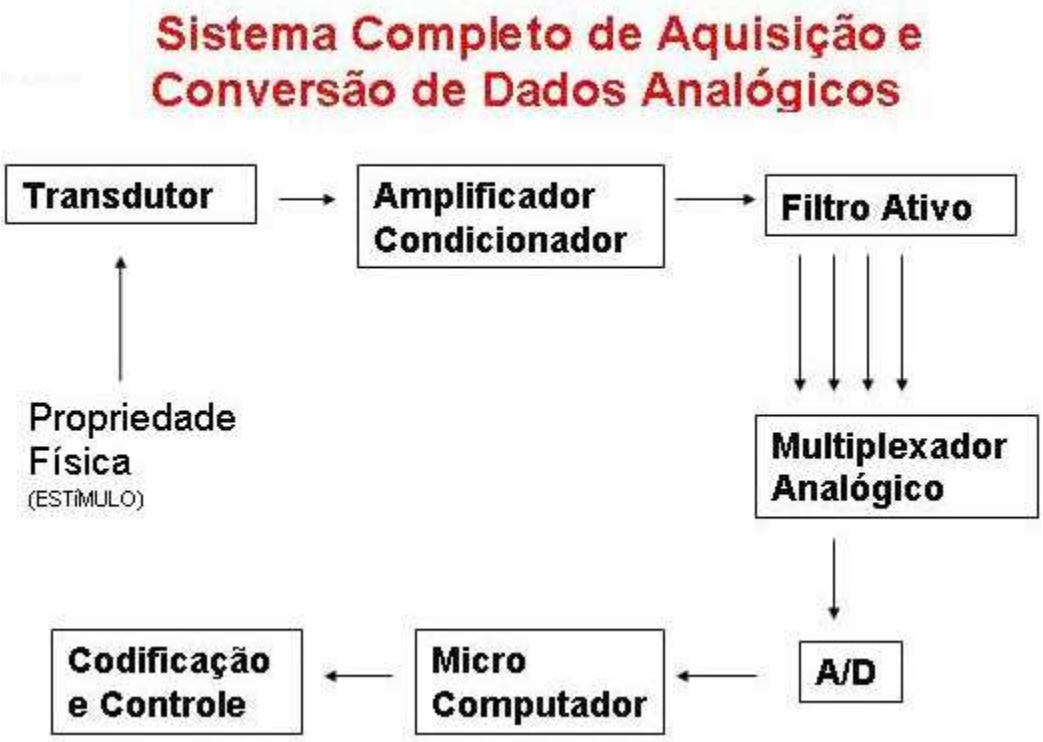

Figura 4.2.5. Sistema Completo de Aquisição/Conversão de dados Analógicos.

Supondo-se que cada canal de entrada corresponda a um vetor de entrada, relacionado ao valor de uma propriedade física de um fruto, como Peso, Cor, Diâmetro, pode-se então criar vários cenários de classificação, em função do número de parâmetros. Este trabalho propõe uma forma diferente de modelagem do processo classificatório de frutos. Considerando-se que cada propriedade física de um fruto pode ser tratada como um vetor ou canal de entrada, propõe-se a

criação de cenários vetoriais/dimensionais que levam em conta a interação das propriedades individuais na classificação de frutos. A tabela 4.2.3 apresenta os vários cenários da nova proposta para a classificação de frutos.

Tabela 4.2.3 Cenários de Classificação de Frutos.

	Cenários de Classificação de Frutos			
	Cenários	**Propriedades**	**Número Canais/Vetores**	**Bits ***
1	Unidimensional	Peso	1	4
2	Bidimensional	Peso +Diâmetro	2	8
3	Tridimensional	Peso+ Diâmetro + "Firmness"	3	12
4	N dimensional	N	N	N

* O número de Bits é função da resolução do conversor A/D, quanto maior o número de Bits, menor são os intervalos de resolução para uma mesma faixa de valores.

A figura 4.2.6 traz a representação gráfica dos vetores e cenários, onde cada canal pode ser visto como um transdutor de entrada do valor de uma dada propriedade física de um fruto: :

- Cenário Unidimensional: representa o valor de um único sinal ou uma propriedade física.
- Cenário Bidimensional: representa o valor de dois sinais ou duas propriedades físicas.
- Cenário Tridimensional: representa o valor de três sinais ou três propriedades físicas.
- Cenário *N* dimensional: representa o valor de *N* sinais ou *N* propriedades físicas.

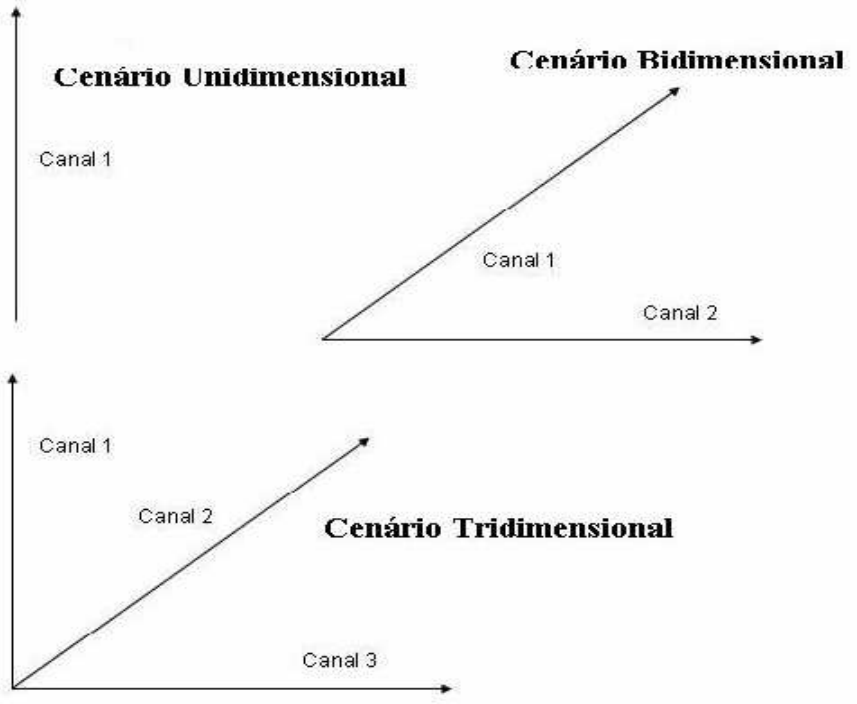

Figura 4.2.6 – Canais e Cenários

O cenário unidimensional é o caso mais simples, onde tem-se a entrada de apenas um parâmetro para classificação, este é o caso mais comum das máquinas de classificação comerciais. Os frutos são classificados com base em apenas um propriedade física, tal como Peso, Diâmetro, Densidade, Cor, ou outra propriedade física mensurável. Cada canal de entrada deve utilizar um transdutor para cada propriedade física respectivamente. No cenário unidimensional deve ser utilizado apenas um transdutor, que deve transformar o sinal analógico da grandeza medida em sinal digital, para isto é necessário o uso de uma porta exclusiva de um conversor Analógico/Digital. O transdutor utilizado para aquisição da propriedade em análise fornece o sinal de apenas um canal de entrada, sendo o número de bits de entrada uma função do conversor A/D. O conjunto de bits fornecidos a uma rede neural, possivelmente programada em uma CPU ou EPROM("*Erasable Programmable Read Only Memory*"), deverá então ser processado pela rede e classificado em um dado valor binário de saída.

Os demais cenários bidimensional, tridimensional e *N* dimensional são criados à partir do aumento do número de entradas, ou seja, à medida que se aumentam o número de canais de entrada tem-se o aumento das dimensões e inter-relacionamento entre vetores de padrões.

O número de bits determina a resolução do valor da propriedade física, ou seja, através do aumento do número de bits verifica-se que o padrão de entrada tem valores cada vez menores,

permitindo uma análise mais minuciosa. Na tabela 4.2.4 é ilustrado que para o uso de 2 Bits somente 4 valores são possíveis para uma faixa de valores, por exemplo, supondo-se que será usado uma faixa de valores de Peso, variando entre 100 gramas e 260 gramas, com uma variação de 160 gramas, o uso de 4 Bits permite ter uma variação de 10 gramas para cada variação no conversor A/D, já no caso de 8 Bits tem-se uma resolução de 0.625 gramas, que pode ser considerada inapropriada para o exame em questão.

Tabela 4.2.4 Intervalos de Resolução.

RESOLUÇÃO							
2 Bits		3 Bits		4 Bits		8 Bits	
1	00	1	000	1	0000	1	00000000
2	10	2	001	2	0001	2	
3	01	3	010	3	0010	3	
4	11	4	100	4	0011	4	
2x2=4		5	011	5	0100	5	
Resolução	=160/4	6	110	6	0101	6	
		7	101	7	0110	7	
		8	111	8	0111	8	
		2x2x2=8		9	1000	9	
		Resolução	=160/8	10	1001	10	
				11	1010	11	
Supondo-se uma variação de 160				12	1011	12	
unidades de uma dada propriedade				13	1100	13	
física tem-se:				14	1101	14	
2 Bits	40			15	1110	15	
3 Bits	20			16	1111	16	
4 Bits	10			2x2x2x2=16		17	
8 Bits	0,625			Resolução	=160/16	...	
						256	11111111
						2x2x2x2x2x2x2x2=256	
						Resolução	=160/256

O presente trabalho procura mostrar a possibilidade do uso simultâneo, ou também chamado uso em paralelo, de vários padrões de entrada, representando as diferentes propriedades físicas possíveis na classificação de frutos. Para facilitar o entendimento serão usados apenas 4 Bits para cada padrão de entrada. A tabela 4.2.5 apresenta a análise de uma propriedade física, neste caso o Peso, para um dado fruto e classificação deste em relação a esta propriedade:

Tabela 4.2.5 Espaço amostral de 4 Bits

	4 Bits	Peso	Classes
1	0000	180	
2	0001	185	A
3	0010	190	1
4	0011	195	00
5	0100	200	
6	0101	205	
7	0110	210	B
8	0111	215	2
9	1000	220	10
10	1001	225	
11	1010	230	C
12	1011	235	3
13	1100	240	01
14	1101	245	D
15	1110	250	4
16	1111	255	11

Cada vetor de entrada corresponde a um valor analógico de uma propriedade física, este vetor traduzido em sua forma binária representa o formato necessário para a entrada em uma rede neural com entrada binária. Dependendo do aprendizado da rede neural este vetor binário de saída pode ter a classificação apropriada desta propriedade física.

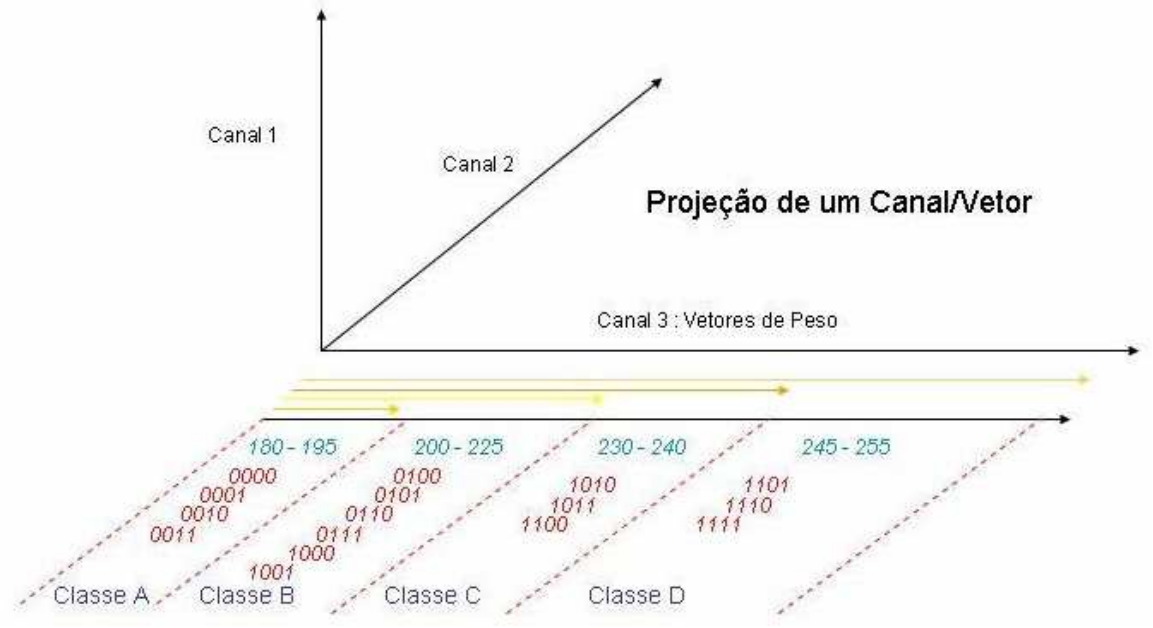

Figura 4.2.7. Projeção binária/analógica dos valores de vetores para um canal de Peso.

Na figura 4.2.7 tem-se a projeção de apenas um vetor binário, ou também chamado de canal de entrada, para uma rede neural. Como pode ser observado existem 4 classes (A, B, C, D) projetadas, com respectivas faixas de valores binário/analógico. Caso a entrada do canal tivesse o número de bits aumentado de 4 para 5 bits o número de intervalos já seria duplicado, ou seja, a resolução seria aumentada de 16 intervalos (2^4) para 32 intervalos (2^5), melhorando a sensibilidade das entradas.

À medida que o número de canais de entrada é aumentado, ocorre a composição vetorial destes vetores, correspondendo a um vetor resultante que representará a interatividade de todas estas propriedades físicas. A figura 4.2.8 apresenta 3 cenários dimensionais diferentes, com suas respectivas projeções dependendo do número de entradas/canais. O vetor resultante nada mais é que o resultado da composição vetorial das componentes individuais, ou seja, da interação dos canais de entrada analisados.

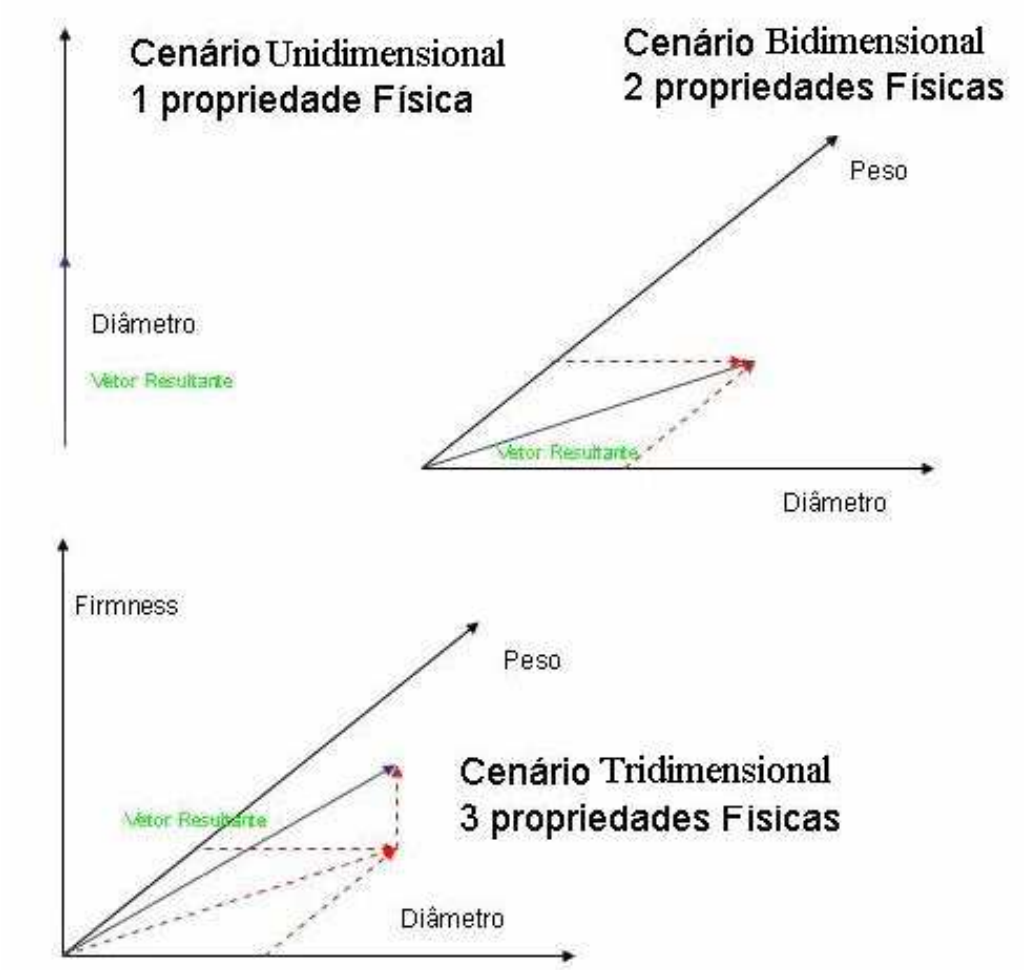

Figura 4.2.8. Projeção de vetores resultantes por dimensão.

Entende-se como padrão o vetor de entrada da rede neural que representa o valor individual de uma propriedade física do fruto. O aumento do número de padrões traz como conseqüência o aumento do número de bits e o número de amostras. Nos exemplos abaixo os 4 primeiros bits correspondem à propriedade física 1(bits 1 a 4), os 4 seguintes à propriedade física 2 (bits 5 a 8) e finalmente os demais bits à propriedade física 3 (bits 9 a 12).

Tabela 4.2.6 Exemplo dimensional de vetores

Número de Padrões	Bits	Exemplo	Amostras
1	4	0010	16
2	8	00101101	256
3	12	001011011001	4096

Quando é dito que ocorre um paralelismo de sinais, na verdade afirma-se que cada transdutor deverá receber o sinal referente à propriedade física a qual esteja monitorando, converter este sinal analógico em sinal digital (bits), e após a fase de conversão de todos os transdutores em sinais individuais na forma digital, todos estes bits deverão ser integrados simultaneamente, de forma que se crie um único vetor. O exemplo da figura 4.2.9 apresenta o conceito da avaliação simultânea de duas propriedades físicas, cada transdutor transformando o sinal analógico (voltagem) em um sinal digital individual, composto de 4 bits. Através da associação dos dois vetores de 4 bits, forma-se um novo vetor de 8 bits que apresentado à rede neural irá classificar esta entrada à uma determinada classe no espaço. A explicação mais detalhada da composição resultante será explicada mais à frente no capítulo referente ao cenário bidimensional.

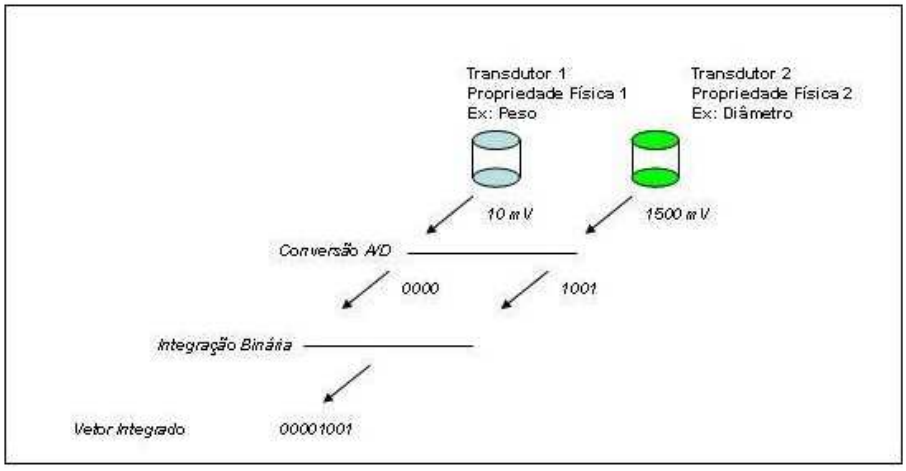

Figura 4.2.9. Integração de sinais digitais dos transdutores .

A figura 4.2.10 apresenta a projeção de 2 vetores resultantes da composição vetorial de suas respectivas componentes de Peso e Diâmetro. O vetor resultante 1 é a somatória de um vetor de Peso e um vetor de Diâmetro, que conforme as dimensões mapeia o sinal de entrada para a classe B. O vetor resultante 2 usa o mesmo processo e mapeia para a Classe A, no espaço.

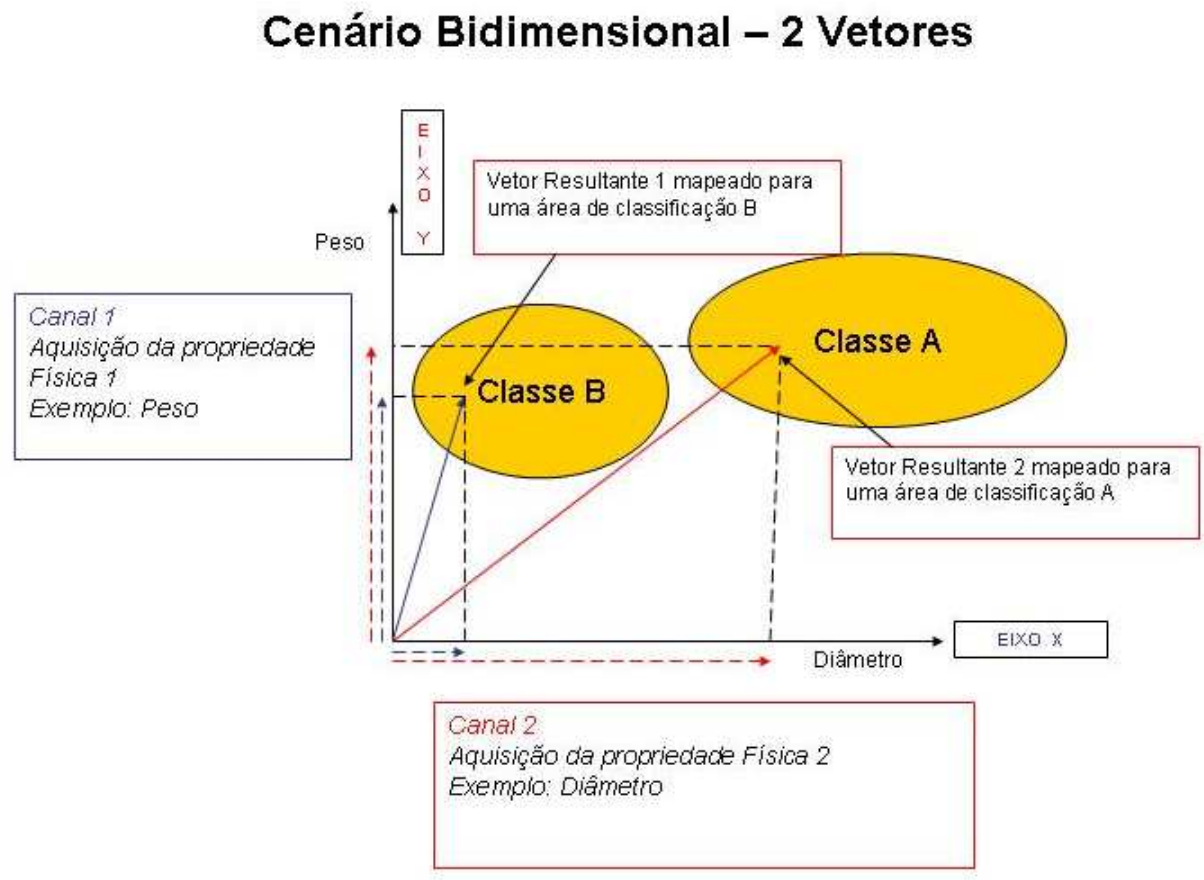

Figura 4.2.10. Composição Vetorial de duas propriedades físicas

A figura 4.2.11 apresenta como pode ser variado o processo de integração de bits e número de entradas, como pode ser observado tem-se:

1. O número de bits individuais por entrada pode ser aumentado, ou seja, o vetor de entrada pode ter um número maior de bits de entrada e;
2. Pode-se aumentar o número de componentes vetoriais de entrada, ou seja, mais propriedades podem ser analisadas simultaneamente.

Quanto maior o número de bits de entrada e maior o número de propriedades por vetor resultante, maior será o processamento de cálculo para a convergência da rede neural.

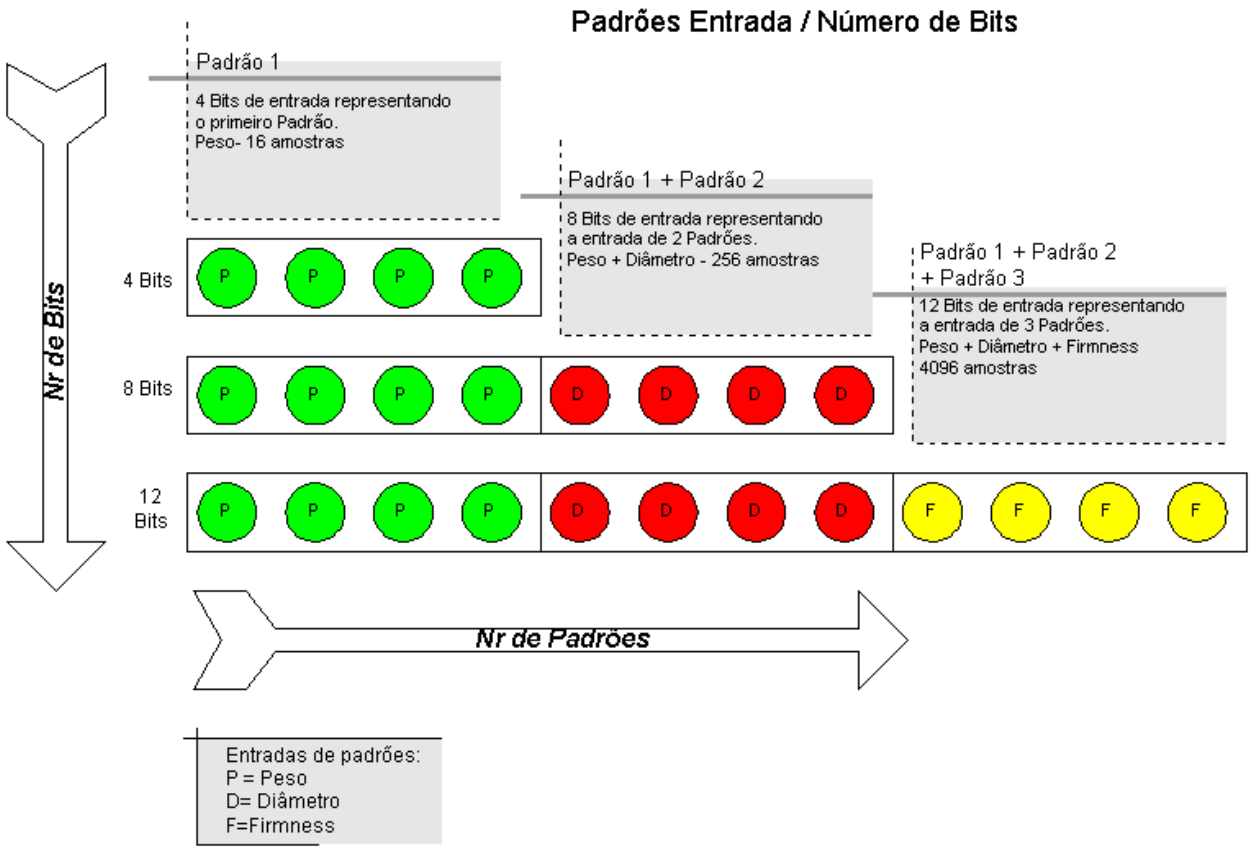

Figura 4.2.11. Integração de sinais digitais dos transdutores

A discussão apresentada neste capítulo será utilizada como base para criar um sistema classificatório variando o número de bits para cada canal de entrada assim como o número de canais. Nos próximos tópicos serão descritos individualmente os cenários:

- Cenário Unidimensional
- Cenário Bidimensional
- Cenário Tridimensional
- Cenário N dimensional

4.3 Simulação por Software – SNNS

SNNS (Stuttgart Neural Network Simulator) é um programa de simulação de redes neurais, focado principalmente para ser executado em estações Unix, desenvolvido no Instituto para Sistemas Distribuídos e Paralelos de alta performance dentro da Universidade de Stuttgart. O objetivo do projeto SNNS é criar um ambiente de simulação eficiente e flexível para pesquisas e aplicações de redes neurais. O simulador é gratuito para pesquisadores e pode ser obtido para "*download*" em:

ftp.informatik.uni-stuttgart.de

no subdiretório /pub/SNNS/ SNNSv4.1.tar.Z

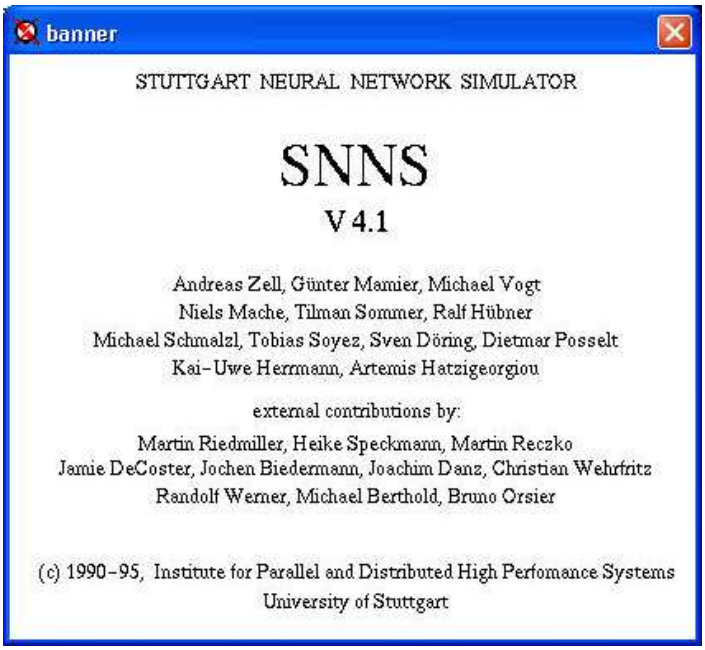

Figura 4.3.1 Tela inicial do Simulador SNNS V 4.1

O simulador SNNS consiste de dois componentes principais:

1. Simulador com o programa principal (Kernel) escrito em C++
2. Interface gráfica

O simulador opera na estrutura de dados das redes neurais, realiza todas as operações de aprendizado e testes, ajuda no processo de configuração e testes dos parâmetros da rede, topologia da rede, assim como tipos de implementações de redes neurais.

Dentre os tipos de redes possíveis de implementação usando o SNNS tem-se:

- Perceptron de múltiplas camadas ("*Backpropagation (BP) for feedforward networks"*)
    - vanilla ("*online*") BP
    - BP com "momentum term"
    - BP em lote ("*batch*")
- Counterpropagation
- Quickprop
- Backpercolation 1
- RProp
- Generalized radial basis functions (RBF)
- ART1 , ART2 e ARTMAP
- Cascade Correlation
- Recurrent Cascade Correlation
- Dynamic LVQ
- Backpropagation through time (for recurrent networks)
- Quickprop through time (for recurrent networks)
- Self-organizing maps (Kohonen maps)
- TDNN (time-delay networks) with Backpropagation
- Jordan networks
- Associative Memory

Para executar o SNNS em ambiente Windows é necessário que seja executado um programa chamado X-Win32 responsável pela simulação do ambiente Unix no ambiente Windows, desta forma o SNNS pode ser executado.O utilitário X-Win32 pode ser obtido no mesmo site do simulador SNNS.

Todos os testes foram realizados em um computador PC Pentium III , 700 MHz, com 256 MB RAM, executando o Sistema Operacional Windows 2000 ® Professional. Todos os cenários testados neste trabalho tiveram convergência com intervalos de tempo inferior a duas horas.

® - Marca Registrada - Microsoft Corporation

O simulador SNNS permite que as redes sejam implementadas tanto em linha de comando quanto usando a interface gráfica.

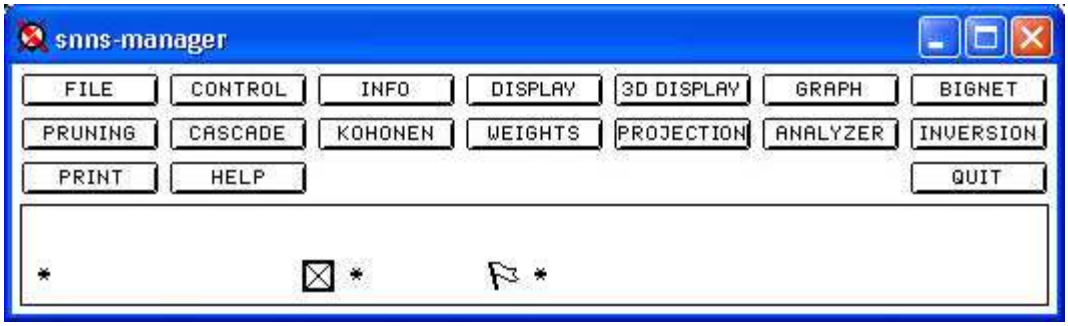

Figura 4.3.2 Interface gráfica do simulador SNNS V4.1

A Janela ilustrada na Figura 4.3.2 representa a interface principal de operação do SNNS. Todas as funcionalidades são acessadas e gerenciadas através desta janela, permitindo que o usuário crie redes (Bignet), faça o treinamento da rede (Control), salve e restaure arquivos e resultados (File).

As implementações das redes neurais necessitam de vários arquivos de suporte.

.NET - arquivo de configuração propriamente da topologia da rede, responsável pela configuração do tipo de rede e parâmetros de treinamento.

.PAT - arquivo de vetores de entradas e saídas, responsável pelo espaço de vetores para treinamento da rede.

.RES - arquivo de resultados do treinamento.

.CFG - configurações da interface visual.TXT – arquivo de log da convergência de Erros.

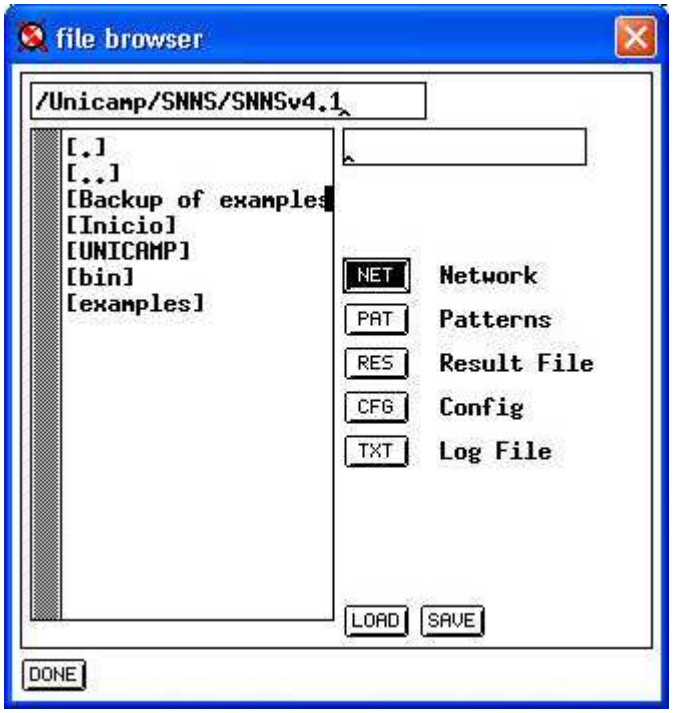

Figura 4.3.3 Janela de Carga e Salvamento de arquivos da Rede Neural

Na janela de arquivos, ilustrado na Figura 4.3.3, são abertos ou salvos os arquivos de configuração e resultados da rede, permitindo simulações com as diversas arquiteturas de rede (topologia).

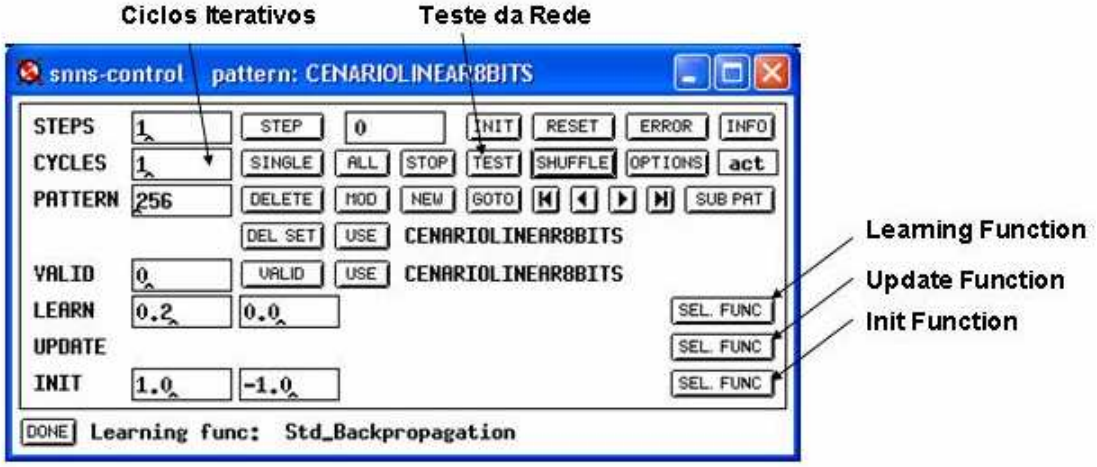

Figura 4.3.4 – Janela de controle do SNNS e treinamento da Rede Neural

A janela de controle, apresentada através da Figura 4.3.4, permite que o usuário do programa SNNS, altere as funções de aprendizado, atualização e inicialização de uma rede criada para simulação. Os valores de taxa de aprendizado e inicialização da rede são atualizados na janela de controle.

Segue abaixo o arquivo para o treinamento de uma rede neural de 4 bits de entrada e 2 bits de saída, com duas camadas escondidas com bits cada uma.

```
SNNS pattern definition file V3.2

generated at 9/26/2001 12:39:38 PM


No. of patterns : 16

No. of input units : 4

No. of output units : 2


# Input pattern 1:

0 0 0 0

# Output pattern 1:

0 0

# Input pattern 2:

0 0 0 1

# Output pattern 2:

0 0

# Input pattern 3:

0 0 1 0

# Output pattern 3:

0 0

# Input pattern 4:

0 0 1 1

# Output pattern 4:
```

```
0 0

# Input pattern 5:

0 1 0 0

# Output pattern 5:

0 0

# Input pattern 6:

0 1 0 1

# Output pattern 6:

1 0

# Input pattern 7:

0 1 1 0

# Output pattern 7:

1 0

# Input pattern 8:

0 1 1 1

# Output pattern 8:

1 0

# Input pattern 9:

1 0 0 0

# Output pattern 9:

1 0

# Input pattern 10:

1 0 0 1

# Output pattern 10:

1 0

# Input pattern 11:

1 0 1 0

# Output pattern 11:
```

```
1 0

# Input pattern 12:

1 0 1 1

# Output pattern 12:

0 1

# Input pattern 13:

1 1 0 0

# Output pattern 13:

0 1

# Input pattern 14:

1 1 0 1

# Output pattern 14:

0 1

# Input pattern 15:

1 1 1 0

# Output pattern 15:

1 1

# Input pattern 16:

1 1 1 1

# Output pattern 16:

1 1
```

Segue abaixo o modelo de arquivos.NET gerado pela interface de controle do SNNS e correspondente a topologia da figura 4.3.6. Estes arquivos são do tipo texto (.txt) onde apresentam os dados relativos a inicialização e topologia da rede neural.

SNNS network definition file V1.4-3D
generated at Sat Sep 22 19:40:30 2001

network name : CENARIOLINEAR4BITS
source files :

no. of units : 10 [Número de neurônios]

no. of connections : 16 [Número de conexões entre neurônios]

no. of unit types : 0

no. of site types : 0

learning function : Std_Backpropagation [Função de aprendizado]

update function : Topological_Order [Função de atualização]

unit default section :

act	bias	st	subnet	layer	act func	out func
0.00000	0.00000	h	0	1	Act_Logistic	Out_Identity

unit definition section :

no.	typeName	unitName	act	bias	st	position	act func	out func	sites
1			1.00000	-0.02774	i	2, 2, 0			
2			1.00000	0.64852	i	3, 2, 0			
3			1.00000	0.38270	i	4, 2, 0			
4			1.00000	-0.06907	i	5, 2, 0			
5			0.51784	-2.10892	h	2, 5, 0			
6			0.71998	3.88833	h	3, 5, 0			
7			0.56710	1.89269	h	2, 8, 0			
8			0.99838	-1.95729	h	3, 8, 0			
9			0.99050	-24.60246	o	2,11, 0			
10			0.99159	7.83340	o	3,11, 0			

connection definition section :

target	site	source:weight [Pesos após o treinamento]
5		4:-0.69187, 3:-0.72856, 2: 2.01415, 1: 1.58659
6		4:-3.44482, 3:-7.74950, 2: 7.09261, 1: 1.15775
7		6:12.76987, 5:-20.88814
8		6:-2.53190, 5:19.70807
9		8:18.93605, 7:18.23966
10		8: 9.33947, 7:-21.84351

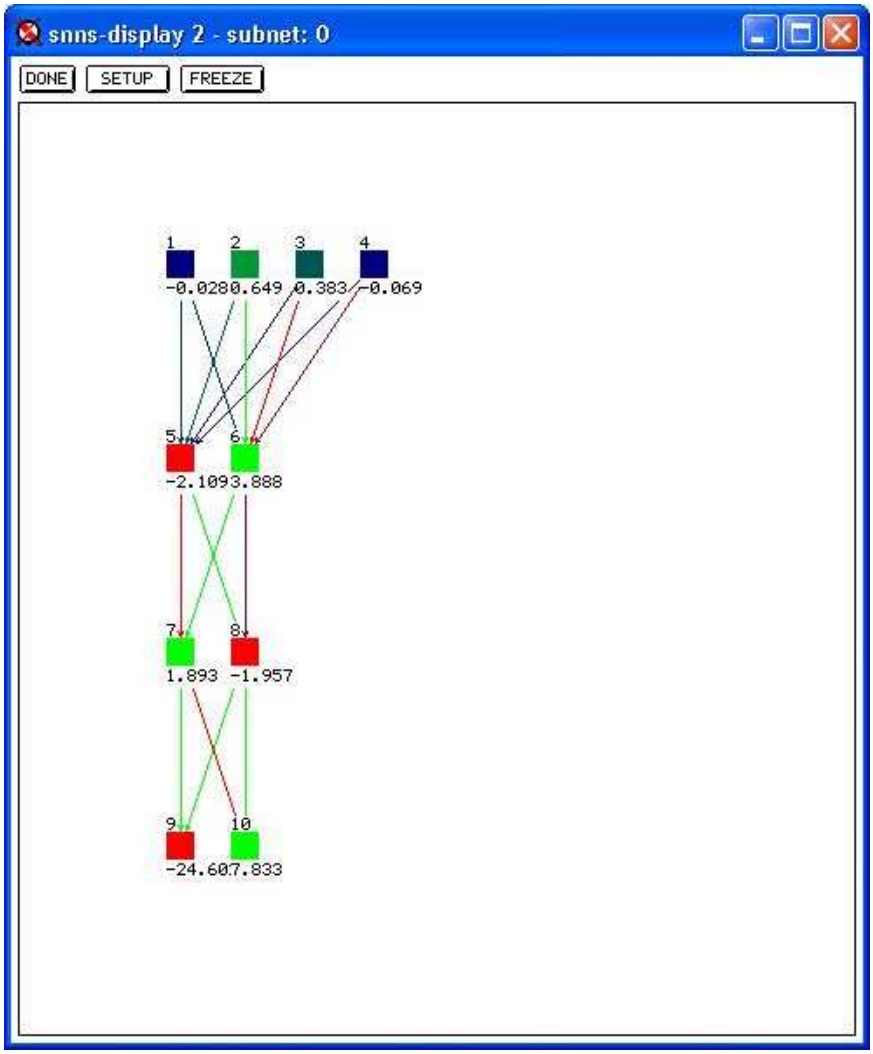

Figura 4.3.5 – Topologia de uma rede neural com camadas de neurônios e conexões com a apresentação dos valores de *bias* em cada neurônio.

Os pesos definem a força entre as conexões e guardam todo o conhecimento da rede, através destes pesos é que a rede consegue determinar o relacionamento entre as entradas e saídas da rede. Como pode ser observado no arquivo exemplo, tem-se uma rede neural formada por 4 camadas:

1. Camada de entrada [Neurônios 1 - 4]
2. Camada de Escondida 1 [Neurônios 5 - 6]
3. Camada de Escondida 2 [Neurônios 7 - 8]
4. Camada de Saída [Neurônios 9 - 10]

A rede possui 10 neurônios e algumas Conexões: 16 conexões = 8 (camada 1-2) + 4 (camada 2-3) + 4 (camada 3-4). Como pode ser observado na figura 4.3.7, o padrão de entrada [1,1,1,1] fornecido à camada de entrada produz o vetor de saída [0.990,0.992] como desejado, uma vez que os pesos e *bias* foram determinados corretamente.

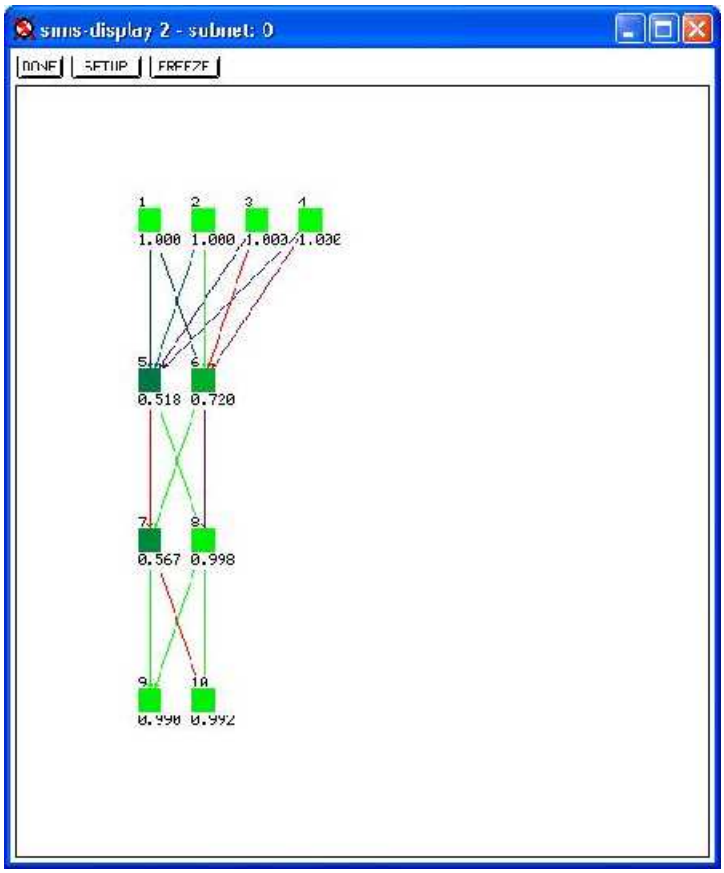

Figura 4.3.6 – Topologia de uma rede neural com respectivas ativações.

4.4 Classificação Unidimensional: 1 propriedade Física

A classificação unidimensional é o modelo proposto de classificação que considera apenas uma única propriedade física de uma dada espécie de fruto, a classificação ocorre através da comparação do valor da propriedade com um conjunto de faixa de valores denominada classe. Na figura 4.4.1 pode-se observar a existência de 4 classes de separação baseadas na propriedade física Peso de um dado fruto, cada classe define uma faixa de valores de Peso, sendo importante observar que os valores de pesos podem ser considerados vetores dentro de um espaço vetorial linear. O vetor Peso (canal de entrada - propriedade Peso) correspondente à

faixa de valores 180 a 199 é mapeado para a Classe A de classificação, da mesma forma os outros vetores são mapeados para as respectivas classes.

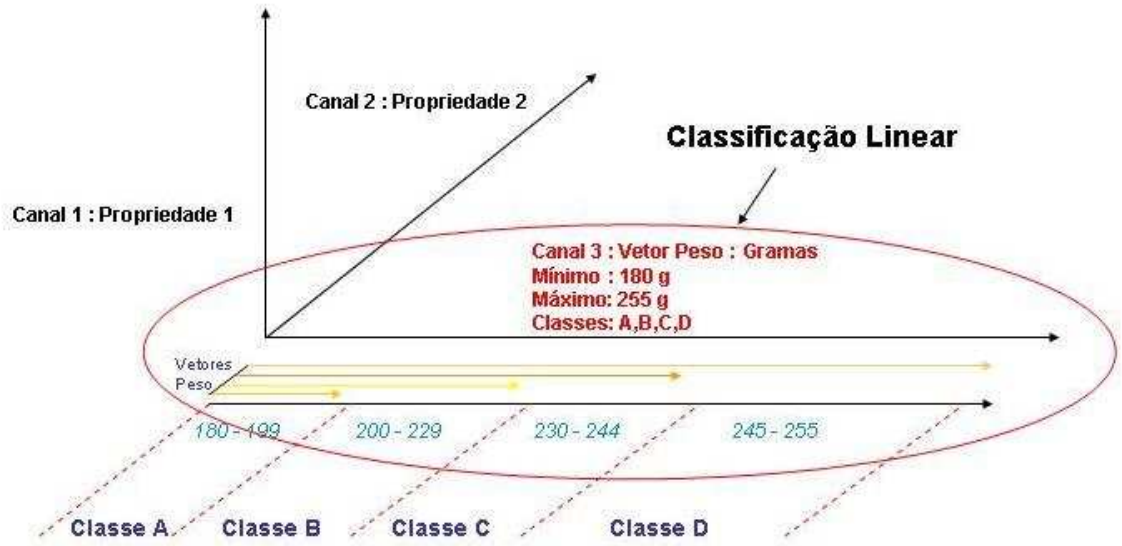

Figura 4.4.1 Projeção de um vetor: Cenário Unidimensional

Em máquinas comerciais de classificação de produtos agrícola, a classificação de frutos é baseada na análise de somente uma única propriedade física, a classificação pode ser feita com base na propriedade física Peso, Diâmetro, Cor, ou qualquer outra propriedade física do fruto. Além desta classificação primária, uma classificação humana secundária, poderá ser necessária dependendo do padrão de qualidade desejado. As figuras 4.4.2 e 4.4.3 apresentam a foto e esquema de máquinas de classificação comerciais, respectivamente, onde a classificação segue de forma serializada, os frutos percorrem um fluxo de separação seqüencial para a classificação do fruto. É possível observar que não existe paralelismo na análise das propriedades físicas, mesmo que ocorra a análise de duas ou mais propriedades físicas, não é considerada a interatividade das mesmas no processo classificatório.

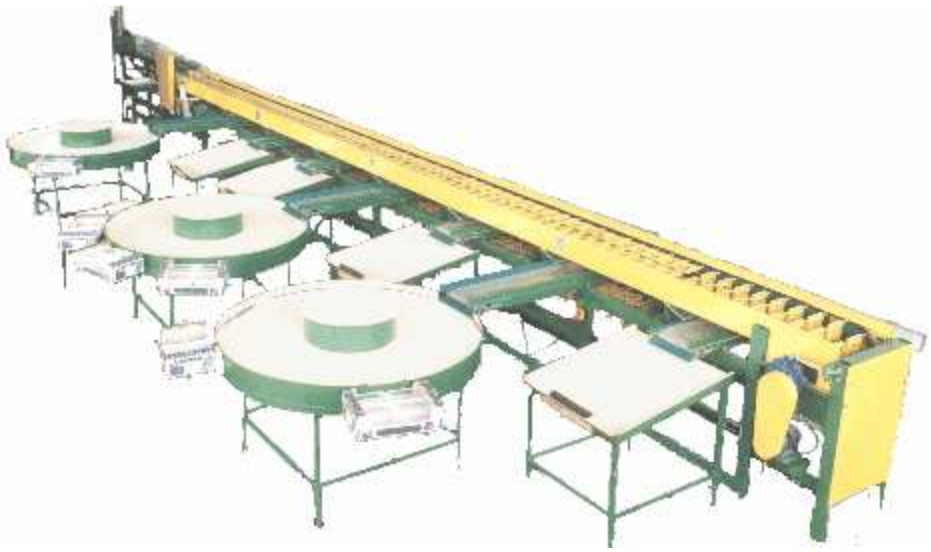

Figura 4.4.2 Máquina de classificação serializada (“Eshet Eilon Agro-Industrial Systems - Israel”)

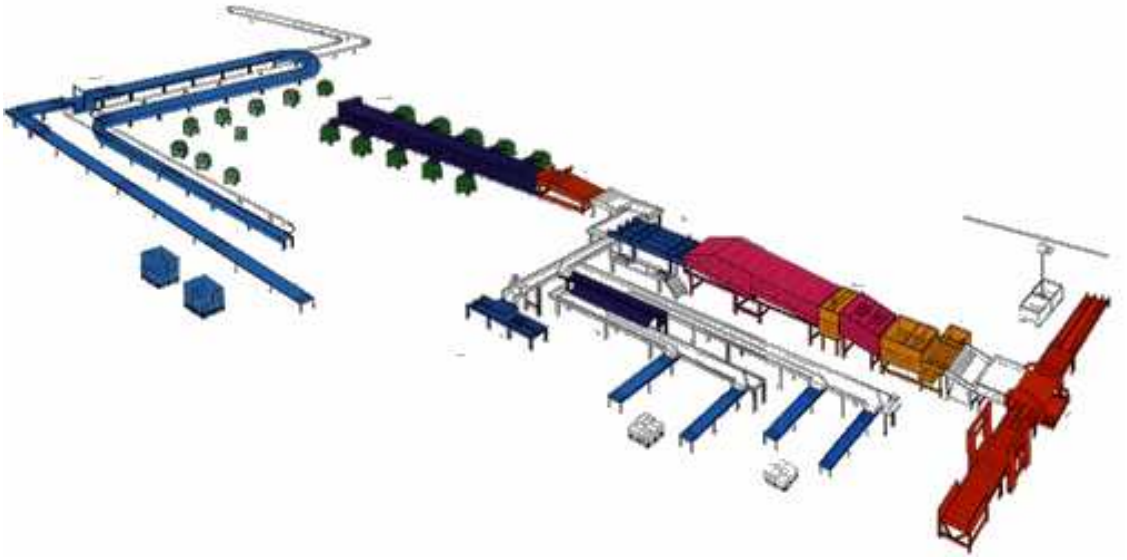

Figura 4.4.3 Layout de uma planta de classificação (“Eshet Eilon Agro-Industrial Systems - Israel”)

A forma mais comum de classificação é através de dispositivos mecânicos que fazem a separação segundo o parâmetro físico de classificação projetado, além da forma mecânica, dispositivos eletrônicos podem fazer a classificação. A propriedade física pode ser obtida através de um sensor eletro-mecânico (transdutor), que repassa o sinal elétrico na forma de voltagem

(sinal analógico) para um conversor A/D, que transforma o sinal de entrada em um sinal digital de saída.

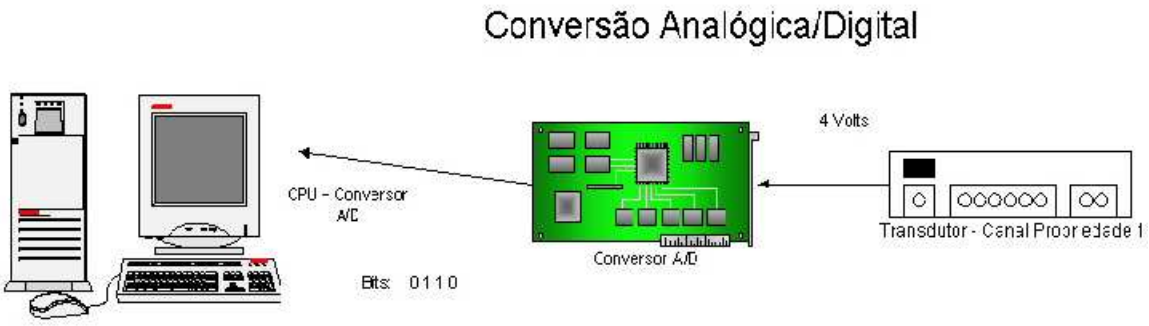

Figura 4.4.4 Fluxo de aquisição de dados

Para facilitar o entendimento dos vários cenários fica definido que as propriedades físicas sejam mapeadas para vetores de apenas 4 bits, desta forma a rede neural terá apenas 4 bits de entrada, cada bit de entrada correspondendo à um neurônio de entrada. Este modelo genérico é extensível a um número infinito de propriedades físicas de entrada e tamanho de vetores de entrada (número de bits), como será apresentado no capítulo 5. É possível fazer a classificação simultânea/paralela de frutos, analisando mais de uma propriedade física do fruto. A título de exemplo inicial será usada a propriedade física Peso como parâmetro de classificação da rede exemplo. O uso de 4 bits de entrada implica em 16 intervalos de resolução, ou seja, 2^4. Ao invés do uso de 4 bits de entrada poderia ser usados 8 bits de entrada para um mesmo parâmetro o que significaria 2^8 resultando em 256 intervalos para uso na classificação, ou seja, seria usada uma resolução ainda melhor, porém a rede teria um esforço maior de processamento para descobrir o relacionamento para tal resolução.

A resolução pode ser definida como o menor intervalo obtido por um conversor Analógico/Digital sendo este resultado função do número de bits do conversor A/D. Quanto maior o número de bits melhor será a resolução (ZAMBALDE, 1991).

A resolução do exemplo proposto pode ser calculada através da diferença entre o valor máximo (L_{max}) e o valor mínimo (L_{min}) da faixa de valores e a divisão desta diferença pelo número de intervalos permitido pelo vetor de entrada:

$$\text{Resolução} = (L_{max} - L_{min}) / 2^{\text{Número de Bits}} \qquad (4.4.1)$$

Desta forma para a faixa de valores do exemplo tem-se:

Valor Máximo = 255 gramas

Valor Mínimo = 180 gramas

Número de Bits = 4

Resolução = $(255 - 180) / 2^4$

Resolução ± 5 gramas

Fazendo o mapeamento das classes de valores com suas respectivas entradas no formato binário obtêm-se a tabela 4.4.1.

Tabela 4.4.1 Classificação de um Vetor linear de 4 bits

4 Bits	Peso	Classes
0000	180	A
0001	185	Classe 1
0010	190	**Binário: 00**
0011	195	
0100	200	
0101	205	B
0110	210	Classe 2
0111	215	**Binário: 10**
1000	220	
1001	225	
1010	230	C
1011	235	Classe 3
1100	240	**Binário: 01**
1101	245	D
1110	250	Classe 4
1111	255	**Binário: 11**

Como pode ser observado na tabela 4.4.1, cada valor analógico da propriedade física Peso deverá ser mapeado a um respectivo vetor binário, que por sua vez também é relacionado a uma classe com saída binária. O uso de um vetor binário de saída constituído de 2 bits é necessário devido ao uso de 4 classes distintas de peso.

O mapeamento binário dos vetores de pesos pode ser expresso graficamente segundo a figura 4.4.5.

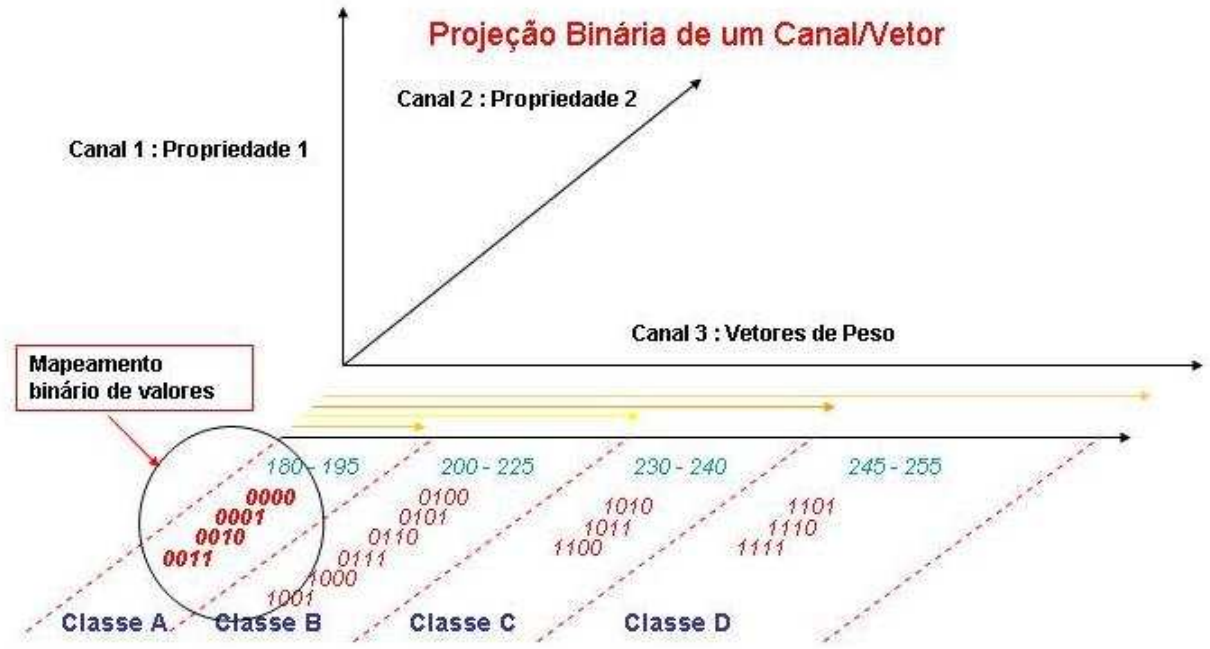

Figura 4.4.5 Mapeamento Binário de Valores

Como pode ser observado cada vetor de peso pode ser expresso tanto no formato analógico como no formato binário. A expressão do vetor de peso no formato binário será utilizado nos vários cenários de classificação deste trabalho. Os conversores A/D, responsáveis pela quantificação dos sinais das propriedades dos frutos, apresentam os sinais de entrada no formato digital, exatamente no mesmo formato que o modelo proposto neste trabalho.

BLINDER (1994), define Topologia como sendo a decisão de quantas camadas ocultas serão necessárias para resolver a tarefa e quantos elementos cada camada deverá conter.

A seguir tem-se os passos necessários para a implementação do processo classificatório utilizando redes neurais artificiais:

Passo 1 - Coleta de dados: os dados da tabela 4.4.1 apresentam o domínio do problema para uma entrada de um vetor Peso utilizando-se 4 bits, obtendo desta forma 16 intervalos de resolução para a faixa de pesos.

Passo 2 - **Separação em Conjuntos:** a determinação das classes de pesos representam a saída destino da rede, sendo que o número de neurônios de saída é função do número de classes.

Passo 3 - **Configuração da rede:** nesta parte é necessário o estudo da topologia da rede, determinando informações tais como número de neurônios nas camadas de entrada, escondida e saída, tipo de rede, algoritmo de aprendizado entre outros parâmetros iniciais. A topologia influência diretamente na convergência da rede.

Passo 4 - **Treinamento:** uma vez que o passo 3 tenha sido determinado, deve-se perseguir então a convergência da rede para a tolerância de erro desejada no caso da rede escolhida.

Passo 5 - **Teste:** Tolerância define um valor aceitável para a diferença entre o valor obtido e o desejado da saída da rede. Uma vez que os erros de saída sejam inferiores aos definidos na tolerância, a rede é considerada treinada e pronta para ser testada.

Passo 6 - **Integração:** a implementação efetiva da rede em ambiente de produção.

Uma vez que os passos 1 e 2, fases de coleta e separação de conjunto, respectivamente, tenham sido realizadas é necessário então um estudo da configuração da rede. A figura 4.4.6 apresenta o fluxo de implementação de uma rede neural, ou mais precisamente, define o passo 3, estudo da configuração da rede.

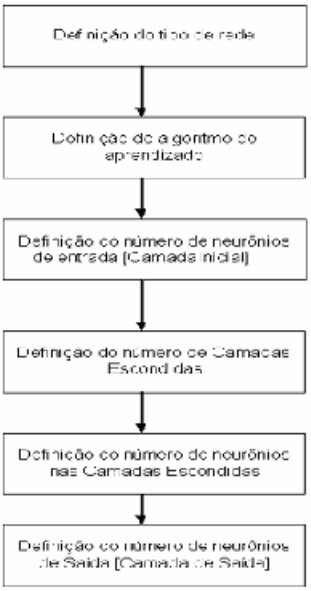

Figura 4.4.6 Fluxo de implementação de uma rede neural

Como pode ser observado na tabela 4.4.2 vários são os parâmetros que podem ser alterados na busca de uma rede neural que consiga descrever o relacionamento entre um conjunto de dados de entrada e de saída. A convergência de uma rede neural, que descreve com sucesso o relacionamento do conjunto de dados de entrada e saída estudados, é o processo de determinação dos parâmetros corretos de configuração da rede neural, por exemplo, quantas camadas escondidas e neurônios são necessários para descrever este relacionamento.

Tabela 4.4.2 Variáveis de uma rede neural

Variações de Configuração	
1	Número de Neurônios da Camada Inicial
2	Número de Neurônios da Camada de Saída
3	Número de Neurônios da Camada Escondida
4	Tipo de rede
5	Algoritmo de aprendizado

Durante a escolha da topologia da rede para o cenário unidimensional foram feitos inúmeros testes, variando principalmente o número de camadas intermediárias e neurônios destas. A convergência ocorreu para as redes com duas camadas escondidas. A tabela 4.4.3 representa a topologia final apresentada para o cenário unidimensional com uso de 4 bits.

Tabela 4.4.3 Exemplo de Cenário Unidimensional

Informações da Rede : [Cenário Unidimensional]	
Camada de Entrada [Neurônios]	4
Camada de Escondida 1 [Neurônios]	2
Camada de Escondida 2 [Neurônios]	2
Camada de Saída [Neurônios]	2
Função de Aprendizado	Std_Backpropagation
Função de Atualização	Topological_Order
Função de Incialização	

Apenas para exemplificação tem-se abaixo a topologia de uma rede parcialmente treinada. A camada de entrada é composta de 4 bits, a entrada binária [1,1,0,1], formada pelos neurônios 1,2,3,4. Duas camadas intermediárias, ou também chamadas escondidas, contém 2 neurônios em cada uma, representadas pelos neurônios 5,6 e 7,8, nas camadas escondida 1 e 2, respectivamente. A camada de saída constituída pelos neurônios 9 e 10, oferece como saída os valores 0,979 e 0.980, indicando que para uma tolerância de erro igual a 0.01, necessita de um tempo maior ou um número de ciclos de convergência maior para diminuir o erro de saída.

Tabela 4.4.4 Saída de uma rede neural

Neurônio	Camada de Entrada	Camada Escondida 1	Camada Escondida 2	Camada de Saída
1	1			
2	1			
3	0			
4	1			
5		0		
6		0.818		
7			0.500	
8			04.24	
9				0.979
10				0.980

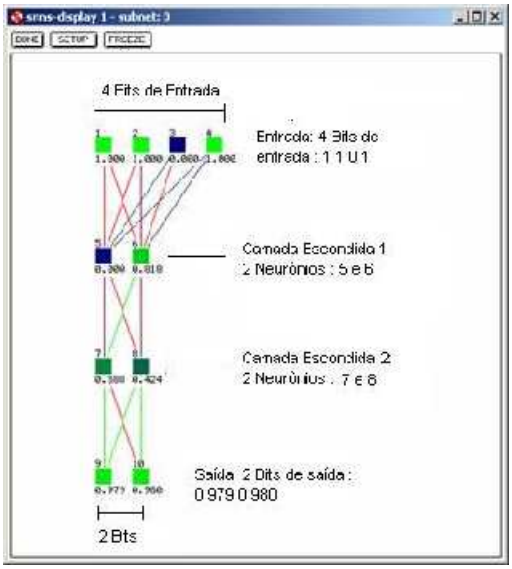

Figura 4.4.7 Topologia de um modelo neural - 4 camadas

4.5 Classificação Bidimensional: 2 propriedades Físicas

4.5.1 Classificação de Padrões e Espaço de Entradas

Para explicar o conceito de classificação de padrões será utilizado o neurônio representado na figura 4.5.1 proposto por McCULLOCH e PITTS (1943), contendo duas entradas de valores booleanos '0' e '1' somente.

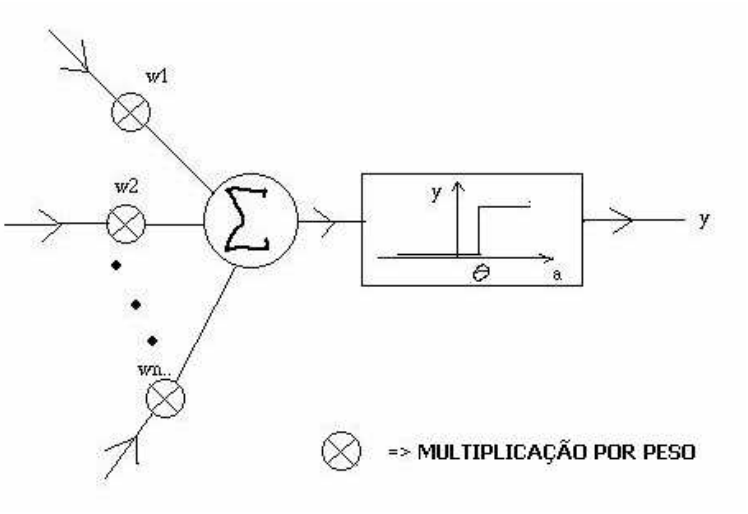

Figura 4.5.1. – Representação de um neurônio de McCULLOCH e PITTS.

A tabela 4.5.1 de resposta possui duas entradas x_1 e x_2 com 4 valores possíveis e uma saída com 2 valores possíveis.

Tabela 4.5.1. Porta "*AND*".

	x_1	x_2	Saída
1	0	0	0
2	0	1	0
3	1	0	0
4	1	1	1

O neurônio pode considerar a classificação de seus padrões de entrada em duas classes: aqueles que tem saída '1' e aqueles que tem saída '0'. Cada padrão de entrada possui dois componentes. Estes padrões podem ser expressos em um espaço de duas dimensões, segundo a figura 4.5.2.

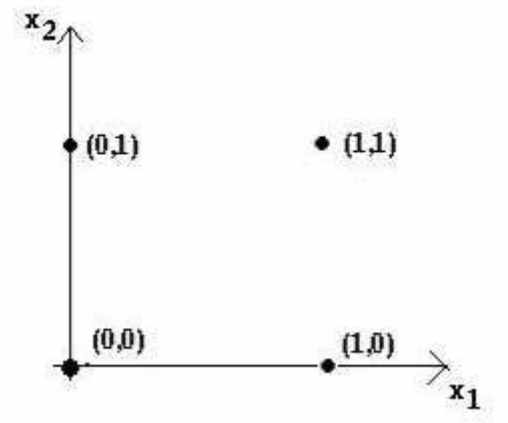

Figura 4.5.2. Porta "*AND*" - E

O espaço no qual estas entradas residem é referido como espaço de padrões. Cada padrão determina um ponto no plano através do uso dos valores de suas componentes como coordenadas no espaço. Genericamente, para *n*-entradas, o espaço de padrões será *n*-dimensional.

O espaço de padrões da figura 4.5.3 pode ser linearmente separado através de uma linha de decisão que corta o conjunto de padrões separando os padrões 1,2,3 na classe 0 e o padrão 1 na classe 1.

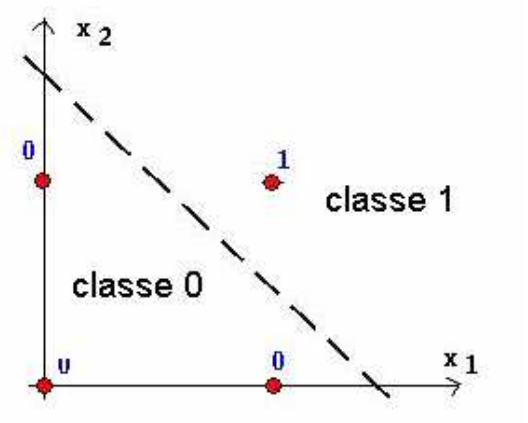

Figura 4.5.3. – Separação linear do Espaço de Padrões

4.5.2 Vetores

Vetores são usualmente introduzidos como representações de quantidades que tem magnitude e direção. A velocidade de um carro é definida pela velocidade e direção. No papel pode-se desenhar uma flecha na qual o comprimento é proporcional à velocidade do carro e a direção é a mesma do carro.

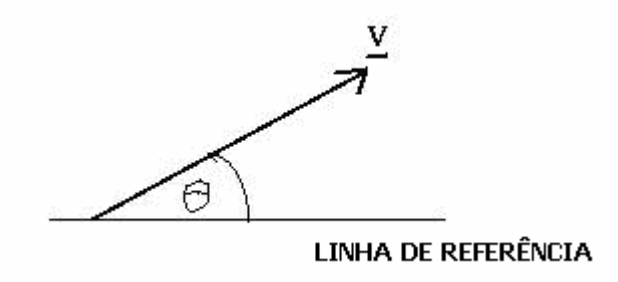

Figura 4.5.2.1 Vetor 2D

De forma a generalizar dimensões maiores e relacionar ao padrão de espaço os vetores podem ser colocados dentro de um sistema de coordenadas cartesianas, onde o comprimento dos vetores é colocado em dois eixos perpendiculares.

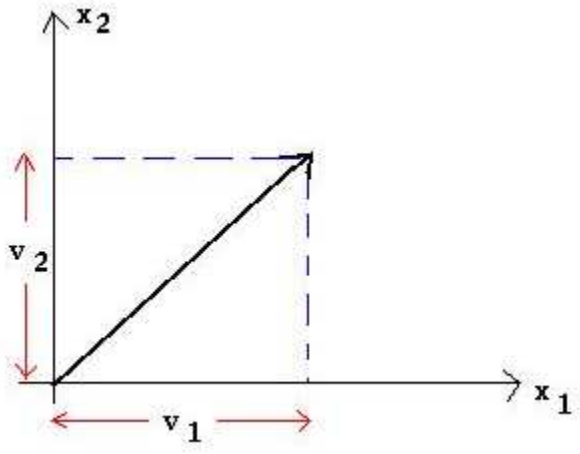

Figura 4.5.4 Vetor projetado em eixos cartesianos

O vetor pode ser descrito por um par de números. Estes números são as componentes escolhidas dentro do sistema de coordenadas. Uma vez que as componentes completamente determinam o vetor pode-se pensar no vetor com um par de valores de componentes. O vetor é uma lista de números ordenados. É importante notar que a ordem dos vetores é importante pois (1,3) e (3,1) são vetores diferentes.

Esta definição pode ser generalizada para mais do que 2 dimensões. Um vetor *n* - dimensional é simplesmente uma lista ordenada de *n* números.

Nos tópico anterior foi apresentado o cenário de classificação linear, onde apenas um parâmetro físico foi analisado. Neste tópico será apresentado o cenário bidimensional, o qual representa a união de 2 instâncias lineares, ou mais precisamente da somatória vetorial de 2 componentes vetoriais, representando cada um deles uma propriedades física diferente.

A representação da figura 4.5.5 ilustra que a somatória dos vetores de diâmetro e peso, convergem para áreas de classificação de interesse. Estas áreas devem ser definidas segundo cada espécie de fruto.

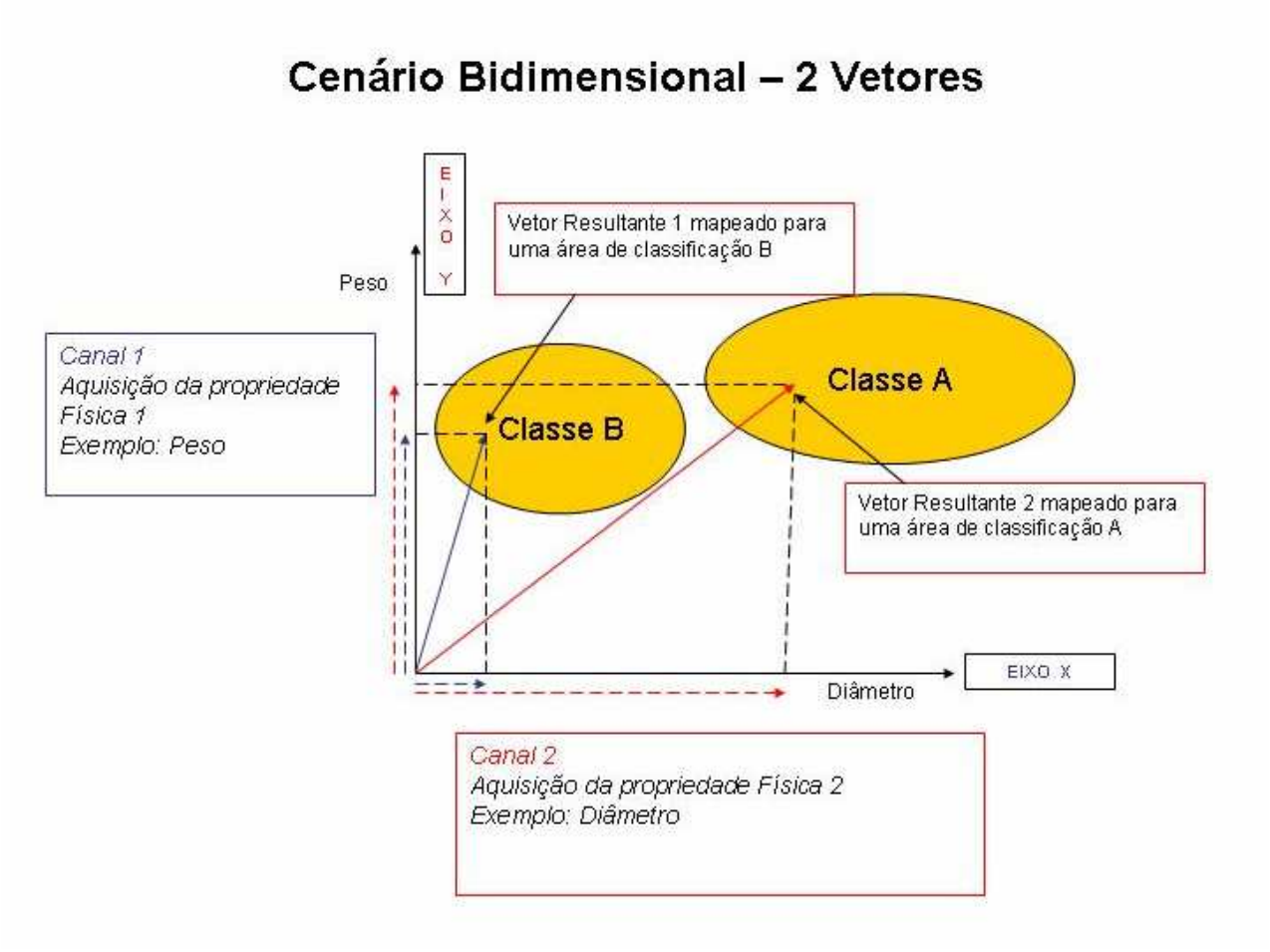

Figura 4.5.5. Representação Vetorial Cenário Bidimensional.

Tabela 4.5.2 Vetores de Peso e Diâmetro

	4 Bits	Peso - [g]	Classes	4 Bits	Diâmetro - [cm]	Classes
1	0000	180		0000	8	
2	0001	185	A	0001	8.1	A
3	0010	190		0010	8.2	
4	0011	195	0 - 0	0011	8.3	0 - 0
5	0100	200		0100	8.4	
6	0101	205		0101	8.5	
7	0110	210	B	0110	8.6	B
8	0111	215		0111	8.7	
9	1000	220	1 - 0	1000	8.8	1 - 0
10	1001	225		1001	8.9	
11	1010	230	C	1010	9	C
12	1011	235		1011	9.1	0 - 1
13	1100	240	0 - 1	1100	9.2	
14	1101	245	D	1101	9.3	D
15	1110	250		1110	9.4	
16	1111	255	1 - 1	1111	9.5	1 - 1

Como pode ser observado na Tabela 4.5.2, cada propriedade linear, possui uma determinada faixa de valores para a representação das classes. Quando são colocadas as duas representações simultaneamente e no formato vetorial, tem-se uma representação igual à figura 4.5.5. Esta representação foi chamada cenário bidimensional onde duas propriedades físicas/canais de entrada, são analisadas simultaneamente.

No caso do cenário bidimensional foi utilizada a matriz da figura 4.5.6 como espaço amostral/padrões para o treinamento e testes das redes neurais bidimensionais, onde as propriedades Diâmetro e Peso correspondem ao eixo **X** e **Y**, respectivamente. Em cada eixo existe a projeção do valor binário e decimal de cada vetor, onde a cor nas linhas e colunas destas projeções representam a classificação em termos lineares.

O modelo de mapeamento da figura 4.5.6 permite que exista a combinação de duas propriedades simultaneamente ou também computacionalmente paralelismo de informações. Baseado na análise real destas duas propriedades para um dado fruto pode-se chegar ao mapeamento da figura 4.5.6, este trabalho propõe um modelo genérico de entrada de dados para

as redes neurais, qualquer fruto descrito segundo o tipo de mapeamento da figura 4.5.6 permite que seja feita a classificação utilizando-se redes neurais.

O levantamento das informações de duas propriedades relevantes, neste caso, Peso e Diâmetro, criarão um mapeamento semelhante ao da figura 4.5.6. A análise simultânea destas duas propriedades determina as áreas de classificação, que nada mais são do que o agrupamento dos vetores de mesma saída. Como exemplo no mapeamento figura 4.5.6 existem 4 classes, representadas no formato binário: Verde: [00], Azul Escuro: [01], Vermelha: [10], Azul Claro: [11].

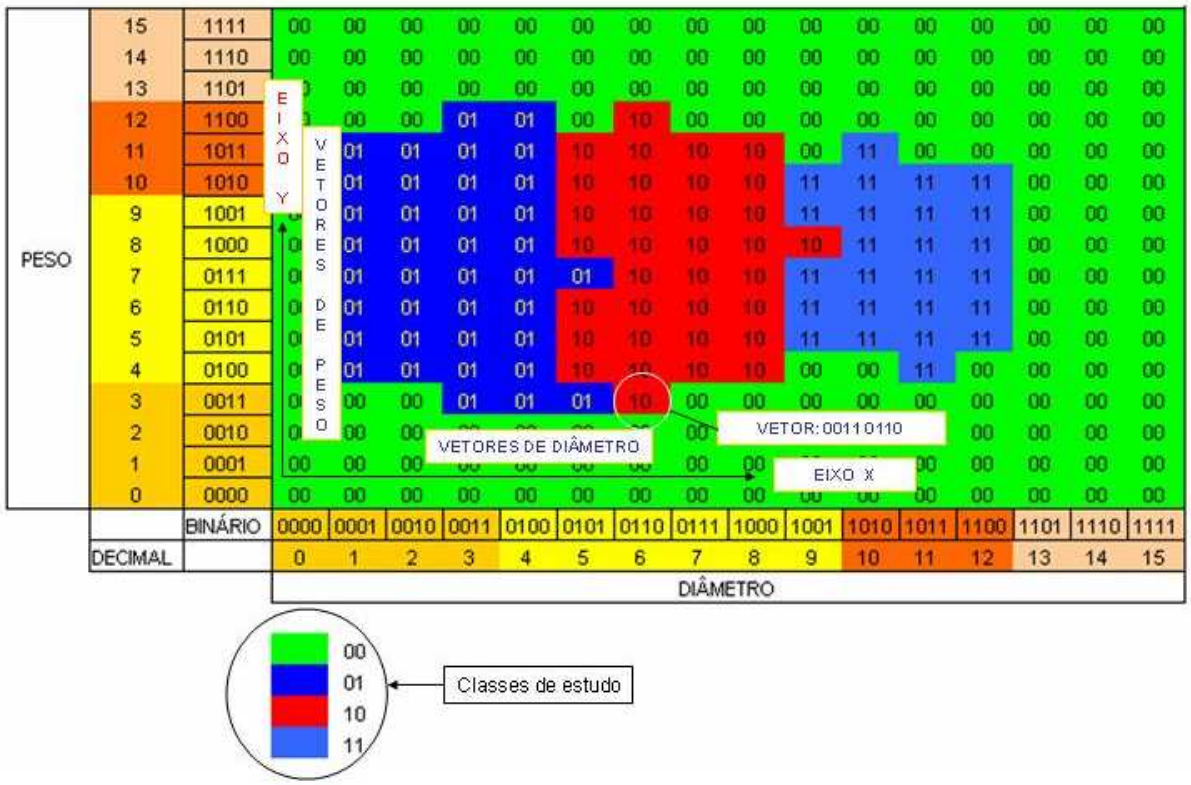

Figura 4.5.6 Espaço Amostral/ Padrões bidimensional

A soma dos dois vetores lineares cria uma matriz bidimensional com definição de áreas de classificação. O vetor de entrada da rede será uma representação de um vetor de 8 Bits formado por 2 vetores individuais de 4 Bits, cada vetor representando uma propriedade física diferente:

Os primeiros 4 bits correspondem à propriedade física Peso – [0011] e os últimos 4 bits correspondem à propriedade física Diâmetro – [0110]. Desta forma o vetor bidimensional resultante é o vetor [0011 0110]. O vetor resultante está mapeado para a classe 10.

A matriz bidimensional com o espaço de padrões proposto na figura 4.5.6 apresenta 4 classes diferentes com áreas de interesse não regulares. Como pode ser observado existe uma área de borda chamada de classe de descarte (classe verde [00]), simulando o mundo real.

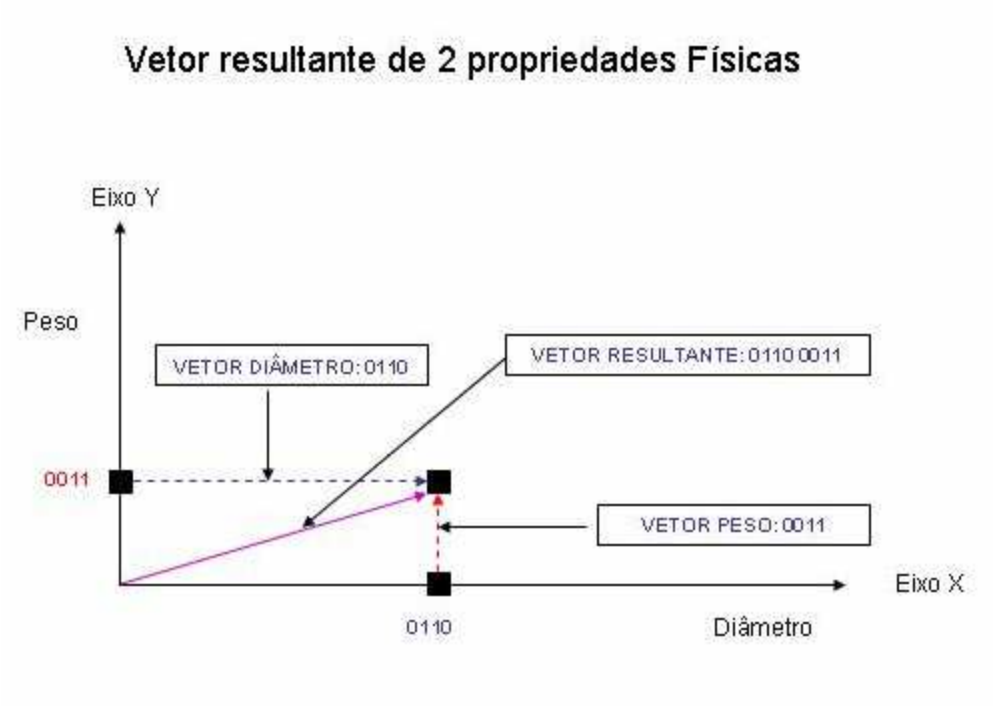

Figura 4.5.7 Vetor Resultante de 2 propriedades físicas

A representação vetorial da figura 4.5.7 pode ser expressa em termos tabulares também como demonstrado no ANEXO III, onde as entradas podem estar separadas com seus respectivos valores decimais e binários, assim como a composição dos diversos vetores.

Como pode ser observado este trabalho propõe a modularidade da solução através da composição do problema com módulos menores. Um sistema computacional é dito ser *modular* se a arquitetura puder ser separada em dois ou mais pedaços responsáveis por partes distintas e possivelmente mais simples do conjunto de entradas (SILVA, 1998).

Baseando-se no mapeamento vetorial da figura 4.5.6 foi criada uma rede neural que considera simultaneamente as propriedades físicas Peso e Diâmetro. Após inúmeros testes de convergência a topologia final adequada a resolução do problema de classificação foi uma rede com 8 neurônios de entrada, 2 camadas escondidas com 2 neurônios cada uma e uma camada de saída com 2 neurônios. Uma explicação mais profunda será apresentada no capítulo de resultados experimentais A figura 4.5.8 representa a topologia da rede bidimensional pesquisada para a classificação de frutos baseada em 2 propriedades físicas.

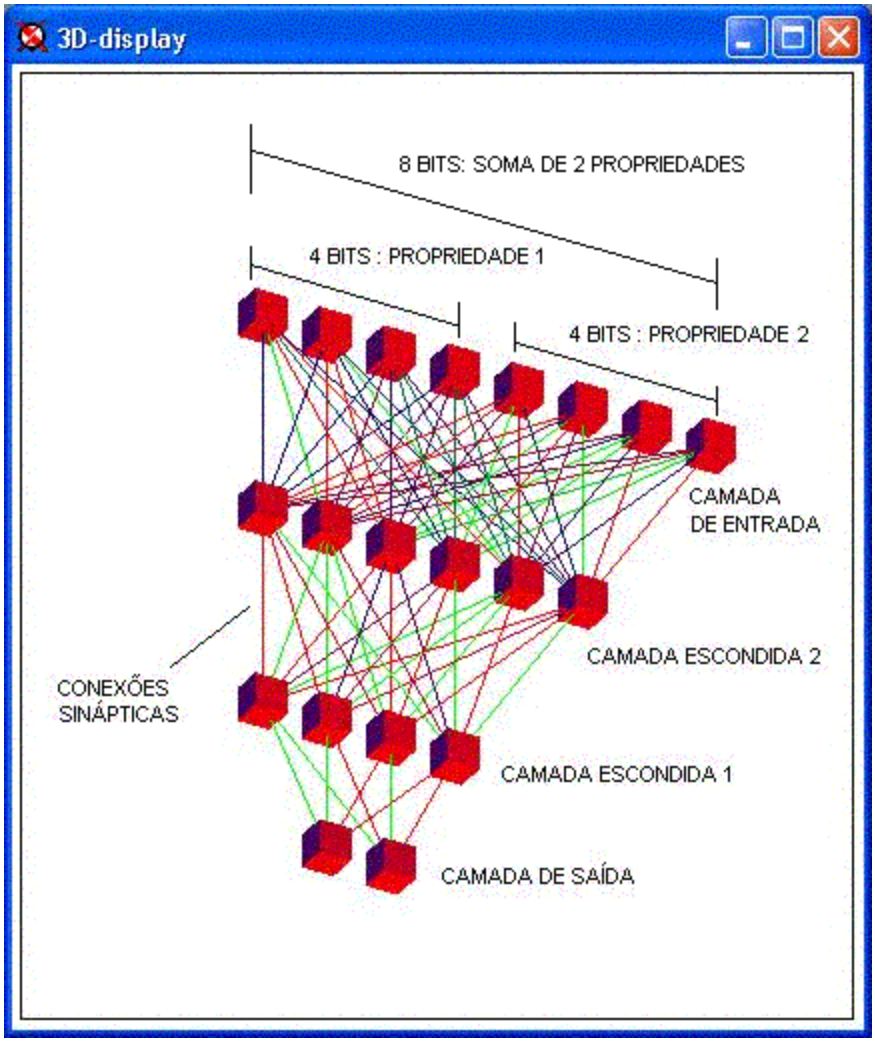

Figura 4.5.8 Representação gráfica da topologia de uma rede do cenário bidimensional

4.6 Classificação Tridimensional: 3 propriedades Físicas

O caso tridimensional representa a união vetorial de 3 instâncias lineares, neste caso o vetor tridimensional, pode ser constituído de 3 vetores de 4 bits, sendo um vetor de 4 Bits de entrada representando um parâmetro de peso (1-4), 4 bits representando um parâmetro Diâmetro (5-8) e mais 4 bits de Entrada representando um parâmetro de "*Firmness*" (9-12), totalizando 12 bits. Desta forma é possível obter a primeira camada de entrada da rede com 12 bits de entrada, tendo 16 posições de resolução para cada conjunto de 4 bits. Juntando-se os três vetores individuais obtêm-se uma matriz tridimensional com 2^{12} componentes, ou 4096 saídas de

classificação. A composição dos três vetores cria uma matriz tridimensional com definição de volumes de classificação.

Compondo um Vetor de 12 Bits como a soma de 3 de vetores de 4 bits tem-se:

Os primeiros 4 bits (1 - 4): Peso [0111]
Os intermediários 4 bits (5 - 8): Diâmetro [0100]
Os últimos 4 bits (9 - 12) : Firmness [0100]

Logo a somatória vetorial compõe um vetor de 12 bits – [0111010000100] que deverá convergir para uma classe do tipo [01].

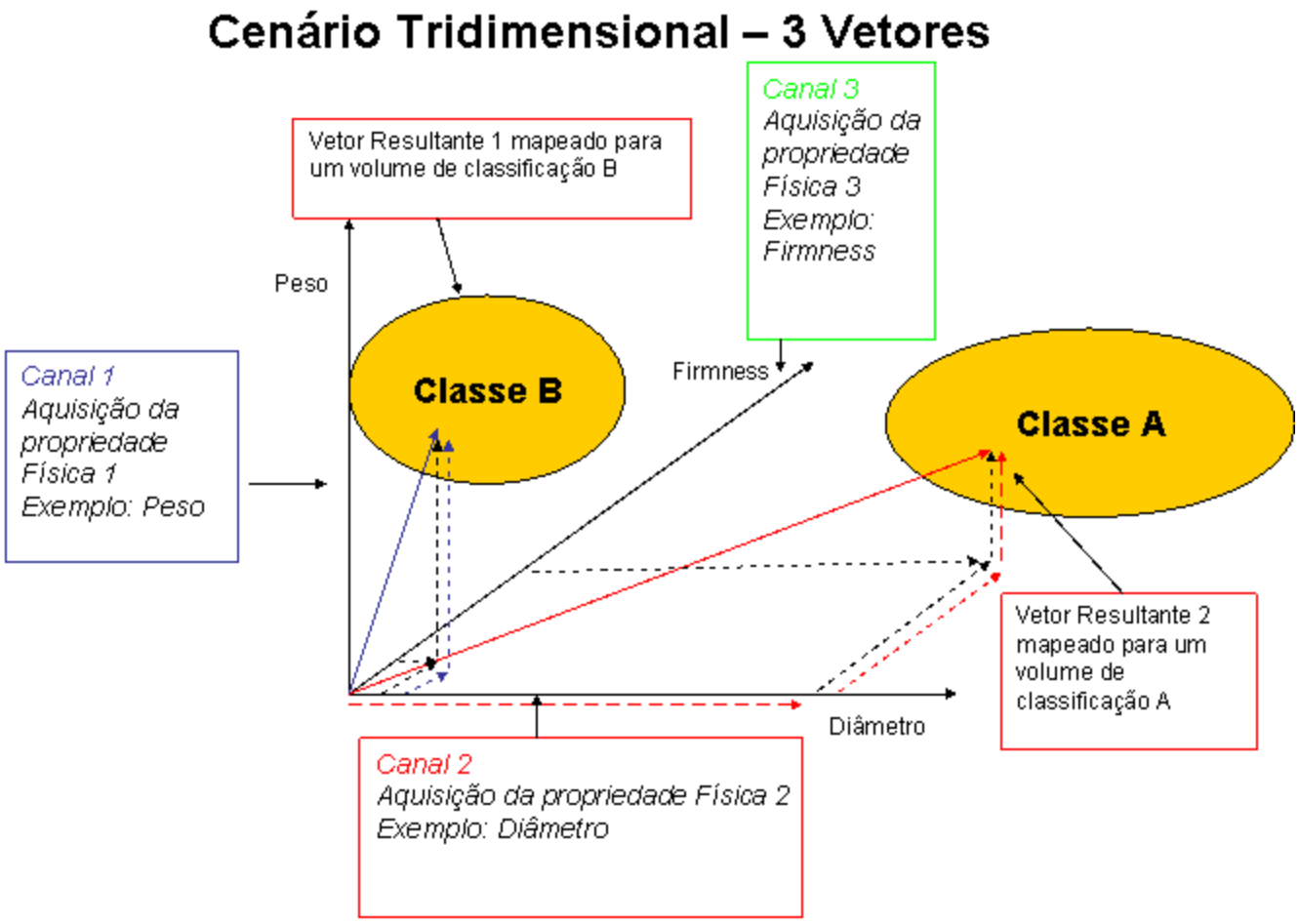

Figura 4.6.1 – Representação Vetorial Cenário Tridimensional

A representação vetorial da figura 4.6.1. descreve graficamente a somatória das componentes vetoriais individuais de cada propriedade, conduzindo a um volume no espaço de classes de interesse. Como pode ser observado o vetor resultante é composto da interatividade dos três canais de entrada, levando-se em conta a influência das três propriedades simultaneamente, Peso, Diâmetro e Firmeza ("*Firmness*").

A figura 4.6.2 apresenta o exemplo de um vetor resultante da somatória vetorial das três propriedades físicas estudadas neste cenário. O vetor resultante converge para um volume de classificação formado pelo conjunto de vetores endereçados à mesma classe. Os eixos X, Y e Z, correspondem à projeção das propriedades, Peso, Diâmetro e "*Firmness*", respectivamente. Os volumes de classes de estudo são formados com a sobreposição dos espaços de padrões de Peso e Diâmetro e somatória da componente de "*Firmness*".

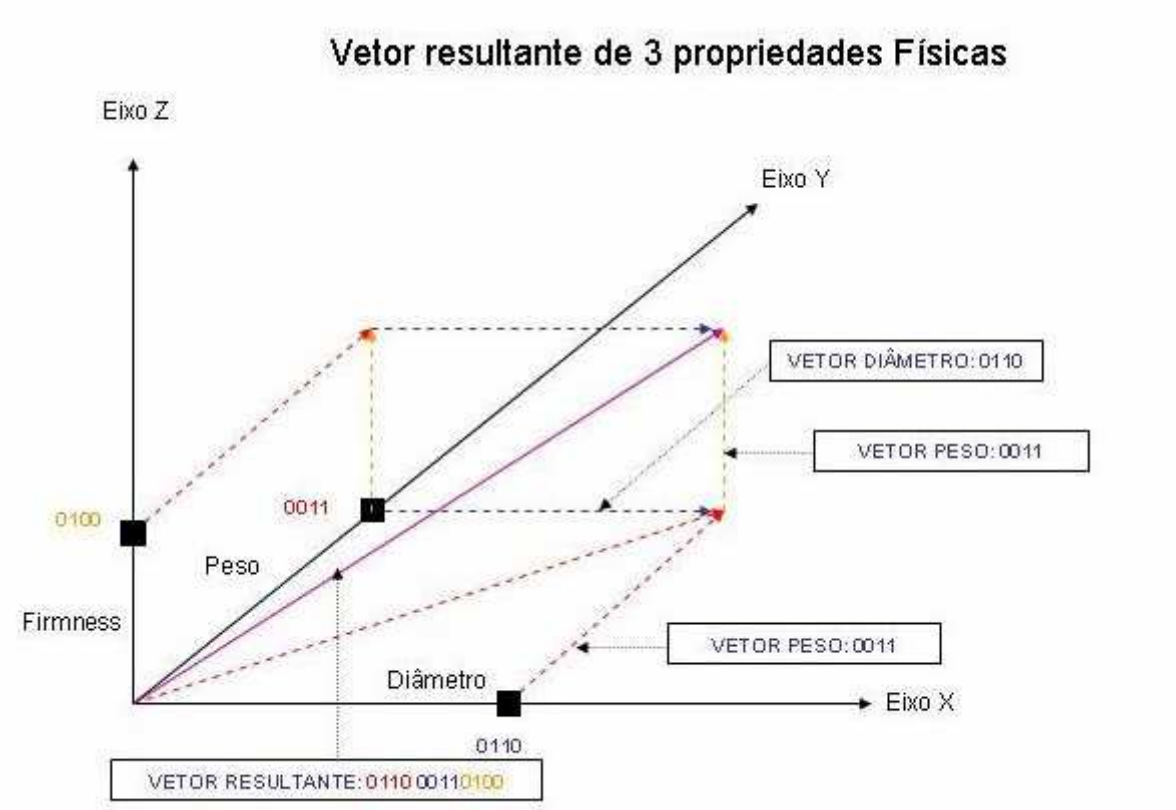

Figura 4.6.2 Vetor Resultante de 3 propriedades físicas

A figura 4.6.3 em sua primeira camada apresenta a entrada de dados da rede neural para o cenário tridimensional. Neste caso cada 4 bits correspondem a componente vetorial de uma das 3 propriedades físicas analisadas.

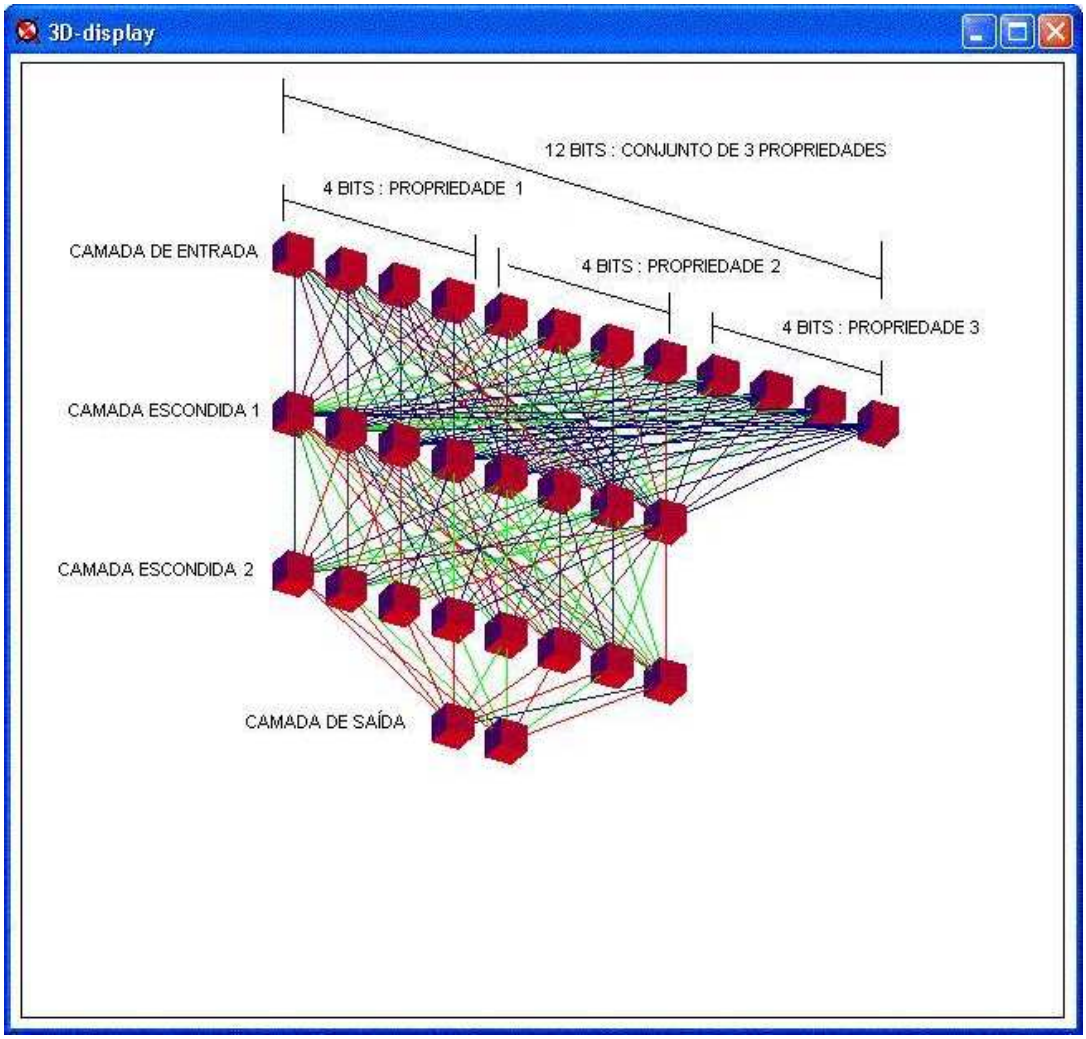

Figura 4.6.3 Topologia Cenário Tridimensional

Considerando que o novo parâmetro de classificação "*Firmness*" possui 4 bits teremos 2^4 = 16 intervalos ou 16 camadas diferentes que quando sobrepostas criam o volume de análise.

Seguem abaixo a representação gráfica das camadas de análise considerando-se as três propriedades físicas Peso, Diâmetro e "*Firmness*":

Firmness:	1																	
Camada:	0000																	
	4 BITS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
	1111	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	16
	1110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	15
	1101	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	14
	1100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	13
	1011	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	12
	1010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	11
P	1001	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	10
E	1000	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	9
S	0111	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	8
O	0110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	7
	0101	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	6
	0100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	5
	0011	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	4
	0010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	3
	0001	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	2
	0000	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	1
		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	
		DIÂMETRO																

Figura 4.6.4 Espaço Amostral/ Padrões Tridimensional: Camada 0000: Firmness 1

Firmness:	2																	
Camada:	0001																	
	4 BITS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
	1111	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	16
	1110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	15
	1101	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	14
	1100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	13
	1011	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	12
	1010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	11
P	1001	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	10
E	1000	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	9
S	0111	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	8
O	0110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	7
	0101	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	6
	0100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	5
	0011	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	4
	0010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	3
	0001	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	2
	0000	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	1
		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	
		DIÂMETRO																

Figura 4.6.5 Espaço Amostral/ Padrões Tridimensional: Camada 0001: Firmness 2

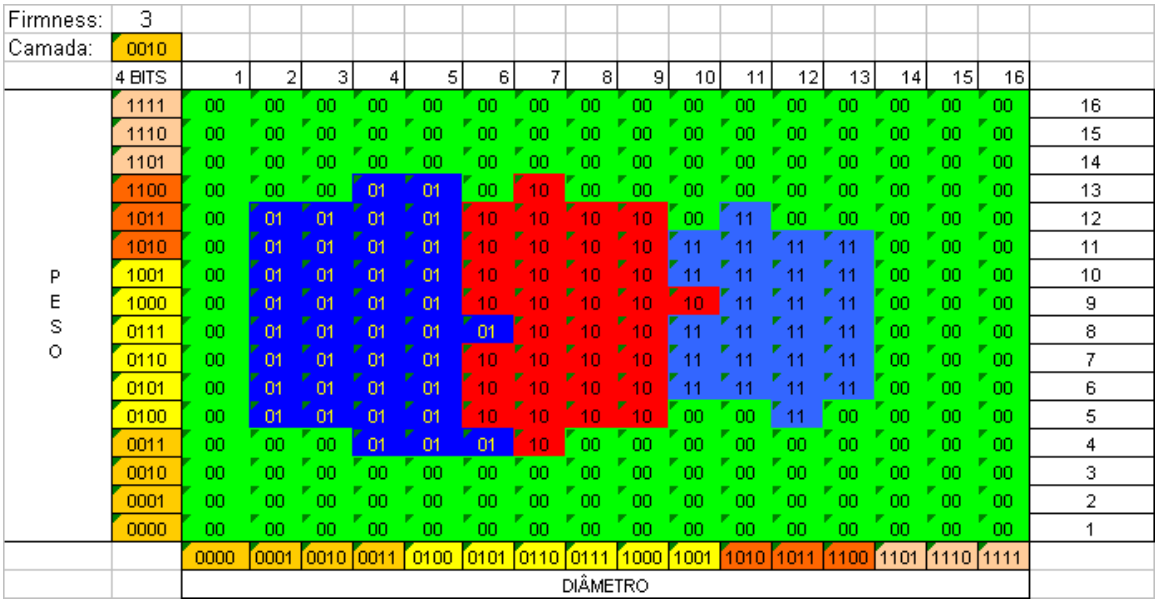

Figura 4.6.6 Espaço Amostral/ Padrões Tridimensional: Camada 0010: Firmness 3

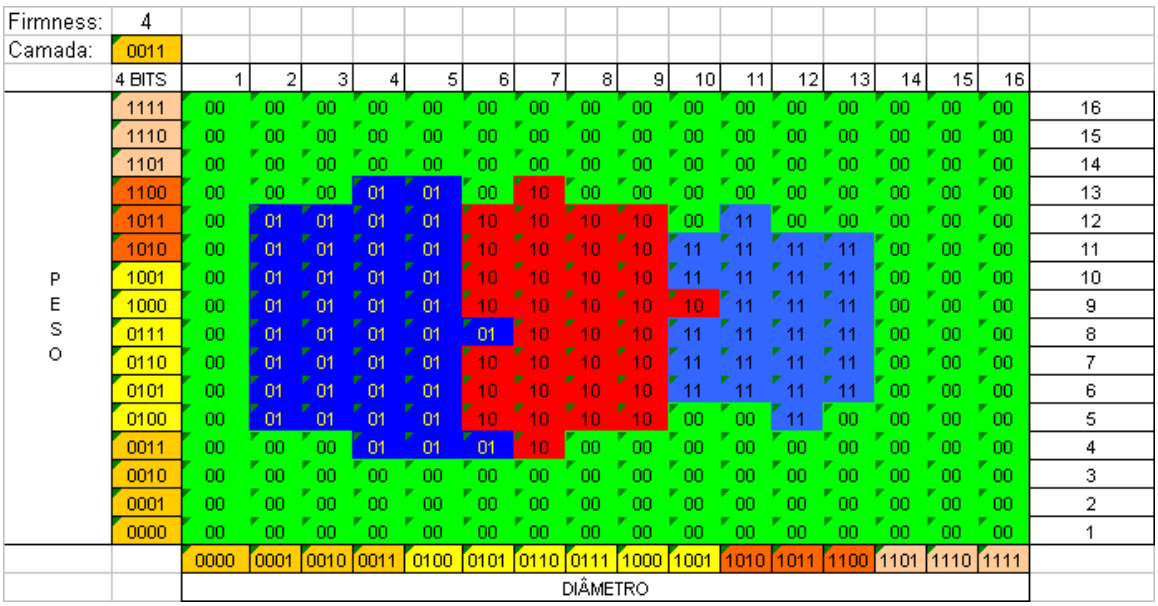

Figura 4.6.7 Espaço Amostral/ Padrões Tridimensional: Camada 0011: Firmness 4

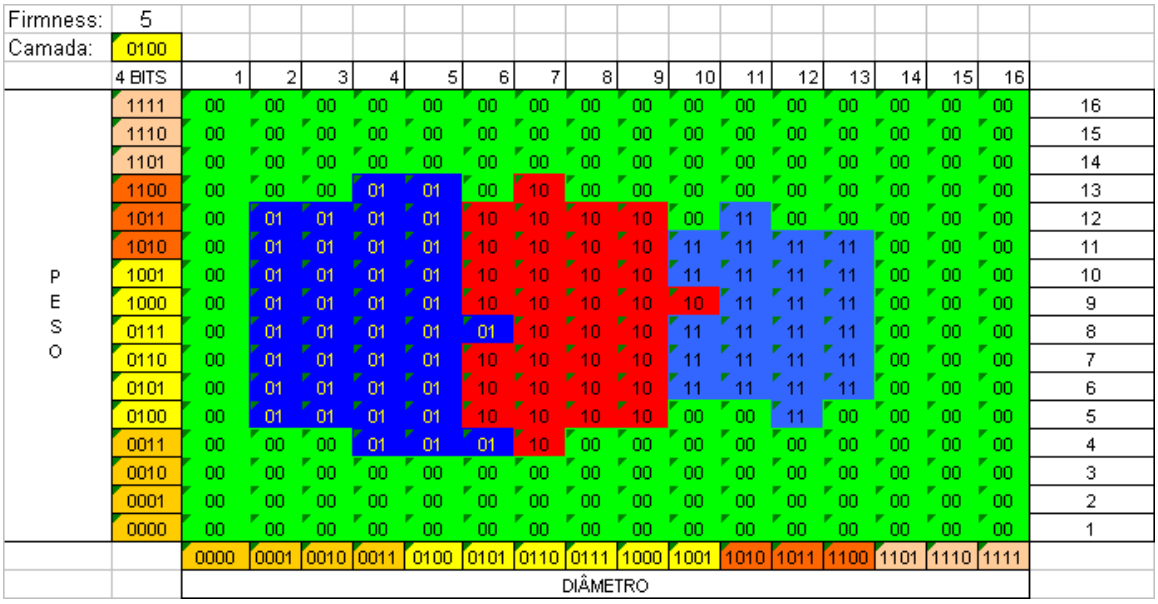

Figura 4.6.8 Espaço Amostral/ Padrões Tridimensional: Camada 0100: Firmness 5

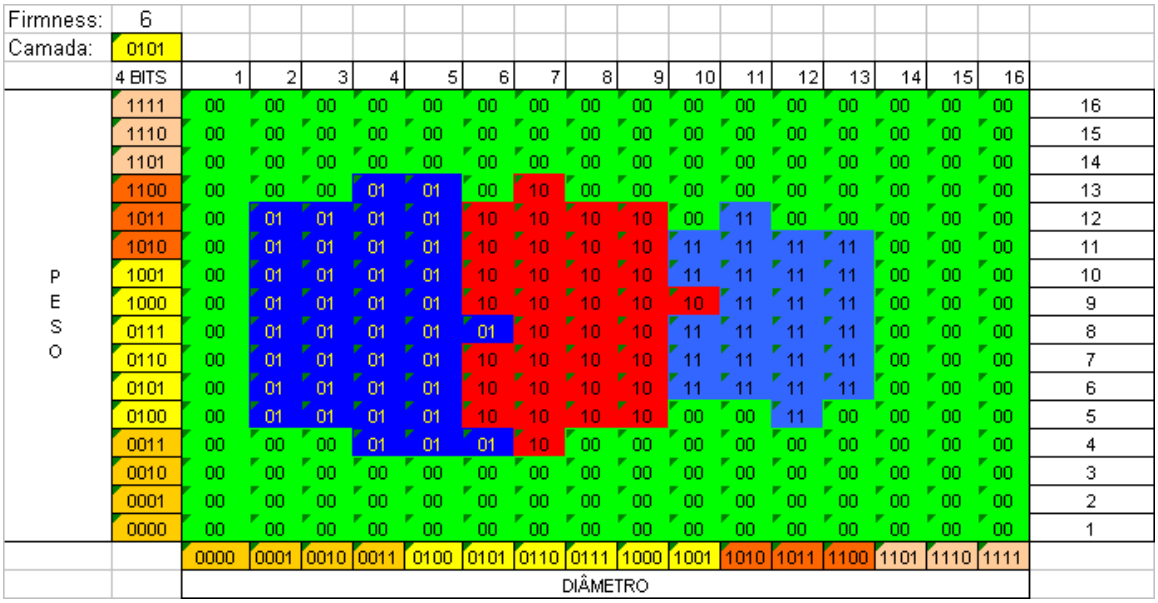

Figura 4.6.9 Espaço Amostral/ Padrões Tridimensional: Camada 0101: Firmness 6

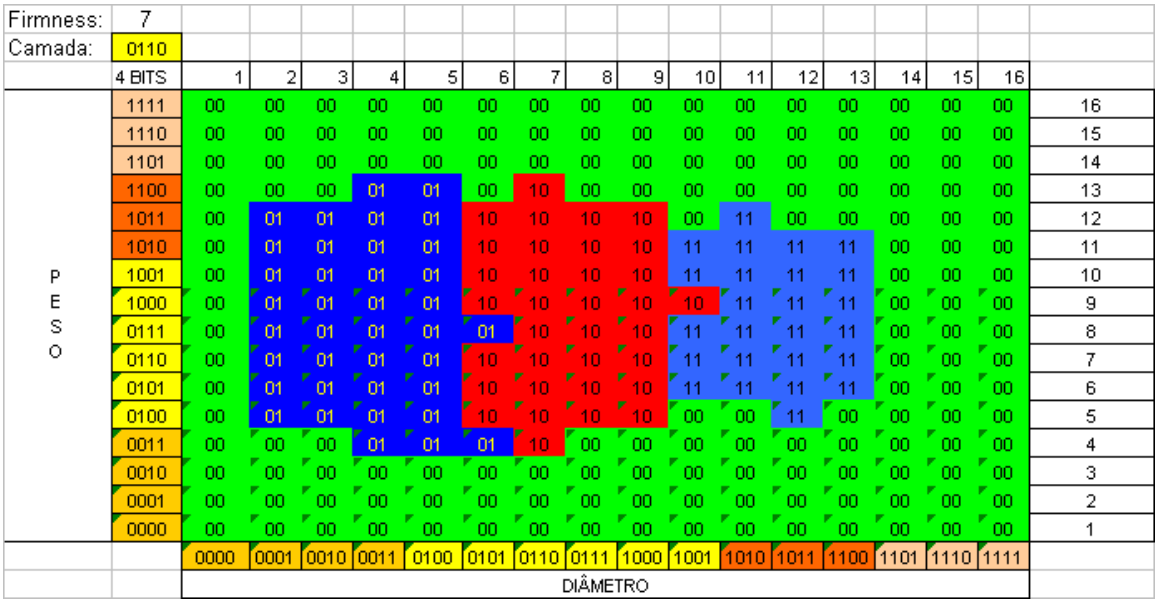

Figura 4.6.10 Espaço Amostral/ Padrões Tridimensional: Camada 0110: Firmness 7

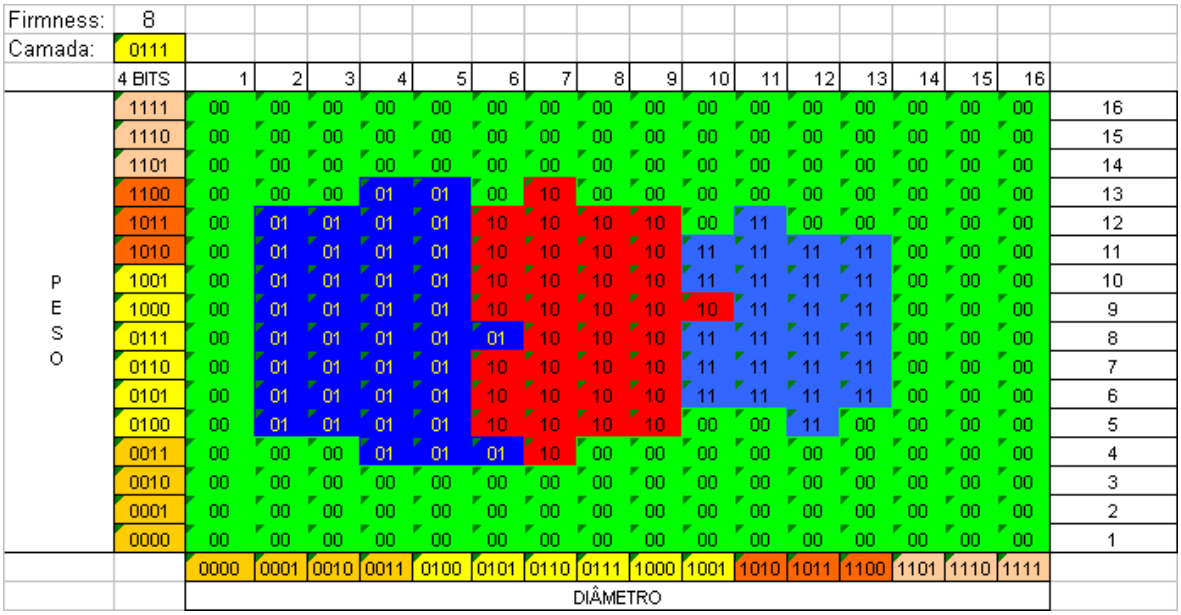

Figura 4.6.11 Espaço Amostral/ Padrões Tridimensional: Camada 0111: Firmness 8

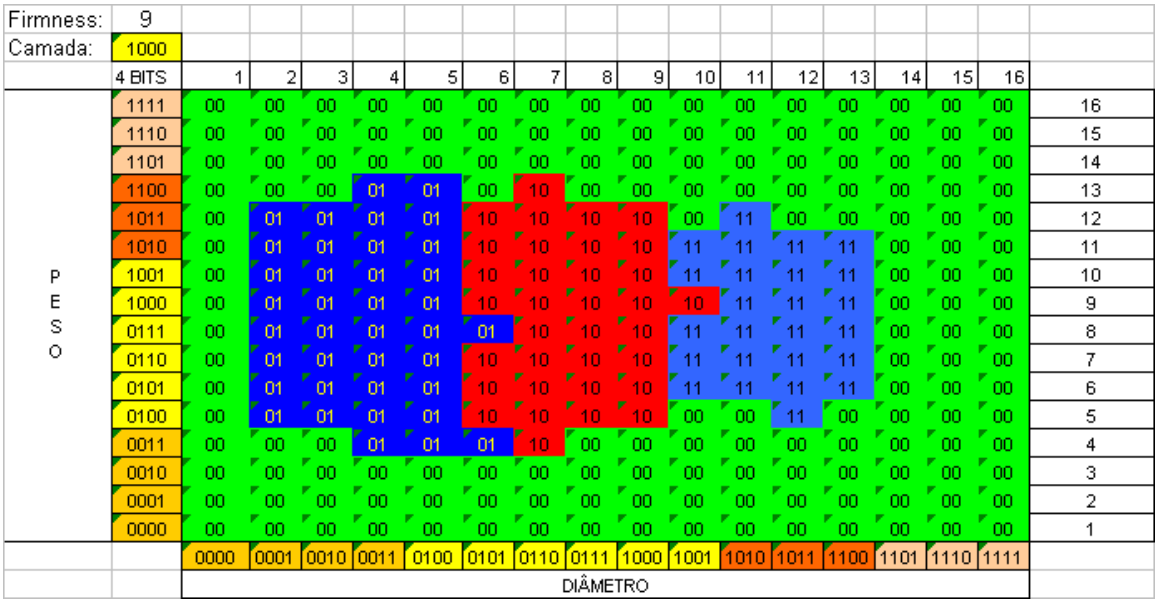

Figura 4.6.12 Espaço Amostral/ Padrões Tridimensional: Camada 1000: Firmness 9

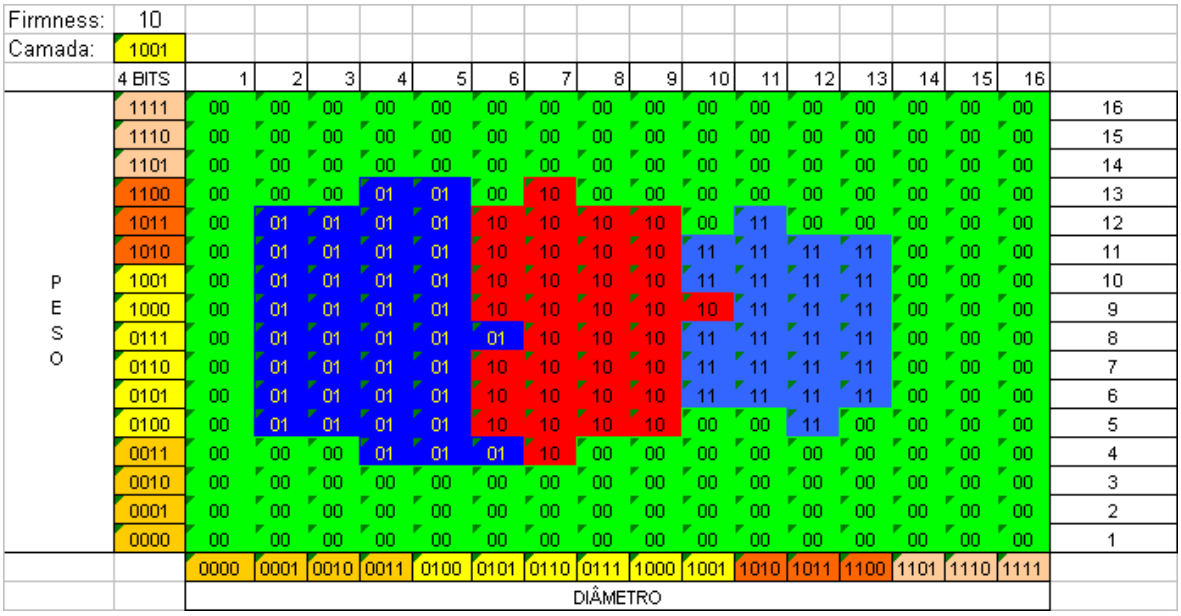

Figura 4.6.13 Espaço Amostral/ Padrões Tridimensional: Camada 1001: Firmness 10

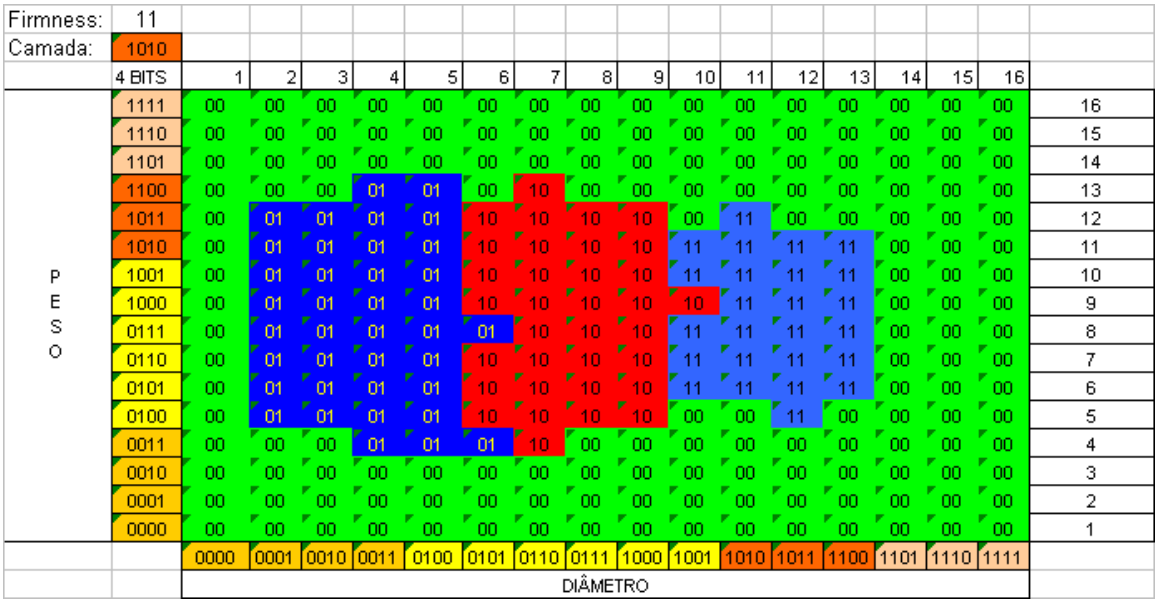

Figura 4.6.14 Espaço Amostral/ Padrões Tridimensional: Camada 1010: Firmness 11

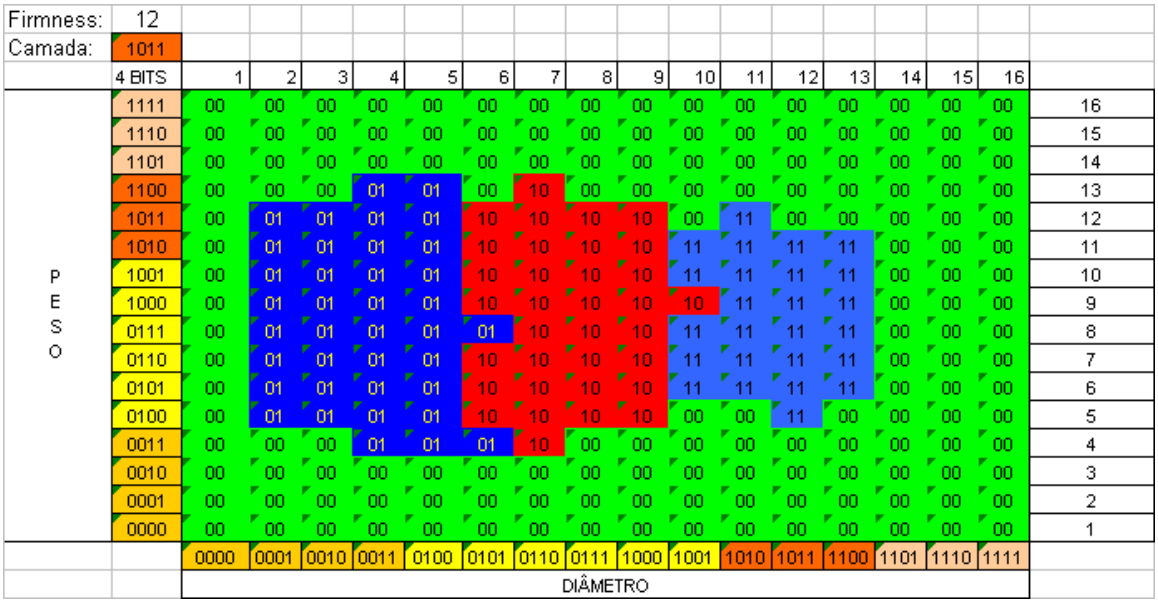

Figura 4.6.15 Espaço Amostral/ Padrões Tridimensional: Camada 1011: Firmness 12

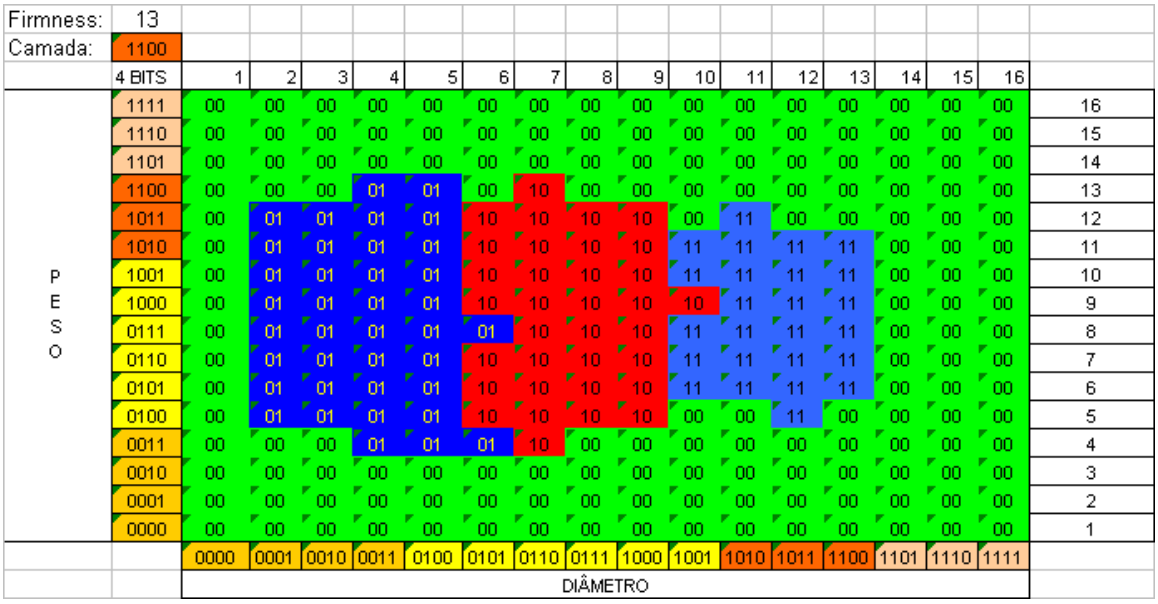

Figura 4.6.16 Espaço Amostral/ Padrões Tridimensional: Camada 1100: Firmness 13

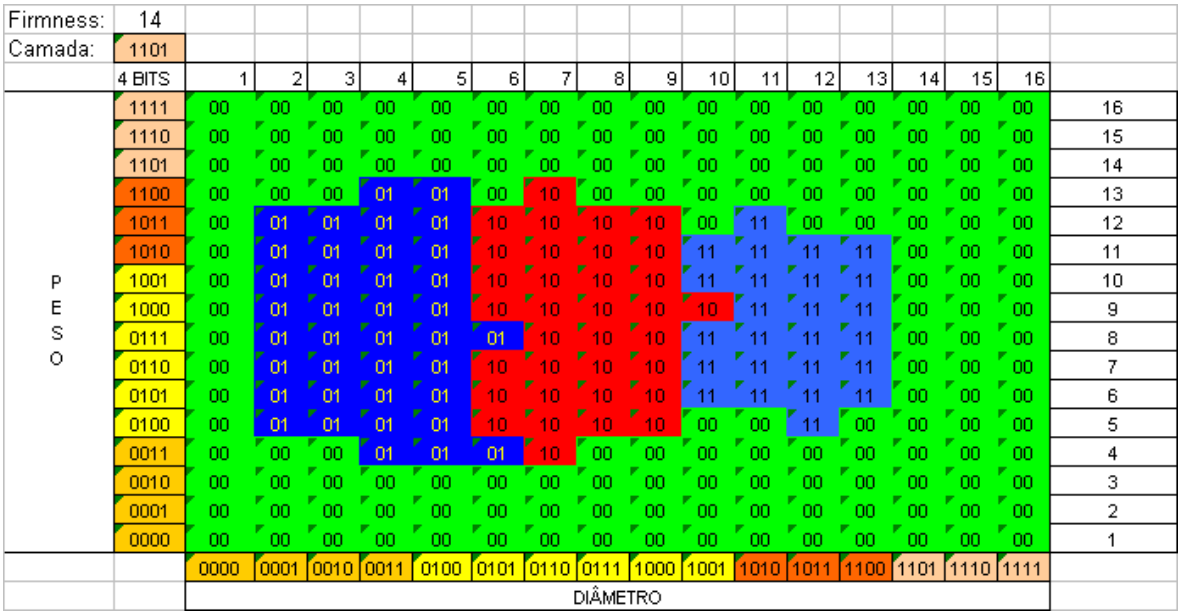

Figura 4.6.17 Espaço Amostral/ Padrões Tridimensional: Camada 1101: Firmness 14

Firmness:	15																	
Camada:	1110																	
	4 BITS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
	1111	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	16
	1110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	15
	1101	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	14
	1100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	13
	1011	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	12
	1010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	11
P	1001	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	10
E	1000	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	9
S	0111	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	8
O	0110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	7
	0101	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	6
	0100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	5
	0011	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	4
	0010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	3
	0001	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	2
	0000	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	1
		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	
		DIÂMETRO																

Figura 4.6.18 Espaço Amostral/ Padrões Tridimensional: Camada 1110: Firmness 15

Firmness:	16																	
Camada:	1111																	
	4 BITS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
	1111	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	16
	1110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	15
	1101	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	14
	1100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	13
	1011	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	12
	1010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	11
P	1001	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	10
E	1000	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	9
S	0111	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	8
O	0110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	7
	0101	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	6
	0100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	5
	0011	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	4
	0010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	3
	0001	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	2
	0000	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	1
		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	
		DIÂMETRO																

Figura 4.6.19 Espaço Amostral/ Padrões Tridimensional: Camada 1111: Firmness 16

Capítulo 5

5 Resultados e Discussões

5.1 Introdução

Inúmeros testes foram realizados na busca da topologia das redes neurais dos diversos cenários propostos neste trabalho. Neste capítulo são apresentados os resultados obtidos para os três cenários propostos: Unidimensional, Bidimensional e Tridimensional.

Todos os parâmetros de treinamento utilizados nos vários cenários são apresentados de forma que futuros pesquisadores possam se utilizar dos conceitos básicos para novas pesquisas na área de redes neurais.

O capítulo também apresenta os cálculos de teste do cenário unidimensional 4 bits, de forma que facilite o entendimento dos pesquisadores desta obra.

5.2 Resultados da Classificação Unidimensional

A parte experimental deste trabalho, projeto e testes das redes neurais, foi realizada utilizando-se o programa de simulação SNNS versão 4.1. Algumas considerações foram feitas e influenciaram na escolha de um pacote de simulação e especificamente ao SNNS para a criação e análise das redes, entre eles temos:

- YANG et al. (1999) e outros autores também utilizaram pacotes de simulação de redes neurais, devido a agilidade e facilidade para testes de configurações das redes neurais estudadas por eles.
- O pacote utilizado é de uso livre para pesquisas, facilitando para futuros pesquisadores deste trabalho, a criação de novas pesquisas sem custo.

ELIZONDO et al. (1994), escolheu uma rede neural tipo Perceptron de múltiplas camadas com algoritmo de treinamento retro-propagação ("*backpropagation*") para realizar suas pesquisas devido a este algoritmo ser um dos mais comumente utilizados para aprendizado

supervisionado. Utilizou-se em seu trabalho do pacote de redes neurais NeuroShell ™ destinado a criar modelos, o qual possui vários utilitários, concluindo que os modelos desenvolvidos que usam as redes neurais podem predizer o florescimento da soja e a maturidade fisiológica. Assim como ELIZONDO, SILVA (1998), BUFO (2000) e outros fizeram suas pesquisas utilizando-se de redes tipo Perceptron de múltiplas camadas.

O modelo de rede neural tipo Perceptron de múltiplas camadas com algoritmo retro-propagação (“*backpropagation”)* também foi utilizado no estudo do classificador artificial de frutos para todos os três cenários propostos: unidimensional, bidimensional, tridimensional. Os pontos principais considerados para a escolha deste tipo de modelo de redes foram:

- O tipo de rede mais utilizado nos trabalhos científicos revisados é o modelo Perceptron múltiplas camadas [MLP] com algoritmo de aprendizado retro-propagação (“*backpropagation*”).
- A forma de processamento é uma das mais fáceis de implementação para modelos reais.
- Programas computacionais são facilmente implementados em C++ ou Basic.

O programa de simulação SNNS 4.1 foi ajustado para treinamento das redes utilizando-se as funções de aprendizado tipo Std_Backpropagation, atualização tipo Topological_Order e inicialização tipo Randomize_Weights citadas no tópico 4.3 simulação por software relativo ao pacote de simulação SNNS.

Os ensaios das redes neurais em estudo foram feitos para a verificação da capacidade de expansão da camada de entrada, variando-se a topologia da rede, através da implementação de diferente número de neurônios na camada de entrada. O objetivo destes testes foi a avaliação do comportamento das redes com o aumento do processamento através do incremento de neurônios na camada de entrada.

Os neurônios da camada de entrada foram incrementados, criando padrões de entrada compostos de 4 a 8 neurônios no vetor de entrada da rede, como pode ser observado na figura 5.2.1:

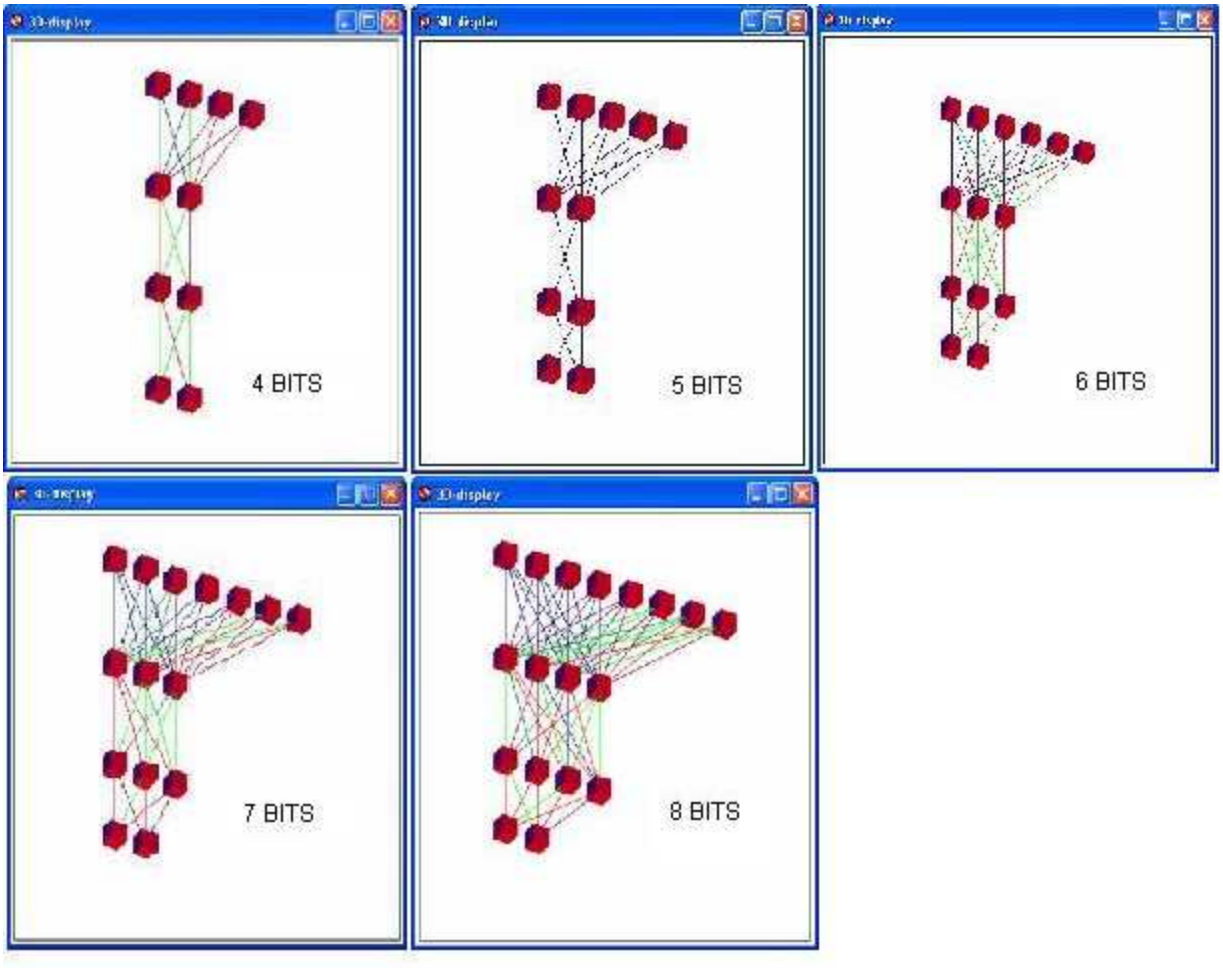

Figura 5.2.1 Topologias de Redes Neurais Unidimensionais – 1 propriedade Física

Tabela 5.2.1 – Topologia das Redes Unidimensionais e conexões.

	Neurônios por camada				
Cenário	Camada de Entrada	Camada Escondida 1	Camada Escondida 2	Camada de Saída	Conexões
4 bits	4	2	2	2	16
5 bits	5	2	2	2	18
6 bits	6	3	3	2	33
7 bits	7	3	3	2	36
8 bits	8	4	4	2	56

A tabela 5.2.1 apresenta os cenários unidimensionais estudados. A variação do vetor de entrada provocou o aumento do número total dos vetores para cada cenário. Todas as redes foram treinadas literalmente com todos os vetores de entrada, não utilizando a capacidade de

generalização da rede. Esta decisão, assumida por projeto, pretende criar um modelo onde a generalização ocorre nos limites mínimo e máximo de cada canal de entrada, ou seja, uma vez o modelo elaborado, os valores dos limites são mapeados para vetores binários que fazem a representação dos valores.

A seguir será apresentado um modelo de rede neural de 4 bits, com 16 intervalos apenas, de forma que facilite o entendimento do processo adotado para todos os cenários estudados.

A tabela 5.2.2 apresenta o espaço de padrões informados para aprendizado/classificação de uma rede neural Perceptron de múltiplas camadas e algoritmo de aprendizado "*backpropagation*", com 4 bits na camada de entrada e 2 bits na camada de saída. Como pode ser observado a rede convergiu para a classificação correta, dentro de um critério de parada inferior a 0.01 da diferença entre a saída desejada e saída obtida.

Tabela 5.2.2 Resultados de Classificação de uma rede neural de 4 bits utilizando rede Perceptron de múltiplas camadas e algoritmo de aprendizado "*backpropagation"*.

Tabela de Resultados						
	Amostra	Entrada	Saída Desejada		Saída Obtida	
1	1	0 0 0 0	0	0	0.00289	0.00073
2	2	0 0 0 1	0	0	0.0029	0.00073
3	3	0 0 1 0	0	0	0.00282	0.00073
4	4	0 0 1 1	0	0	0.00283	0.00072
5	5	0 1 0 0	1	0	0.99996	0.00252
6	6	0 1 0 1	1	0	0.99997	0.00252
7	7	0 1 1 0	1	0	0.99931	0.00249
8	8	0 1 1 1	1	0	0.99933	0.0022
9	9	1 0 0 0	1	0	0.99936	0.00244
10	10	1 0 0 1	1	0	0.99937	0.00219
11	11	1 0 1 0	0	1	0	1
12	12	1 0 1 1	0	1	0	1
13	13	1 1 0 0	0	1	0.00997	1
14	14	1 1 0 1	1	1	0.99133	0.99223
15	15	1 1 1 0	1	1	0.99479	1
16	16	1 1 1 1	1	1	0.99456	1

As tabelas 5.2.3, 5.2.4,5. 2.5 e 5.2.6, apresentam os parâmetros utilizados para o treinamento e convergência da rede unidimensional de 4 bits estudada.

Tabela 5.2.3 Parâmetros de treinamento

Rede Neural de Classificação Unidimensional : 4 BITS : PROPRIEDADE FÍSICA : PESO	
Criação	SNNS network definition file V1.4-3D generated at Wed Oct 10 08:10:28 2001
network name	CENARIOLINEAR4BITS_1000000
source files	
no. of units	10
no. of connections	16
no. of unit types	0
no. of site types	0
learning function	Std_Backpropagation
update function	Topological_Order

Tabela 5.2.4 função de ativação

Act	Bias	St	subnet	layer	act func	out func
0.00000	0.00000	H	0	1	Act_Logistic	Out_Identity

Tabela 5.2.5 Topologia final da Rede Neural Unidimensional

Topologia final da Rede Neural Unidimensional 4 Bits : 1 Propriedade : Peso									
no	typeName	unitName	act	bias	St	position	act func	out func	sites
1			1.00000	-0.03952	I	2, 2, 0			
2			1.00000	0.19769	I	3, 2, 0			
3			1.00000	-0.21352	I	4, 2, 0			
4			1.00000	-0.57080	I	5, 2, 0			
5			0.00054	1.60087	H	2, 5, 0			
6			0.98578	-12.27841	H	3, 5, 0			
7			0.00000	4.91201	H	2, 8, 0			
8			0.06088	0.22223	H	3, 8, 0			
9			0.99456	7.45590	O	2, 11, 0			
10			1.00000	19.89206	O	3, 11, 0			

Tabela 5.2.6 Pesos das conexões Rede Neural Cenário Unidimensional

target	Site	source:weight
5		1:-4.53884, 2:-7.34133, 3: 2.83024, 4:-0.07339
6		1: 8.57594, 2: 2.75296, 3: 5.90310, 4:-0.71482
7		5: 0.41064, 6:-29.21679
8		5: 8.10780, 6:-3.00536
9		7:23.70916, 8:-36.91986
10		7:-24.52733, 8:-2.71728

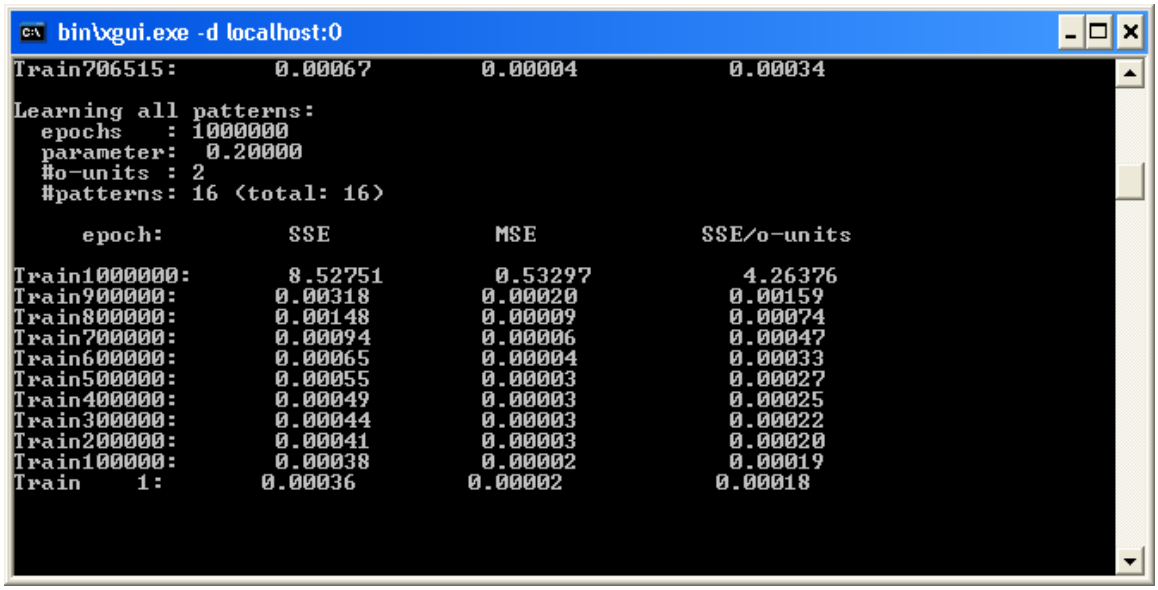

Figura 5.2.2 Dados de convergência durante treinamento da Rede Unidimensional 4 bits

A convergência de erros para o ciclo de 1.000.000 foi reduzida aos valores:

- SSE = 0.00036
- MSE = 0.00002
- SSE /O units = 0.00018

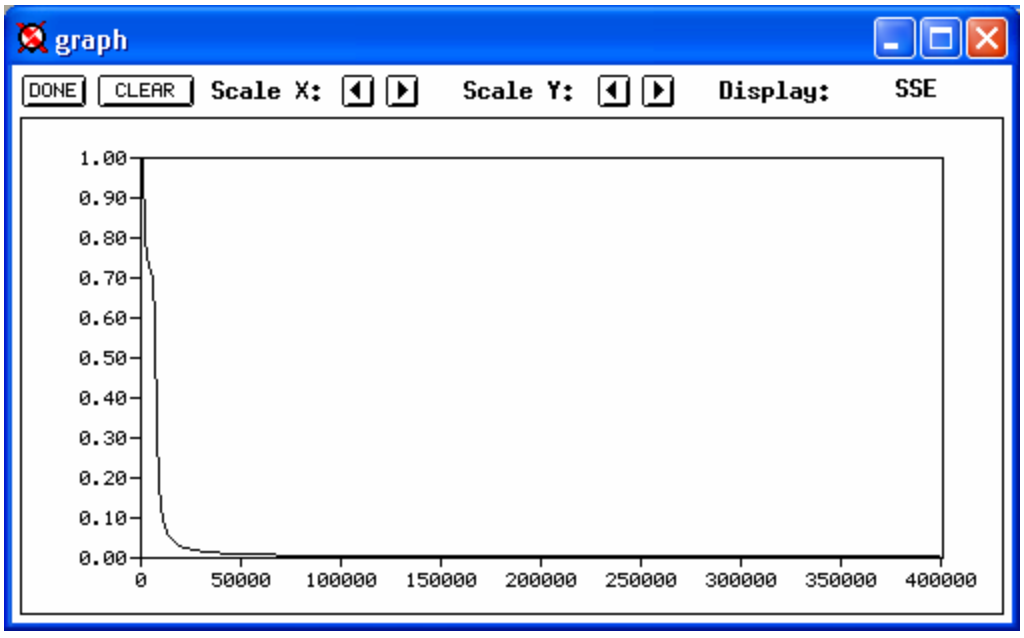

Figura 5.2.3 Curva de Convergência do Erro SSE para o cenário unidimensional 4 bits: Peso

Os resultados dos cálculos de classificação da rede neural unidimensional 4 bits de Peso, após a rede ter sido estabilizada, são apresentados nas tabelas 5.2.7 à Tabela 5.2.22. Como pode ser observado nos cálculos do vetor de saída e tabela 4.4.1, todos os vetores de entrada foram classificados corretamente com as respectivas classes de saída, considerando-se o critério de erro inferior à 0.01 entre os valores de saída desejados e obtidos.

Tomando-se como exemplo o vetor de entrada: [1,1,00] da Tabela 5.2.19 espera-se que a rede classifique o vetor de entrada para o vetor de saída [0,1]. A saída do neurônio (Ativação) é calculado através da expressão, Act = 1/(1+EXP(-? [PESO*ACT]-bias)), calculada da camada escondida 1 até a camada de saída.

Tabela 5.2.7 Cálculo de validação rede unidimensional 4 Bits: Vetor de Entrada : 0000

REDE UNIDIMENSIONAL 4 BITS : ENTRADA :			0000				
		act	**bias**				
1	Neurônio 1	0	-0.039520	Saída Desejada 1:	0.000000	Saída Obtida 1:	0.002890
2	Neurônio 2	0	0.197690	Saída Desejada 2:	0.000000	Saída Obtida 2:	0.000730
3	Neurônio 3	0	-0.213520				
4	Neurônio 4	0	-0.570800	PESOS			
5		1.000000	1.600870	-0.07339	2.83024	-7.34133	-4.53884
6		0.999980	-12.278410	-0.71482	5.9031	2.75296	8.57594
7		0.563800	4.912010	0.41064	-29.21679		
8		0.008900	0.222230	8.1078	-3.00536		
9		0.996550	7.455900	23.70916	-36.91986		
10		0.996380	19.892060	-24.52733	-2.71728		

Entrada	1/(1+EXP(-? [PESO*ACT]-bias))	**bias**	PESO * ACT				? [PESO*ACT]
5	0.832139945	1.60087	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
6	4.65106E-06	-12.2784	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
Entrada			0.83213994	4.65106E-06	? [PESO*ACT]		
7	0.994798453	4.91201	0.34170995	-0.000135889	0.341574058		
8	0.999060329	0.22223	6.74682424	-1.39781E-05	6.746810264		
Entrada	Saídas		0.99479845	0.999060329	? [PESO*ACT]		
9	0.002890	7.4559	23.5858357	-36.88516747	-13.29933178		
10	0.000730	19.8921	-24.3997499	-2.71472665	-27.11447659		

Tabela 5.2.8 Cálculo de validação Rede Unidimensional 4 Bits : Vetor de Entrada : 0001

REDE UNIDIMENSIONAL 4 BITS : ENTRADA :			0001				
		act	**bias**				
1	Neurônio 1	0	-0.039520	Saída Desejada 1:	0.000000	Saída Obtida 1:	0.002898
2	Neurônio 2	0	0.197690	Saída Desejada 2:	0.000000	Saída Obtida 2:	0.000730
3	Neurônio 3	0	-0.213520				
4	Neurônio 4	1	-0.570800	PESOS			
5		1.000000	1.600870	-0.07339	2.83024	-7.34133	-4.53884
6		0.999980	-12.278410	-0.71482	5.9031	2.75296	8.57594
7		0.563800	4.912010	0.41064	-29.21679		
8		0.008900	0.222230	8.1078	-3.00536		
9		0.996550	7.455900	23.70916	-36.91986		
10		0.996380	19.892060	-24.52733	-2.71728		

Entrada	1/(1+EXP(-?[PESO*ACT]-bias))	**bias**	PESO * ACT				?[PESO*ACT]
5	0.821637309	1.60087	0.0000000	0.0000000	0.0000000	-0.0733900	-0.0733900
6	2.27568E-06	-12.2784	0.0000000	0.0000000	0.0000000	-0.7148200	-0.7148200
Entrada			0.82163731	2.27568E-06	? [PESO*ACT]		
7	0.994776449	4.91201	0.33739714	-6.6488E-05	0.337330657		
8	0.9989769	0.22223	6.66167097	-6.83923E-06	6.661664134		
Entrada	Saídas		0.99477645	0.9989769	? [PESO*ACT]		
9	0.002898	7.4559	23.585314	-36.88208728	-13.29677328		
10	0.000730	19.8921	-24.3992102	-2.71449995	-27.1137102		

Tabela 5.2.9 Cálculo de validação Rede Unidimensional 4 Bits : Vetor de Entrada : 0010

REDE UNIDIMENSIONAL 4 BITS : ENTRADA :			0010				
		act	**bias**				
1	Neurônio 1	0	-0.039520	Saída Desejada 1:	0.000000	Saída Obtida 1:	0.002825
2	Neurônio 2	0	0.197690	Saída Desejada 2:	0.000000	Saída Obtida 2:	0.000727
3	Neurônio 3	1	-0.213520				
4	Neurônio 4	0	-0.570800	PESOS			
5		1.000000	1.600870	-0.07339	2.83024	-7.34133	-4.53884
6		0.999980	-12.278410	-0.71482	5.9031	2.75296	8.57594
7		0.563800	4.912010	0.41064	-29.21679		
8		0.008900	0.222230	8.1078	-3.00536		
9		0.996550	7.455900	23.70916	-36.91986		
10		0.996380	19.892060	-24.52733	-2.71728		

Entrada	1/(1+EXP(-?[PESO*ACT]-bias))	**bias**	PESO * ACT				?[PESO*ACT]
5	0.988238702	1.60087	0.0000000	0.0000000	2.8302400	0.0000000	2.8302400
6	0.001700196	-12.2784	0.0000000	0.0000000	5.9031000	0.0000000	5.9031000
Entrada			0.9882387	0.001700196	? [PESO*ACT]		
7	0.994873264	4.91201	0.40581034	-0.049674274	0.356136066		
8	0.999733418	0.22223	8.01244175	-0.005109702	8.00733205		
Entrada	Saídas		0.99487326	0.999733418	? [PESO*ACT]		
9	0.002825	7.4559	23.5876094	-36.91001783	-13.32240844		
10	0.000727	19.8921	-24.4015848	-2.716555622	-27.11814047		

Tabela 5.2.10 Cálculo de validação Rede Unidimensional 4 Bits : Vetor de Entrada : 0011

REDE UNIDIMENSIONAL 4 BITS : ENTRADA :		0011					
		act	**bias**				
1	Neurônio 1	0	-0.039520	Saída Desejada 1:	0.000000	Saída Obtida 1:	0.002833
2	Neurônio 2	0	0.197690	Saída Desejada 2:	0.000000	Saída Obtida 2:	0.000725
3	Neurônio 3	1	-0.213520				
4	Neurônio 4	1	-0.570800	PESOS			
5		1.000000	1.600870	-0.07339	2.83024	-7.34133	-4.53884
6		0.999980	-12.278410	-0.71482	5.9031	2.75296	8.57594
7		0.563800	4.912010	0.41064	-29.21679		
8		0.008900	0.222230	8.1078	-3.00536		
9		0.996550	7.455900	23.70916	-36.91986		
10		0.996380	19.892060	-24.52733	-2.71728		

Entrada	1/(1+EXP(-?[PESO*ACT]-bias))	**bias**	PESO * ACT				? [PESO*ACT]
5	0.987354404	1.60087	0.0000000	0.0000000	2.8302400	-0.0733900	2.7568500
6	0.000832595	-12.2784	0.0000000	0.0000000	5.9031000	-0.7148200	5.1882800
Entrada			0.9873544	0.000832595	? [PESO*ACT]		
7	0.994999137	4.91201	0.40544721	-0.02432576	0.381121452		
8	0.999732199	0.22223	8.00527203	-0.002502248	8.002769785		
Entrada	Saídas		0.99499914	0.999732199	? [PESO*ACT]		
9	0.002833	7.4559	23.5905937	-36.90997284	-13.31937909		
10	0.000725	19.8921	-24.4046722	-2.716552311	-27.1212245		

Tabela 5.2.11 Cálculo de validação Rede Unidimensional 4 Bits : Vetor de Entrada : 0100

REDE UNIDIMENSIONAL 4 BITS : ENTRADA :				0100			
		act	**bias**				
1	Neurônio 1	0	-0.039520	Saída Desejada 1:	1.000000	Saída Obtida 1:	0.999965
2	Neurônio 2	1	0.197690	Saída Desejada 2:	0.000000	Saída Obtida 2:	0.002517
3	Neurônio 3	0	-0.213520				
4	Neurônio 4	0	-0.570800	PESOS			
5		1.000000	1.600870	-0.07339	2.83024	-7.34133	-4.53884
6		0.999980	-12.278410	-0.71482	5.9031	2.75296	8.57594
7		0.563800	4.912010	0.41064	-29.21679		
8		0.008900	0.222230	8.1078	-3.00536		
9		0.996550	7.455900	23.70916	-36.91986		
10		0.996380	19.892060	-24.52733	-2.71728		

Entrada	1/(1+EXP(-? [PESO*ACT]-bias))	**bias**	PESO * ACT				? [PESO*ACT]
5	0.003202998	1.60087	0.0000000	-7.3413300	0.0000000	0.0000000	-7.3413300
6	7.29656E-05	-12.2784	0.0000000	2.7529600	0.0000000	0.0000000	2.7529600
Entrada			0.003203	7.29656E-05	? [PESO*ACT]		
7	0.992690133	4.91201	0.00131528	-0.00213182	-0.000816541		
8	0.561679242	0.22223	0.02596926	-0.000219288	0.025749977		
Entrada	Saídas		0.99269013	0.561679242	? [PESO*ACT]		
9	0.999965	7.4559	23.5358492	-20.73711898	2.79873021		
10	0.002517	19.8921	-24.3480385	-1.526239771	-25.87427825		

Tabela 5.2.12 Cálculo de validação Rede Unidimensional 4 Bits : Vetor de Entrada : 0101

REDE UNIDIMENSIONAL 4 BITS : ENTRADA :				0101			
		act	**bias**				
1	Neurônio 1	0	-0.039520	Saída Desejada 1:	1.000000	Saída Obtida 1:	0.999965
2	Neurônio 2	1	0.197690	Saída Desejada 2:	0.000000	Saída Obtida 2:	0.002519
3	Neurônio 3	0	-0.213520				
4	Neurônio 4	1	-0.570800	PESOS			
5		1.000000	1.600870	-0.07339	2.83024	-7.34133	-4.53884
6		0.999980	-12.278410	-0.71482	5.9031	2.75296	8.57594
7		0.563800	4.912010	0.41064	-29.21679		
8		0.008900	0.222230	8.1078	-3.00536		
9		0.996550	7.455900	23.70916	-36.91986		
10		0.996380	19.892060	-24.52733	-2.71728		

Entrada	1/(1+EXP(-? [PESO*ACT]-bias))	**bias**	PESO * ACT				? [PESO*ACT]
5	0.002977023	1.60087	0.0000000	-7.3413300	0.0000000	-0.0733900	-7.4147200
6	3.57019E-05	-12.2784	0.0000000	2.7529600	0.0000000	-0.7148200	2.0381400
Entrada			0.00297702	3.57019E-05	? [PESO*ACT]		
7	0.992697356	4.91201	0.00122248	-0.001043096	0.000179389		
8	0.5612557	0.22223	0.02413711	-0.000107297	0.02402981		
Entrada	Saídas		0.99269736	0.5612557	? [PESO*ACT]		
9	0.999965	7.4559	23.5360204	-20.72148185	2.814538595		
10	0.002519	19.8921	-24.3482156	-1.525088887	-25.87330453		

Tabela 5.2.13 Cálculo de validação Rede Unidimensional 4 Bits : Vetor de Entrada : 0110

REDE UNIDIMENSIONAL 4 BITS : ENTRADA :				0110			
		act	**bias**				
1	Neurônio 1	0	-0.039520	Saída Desejada 1:	1.000000	Saída Obtida 1:	0.999314
2	Neurônio 2	1	0.197690	Saída Desejada 2:	0.000000	Saída Obtida 2:	0.002487
3	Neurônio 3	1	-0.213520				
4	Neurônio 4	0	-0.570800	PESOS			
5		1.000000	1.600870	-0.07339	2.83024	-7.34133	-4.53884
6		0.999980	-12.278410	-0.71482	5.9031	2.75296	8.57594
7		0.563800	4.912010	0.41064	-29.21679		
8		0.008900	0.222230	8.1078	-3.00536		
9		0.996550	7.455900	23.70916	-36.91986		
10		0.996380	19.892060	-24.52733	-2.71728		

Entrada	1/(1+EXP(-? [PESO*ACT]-bias))	**bias**	PESO * ACT				? [PESO*ACT]
5	0.051650658	1.60087	0.0000000	-7.3413300	2.8302400	0.0000000	-4.5110900
6	0.026024443	-12.2784	0.0000000	2.7529600	5.9031000	0.0000000	8.6560600
Entrada			0.05165066	0.026024443	? [PESO*ACT]		
7	0.984825815	4.91201	0.02120983	-0.760350683	-0.739140857		
8	0.637097936	0.22223	0.41877321	-0.07821282	0.340560387		
Entrada	Saídas		0.98482581	0.637097936	? [PESO*ACT]		
9	0.999314	7.4559	23.3493928	-23.52156659	-0.17217378		
10	0.002487	19.8921	-24.1551478	-1.731173479	-25.88632123		

Tabela 5.2.14 Cálculo de validação Rede Unidimensional 4 Bits : Vetor de Entrada : 0111

REDE UNIDIMENSIONAL 4 BITS : ENTRADA :				0111			
		act	**bias**				
1	Neurônio 1	0	-0.039520	Saída Desejada 1:	1.000000	Saída Obtida 1:	0.999326
2	Neurônio 2	1	0.197690	Saída Desejada 2:	0.000000	Saída Obtida 2:	0.002198
3	Neurônio 3	1	-0.213520				
4	Neurônio 4	1	-0.570800	PESOS			
5		1.000000	1.600870	-0.07339	2.83024	-7.34133	-4.53884
6		0.999980	-12.278410	-0.71482	5.9031	2.75296	8.57594
7		0.563800	4.912010	0.41064	-29.21679		
8		0.008900	0.222230	8.1078	-3.00536		
9		0.996550	7.455900	23.70916	-36.91986		
10		0.996380	19.892060	-24.52733	-2.71728		

Entrada	1/(1+EXP(-? [PESO*ACT]-bias))	**bias**	PESO * ACT				? [PESO*ACT]
5	0.048171835	1.60087	0.0000000	-7.3413300	2.8302400	-0.0733900	-4.5844800
6	0.012904764	-12.2784	0.0000000	2.7529600	5.9031000	-0.7148200	7.9412400
Entrada			0.04817184	0.012904764	? [PESO*ACT]		
7	0.989592386	4.91201	0.01978128	-0.377035772	-0.35725449		
8	0.639688901	0.22223	0.39056761	-0.038783461	0.351784145		
Entrada	Saídas		0.98959239	0.639688901	? [PESO*ACT]		
9	0.999326	7.4559	23.4624042	-23.61722468	-0.154820456		
10	0.002198	19.8921	-24.272059	-1.738213858	-26.01027288		

Tabela 5.2.15 Cálculo de validação Rede Unidimensional 4 Bits : Vetor de Entrada : 1000

REDE UNIDIMENSIONAL 4 BITS : ENTRADA :				1000			
		act	**bias**				
1	Neurônio 1	1	-0.039520	Saída Desejada 1:	1.000000	Saída Obtida 1:	0.999355
2	Neurônio 2	0	0.197690	Saída Desejada 2:	0.000000	Saída Obtida 2:	0.002445
3	Neurônio 3	0	-0.213520				
4	Neurônio 4	0	-0.570800	PESOS			
5		1.000000	1.600870	-0.07339	2.83024	-7.34133	-4.53884
6		0.999980	-12.278410	-0.71482	5.9031	2.75296	8.57594
7		0.563800	4.912010	0.41064	-29.21679		
8		0.008900	0.222230	8.1078	-3.00536		
9		0.996550	7.455900	23.70916	-36.91986		
10		0.996380	19.892060	-24.52733	-2.71728		

Entrada	1/(1+EXP(-? [PESO*ACT]-bias))	**bias**	PESO * ACT				? [PESO*ACT]
5	0.050308173	1.60087	-4.5388400	0.0000000	0.0000000	0.0000000	-4.5388400
6	0.024068934	-12.2784	8.5759400	0.0000000	0.0000000	0.0000000	8.5759400
Entrada			0.05030817	0.024068934	? [PESO*ACT]		
7	0.985648588	4.91201	0.02065855	-0.703216984	-0.682558436		
8	0.635939366	0.22223	0.4078886	-0.072335811	0.33555279		
Entrada	Saídas		0.98564859	0.635939366	? [PESO*ACT]		
9	0.999355	7.4559	23.3689001	-23.47879235	-0.109892271		
10	0.002445	19.8921	-24.1753282	-1.72802532	-25.9033535		

Tabela 5.2.16 Cálculo de validação Rede Unidimensional 4 Bits : Vetor de Entrada : 1001

REDE UNIDIMENSIONAL 4 BITS : ENTRADA :				1001			
		act	**bias**				
1	Neurônio 1	1	-0.039520	Saída Desejada 1:	1.000000	Saída Obtida 1:	0.999370
2	Neurônio 2	0	0.197690	Saída Desejada 2:	0.000000	Saída Obtida 2:	0.002192
3	Neurônio 3	0	-0.213520				
4	Neurônio 4	1	-0.570800	PESOS			
5		1.000000	1.600870	-0.07339	2.83024	-7.34133	-4.53884
6		0.999980	-12.278410	-0.71482	5.9031	2.75296	8.57594
7		0.563800	4.912010	0.41064	-29.21679		
8		0.008900	0.222230	8.1078	-3.00536		
9		0.996550	7.455900	23.70916	-36.91986		
10		0.996380	19.892060	-24.52733	-2.71728		

Entrada	1/(1+EXP(-? [PESO*ACT]-bias))	**bias**	PESO * ACT				? [PESO*ACT]
5	0.046915297	1.60087	-4.5388400	0.0000000	0.0000000	-0.0733900	-4.6122300
6	0.011923016	-12.2784	8.5759400	0.0000000	0.0000000	-0.7148200	7.8611200
Entrada			0.0469153	0.011923016	? [PESO*ACT]		
7	0.989878527	4.91201	0.0192653	-0.348352242	-0.329086945		
8	0.638019127	0.22223	0.38037984	-0.035832954	0.344546889		
Entrada	Saídas		0.98987853	0.638019127	? [PESO*ACT]		
9	0.999370	7.4559	23.4691884	-23.55557685	-0.086388465		
10	0.002192	19.8921	-24.2790773	-1.733676614	-26.01275391		

Tabela 5.2.17 Cálculo de validação Rede Unidimensional 4 Bits : Vetor de Entrada : 1010

REDE UNIDIMENSIONAL 4 BITS : ENTRADA :				1010			
		act	**bias**				
1	Neurônio 1	1	-0.039520	Saída Desejada 1:	0.000000	Saída Obtida 1:	0.000000
2	Neurônio 2	0	0.197690	Saída Desejada 2:	1.000000	Saída Obtida 2:	1.000000
3	Neurônio 3	1	-0.213520				
4	Neurônio 4	0	-0.570800	PESOS			
5		1.000000	1.600870	-0.07339	2.83024	-7.34133	-4.53884
6		0.999980	-12.278410	-0.71482	5.9031	2.75296	8.57594
7		0.563800	4.912010	0.41064	-29.21679		
8		0.008900	0.222230	8.1078	-3.00536		
9		0.996550	7.455900	23.70916	-36.91986		
10		0.996380	19.892060	-24.52733	-2.71728		

Entrada	1/(1+EXP(-? [PESO*ACT]-bias))	**bias**	PESO * ACT				? [PESO*ACT]
5	0.473093517	1.60087	-4.5388400	0.0000000	2.8302400	0.0000000	-1.7086000
6	0.900306071	-12.2784	8.5759400	0.0000000	5.9031000	0.0000000	14.4790400
Entrada			0.47309352	0.900306071	? [PESO*ACT]		
7	6.22192E-10	4.91201	0.19427112	-26.30405341	-26.10978228		
8	0.794494582	0.22223	3.83574762	-2.705743853	1.130003768		
Entrada	Saídas		6.2219E-10	0.794494582	? [PESO*ACT]		
9	0.000000	7.4559	1.4752E-08	-29.33262874	-29.33262872		
10	1.000000	19.8921	-1.5261E-08	-2.158864238	-2.158864253		

Tabela 5.2.18 Cálculo de validação Rede Unidimensional 4 Bits : Vetor de Entrada : 1011

REDE UNIDIMENSIONAL 4 BITS : ENTRADA :			1011				
		act	**bias**				
1	Neurônio 1	1	-0.039520	Saída Desejada 1:	0.000000	Saída Obtida 1:	0.000000
2	Neurônio 2	0	0.197690	Saída Desejada 2:	1.000000	Saída Obtida 2:	1.000000
3	Neurônio 3	1	-0.213520				
4	Neurônio 4	1	-0.570800	PESOS			
5		1.000000	1.600870	-0.07339	2.83024	-7.34133	-4.53884
6		0.999980	-12.278410	-0.71482	5.9031	2.75296	8.57594
7		0.563800	4.912010	0.41064	-29.21679		
8		0.008900	0.222230	8.1078	-3.00536		
9		0.996550	7.455900	23.70916	-36.91986		
10		0.996380	19.892060	-24.52733	-2.71728		

Entrada	1/(1+EXP(-? [PESO*ACT]-bias))	**bias**	PESO * ACT				? [PESO*ACT]
5	0.454843377	1.60087	-4.5388400	0.0000000	2.8302400	-0.0733900	-1.7819900
6	0.815448543	-12.2784	8.5759400	0.0000000	5.9031000	-0.7148200	13.7642200
Entrada			0.45484338	0.815448543	? [PESO*ACT]		
7	7.36887E-09	4.91201	0.18677688	-23.82478885	-23.63801197		
8	0.811424471	0.22223	3.68777914	-2.450716435	1.237062701		
Entrada	Saidas		7.3689E-09	0.811424471	? [PESO*ACT]		
9	0.000000	7.4559	1.7471E-07	-29.95767788	-29.95767771		
10	1.000000	19.8921	-1.8074E-07	-2.204867488	-2.204867668		

Tabela 5.2.19 Cálculo de validação Rede Unidimensional 4 Bits : Vetor de Entrada : 1100

REDE UNIDIMENSIONAL 4 BITS : ENTRADA :			1100				
		act	**bias**				
1	Neurônio 1	1	-0.039520	Saída Desejada 1:	0.000000	Saída Obtida 1:	0.009975
2	Neurônio 2	1	0.197690	Saída Desejada 2:	1.000000	Saída Obtida 2:	1.000000
3	Neurônio 3	0	-0.213520				
4	Neurônio 4	0	-0.570800	PESOS			
5		1.000000	1.600870	-0.07339	2.83024	-7.34133	-4.53884
6		0.999980	-12.278410	-0.71482	5.9031	2.75296	8.57594
7		0.563800	4.912010	0.41064	-29.21679		
8		0.008900	0.222230	8.1078	-3.00536		
9		0.996550	7.455900	23.70916	-36.91986		
10		0.996380	19.892060	-24.52733	-2.71728		

Entrada	1/(1+EXP(-? [PESO*ACT]-bias))	**bias**	PESO * ACT				? [PESO*ACT]
5	3.43354E-05	1.60087	-4.5388400	-7.3413300	0.0000000	0.0000000	-11.8801700
6	0.278983376	-12.2784	8.5759400	2.7529600	0.0000000	0.0000000	11.3289000
Entrada			3.4335E-05	0.278983376	? [PESO*ACT]		
7	0.037725097	4.91201	1.4099E-05	-8.150998699	-8.150984599		
8	0.350706059	0.22223	0.00027838	-0.838445478	-0.838167093		
Entrada	Saidas		0.0377251	0.350706059	? [PESO*ACT]		
9	0.009975	7.4559	0.89443036	-12.9480186	-12.05358825		
10	1.000000	19.8921	-0.9252959	-0.95296656	-1.878262461		

Tabela 5.2.20 Cálculo de validação Rede Unidimensional 4 Bits : Vetor de Entrada : 1101

REDE UNIDIMENSIONAL 4 BITS : ENTRADA :			1101				
		act	**bias**				
1	Neurônio 1	1	-0.039520	Saída Desejada 1:	1.000000	Saída Obtida 1:	0.991333
2	Neurônio 2	1	0.197690	Saída Desejada 2:	1.000000	Saída Obtida 2:	0.992231
3	Neurônio 3	0	-0.213520				
4	Neurônio 4	1	-0.570800	PESOS			
5		1.000000	1.600870	-0.07339	2.83024	-7.34133	-4.53884
6		0.999980	-12.278410	-0.71482	5.9031	2.75296	8.57594
7		0.563800	4.912010	0.41064	-29.21679		
8		0.008900	0.222230	8.1078	-3.00536		
9		0.996550	7.455900	23.70916	-36.91986		
10		0.996380	19.892060	-24.52733	-2.71728		

Entrada	1/(1+EXP(-? [PESO*ACT]-bias))	**bias**	PESO * ACT				? [PESO*ACT]
5	3.19058E-05	1.60087	-4.5388400	-7.3413300	0.0000000	-0.0733900	-11.9535600
6	0.159181602	-12.2784	8.5759400	2.7529600	0.0000000	-0.7148200	10.6140800
Entrada			3.1906E-05	0.159181602	? [PESO*ACT]		
7	0.564942971	4.91201	1.3102E-05	-4.650775435	-4.650762334		
8	0.436369549	0.22223	0.00025869	-0.478398019	-0.478139333		
Entrada	Saidas		0.56494297	0.436369549	? [PESO*ACT]		
9	0.991333	7.4559	13.3943233	-16.11070267	-2.716379375		
10	0.992231	19.8921	-13.8565427	-1.185738249	-15.04228093		

Tabela 5.2.21 Cálculo de validação Rede Unidimensional 4 Bits : Vetor de Entrada : 1110

REDE UNIDIMENSIONAL 4 BITS : ENTRADA :			1110				
		act	**bias**				
1	Neurônio 1	1	-0.039520	Saída Desejada 1:	1.000000	Saída Obtida 1:	0.994794
2	Neurônio 2	1	0.197690	Saída Desejada 2:	1.000000	Saída Obtida 2:	1.000000
3	Neurônio 3	1	-0.213520				
4	Neurônio 4	0	-0.570800	PESOS			
5		1.000000	1.600870	-0.07339	2.83024	-7.34133	-4.53884
6		0.999980	-12.278410	-0.71482	5.9031	2.75296	8.57594
7		0.563800	4.912010	0.41064	-29.21679		
8		0.008900	0.222230	8.1078	-3.00536		
9		0.996550	7.455900	23.70916	-36.91986		
10		0.996380	19.892060	-24.52733	-2.71728		

Entrada	1/(1+EXP(-? [PESO*ACT]-bias))	**bias**	PESO * ACT				? [PESO*ACT]
5	0.00058165	1.60087	-4.5388400	-7.3413300	2.8302400	0.0000000	-9.0499300
6	0.992991441	-12.2784	8.5759400	2.7529600	5.9031000	0.0000000	17.2320000
Entrada			0.00058165	0.992991441	? [PESO*ACT]		
7	3.41666E-11	4.91201	0.00023885	-29.01202242	-29.01178357		
8	0.059672841	0.22223	0.0047159	-2.984296759	-2.979580857		
Entrada	Saidas		3.4167E-11	0.059672841	? [PESO*ACT]		
9	0.994794	7.4559	8.1006E-10	-2.203112949	-2.203112948		
10	1.000000	19.8921	-8.3801E-10	-0.162147818	-0.162147819		

Tabela 5.2.22 Cálculo de validação Rede Unidimensional 4 Bits : Vetor de Entrada : 1111

REDE UNIDIMENSIONAL 4 BITS : ENTRADA :			1111				
		act	**bias**				
1	Neurônio 1	1	-0.039520	Saída Desejada 1:	1.000000	Saída Obtida 1:	0.994558
2	Neurônio 2	1	0.197690	Saída Desejada 2:	1.000000	Saída Obtida 2:	1.000000
3	Neurônio 3	1	-0.213520				
4	Neurônio 4	1	-0.570800	PESOS			
5		1.000000	1.600870	-0.07339	2.83024	-7.34133	-4.53884
6		0.999980	-12.278410	-0.71482	5.9031	2.75296	8.57594
7		0.563800	4.912010	0.41064	-29.21679		
8		0.008900	0.222230	8.1078	-3.00536		
9		0.996550	7.455900	23.70916	-36.91986		
10		0.996380	19.892060	-24.52733	-2.71728		

Entrada	1/(1+EXP(-? [PESO*ACT]-bias))	**bias**	PESO * ACT				? [PESO*ACT]
5	0.000540514	1.60087	-4.5388400	-7.3413300	2.8302400	-0.0733900	-9.1233200
6	0.985779807	-12.2784	8.5759400	2.7529600	5.9031000	-0.7148200	16.5171800
Entrada			0.00054051	0.985779807	? [PESO*ACT]		
7	4.21794E-11	4.91201	0.00022196	-28.8013216	-28.80109965		
8	0.060881585	0.22223	0.00438238	-2.9626232	-2.958240824		
Entrada	Saidas		4.2179E-11	0.060881585	? [PESO*ACT]		
9	0.994558	7.4559	1E-09	-2.24773961	-2.247739609		
10	1.000000	19.8921	-1.0345E-09	-0.165432314	-0.165432315		

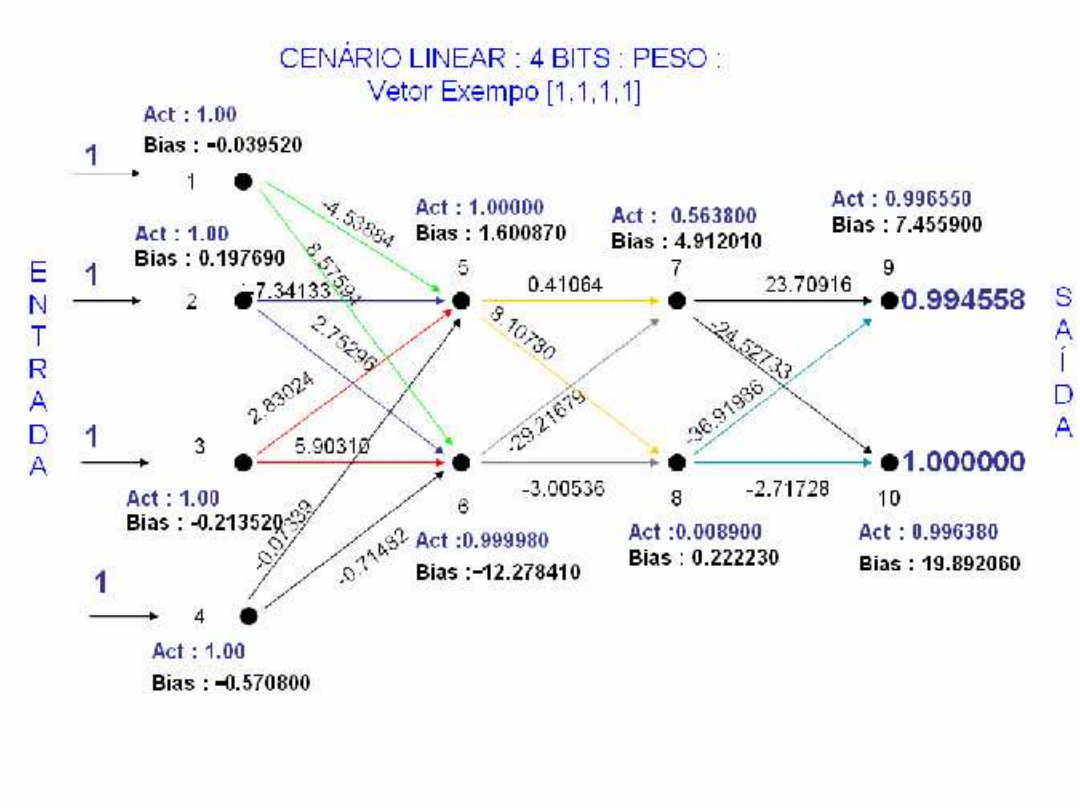

Figura 5.2.4 Disposição de Pesos e Ativações: Rede Unidimensional 4 bits

Tabela 5.2.22 Resultados da Rede Neural 4 Bits

		CLASSES		
4 Bits	Peso	Resultado Desejado BIT 0	BIT 1	Resultado Obtido BIT 0	BIT 1
0000	180	CLASSE A : 0.00000 – 0.00000		0.00289 - 0.00073	
0001	185	CLASSE A : 0.00000 – 0.00000		0.00290 - 0.00073	
0010	190	CLASSE A : 0.00000 – 0.00000		0.00282 - 0.00073	
0011	195	CLASSE A : 0.00000 – 0.00000		0.00283 - 0.00072	
0100	200	CLASSE B : 1.00000 – 0.00000		0.99996 - 0.00252	
0101	205	CLASSE B : 1.00000 – 0.00000		0.99997 - 0.00252	
0110	210	CLASSE B : 1.00000 – 0.00000		0.99931 - 0.00249	
0111	215	CLASSE B : 1.00000 – 0.00000		0.99933 - 0.00220	
1000	220	CLASSE B : 1.00000 – 0.00000		0.99936 - 0.00244	
1001	225	CLASSE B : 1.00000 – 0.00000		0.99937 - 0.00219	
1010	230	CLASSE C : 0.00000 – 1.00000		0.00000 – 1.00000	
1011	235	CLASSE C : 0.00000 – 1.00000		0.00000 – 1.00000	
1100	240	CLASSE C : 0.00000 – 1.00000		0.00997 – 1.00000	
1101	245	CLASSE D : 1.00000 – 1.00000		0.99133 - 0.99223	
1110	250	CLASSE D : 1.00000 – 1.00000		0.99479 – 1.00000	
1111	255	CLASSE D : 1.00000 – 1.00000		0.99456 – 1.00000	

A figura 5.2.4 apresenta a topologia final com os pesos e ativações das conexões entre neurônios para a rede de 4 bits.

Fazendo–se uma análise deste modelo tem-se que para um cenário de 8 bits, existem 256 vetores possíveis de entrada. A resolução de cada vetor é calculada através da diferença entre o valor do limite máximo e limite mínimo. O mapeamento de um vetor contínuo para um vetor discreto, é feito através do cálculo.

Valor Contínuo = Limite Mínimo + (Valor Vetor Binário x Resolução) (5.2.1)

É importante observar nas tabelas 5.2.23 e 5.2.24 de Vetores de entrada que as diversas classes de saída possuem mapeamento de um número de vetores diferentes:

Tabela 5.2.23 Mapeamento Contínuo e Discreto

Contínuo	Discreto	Resolução	Peso [g]	Classes
0	0000	180 + (0 x 5)	180	A
1	0001	180 + (1 x 5)	185	Classe 1
2	0010	180 + (2 x 5)	190	**Binário: 00**
3	0011	180 + (3 x 5)	195	
4	0100	180 + (4 x 5)	200	
5	0101	180 + (5 x 5)	205	B
6	0110	180 + (6 x 5)	210	Classe 2
7	0111	180 + (7 x 5)	215	**Binário: 10**
8	1000	180 + (8 x 5)	220	
9	1001	180 + (9 x 5)	225	
10	1010	180 + (10 x 5)	230	C
11	1011	180 + (11 x 5)	235	Classe 3
12	1100	180 + (12 x 5)	240	**Binário: 01**
13	1101	180 + (13 x 5)	245	D
14	1110	180 + (14 x 5)	250	Classe 4
15	1111	180 + (15 x 5)	255	**Binário: 11**

Tabela 5.2.24. Classes e Vetores

Classe	Vetores de Entrada
00	4
10	6
01	3
11	3

O incremento de um bit seqüencial "ligado" faz com que o vetor de entrada ganhe o incremento de mais uma unidade de resolução, o que permite o mapeamento do vetor binário ao vetor contínuo. O modelo genérico proposto neste trabalho baseia-se nos limites máximo e mínimo de um espaço de padrões e no número de bits de entrada. O processo é simples, analise-se o caso de apenas uma propriedade física:

1. Estabelece-se a faixa de valores:
   - Limite $_{max}$
   - Limite $_{min}$
2. Determina-se o número de bits do canal de entrada
3. Calcula-se a resolução, ou o valor de incremento por intervalo:

$$\text{Resolução} = (L_{max} - L_{min}) / 2^{\text{Número de Bits}} \qquad (5.2.2)$$

4. Cria-se o mapeamento dos vetores binários e contínuos ao vetor binário de saída, através do incremento de um bit no vetor de entrada.

$$\text{Valor Contínuo} = \text{Limite Mínimo} + (\text{Valor Vetor Binário} \times \text{Resolução}) \qquad (5.2.3)$$

5. Aplica-se o espaço vetorial à rede neural pesquisada.

Aplicando-se o algoritmo acima obtêm-se um modelo robusto, independente da propriedade física em estudo e dos valores contínuos implementados neste espaço de valores. Neste trabalho não foi testada a capacidade de generalização das redes, ou seja, todo o espaço amostral foi utilizado, poderia ter sido apresentado para treinamento da rede apenas uma parcialidade do conjunto total de vetores do espaço de padrões, e esperar-se-ia o mesmo comportamento.

Tabela 5.2.25 Cenário Unidimensional

Cenário Unidimensional	Número de Neurônios Camada de entrada	Vetor Exemplo	Cálculo	Número Total de Amostras
4 Bits	4	1010	2^4	16
5 Bits	5	10101	2^5	32
6 Bits	6	101010	2^6	64
7 Bits	7	1010101	2^7	128
8 Bits	8	10101010	2^8	256

Todos os vetores de entrada são unidimensionais, cujos componentes são valores booleanos pertencentes ao intervalo [0,1] representando apenas a entrada de apenas uma propriedade física, Peso.

O espaço amostral/padrões apresentado para treinamento da rede estava dividido em 4 classes, representadas na tabela 5.2.26, todos os vetores binários de entrada convergem para uma das quatro classes binárias.

Tabela 5.2.26 Classes de separação de amostras

CLASSES DE SEPARAÇÃO		
	CLASSE LITERAL	CLASSE BINÁRIA
1	A	00
2	B	10
3	C	01
4	D	11

O Anexo III, apresenta os vetores de entrada do cenário Unidimensional de 8 bits.

Os treinamentos foram realizados com todos os 256 vetores de entrada e respectivas classes desejadas de saída (ANEXO III), apresentados à rede seqüencialmente com a variação de um bit por amostra seqüencial. Os pesos foram inicializados em valores aleatórios. A tabela 5.2.27 apresenta os vetores inferiores e superior de entrada e os vetores de cada classe de saída do cenário unidimensional. A tabela 5.2.27 (ANEXO III) pode ser confrontada com o cenário unidimensional representado na figura 4.2.7, projeção binária/analógica dos valores de vetores para o canal da propriedade física Peso. Como pode ser observado na coluna Vetor Saída Binário Rede Neural da tabela 5.2.27, o valor de saída é igual ao valor esperado, considerando-se a tolerância de 0.01.

Tabela 5.2.27 Representação parcial do espaço vetorial rede neural 8 Bits : cenário unidimensional (ANEXO III) com resultados reais da rede neural treinada.

Entrada	Vetor Entrada : Binário	Vetor Saída : Binário Desejado	Vetor Saída : Binário Rede Neural	Classe	Vetor Entrada : Valor Analógico
0	00000000	0 - 0	0.00011 - 0.00000	A	180
…	…	…	…	A	…
68	01000100	0 - 0	0.00232 - 0.00000	A	199.921875
69	01000101	1 - 0	0.99737 - 0.00000	B	200.2148438
…	…	…	…	B	…
170	10101010	1 - 0	0.99726 - 0.00000	B	229.8046875
171	10101011	0 - 1	0.0000 - 0.99759	C	230.0976563
…	…	…	…	C	…
221	11011101	0 -1	0.00097 - 0.99992	C	244.7460938
222	11011110	1 - 1	0.99663 - 1.00000	D	245.0390625
…	…	…	…	D	…
255	11111111	1 - 1	0.99816 - 1.00000	D	254.7070313

Foram feitos 21 ciclos de treinamento usando variações incrementais de 10.000, 50.000 e 100.000 iterações. A tabela 5.2.28 representa o plano de testes realizados para o cenário unidimensional com 8 bits de entrada, relacionado à análise da propriedade física Peso.

Tabela 5.2.28 Parâmetros de treinamento da rede neural unidimensional

CENÁRIO UNIDIMENSIONAL : 8 BITS							
CICLO	ITERAÇÕES	UPDATE FUNCTION	LEARN FUNCTION	INIT FUNCTION	SSE	MSE	SSE/o-units
1	0	Std_Backpropagation	Topological_Order	Randomize Weights	227.6376	0.88921	113.8188
2	10000	Std_Backpropagation	Topological_Order	Randomize Weights	0.01293	0.00005	0.00647
3	20000	Std_Backpropagation	Topological_Order	Randomize Weights	0.00588	0.00002	0.00294
4	30000	Std_Backpropagation	Topological_Order	Randomize Weights	0.00410	0.00002	0.00205
5	40000	Std_Backpropagation	Topological_Order	Randomize Weights	0.00328	0.00001	0.00164
6	50000	Std_Backpropagation	Topological_Order	Randomize Weights	0.00273	0.00001	0.00137
7	60000	Std_Backpropagation	Topological_Order	Randomize Weights	0.00235	0.00001	0.00117
8	70000	Std_Backpropagation	Topological_Order	Randomize Weights	0.00208	0.00001	0.00104
9	80000	Std_Backpropagation	Topological_Order	Randomize Weights	0.00185	0.00001	0.00093
10	90000	Std_Backpropagation	Topological_Order	Randomize Weights	0.00166	0.00001	0.00083

11	100000	Std_BackpropagationTopological_Order	Randomize Weights	0.00148	0.00001	0.00074
12	110000	Std_BackpropagationTopological_Order	Randomize Weights	0.00133	0.00001	0.00067
13	120000	Std_BackpropagationTopological_Order	Randomize Weights	0.00120	0.00000	0.00060
14	130000	Std_BackpropagationTopological_Order	Randomize Weights	0.00110	0.00000	0.00055
15	140000	Std_BackpropagationTopological_Order	Randomize Weights	0.00102	0.00000	0.00051
16	150000	Std_BackpropagationTopological_Order	Randomize Weights	0.00095	0.00000	0.00048
17	200000	Std_BackpropagationTopological_Order	Randomize Weights	0.00080	0.00000	0.00040
18	300000	Std_BackpropagationTopological_Order	Randomize Weights	0.00068	0.00000	0.00034
19	400000	Std_BackpropagationTopological_Order	Randomize Weights	0.00059	0.00000	0.00030
20	500000	Std_BackpropagationTopological_Order	Randomize Weights	0.00056	0.00000	0.00028
21	600000	Std_BackpropagationTopological_Order	Randomize Weights	0.00054	0.00000	0.00027
22	700000	Std_BackpropagationTopological_Order	Randomize Weights	0.00054	0.00000	0.00027

As figuras 5.2.5, 5.2.6 e 5.2.7 mostram as funções e parâmetros utilizados dentro do simulador SNNS para o treinamento de todas as redes neurais de 4 à 8 bits de entrada:

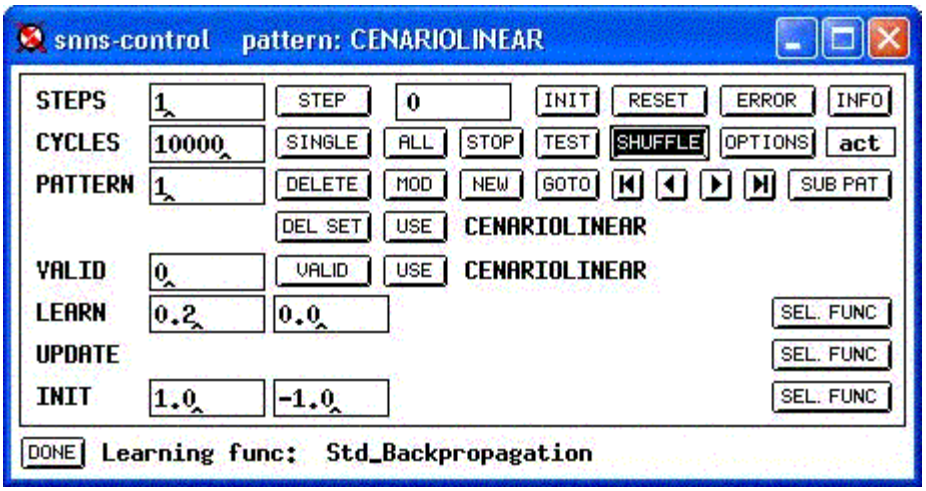

Figura 5.2.5 Função de Aprendizado: Std_Backpropagation

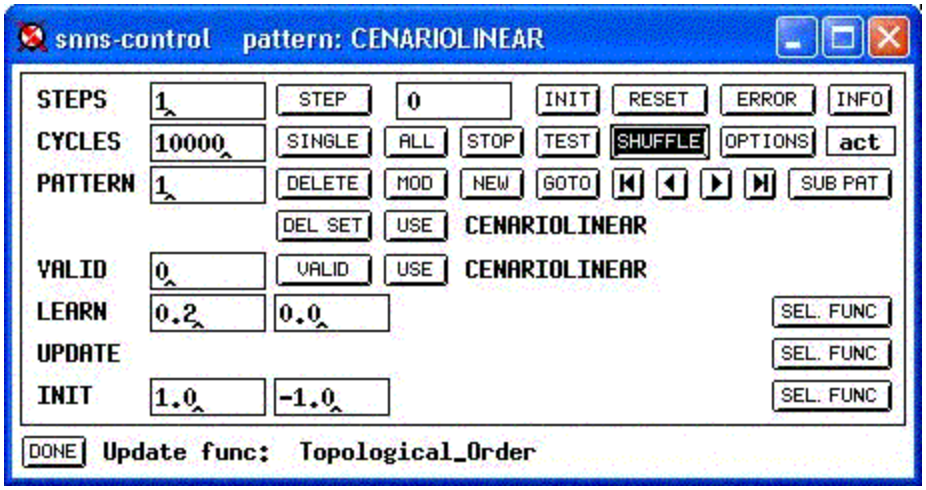

Figura 5.2.6 Função de Atualização: Topological_Order

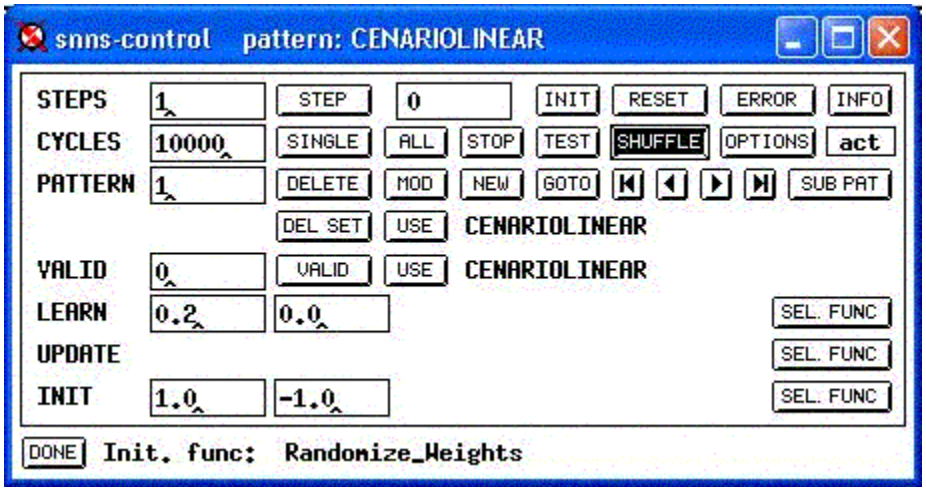

Figura 5.2.7 Função de Inicialização: Randomize_Weights

As figuras 5.2.8 e 5.2.9 apresentam a topologia de rede utilizada para treinamento, no caso unidimensional com 8 bits de entrada.

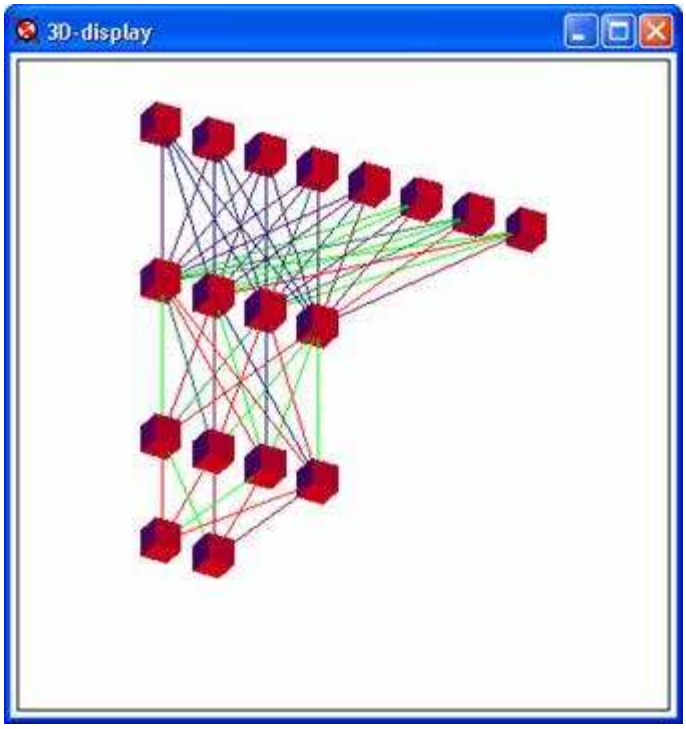

Figura 5.2.8 Topologia de rede unidimensional 8 bits vista em perspectiva

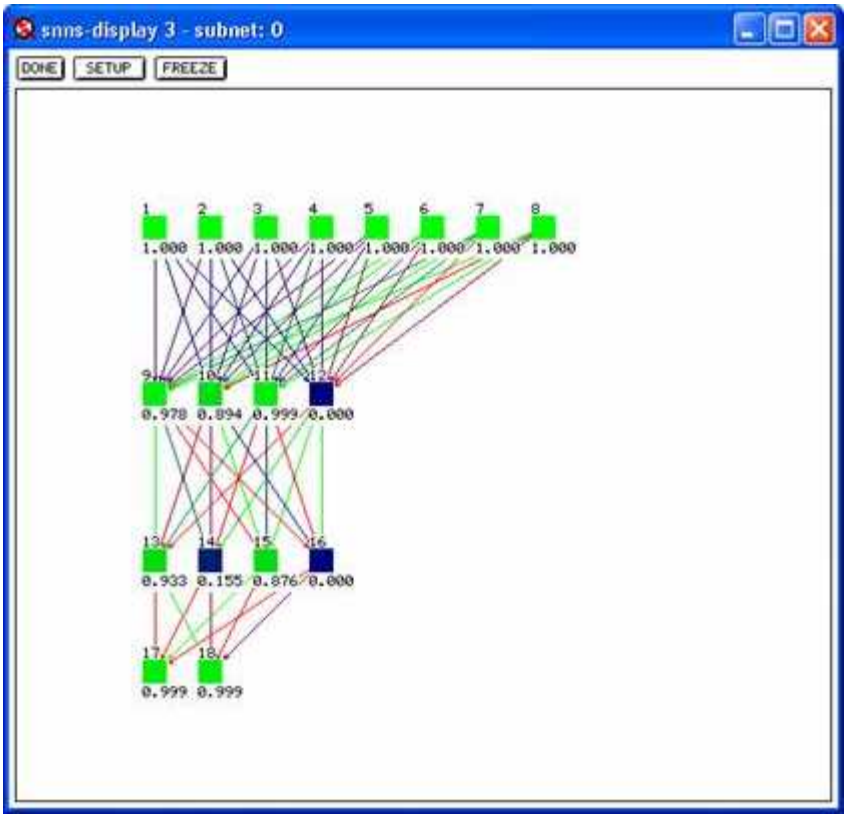

Figura 5.2.9 Topologia de rede unidimensional 8 bits e conexões

Os vetores de entrada utilizados para o cenário unidimensional foram vetores de 8 bits, sendo que os neurônios 1 a 8 representam a entrada referente à uma propriedade física, no caso o Peso. A topologia abaixo representa a disposição dos neurônios e camadas. Foram utilizadas 4 camadas de neurônios:

Tabela 5.2.29 Dados da rede : 8 bits de Entrada

CENÁRIO UNIDIMENSIONAL: 8 BITS : PESO		
	Neurônios	Conexões
Camada de entrada	1 – 8	
Camada de Escondida 1	9 – 14	[camada 1-2] 8x6= 48
Camada de Escondida 2	15 – 18	[camada 2-3] 4x6= 24
Camada de Saída	19 – 20	[camada 3-4] 2x4 = 8
Neurônios	20	
Total Conexões		80

O número de iterações durante o treinamento define o número máximo de vezes que a rede será submetida a massa de testes de forma a ter o erro convergido para uma energia mínima. O gráfico da figura 5.2.10 apresenta o comportamento da curva de erro ou energia mínima para as redes neurais unidimensionais no início do treinamento para 10000 iterações.

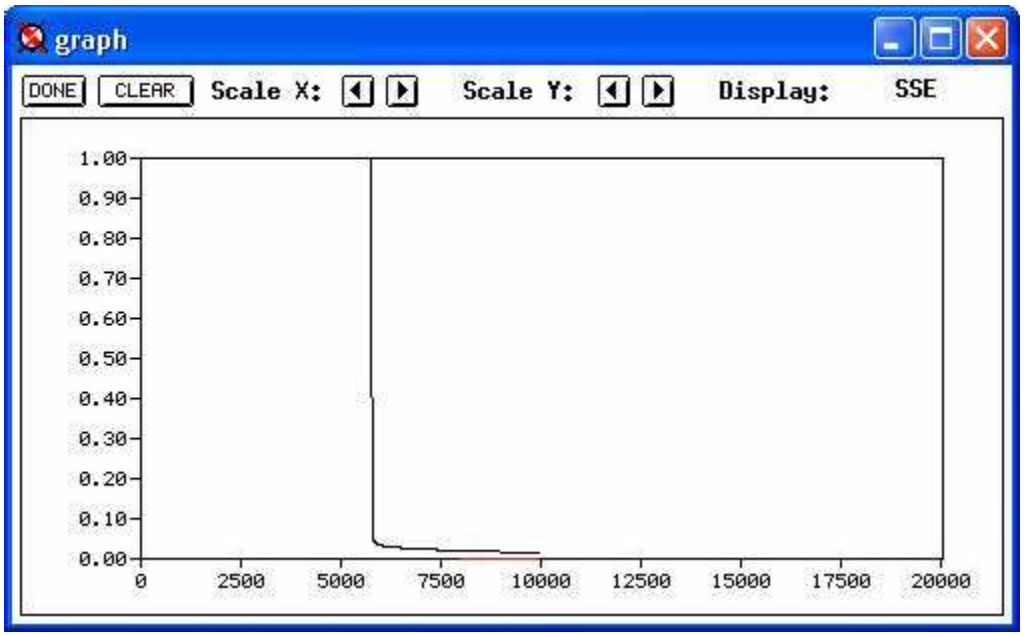

Figura 5.2.10 Gráfico de Redução de Erros: 10.000 iterações

Como pode ser observado no ciclo 2 da tabela 5.2.28, o erro SSE após 10.000 iterações foi reduzido do valor 227.6376 ao valor 0.01293 para o peso inicial. Analisando-se a diferença entre os valores esperados e valores obtidos após o primeiro ciclo de 10.000 iterações, obteve-se o gráfico de erros da Figura 5.2.11, o qual apresenta que os valores calculados encontram-se muito próximos dos valores desejados.

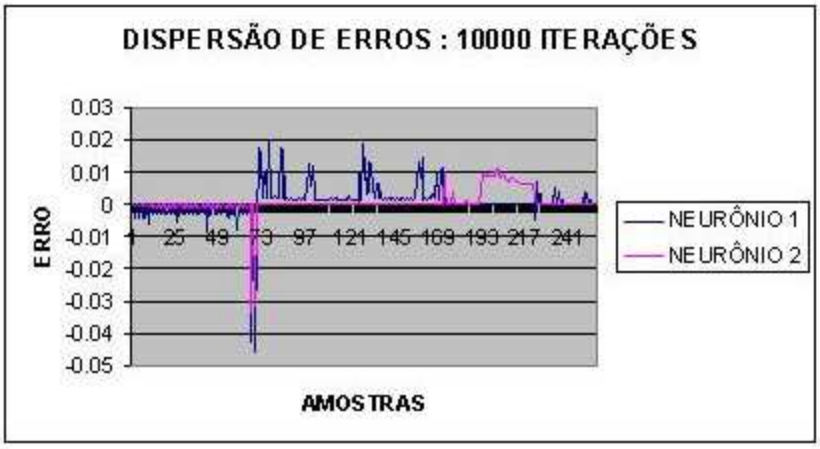

Figura 5.2.11 Gráfico de Dispersão de Erros : 10.000 iterações

A tabela 5.2.28 representa os ciclos de testes realizados. O ciclo 22 com 700.000 iterações aproximou mais ainda os valores dos neurônios de saída com relação aos respectivos valores desejados.

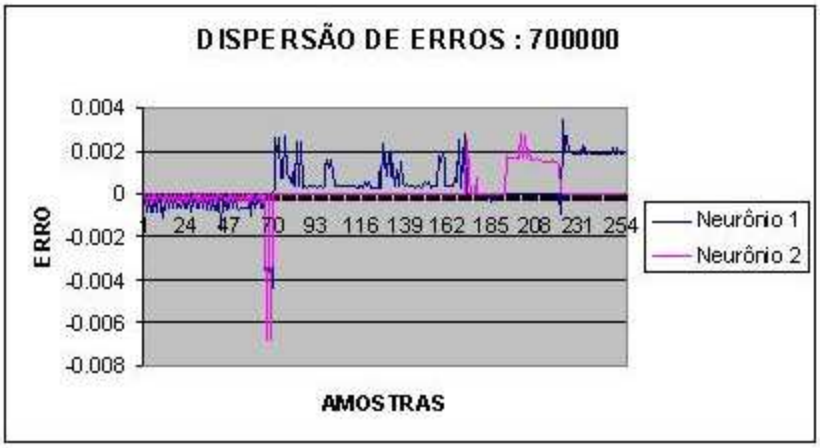

Figura 5.2.12 Gráfico de Dispersão de Erros : 700.000 iterações

A convergência de erros para o ciclo de 700.000 foi reduzida aos valores:

- SSE = 0.00054
- MSE = 0.00000
- SSE /O units = 0.00027

As figuras 5.2.13, 5.2.14 e 5.2.15 apresentam as curvas de convergência dos Erros SSE, MSE, SSE /o-units, respectivamente.

Figura 5.2.13 Gráfico de Convergência de Erros SSE

Figura 5.2.14 Gráfico de Convergência de Erros MSE

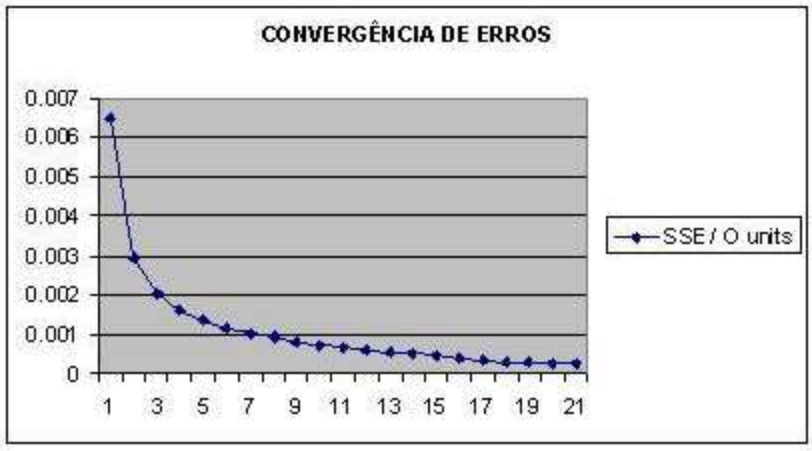

Figura 5.2.15 Gráfico de Convergência de Erros SSE / o-units

Segundo PANDYA e MACY (1995), precisão de uma rede neural é igual à divisão do número resultados de classificação corretas pelo total de número de padrões.

P = Número de classificações corretas/ Número total de padrões (5.2.4)

Supondo-se que as amostras estimadas como corretas fossem resultados que tivessem valor inferior à 0.01 para a diferença entre o valor obtido e o valor desejado, pode-se calcular a precisão para o espaço amostral estudado com a condição exposta:

P = (256)/256

P = 100 %

Como pode ser observado todas as amostras treinadas e testadas atenderam na totalidade o critério de convergência da rede para erros inferiores a 0.01 para a diferença entre os valores desejados e obtidos para as classes de saída.

5.2.1 Arquivo Rede Neural Final : Cenário Unidimensional

Após o treinamento de 700.000 iterações considera-se que a rede neural de classificação unidimensional (1 propriedade física) convergiu para um estado mínimo de energia, ou seja, a rede consegue estabelecer a correta correlação entre os dados de entrada e as classes de saídas desejadas. O pacote de simulação treinou a rede neural e registrou os respectivos pesos para as conexões sinápticas entre neurônios das camadas da rede. A tabela 5.2.33 representa os pesos das

conexões Rede Neural Unidimensional. A tabela 5.2.32 representa a topologia final da Rede Neural Unidimensional.

Tabela 5.2.30 Parâmetros de treinamento Rede Neural Unidimensional : 8 Bits : Propriedade física Peso

Rede Neural de Classificação Unidimensional	
Criação	SNNS network definition file V1.4-3D generated at Thu Oct 11 07:01:52 2001
network name	CENARIOLINEAR8BITS_700000
source files	
no. of units	18
no. of connections	56
no. of unit types	0
no. of site types	0
learning function	Std_Backpropagation
update function	Topological_Order

Tabela 5.2.31 funções de ativação

Act	bias	St	Subnet	layer	act func	out func
0.00000	0.00000	H	0	1	Act_Logistic	Out_Identity

Tabela 5.2.32 Topologia final da Rede Neural Unidimensional : 8 Bits : Propriedade física Peso

Topologia final da Rede Neural Unidimensional 8 Bits : 1 Propriedade : Peso									
No	typeName	unitName	act	Bias	St	position	act func	out func	sites
1			1.00000	0.91364	I	2, 2, 0			
2			1.00000	0.67883	I	3, 2, 0			
3			1.00000	-0.04470	I	4, 2, 0			
4			1.00000	-0.39316	I	5, 2, 0			
5			1.00000	0.80580	I	6, 2, 0			
6			1.00000	0.19831	I	7, 2, 0			
7			1.00000	-0.54686	I	8, 2, 0			
8			1.00000	0.51035	I	9, 2, 0			
9			1.00000	-1.25074	H	2, 5, 0			
10			0.99973	-9.28778	H	3, 5, 0			
11			1.00000	-7.92518	H	4, 5, 0			
12			0.00000	35.32094	H	5, 5, 0			
13			1.00000	2.46622	H	6, 5, 0			
14			0.00010	5.20898	H	7, 5, 0			
15			0.99997	0.14815	H	2, 8, 0			
16			1.00000	0.65057	H	3, 8, 0			
17			0.99816	16.10884	O	4, 8, 0			
18			0.00000	-0.80236	O	5, 8, 0			

Tabela 5.2.33 Pesos das conexões Rede Neural Unidimensional : 8 Bits : Propriedade física Peso

target	Site	source:weight
9		1: 9.57372, 2:-6.19242, 3:-0.63935, 4: 0.67699, 5:-8.38535, 6: 5.50138, 7: 2.45965, 8:13.00764
10		1: 9.45475, 2:10.06520, 3:-0.62813, 4:-0.58254, 5:-0.32524, 6:-0.57055, 7:-0.00188, 8: 0.08468
11		1: 5.58222, 2: 6.87108, 3: 1.56449, 4: 2.62319, 5: 1.73140, 6:-0.05866, 7: 0.00315, 8:10.31635
12		1:-17.35323, 2:-7.07625, 3:-12.38412, 4:-5.31785, 5:-3.36954, 6:-2.19102, 7:-1.25513, 8:-0.40013
13		9: 1.75380, 10:17.99079, 11: 8.64478, 12:-25.57738
14		9:-4.03555, 10:-5.03476, 11:-5.32377, 12:22.89824
15		9:-6.91146, 10:18.67348, 11:-1.38477, 12: 6.15153
16		9:-7.19081, 10:-1.24167, 11:-12.71369, 12: 5.60854
17		13:-27.38999, 14:-25.50277, 15:17.57958, 16:-17.34503
18		13:22.23100, 14:-5.97479, 15:-8.93271, 16:-2.70833

5.3 Resultados Classificação Bidimensional

Nos testes feitos para o cenário bidimensional o software de simulação SNNS 4.1 foi ajustado para treinamento das redes utilizando-se das funções de aprendizado tipo Backpropmomentum, atualização tipo Topological_Order e inicialização tipo Randomize_Weights.

Todos os vetores de entrada são compostos de 2 vetores unidimensionais de 4 bits cada um, cujos componentes são valores booleanos pertencentes ao intervalo [0,1] representando a entrada de duas propriedades físicas, Peso e Diâmetro.

O espaço amostral/padrões apresentado para treinamento da rede estava dividido em 4 classes, representadas na tabela 5.3.1, todos os vetores binários de entrada convergem para uma das quatro classes binárias. Assim como no cenário unidimensional, o cenário bidimensional também usou 4 classes, porém o espaço amostral/padrões considera a interação das 2 propriedades físicas do fruto, Peso e Diâmetro, como pode ser observado na figura 5.3.1.

Tabela 5.3.1 Classes de separação de amostras

	CLASSE LITERAL	CLASSE BINÁRIA
1	A	00
2	B	10
3	C	01
4	D	11

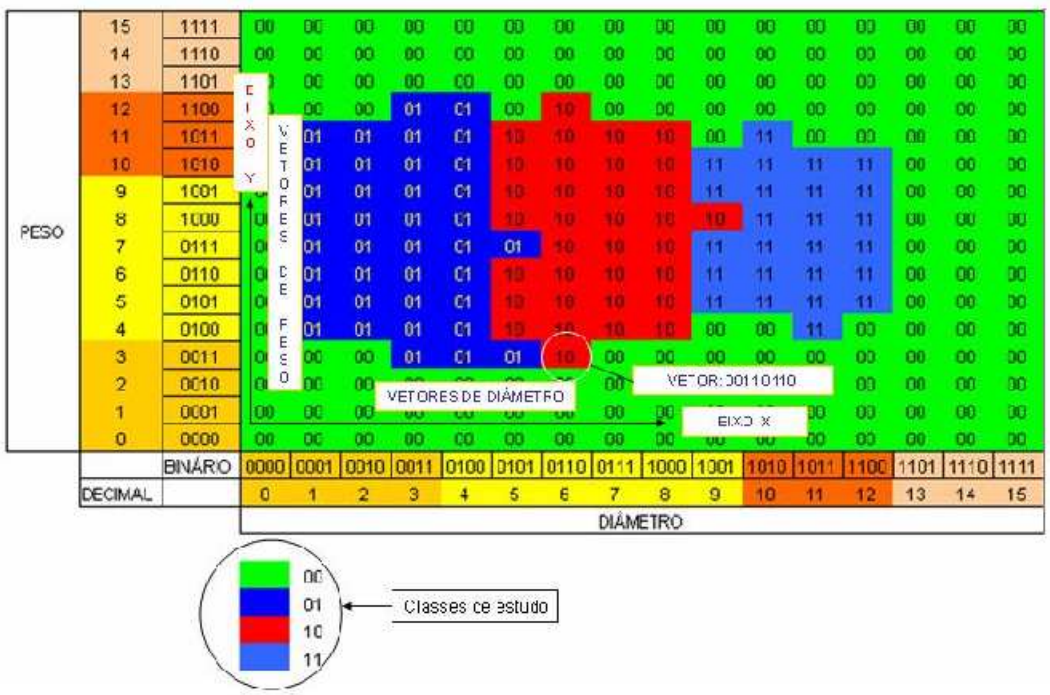

Figura 5.3.1 Espaço Amostral / Padrões

Os treinamentos foram realizados com todos os 256 vetores de entrada (2^8) e respectiva classes desejadas de saída (Espaço vetorial - Figura 5.3.1), apresentados à rede seqüencialmente com a variação de um bit seqüencial por amostra. Os pesos foram inicializados em valores aleatórios.

Foram feitos 15 ciclos de treinamento usando variações incrementais de 10.000, 50.000 e 100.000 iterações. A tabela 5.3.3. representa o plano de testes realizados para o cenário bidimensional.

Foram utilizados vetores de 8 Bits para as entradas bidimensionais, sendo que os neurônios 1 a 4 representam a entrada referente à propriedade física Peso e os neurônios 5 a 8 representam a entrada referente à propriedade física Diâmetro. A topologia da tabela 5.3.2 representa a disposição dos neurônios e camadas. Foram utilizadas 4 camadas de neurônios:

Tabela 5.3.2 Classes de separação de amostras

CENÁRIO BIDIMENSIONAL : 8 BITS : PESO + DIÂMETRO		
	Neurônios	Conexões
Camada de entrada	1 – 8	
Camada de Escondida 1	9 – 14	[camada 1-2] 8x6=48
Camada de Escondida 2	15 – 18	[camada 2-3] 4x6= 24
Camada de Saída	19 – 20	[camada 3-4] 2x4 = 8
Neurônios	20	
Total Conexões		80

Observa-se na figura 5.3.3 que a topologia das camadas escondidas 1 e 2 tiveram o número de neurônios alterados quando comparado ao cenário unidimensional. Foi necessário um número maior de conexões para o armazenamento do relacionamento entre os vetores de entrada e classes de saída, isto ocorreu devido a variação do espaço de padrões. Neste espaço vetorial existem 3 “buracos” diferentes circunscritos dentro do espaço global.

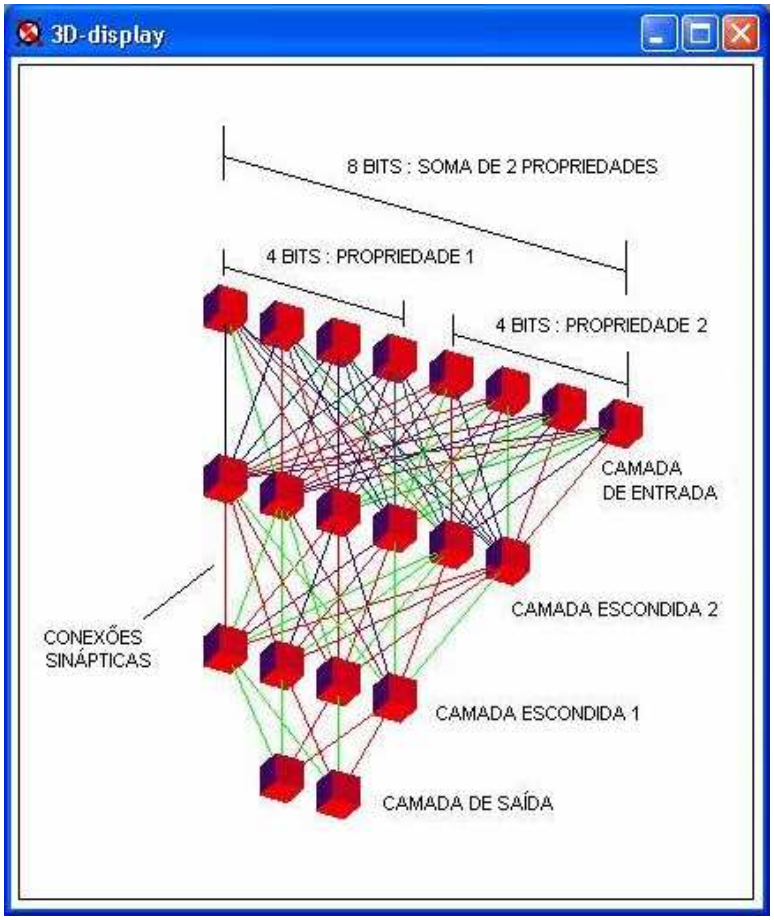

Figura 5.3.2 Topologia de rede Bidimensional 8 bits vista em perspectiva

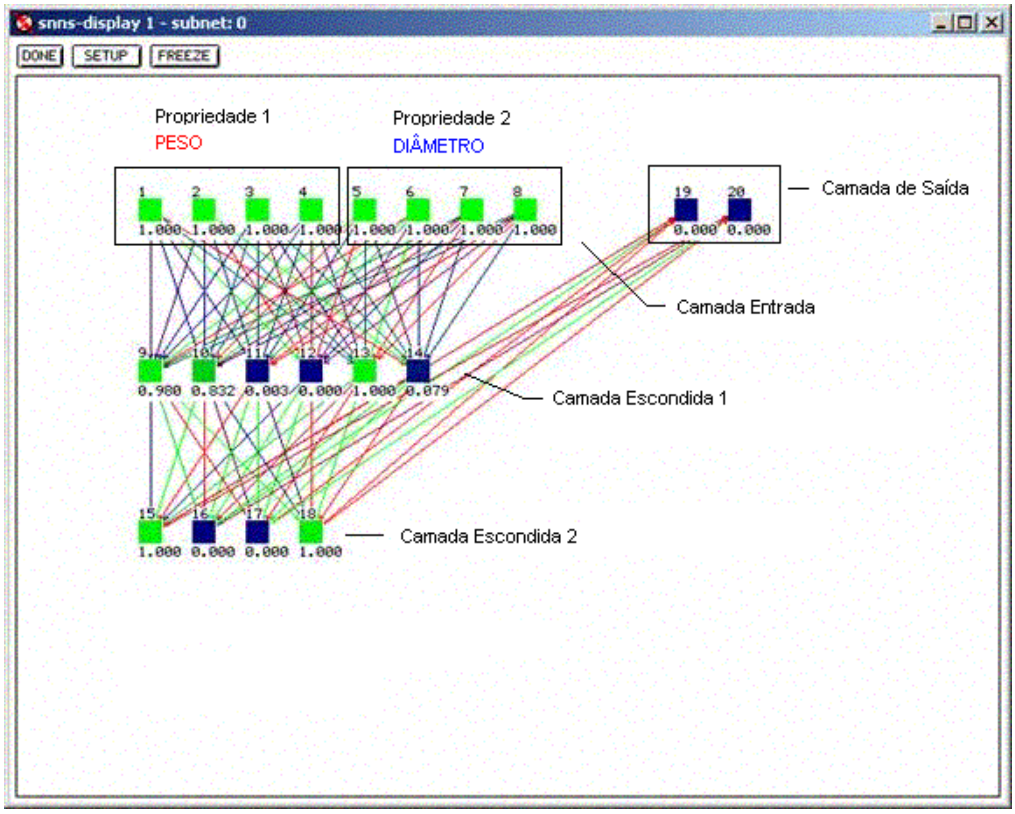

Figura 5.3.3 Topologia de rede Bidimensional 8 bits e conexões.

Tabela 5.3.3 Parâmetros de treinamento da rede neural Bidimensional

CENÁRIO BIDIMENSIONAL : 8 BITS : 4 BITS DE PESO E 4 BITS DE DIÂMETRO							
TESTE	ITERAÇÕES	UPDATE FUNCTION	LEARN FUNCTION	INIT FUNCTION	SSE	MSE	SSE/o-units
1	0	Backpropmomentum	Topological_Order	Randomize Weights	96.30925	0.37621	48.15462
2	10000	Backpropmomentum	Topological_Order	Randomize Weights	1.03924	0.00406	0.51962
3	20000	Backpropmomentum	Topological_Order	Randomize Weights	1.01502	0.00396	0.50751
4	30000	Backpropmomentum	Topological_Order	Randomize Weights	0.01276	0.00005	0.00638
5	40000	Backpropmomentum	Topological_Order	Randomize Weights	0.00831	0.00003	0.00415
6	50000	Backpropmomentum	Topological_Order	Randomize Weights	0.00642	0.00003	0.00321
7	60000	Backpropmomentum	Topological_Order	Randomize Weights	0.00528	0.00002	0.00264
8	70000	Backpropmomentum	Topological_Order	Randomize Weights	0.00452	0.00002	0.00226
9	80000	Backpropmomentum	Topological_Order	Randomize Weights	0.00395	0.00002	0.00198
10	90000	Backpropmomentum	Topological_Order	Randomize Weights	0.00351	0.00001	0.00175
11	100000	Backpropmomentum	Topological_Order	Randomize Weights	0.00315	0.00001	0.00157
12	150000	Backpropmomentum	Topological_Order	Randomize Weights	0.00206	0.00001	0.00103
13	200000	Backpropmomentum	Topological_Order	Randomize Weights	0.00154	0.00001	0.00077
14	300000	Backpropmomentum	Topological_Order	Randomize Weights	0.00108	0.00000	0.00054
15	400000	Backpropmomentum	Topological_Order	Randomize Weights	0.00087	0.00000	0.00044

O gráfico da figura 5.3.4. apresenta o comportamento da curva de erro ou energia mínima para as redes neurais bidimensionais no início do treinamento para 30.000 iterações. A convergência de erro diminuiu significativamente como pode ser observado, porém 30.000 iterações não foram suficientes para aproximar o máximo possível de zero.

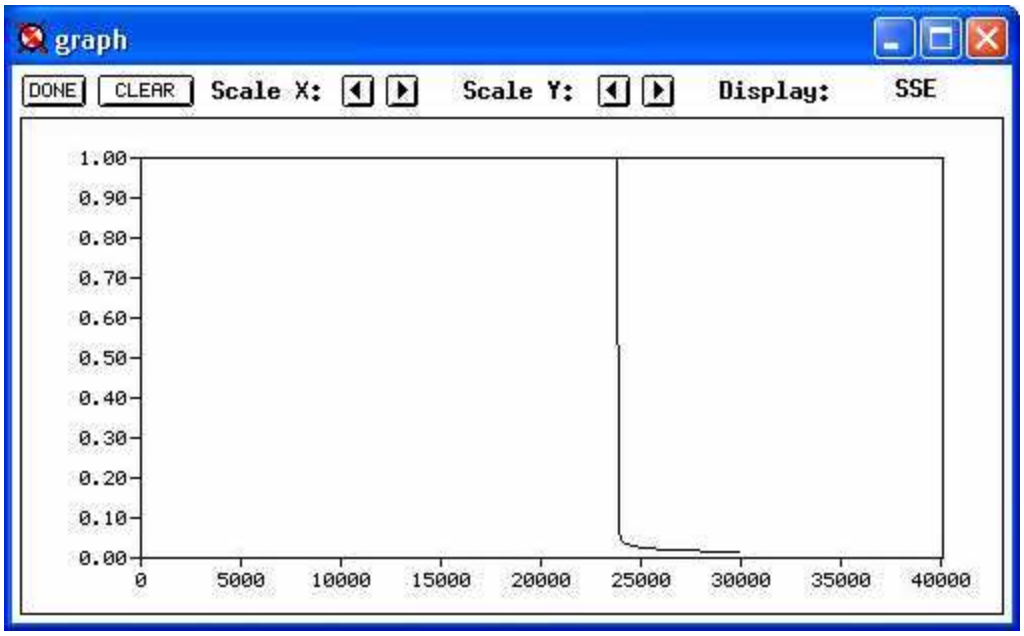

Figura 5.3.4 Gráfico de Redução de Erros : 30.000 iterações

Como pode ser observado no ciclo 4 da tabela 5.3.3, o erro SSE após 30000 iterações foi reduzido do valor 96.30925 ao valor 0.00831 para o peso inicial.Analisando-se a diferença entre os valores esperados e valores obtidos após o ciclo 4 de 30000 iterações, obteve-se o gráfico de erros da Figura 5.3.4, o qual apresenta que os valores calculados encontram-se muito próximos dos valores desejados.

Apesar do valor MSE para ciclo 3 (0.00396) ter caído significativamente no ciclo 4 (0.00005), este valor poderia ser zerado com um número maior de iterações, sendo reduzido a zero para o ciclo 15 de 400.000 iterações.

A convergência de erros para o ciclo de 400.000 foi reduzida aos valores:

- SSE = 0.00087
- MSE = 0.00000
- SSE /O units = 0.00044

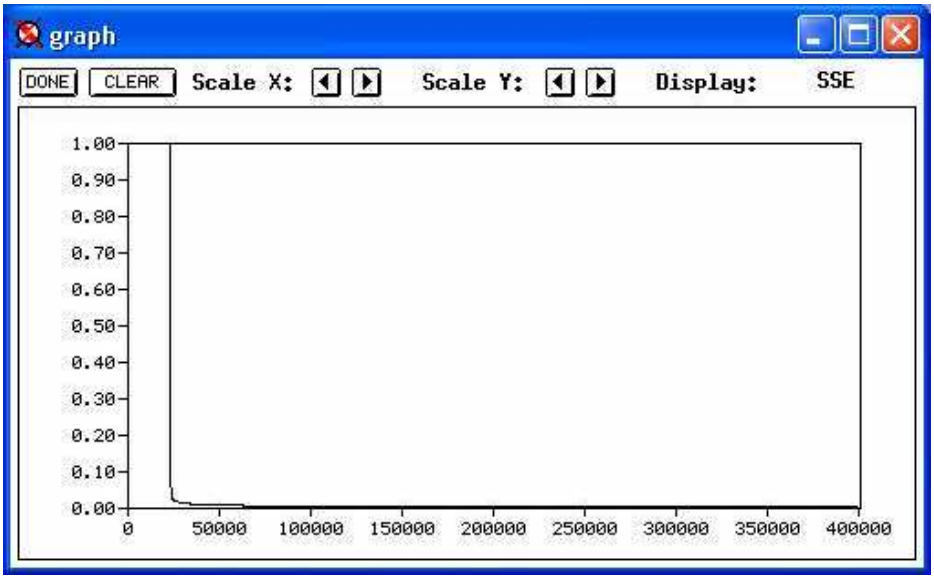

Figura 5.3.5 Gráfico de Redução de Erros : 400.000 iterações

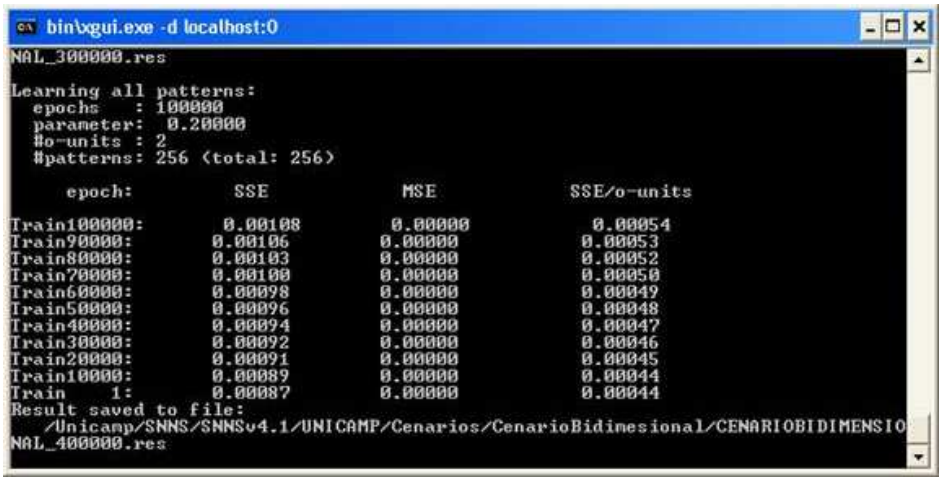

Figura 5.3.6 Convergência de Erros : 400.000 iterações

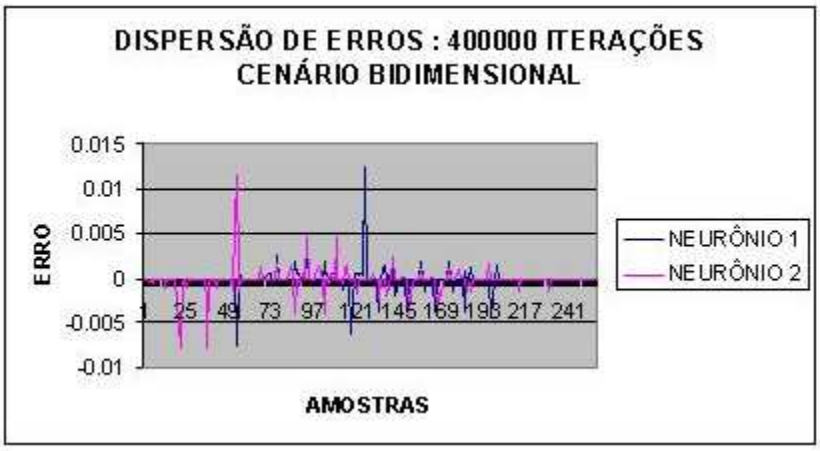

Figura 5.3.7 Gráfico de Dispersão de Erros : 400.000 iterações

Segundo os resultados obtidos para o treinamento do espaço amostral da figura 5.3.1 apenas duas amostras tiveram resultados com diferença entre resultado obtido e desejado, com valor de diferença superior 0.01.

Supondo-se que as amostras estimadas como corretas fossem resultados que tivessem diferença entre o valor obtido e o valor desejado inferior à 0.01, a precisão para o espaço amostral estudado seria:

$$P = (256-2)/256 \quad -> \quad P = 0.9921875 \text{ ou } 99.22\ \%$$

As figuras 5.3.8, 5.3.9 e 5.3.10 apresentam as curvas de convergência dos Erros SSE, MSE, SSE /o-units, respectivamente.

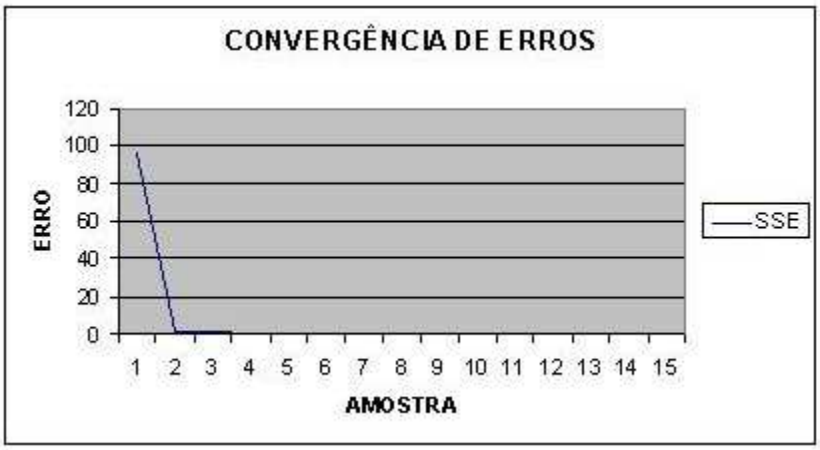

Figura 5.3.8 Gráfico de Convergência de Erros SSE

Figura 5.3.9 Gráfico de Convergência de Erros MSE

Figura 5.3.10 Gráfico de Convergência de Erros SSE / o-units

5.3.1 Arquivo Rede Neural Final : Cenário Bidimensional

Após o treinamento de 400.000 iterações considera-se que a rede neural de classificação bidimensional convergiu para um estado mínimo de energia, ou seja, a rede consegue estabelecer a correta correlação entre os dados de entrada e as classes de saídas desejadas. O pacote de simulação treinou a rede neural e registrou os respectivos pesos para as conexões sinápticas entre neurônios das camadas da rede. A tabela 5.3.7 representa os pesos das conexões Rede Neural Bidimensional. A tabela 5.3.6 representa a topologia final da Rede Neural Bidimensional.

Tabela 5.3.4 Parâmetros de treinamento Rede Neural Bidimensional : 8 Bits : Propriedades física Peso + Diâmetro

Rede Neural de Classificação Bidimensional	
Criação	SNNS network definition file V1.4-3D generated at Fri Oct 12 22:42:15 2001
network name	CENARIOBIDIMENSIONAL_400000
source files	
no. of units	20
no. of connections	80
no. of unit types	0
no. of site types	0
learning function	BackpropMomentum
update function	Topological_Order

Tabela 5.3.5 funções de ativação

Act	bias	st	subnet	layer	act func	out func
0.00000	0.00000	h	0	1	Act_Logistic	Out_Identity

Tabela 5.3.6 Topologia final da Rede Neural Bidimensional : 8 Bits : Propriedades física Peso + Diâmetro

Topologia final da Rede Neural Bidimensional : 8 Bits : 2 Propriedade : Peso + Diâmetro									
no	typeName	unitName	act	bias	St	position	act func	out func	sites
1			1.00000	-0.29962	I	2, 2, 0			
2			1.00000	-0.79700	I	3, 2, 0			
3			1.00000	-0.18923	I	4, 2, 0			
4			1.00000	0.22120	I	5, 2, 0			
5			1.00000	0.06005	I	6, 2, 0			
6			1.00000	0.62085	I	7, 2, 0			
7			1.00000	0.74703	I	8, 2, 0			
8			1.00000	0.06398	I	9, 2, 0			
9			1.00000	-9.04183	H	2, 5, 0			
10			0.99999	0.20076	H	3, 5, 0			
11			0.00745	-7.61893	H	4, 5, 0			
12			1.00000	-1.80096	H	5, 5, 0			
13			1.00000	2.02125	H	6, 5, 0			
14			0.00016	2.85508	H	7, 5, 0			
15			1.00000	-9.94043	H	2, 8, 0			
16			1.00000	0.67055	H	3, 8, 0			
17			0.99949	-3.50147	H	4, 8, 0			
18			0.00000	-0.27578	H	5, 8, 0			
19			0.00015	8.04456	O	3,10, 0			
20			0.00000	16.83603	O	4,10, 0			

Tabela 5.3.7 Pesos das conexões Rede Neural Bidimensional : 8 Bits : Propriedades física Peso + Diâmetro.

target	Site	source:weight
9		1: 5.05945, 2: 2.58952, 3:-1.23175, 4:-1.23182, 5: 5.68646, 6:-1.37180, 7:11.41168, 8: 7.94388
10		1:12.36235, 2:-7.72343, 3:-0.79612, 4:-0.79614, 5: 6.57854, 6: 1.80413, 7: 0.27871, 8: 0.29926
11		1: 2.26721, 2:-2.19424, 3:-3.30388, 4:-3.30610, 5: 4.20643, 6: 1.11395, 7: 3.12797, 8: 0.81557
12		1:-0.40597, 2:-1.43221, 3:-0.75859, 4:-0.75828, 5: 4.43500, 6:11.60745, 7: 0.86340, 8: 1.29106
13		1:-17.10299, 2: 9.30590, 3: 3.67814, 4: 3.67824, 5: 6.04098, 6: 4.13640, 7:-0.36599, 8: 2.61904
14		1:-4.49445, 2:-5.53412, 3: 2.52011, 4: 2.52034, 5:-2.94933, 6:-12.74216, 7: 5.03627, 8: 4.04919
15		9: 4.68756, 10:19.43124, 11: 4.23893, 12:-16.06325, 13:14.28514, 14:-4.61801
16		9:-7.85008, 10:14.29739, 11:-3.02678, 12: 1.52530, 13:10.36193, 14:-10.83214
17		9:-6.08315, 10:14.99747, 11:-1.17572, 12: 0.64636, 13: 1.53432, 14:-1.46559
18		9:-16.39106, 10:-3.20392, 11: 6.79164, 12: 9.22567, 13:-7.32436, 14: 2.01263
19		15:-29.77777, 16:-0.40081, 17:13.33521, 18:-26.59345
20		15:-10.49416, 16:-25.27825, 17: 3.49498, 18:33.85873

5.4 Resultados Classificação Tridimensional

A tabela 5.4.2 apresenta o plano de treinamento realizado para o cenário tridimensional. Os testes realizados para este cenário, utilizando-se do pacote de simulação SNNS 4.1, foram ajustados para treinamento das redes utilizando-se as funções de aprendizado tipo Std_Backpropagation, atualização tipo Topological_Order e inicialização tipo Randomize_Weights. Foram feitos ciclos de treinamento usando 10.000, 20.000, 30.000, 40.000, 50.000 iterações.

Foram utilizados vetores de 12 bits para as entradas tridimensionais, sendo que os neurônios 1 a 4 representam a entrada referente à propriedade física Peso, os neurônios 5 a 8 representam a entrada referente à propriedade física Diâmetro e finalmente os neurônios 9 a 12 representam a propriedade física Firmeza ("*Firmness*"). A topologia da tabela 5.4.3 representa a disposição dos neurônios e camadas para o cenário tridimensional.

Todos os vetores de entrada são compostos de 3 vetores unidimensionais de 4 bits cada um, cujos componentes são valores booleanos pertencentes ao intervalo [0,1] representando a entrada de três propriedades físicas, Peso, Diâmetro e Firmeza ("*Firmness*").

O espaço amostral/padrões apresentado para treinamento da rede estava dividido em 4 classes, representadas na tabela 5.4.1, todos os vetores binários de entrada convergem para uma das quatro classes binárias. Assim como nos cenários unidimensional e bidimensional, o cenário

tridimensional também usou 4 classes, porém o espaço amostral/padrões considera a interação de 3 propriedades.

Tabela 5.4.1 Classes de separação de amostras

	CLASSE LITERAL	CLASSE BINÁRIA
1	A	00
2	B	10
3	C	01
4	D	11

Os treinamentos foram realizados com todos os 4096 vetores de entrada (Espaço vetorial Figura 4.6.4 à Figura 4.6.19) e respectiva classe desejada de saída, apresentados à rede seqüencialmente com a variação de um bit por amostra seqüencial. Os pesos foram inicializados em valores aleatórios.

Foram feitos 5 ciclos de treinamento usando variações incrementais de 10.000 iterações. A tabela 5.4.2 representa o plano de testes realizados para o cenário tridimensional.

Tabela 5.4.2 Parâmetros de treinamento da rede neural tridimensional

CENÁRIO TRIDIMENSIONAL : 12 BITS : 4 BITS DE PESO + 4 BITS DE DIÂMETRO + 4 BITS DE "*FIRMNESS*"							
TESTE	ITERAÇÕES	UPDATE FUNCTION	LEARN FUNCTION	INIT FUNCTION	SSE	MSE	SSE/o-units
1	0	Std_Backpropagation	Topological_Order	Randomize Weights	2.93713	0.00072	1.46857
2	10000	Std_Backpropagation	Topological_Order	Randomize Weights	0.01039	0.00000	0.00520
3	20000	Std_Backpropagation	Topological_Order	Randomize Weights	0.00513	0.00000	0.00256
4	30000	Std_Backpropagation	Topological_Order	Randomize Weights	0.00340	0.00000	0.00170
5	40000	Std_Backpropagation	Topological_Order	Randomize Weights	0.00261	0.00000	0.00130
6	50000	Std_Backpropagation	Topological_Order	Randomize Weights	0.00218	0.00000	0.00109

As figuras 101 e 102 apresentam a topologia de rede utilizada para treinamento no caso tridimensional com 12 bits de entrada.

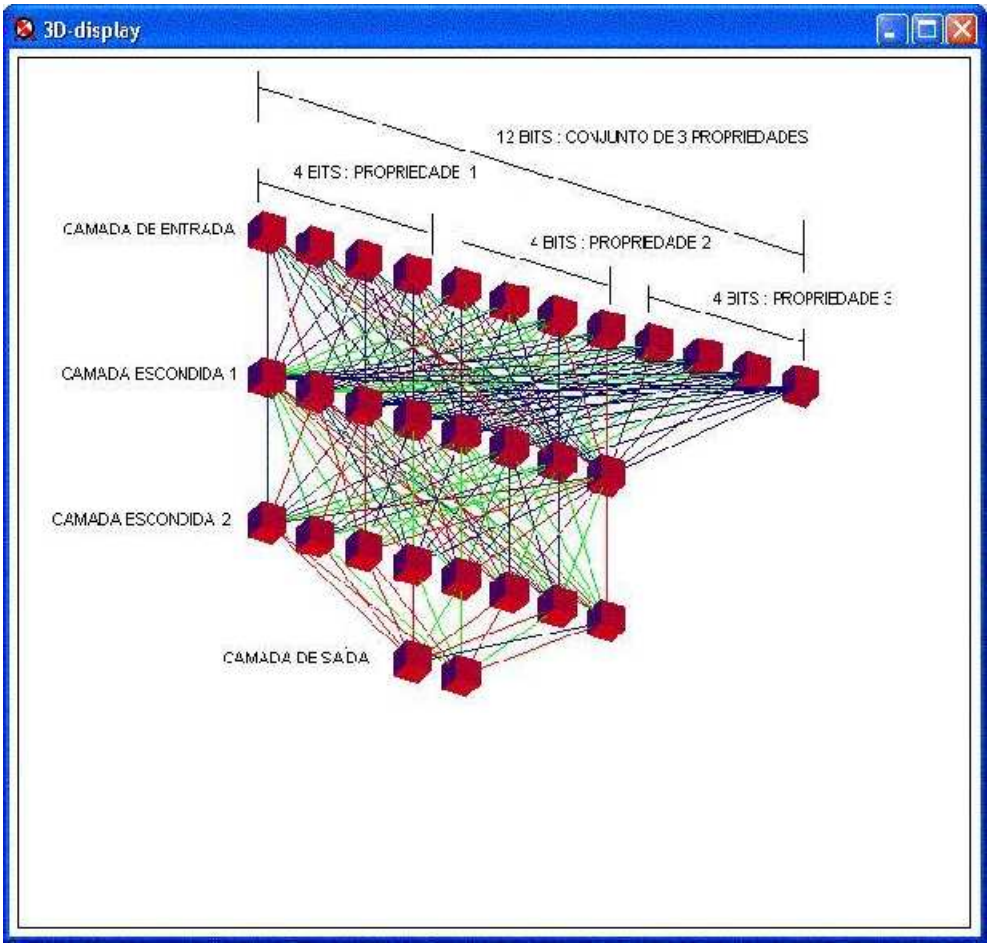

Figura 5.4.1 Topologia de rede tridimensional 12 Bits vista em perspectiva

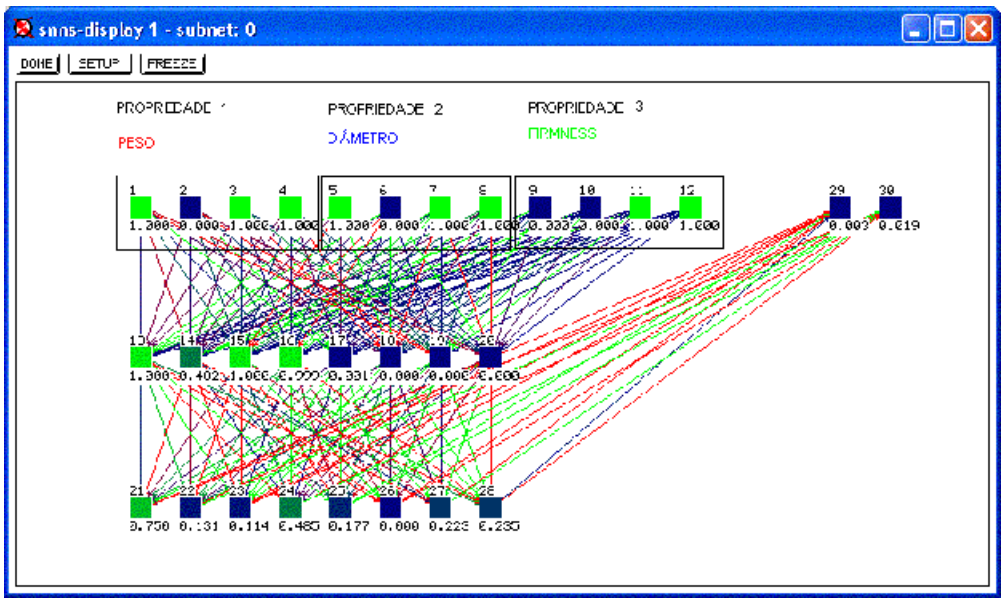

Figura 5.4.2 Topologia de rede tridimensional 12 bits e conexões

Foram utilizados vetores de 12 bits para a entrada tridimensional, sendo que os neurônios 1 a 4 representam a entrada referente à propriedade física Peso, os neurônios 5 a 8 representam a entrada referente à propriedade física Diâmetro e os neurônios 9 a 12 representam a entrada da propriedade física "*Firmness*". A topologia abaixo representa a disposição dos neurônios e camadas. Foram utilizadas 4 camadas:

Tabela 5.4.3 Dados da rede : 12 bits de Entrada

CENÁRIO TRIDIMENSIONAL: 12 BITS : PESO + DIÂMETRO + FIRMNESS		
	Neurônios	Conexões
Camada de entrada	1 - 12	
Camada de Escondida 1	13 - 20	[camada 1-2] 12x8=96
Camada de Escondida 2	21 - 28	[camada 2-3] 8x8= 64
Camada de Saída	29 - 30	[camada 3-4] 2x8 = 16
Neurônios	30	
Total Conexões		176

O gráfico da figura 5.4.3 apresenta o comportamento da curva de erro ou energia mínima para as redes neurais no início do treinamento para 10000 iterações.

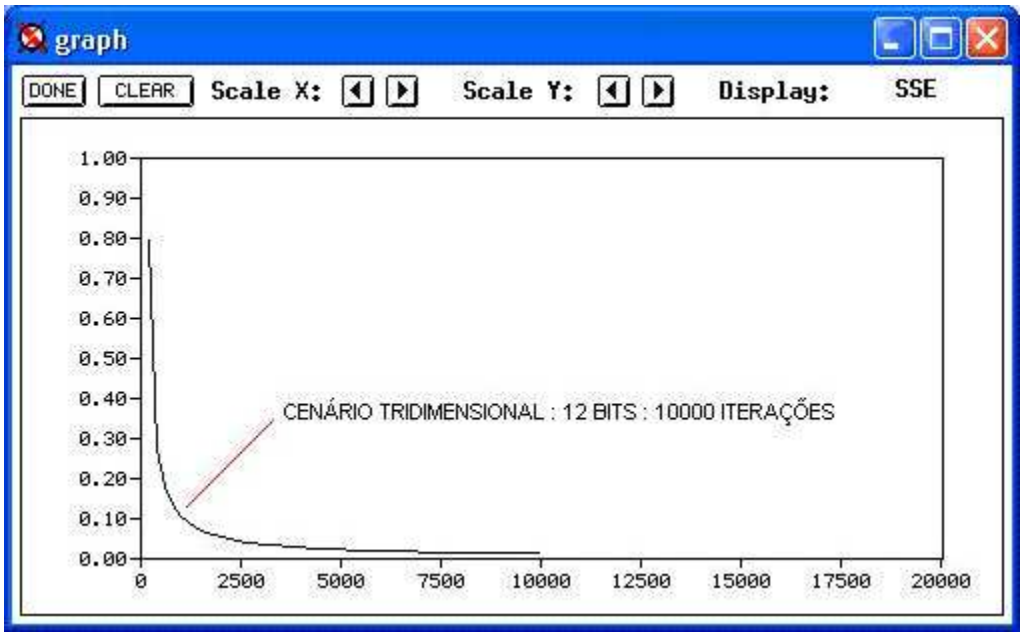

Figura 5.4.3 Gráfico de Redução de Erros : 10.000 iterações

As figuras 5.4.4, 5.4.5 e 5.4.6 mostram as funções e parâmetros utilizados para o treinamento de todas as redes neurais de 12 bits de entrada:

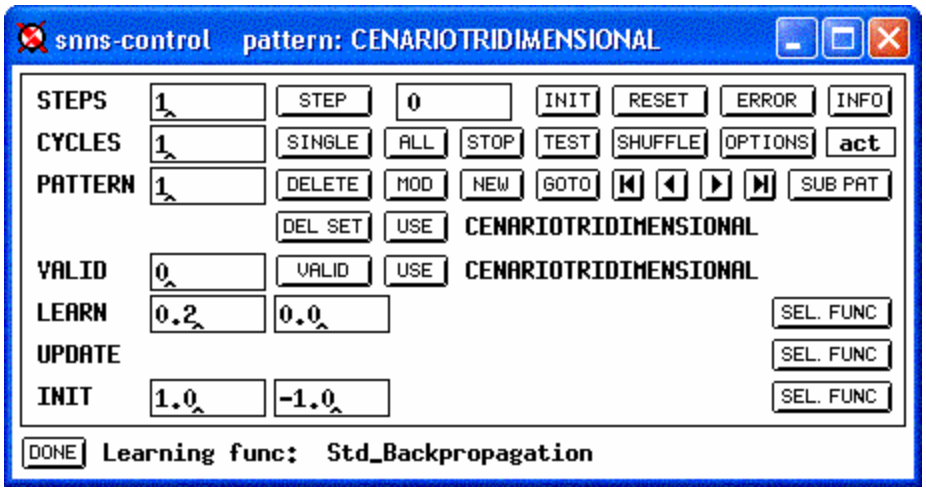

Figura 5.4.4 Função de Aprendizado: Std_Backpropagation

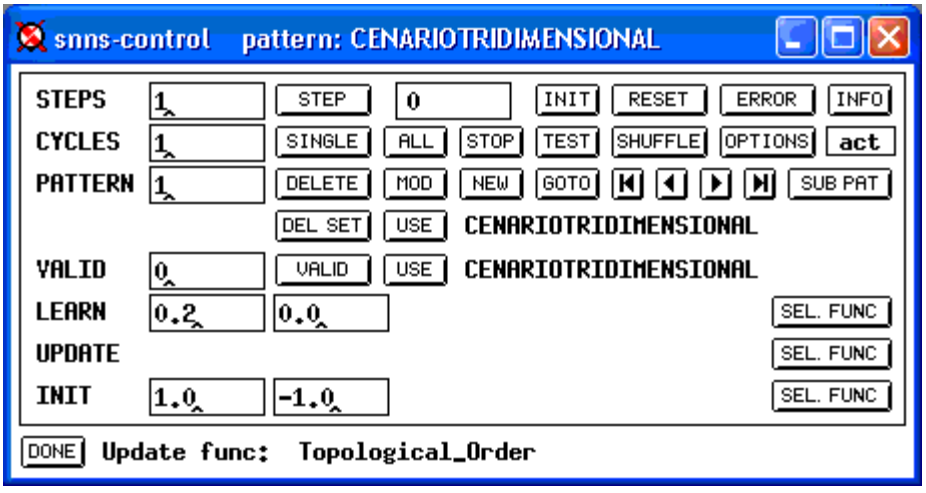

Figura 5.4.5 Função de Atualização: Topological_Order

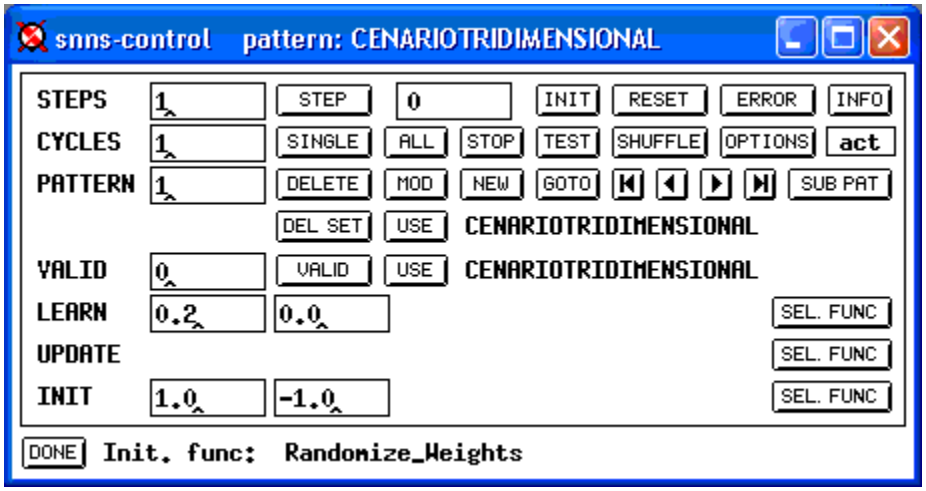

Figura 5.4.6 Função de Inicialização: Randomize_Weights

Como pode ser observado no ciclo 2 da tabela 5.4.2, o erro SSE após 10.000 iterações foi reduzido do valor 2.93713 ao valor 0.01039 para os pesos iniciais. O gráfico da Figura 5.4.3 representa a curva de convergência de Erro SSE para o cenário tridimensional com 10.000 iterações.

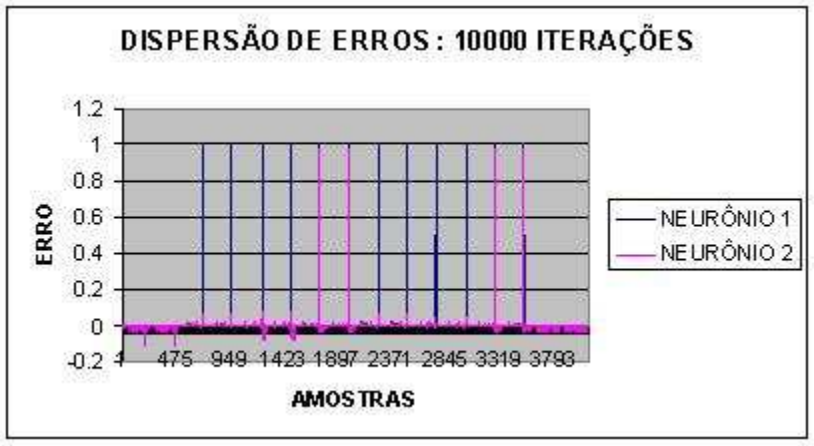

Figura 5.4.7 Gráfico de Dispersão de Erros : 10.000 iterações

Analisando-se a diferença entre os valores esperados e valores obtidos após o ciclo 2 de 10000 iterações, obteve-se o gráfico de erros da Figura 5.4.7, o qual apresenta que os valores calculados encontram-se muito próximos dos valores desejados, porém observa-se que existem

alguns valores obtidos que fogem totalmente dos valores desejados, representados pelos picos de erro com valor igual a 1, indicando a necessidade de um número maior de iterações.

Calculando-se a precisão para este estado da rede e analisando-se apenas os erros com valor igual a 1, observa-se que 14 amostras desviam totalmente do valor desejado, ou seja, espera-se o valor zero e o resultado obtido foi o valor 1 e vice-versa.

Logo a precisão para 10.000 iterações é :

Precisão = (4096-14)/4096 -> Precisão = 99.65 %

As figuras 5.4.8, 5.4.9, 5.4.10 e 5.4.11 representam a dispersão de erros para os ciclos iterativos 20000, 30000, 40000 e 50000 respectivamente.

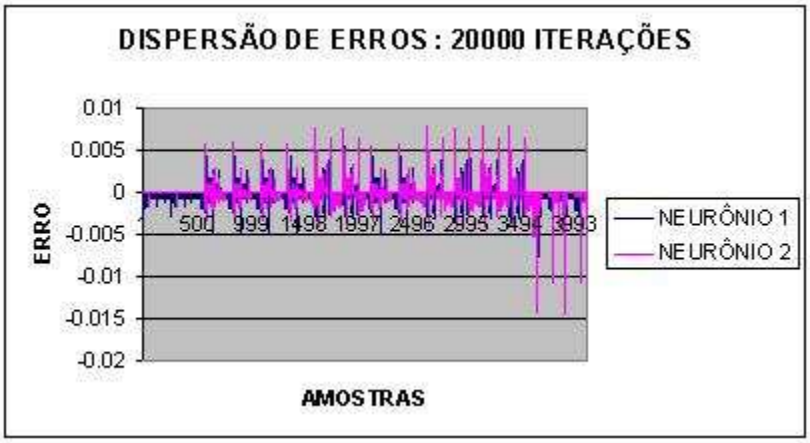

Figura 5.4.8 Gráfico de Dispersão de Erros : 20.000 iterações : Precisão 99.90 %

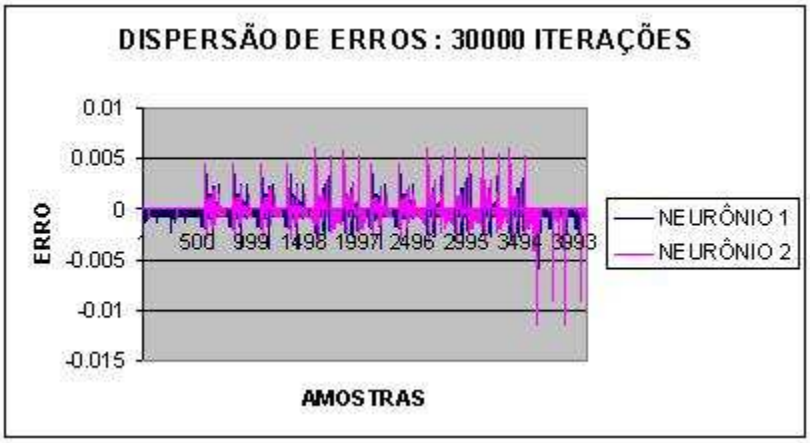

Figura 5.4.9 Gráfico de Dispersão de Erros : 30.000 iterações : Precisão 99.95 %

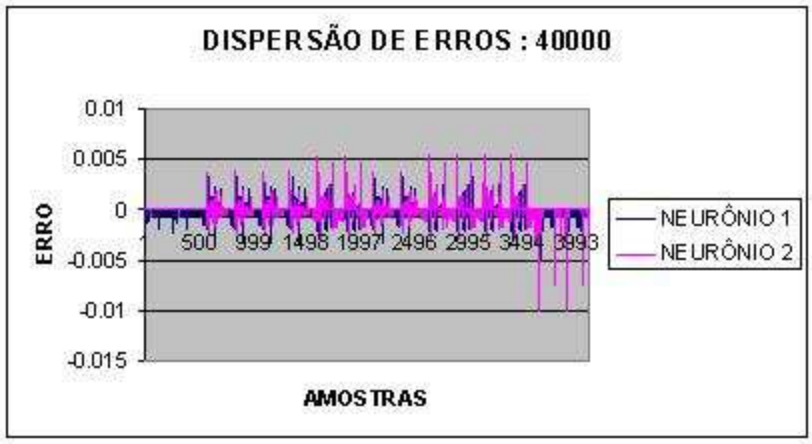

Figura 5.4.10 Gráfico de Dispersão de Erros : 40.000 iterações : Precisão 99.97 %

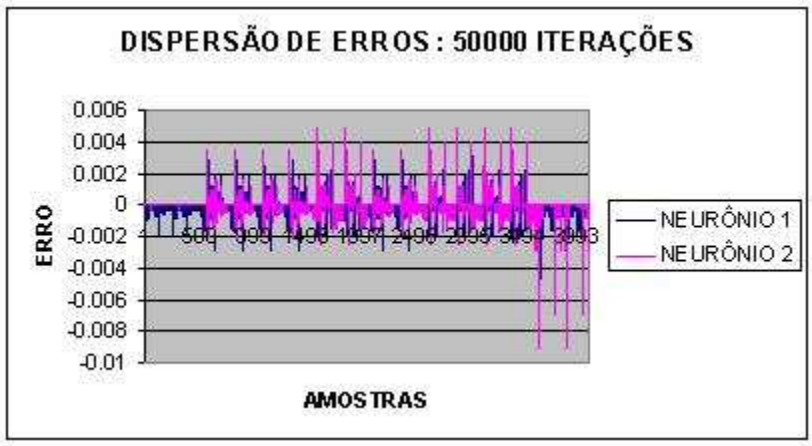

Figura 5.4.11 Gráfico de Dispersão de Erros : 50.000 iterações : Precisão 100 %

Colocando os dados da tabela 5.4.2 na forma gráfica temos a Figura 5.4.12 com as curvas de redução de erros SSE, MSE, SSE /O units.

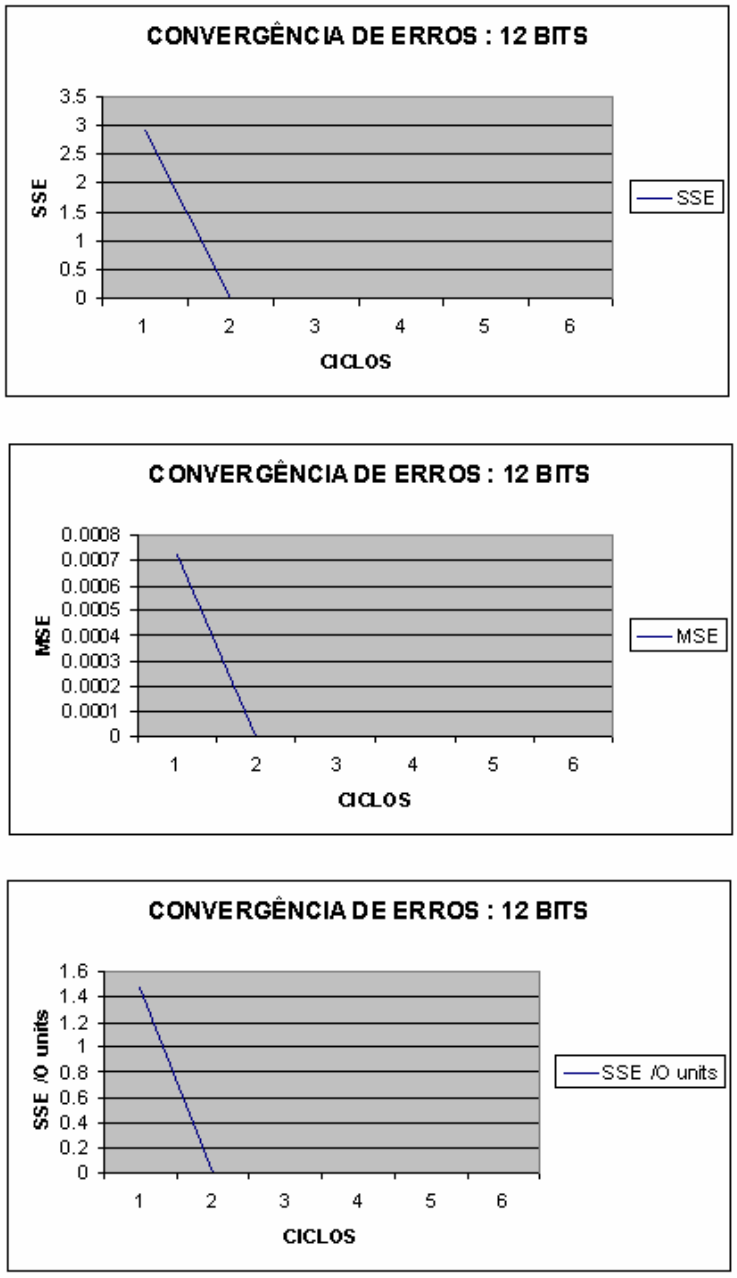

Figura 5.4.12 – Curvas de Convergências de Erros para os ciclos iterativos

A rede neural projetada para o cenário tridimensional convergiu para um ciclo inicial de 10000 iterações com precisão de 99.65%, porém o número de ciclos foi avançado até 50000

iterações de forma a reduzir o erro MSE ao valor zero (0) e obter precisão com valor de 100%. Os dados abaixo representam os valores de erros após 50000 convergências:

5.4.1 Arquivo Rede Neural Final : Cenário Tridimensional

Após o treinamento de 50.000 iterações considera-se que a rede neural de classificação tridimensional convergiu para um estado mínimo de energia, ou seja, a rede consegue estabelecer a correta correlação entre os dados de entrada e as classes de saídas desejadas. O pacote de simulação treinou a rede neural e registrou os respectivos pesos para as conexões sinápticas entre neurônios das camadas da rede. A tabela 5.4.7 representa os pesos das conexões Rede Neural Tridimensional. A tabela 5.4.6 representa a topologia final da Rede Neural Tridimensional.

Tabela 5.4.4 Parâmetros de treinamento Rede Neural Tridimensional : 12 Bits : Propriedades física Peso + Diâmetro + "*Firmness*".

Rede Neural de Classificação Tridimensional	
Criação	SNNS network definition file V1.4-3D generated at Tue Oct 09 10:54:26 2001
network name	CENARIOTRIDIMENSIONAL_50000
source files	
no. of units	30
no. of connections	176
no. of unit types	0
no. of site types	0
learning function	Std_Backpropagation
update function	Topological_Order

Tabela 5.4.5 funções de ativação.

act	Bias	St	subnet	layer	act func	out func
0.00000	0.00000	H	0	1	Act_Logistic	Out_Identity

Tabela 5.4.6 Topologia final da Rede Neural Tridimensional : 12 Bits : Propriedades física Peso + Diâmetro + "*Firmness*".

Topologia final da Rede Neural Tridimensional : 12 Bits : 2 Propriedade : Peso + Diâmetro + "*Firmness*"									
No	typeName	unitName	act	bias	St	position	act func	Out func	sites
1			1.00000	0.50027	I	2, 2, 0			
2			1.00000	-0.73894	I	3, 2, 0			
3			1.00000	0.91244	I	4, 2, 0			
4			1.00000	-0.11620	I	5, 2, 0			
5			1.00000	-0.32640	I	6, 2, 0			
6			1.00000	0.31382	I	7, 2, 0			
7			1.00000	0.68166	I	8, 2, 0			
8			1.00000	0.22002	I	9, 2, 0			
9			1.00000	-0.12740	I	10, 2, 0			
10			1.00000	-0.71933	I	11, 2, 0			
11			1.00000	0.28858	I	12, 2, 0			
12			1.00000	0.88620	I	13, 2, 0			
13			1.00000	-1.33814	H	2, 5, 0			
14			1.00000	-2.02058	H	3, 5, 0			
15			0.70267	7.07426	H	4, 5, 0			
16			0.26436	-3.74534	H	5, 5, 0			
17			0.99942	-10.22155	H	6, 5, 0			
18			0.54973	-4.18714	H	7, 5, 0			
19			0.00001	-5.79982	H	8, 5, 0			
20			0.00000	-0.90737	H	9, 5, 0			
21			0.00680	5.74617	H	2, 8, 0			
22			0.31644	2.02090	H	3, 8, 0			
23			0.99998	0.82718	H	4, 8, 0			
24			0.00011	-1.60057	H	5, 8, 0			
25			1.00000	-3.33314	H	6, 8, 0			
26			0.00004	1.15034	H	7, 8, 0			
27			0.63421	0.91155	H	8, 8, 0			
28			1.00000	4.55715	H	9, 8, 0			
29			0.00085	11.99312	O	5,10, 0			
30			0.00000	0.57368	O	6,10, 0			

Tabela 5.4.7 Pesos das conexões Rede Neural Tridimensional : 12 Bits : Propriedades física Peso + Diâmetro + "*Firmness*".

Target	site	source:weight
13		1:-2.30317, 2:-3.88657, 3:-2.96447, 4:-2.96412, 5:15.21106, 6: 5.43928, 7: 5.40588, 8: 3.29622, 9:-1.05744, 10:-1.05689, 11:-1.05638, 12: 0.00056
14		1:-3.50124, 2:-1.65740, 3:-1.89485, 4:-1.90679, 5:-2.14870, 6:-0.55606, 7: 0.18970, 8:-1.64882, 9:12.27549, 10:12.27537, 11:12.27529, 12:-0.01024
15		1: 1.99436, 2: 1.51683, 3: 1.21588, 4: 1.21930, 5:-8.24572, 6:-11.32003, 7: 4.75111, 8: 2.40639, 9: 0.08304, 10: 0.08291, 11: 0.08272, 12:-0.00103
16		1: 1.69688, 2: 2.63751, 3:-0.62573, 4:-0.61237, 5:-0.19497, 6:-10.42797, 7: 5.42787, 8: 5.30574, 9:-0.15811, 10:-0.15786, 11:-0.15759, 12:-0.01145
17		1:11.39575, 2:12.34144, 3:-0.96653, 4:-0.96788, 5:-1.61750, 6:-0.95512, 7:-0.50944, 8:-0.94574, 9:-0.03228, 10:-0.03233, 11:-0.03236, 12:-0.00148
18		1:-2.55205, 2:-0.09329, 3:-5.82515, 4:-5.72406, 5: 8.39038, 6:12.65833, 7:-4.82468, 8:-2.18262, 9: 1.51364, 10: 1.51352, 11: 1.51348, 12:-0.00076
19		1:-20.07758, 2:-22.04188, 3: 2.18640, 4: 2.13399, 5: 2.39696, 6: 0.51654, 7: 0.41438, 8: 1.38823, 9: 9.25048, 10: 9.25040, 11: 9.25024, 12: 0.00090
20		1:-4.57940, 2:-2.61557, 3: 1.48640, 4: 1.51437, 5:-0.64754, 6:13.13475, 7:-11.87645, 8:-9.41634, 9:-1.62804, 10:-1.62783, 11:-1.62754, 12:-0.00151
21		13: 0.81395, 14:-9.81472, 15:-3.02025, 16: 2.27621, 17:-2.44590, 18: 4.06677, 19: 8.58545, 20:-4.85479
22		13: 3.31629, 14:-2.68268, 15:-9.34397, 16: 6.60481, 17: 2.62934, 18:-2.24270, 19: 2.10515, 20: 6.00563
23		13:11.48695, 14:-10.84036, 15:-6.40109, 16:-2.27549, 17:11.86656, 18: 4.74181, 19:10.39951, 20:-9.63216
24		13:-8.13205, 14:-0.36618, 15: 1.38096, 16: 8.07499, 17:-1.11294, 18:-1.80525, 19:-1.31234, 20:-1.17961
25		13:11.39155, 14: 2.48576, 15:-6.17158, 16:-3.16158, 17: 6.75456, 18: 1.73327, 19:-0.87338, 20:-3.28917
26		13:-12.62681, 14:-8.88865, 15: 6.01345, 16:-3.85151, 17: 7.60807, 18:-1.25066, 19: 8.75543, 20:-3.16266
27		13:-11.46012, 14: 0.04196, 15:-4.81813, 16:11.81684, 17:12.12150, 18:-1.44797, 19: 1.60269, 20: 7.18183
28		13: 2.48409, 14:-10.04564, 15: 4.46700, 16:-9.70237, 17:11.80063, 18: 7.56694, 19: 6.90973, 20:-7.34480
29		21:-16.05082, 22:-9.57809, 23:-13.10265, 24:-6.11231, 25: 9.51855, 26:-8.14819, 27:-20.10485, 28: 0.41422
30		21:-9.43870, 22: 8.65670, 23:-18.00660, 24: 6.48977, 25: 9.80934, 26:-17.95796, 27: 9.69274, 28:-18.75752

5.5 Cenário *N* Dimensional

Neste trabalho foram avaliados três cenários:

1. Unidimensional: 1 propriedade física: Peso
2. Bidimensional: 2 propriedades Físicas: Peso + Diâmetro
3. Tridimensional: 3 propriedades Físicas: Peso + Diâmetro + Firmeza ("*Firmness*")

Entende-se que a progressão para o cenário *N* dimensional depende apenas dos levantamentos dos mapas vetorias/padrões para a implementação de um conjunto maior de propriedades no processo de classificação de frutos utilizando-se de redes neurais. O modelo proposto neste trabalho permite que sejam utilizados *N* canais de entrada dentro de uma rede neural. Para tanto é necessário seguir o fluxo apresentado na figura 4.4.6 do capitulo 4.

Capítulo 6

6 Conclusões

O estudo das redes neurais artificiais para uso em processos de classificação de frutos, especificamente com a utilização das redes neurais de múltiplas camadas e algoritmo "*backpropagation*" permite concluir que:

1. O modelo proposto por este trabalho possibilita a utilização de redes neurais artificiais na classificação artificial de frutos, usando-se *n* propriedades físicas dos mesmos.
2. Todos os três cenários estudados convergiram para uma superfície de erro mínimo global.
3. Os testes do cenário unidimensional com uso de 4 à 8 bits na camada de entrada permitiu verificar a escalabilidade do modelo proposto, dependendo apenas de maior processamento de cálculo para a convergência e uso das redes neurais.
4. O pacote de simulação SNNS 4.1 de Stuttgart diminuiu significativamente o tempo de estudo da topologia das redes. Todas as redes criadas no pacote foram implementadas/testadas em programas de teste, comprovando a efetividade dos resultados obtidos com o pacote de simulação.
5. O aumento do número de bits do vetor de entrada, na camada de entrada da rede permite a obtenção de uma melhor resolução do espaço de padrões, aumentando as chances de convergência da rede.
6. Durante o treinamento inúmeras foram as vezes que as redes neurais não convergiram para os valores de saída desejados, pois os pesos iniciais ajustados randomicamente, faziam com que a rede saturasse em um erro mínimo local, obrigando à submissão da rede à um novo treinamento com novos peso, de forma a garantir o aprendizado adequado.
7. A parte mais trabalhosa na busca de redes que satisfizessem os espaços vetoriais propostos foi o estudo da topologia adequada para o aprendizado do relacionamento entre os dados de entrada e de saída.

8. O sucesso da convergência para um erro mínimo global e aprendizado do relacionamento entre dados de entrada e saída da rede é função do "ajuste" inicial dos pesos corretos.
9. O número de iterações necessárias para o aprendizado da relação entrada de padrões e saída de classes diminuiu à medida que o número de amostras aumentou.
10. Algumas redes apresentaram comportamentos "espúrios" devido à saturação da rede em mínimos locais.
11. O cenário bidimensional não convergiu com o algoritmo std_Backpropagation, a rede foi treinada com o algoritmo de erro Backpropmomentum. Este fato deveu-se à provável existência de mínimos locais na superfície de erros.

Acknowlegments:

Thanks to team who developed the SNNS 4.1 Sttutgart Neural Networks Simulator, which helped to develop this work.

7 Referências Bibliográficas

ABBOT, J. A. Firmness measurement of freshly harvested 'Delicious' apples by sensory methods, sonic transmission, magness-taylor, and compression. **J. Am. Soc. Hortic. Sci** 119(3):510-515, 1994.

ANDERSON, J.A. A simple neural network generating an interactive memory, **Math. Biosci.**, 14, 197-220, 1972.

BLUM, A. **NEURAL NETWORKS IN C^{++} - An Object Oriented Framework for building Connectionist Systems.** New York, 1992.

BLINDER, P.B. **Implementação da lógica Fuzzy em redes neurais artificiais e aplicações em biologia**, Campinas, 1994. 145p. Dissertação (Mestrado) Universidade Estadual de Campinas.

BOLLEN, A.F.; KUSABS, N. J.; HOLMES, G.; Hall, M.A. Comparison of consumer and producer perceptions of mushroom quality. **Proc Integrated View of Fruit and Vegetable Quality International Multidisciplinary Conference**, Georgia, USA, may, pp 303-311, 2000.

BUFO, M. J. **Aplicação de rede neural artificial como auxiliar na predição do desempenho de um landfarming**, Campinas, 2000. 94p. Dissertação (Mestrado), Faculdade de Engenharia Química, Universidade Estadual de Campinas.

BULLOCK, D. et al. **Neural Networks for yout toolbox.** Agricultural Engineering, July, 1992.

CASTRO, L.N.; LIMA, W. da S. e MARTINS, W. Redes Neurais Artificiais Aplicadas a Previsão de Produtividade de Soja, **Anais do III Congresso Brasileiro de Redes Neurais**, vol. único, pp. 308-312, 1997.

CASTRO, L.N.; LIMA, W. da S. e MARTINS, W. Classificador Neural para Previsão de Carga a Curto Prazo. **Anais do IV Simpósio Brasileiro de Redes Neurais**, vol. único, pp. 68-70, 1997.

CASTRO, L.N.; VON ZUBEN, F. J. e MARTINS, W. Hybrid and Constructive Learning Applied to a Prediction Problem in Agriculture. **Proceedings of the IEEE International Joint Conference on Neural Networks**, Alaska/USA, vol. 3, pp. 1932- 1926, Maio de 1998.

DELWICHE, M. J.; McDONALD T; BOWERS, S. V. Determination of peach firmness by analysis of impact forces. **Transactions ASAE**, 30(4):1160-1116, 1987.

DOWELL, F. E. Neural Network classification of undamaged and damaged peanut kernels using spectral data. **ASAE Paper**, No 93-3050, 1993.

ELIZONDO, D.A.; McCLENDON, R.W.; HOOGENBOOM, G. Neural Network Models for Predicting Flowering and Physiological Maturity of Soybean. **Transactions ASAE**, 37 (3): 981-988, 1994.

FREEMAN, J.A Backpropagation in a neural network . ***AI Expert*** **Neural Network Special Report, January:** 55-63, 1993.

GHAZANFARI, A.; IRUDAYARAJ, J.; KUSALIK, A. Grading Pistachio Nuts using Neural Network approach. **Transactions ASAE**, 39 (6): 2319-2324 1996.

GROSSBERG, S. Adaptative pattern classification and universal recording: Parallel development and coding of neural feacture detectors. **Biol. Cyber.**, 23 121-134, 1976.

GROSSBERG, S. **A theory of human memory: Self Organization and performance of sensory motor codes, maps and plans**, Academic Press, New York, 1978.

HAYKIN S. **Neural Networks – A Comprehensive Foundation**, Macmillan College Publishing Inc., 1994.

HEBB, D. **The organization of behaviour.** Wiley, New York, 1949.

HOPPFIELD, J. Neuron with graded response have collective computatioonal properties like those of two state neurons. **Proceedings of the National Academy of Sci.** may, p. 3088, 1984.

HOPPFIELD, J. Neural Networks and Physical Systems with Emergent Collective Computational Abilities. **Proceedings of the National Academy of Sci.** vol 79, 1982.

HOPPFIELD, J.; TANK, D. W. Neural Computation of decisions in optimization problems. **Biological Cybernetics** v. 52, p. 141-52, 1985.

HOPPFIELD, J. Neural Networks and Physical Systems with Emergent Collective Computational Abilities. **Proceedings of the National Academy of Sci.** vol 79, 1982.

KOHONEN, T. Correlation Matrix Memories - **IEEE Trans**. on Computers, c.21,1972.

KOHONEN, T. **Self-Organization and Associative Memory** – Springer-Verlag, second edition,1984.

KOSKO B. **Neural Networks and Fuzzy Systems – A dynamical Systems Approach to Machine Intelligence**, University of Southern California, Prentice-Hall International, Inc., 1992.

KOVACS, Z. L. **Redes Neurais Artificiais Fundamentos e Aplicações**, EDUSP, São Paulo, Edição Acadêmica, São Paulo 1996 174p.

KOVACS, Z. L. **O cérebro e sua mente: Uma introdução à Neurociência**, EDUSP, São Paulo, Edição Acadêmica, São Paulo 1997 213p.

LONA, L. M. F. **Desenvolvimento de software Usando Modelos Determinísticos e Redes Neurais Para o Processo de Craqueamento Catalítico.** Campinas, 2000. Tese (Doutorado), Faculdade de Engenharia Química, Universidade Estadual de Campinas.

MARTINS, M. P. **Comportamento Mecânico da Laranja Pêra.** Campinas, 1993. 100p 1993. Dissertação (Mestrado), Faculdade de Engenharia Agrícola, Universidade Estadual de Campinas.

McCLENDON, R. W.; BATCHELOR, W.D. Inspect pest management neural network **ASAE Paper**, No 95-3560, 1994

McCULLOCH, W. E, PITTS, W. A logical Calculus of the Ideas Immanent in Nervous Activity - **Bulettin of Mathematical Biophysics**, 1943.

MINSKY, M.; PAPPERT, S. **Perceptrons.** MIT Press, Cambridge, 1969.

MICCOLIS, V., MIKAL, E. S. Morphological and physiologica changes during fruit groth and maturation of seven melon cultivars. **J. Am. Soc. Hortic. Sci**, 116(6): 1025-1029, 1991.

MILLER, B. K., DELWICHE, M.J. Peach defect detection with machine vision. **Transactions ASAE**, 34(6):2588-2597, 1991.

NEURALWARE. 1993. **Advanced reference guide: Software reference for Professional II/PLUS and NeuralWorks Explorer.** Pittsburgh, Pa.: Neuraware, Inc.

OLIVEIRA, C. A; SILVA, F.A. T. F.; OLIVEIRA, T. N. Processamento de língua natural: abordagem simbólica ou conexionista?. S.J. dos Campos, **Boletim Instituto de pesquisas Espaciais**, 1996. 50 p.

OZER, N.; ENGEL, B. A.; SIMON, J.E. Fusion Classification Techniques for fruit quality. **Transactions ASAE**, 38 (6): 1927-1934, 1995.

PANDYA, A. S.; MACY, R. B. **Pattern recognition with Neural Networks in C++**, IEEE Press, 1995.

PELEG, K. Multi Sensor adaptive sorting of fruits and vegetables. **In Proc. of the Int. Workshop**, St. Joseph, MI, p. 111-119, 1993.

RIGNEY, M. PE., BROSEWITZ, G. H., KRANZLER, G. A. Asparagus defect inspection with machine vision. **Transactions ASAE**, 35(6):1873-1878, 1992.

RITTER, H.; MARTINETZ, T.; SEHULTEN, K. **Neural Computation and Self Organizing Maps.** New York, 1992.

ROSENBLATT F. **The Perceptron: A Perceiving and Recognizing Automation**, Technical Report 85-460-1, Cornell Aeronautical Laboratory, 1957.

RUMELHART, D.; HINTON, WILLIAMS **Learning Internal Representations by Error Propagation, em Parallel Distributed Processing.** MIT Press, Cambridge, 1986.

RUMELHART D.E.; McCLELLAND J.L. and the PDP Research Group. **Parallel Distributed Processing: Exploration in the Microstructure of Cognition**, vol. 1. MIT Press, Cambridge, Massachussetts, 1986.

SALESHI, F.; LACROIX, R.; WADE, K. M. Effects of Learning Parameters and Data Presentation on the Performance of Backpropagation Networks for Milk Yield Prediction **Transactions ASAE** , 41 (1): 253-259, 1998.

SATO, K.; SALOKHE, V.M. Voice Recognition by Neural Network Under Tractor Noise. **Transactions ASAE**, 36 (4): 1223-1227, 1993.

SAYEED, M.S.; WHITTAKER, A.D.; KEHTANARNAVAZ, N.D. Snack Quality Evaluation Method Based on Image Features and Neural Network Prediction. **Transactions ASAE**, 38 (4):1239-1245, 1995.

SHMULEVICH, I.; GALILI, N.; ROSENFELD, D. , Accoustic excitation for quality control of fruit. **ASAE Paper** No. 93-6027, St. Joseph, Mich, ASAE, 1993.

SILVA, L. N. C. **Análise e síntese de estratégias de aprendizado para redes neurais artificiais.** Campinas, 1998. 247p. Dissertação (Mestrado), Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas.

SNNS - **Stuttgart Neural Network Simulator V 4.1**, Institute for Parallel and Distributed High Performance Systems, University of Stuttgart, 1995.

SWETS, J. A.; PICKETT, R. M. **Evaluation of Diagnostic Systems.** Academic Press, Inc. New York, 1982.

TAO, Y. et al. Machine Vision for Color Inspection of potatoes and apples. **Transactions ASAE**, 38(5): 1555-1559, 1995.

THAI, C.N.; SHEWFELT, R.L. Modeling Sensory Color Quality of Tomato and Peach: Neural Networks and Statistical Regression. **Transactions ASAE**, 34 (3): 950-954, 1991.

TIMM, E. J.; SCHULTE, N. L.; BROWN, G. K.; GUYER, D. E. Color measurement for dark sweet cherry maturity estimation in Michigan. **ASAE Paper** No. 93-1045, St. Joseph, Mich, ASAE, 1993.

WERBOS, P.J. Neural Networks for control and system identification – **Proceedings of the 28 th CDC**, Tampa, Fl. 1989.

WIDROW, B. **Generalization and Information Storage in Networks of Adaline Neurons, em Self-Organizing Systems**, 1962.

WASSERMAN, P.D. **Neural Computing: Theory and Practice.** New York, 1989.

YANG, X.Z; LACROIX, R.; WADE, K.M. Neural Detection of Mastitis from Dairy Herd Improvement Records. **Transactions ASAE**, 42 (4):1063-1071, 1999.

ZAMBALDE, A. L. **Sistema computadorizado para aquisição e análise de dados agrometeorológicos.** Itajubá, Minas Gerais, 1991. 144p. Dissertação (Mestrado), Escola Federal de Engenharia de Itajubá.

Anexo I - Tabela Conversão Binária - Decimal

No	Binary	Hex	No	Binary	Hex	No	Binary	Hex	No	Binary	Hex
000	0000 0000	00	**032**	0010 0000	20	**064**	0100 0000	40	**096**	0110 0000	60
001	0000 0001	01	**033**	0010 0001	21	**065**	0100 0001	41	**097**	0110 0001	61
002	0000 0010	02	**034**	0010 0010	22	**066**	0100 0010	42	**098**	0110 0010	62
003	0000 0011	03	**035**	0010 0011	23	**067**	0100 0011	43	**099**	0110 0011	63
004	0000 0100	04	**036**	0010 0100	24	**068**	0100 0100	44	**100**	0110 0100	64
005	0000 0101	05	**037**	0010 0101	25	**069**	0100 0101	45	**101**	0110 0101	65
006	0000 0110	06	**038**	0010 0110	26	**070**	0100 0110	46	**102**	0110 0110	66
007	0000 0111	07	**039**	0010 0111	27	**071**	0100 0111	47	**103**	0110 0111	67
008	0000 1000	08	**040**	0010 1000	28	**072**	0100 1000	48	**104**	0110 1000	68
009	0000 1001	09	**041**	0010 1001	29	**073**	0100 1001	49	**105**	0110 1001	69
010	0000 1010	0A	**042**	0010 1010	2A	**074**	0100 1010	4A	**106**	0110 1010	6A
011	0000 1011	0B	**043**	0010 1011	2B	**075**	0100 1011	4B	**107**	0110 1011	6B
012	0000 1100	0C	**044**	0010 1100	2C	**076**	0100 1100	4C	**108**	0110 1100	6C
013	0000 1101	0D	**045**	0010 1101	2D	**077**	0100 1101	4D	**109**	0110 1101	6D
014	0000 1110	0E	**046**	0010 1110	2E	**078**	0100 1110	4E	**110**	0110 1110	6E
015	0000 1111	0F	**047**	0010 1111	2F	**079**	0100 1111	4F	**111**	0110 1111	6F
016	0001 0000	10	**048**	0011 0000	30	**080**	0101 0000	50	**112**	0111 0000	70
017	0001 0001	11	**049**	0011 0001	31	**081**	0101 0001	51	**113**	0111 0001	71
018	0001 0010	12	**050**	0011 0010	32	**082**	0101 0010	52	**114**	0111 0010	72
019	0001 0011	13	**051**	0011 0011	33	**083**	0101 0011	53	**115**	0111 0011	73
020	0001 0100	14	**052**	0011 0100	34	**084**	0101 0100	54	**116**	0111 0100	74
021	0001 0101	15	**053**	0011 0101	35	**085**	0101 0101	55	**117**	0111 0101	75
022	0001 0110	16	**054**	0011 0110	36	**086**	0101 0110	56	**118**	0111 0110	76
023	0001 0111	17	**055**	0011 0111	37	**087**	0101 0111	57	**119**	0111 0111	77
024	0001 1000	18	**056**	0011 1000	38	**088**	0101 1000	58	**120**	0111 1000	78
025	0001 1001	19	**057**	0011 1001	39	**089**	0101 1001	59	**121**	0111 1001	79
026	0001 1010	1A	**058**	0011 1010	3A	**090**	0101 1010	5A	**122**	0111 1010	7A
027	0001 1011	1B	**059**	0011 1011	3B	**091**	0101 1011	5B	**123**	0111 1011	7B
028	0001 1100	1C	**060**	0011 1100	3C	**092**	0101 1100	5C	**124**	0111 1100	7C
029	0001 1101	1D	**061**	0011 1101	3D	**093**	0101 1101	5D	**125**	0111 1101	7D
030	0001 1110	1E	**062**	0011 1110	3E	**094**	0101 1110	5E	**126**	0111 1110	7E
031	0001 1111	1F	**063**	0011 1111	3F	**095**	0101 1111	5F	**127**	0111 1111	7F

No	Binary	Hex	No	Binary	Hex	No	Binary	Hex	No	Binary	Hex
128	1000 0000	80	**160**	1010 0000	A0	**192**	1100 0000	C0	**224**	1110 0000	E0
129	1000 0001	81	**161**	1010 0001	A1	**193**	1100 0001	C1	**225**	1110 0001	E1
130	1000 0010	82	**162**	1010 0010	A2	**194**	1100 0010	C2	**226**	1110 0010	E2
131	1000 0011	83	**163**	1010 0011	A3	**195**	1100 0011	C3	**227**	1110 0011	E3
132	1000 0100	84	**164**	1010 0100	A4	**196**	1100 0100	C4	**228**	1110 0100	E4
133	1000 0101	85	**165**	1010 0101	A5	**197**	1100 0101	C5	**229**	1110 0101	E5
134	1000 0110	86	**166**	1010 0110	A6	**198**	1100 0110	C6	**230**	1110 0110	E6
135	1000 0111	87	**167**	1010 0111	A7	**199**	1100 0111	C7	**231**	1110 0111	E7
136	1000 1000	88	**168**	1010 1000	A8	**200**	1100 1000	C8	**232**	1110 1000	E8
137	1000 1001	89	**169**	1010 1001	A9	**201**	1100 1001	C9	**233**	1110 1001	E9
138	1000 1010	8A	**170**	1010 1010	AA	**202**	1100 1010	CA	**234**	1110 1010	EA
139	1000 1011	8B	**171**	1010 1011	AB	**203**	1100 1011	CB	**235**	1110 1011	EB
140	1000 1100	8C	**172**	1010 1100	AC	**204**	1100 1100	CC	**236**	1110 1100	EC
141	1000 1101	8D	**173**	1010 1101	AD	**205**	1100 1101	4D	**237**	1110 1101	ED
142	1000 1110	8E	**174**	1010 1110	AE	**206**	1100 1110	CE	**238**	1110 1110	EE
143	1000 1111	8F	**175**	1010 1111	AF	**207**	1100 1111	CF	**239**	1110 1111	EF
144	1001 0000	90	**176**	1011 0000	B0	**208**	1101 0000	D0	**240**	1111 0000	F0
145	1001 0001	91	**177**	1011 0001	B1	**209**	1101 0001	D1	**241**	1111 0001	F1
146	1001 0010	92	**178**	1011 0010	B2	**210**	1101 0010	D2	**242**	1111 0010	F2
147	1001 0011	93	**179**	1011 0011	B3	**211**	1101 0011	D3	**243**	1111 0011	F3
148	1001 0100	94	**180**	1011 0100	B4	**212**	1101 0100	D4	**244**	1111 0100	F4
149	1001 0101	95	**181**	1011 0101	B5	**213**	1101 0101	D5	**245**	1111 0101	F5
150	1001 0110	96	**182**	1011 0110	B6	**214**	1101 0110	D6	**246**	1111 0110	F6
151	1001 0111	97	**183**	1011 0111	B7	**215**	1101 0111	D7	**247**	1111 0111	F7
152	1001 1000	98	**184**	1011 1000	B8	**216**	1101 1000	D8	**248**	1111 1000	F8
153	1001 1001	99	**185**	1011 1001	B9	**217**	1101 1001	D9	**249**	1111 1001	F9
154	1001 1010	9A	**186**	1011 1010	BA	**218**	1101 1010	DA	**250**	1111 1010	FA
155	1001 1011	9B	**187**	1011 1011	BB	**219**	1101 1011	DB	**251**	1111 1011	FB
156	1001 1100	9C	**188**	1011 1100	BC	**220**	1101 1100	DC	**252**	1111 1100	FC
157	1001 1101	9D	**189**	1011 1101	BD	**221**	1101 1101	DD	**253**	1111 1101	FD
158	1001 1110	9E	**190**	1011 1110	BE	**222**	1101 1110	DE	**254**	1111 1110	FE
159	1001 1111	9F	**191**	1011 1111	BF	**223**	1101 1111	DF	**255**	1111 1111	FF

Anexo II - Dados Cenário Unidimensional

CENÁRIO UNIDIMENSIONAL : 8 BITS				
Entrada	Vetor Entrada : Binário	Vetor Saída : Binário	Vetor Entrada : Analógico	
0	00000000	00	180	Classe A: 00
1	00000001	00	180.2929688	Classe B: 10
2	00000010	00	180.5859375	Classe C: 01
3	00000011	00	180.8789063	Classe D: 11
4	00000100	00	181.171875	
5	00000101	00	181.4648438	
6	00000110	00	181.7578125	
7	00000111	00	182.0507813	
8	00001000	00	182.34375	
9	00001001	00	182.6367188	
10	00001010	00	182.9296875	
11	00001011	00	183.2226563	
12	00001100	00	183.515625	
13	00001101	00	183.8085938	
14	00001110	00	184.1015625	
15	00001111	00	184.3945313	
16	00010000	00	184.6875	
17	00010001	00	184.9804688	
18	00010010	00	185.2734375	
19	00010011	00	185.5664063	
20	00010100	00	185.859375	
21	00010101	00	186.1523438	
22	00010110	00	186.4453125	
23	00010111	00	186.7382813	
24	00011000	00	187.03125	
25	00011001	00	187.3242188	
26	00011010	00	187.6171875	
27	00011011	00	187.9101563	
28	00011100	00	188.203125	
29	00011101	00	188.4960938	
30	00011110	00	188.7890625	
31	00011111	00	189.0820313	
32	00100000	00	189.375	
33	00100001	00	189.6679688	
34	00100010	00	189.9609375	
35	00100011	00	190.2539063	
36	00100100	00	190.546875	
37	00100101	00	190.8398438	
38	00100110	00	191.1328125	
39	00100111	00	191.4257813	
40	00101000	00	191.71875	
41	00101001	00	192.0117188	
42	00101010	00	192.3046875	
43	00101011	00	192.5976563	

44	00101100	00	192.890625
45	00101101	00	193.1835938
46	00101110	00	193.4765625
47	00101111	00	193.7695313
48	00110000	00	194.0625
49	00110001	00	194.3554688
50	00110010	00	194.6484375
51	00110011	00	194.9414063
52	00110100	00	195.234375
53	00110101	00	195.5273438
54	00110110	00	195.8203125
55	00110111	00	196.1132813
56	00111000	00	196.40625
57	00111001	00	196.6992188
58	00111010	00	196.9921875
59	00111011	00	197.2851563
60	00111100	00	197.578125
61	00111101	00	197.8710938
62	00111110	00	198.1640625
63	00111111	00	198.4570313
64	01000000	00	198.75
65	01000001	00	199.0429688
66	01000010	00	199.3359375
67	01000011	00	199.6289063
68	01000100	00	199.921875
69	01000101	10	200.2148438
70	01000110	10	200.5078125
71	01000111	10	200.8007813
72	01001000	10	201.09375
73	01001001	10	201.3867188
74	01001010	10	201.6796875
75	01001011	10	201.9726563
76	01001100	10	202.265625
77	01001101	10	202.5585938
78	01001110	10	202.8515625
79	01001111	10	203.1445313
80	01010000	10	203.4375
81	01010001	10	203.7304688
82	01010010	10	204.0234375
83	01010011	10	204.3164063
84	01010100	10	204.609375
85	01010101	10	204.9023438
86	01010110	10	205.1953125
87	01010111	10	205.4882813
88	01011000	10	205.78125
89	01011001	10	206.0742188
90	01011010	10	206.3671875
91	01011011	10	206.6601563
92	01011100	10	206.953125
93	01011101	10	207.2460938

94	01011110	10	207.5390625
95	01011111	10	207.8320313
96	01100000	10	208.125
97	01100001	10	208.4179688
98	01100010	10	208.7109375
99	01100011	10	209.0039063
100	01100100	10	209.296875
101	01100101	10	209.5898438
102	01100110	10	209.8828125
103	01100111	10	210.1757813
104	01101000	10	210.46875
105	01101001	10	210.7617188
106	01101010	10	211.0546875
107	01101011	10	211.3476563
108	01101100	10	211.640625
109	01101101	10	211.9335938
110	01101110	10	212.2265625
111	01101111	10	212.5195313
112	01110000	10	212.8125
113	01110001	10	213.1054688
114	01110010	10	213.3984375
115	01110011	10	213.6914063
116	01110100	10	213.984375
117	01110101	10	214.2773438
118	01110110	10	214.5703125
119	01110111	10	214.8632813
120	01111000	10	215.15625
121	01111001	10	215.4492188
122	01111010	10	215.7421875
123	01111011	10	216.0351563
124	01111100	10	216.328125
125	01111101	10	216.6210938
126	01111110	10	216.9140625
127	01111111	10	217.2070313
128	10000000	10	217.5
129	10000001	10	217.7929688
130	10000010	10	218.0859375
131	10000011	10	218.3789063
132	10000100	10	218.671875
133	10000101	10	218.9648438
134	10000110	10	219.2578125
135	10000111	10	219.5507813
136	10001000	10	219.84375
137	10001001	10	220.1367188
138	10001010	10	220.4296875
139	10001011	10	220.7226563
140	10001100	10	221.015625
141	10001101	10	221.3085938
142	10001110	10	221.6015625
143	10001111	10	221.8945313

144	10010000	10	222.1875
145	10010001	10	222.4804688
146	10010010	10	222.7734375
147	10010011	10	223.0664063
148	10010100	10	223.359375
149	10010101	10	223.6523438
150	10010110	10	223.9453125
151	10010111	10	224.2382813
152	10011000	10	224.53125
153	10011001	10	224.8242188
154	10011010	10	225.1171875
155	10011011	10	225.4101563
156	10011100	10	225.703125
157	10011101	10	225.9960938
158	10011110	10	226.2890625
159	10011111	10	226.5820313
160	10100000	10	226.875
161	10100001	10	227.1679688
162	10100010	10	227.4609375
163	10100011	10	227.7539063
164	10100100	10	228.046875
165	10100101	10	228.3398438
166	10100110	10	228.6328125
167	10100111	10	228.9257813
168	10101000	10	229.21875
169	10101001	10	229.5117188
170	10101010	10	229.8046875
171	10101011	01	230.0976563
172	10101100	01	230.390625
173	10101101	01	230.6835938
174	10101110	01	230.9765625
175	10101111	01	231.2695313
176	10110000	01	231.5625
177	10110001	01	231.8554688
178	10110010	01	232.1484375
179	10110011	01	232.4414063
180	10110100	01	232.734375
181	10110101	01	233.0273438
182	10110110	01	233.3203125
183	10110111	01	233.6132813
184	10111000	01	233.90625
185	10111001	01	234.1992188
186	10111010	01	234.4921875
187	10111011	01	234.7851563
188	10111100	01	235.078125
189	10111101	01	235.3710938
190	10111110	01	235.6640625
191	10111111	01	235.9570313
192	11000000	01	236.25
193	11000001	01	236.5429688

194	11000010	01	236.8359375
195	11000011	01	237.1289063
196	11000100	01	237.421875
197	11000101	01	237.7148438
198	11000110	01	238.0078125
199	11000111	01	238.3007813
200	11001000	01	238.59375
201	11001001	01	238.8867188
202	11001010	01	239.1796875
203	11001011	01	239.4726563
204	11001100	01	239.765625
205	11001101	01	240.0585938
206	11001110	01	240.3515625
207	11001111	01	240.6445313
208	11010000	01	240.9375
209	11010001	01	241.2304688
210	11010010	01	241.5234375
211	11010011	01	241.8164063
212	11010100	01	242.109375
213	11010101	01	242.4023438
214	11010110	01	242.6953125
215	11010111	01	242.9882813
216	11011000	01	243.28125
217	11011001	01	243.5742188
218	11011010	01	243.8671875
219	11011011	01	244.1601563
220	11011100	01	244.453125
221	11011101	01	244.7460938
222	11011110	11	245.0390625
223	11011111	11	245.3320313
224	11100000	11	245.625
225	11100001	11	245.9179688
226	11100010	11	246.2109375
227	11100011	11	246.5039063
228	11100100	11	246.796875
229	11100101	11	247.0898438
230	11100110	11	247.3828125
231	11100111	11	247.6757813
232	11101000	11	247.96875
233	11101001	11	248.2617188
234	11101010	11	248.5546875
235	11101011	11	248.8476563
236	11101100	11	249.140625
237	11101101	11	249.4335938
238	11101110	11	249.7265625
239	11101111	11	250.0195313
240	11110000	11	250.3125
241	11110001	11	250.6054688
242	11110010	11	250.8984375
243	11110011	11	251.1914063

244	11110100	11	251.484375
245	11110101	11	251.7773438
246	11110110	11	252.0703125
247	11110111	11	252.3632813
248	11111000	11	252.65625
249	11111001	11	252.9492188
250	11111010	11	253.2421875
251	11111011	11	253.5351563
252	11111100	11	253.828125
253	11111101	11	254.1210938
254	11111110	11	254.4140625
255	11111111	11	254.7070313

Anexo III - Dados Cenário Bidimensional

	ENTRADA BIDIMENSIONAL : 8 BITS					
	PESO		DIÂMETRO		PESO+DIAMETRO	
ENTRADA	DECIMAL	BINÁRIO	DECIMAL	BINÁRIO	DECIMAL	CLASSE
1	180	0000	8	0000	00000000	00
2	180	0000	8.1	0001	00000001	00
3	180	0000	8.2	0010	00000010	00
4	180	0000	8.3	0011	00000011	00
5	180	0000	8.4	0100	00000100	00
6	180	0000	8.5	0101	00000101	00
7	180	0000	8.6	0110	00000110	00
8	180	0000	8.7	0111	00000111	00
9	180	0000	8.8	1000	00001000	00
10	180	0000	8.9	1001	00001001	00
11	180	0000	9	1010	00001010	00
12	180	0000	9.1	1011	00001011	00
13	180	0000	9.2	1100	00001100	00
14	180	0000	9.3	1101	00001101	00
15	180	0000	9.4	1110	00001110	00
16	180	0000	9.5	1111	00001111	00
17	184.6875	0001	8	0000	00010000	00
18	184.6875	0001	8.1	0001	00010001	00
19	184.6875	0001	8.2	0010	00010010	00
20	184.6875	0001	8.3	0011	00010011	00
21	184.6875	0001	8.4	0100	00010100	00
22	184.6875	0001	8.5	0101	00010101	00
23	184.6875	0001	8.6	0110	00010110	00
24	184.6875	0001	8.7	0111	00010111	00
25	184.6875	0001	8.8	1000	00011000	00
26	184.6875	0001	8.9	1001	00011001	00
27	184.6875	0001	9	1010	00011010	00
28	184.6875	0001	9.1	1011	00011011	00
29	184.6875	0001	9.2	1100	00011100	00
30	184.6875	0001	9.3	1101	00011101	00
31	184.6875	0001	9.4	1110	00011110	00
32	184.6875	0001	9.5	1111	00011111	00
33	189.375	0010	8	0000	00100000	00
34	189.375	0010	8.1	0001	00100001	00
35	189.375	0010	8.2	0010	00100010	00
36	189.375	0010	8.3	0011	00100011	00
37	189.375	0010	8.4	0100	00100100	00
38	189.375	0010	8.5	0101	00100101	00
39	189.375	0010	8.6	0110	00100110	00
40	189.375	0010	8.7	0111	00100111	00
41	189.375	0010	8.8	1000	00101000	00
42	189.375	0010	8.9	1001	00101001	00
43	189.375	0010	9	1010	00101010	00
44	189.375	0010	9.1	1011	00101011	00

45	189.375	0010	9.2	1100	00101100	00
46	189.375	0010	9.3	1101	00101101	00
47	189.375	0010	9.4	1110	00101110	00
48	189.375	0010	9.5	1111	00101111	00
49	194.0625	0011	8	0000	00110000	00
50	194.0625	0011	8.1	0001	00110001	00
51	194.0625	0011	8.2	0010	00110010	00
52	194.0625	0011	8.3	0011	00110011	01
53	194.0625	0011	8.4	0100	00110100	01
54	194.0625	0011	8.5	0101	00110101	01
55	194.0625	0011	8.6	0110	00110110	10
56	194.0625	0011	8.7	0111	00110111	00
57	194.0625	0011	8.8	1000	00111000	00
58	194.0625	0011	8.9	1001	00111001	00
59	194.0625	0011	9	1010	00111010	00
60	194.0625	0011	9.1	1011	00111011	00
61	194.0625	0011	9.2	1100	00111100	00
62	194.0625	0011	9.3	1101	00111101	00
63	194.0625	0011	9.4	1110	00111110	00
64	194.0625	0011	9.5	1111	00111111	00
65	198.75	0100	8	0000	01000000	00
66	198.75	0100	8.1	0001	01000001	01
67	198.75	0100	8.2	0010	01000010	01
68	198.75	0100	8.3	0011	01000011	01
69	198.75	0100	8.4	0100	01000100	01
70	198.75	0100	8.5	0101	01000101	10
71	198.75	0100	8.6	0110	01000110	10
72	198.75	0100	8.7	0111	01000111	10
73	198.75	0100	8.8	1000	01001000	10
74	198.75	0100	8.9	1001	01001001	00
75	198.75	0100	9	1010	01001010	00
76	198.75	0100	9.1	1011	01001011	11
77	198.75	0100	9.2	1100	01001100	00
78	198.75	0100	9.3	1101	01001101	00
79	198.75	0100	9.4	1110	01001110	00
80	198.75	0100	9.5	1111	01001111	00
81	203.4375	0101	8	0000	01010000	00
82	203.4375	0101	8.1	0001	01010001	01
83	203.4375	0101	8.2	0010	01010010	01
84	203.4375	0101	8.3	0011	01010011	01
85	203.4375	0101	8.4	0100	01010100	01
86	203.4375	0101	8.5	0101	01010101	10
87	203.4375	0101	8.6	0110	01010110	10
88	203.4375	0101	8.7	0111	01010111	10
89	203.4375	0101	8.8	1000	01011000	10
90	203.4375	0101	8.9	1001	01011001	11
91	203.4375	0101	9	1010	01011010	11
92	203.4375	0101	9.1	1011	01011011	11
93	203.4375	0101	9.2	1100	01011100	11
94	203.4375	0101	9.3	1101	01011101	00
95	203.4375	0101	9.4	1110	01011110	00
96	203.4375	0101	9.5	1111	01011111	00

97	208.125	0110	8	0000	01100000	00
98	208.125	0110	8.1	0001	01100001	01
99	208.125	0110	8.2	0010	01100010	01
100	208.125	0110	8.3	0011	01100011	01
101	208.125	0110	8.4	0100	01100100	01
102	208.125	0110	8.5	0101	01100101	10
103	208.125	0110	8.6	0110	01100110	10
104	208.125	0110	8.7	0111	01100111	10
105	208.125	0110	8.8	1000	01101000	10
106	208.125	0110	8.9	1001	01101001	11
107	208.125	0110	9	1010	01101010	11
108	208.125	0110	9.1	1011	01101011	11
109	208.125	0110	9.2	1100	01101100	11
110	208.125	0110	9.3	1101	01101101	00
111	208.125	0110	9.4	1110	01101110	00
112	208.125	0110	9.5	1111	01101111	00
113	212.8125	0111	8	0000	01110000	00
114	212.8125	0111	8.1	0001	01110001	01
115	212.8125	0111	8.2	0010	01110010	01
116	212.8125	0111	8.3	0011	01110011	01
117	212.8125	0111	8.4	0100	01110100	01
118	212.8125	0111	8.5	0101	01110101	01
119	212.8125	0111	8.6	0110	01110110	10
120	212.8125	0111	8.7	0111	01110111	10
121	212.8125	0111	8.8	1000	01111000	10
122	212.8125	0111	8.9	1001	01111001	11
123	212.8125	0111	9	1010	01111010	11
124	212.8125	0111	9.1	1011	01111011	11
125	212.8125	0111	9.2	1100	01111100	11
126	212.8125	0111	9.3	1101	01111101	00
127	212.8125	0111	9.4	1110	01111110	00
128	212.8125	0111	9.5	1111	01111111	00
129	217.5	1000	8	0000	10000000	00
130	217.5	1000	8.1	0001	10000001	01
131	217.5	1000	8.2	0010	10000010	01
132	217.5	1000	8.3	0011	10000011	01
133	217.5	1000	8.4	0100	10000100	01
134	217.5	1000	8.5	0101	10000101	10
135	217.5	1000	8.6	0110	10000110	10
136	217.5	1000	8.7	0111	10000111	10
137	217.5	1000	8.8	1000	10001000	10
138	217.5	1000	8.9	1001	10001001	10
139	217.5	1000	9	1010	10001010	11
140	217.5	1000	9.1	1011	10001011	11
141	217.5	1000	9.2	1100	10001100	11
142	217.5	1000	9.3	1101	10001101	00
143	217.5	1000	9.4	1110	10001110	00
144	217.5	1000	9.5	1111	10001111	00
145	222.1875	1001	8	0000	10010000	00
146	222.1875	1001	8.1	0001	10010001	01
147	222.1875	1001	8.2	0010	10010010	01
148	222.1875	1001	8.3	0011	10010011	01

149	222.1875	1001	8.4	0100	10010100	01
150	222.1875	1001	8.5	0101	10010101	10
151	222.1875	1001	8.6	0110	10010110	10
152	222.1875	1001	8.7	0111	10010111	10
153	222.1875	1001	8.8	1000	10011000	10
154	222.1875	1001	8.9	1001	10011001	11
155	222.1875	1001	9	1010	10011010	11
156	222.1875	1001	9.1	1011	10011011	11
157	222.1875	1001	9.2	1100	10011100	11
158	222.1875	1001	9.3	1101	10011101	00
159	222.1875	1001	9.4	1110	10011110	00
160	222.1875	1001	9.5	1111	10011111	00
161	226.875	1010	8	0000	10100000	00
162	226.875	1010	8.1	0001	10100001	01
163	226.875	1010	8.2	0010	10100010	01
164	226.875	1010	8.3	0011	10100011	01
165	226.875	1010	8.4	0100	10100100	01
166	226.875	1010	8.5	0101	10100101	10
167	226.875	1010	8.6	0110	10100110	10
168	226.875	1010	8.7	0111	10100111	10
169	226.875	1010	8.8	1000	10101000	10
170	226.875	1010	8.9	1001	10101001	11
171	226.875	1010	9	1010	10101010	11
172	226.875	1010	9.1	1011	10101011	11
173	226.875	1010	9.2	1100	10101100	11
174	226.875	1010	9.3	1101	10101101	00
175	226.875	1010	9.4	1110	10101110	00
176	226.875	1010	9.5	1111	10101111	00
177	231.5625	1011	8	0000	10110000	00
178	231.5625	1011	8.1	0001	10110001	01
179	231.5625	1011	8.2	0010	10110010	01
180	231.5625	1011	8.3	0011	10110011	01
181	231.5625	1011	8.4	0100	10110100	01
182	231.5625	1011	8.5	0101	10110101	10
183	231.5625	1011	8.6	0110	10110110	10
184	231.5625	1011	8.7	0111	10110111	10
185	231.5625	1011	8.8	1000	10111000	10
186	231.5625	1011	8.9	1001	10111001	00
187	231.5625	1011	9	1010	10111010	11
188	231.5625	1011	9.1	1011	10111011	00
189	231.5625	1011	9.2	1100	10111100	00
190	231.5625	1011	9.3	1101	10111101	00
191	231.5625	1011	9.4	1110	10111110	00
192	231.5625	1011	9.5	1111	10111111	00
193	236.25	1100	8	0000	11000000	00
194	236.25	1100	8.1	0001	11000001	00
195	236.25	1100	8.2	0010	11000010	00
196	236.25	1100	8.3	0011	11000011	01
197	236.25	1100	8.4	0100	11000100	01
198	236.25	1100	8.5	0101	11000101	00
199	236.25	1100	8.6	0110	11000110	10
200	236.25	1100	8.7	0111	11000111	00

201	236.25	1100	8.8	1000	11001000	00
202	236.25	1100	8.9	1001	11001001	00
203	236.25	1100	9	1010	11001010	00
204	236.25	1100	9.1	1011	11001011	00
205	236.25	1100	9.2	1100	11001100	00
206	236.25	1100	9.3	1101	11001101	00
207	236.25	1100	9.4	1110	11001110	00
208	236.25	1100	9.5	1111	11001111	00
209	240.9375	1101	8	0000	11010000	00
210	240.9375	1101	8.1	0001	11010001	00
211	240.9375	1101	8.2	0010	11010010	00
212	240.9375	1101	8.3	0011	11010011	00
213	240.9375	1101	8.4	0100	11010100	00
214	240.9375	1101	8.5	0101	11010101	00
215	240.9375	1101	8.6	0110	11010110	00
216	240.9375	1101	8.7	0111	11010111	00
217	240.9375	1101	8.8	1000	11011000	00
218	240.9375	1101	8.9	1001	11011001	00
219	240.9375	1101	9	1010	11011010	00
220	240.9375	1101	9.1	1011	11011011	00
221	240.9375	1101	9.2	1100	11011100	00
222	240.9375	1101	9.3	1101	11011101	00
223	240.9375	1101	9.4	1110	11011110	00
224	240.9375	1101	9.5	1111	11011111	00
225	245.625	1110	8	0000	11100000	00
226	245.625	1110	8.1	0001	11100001	00
227	245.625	1110	8.2	0010	11100010	00
228	245.625	1110	8.3	0011	11100011	00
229	245.625	1110	8.4	0100	11100100	00
230	245.625	1110	8.5	0101	11100101	00
231	245.625	1110	8.6	0110	11100110	00
232	245.625	1110	8.7	0111	11100111	00
233	245.625	1110	8.8	1000	11101000	00
234	245.625	1110	8.9	1001	11101001	00
235	245.625	1110	9	1010	11101010	00
236	245.625	1110	9.1	1011	11101011	00
237	245.625	1110	9.2	1100	11101100	00
238	245.625	1110	9.3	1101	11101101	00
239	245.625	1110	9.4	1110	11101110	00
240	245.625	1110	9.5	1111	11101111	00
241	250.3125	1111	8	0000	11110000	00
242	250.3125	1111	8.1	0001	11110001	00
243	250.3125	1111	8.2	0010	11110010	00
244	250.3125	1111	8.3	0011	11110011	00
245	250.3125	1111	8.4	0100	11110100	00
246	250.3125	1111	8.5	0101	11110101	00
247	250.3125	1111	8.6	0110	11110110	00
248	250.3125	1111	8.7	0111	11110111	00
249	250.3125	1111	8.8	1000	11111000	00
250	250.3125	1111	8.9	1001	11111001	00
251	250.3125	1111	9	1010	11111010	00
252	250.3125	1111	9.1	1011	11111011	00

253	250.3125	1111	9.2	1100	11111100	00
254	250.3125	1111	9.3	1101	11111101	00
255	250.3125	1111	9.4	1110	11111110	00
256	250.3125	1111	9.5	1111	11111111	00

Anexo IV - Programa C++ XOR

```
/* ========================================        *
 * Filename:        main.cpp                                        *
 * Author:          Jean Paulo Silva Ramos.                           *
 *                                                                    *
 * Descrição:                                                         *
 * Este é um pequena rede neural que usa                        *
 * back propagation para ajustes de pesos.                      *
 * ========================================  */



#include <iostream.h>
#include "bpnet.h"
#include <stdio.h>

#define BPM_ITER          50000

/********************** ROTINAS DE TRACER INPUT/OUTPUT DE DADOS *************/

FILE *stream;

/********************** ROTINAS DE TRACER INPUT/OUTPUT DE DADOS *************/


void main() {

            CBPNet bp;

// Abre o arquivo para escrita
   if( (stream = fopen( "tracer.txt", "w+" )) == NULL )
      printf( "O arquivo 'tracer.txt' não foi aberto para log\n" );
   else
      printf( "O arquivo 'tracer.txt' foi aberto para log\n" );
//              fprintf( stream2, "teste\n");
            //fprintf( stream,"[Le_n_Input] -- /*Funcao que tem por finalidade abrir o arquivo de entrada e
atualizar o numero de patterns de entrada da rede.*/\n");
              //fprintf( stream2, "%d\n", i );
     //fprintf( stream2, "%f\n", fp );


            //fprintf( stream2,"\t\n Wfh [%d] [%d] = %d\n\n", f,h,Wfh[f][h] );


            //  Abre o loop para iterações de treinamento

            for (int i=0;i<BPM_ITER;i++)
            {

                        // apresenta os padrões para treinamento

                        bp.Train(0,0,0);
```

```
                bp.Train(0,1,1);
                bp.Train(1,0,1);
                bp.Train(1,1,0);

        }

        // apresenta os padrões para conferência da rede

        cout << "0,0 = " << bp.Run(0,0) << endl;
        cout << "0,1 = " << bp.Run(0,1) << endl;
        cout << "1,0 = " << bp.Run(1,0) << endl;
        cout << "1,1 = " << bp.Run(1,1) << endl;

        //  fecha o arquivo de log
        fclose( stream );
        printf( "O arquivo 'tracer.txt' foi fechado\n" );
}

/* ========================================        *
 * Filename:        bpnet.h                          *
 * Author:          Jean Paulo Silva Ramos.                    *
 *                                                             *
 * Descrição:                                                  *
 * Este é um pequena rede neural que usa                  *
 * back propagation para ajustes de pesos.                *
 * ========================================  */

#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <conio.h>
#include <stdio.h>
#include <string.h>
#include <process.h>

/********************** ROTINAS DE TRACER INPUT/OUTPUT DE DADOS *************/

FILE *streamInterno;

/********************** ROTINAS DE TRACER INPUT/OUTPUT DE DADOS *************/


#define BP_LEARNING   (float)(0.5)          // Coeficiente de aprendizado.

class CBPNet {
        public:
                CBPNet();
                ~CBPNet() {};

                float Train(float, float, float);
                float Run(float, float);

        private:
                float m_fWeights[3][3];             // Pesos para os três neurônios.

                float Sigmoid(float);               // Função sigmoid.
};
```

```
CBPNet::CBPNet() {

        // Abre o arquivo para escrita
  if( (streamInterno = fopen( "tracerInterno.txt", "w+" )) == NULL )
    printf( "The file 'tracer.txt' was not opened\n" );
   else
    //printf( "O arquivo 'tracer.txt' was opened\n" );
         // fprintf( streamInterno,"[Le_n_Input] -- /*Funcao que tem por finalidade abrir o arquivo de
entrada e atualizar o numero de patterns de entrada da rede.*/\n");
         //fprintf( stream2, "%d\n", i );
    //fprintf( stream2, "%f\n", fp );

        //fprintf( stream2,"\t\n Wfh [%d] [%d] = %d\n\n", f,h,Wfh[f][h] );


        srand((unsigned)(time(NULL)));

        for (int i=0;i<3;i++) {
                for (int j=0;j<3;j++) {



                        // Por alguma razão, a função rand() gera um inteiro randomico
                        // Desta forma dividimos este por um número MAXINT/2, para pegar
                        // um número entre 0 e 2,




                        m_fWeights[i][j] = (float)(rand())/(32767/2) - 1;
                        //fprintf( streamInterno, "m_fWeights[%d][%d] = %f\n",i,j,m_fWeights[i][j]);
                        fprintf( streamInterno, "%f\n",m_fWeights[i][j]);
                }
        }
}

//  Rotina de treinamento da rede

float CBPNet::Train(float i1, float i2, float d) {

        // Todas as variáveis desta subrotina

        float net1, net2, i3, i4, out;
        int s = 1 ;
        int c = 1 ;


        // Calcula os valores da rede para os neurônios da camada escondida.

        net1 = 1 * m_fWeights[0][0] + i1 * m_fWeights[1][0] +
                i2 * m_fWeights[2][0];
        fprintf( streamInterno, "net1 = %f\n",net1);
        net2 = 1 * m_fWeights[0][1] + i1 * m_fWeights[1][1] +
                i2 * m_fWeights[2][1];
```

```
fprintf( streamInterno, "net2 = %f\n",net2);


// Usando a função Hardlimiter - Sigmoid.
i3 = Sigmoid(net1);
i4 = Sigmoid(net2);

fprintf( streamInterno, "i3 = %f\n",i3);
fprintf( streamInterno, "i4 = %f\n",i4);

// Agora, calcula o valor para a saída da última camada.

net1 = 1 * m_fWeights[0][2] + i3 * m_fWeights[1][2] +
            i4 * m_fWeights[2][2];
fprintf( streamInterno, "net1 = %f\n",net1);
out = Sigmoid(net1);
fprintf( streamInterno, "out = %f\n",out);

// Agora é necessário calcular os deltas para as duas últimas
// camadas, vamos calcular os erros para trás da camada de saída
// para as camada escondida. (daí o nome retro-propagação ou
// backpropagation.


float deltas[3];

deltas[2] = out*(1-out)*(d-out);
deltas[1] = i4*(1-i4)*(m_fWeights[2][2])*(deltas[2]);
deltas[0] = i3*(1-i3)*(m_fWeights[1][2])*(deltas[2]);

fprintf( streamInterno, "deltas[2] = %f\n",deltas[2]);
fprintf( streamInterno, "deltas[1] = %f\n",deltas[1]);
fprintf( streamInterno, "deltas[0] = %f\n",deltas[0]);



// Agora, vamos alterar os pesos de acordo.

float v1 = i1, v2 = i2;
for(int i=0;i<3;i++) {

        // vamos trocar os valores da camada de saída, se necessário
        if (i == 2) {
                v1 = i3;
                v2 = i4;
        }

        m_fWeights[0][i] += BP_LEARNING*1*deltas[i];
        m_fWeights[1][i] += BP_LEARNING*v1*deltas[i];
        m_fWeights[2][i] += BP_LEARNING*v2*deltas[i];

        fprintf( streamInterno, "m_fWeights[0][%d] = %f\n",i,m_fWeights[0][i]);
        fprintf( streamInterno, "m_fWeights[1][%d] = %f\n",i,m_fWeights[1][i]);
        fprintf( streamInterno, "m_fWeights[2][%d] = %f\n",i,m_fWeights[2][i]);
}

return out;
```

```
}

float CBPNet::Sigmoid(float num) {
	return (float)(1/(1+exp(-num)));
}

float CBPNet::Run(float i1, float i2) {
	// I just copied and pasted the code from the Train() function,
	// so see there for the necessary documentation.

	float net1, net2, i3, i4;

	net1 = 1 * m_fWeights[0][0] + i1 * m_fWeights[1][0] +
		i2 * m_fWeights[2][0];
	net2 = 1 * m_fWeights[0][1] + i1 * m_fWeights[1][1] +
		i2 * m_fWeights[2][1];

	i3 = Sigmoid(net1);
	i4 = Sigmoid(net2);

	net1 = 1 * m_fWeights[0][2] + i3 * m_fWeights[1][2] +
		i4 * m_fWeights[2][2];
	return Sigmoid(net1);
}
```

Anexo V - Programa Visual Basic XOR

```
Option Explicit
Dim m_fWeights(10, 10) As Double

Const BP_LEARNING As Single = 0.5
Const BPM_ITER As Long = 50000
Dim LigaTracer As Boolean



Private Sub ChkLigarTracer_Click()

   If ChkLigarTracer.Value = 1 Then
      LigaTracer = True
   Else
      LigaTracer = False
   End If

End Sub

Private Sub CmdGravarTrace_Click()

   Dim inti As Integer
   Dim intj As Integer
   Dim aux
   Dim String1 As String
   Dim String2 As String
   Dim String3 As String

   Dim NrEpocas As Integer
   Dim NrColunas As Integer

   String1 = "<html> " _
   & "<head>" _
   & "<body>" _
   & "<TABLE BORDER=""1"" cellpadding=""0"" cellspacing=""0"" style=""border-collapse: collapse""
width=""100"" id=""AutoNumber1""> " _


   String3 = "</TABLE>" _
   & "</body>" _
   & "</head>" _
   & "</html>" _


   Open "C:\TRACE.html" For Output As #1

      Print #1, String1
      Print #1,
"<TR><TD>Epoque</TD><TD>Net1</TD><TD>Net2</TD><TD>Net1</TD><TD>Deltas(2)</TD><T
D>Deltas(1)</TD><TD>Deltas(0)</TD><TD>m_fWeights(0, 0)</TD><TD>m_fWeights(0,
1)</TD><TD>m_fWeights(0, 2)</TD></TR>"

      For NrEpocas = 0 To MSFlexGrid1.Rows - 1
```

```
            Print #1, "<TR>"

            For NrColunas = 0 To MSFlexGrid1.Cols - 1

                MSFlexGrid1.Row = NrEpocas
                MSFlexGrid1.Col = NrColunas

                'Print #1, "<TD>" & MSFlexGrid1.Text & "</TD>"

                Print #1, "<TD WIDTH=""1000"">" & MSFlexGrid1.Text & "</TD>"

            Next NrColunas

            Print #1, "</TR>"

        Next NrEpocas

        Print #1, String3

    Close #1

    Open "C:\TRACE.TXT" For Output As #1

    For inti = 0 To LstTracer.ListCount - 1

        LstTracer.ListIndex = inti
        Print #1, LstTracer.Text

    Next inti

    Close #1



End Sub



'    for (int i=0;i<3;i++) {
'        for (int j=0;j<3;j++) {
'            // Por alguma razão a função Microsoft rand()
'            // gera um número randomico do tipo integer. Desta forma, dividimos
'            // o número por MAXINT/2, para obter um número entre 0 e 2,
'            // então subtraímos um para obter um número entre  -1 e 1.
'            m_fWeights[i][j] = (float)(rand())/(32767/2) - 1;
'            fprintf( streamInterno, "m_fWeights[%d][%d] = %f\n",i,j,m_fWeights[i][j]);
'            //fprintf( streamInterno, "%f\n",m_fWeights[i][j]);
'        }
'    }
'

Private Sub CmdLePesos_Click()

'    Stop

    Dim inti As Integer
```

```
    Dim intj As Integer
    Dim aux

    Open "C:\TRACERINTERNO.TXT" For Input As #1   ' Abre o arquivo para obter os mesmo pesos da
função C++.

    For inti = 0 To 2

        For intj = 0 To 2

            Line Input #1, aux
            m_fWeights(inti, intj) = CDbl(aux)

            Debug.Print m_fWeights(inti, intj)

        Next intj

    Next inti

    Close #1

End Sub

'
'void main() {
'
'    cout << "0,0 = " << bp.Run(0,0) << endl;
'    cout << "0,1 = " << bp.Run(0,1) << endl;
'    cout << "1,0 = " << bp.Run(1,0) << endl;
'    cout << "1,1 = " << bp.Run(1,1) << endl;
'    fclose( stream );
'}

Private Sub CmdRun_Click()

    Dim retorno1 As Double
    Dim retorno2 As Double
    Dim retorno3 As Double
    Dim retorno4 As Double

    retorno1 = Run(0, 0)
    If LigaTracer Then LstTracer.AddItem "Entrada (0,0): Retorno : " & retorno1
    retorno2 = Run(0, 1)
    If LigaTracer Then LstTracer.AddItem "Entrada (0,1): Retorno : " & retorno2
    retorno3 = Run(1, 0)
    If LigaTracer Then LstTracer.AddItem "Entrada (1,0): Retorno :" & retorno3
    retorno4 = Run(1, 1)
    If LigaTracer Then LstTracer.AddItem "Entrada (1,1): Retorno :" & retorno4

End Sub

Private Sub CmdSair_Click()
    End
End Sub

Private Sub CmdTreina_Click()
```

```
    Dim i As Long
    Dim retorno1 As Double
    Dim retorno2 As Double
    Dim retorno3 As Double
    Dim retorno4 As Double
    Dim inti
    Dim intj

    For i = 0 To BPM_ITER

        retorno1 = Train(0, 0, 0)
        retorno2 = Train(0, 1, 1)
        retorno3 = Train(1, 0, 1)
        retorno4 = Train(1, 1, 0)

        'Debug.Print i
    Next i

'     For inti = 0 To 2
'
'         For intj = 0 To 2
'
'             'Debug.Print m_fWeights(inti, intj)
'
'         Next intj
'
'     Next inti
'
End Sub

'float CBPNet::Train(float i1, float i2, float d) {
'    // Estas são todas as variáveis principais usadas na
'    // rotina.
'    float net1, net2, i3, i4, out;
'    int s = 1 ;
'    int c = 1 ;
'
'    // Calcula o valor dos neurônios da camada escondida.
'    net1 = 1 * m_fWeights[0][0] + i1 * m_fWeights[1][0] +
'         i2 * m_fWeights[2][0];
'    fprintf( streamInterno, "net1 = %f\n",net1);
'    net2 = 1 * m_fWeights[0][1] + i1 * m_fWeights[1][1] +
'         i2 * m_fWeights[2][1];
'    fprintf( streamInterno, "net2 = %f\n",net2);
'
'    // Uso da função- a Sigmoid.
'    i3 = Sigmoid(net1);
'    i4 = Sigmoid(net2);
'
'    fprintf( streamInterno, "i3 = %f\n",i3);
'    fprintf( streamInterno, "i4 = %f\n",i4);
'
'    // Agora, calculamos a rede para a última camada, a camada de saída.
'    net1 = 1 * m_fWeights[0][2] + i3 * m_fWeights[1][2] +
'         i4 * m_fWeights[2][2];
'    fprintf( streamInterno, "net1 = %f\n",net1);
'    out = Sigmoid(net1);
```

```
'    fprintf( streamInterno, "out = %f\n",out);
'
'    // Precisamos calcular o deltas para as duas camadas.
'    // Lembre-se, precisamos calcular os erros para trás.
'    // da camada de saída para a camada escondida (daí retropropagação ou
'    // o nome 'BACK-propagation').
'    float deltas[3];
'
'    deltas[2] = out*(1-out)*(d-out);
'    deltas[1] = i4*(1-i4)*(m_fWeights[2][2])*(deltas[2]);
'    deltas[0] = i3*(1-i3)*(m_fWeights[1][2])*(deltas[2]);
'
'    fprintf( streamInterno, "deltas[2] = %f\n",deltas[2]);
'    fprintf( streamInterno, "deltas[1] = %f\n",deltas[1]);
'    fprintf( streamInterno, "deltas[0] = %f\n",deltas[0]);
'
'
'    // Agora, vamos alterar os pesos.
'    float v1 = i1, v2 = i2;
'    for(int i=0;i<3;i++) {
'       // Troquemos os valores da camada de saída, se necessário.
'       if (i == 2) {
'          v1 = i3;
'          v2 = i4;
'       }
'
'       m_fWeights[0][i] += BP_LEARNING*1*deltas[i];
'       m_fWeights[1][i] += BP_LEARNING*v1*deltas[i];
'       m_fWeights[2][i] += BP_LEARNING*v2*deltas[i];
'
'       fprintf( streamInterno, "m_fWeights[0][%d] = %f\n",i,m_fWeights[0][i]);
'       fprintf( streamInterno, "m_fWeights[1][%d] = %f\n",i,m_fWeights[1][i]);
'       fprintf( streamInterno, "m_fWeights[2][%d] = %f\n",i,m_fWeights[2][i]);
'    }
'
'    return out;
'}

Private Function Train(i1 As Single, i2 As Single, d As Single)

    Dim net1 As Double
    Dim net2 As Double
    Dim i3 As Single
    Dim i4 As Single
    Dim out As Single
    Dim i As Integer
    Dim s As Integer
    Dim deltas(10) As Single
    Static contador As Long

    i = 1
    s = 1

    MSFlexGrid1.Cols = 10

    If LigaTracer Then LstTracer.AddItem "Epoque: " & contador
    If LigaTracer Then
```

```
        MSFlexGrid1.ColWidth(0) = 1000
        MSFlexGrid1.Col = 0
        MSFlexGrid1.Text = contador
    End If

    net1 = 1 * m_fWeights(0, 0) + i1 * m_fWeights(1, 0) + i2 * m_fWeights(2, 0)
    If LigaTracer Then LstTracer.AddItem "net1: " & net1
    If LigaTracer Then
        MSFlexGrid1.Rows = MSFlexGrid1.Rows + 1
        MSFlexGrid1.ColWidth(1) = 2000
        MSFlexGrid1.Col = 1
        MSFlexGrid1.Text = net1
    End If

    net2 = 1 * m_fWeights(0, 1) + i1 * m_fWeights(1, 1) + i2 * m_fWeights(2, 1)
    If LigaTracer Then
        MSFlexGrid1.ColWidth(2) = 2000
        MSFlexGrid1.Col = 2
        MSFlexGrid1.Text = net2
    End If
    If LigaTracer Then LstTracer.AddItem "net2: " & net2

    i3 = Sigmoid(net1): 'Debug.Print i3
    i4 = Sigmoid(net2): 'Debug.Print i4

    net1 = 1 * m_fWeights(0, 2) + i3 * m_fWeights(1, 2) + i4 * m_fWeights(2, 2)
    If LigaTracer Then
        MSFlexGrid1.ColWidth(3) = 2000
        MSFlexGrid1.Col = 3
        MSFlexGrid1.Text = net1
    End If

    If LigaTracer Then LstTracer.AddItem "net1: " & net1
    out = Sigmoid(net1)

    deltas(2) = out * (1 - out) * (d - out): If LigaTracer Then LstTracer.AddItem "Deltas(2): " & deltas(2)
    If LigaTracer Then
        MSFlexGrid1.ColWidth(4) = 2000
        MSFlexGrid1.Col = 4
        MSFlexGrid1.Text = deltas(2)
    End If

    deltas(1) = i4 * (1 - i4) * m_fWeights(2, 2) * deltas(2): If LigaTracer Then LstTracer.AddItem
"Deltas(1): " & deltas(1)
    If LigaTracer Then
        MSFlexGrid1.ColWidth(5) = 2000
        MSFlexGrid1.Col = 5
        MSFlexGrid1.Text = deltas(1)
    End If

    deltas(0) = i3 * (1 - i3) * m_fWeights(1, 2) * deltas(2): If LigaTracer Then LstTracer.AddItem
"Deltas(0): " & deltas(0)
    If LigaTracer Then
        MSFlexGrid1.ColWidth(6) = 2000
        MSFlexGrid1.Col = 6
        MSFlexGrid1.Text = deltas(0)
    End If
```

```
    Dim v1 As Single
    Dim v2 As Single

    v1 = i1
    v2 = i2

    For i = 0 To 2

        If i = 2 Then
            v1 = i3
            v2 = i4

        End If

        m_fWeights(0, i) = m_fWeights(0, i) + (BP_LEARNING * 1 * deltas(i)): If LigaTracer Then
LstTracer.AddItem "m_fWeights(0, i): " & m_fWeights(0, i)
        If LigaTracer Then
            MSFlexGrid1.ColWidth(7) = 2000
            MSFlexGrid1.Col = 7
            MSFlexGrid1.Text = m_fWeights(0, i)
        End If

        m_fWeights(1, i) = m_fWeights(1, i) + (BP_LEARNING * v1 * deltas(i)): If LigaTracer Then
LstTracer.AddItem "m_fWeights(1, i): " & m_fWeights(1, i)
        If LigaTracer Then
            MSFlexGrid1.ColWidth(8) = 2000
            MSFlexGrid1.Col = 8
            MSFlexGrid1.Text = m_fWeights(1, i)
        End If

        m_fWeights(2, i) = m_fWeights(2, i) + (BP_LEARNING * v2 * deltas(i)): If LigaTracer Then
LstTracer.AddItem "m_fWeights(2, i): " & m_fWeights(2, i)
        If LigaTracer Then
            MSFlexGrid1.ColWidth(9) = 2000
            MSFlexGrid1.Col = 9
            MSFlexGrid1.Text = m_fWeights(2, i)
        End If

    Next i

    contador = contador + 1

    If LigaTracer Then
        MSFlexGrid1.Row = MSFlexGrid1.Row + 1
    End If


End Function
'float CBPNet::Sigmoid(float num) {
'    return (float)(1/(1+exp(-num)));
'}
'
Private Function Sigmoid(Num)
    Sigmoid = (1 / (1 + Exp(-Num)))
End Function
'float CBPNet::Run(float i1, float i2) {
```

```
'
'    float net1, net2, i3, i4;
'
'    net1 = 1 * m_fWeights[0][0] + i1 * m_fWeights[1][0] +
'         i2 * m_fWeights[2][0];
'    net2 = 1 * m_fWeights[0][1] + i1 * m_fWeights[1][1] +
'         i2 * m_fWeights[2][1];
'
'    i3 = Sigmoid(net1);
'    i4 = Sigmoid(net2);
'
'    net1 = 1 * m_fWeights[0][2] + i3 * m_fWeights[1][2] +
'         i4 * m_fWeights[2][2];
'    return Sigmoid(net1);
'}

Private Function Run(i1 As Double, i2 As Double)

'float CBPNet::Run(float i1, float i2) {
'
'    float net1, net2, i3, i4;

    Dim net1, net2, i3, i4

'
    net1 = 1 * m_fWeights(0, 0) + i1 * m_fWeights(1, 0) + i2 * m_fWeights(2, 0)
    net2 = 1 * m_fWeights(0, 1) + i1 * m_fWeights(1, 1) + i2 * m_fWeights(2, 1)
    i3 = Sigmoid(net1)
    i4 = Sigmoid(net2)

    net1 = 1 * m_fWeights(0, 2) + i3 * m_fWeights(1, 2) + i4 * m_fWeights(2, 2)
    Run = Sigmoid(net1)

End Function

Private Sub Label2_Click()

End Sub

Private Sub Form_Load()
    LigaTracer = False
End Sub


Private Sub LblNet1_Click()

End Sub
```

Anexo VI - Programa C++ - Rede Neural - Cenário Unidimensional

```
/*****************************************************
  CENARIOLINEAR8BITS.h
  --------------------------------------------------
  generated at Thu Oct 25 20:10:09 2001
  by snns2c ( Bernward Kett 1995 )
*****************************************************/

extern int CENARIOLINEAR8BITS(float *in, float *out, int init);

static struct {
  int NoOfInput;    /* Number of Input Units  */
  int NoOfOutput;   /* Number of Output Units */
  int(* propFunc)(float *, float*, int);
} CENARIOLINEAR8BITSREC = {8,2,CENARIOLINEAR8BITS};
```

```
/*****************************************************
  CENARIOLINEAR8BITS.C
  --------------------------------------------------
  generated at Thu Oct 25 20:10:09 2001
  by snns2c ( Bernward Kett 1995 )
*****************************************************/

#include <math.h>
#include <iostream.h>
#include <stdio.h>

#define Act_Logistic(sum, bias)  ( (sum+bias<10000.0) ? ( 1.0/(1.0 + exp(-sum-bias) ) ) : 0.0 )
#define NULL (void *)0

typedef struct UT {
        float act;        /* Activation       */
        float Bias;       /* Bias of the Unit */
        int   NoOfSources; /* Number of predecessor units */
   struct UT   **sources; /* predecessor units */
        float *weights; /* weights from predecessor units */
       } UnitType, *pUnit;

  /* Forward Declaration for all unit types */
  static UnitType Units[19];
  /* Sources definition section */
  static pUnit Sources[] =  {
Units + 1, Units + 2, Units + 3, Units + 4, Units + 5, Units + 6, Units + 7, Units + 8,
Units + 1, Units + 2, Units + 3, Units + 4, Units + 5, Units + 6, Units + 7, Units + 8,
Units + 1, Units + 2, Units + 3, Units + 4, Units + 5, Units + 6, Units + 7, Units + 8,
Units + 1, Units + 2, Units + 3, Units + 4, Units + 5, Units + 6, Units + 7, Units + 8,
Units + 9, Units + 10, Units + 11, Units + 12,
Units + 9, Units + 10, Units + 11, Units + 12,
Units + 9, Units + 10, Units + 11, Units + 12,
Units + 9, Units + 10, Units + 11, Units + 12,
Units + 13, Units + 14, Units + 15, Units + 16,
```

```
Units + 13, Units + 14, Units + 15, Units + 16,

  };

  /* Weigths definition section */
  static float Weights[] =  {
9.573720, -6.192420, -0.639350, 0.676990, -8.385350, 5.501380, 2.459650, 13.007640,
9.454750, 10.065200, -0.628130, -0.582540, -0.325240, -0.570550, -0.001880, 0.084680,
5.582220, 6.871080, 1.564490, 2.623190, 1.731400, -0.058660, 0.003150, 10.316350,
-17.353230, -7.076250, -12.384120, -5.317850, -3.369540, -2.191020, -1.255130, -0.400130,
1.753800, 17.990789, 8.644780, -25.577379,
-4.035550, -5.034760, -5.323770, 22.898239,
-6.911460, 18.673479, -1.384770, 6.151530,
-7.190810, -1.241670, -12.713690, 5.608540,
-27.389990, -25.502769, 17.579580, -17.345030,
22.231001, -5.974790, -8.932710, -2.708330,

  };

  /* unit definition section (see also UnitType) */
  static UnitType Units[19] =
  {
    { 0.0, 0.0, 0, NULL , NULL },
    { /* unit 1 (Old: 1) */
      0.0, 0.913640, 0,
       &Sources[0] ,
       &Weights[0] ,
      },
    { /* unit 2 (Old: 2) */
      0.0, 0.678830, 0,
       &Sources[0] ,
       &Weights[0] ,
      },
    { /* unit 3 (Old: 3) */
      0.0, -0.044700, 0,
       &Sources[0] ,
       &Weights[0] ,
      },
    { /* unit 4 (Old: 4) */
      0.0, -0.393160, 0,
       &Sources[0] ,
       &Weights[0] ,
      },
    { /* unit 5 (Old: 5) */
      0.0, 0.805800, 0,
       &Sources[0] ,
       &Weights[0] ,
      },
    { /* unit 6 (Old: 6) */
      0.0, 0.198310, 0,
       &Sources[0] ,
       &Weights[0] ,
      },
    { /* unit 7 (Old: 7) */
      0.0, -0.546860, 0,
       &Sources[0] ,
       &Weights[0] ,
```

```
  },
{ /* unit 8 (Old: 8) */
  0.0, 0.510350, 0,
   &Sources[0] ,
   &Weights[0] ,
  },
{ /* unit 9 (Old: 9) */
  0.0, -1.250740, 8,
   &Sources[0] ,
   &Weights[0] ,
  },
{ /* unit 10 (Old: 10) */
  0.0, -9.287780, 8,
   &Sources[8] ,
   &Weights[8] ,
  },
{ /* unit 11 (Old: 11) */
  0.0, -7.925180, 8,
   &Sources[16] ,
   &Weights[16] ,
  },
{ /* unit 12 (Old: 12) */
  0.0, 35.320938, 8,
   &Sources[24] ,
   &Weights[24] ,
  },
{ /* unit 13 (Old: 13) */
  0.0, 2.466220, 4,
   &Sources[32] ,
   &Weights[32] ,
  },
{ /* unit 14 (Old: 14) */
  0.0, 5.208980, 4,
   &Sources[36] ,
   &Weights[36] ,
  },
{ /* unit 15 (Old: 15) */
  0.0, 0.148150, 4,
   &Sources[40] ,
   &Weights[40] ,
  },
{ /* unit 16 (Old: 16) */
  0.0, 0.650570, 4,
   &Sources[44] ,
   &Weights[44] ,
  },
{ /* unit 17 (Old: 17) */
  0.0, 16.108841, 4,
   &Sources[48] ,
   &Weights[48] ,
  },
{ /* unit 18 (Old: 18) */
  0.0, -0.802360, 4,
   &Sources[52] ,
   &Weights[52] ,
  }
```

```
 };


int CENARIOLINEAR8BITS(float *in, float *out, int init)
{
 int member, source;
 float sum;
 enum{OK, Error, Not_Valid};
 pUnit unit;


 /* layer definition section (names & member units) */

 static pUnit Input[8] = {Units + 1, Units + 2, Units + 3, Units + 4, Units + 5, Units + 6, Units + 7, Units +
8}; /* members */

 static pUnit Hidden1[4] = {Units + 9, Units + 10, Units + 11, Units + 12}; /* members */

 static pUnit Hidden2[4] = {Units + 13, Units + 14, Units + 15, Units + 16}; /* members */

 static pUnit Output1[2] = {Units + 17, Units + 18}; /* members */

 static int Output[2] = {17, 18};

 for(member = 0; member < 8; member++) {
  Input[member]->act = in[member];
 }

 for (member = 0; member < 4; member++) {
  unit = Hidden1[member];
  sum = 0.0;
  for (source = 0; source < unit->NoOfSources; source++) {
   sum += unit->sources[source]->act
        * unit->weights[source];
  }
  unit->act = Act_Logistic(sum, unit->Bias);
 };

 for (member = 0; member < 4; member++) {
  unit = Hidden2[member];
  sum = 0.0;
  for (source = 0; source < unit->NoOfSources; source++) {
   sum += unit->sources[source]->act
        * unit->weights[source];
  }
  unit->act = Act_Logistic(sum, unit->Bias);
 };

 for (member = 0; member < 2; member++) {
  unit = Output1[member];
  sum = 0.0;
  for (source = 0; source < unit->NoOfSources; source++) {
   sum += unit->sources[source]->act
        * unit->weights[source];
  }
  unit->act = Act_Logistic(sum, unit->Bias);
```

```
  };

  for(member = 0; member < 2; member++) {
    out[member] = Units[Output[member]].act;
  }

  return(OK);
}

void main()
{

#include "CENARIOLINEAR8BITS.h"

//float *netInput, *netOutput;

// Vetores de Entrada para teste de convergência.

//float netInput[] = {0, 0, 0, 0};
//float netInput[] = {0, 0, 0, 1};
//float netInput[] = {0, 0 ,1, 0};
//float netInput[] = {0, 0, 1, 1};
//float netInput[] = {0, 1, 0, 0};
//float netInput[] = {0, 1, 0, 1};
//float netInput[] = {0, 1, 1, 0};
//float netInput[] = {0, 1, 1, 1};
//float netInput[] = {1, 0, 0, 0};
//float netInput[] = {1, 0, 0, 1};
//float netInput[] = {1, 0, 1, 0};
//float netInput[] = {1, 0, 1, 1};
//float netInput[] = {1, 1, 0, 0};
//float netInput[] = {1, 1, 0, 1};
//float netInput[] = {1, 1, 1, 0};

float netOutput[] = {0, 0};

CENARIOLINEAR8BITS(netInput, netOutput, 0);

printf( "netOutput 0 %f\n",netOutput[0]);
printf( "netOutput 1 %f\n",netOutput[1]);

return;
}
```

Anexo VII - Programa C++ - Rede Neural – Cenário Bidimensional

```
/*****************************************************
  C:\CENARIOBIDIMENSIONAL_400000.h
  -------------------------------------------------
  generated at Sun Oct 07 08:14:05 2001
  by snns2c ( Bernward Kett 1995 )
*****************************************************/

extern int C__CENARIOBIDIMENSIONAL_400000(float *in, float *out, int init);

static struct {
  int NoOfInput;    /* Number of Input Units  */
  int NoOfOutput;   /* Number of Output Units */
  int(* propFunc)(float *, float*, int);
} C__CENARIOBIDIMENSIONAL_400000REC = {8,2,CENARIOBIDIMENSIONAL_400000};
```

```
/*****************************************************
  C:\CENARIOBIDIMENSIONAL_400000.C
  -------------------------------------------------
  generated at Sun Oct 07 08:14:01 2001
  by snns2c ( Bernward Kett 1995 )
*****************************************************/

#include <math.h>
#include <iostream.h>
#include <stdio.h>

#define Act_Logistic(sum, bias)  ( (sum+bias<10000.0) ? ( 1.0/(1.0 + exp(-sum-bias) ) ) : 0.0 )
#define NULL (void *)0

typedef struct UT {
        float act;        /* Activation       */
        float Bias;       /* Bias of the Unit */
        int   NoOfSources; /* Number of predecessor units */
   struct UT   **sources; /* predecessor units */
        float *weights; /* weights from predecessor units */
       } UnitType, *pUnit;

  /* Forward Declaration for all unit types */
  static UnitType Units[21];
  /* Sources definition section */
  static pUnit Sources[] =  {
Units + 1, Units + 2, Units + 3, Units + 4, Units + 5, Units + 6, Units + 7, Units + 8,
Units + 1, Units + 2, Units + 3, Units + 4, Units + 5, Units + 6, Units + 7, Units + 8,
Units + 1, Units + 2, Units + 3, Units + 4, Units + 5, Units + 6, Units + 7, Units + 8,
Units + 1, Units + 2, Units + 3, Units + 4, Units + 5, Units + 6, Units + 7, Units + 8,
Units + 1, Units + 2, Units + 3, Units + 4, Units + 5, Units + 6, Units + 7, Units + 8,
Units + 1, Units + 2, Units + 3, Units + 4, Units + 5, Units + 6, Units + 7, Units + 8,
Units + 9, Units + 10, Units + 11, Units + 12, Units + 13, Units + 14,
```

```
Units + 9, Units + 10, Units + 11, Units + 12, Units + 13, Units + 14,
Units + 9, Units + 10, Units + 11, Units + 12, Units + 13, Units + 14,
Units + 9, Units + 10, Units + 11, Units + 12, Units + 13, Units + 14,
Units + 15, Units + 16, Units + 17, Units + 18,
Units + 15, Units + 16, Units + 17, Units + 18,

  };

  /* Weigths definition section */
  static float Weights[] =  {
5.059450, 2.589520, -1.231750, -1.231820, 5.686460, -1.371800, 11.411680, 7.943880,
12.362350, -7.723430, -0.796120, -0.796140, 6.578540, 1.804130, 0.278710, 0.299260,
2.267210, -2.194240, -3.303880, -3.306100, 4.206430, 1.113950, 3.127970, 0.815570,
-0.405970, -1.432210, -0.758590, -0.758280, 4.435000, 11.607450, 0.863400, 1.291060,
-17.102989, 9.305900, 3.678140, 3.678240, 6.040980, 4.136400, -0.365990, 2.619040,
-4.494450, -5.534120, 2.520110, 2.520340, -2.949330, -12.742160, 5.036270, 4.049190,
4.687560, 19.431240, 4.238930, -16.063250, 14.285140, -4.618010,
-7.850080, 14.297390, -3.026780, 1.525300, 10.361930, -10.832140,
-6.083150, 14.997470, -1.175720, 0.646360, 1.534320, -1.465590,
-16.391060, -3.203920, 6.791640, 9.225670, -7.324360, 2.012630,
-29.777769, -0.400810, 13.335210, -26.593451,
-10.494160, -25.278250, 3.494980, 33.858730,

  };

  /* unit definition section (see also UnitType) */
  static UnitType Units[21] =
  {
    { 0.0, 0.0, 0, NULL , NULL },
    { /* unit 1 (Old: 1) */
      0.0, -0.299620, 0,
       &Sources[0] ,
       &Weights[0] ,
      },
    { /* unit 2 (Old: 2) */
      0.0, -0.797000, 0,
       &Sources[0] ,
       &Weights[0] ,
      },
    { /* unit 3 (Old: 3) */
      0.0, -0.189230, 0,
       &Sources[0] ,
       &Weights[0] ,
      },
    { /* unit 4 (Old: 4) */
      0.0, 0.221200, 0,
       &Sources[0] ,
       &Weights[0] ,
      },
    { /* unit 5 (Old: 5) */
      0.0, 0.060050, 0,
       &Sources[0] ,
       &Weights[0] ,
      },
    { /* unit 6 (Old: 6) */
      0.0, 0.620850, 0,
       &Sources[0] ,
```

```
  &Weights[0] ,
  },
{ /* unit 7 (Old: 7) */
  0.0, 0.747030, 0,
  &Sources[0] ,
  &Weights[0] ,
  },
{ /* unit 8 (Old: 8) */
  0.0, 0.063980, 0,
  &Sources[0] ,
  &Weights[0] ,
  },
{ /* unit 9 (Old: 9) */
  0.0, -9.041830, 8,
  &Sources[0] ,
  &Weights[0] ,
  },
{ /* unit 10 (Old: 10) */
  0.0, 0.200760, 8,
  &Sources[8] ,
  &Weights[8] ,
  },
{ /* unit 11 (Old: 11) */
  0.0, -7.618930, 8,
  &Sources[16] ,
  &Weights[16] ,
  },
{ /* unit 12 (Old: 12) */
  0.0, -1.800960, 8,
  &Sources[24] ,
  &Weights[24] ,
  },
{ /* unit 13 (Old: 13) */
  0.0, 2.021250, 8,
  &Sources[32] ,
  &Weights[32] ,
  },
{ /* unit 14 (Old: 14) */
  0.0, 2.855080, 8,
  &Sources[40] ,
  &Weights[40] ,
  },
{ /* unit 15 (Old: 15) */
  0.0, -9.940430, 6,
  &Sources[48] ,
  &Weights[48] ,
  },
{ /* unit 16 (Old: 16) */
  0.0, 0.670550, 6,
  &Sources[54] ,
  &Weights[54] ,
  },
{ /* unit 17 (Old: 17) */
  0.0, -3.501470, 6,
  &Sources[60] ,
  &Weights[60] ,
  },
```

```
   { /* unit 18 (Old: 18) */
     0.0, -0.275780, 6,
      &Sources[66] ,
      &Weights[66] ,
     },
   { /* unit 19 (Old: 19) */
     0.0, 8.044560, 4,
      &Sources[72] ,
      &Weights[72] ,
     },
   { /* unit 20 (Old: 20) */
     0.0, 16.836029, 4,
      &Sources[76] ,
      &Weights[76] ,
     }

  };


int  CENARIOBIDIMENSIONAL_400000(float *in, float *out, int init)
{
  int member, source;
  float sum;
  enum{OK, Error, Not_Valid};
  pUnit unit;


  /* layer definition section (names & member units) */

  static pUnit Input[8] = {Units + 1, Units + 2, Units + 3, Units + 4, Units + 5, Units + 6, Units + 7, Units +
8}; /* members */

  static pUnit Hidden1[6] = {Units + 9, Units + 10, Units + 11, Units + 12, Units + 13, Units + 14}; /*
members */

  static pUnit Hidden2[4] = {Units + 15, Units + 16, Units + 17, Units + 18}; /* members */

  static pUnit Output1[2] = {Units + 19, Units + 20}; /* members */

  static int Output[2] = {19, 20};

  for(member = 0; member < 8; member++) {
    Input[member]->act = in[member];
  }

  for (member = 0; member < 6; member++) {
    unit = Hidden1[member];
    sum = 0.0;
    for (source = 0; source < unit->NoOfSources; source++) {
      sum += unit->sources[source]->act
           * unit->weights[source];
    }
    unit->act = Act_Logistic(sum, unit->Bias);
  };

  for (member = 0; member < 4; member++) {
    unit = Hidden2[member];
```

```
    sum = 0.0;
    for (source = 0; source < unit->NoOfSources; source++) {
      sum += unit->sources[source]->act
            * unit->weights[source];
    }
    unit->act = Act_Logistic(sum, unit->Bias);
   };

   for (member = 0; member < 2; member++) {
    unit = Output1[member];
    sum = 0.0;
    for (source = 0; source < unit->NoOfSources; source++) {
      sum += unit->sources[source]->act
            * unit->weights[source];
    }
    unit->act = Act_Logistic(sum, unit->Bias);
   };

   for(member = 0; member < 2; member++) {
    out[member] = Units[Output[member]].act;
   }

   return(OK);
}

void main()
{

#include "CENARIOBIDIMENSIONAL_400000.h"

//float *netInput, *netOutput;
//float netInput[] = {1, 1, 1, 1, 1, 1, 1, 1};
//float netInput[] = {0, 1, 1, 0, 1, 0, 1, 1};
//float netInput[] = {0, 1, 0, 0, 0, 0, 0, 1};

float netInput[] = {0, 1, 0, 0, 0, 1, 1, 0};

float netOutput[] = {0, 0};

CENARIOBIDIMENSIONAL_400000(netInput, netOutput, 0);

printf( "netOutput 0 %f\n",netOutput[0]);
printf( "netOutput 1 %f\n",netOutput[1]);

return;
}
```

Anexo VIII - Programa C++ - Rede Neural – Cenário Tridimensional

```
/*********************************************************
  CENARIOTRIDIMENSIONAL_500000.h
  --------------------------------------------------------
  generated at Fri Oct 26 16:19:21 2001
  by snns2c ( Bernward Kett 1995 )
*********************************************************/

extern int CENARIOTRIDIMENSIONAL_500000(float *in, float *out, int init);

static struct {
  int NoOfInput;    /* Number of Input Units  */
  int NoOfOutput;   /* Number of Output Units */
  int(* propFunc)(float *, float*, int);
} CENARIOTRIDIMENSIONAL_500000REC = {12,2,CENARIOTRIDIMENSIONAL_500000};
```

```
/*********************************************************
  CENARIOTRIDIMENSIONAL_500000.C
  --------------------------------------------------------
  generated at Fri Oct 26 16:19:21 2001
  by snns2c ( Bernward Kett 1995 )
*********************************************************/

#include <math.h>

#define Act_Logistic(sum, bias)  ( (sum+bias<10000.0) ? ( 1.0/(1.0 + exp(-sum-bias) ) ) : 0.0 )
#define NULL (void *)0

typedef struct UT {
          float act;         /* Activation       */
          float Bias;        /* Bias of the Unit */
          int   NoOfSources; /* Number of predecessor units */
   struct UT   **sources; /* predecessor units */
          float *weights; /* weights from predecessor units */
        } UnitType, *pUnit;

  /* Forward Declaration for all unit types */
  static UnitType Units[31];
  /* Sources definition section */
  static pUnit Sources[] =  {
Units + 1, Units + 2, Units + 3, Units + 4, Units + 5, Units + 6, Units + 7, Units + 8, Units + 9, Units + 10,
Units + 11, Units + 12,
Units + 1, Units + 2, Units + 3, Units + 4, Units + 5, Units + 6, Units + 7, Units + 8, Units + 9, Units + 10,
Units + 11, Units + 12,
Units + 1, Units + 2, Units + 3, Units + 4, Units + 5, Units + 6, Units + 7, Units + 8, Units + 9, Units + 10,
Units + 11, Units + 12,
Units + 1, Units + 2, Units + 3, Units + 4, Units + 5, Units + 6, Units + 7, Units + 8, Units + 9, Units + 10,
Units + 11, Units + 12,
Units + 1, Units + 2, Units + 3, Units + 4, Units + 5, Units + 6, Units + 7, Units + 8, Units + 9, Units + 10,
Units + 11, Units + 12,
Units + 1, Units + 2, Units + 3, Units + 4, Units + 5, Units + 6, Units + 7, Units + 8, Units + 9, Units + 10,
Units + 11, Units + 12,
```

```
Units + 1, Units + 2, Units + 3, Units + 4, Units + 5, Units + 6, Units + 7, Units + 8, Units + 9, Units + 10,
Units + 11, Units + 12,
Units + 1, Units + 2, Units + 3, Units + 4, Units + 5, Units + 6, Units + 7, Units + 8, Units + 9, Units + 10,
Units + 11, Units + 12,
Units + 13, Units + 14, Units + 15, Units + 16, Units + 17, Units + 18, Units + 19, Units + 20,
Units + 13, Units + 14, Units + 15, Units + 16, Units + 17, Units + 18, Units + 19, Units + 20,
Units + 13, Units + 14, Units + 15, Units + 16, Units + 17, Units + 18, Units + 19, Units + 20,
Units + 13, Units + 14, Units + 15, Units + 16, Units + 17, Units + 18, Units + 19, Units + 20,
Units + 13, Units + 14, Units + 15, Units + 16, Units + 17, Units + 18, Units + 19, Units + 20,
Units + 13, Units + 14, Units + 15, Units + 16, Units + 17, Units + 18, Units + 19, Units + 20,
Units + 13, Units + 14, Units + 15, Units + 16, Units + 17, Units + 18, Units + 19, Units + 20,
Units + 13, Units + 14, Units + 15, Units + 16, Units + 17, Units + 18, Units + 19, Units + 20,
Units + 21, Units + 22, Units + 23, Units + 24, Units + 25, Units + 26, Units + 27, Units + 28,
Units + 21, Units + 22, Units + 23, Units + 24, Units + 25, Units + 26, Units + 27, Units + 28,

  };

  /* Weigths definition section */
  static float Weights[] =  {
-2.303170, -3.886570, -2.964470, -2.964120, 15.211060, 5.439280, 5.405880, 3.296220, -1.057440, -
1.056890,
-1.056380, 0.000560,
-3.501240, -1.657400, -1.894850, -1.906790, -2.148700, -0.556060, 0.189700, -1.648820, 12.275490,
12.275370,
12.275290, -0.010240,
1.994360, 1.516830, 1.215880, 1.219300, -8.245720, -11.320030, 4.751110, 2.406390, 0.083040,
0.082910,
0.082720, -0.001030,
1.696880, 2.637510, -0.625730, -0.612370, -0.194970, -10.427970, 5.427870, 5.305740, -0.158110, -
0.157860,
-0.157590, -0.011450,
11.395750, 12.341440, -0.966530, -0.967880, -1.617500, -0.955120, -0.509440, -0.945740, -0.032280, -
0.032330,
-0.032360, -0.001480,
-2.552050, -0.093290, -5.825150, -5.724060, 8.390380, 12.658330, -4.824680, -2.182620, 1.513640,
1.513520,
1.513480, -0.000760,
-20.077579, -22.041880, 2.186400, 2.133990, 2.396960, 0.516540, 0.414380, 1.388230, 9.250480,
9.250400,
9.250240, 0.000900,
-4.579400, -2.615570, 1.486400, 1.514370, -0.647540, 13.134750, -11.876450, -9.416340, -1.628040, -
1.627830,
-1.627540, -0.001510,
0.813950, -9.814720, -3.020250, 2.276210, -2.445900, 4.066770, 8.585450, -4.854790,
3.316290, -2.682680, -9.343970, 6.604810, 2.629340, -2.242700, 2.105150, 6.005630,
11.486950, -10.840360, -6.401090, -2.275490, 11.866560, 4.741810, 10.399510, -9.632160,
-8.132050, -0.366180, 1.380960, 8.074990, -1.112940, -1.805250, -1.312340, -1.179610,
11.391550, 2.485760, -6.171580, -3.161580, 6.754560, 1.733270, -0.873380, -3.289170,
-12.626810, -8.888650, 6.013450, -3.851510, 7.608070, -1.250660, 8.755430, -3.162660,
-11.460120, 0.041960, -4.818130, 11.816840, 12.121500, -1.447970, 1.602690, 7.181830,
2.484090, -10.045640, 4.467000, -9.702370, 11.800630, 7.566940, 6.909730, -7.344800,
-16.050819, -9.578090, -13.102650, -6.112310, 9.518550, -8.148190, -20.104851, 0.414220,
-9.438700, 8.656700, -18.006599, 6.489770, 9.809340, -17.957960, 9.692740, -18.757521,

  };

  /* unit definition section (see also UnitType) */
```

```
static UnitType Units[31] =
{
  { 0.0, 0.0, 0, NULL , NULL },
  { /* unit 1 (Old: 1) */
    0.0, 0.500270, 0,
     &Sources[0] ,
     &Weights[0] ,
    },
  { /* unit 2 (Old: 2) */
    0.0, -0.738940, 0,
     &Sources[0] ,
     &Weights[0] ,
    },
  { /* unit 3 (Old: 3) */
    0.0, 0.912440, 0,
     &Sources[0] ,
     &Weights[0] ,
    },
  { /* unit 4 (Old: 4) */
    0.0, -0.116200, 0,
     &Sources[0] ,
     &Weights[0] ,
    },
  { /* unit 5 (Old: 5) */
    0.0, -0.326400, 0,
     &Sources[0] ,
     &Weights[0] ,
    },
  { /* unit 6 (Old: 6) */
    0.0, 0.313820, 0,
     &Sources[0] ,
     &Weights[0] ,
    },
  { /* unit 7 (Old: 7) */
    0.0, 0.681660, 0,
     &Sources[0] ,
     &Weights[0] ,
    },
  { /* unit 8 (Old: 8) */
    0.0, 0.220020, 0,
     &Sources[0] ,
     &Weights[0] ,
    },
  { /* unit 9 (Old: 9) */
    0.0, -0.127400, 0,
     &Sources[0] ,
     &Weights[0] ,
    },
  { /* unit 10 (Old: 10) */
    0.0, -0.719330, 0,
     &Sources[0] ,
     &Weights[0] ,
    },
  { /* unit 11 (Old: 11) */
    0.0, 0.288580, 0,
     &Sources[0] ,
     &Weights[0] ,
```

```
  },
{ /* unit 12 (Old: 12) */
  0.0, 0.886200, 0,
   &Sources[0] ,
   &Weights[0] ,
  },
{ /* unit 13 (Old: 13) */
  0.0, -1.338140, 12,
   &Sources[0] ,
   &Weights[0] ,
  },
{ /* unit 14 (Old: 14) */
  0.0, -2.020580, 12,
   &Sources[12] ,
   &Weights[12] ,
  },
{ /* unit 15 (Old: 15) */
  0.0, 7.074260, 12,
   &Sources[24] ,
   &Weights[24] ,
  },
{ /* unit 16 (Old: 16) */
  0.0, -3.745340, 12,
   &Sources[36] ,
   &Weights[36] ,
  },
{ /* unit 17 (Old: 17) */
  0.0, -10.221550, 12,
   &Sources[48] ,
   &Weights[48] ,
  },
{ /* unit 18 (Old: 18) */
  0.0, -4.187140, 12,
   &Sources[60] ,
   &Weights[60] ,
  },
{ /* unit 19 (Old: 19) */
  0.0, -5.799820, 12,
   &Sources[72] ,
   &Weights[72] ,
  },
{ /* unit 20 (Old: 20) */
  0.0, -0.907370, 12,
   &Sources[84] ,
   &Weights[84] ,
  },
{ /* unit 21 (Old: 21) */
  0.0, 5.746170, 8,
   &Sources[96] ,
   &Weights[96] ,
  },
{ /* unit 22 (Old: 22) */
  0.0, 2.020900, 8,
   &Sources[104] ,
   &Weights[104] ,
  },
{ /* unit 23 (Old: 23) */
```

```
    0.0, 0.827180, 8,
     &Sources[112] ,
     &Weights[112] ,
    },
   { /* unit 24 (Old: 24) */
    0.0, -1.600570, 8,
     &Sources[120] ,
     &Weights[120] ,
    },
   { /* unit 25 (Old: 25) */
    0.0, -3.333140, 8,
     &Sources[128] ,
     &Weights[128] ,
    },
   { /* unit 26 (Old: 26) */
    0.0, 1.150340, 8,
     &Sources[136] ,
     &Weights[136] ,
    },
   { /* unit 27 (Old: 27) */
    0.0, 0.911550, 8,
     &Sources[144] ,
     &Weights[144] ,
    },
   { /* unit 28 (Old: 28) */
    0.0, 4.557150, 8,
     &Sources[152] ,
     &Weights[152] ,
    },
   { /* unit 29 (Old: 29) */
    0.0, 11.993120, 8,
     &Sources[160] ,
     &Weights[160] ,
    },
   { /* unit 30 (Old: 30) */
    0.0, 0.573680, 8,
     &Sources[168] ,
     &Weights[168] ,
    }

  };

int CENARIOTRIDIMENSIONAL_500000(float *in, float *out, int init)
{
  int member, source;
  float sum;
  enum{OK, Error, Not_Valid};
  pUnit unit;


  /* layer definition section (names & member units) */

  static pUnit Input[12] = {Units + 1, Units + 2, Units + 3, Units + 4, Units + 5, Units + 6, Units + 7, Units
+ 8, Units + 9, Units + 10, Units + 11, Units + 12}; /* members */

  static pUnit Hidden1[8] = {Units + 13, Units + 14, Units + 15, Units + 16, Units + 17, Units + 18, Units +
19, Units + 20}; /* members */
```

```
  static pUnit Hidden2[8] = {Units + 21, Units + 22, Units + 23, Units + 24, Units + 25, Units + 26, Units +
27, Units + 28}; /* members */

  static pUnit Output1[2] = {Units + 29, Units + 30}; /* members */

  static int Output[2] = {29, 30};

  for(member = 0; member < 12; member++) {
   Input[member]->act = in[member];
  }

  for (member = 0; member < 8; member++) {
   unit = Hidden1[member];
   sum = 0.0;
   for (source = 0; source < unit->NoOfSources; source++) {
    sum += unit->sources[source]->act
         * unit->weights[source];
   }
   unit->act = Act_Logistic(sum, unit->Bias);
  };

  for (member = 0; member < 8; member++) {
   unit = Hidden2[member];
   sum = 0.0;
   for (source = 0; source < unit->NoOfSources; source++) {
    sum += unit->sources[source]->act
         * unit->weights[source];
   }
   unit->act = Act_Logistic(sum, unit->Bias);
  };

  for (member = 0; member < 2; member++) {
   unit = Output1[member];
   sum = 0.0;
   for (source = 0; source < unit->NoOfSources; source++) {
    sum += unit->sources[source]->act
         * unit->weights[source];
   }
   unit->act = Act_Logistic(sum, unit->Bias);
  };

  for(member = 0; member < 2; member++) {
   out[member] = Units[Output[member]].act;
  }

  return(OK);
}

void main()
{

#include "CENARIOTRIDIMENSIONAL_500000.h"

//float *netInput, *netOutput;
//float netInput[] = {1, 1, 1, 1, 1, 1, 1, 1};
//float netInput[] = {0, 1, 1, 0, 1, 0, 1, 1};
```

```
//float netInput[] = {0, 1, 0, 0, 0, 0, 0, 1};
// float netInput[] = {0, 1, 0, 0, 0, 1, 1, 0};

//float netInput[] = {1,0,1,1,1,0,0,0,0,0,0,0};
//float netInput[] = {0,1,1,1,0,1,1,0,0,0,0,1};
float netInput[] = {0,1,1,1,0,1,0,1,0,1,1,0};

float netOutput[] = {0, 0};

CENARIOTRIDIMENSIONAL_500000(netInput, netOutput, 0);

printf( "netOutput 0 %f\n",netOutput[0]);
printf( "netOutput 1 %f\n",netOutput[1]);

return;
}
```