REDES NEURAIS APLICADAS À AQUISIÇÃO DO CONHECIMENTO DE ESPECIALISTAS EM SISTEMAS ELÉTRICOS DE POTÊNCIA

Marcello Baptista de Martino

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

Aprovada por:

Prof. Luis Alfredo Vidal de Carvalho, D.Sc.
(PRESIDENTE)

Profª. Sueli Bandeira Teixeira Mendes, Ph.D.

Marcus Theodor Schilling, D.Sc.

Rio de Janeiro, RJ - BRASIL
Abril de 1994

MARTINO, MARCELLO BAPTISTA DE
Redes Neurais aplicadas à Aquisição do
Conhecimento de Especialistas em Sistemas
Elétricos de Potência [Rio de Janeiro, 1994]
VIII, 239p. 29,7cm (COPPE-UFRJ, M.Sc.,
Engenharia de Sistemas e Computação, 1994)
Tese - Universidade Federal do Rio de Janeiro,
COPPE
1.Redes Neurais 2.Sistemas Elétricos de Potência
3.Redes Neurais Parcialmente Recursivas
I.COPPE-UFRJ II.Título (Série)

A Adriana, Fabricio e Sônia.

Agradecimentos:

Inicialmente, é preciso agradecer ao meu orientador Luís Alfredo pela oportunidade, ao doutor Marcus pela sua confiança, e à professora Sueli pela sua boa vontade.

Agradeço também aos companheiros do CEPEL: Antônio Joaquim, por ter permitido que me dedicasse à tese nesses últimos meses; Guilherme Nelson pelas discussões; Maria Fernanda pelos conselhos; Aroldo, Cláudio e Victor pela amizade.

É impossível deixar de agradecer às pessoas que conviveram comigo ao longo destes quatro anos: minha namorada pela sua compreensão nestes anos e por seu incansável trabalho de revisão ortográfica; minha mãe pelo apoio e por não me expulsar de casa apesar de toda a bagunça; e meu irmão pelo auxílio nos momentos críticos.

Finalmente, agradeço a todos aqueles que, direta ou indiretamente, contribuiram para a realização desta tese.

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

REDES NEURAIS APLICADAS À AQUISIÇÃO DO CONHECIMENTO EM SISTEMAS ELÉTRICOS DE POTÊNCIA

Marcello Baptista de Martino

Abril de 1994

Orientador: Luís Alfredo Vidal de Carvalho

Programa: Engenharia de Sistemas e Computação

Os órgãos internacionais qualificados têm reportado dificuldades sistemáticas nas aplicações existentes de **Inteligência Artificial** em **Sistemas Elétricos de Potência**, principalmente na aquisição do conhecimento de especialistas e na manutenção dos sistemas implantados. Esta tese sugere que estas dificuldades podem ser resolvidas através da utilização de **Redes Neurais**.

Inicialmente, mostramos que os especialistas utilizam na resolução de parte dos problemas dos Sistemas Elétricos de Potência, como a Detecção de Falhas em Transformadores, alguma forma de **Classificação**. Estudamos, portanto, diversos paradigmas de Redes Neurais com ênfase na sua aplicação em problemas de Classificação. Apresentamos diversas formas de representação de problemas para uma Rede Neural e a influência destas em seu comportamento, principalmente quando da ausência de dados completos. Desenvolvemos uma nova arquitetura parcialmente recursiva, baseada nos modelos de *Jordan(89)* e *Elman(90)*. Selecionamos o algoritmo **Back-Propagation Through Time** como o mais indicado para ser utilizado na arquitetura proposta. Finalmente, comprovamos sua eficácia na Detecção de Falhas em Transformadores e em um exemplo de classificação fornecido por *Telfer(91)*.

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

NEURAL NETWORKS ON ELECTRIC POWER SYSTEMS

Marcello Baptista de Martino

April, 1994

Thesis Supervisor: Luís Alfredo Vidal de Carvalho

Department: Systems Engineering and Computer Science

The international qualified institutes have been reporting systematic dificulties in the actual aplications on **Artificial Inteligence** applied to **Electric Power Systems**, mostly in knowledge aquisition from experts and present systems support. This thesis suggest that these difficulties can be solved by the utilization of **Neural Networks**.

First, we show that experts solve most of the Electric Power Systems problems, as the Transformer's Fault Detection, using some sort of **Classification**. So, paradigms for Neural Networks are studied, giving special emphasis to their Classification applicability. We show many mechanisms to represent problems on a Neural Network and their influence over it's behavior specially when missing data in training and recall patterns exists. A new architecture, partially recursive, based on *Jordan's(89)* and *Elman's(90)* models, is developed. The **Back-Propagation Through Time** algorithm is indicated to be used in the suggested architecture. And last, but not least, we check out it's efficiency in the Transformer's Fault Detection and in a classification example given by *Telfer(91)*.

ÍNDICE

Capítulo I

I - INTRODUÇÃO

"Tudo deveria ser feito do modo mais simples possível,

mas não mais simples do que isso."

Albert Einstein

I.1 - Motivação

Os ***Sistemas Elétricos de Potência*** são sistemas dinâmicos altamente complexos e possuem inúmeros problemas que requerem o conhecimento de um especialista, desde o planejamento até a própria operação de cada usina e subestação. Porém, com a aposentadoria de grande parte dos mais experientes destes especialistas, teme-se perder o conhecimento por eles adquirido sem que este seja repassado aos mais jovens.

Algumas tentativas de preservar este conhecimento foram feitas utilizando-se técnicas de Inteligência Artificial, principalmente através de Sistemas Especialistas, cujas maiores dificuldades encontram-se em adquirir o conhecimento dos especialistas e utilizá-lo em tempo real.

Existe uma perspectiva de que estes problemas possam ser resolvidos com a utilização de Redes Neurais, que têm como característica justamente o aprendizado através de exemplos e a utilização em tempo real.

O objetivo desta tese é estudar a aplicação de Redes Neurais em Sistemas Elétricos de Potência.

I.2 - Considerações Iniciais

A ***Inteligência Artificial*** é o estudo de como fazer os computadores realizarem tarefas em que, no momento, as pessoas são melhores *(Rich,88)*. Estas tarefas incluem a visão, a fala, a compreensão da linguagem natural e muitos outros problemas em domínios especializados, como jogos, prova de teoremas e diagnose.

Atualmente, as pesquisas de Inteligência Artificial dividem-se entre o ***Paradigma Simbolista*** (ou Cognitivista), que considera que a mente é, essencialmente, um sistema de manipulação de símbolos, e o ***Paradigma Conexionista***, que procura entender e emular as propriedades decorrentes do alto grau de paralelismo e conectividade existentes no cérebro humano.

Pelo Paradigma Cognitivista, os símbolos são as raízes da inteligência e no cerne desta linha de pesquisa situa-se o que *Newell (76)* denomina ***Hipótese do Sistema de Símbolos Físicos*** ("um sistema de símbolos físicos possui os meios necessários e suficientes para a ação inteligente geral").

Pelo Paradigma Conexionista, o comportamento inteligente "emerge" da coletividade e a inteligência encontra-se nas numerosas conexões entre elementos processadores independentes. Esta linha de pesquisa fundamenta-se na neurobiologia e no processamento paralelo e distribuído de informações. As ***Redes Neurais*** são modelos baseados no Paradigma Conexionista e compostos por elementos processadores simples amplamente conectados entre si.

Os problemas de Inteligência Artificial abrangem um espectro muito amplo, e um dos poucos resultados concretos obtidos é que ***Inteligência*** requer ***Conhecimento***. Para resolver problemas de Inteligência Artificial é preciso representar este conhecimento com algum formalismo, de modo a sermos capazes de manipulá-lo. Assim, é possível definir suas técnicas como métodos de manipular o conhecimento com eficácia e eficiência.

Representar o conhecimento é a base das técnicas de Inteligência Artificial e, apesar disso, a representação de conhecimento em Redes Neurais tem sido, até o momento, pouco explorada. Dentro da comunidade conexionista tem existido uma

longa, e não resolvida, discussão entre os favoráveis a uma representação local do conhecimento e os favoráveis a uma representação distribuída. O ***Paradigma Simbólico-Conexionista*** é uma linha de pesquisa que tenta combinar o Paradigma Simbolista com o Paradigma Conexionista. A construção de sistemas conexionistas para processamento simbólico tem despertado o interesse da comunidade internacional de Inteligência Artificial. Um levantamento bibliográfico recentemente compilado por *Sun (94)* apresenta cerca de 250 publicações (evitando artigos redundantes ou repetitivos) sobre o assunto.

Embora o tratamento simbólico por sistemas conexionistas ainda esteja em fase de gestação, acreditamos ser este o caminho para uma melhor compreensão do cérebro humano e para a aplicação prática de sistemas conexionistas.

I.3 - Contribuições

a) Identificação dos principais problemas das aplicações existentes de Inteligência Artificial em Sistemas Elétricos de Potência:

A partir da bibliografia disponível foi realizado um levantamento das aplicações existentes de Inteligência Artificial nos Sistemas Elétricos de Potência, com ênfase em Redes Neurais, no qual pôde-se perceber a dificuldade na aquisição do conhecimento dos especialistas e na manutenção dos sistemas implantados reportada pelos órgãos internacionais qualificados.

A análise deste levantamento mostra que existe a necessidade preemente de um sistema que permita adquirir o conhecimento do especialista de um modo mais natural e que seja mais adaptativo a possíveis alterações no problema em questão.

b) Caracterização de diversos problemas dos Sistemas Elétricos de Potência como problemas de Classificação:

Os problemas dos Sistemas Elétricos de Potência também abrangem um espectro muito amplo. No entanto, nesta tese mostramos que estes podem ser divididos em problemas de ***Otimização*** e/ou de ***Classificação***.

A partir da análise de alguns exemplos de aplicações dos Sistema Elétricos de Potência, foi possível concluir que grande parte destas podem ser consideradas como problemas de Classificação e que, mesmo as demais aplicações (as que não podem ser enquadradas), utilizam alguma forma de Classificação para auxiliar em sua resolução.

A Classificação é uma poderosa estratégia de organização do conhecimento que pode ser utilizada em diversas aplicações, tais como monitoração, interpretação, predição, diagnose, depuração e outros. Ou seja, todas estas aplicações envolvem alguma forma de Classificação.

c) Levantamento dos requisitos necessários para um sistema de Aquisição do Conhecimento de especialistas em Sistemas Elétricos de Potência:

A análise das aplicações existentes em Sistemas Elétricos de Potência e a caracterização dos problemas como Classificação, acrescidas de entrevistas com especialistas do setor, permitiu levantar os requisitos necessários para um sistema que permita adquirir, armazenar e manter o conhecimento de especialistas.

d) Resumo de Redes Neurais:

Existem diversos modelos de redes, cada qual com suas características singulares e mais adequados à resolução de diferentes classes de problemas. Foi preciso estudar o paradigma conexionista e averiguar quais as características mais importantes e os modelos mais indicados para esta tarefa. Como conseqüência, foram reunidos vários conceitos teóricos dispersos na literatura e para cada conceito criaram-se explanações simples e sucintas. Para possíveis referências durante o restante do

texto, unificaram-se as terminologias utilizadas. Esperamos que o resultado obtido possa ser utilizado como referência para futuros trabalhos na área.

e) Estudo da Entrada e Saída de Dados em Redes Neurais

Para utilizar uma Rede Neural na resolução de qualquer problema, é preciso treiná-la a partir de um conjunto de exemplos, compostos de padrões de entrada e saída, que o representem. Apesar da limitação imposta pela necessidade de padrões completos não foi encontrada nenhuma referência nem estudo sobre o assunto. Isto motivou um estudo mais detalhado da entrada e saída de dados em redes neurais.

Este estudo, apesar de não ser exaustivo, mostrou que o formato dos dados limita os resultados que podem ser obtidos. Esta é uma das principais contribuições desta tese. Infelizmente, o estudo também mostrou que a representação da ausência de evidência nos dados não é suficiente para permitir a utilização de exemplos incompletos. Acreditamos ser necessário um mecanismo de aprendizado capaz de lidar com esta representação.

f) Representação de Conhecimento relativo à problemas de Classificação em Redes Neurais

Para representar o conhecimento necessário à resolução de problemas de classificação é preciso dividir este conhecimento em conceitos e mapeamentos entre conceitos. Os conceitos devem possuir uma representação externa localizada, de tal modo que o especialista não tenha dificultades em representá-los. Já o mapeamento entre estes deve ser induzido a partir dos exemplos e armazenado em uma representação interna distribuída. Este princípio é utilizado em redes Back-Propagation.

g) Levantamento das Características Necessárias para a utilização de Redes Neurais em problemas de Classificação

Apenas um especialista é capaz de determinar a associação entre os conceitos de um problema de classificação. Deste modo, o aprendizado deve ser supervisionado. Em um problema de classificação, os conceitos de entrada são diferentes dos conceitos de saídas (ou seja, os dados do problema são diferentes das respostas desejadas), indicando a utilização de redes hetero-associativas. Porém, para descrever o problema, o especialista utiliza conceitos intermediários, que podem ser inferidos a partir das entradas e devem ser utilizados na obtenção das saídas. Note que, inclusive, um conceito intermediário pode depender de outro conceito intermediário. Para se evitar topologias dependentes do problema é necessário que o modelo possua uma topologia recursiva. Finalmente, somente parte dos conceitos de entrada são utilizados nos exemplos para obtenção de mais alguns conceitos de saída, de modo que a rede neural deve poder utilizar exemplos incompletos.

h) Uma Arquitetura de Redes Neurais para problemas de Classificação

As Redes Neurais para Classificação existentes se preocupam com um caso particular denominado ***Categorização***. Para resolver o caso mais geral normalmente utilizam-se ***Redes Neurais Hierárquicas***, que combinam hierarquicamente diversas sub-redes para categorização. Isto significa que é necessária uma arquitetura diferente para cada problema. Apresentamos uma arquitetura única, parcialmente recursiva, capaz de resolver problemas de classificação.

i) Seleção de um Algoritmo para a Arquitetura Proposta

Uma variação da generalização do algoritmo ***Back-Propagation*** para redes recursivas, denominado ***Back-Propagation Through Time***, foi selecionado para ser utilizado na arquitetura proposta.

j) Implementação e testes da arquitetura:

A arquitetura proposta foi testada com um exemplo de Classificação dos Sistemas Elétricos de Potência (Detecção de Falhas em Transformadores) e com um exemplo de um artigo publicado por *Telfer (91)*. Ambos os testes foram realizados utilizando o simulador SNNS v3.1 da Universidade de Stuttgart.

I.4 - Organização do Texto

Esta tese está dividida em oito capítulos, sendo este primeiro introdutório. O capítulo seguinte apresenta o problema e indica o caminho para sua resolução. Os três subseqüentes fornecem um mínimo da teoria necessária aos leigos, e uma terminologia unificada aos pesquisadores, para que ambos possam entender os demais capítulos, que contêm as principais contribuições desta tese nos temas abordados. Por último, são apresentadas as conclusões obtidas e sugestões para a continuação do trabalho aqui iniciado.

O **Capítulo II** mostra a necessidade de se adquirir o conhecimento acerca dos principais problemas existentes nos Sistemas Elétricos de Potência a partir dos especialistas do setor. Um simples levantamento da bibliografia existente sobre as aplicações de Inteligência Artificial, principalmente em Redes Neurais, evidencia esta necessidade. O detalhamento de dois exemplos (Manutenção Preditiva de Hidrogeradores e Detecção de Falhas em Transformadores) permite mostrar que os problemas dos Sistemas Elétricos de Potência são problemas de Classificação e fazer um levantamento dos requisitos necessários de um sistema capaz de adquirir conhecimento de especialistas.

O **Capítulo III** contém um estudo do Paradigma Conexionista e das Redes Neurais. Não temos a pretensão de abordar exaustivamente o assunto, uma vez que este objetivo pode ser alcançado com a utilização de excelentes livros *(Rumelhart,86) (Wasserman,89) (Dayhoff,90) (Hertz,90) (Levine,91)*.

O **Capítulo IV** apresenta os modelos existentes de Redes Neurais aplicados à problemas de Classificação e mostra que, na realidade, procuram resolver um caso particular, denominado Categorização.

O **Capítulo V** procura na limitada compreensão que existe acerca da Representação de Conhecimento em Redes Neurais, a resolução deste problema. São mostrados dois modelos desenvolvidos por pesquisadores brasileiros, um que utiliza Representação Localizada de Conhecimento e outro que utiliza Representação Distribuída de Conhecimento.

É no **Capítulo VI** que se inicia a apresentação dos resultados originais da tese. Nele encontra-se um estudo realizado sobre a representação dos conceitos como dados nos padrões de entrada e saída de uma rede neural.

O **Capítulo VII** mostra quais são as características mais adequadas a uma Rede Neural para que possa ser utilizada em problemas de Classificação, e apresenta uma arquitetura que reúne estas características. Depois, explica como utilizar o algoritmo "Back-Propagation Through Time" na arquitetura proposta. Finalmente, realiza testes com um exemplo fornecido por *Telfer (91)* e com a Detecção de Falhas em Transformadores.

O **Capítulo VIII** apresenta as conclusões obtidas a partir dos capítulos anteriores, sinaliza quais são as perspectivas atuais e oferece sugestões para a continuação deste trabalho.

Esperamos que todas as informações necessárias ao entendimento do texto estejam auto-contidas e que as dúvidas porventura existentes possam ser sanadas através das **Referências Bibliográficas** e de um **Glossário** dos termos utilizados ao longo do texto.

Finalmente, o **Anexo A** reproduz uma discussão entre conexionistas na InterNet acerca da utilização de exemplos incompletos, o **Anexo B** contém testes de alguns dos mecanismos para exemplos incompletos aplicados ao problema do "Disjuntor Isolado por Seccionadoras", o **Anexo C** mostra os testes da arquitetura

proposta com a "Detecção de Falhas em Transformadores" *(Capítulo II)*, e o **Anexo D** também contém testes, porém, com a classificação fornecida por *Telfer (91)*.

Capítulo II

II - INTELIGÊNCIA ARTIFICIAL EM SISTEMAS ELÉTRICOS DE POTÊNCIA

"Um perito é alguém que não precisa refletir. Ele sabe."

Frank Lloyd Wright

II.1 - Estado Atual

O Sistema Elétrico de Potência brasileiro enfrenta uma grave crise devido ao momento econômico, causando evasão da mão-de-obra especializada e escassez de investimentos. Como conseqüência, os esforços do setor têm se concentrado na automação da rede elétrica existente.

Embora atualmente o setor elétrico possua sistemas digitalizados de supervisão e controle (SDSC), estes somente substituem ou complementam os sistemas convencionais. Todo o conhecimento sobre o planejamento e a operação do sistema continua sendo privilégio de alguns ***especialistas***, isto é, de pessoas capazes de tomar decisões a partir da lógica, heurística, experiência e, até mesmo, intuição.

O número de situações de emergência enfrentadas por operadores de usinas e subestações tem diminuído significativamente enquanto que, com a expansão da redes de energia elétrica, mais operações complexas têm-se tornado necessárias. Assim, a quantidade de potência sob responsabilidade de cada operador tem aumentado a cada ano (*Figura II.1*).

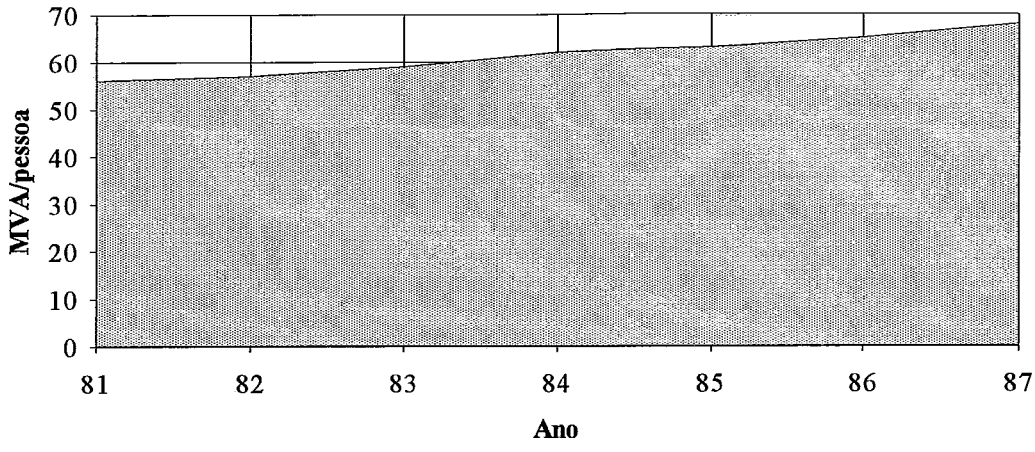


Figura II.1 - Capacidade de operação por pessoa ao longo dos anos (Kawada,89)

Existe, portanto, uma necessidade de fornecer subsídios para que estes especialistas consigam lidar com situações que nunca viveram. Este problema torna-se ainda mais crítico ao considerar-se que o número de operadores experientes tem diminuído cada vez mais ao longo dos anos (*Figura II.2*). Com a aposentadoria dos operadores mais experientes teme-se perder o conhecimento por eles armazenado.

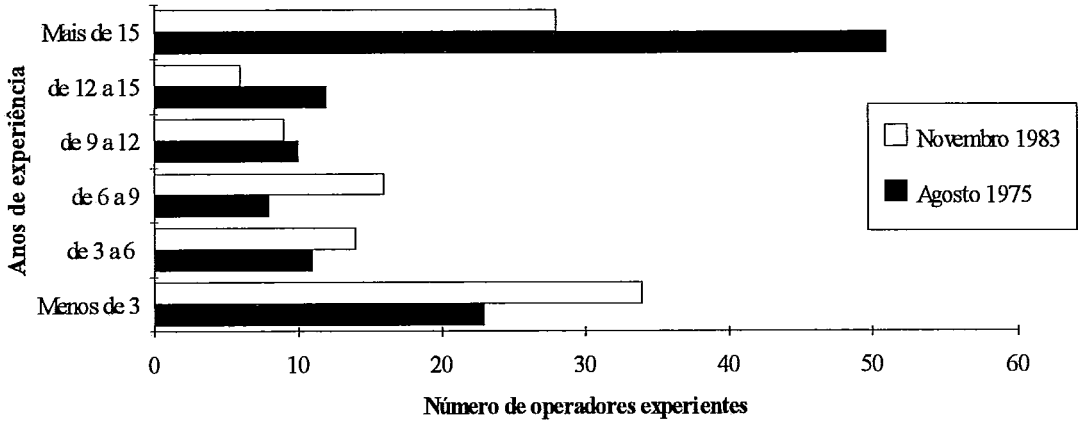


Figura II.2 - Experiência Profissional (Wollemberg,87)

Inúmeros problemas nos Sistemas Elétricos de Potência requerem o conhecimento de um especialista, capaz de resolvê-los bem melhor do que qualquer

máquina. Esta característica torna atraente a utilização de técnicas de Inteligência Artificial em sua resolução *(Capítulo I)*.

II.2 - Inteligência Artificial

A Inteligência Artificial tem começado a desempenhar um importante papel nas pesquisas realizadas em Sistemas Elétricos de Potência. Em 1985, a Conferénce Internationale des Grands Résaux Életriques à Hauté Tension (CIGRÉ) criou a força-tarefa 38.02/07, reconhecendo a forte necessidade de pesquisas sobre a aplicação das técnicas de Inteligência Artificial na área.

Em 1987, *Wollenberg (87)* publicou uma pesquisa realizada por membros do Institute of Electrical and Electronic Engineering (IEEE) que apresentava os primeiros resultados sobre a utilização de Sistemas Especialistas no setor e reconhecia que a utilidade das técnicas de Inteligência Artificial depende, fortemente, da qualidade do conhecimento embutido no sistema.

Neste mesmo ano, *Schulte (87)* reuniu pequenos artigos de pesquisadores da área e concluiu que o ponto central do uso de Sistemas Especialistas é a consciência de que estes representam o conhecimento daqueles que criaram o sistema, e nada além disto.

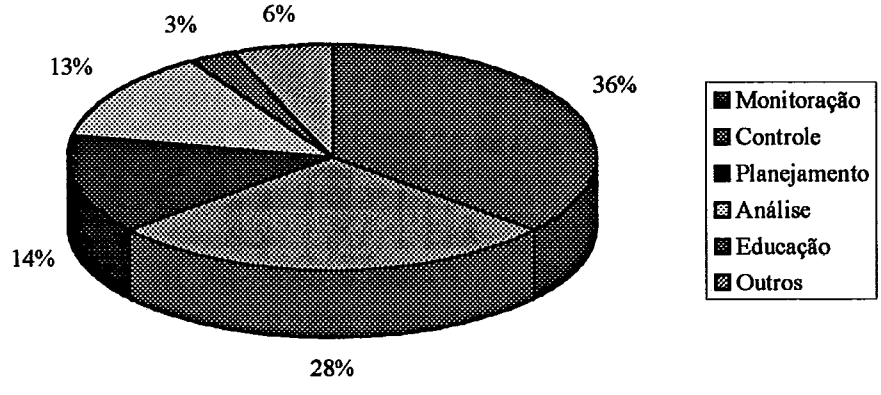


Figura II.3 - Aplicações de Sistemas Especialistas (Tamura,89)

Em 1989, *Tamura (89)* publicou o resultado de uma pesquisa realizada pela CIGRÉ sobre as aplicações (*Figura II.3*) e o estágio de desenvolvimento dos Sistemas Especialistas na área. A maior parte dos sistemas até então existentes encontrava-se ainda na fase de idealização ou estudo, havendo muito poucos logrado implementações práticas (*Figura II.4*).

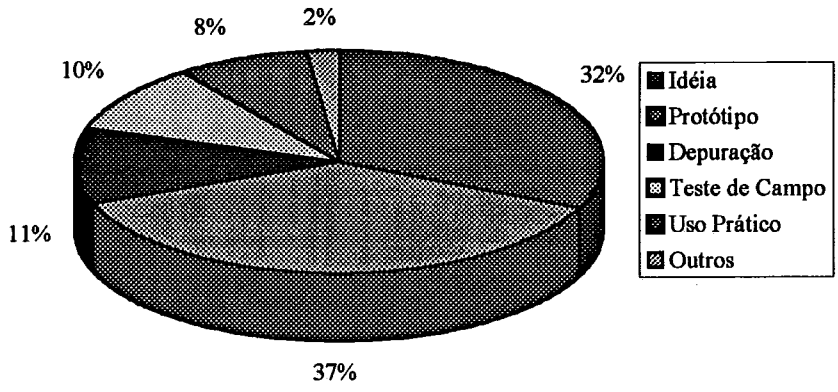


Figura II.4 - Estágio de Desenvolvimento (Tamura,89)

Neste levantamento ficou demonstrado que existia uma grande variedade de formas para o processamento da informação, mas quase nenhuma para a aquisição do conhecimento necessário. Além disso, foi levantada a necessidade de um sistema que conseguisse lidar com situações em que houvesse grande massa de dados, maior complexidade da rede elétrica, natureza combinatorial da solução, informações incompletas e conflito entre dados.

Em 1992, pesquisa semelhante realizada pela força-tarefa 38-06-03 (criada em 1990) do CIGRÉ, com aplicações em uso prático e publicada por *Liu (92)*, destacou a importância e a necessidade de técnicas mais eficientes para aquisição e verificação do conhecimento.

Atualmente, técnicas como Sistemas Especialistas e Redes Neurais têm sido utilizadas com sucesso em diversas aplicações. Além disso, aplicações baseadas nestas técnicas têm sido instaladas em Centros de Supervisão e Controle, embora a maioria ainda esteja em fase experimental.

Para que se possa utilizar efetivamente a Inteligência Artificial na resolução de problemas do setor elétrico, é necessário que os projetos a serem desenvolvidos integrem-se aos sistemas digitalizados de supervisão e controle já existentes e em funcionamento.

II.3 - Redes Neurais

Como relatado anteriormente, a maior parte das aplicações existentes de Inteligência Artificial nos Sistemas Elétricos de Potência utilizavam Sistemas Especialistas. Porém, certas classes de problemas têm tido soluções insatisfatórias devido ao tempo de processamento (e.g. "planejamento da operação" e "análise de contingência"), ou porque envolvem complexas classificações (e.g. "predição de carga", "processamento de sinais" e "processamento de alarmes"), ou mesmo porque não se possui um modelo matemático adequado (e.g. "identificação de sistemas" e "controle"). Para estes e outros problemas, as Redes Neurais apresentam-se como uma solução promissora a ser explorada.

Sistemas Elétricos de Potência são sistemas complexos, envolvendo muitos componentes elétricos, cuja operação deve ser planejada, analisada, monitorada e controlada. Seu comportamento é altamente não-linear e as variáveis envolvidas estão apenas parcialmente disponíveis, indicando a utilização de Redes Neurais.

As Redes Neurais despertaram grande interesse no setor elétrico, tendo sido abordada em centenas de publicações na área, principalmente em aplicações como "Predição de Carga", "Segurança", "Controle" e "Diagnóstico de Falhas".

Wildberger (92) reportou que as pesquisas do Eletric Power Research Institute (EPRI) em Redes Neurais têm objetivado descobrir novas aplicações nos Sistemas Elétricos de Potência, além de, estabelecer teorias e procedimentos para o seu desenvolvimento e utilização.

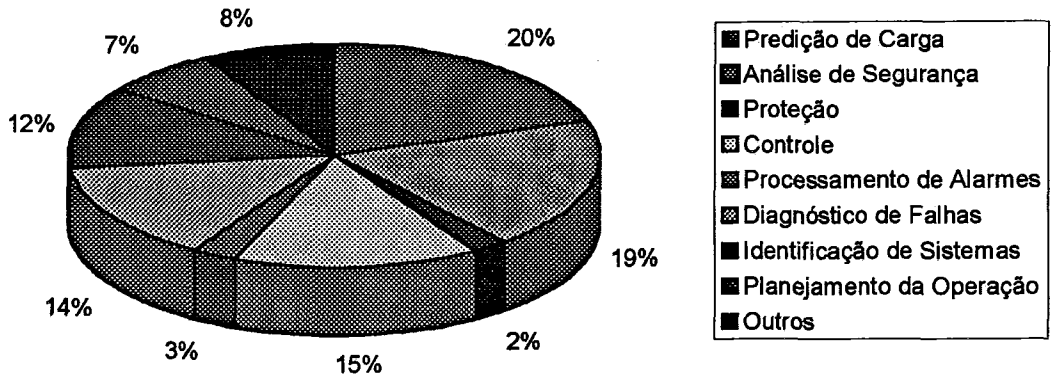


Figura II.5- Aplicações de Redes Neurais em Sistemas Elétricos de Potência

Em 1993, a força-tarefa 38-06-06 do CIGRÉ publicou os resultados de uma pesquisa realizada por *Niebur (93)* na literatura existente, abrangendo mais de 200 artigos, que permitiu levantar a distribuição atual das aplicações *(Figura II.5)* de Redes Reurais e de seus modelos *(Figura II.6)* em Sistemas Elétricos de Potências:

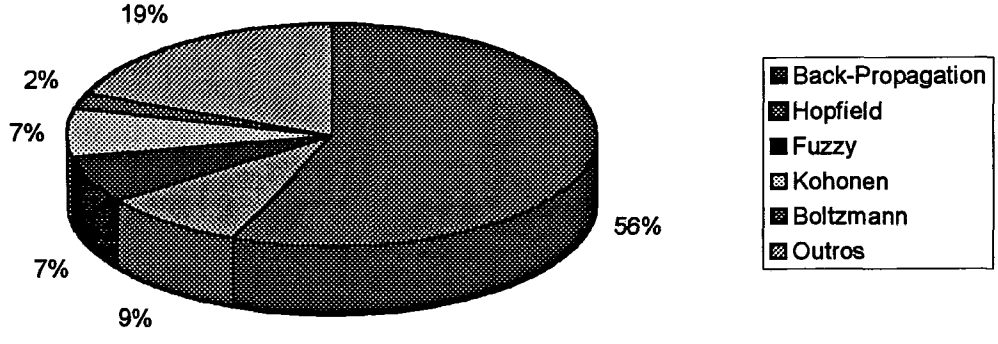


Figura II.6 - Modelos de Redes Neurais em Sistemas Elétricos de Potência

Sem dúvida alguma, o modelo de Redes Neurais mais utilizado é o Back-Propagation, porém, já se pode notar um crescente interesse em modelos recursivos (Hopfield) e com camadas classificatórias (Kohonen).

A maioria dos artigos analisados apenas se referiam à resultados em implementações de pequeno porte, carecendo uma maior análise de casos reais. Mesmo assim, as comparações entre a utilização de Redes Neurais e as técnicas convencionais sempre reportaram as vantagens de sua utilização.

A maioria dos artigos analisados apenas se referiam à resultados em implementações de pequeno porte, carecendo uma maior análise de casos reais. Mesmo assim, as comparações entre a utilização de Redes Neurais e as técnicas convencionais sempre reportaram as vantagens de sua utilização.

A utilização de Redes Neurais tem que ser melhor investigada nas aplicações de larga escala e naquelas com elevado índice de modificações, principalmente, no que se refere ao retreinamento em caso de atualizações e mudanças no Sistema Elétrico de Potência.

Finalmente, sistemas híbridos, combinando Redes Neurais a outras técnicas como Sistemas Especialistas e Lógica Fuzzy, também carecem de melhores estudos.

II.4 - Possíveis Aplicações

Alguns problemas dos Sistemas Elétricos de Potência são particularmente indicados para aplicações de Inteligência Artificial:

- **Planejamento** (Planning)

A expansão da geração de energia pode ser definida como sendo o problema de determinar a quantidade de energia elétrica a partir de um conjunto de restrições. *Sasaki (91)* usa um modelo modificado de Hopfield onde há um grupo de elementos processadores para as variáveis e outro para as restrições.

- **Predição de Carga** (Load Forecasting)

Como a energia elétrica não pode ser estocada eficientemente, a acurácia na predição de carga pode levar a uma redução significativa de custos. Porém, é extremamente difícil modelar o relacionamento existente entre as cargas e os fatores que podem influenciá-las, como condições de tempo e feriados. Tais conhecimentos ainda não estão formalizados. *Peng (92)* utiliza uma rede neural Adaline modificada para prever a carga futura a partir da análise de componentes periódicos da carga dos últimos meses.

- **Processamento de Alarmes**

Os modernos Sistemas Elétricos de Potência são operados por pessoas altamente especializadas através de Sistemas Digitais de Supervisão e Controle (SDSC) que são feitos, principalmente, para serem utilizados durante a operação normal, de modo a assegurar um funcionamento seguro. Porém, no caso de um evento imprevisto ou de falha em algum componente, estes sistemas são sobrecarregados com uma enorme massa de informações, tornando-se de pouca utilidade *(Wollenberg,87)*. Ou seja, quando ocorre uma falha mais séria, os operadores podem ser "inundados" com mensagens de alarmes, de tal modo que, devido à redundância entre estes alarmes ou à presença de muitas informações referentes ao mesmo evento, eles têm dificuldade em interpretar corretamente a situação. Costuma-se dizer que um especialista precisa de informação ao invés de dados. É possível utilizar técnicas de Inteligência Artificial para interceptar mensagens de alarme e apresentar um diagnóstico conciso, ou pelo menos um conjunto reduzido de alarmes. *Chan (90)* procurou usar Redes Neurais para criar um processador inteligente de alarmes, porém, este foi apenas parcialmente testado *(Chan,91)*.

- **Diagnóstico de Falhas**

Quando ocorrem eventos inesperados, os operadores têm que entender (diagnosticar) a situação e determinar ações para retornar o sistema ao estado normal (ver ítem abaixo) no mais curto intervalo de tempo possível, submetendo-os a um extremo "stress" mental. Embora os operadores sejam capazes de determinar com acurácia as causas das falhas ocorridas em sistemas de potência, não conseguem facilmente explicar à outras pessoas como essas falhas foram determinadas. O relacionamento causa-efeito é altamente não-linear e não existem modelos matemáticos capazes de realizar a análise necessária. *Chow (92)* utilizou uma rede Back-propagation para identificar falhas causadas por animais em redes de distribuição. *Kraft (91)* utilizou um sistema híbrido, para monitoração de plantas de combustível fóssil, que possuía um modelo recursivo de Redes Neurais.

- **Restauração do Sistema**

Após o diagnóstico correto da situação em que se encontra o sistema é necessário tomar ações corretivas caso este não se encontre em estado normal. Este problema é particularmente interessante no caso de "black-outs", onde existe a necessidade de uma restauração ao mesmo tempo rápida e segura. Ele torna-se ainda mais grave em uma rede de transmissão de energia, onde existem mais restrições, tais como escala de operação entre unidades e tempo mínimo de reentrada. O objetivo, neste caso, é determinar, a partir do estado do sistema, um conjunto finito de ações corretivas a serem tomadas. A natureza não-linear do problema, aliada à heurísticas, torna necessária a utilização de técnicas não-convencionais, particularmente na presença de grande quantidade de dados. *Novosel (91)* utilizou uma rede Back-propagation para reescalonar a geração de energia (alterando a topologia da rede - ver abaixo) a partir das correntes entre os barramentos.

- **Topologia de Rede** (Network Reconfiguration)

Quando ocorrem falhas ou manutenções periódicas torna-se necessário reconfigurar a rede de distribuição, de modo a minimizar a perda de potência ou o número de usuários não atendidos. As estatísticas mostram que cerca de quarenta por cento das atividades realizadas em um centro de operações estão relacionadas com disjuntores e seccionadoras. Os atuais sistemas não verificam se as seqüências de manobra estão corretas, nem tampouco orientam os operadores nas possíveis opções. *Kim (93)* utiliza um modelo Back-propagation na minimização da perda de potência, criando dois conjuntos de pequenas redes: o primeiro para estimar a devida carga de cada zona; e o segundo para determinar a topologia a partir da potência disponível.

- **Controle**

Vários equipamentos e sub-sistemas controlados pelos Sistemas Elétricos de Potência possuem características não-lineares, que indicam a utilização de Redes Neurais como controladores destas plantas ao invés dos tradicionais. *Thomas (91)* estudou as implicações dinâmicas de utilizar Redes Neurais como controladores. *Wu (91)* utilizou duas redes Back-propagation para controlar turbo-geradores *(Wu,92)*.

Saitoh (91) também utilizou uma Back-Propagation para determinar as funções de pertinência de regras "fuzzy" para o controle de uma unidade geradora. Nas instalações onde há mais do que uma máquina síncrona (e.g. geradores), existe a necessidade de se controlar coordenadamente suas saídas reativas. Sistemas deste tipo são denominados JVC ("Joint VAR Control"). *Neily (91)* utilizou um modelo Back-propagation para controlar 4 (quatro) máquinas síncronas.

- **Manutenção Preditiva**

O uso de técnicas de Inteligência Artificial na análise do funcionamento de equipamentos permite aliar heurísticas e conhecimentos de especialistas à análise numérica existente. A detecção antecipada de defeitos incipientes em equipamentos do setor pode economizar milhares de dólares. *Chow (91)* aplicou um modelo Back-propagation na detecção de defeitos incipientes em motores de indução, e usou uma teoria de aprendizado para determinar o número de exemplos de treinamento necessário. *Tomsovic (93)* utilizou lógica Fuzzy para integrar diferentes métodos de diagnósticos de transformadores.

- **Proteção**

As técnicas existentes para a proteção de linhas de transmissão utilizam principalmente relés analógicos estáticos (e.g. relé de sobre-corrente). Porém, com o advento dos relés digitais, outras proteções puderam ser implementadas. Em uma visão moderna, os relés podem ser considerados como equipamentos classificadores, nos quais técnicas de Inteligência Artificial podem, e devem, ser aplicadas. A maioria dos artigos existentes sobre Inteligência Artificial em proteção se referem à aplicação de Redes Neurais. *Feser (91)* utilizou um modelo Back-propagation no pré-processamento e na recuperação de sinais distorcidos de relés de alta-impedância. *Khaparde (91)* utilizou uma rede Adaline em relés de distância. *Lubkeman (91)* utilizou Redes Neurais não-supervisionadas para classificar as perturbações de transientes nas linhas de distribuição, pois os atuais esquemas de relés de proteção são capazes de identificar apenas pertubações maiores, como falhas de baixa impedância.

- **Operação** (Unit Commitment)

A operação dos Sistemas Elétricos de Potência requer a observação de uma série de restrições e a otimização do fornecimento de energia. *Fukuyama (91)* utiliza uma rede de Hopfield para resolver este problema de otimização, primeiro, considerando cada unidade de geração individualmente e, depois, conjuntamente com as demais. *Ronne-Hansen (91)* utiliza um modelo Back-propagation com a mesma finalidade.

- **Treinamento**

O aumento da complexidade operacional nos Sistemas Elétricos de Potência parece exceder a capacidade dos operadores em situações de emergência. Em um ambiente controlado é possível simular situações (cenários) onde aprendizes podem observar e discutir as conseqüências de ações corretivas no sistema, capacitando-os a entender melhor o problema e sua correspondente solução.

- **Qualidade da Energia**

Os consumidores esperam receber uma tensão AC constante e sem pertubações (e.g. surtos, quedas e impulsos). *Daniels (91)* utilizou um modelo Back-propagation de Rede Neural para classificar e propor soluções para estas pertubações. Um dos principais problemas da qualidade de energia refere-se à estabilidade de tensão. A instabilidade de tensão ocorre quando o sistema não é capaz de suprir a carga reativa, sendo caracterizada por um declínio progressivo de voltagem em regiões de consumo onde o sistema está próximo à sua capacidade máxima de transferência. *Vadari (91)* propõe um sistema híbrido para resolver este problema, que ocorre, principalmente, devido à diferença entre a carga prevista e a real. Além disso, tradicionalmente assumem-se as cargas como lineares, porém, muitos equipamentos elétricos (computadores, impressoras, plotters) utilizam fontes de potência não-lineares, causando altas correntes e perdas de calor através de distorções de harmônicos. *Jarayama (91)* propôs a utilização de um modelo Back-propagation para a resolução deste problema.

- **Outros**

Além dos problemas supra-citados, as técnicas de Inteligência Artificial têm sido utilizadas nas mais diversas aplicações. *Ramani (91)* utilizou um modelo Back-propagation de Rede Neural para identificação de peixes em um reservatório através de um sonar.

De um modo geral, podemos dividir as possíveis aplicações de Inteligência Artificial em três grupos: as que buscam otimizar uma ou mais variáveis do sistema (tais como "Planejamento da Operação"), as que relacionam eventos ou estados do sistema à conclusões ou ações sobre o sistema (tais como "Processamento de Alarmes" e "Diagnóstico de Falhas") e as que pertencem a ambos os grupos anteriores.

II.5 - Exemplos de Problemas

II.5.1 - Manutenção Preditiva de Hidrogeradores

Os métodos tradicionais para monitoração e análise de um equipamento são as análises Fisico-Química, Térmica, Acústica, Elétrica e de Vibrações. Cada um destes métodos aplica-se melhor à análise de um determinado tipo de equipamento. No setor elétrico utiliza-se muito a análise de vibrações em máquinas rotativas (e.g. hidrogeradores) e a análise térmica em tranformadores.

O som e as vibrações emitidos por um equipamento (ou mesmo um sistema) podem ser utilizados como sinal de advertência quanto ao desvio de seu funcionamento, podendo ser utilizado para evitar danos irreversíveis em equipamentos. Um exemplo simples desta situação são os ruídos emitidos por um automóvel que são traduzidos, em função da experiência do motorista, em alertas de um uso inadequado ou até mesmo de um defeito. Estes elementos são importantes na identificação dos estados operativos de equipamentos ou sistemas.

As distorções no funcionamento percebidas através do som e das vibrações podem evitar graves acidentes nos equipamentos dos Sistemas Elétricos de Potência e representar uma economia significativa (como em hidrogeradores, que custam milhões de dólares).

A monitoração de som e vibrações é feita através de sensores de vibração (transdutores de vibração, de velocidade, de deslocamento e sonoros). Na análise de vibrações (no domínio da freqüência), os sinais medidos são convertidos em espectros de freqüência e comparados com padrões de valores, que podem ser absolutos ou relativos. Valores absolutos correspondem aos padrões existentes em normas técnicas, especificações de fabricantes ou níveis aceitáveis de operação, enquanto os relativos referem-se à valores obtidos com o equipamento em "bom" funcionamento. Note que este tipo de análise é individualizada para cada equipamento.

Os valores medidos com o equipamento em "bom" funcionamento são convertidos para o domínio da freqüência em um "espectro base". O funcionamento real do equipamento gera um "espectro de freqüência" que é comparado ao espectro-base. A diferença entre os dois espectros permite detectar variações na amplitude de determinadas freqüências quando ocorre algum mal-funcionamento. Estas freqüências são denominadas ***freqüências notáveis***. Ou seja, para uma certa configuração de freqüências notáveis deseja-se saber quais os defeitos que podem estar a elas relacionados. Este "espectro de freqüências notáveis" é variável para cada equipamento.

Do Coutto (91) fez um levantamento (*Tabela II.1*) dos defeitos identificáveis a partir de variações nas faixas de freqüência de vibrações (análise no domínio da freqüência) em hidrogeradores publicados por *Ripper (89)*.

FREQÜENCIAS X DEFEITOS	<0.4 RPM	0.4 a 0.5 RPM	0.5 a 1 RPM	1 RPM	2 RPM	múltiplos superiores	múltiplos inferiores	F ímpar	F altas	60 Hz	120 Hz	no.pás X RPM	>2K Hz	estocástica	F rotor	F estator
Desalinhamento				X	X	X										
Afrouxamento Partes Mecân.				X	X	X		X								
Folga Excessiva dos Mancais	X	X	X													
Caixa Mancal Deformada	O	O	O	X	X	X										
Arrast. Selos Mancais	X	X	X	X	X	X	X	X	X							
Arrast. Axial do Rotor	O	O	O	X	X	X	X	X	X							
Arrast. Radial do Rotor		X														
Excent. Mancais Deslizam.				X	X											
Mancal Guia Danificado	O	O	O	O	X				X							
Mancal Escora Danificado	O	O	O	O	O				X							
Vibr. Excitada pelos Mancais	O	O	O													
Rigidez Desigual Mancal Guia					X	X										
Forças Aero / Hidráulicas				X		X										
Rotação Eixo por Atrito	X	X	X													
Instab. Cunha Óleo nos Mancais		X														
Rotação Eixo por Ressonância		X														
Atrito Seco Mancais									X							
Excentricidade Rotor				X	X											
Desalinhamento Estator / Rotor				X												
Excentricidade Estator				X	X											
Barramento Defeituoso				X												
Eixo Central Fletido				X												
Rotor Eletric. Descentral.				X	X											
Desbalanceamento Massa				X												
Desbalanc. por Quebra Súbita				X												
Desbalanc. por Deterioração				X												
Desbalanceamento Térmico				X												
Desbalanceamento Magnético				X						X	X					
Eixo Fletido / Empenado				X	X											
Eixo Fletido por Atrito				X												
Coordenação entre Pás												X		X		
Cavitação													X	X		
Trancas no Tubo Sucção	X															
Ressonância Partes Rotativas															X	
Ressonância Estator																X
Ressonância Excitação Cavitação													O	O	X	

Tabela II.1 - Freqüências notáveis de um hidrogerador (DoCoutto,91)

Normalmente, na resolução destes problemas são produzidas tabelas a partir de entrevistas com especialistas de cada área. Na tabela exemplificada, mais especificamente, na linha relativa a determinado defeito, encontram-se todos os relacionamentos a ele correspondentes já observados pelos especialistas. É importante ressaltar que uma máquina, ao apresentar um defeito, não necessariamente apresentará em seu espectro de freqüência todas as freqüências a ele relacionadas. Na realidade, a linha indica que no espectro poderão aparecer todas as combinações possíveis das freqüências relacionadas, excetuando-se os relacionamentos simbolizados por um "O" ao invés de por um "X", que indicam que as freqüências envolvidas não poderão aparecer simultaneamente.

Porém, uma análise mais detalhada mostra que estas tabelas apenas resumem o conhecimento para os casos mais comuns, não refletindo na realidade o conhecimento do indivíduo. Em outras palavras, uma única tabela não é suficiente para representar o conhecimento do especialista.

II.5.2 - Detecção de Falhas em Transformadores

Durante a operação de um transformador, o óleo isolante e outros materiais dielétricos sofrem processos de decomposição química que resultam na evolução de gases. No caso de ocorrência de falhas, como superaquecimento, arco ou descargas parciais, uma quantidade suficiente de gases é emitida e pode ser utilizada para alarme e proteção do transformador.

Este método tem sido empregado na resolução de diversos problemas, como a aceitação de novos equipamentos, detecção de falhas incipientes e localização de falhas ocorridas. Existe, inclusive, um critério do IEC ("International Electrotechnical Commission") que utiliza a relação entre a concentração destes gases na detecção de falhas *(Tabela II.2)*.

	Gases		
FAULT TYPE	C_2H_2 / C_2H_4	CH_4 / H_2	C_2H_4 / C_2H_6
Low Parcial	< 0.1	< 0.1	< 1.0
High Parcial	0.1 to 3.0	< 0.1	< 1.0
Low Discharges	0.1 to 3.0 > 3.0	0.1 to 1.0	< 1.0 1.0 to 3.0
High Discharges	0.1 to 3.0	0.1 to 1.0	> 3.0
Very Low Thermal	< 0.1	0.1 to 1.0	1.0 to 3.0
Low Thermal	< 0.1	> 1.0	< 1.0
High Thermal	< 0.1	> 1.0	1.0 to 3.0
Very High Thermal	< 0.1	> 1.0	> 3.0

Tabela II.2 - Critério do IEC

Na tabela acima é possível perceber que o conhecimento por ela expresso embute, pelo menos, uma forma de classificação *(Figura II.6)*:

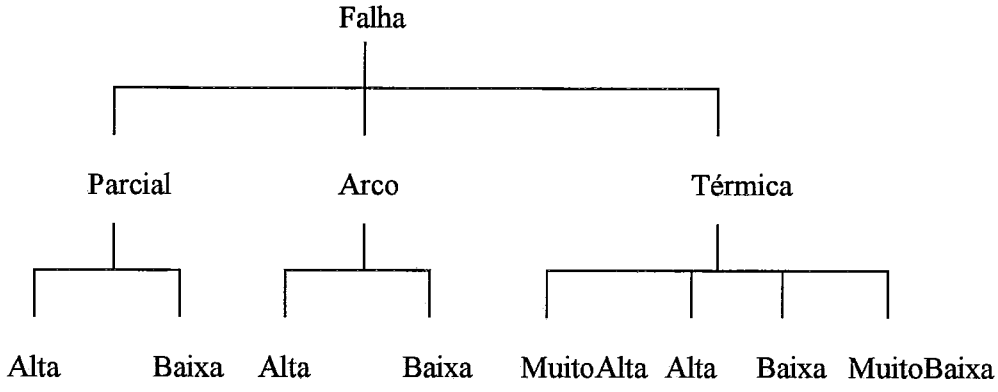


Figura II.6 - Classificação de Falhas em Transformadores (IEC Criteria).

No CEPEL, esta análise é feita a partir da comparação da emissão com um valor padrão, de modo a identificar o meio atingido pela falha: óleo, papel ou água, como descrito por *Santos (79) (Tabela II.3)*.

MEIO	FALHA	GASES-CHAVE
Óleo	Arco	Acetileno
	Parcial	Hidrogênio Metano
	Aquecimento	Etileno
Papel	Parcial	Hidrogênio Monóxido de Carbono
	Aquecimento	Monóxido de Carbono Dióxido de Carbono
Água	Eletrólise	Hidrogênio

Tabela II.3 - Gases-Chave no diagnóstico de falhas (Santos, 79)

As principais falhas a considerar são o superaquecimento, a descarga parcial e o arco elétrico. Estas *(Figura II.7)* podem atingir, principalmente, o óleo e a celulose. Em caso de aquecimento o óleo é decomposto em hidrocarbonetos, aumentando a concentração de etileno em relação ao metano e ao etano. Desta forma, o etileno é o principal produto na caracterização de falhas por superaquecimento. Caso a celulose carbonize, resultará em monóxido e dióxido de carbono. As descargas parciais (corona) produzem hidrogênio acompanhado de menores concentrações de metano e, caso a celulose seja atingida, monóxido de carbono. O arco produz predominantemente acetileno, embora também gere hidrogênio. Finalmente, no caso de eletrólise na água, apenas o hidrogênio é liberado.

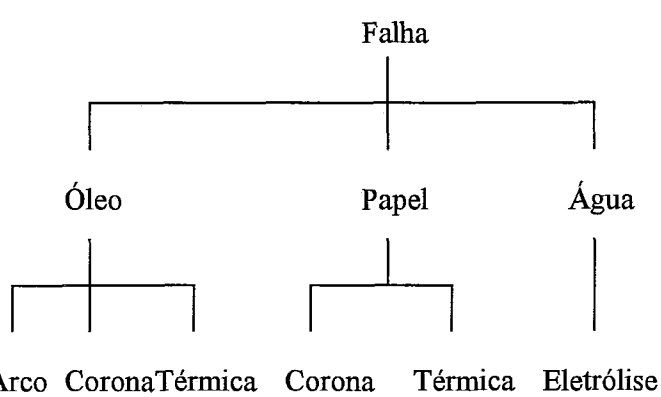


Figura II.7 - Classificação de Falhas em Transformadores (CEPEL).

II.6 - Problemas de Classificação

O detalhamento de alguns exemplos, realizado no ítem anterior, permite concluir que os problemas dos Sistemas Elétricos de Potência podem ser considerados como problemas de Classificação. No primeiro ("Manutenção Preditiva em Hidrogeradores") dos exemplos detalhados pode-se ver que uma simples tabela não é suficiente para representar o conhecimento envolvido. No segundo ("Detecção de Falhas em Transformadores"), pode-se notar que os especialistas se utilizam de uma forma de classificação na representação deste conhecimento. Mesmo os problemas que não possam ser considerados como sendo problemas de classificação, podem utilizar esta estratégia em sua resolução.

Grande parte do aprendizado humano pode ser considerado como um processo gradual de formação de conceitos *(Fisher,89)*. Por este ponto de vista, o ser humano consegue induzir, a partir de uma sucessão de observações e eventos, um conjunto de conceitos e relacionamentos entre estes, que resumem e organizam sua experiência. Este processo, denominado ***classificação***, é uma poderosa estratégia de organização do conhecimento e pode ser utilizada na monitoração, interpretação, predição, diagnose, depuração, controle e muitas outras aplicações.

A ***rede semântica***,desenvolvida por Quillian em 1968, consiste de um grafo no qual os vértices representam objetos, conceitos ou eventos; e os arcos representam relações entre os conceitos. Não existem formalizações para a estrutura de rede semântica, também não existem métodos formais de inferência. As redes semânticas são bastante úteis em situações onde o processamento do conhecimento é classificatório por natureza. A adição e a exclusão de vértices e arestas podem ser realizadas de forma simples, porém, por não existir ainda uma formalização para representar redes semânticas, devemos ter um cuidado muito grande para que os termos utilizados em arcos e vértices não nos levem a conclusões erradas sobre o sistema a ser estudado.

Na quase totalidade das formalizações de redes semânticas existe pelo menos um tipo de arco que relaciona conceitos mais específicos à conceitos mais genéricos.

Estes arcos são utilizados para organizar estes conceitos em uma hierarquia ou taxonomia. Esta organização permite armazenar as informações em níveis apropriados de generalidade (especificidade).

Mecanismos de herança permitem a troca de informações entre os níveis. Pode-se dizer que um conceito mais genérico subordina conceitos mais específicos, e que um conceito mais específico herda informações dos conceitos mais genéricos.

Classificação é o nome do processo pelo qual novos conceitos são adicionados a uma taxonomia existente *(Woods,91)*. Um dos propósitos de se ter uma taxonomia é alocar instâncias ao conceito mais específico possível. Assim, a taxonomia conceitual serve como uma estrutura de organização para classificar indivíduos e agrupá-los de acordo com suas características comuns.

Em problemas de classificação o especialista utiliza-se das observações disponíveis para obter as conclusões desejadas. Se considerarmos tanto as conclusões desejadas quanto as observações disponíveis como conceitos que devem ser relacionados entre si, então, um problema de classificação pode ser interpretado como o problema de mapear corretamente conceitos em conceitos. Estes conceitos podem representar objetos, seus atributos, ou até mesmo relacionamentos entre estes. O mapeamento entre os conceitos representa uma dependência conceitual entre dois conjuntos de conceitos.

Deste modo, um problema de classificação pode ser definido como um conjunto de conceitos e outro conjunto de regras de classificação, que mapeiam um conjunto de conceitos de entrada em um conjunto de conceitos de saída.

Matematicamente, pode ser definido pela tupla $\{\mathbf{C}, \mathbf{R}\}$, onde $\mathbf{C}$ representa o conjunto dos conceitos e $\mathbf{R}$ o conjunto aplicável de regras de formação de conceitos. Cada regra $\mathbf{r} \in \mathbf{R}$ deve mapear um conceito $\mathbf{c} \in \mathbf{C}$ a partir de um sub-conjunto de conceitos de $\mathbf{C}$.

O estudo dos Problemas de Classificação é fundamental na resolução de grande parte dos problemas dos Sistemas Elétricos de Potência.

II.7 - Conclusões

Os estudos *(Tamura,89) (Liu,92)* realizados pelos órgãos competentes (CIGRÉ, EPRI e IEEE) concluiram que o sucesso das aplicações de Inteligência Artificial dependem da qualidade do conhecimento adquirido/representado.

No entanto, pesquisas realizadas sobre aplicações práticas existentes de Inteligência Artificial *(Wildeberger,92)* mostraram que a maioria absoluta destas aplicações utilizaram-se de entrevistas para adquirir o conhecimento necessário e evidenciaram que este conhecimento mostrou-se insuficiente na hora de implantar o sistema. Além disso, alterações no domínio do conhecimento provocam imensos esforços de reengenharia nos sistemas já implantados.

Os artigos referentes às aplicações reais citam freqüentemente que uma de suas maiores dificuldades está no levantamento do conhecimento necessário. É extremamente difícil obter as regras de produção necessárias ao Sistema Especialista, uma vez que o próprio especialista encontra dificuldades em descrever seu conhecimento através de regras. Além disso, os dados existentes nem sempre são suficientes (ou sequer existem) para treinar uma Rede Neural. Finalmente, alterações geralmente levam a reescrever as regras de produção ou a retreinar a Rede.

A obtenção das regras para a base de dados não é tarefa trivial, visto que o operador não costuma fornecer uma idéia muito clara do que está fazendo durante o seu trabalho. Isto é, ele realiza muitas de suas tarefas em um nível subconsciente e pode ser difícil, para ele, explicá-la em um nível de consciência *(Gingrich)*.

Existe, portanto, a necessidade de um sistema que permita adquirir o conhecimento do especialista de um modo mais natural e que seja mais adaptativo a possíveis alterações no problema em questão. O estabelecimento de um método para aquisição e representação do conhecimento é indispensável à implantação de sistemas práticos. A partir das possíveis aplicações e dos exemplos anteriores podemos fazer um levantamento dos requisitos necessários e desejáveis deste sistema:

1. **Capacidade de aprender através de exemplos:** os problemas são complexos e de difícil caracterização, onde o conhecimento envolvido pode ser melhor definido através de exemplos de entradas e suas respectivas saídas.
2. **Capacidade de generalização:** nem sempre todos os dados necessários à resolução de um problema estão disponíveis. O sistema deve ser capaz de responder satisfatoriamente, não só à problemas completos, mas também à incompletos. Além disso, deve ser capaz de responder à problemas nunca antes vistos, a partir da generalização dos que consegue resolver.
3. **Capacidade de classificação:** o sistema deve ser capaz de, a partir dos exemplos, induzir uma classificação de conceitos e relações entre estes, criando um modelo do mundo exterior.
4. **Capacidade de discernimento:** o sistema deve ser capaz de responder simultaneamente a dois (ou mais) problemas distintos, sendo capaz de discernir entre eles.
5. **Não-monotonicidade:** o conhecimento a ser representado no sistema não é monotônico. Ou seja, o conhecimento anteriormente existente pode ser alterado a partir da aquisição ou inferência de um novo conhecimento.
6. **Adaptabilidade:** o sistema deve ser capaz de suportar modificações no problema em questão sem que isto provoque grandes esforços de reengenharia.
7. **Preservação do conhecimento:** no caso de modificações, o sistema deve preservar o conhecimento anterior, de modo a utilizá-lo na resolução do novo problema.

Seria desejável que este sistema pudesse ficar permanentemente ligado, aprendendo mesmo em pleno funcionamento. De preferência, junto com o próprio especialista, acompanhando suas decisões.

A habilidade de aprender, adaptar e modificar o comportamento é parte inalienável da inteligência. Existe um grupo de pesquisadores preocupado em conseguir fazer com que computadores sejam capazes de aprender. Esta parte da

Inteligência Artificial denomina-se Aprendizado de Máquina ("Machine Learning"), e existe uma convicção entre os seus estudiosos que o aprendizado é um pré-requisito para qualquer forma de inteligência *(Carbonell,89)*. Uma boa referência para os interessados no tema é um volume especial do periódico "Artificial Intelligence" sobre Aprendizado de Máquina *(Artificial Intelligence, vol.40, no.1-3, 1989)*.

A maioria dos sistemas para Aprendizado de Máquina são baseados em alguma especialização da Lógica tradicional. A proposição básica da lógica *(Manna,74) (Casanova) (Sundholm,83)* como ferramenta fundamental para a modelagem da inteligência é que o conhecimento humano pode ser codificado em um conjunto de predicados, e que o raciocínio realiza-se como uma aplicação dos métodos de inferência lógica sobre estes axiomas para gerar novos predicados.

Existe a "ilusão" de que o sistema estaria aprendendo novas peças de conhecimento quando, na realidade, está apenas reescrevendo um conhecimento já codificado nos axiomas. A incapacidade da lógica em lidar com o aprendizado decorre do pressuposto de monotonicidade, cuja aplicação obriga que nenhum novo predicado contradiga qualquer demonstração anterior *(Rocha,92)*. Por este motivo, a lógica precisa que seja fornecido, a priori, TODO o conhecimento necessário à resolução de tarefas em um dado domínio (Hipótese de Fechamento do Universo). Esta hipótese, em conjunto com a Prova por Falha Finita, constitui a base da Abdução.

Alguns pesquisadores têm se precupado em embutir mecanismos de raciocínio não-monotônico nos sistemas baseados em lógica *(Reiter,87) (Poole,87) (Poole,88) (Nute,88) (Nute,91) (de Kleer,92) (Stein,92)*. Uma boa referência sobre o assunto pode ser encontrada em *Lukaszewicz (90)*.

Atualmente, cada vez mais utilizamos máquinas na resolução dos mais variados problemas. Porém, para que isto seja possível, todo o conhecimento necessário à sua resolução deve estar explicitamente presente. A Inteligência Artificial tem buscado resolver tipos de problemas, denominados ***incompletos***, onde o sistema precisa criar, representar e utilizar novos conhecimentos. Para resolvê-los, é necessário adquirir novos conhecimentos (através de sensores, banco de dados ou mesmo de uma

Interface Homem-Máquina), ou inferí-los através de outros já existentes (pela aferição dos conhecimentos implícitos). Assim sendo, precisamos representar o conhecimento adequadamente, de modo a podermos adquirir e inferir novos conhecimentos eficientemente, tal como no cérebro humano.

Fahlman (81) afirma que para a Inteligência Artificial obter sucesso na resolução de problemas incompletos envolvendo grande quantidade de informações, é necessário paralelizar o processamento destas. Deste modo, é tentador utilizar Redes Neurais (paralelismo explorado ao máximo) para a aquisição do conhecimento. Estas aparecem como uma plausível opção graças à sua capacidade de aprendizado através de exemplos e utilização em tempo real. Além disso, apesar das dificuldades reportadas, elas satisfazem os 5 primeiros requisitos dos 7 necessários, tornando promissora sua utilização.

Portanto, o estudo de Redes Neurais aplicadas em problemas de Classificação é o ponto de partida para a resolução dos problemas dos Sistemas Elétricos de Potência.

Capítulo III

III - REDES NEURAIS

"What is mind ? No matter.

What is matter ? Never mind."

Thomas Hewitt Key (em *"Punch"*)

III.1 - Redes Biológicas

Muitos pesquisadores consideram Redes Neurais como sistemas de plausibilidade biológica, porém, preferimos dizer que são sistemas que possuem inspiração biológica. As Redes Neurais representam um novo conceito de sistema de processamento onde os modelos são feitos com base no que se sabe dos princípios do processamento neurofisiológico.

O cérebro é composto por células altamente diferenciadas denominadas ***neurônios*** (Figura III.1). Cada neurônio possui um corpo celular, ou ***soma***, dentro do qual encontram-se a maioria dos seus orgânulos. Do soma de cada neurônio partem prolongamentos que podem ser funcionalmente divididos em ***dendritos*** e ***axônios***.

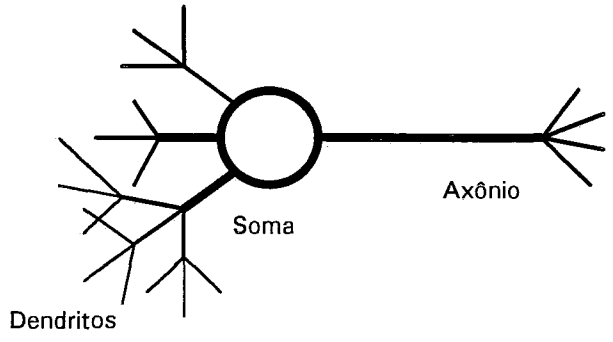


Figura III.1 - Esquema Simplificado de um Neurônio

O neurônio é uma célula altamente especializada, capaz de receber impulsos elétricos a partir de seus dendritos, processá-los no soma e, finalmente, transmití-los através de seu axônio (geralmente único) para os dendritos (ou mesmo para o soma) de outros neurônios.

A conexão entre um axônio de um neurônio e um dendrito (ou soma) de outro é denominada ***sinapse*** *(Figura III.2)*. A sinapse é a unidade funcional básica para a construção de circuitos neurais biológicos *(Shepherd,78)* e envolve a aposição das membranas plasmáticas de dois neurônios de modo a formar uma junção pontual (o tamanho de uma junção sináptica é menor do que 1 μm) e orientada do neurônio pré-sináptico para o pós-sináptico.

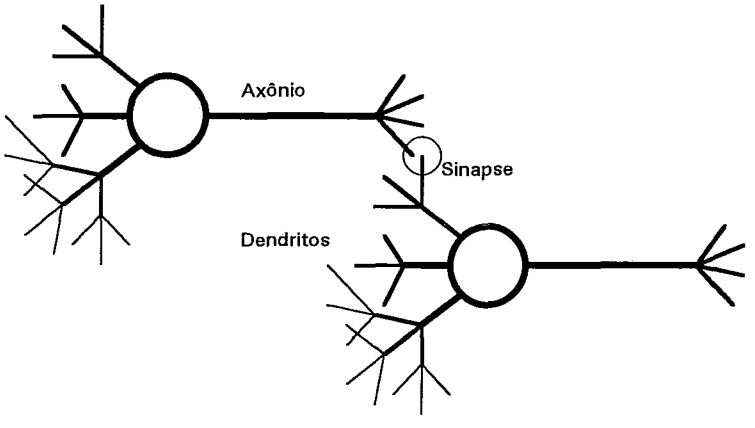


Figura III.2 - Esquema Simplificado de uma Conexão Sináptica

O cérebro humano possui cerca de 100 bilhões (10^{11}) de neurônios, cada qual com aproximadamente 10 mil (10^{4}) interconexões. Deste modo, o número de conexões possíveis é maior do que o número de partículas atômicas que compõem o universo (aproximadamente 10^{100} partículas). Na realidade, estima-se em dezenas de trilhões (10^{13}) o número de conexões existentes no cérebro *(Dayhoff,90)*.

III.2 - Conceitos Básicos

Existem muitas referências sobre a aplicação de redes neurais na resolução de diversos problemas, porém, a maioria refere-se à aplicação de um único tipo de rede (Back-Propagation) cujo desempenho já foi testado e aprovado em determinadas aplicações. Ao contrário do que se imagina, cada modelo de rede possui diferentes propriedades que, devido à características inerentes, as tornam adequadas à resolução de determinada classe de problemas. Estes conceitos são de grande valia para a compreensão dos paradigmas que serão apresentados, além de uniformizar os jargões utilizados na área.

O ***paradigma conexionista*** procura entender e emular as propriedades decorrentes do alto grau de paralelismo e conectividade. As ***redes neurais*** são modelos que utilizam o paradigma conexionista na resolução de determinado tipo de problemas.

Uma rede neural é composta por um elevado número de ***elementos processadores***, também denominados ***unidades de processamento***, amplamente conectados entre si. Cada uma das ***conexões*** interliga somente dois elementos processadores, geralmente em um único sentido, e possui um valor que determina o grau de conectividade entre estes, denominado ***peso*** da conexão.

Deste modo, todo o processamento é realizado distribuidamente entre os elementos processadores da rede, onde cada qual o realiza isolada e paralelamente, enviando seu resultado para outras unidades através das conexões entre eles. Por isso, as Redes Neurais também são conhecidas como sistemas de ***processamento distribuído e paralelo (PDP)***.

A forma pela qual os elementos processadores estão interligados é denominada ***topologia*** ou ***padrão de interconexão***. Uma rede é dita rígida ou fixa quando sua topologia não pode variar; caso contrário, é denominada flexível ou plástica, uma vez que poderá crescer e diminuir de acordo com a necessidade do sistema. As redes rígidas são mais simples e, conseqüentemente, as mais utilizadas. Um dos maiores problemas em sua utilização é determinar o número de elementos processadores a

serem utilizados e a topologia das conexões entre eles. Apesar dos inúmeros artigos sobre este assunto, o método mais utilizado continua sendo o de "tentativa e erro". Existem poucas redes plásticas conhecidas. Estas são particularmente úteis quando o problema não é conhecido ou quando ele varia no tempo.

Normalmente, uma rede é dita estática quando o peso das conexões entre os elementos processadores não pode variar, caso contrário é denominada dinâmica. As redes estáticas são utilizadas na resolução de problemas conhecidos e equacionados. As redes dinâmicas possuem mecanismos de "aprendizado", que possibilitam a modificação dos pesos de suas conexões de modo a melhorar seu desempenho. Geralmente, quando se quer implementar uma rede neural em hardware utilizam-se redes estáticas. Porém, até chegar ao modelo estático ideal, utiliza-se uma rede dinâmica em software.

A dinâmica ou plasticidade da rede neural pode ser feita de dois modos distintos: criação (ou remoção) de novos elementos e/ou conexões; modificação nos elementos e/ou conexões existentes *(Rumelhart,86)*. Normalmente, modificam-se os pesos das conexões sem criar ou remover novos elementos. Porém, este pode ser considerado como um caso especial do primeiro, pois alterar o valor de uma conexão de ZERO para qualquer outro valor equivale a criar uma nova conexão, e zerá-la equivale a eliminá-la. Embora isto não represente um grande problema do ponto de vista matemático, pode representar um sério problema na prática, pois os recursos computacionais são limitados e o tempo de processamento é proporcional à quantidade de recursos necessários.

A capacidade das Redes Neurais em resolver um determinado problema encontra-se embutida na topologia (padrão de interconexão) da rede. Ou seja, o modo pelo qual os elementos processadores estão interconectados e os pesos destas conexões determinam o problema que uma Rede Neural é capaz de resolver.

O processo de ***síntese*** de uma Rede Neural consiste em inicializá-la de modo a ser capaz de resolver o problema ao qual se destina. A idéia dos métodos de síntese é gerar, de uma única vez, o padrão de interconexão capaz de resolvê-lo *(Carvalho,89)*.

Caso contrário, ela deve possuir algum processo de ***treinamento*** capaz de modificar gradualmente um padrão de interconexão inicial, de modo a adaptá-lo à resolução deste problema. A síntese de Redes Neurais é muito aplicada em problemas de otimização, cuja equação de resolução seja bem conhecida, enquanto o treinamento se aplica aos demais problemas. É importante ressaltar que as redes treináveis não estão livres do problema de síntese. A inicialização dos pesos das conexões (síntese) pode ser um fator determinante do sucesso ou fracasso do treinamento destas na resolução de um problema.

Os mecanismos de aprendizado possibilitam a modificação do padrão de interconexão de uma Rede Neural, capacitando-a a resolver um determinado problema. Para treinar uma Rede Neural podem ser utilizados três mecanismos distintos de aprendizado: o ***aprendizado supervisionado***, quando são fornecidos integralmente os resultados desejados; o ***aprendizado por reforço***, quando apenas um parâmetro externo de comparação (ou medida relativa da adequação) para saber se estão agindo corretamente ou erroneamente; e o ***aprendizado não-supervisionado***, quando a própria rede é capaz de ajustar o seu funcionamento.

A maioria absoluta das aplicações existentes compõe-se de redes neurais com aprendizado supervisionado, que pode ser considerado como a capacidade que a rede possui de modificar o seu desempenho a partir da comparação entre a resposta obtida e a a resposta desejada (*Figura III.3*).

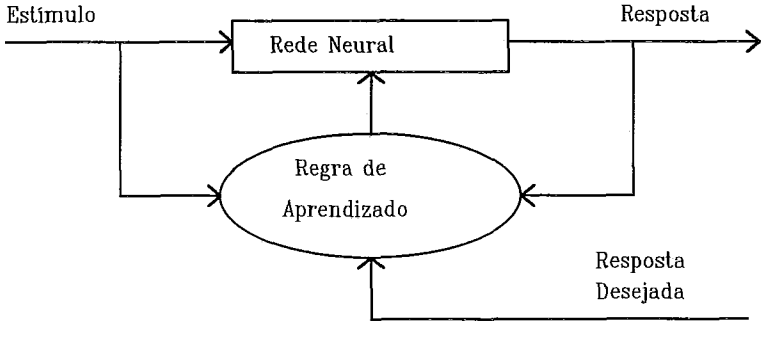


Figura III.3 - Aprendizado em Redes Neurais Supervisionadas

As redes não-supervisionadas, por sua vez, Têm a capacidade de determinar uma correlação entre os possíveis padrões de entrada e são particularmente úteis nos problemas em que as entradas variam com o tempo de forma conhecida. Podemos considerar este mecanismo de aprendizado como sendo a capacidade que a rede possui de abstrair correlações entre os estímulos de modo a obter as respostas desejadas (*Figura III.4*).

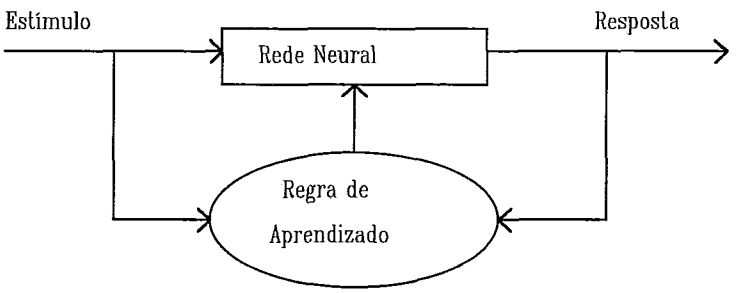


Figura III.4 - Aprendizado em Redes Neurais Não-Supervisionadas

Os demais algoritmos de aprendizado que podem ser encontrados em Redes Neurais utilizam uma mistura desses dois mecanismos de aprendizado.

Costuma-se dizer que são necessárias 3 (três) fases para aplicar Redes Neurais à resolução de um problema qualquer. Na primeira fase, ou ***treinamento***, ensina-se a rede a resolver um conjunto de padrões de saída associados à padrões de entrada. Na segunda fase, ou ***teste***, são apresentados padrões de entrada à rede, e as saídas obtidas são comparadas às saídas desejadas. Na terceira fase, ou aplicação, também é conhecida como ***lembrança*** ("recall"), a rede aprovada na fase anterior é utilizada na resolução do problema. A primeira fase é a única em que há aprendizado, e o processo como um todo pode se repetir até que a rede obtenha resultados satisfatórios. Normalmente, as fases supra-citadas compõem o ciclo de vida de uma rede neural. Porém, nem todos os tipos de redes utilizam este "ciclo de vida".

De um modo geral, o problema de inicialização da rede neural é ignorado, ou seja, a determinação da topologia e a inicialização dos pesos das conexões são problemas desprezados ou minimizados pelos paradigmas existentes. Porém, a prática

mostra que estes aspectos são fundamentais em seu desempenho. Assim, preferimos chamar a resolução deste problema de fase zero, ou ***síntese***.

Note que a interação entre as fases é sempre controlada empiricamente, não existindo uma metodologia de desenvolvimento de redes neurais (como, por exemplo, no desenvolvimento estruturado). Existe atualmente um novo tipo de ciclo de vida que pode ser comparado à prototipação. Quando a rede é capaz de aprender, mesmo durante sua aplicação, a rede é dita evolutiva, pois o aprendizado continua indefinidamente. Porém esta mesma denominação também pode ser aplicada às redes que modificam sua topologia durante o aprendizado (caso particular do caso anterior, onde o aprendizado continua indefinidamente).

O modelo de elemento processador normalmente possui N entradas e 1 única saída (*Figura III.5*) e seu processamento consiste em transferir para sua saída um valor calculado a partir de outros valores presentes em suas entradas, através de uma função denominada ***função de transferência***.

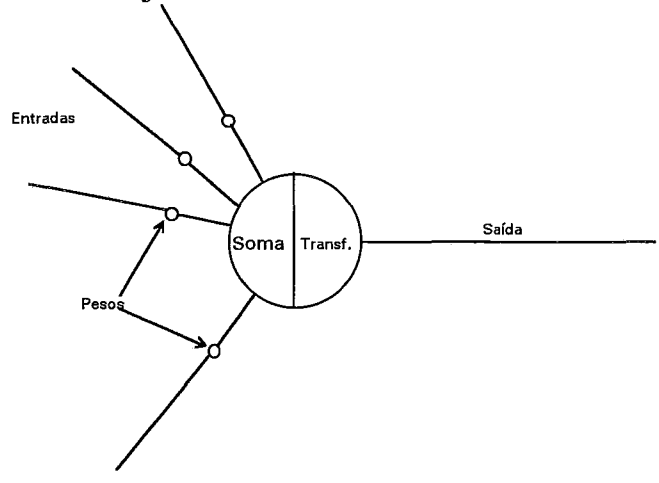


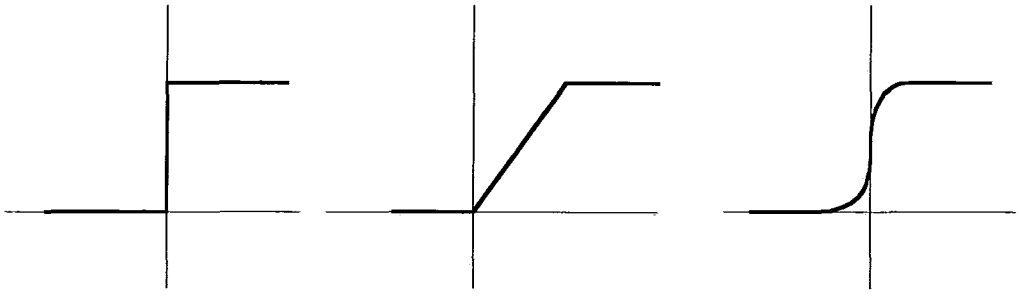
Figura III.5 - Modelo Simplificado de um Elemento Processador

Usualmente, as entradas são combinadas por uma simples soma ponderada e transferidas para a saída através de uma função degrau (*Equação III.1*).

$$\textbf{saída} = \textbf{degrau}\left(\Sigma\ \textbf{entradas}*\textbf{peso}\right)$$

Equação III.1

Existem outras funções de transferência além da ***função threshold*** (*Figura III.6a*), como a ***função linear*** (*Figura III.6b*) e a ***função sigmoidal*** (*Figura III.6c*).



(a) Threshold (b) Linear (c) Sigmóide

Figura III.6 - Funções de Transferência

O valor da saída de um elemento processador em um instante de tempo é denominado ***estado de ativação do elemento processador***. O conjunto dos estados de ativação de cada um dos elementos processadores de uma rede neural é denominado ***função de ativação da rede neural*** ou às vezes também denominadas ***estado de ativação da rede neural***.

Os elementos processadores são, geralmente, agrupados em pelo menos duas ***camadas***. Uma camada para apresentação dos dados à rede (***camada de entrada***) e outra para obtenção dos resultados (***camada de saída***). A maior parte dos modelos possui, além destas, uma ou mais ***camadas intermediárias***, também denominadas ***camadas escondidas***.

As camadas são processadas no sentido da camada de entrada para a de saída, de modo que à cada padrão de entrada corresponderá um padrão de saída. Normalmente, não existe conexão entre os elementos de uma mesma camada, cujos elementos se conectam com os elementos das camadas imediatamente anterior e posterior *(Figura III.7)*. Porém, em alguns modelos, os elementos se conectam internamente na camada, ou, com camadas não adjacentes.

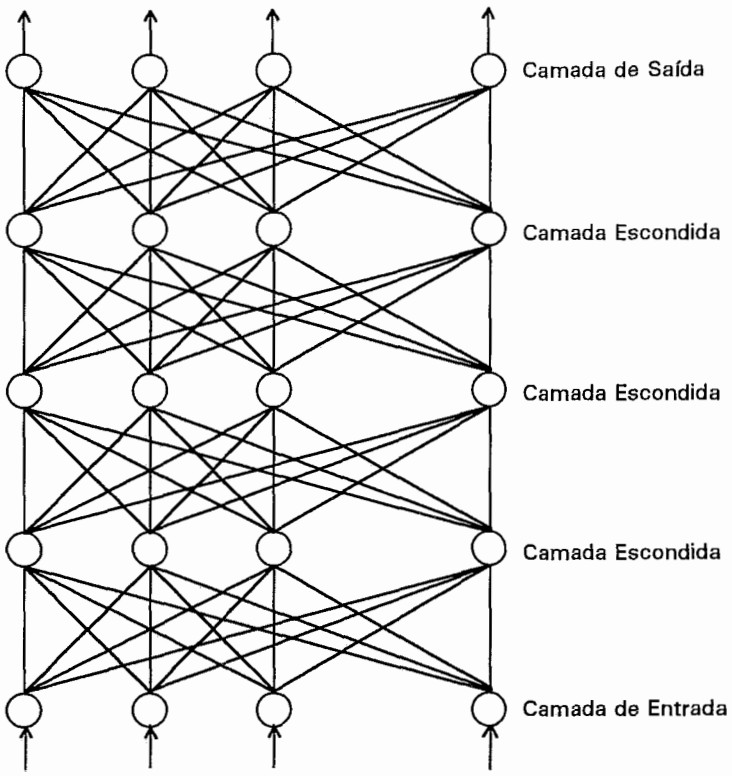


Figura III.7 - Rede Neural de 5 Camadas

O número de elementos em uma camada pode variar de um único (como em algumas camadas do modelo ART) a todos os elementos processadores de uma rede neural (como na rede de Hopfield).

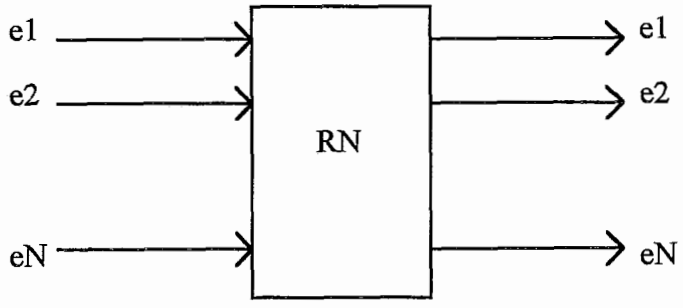


Figura III.8 - Rede Neural Auto-Associativa

Uma rede neural pode ser considerada, basicamente, como uma associadora de padrões *(Carvalho,89)*. Se os padrões de entrada forem iguais aos padrões de saída desejados, a rede é denominada ***auto-associativa*** *(Figura III.8)*, caso contrário, é denominada ***hetero-associativa*** *(Figura III.9)*.

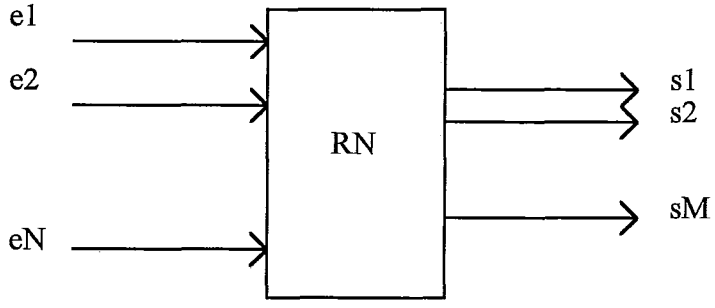


Figura III.9 - Rede Neural Hetero-Associativa

É possível encontrar na bibliografia *(Rumelhart,86a) (Pessoa,90)* quem classifique as Redes Neurais em quatro ou mais paradigmas de funcionamento: "Auto-Associador", "Associador de Padrões", "Classificador" e "Detector de Regularidades". Na realidade, existem mais do que quatro modos de funcionamento, sendo possível qualquer forma de classificação. Porém, se desejamos nos restringir aos padrões apresentados e obtidos, as duas classificações acima representam os dois extremos possíveis.

Os padrões são processados de acordo com a topologia das redes neurais, que dividem-se em dois tipos distintos: ***redes sem realimentação*** (feed-foward, não-recursivas, direcionadas) e ***redes com realimentação*** (realimentadas, recursivas). Nas redes sem realimentação, a informação flui em alguma direção, de modo que, ao ser processada por um elemento processador, não retorne a ele. Já nas redes com realimentação, a informação processada pelo elemento processador poderá ser novamente utilizada por este. Esta nomeclatura é análoga à utilizada em circuitos elétricos e eletrônicos.

A representação gráfica proporciona um meio de saber se uma rede neural possui ou não realimentação. As redes sem realimentação podem ser representadas graficamente por um dígrafo acíclico, já as redes realimentadas são representadas por dígrafos que possuem ciclos. Assim, se o dígrafo que representa a Rede Neural apresentar ciclos a rede possui realimentação, caso contrário não possui.

As redes sem realimentação só possuem uma memória de curtíssimo prazo, sendo as únicas informações armazenadas relativas ao processamento imediatamente anterior de cada elemento. Nelas, os padrões de entrada são processados até originarem os padrões de saída, e pode-se garantir que a rede chegará ao estado final após o processamento de seus elementos.

Porém, em uma rede realimentada é necessário algum mecanismo capaz de garantir que ela oscilará até atingir um estado estável nas suas saídas em um determinado intervalo de tempo.

Devemos ressaltar que existe uma certa confusão entre redes realimentadas ou não-realimentadas e redes auto-associativas ou hetero-associativas. Isto deve-se ao fato de que, normalmente, as redes hetero-associativas utilizam uma topologia sem realimentação (e.g. Back-Propagation), enquanto que as auto-associativas utilizam topologia com realimentação (e.g. Hopfield). Porém, existem contra-exemplos clássicos, como a rede ART, que apesar de hetero-associativa possui realimentação, e a rede Counter-Propagation, que apesar de auto-associativa não possui realimentação.

III.3 - Histórico e Paradigmas

Em 1943, McCulloch desenvolveu um modelo matemático simplificado de um neurônio *(Figura III.10)*. Nele, o neurônio possuia apenas uma saída (**s**), que era uma função degrau ("threshold") da soma de suas diversas entradas (**e**).

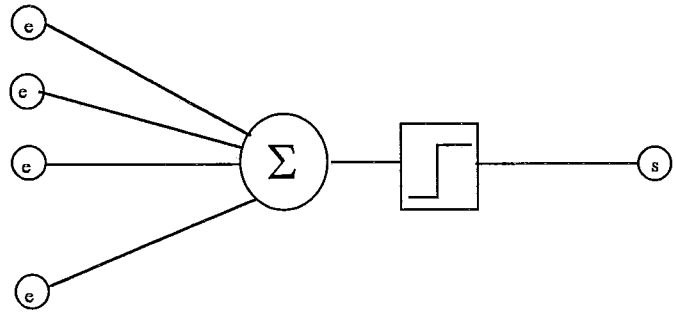


Figura III.10 - Modelo matemático simplificado de um neurônio de McCulloch.

Em 1947, McCulloch e Pitts criaram redes interligando vários elementos processadores (neurônios de McCulloch) e demonstraram que estes eram capazes de executar ações complexas quando conectados a outros elementos semelhantes.

Em 1949, Hebb sugere que a alteração da eficiência sináptica é a base do aprendizado, através do seguinte postulado: "Quando uma célula A está suficientemente próxima para excitar uma célula B e repetida ou persistentemente toma parte no disparo desta, algum processo de crescimento ou mudança metabólica ocorre em uma ou ambas as células de modo que a eficiência de A em excitar B é aumentada.". Então, segundo Hebb, uma determinada conexão somente é reforçada se tanto as células pré-sinápticas quanto as pós-sinápticas estiverem excitadas.

III.3.1 - "Perceptrons"

Em 1957, Rosemblatt mostra o modelo dos ***Perceptrons***. Nele, os elementos processadores (neurônios) estão organizados em duas camadas (uma para entrada e outra para saída) totalmente conectadas entre si (*Figura III.11*). Estas conexões eram direcionadas da camada de entrada para a de saída. O grupo de Rosemblatt enfocava o problema de como achar os pesos apropriados das conexões entre cada elemento processador (eficiência sináptica estabelecida entre o neurônio pré-sináptico e o pós-sináptico). Ele conseguiu criar um algorítmo que alterava estes pesos de modo a realizar a computação desejada, que foi denominado ***algoritmo de aprendizado***.

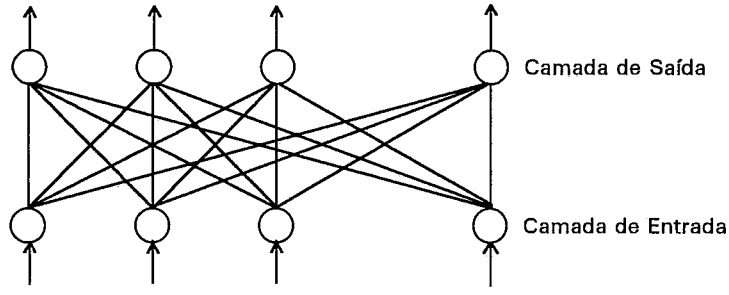


Figura III.11 - Perceptrons, de Rosemblatt

Cada um dos elementos processadores possui uma função degrau de ativação, e calcula sua saída a partir de uma soma ponderada de suas entradas. Ou seja, a saída

s_j representa o resultado da soma das n entradas de um elemento processador j *(Equação III.2)*.

$$s_j = \sum_{i=0}^{n} u_i w_{ji} ,$$

Equação III.2

onde u_i representa o estado de ativição do elemento processador **i**, e w_{ji} é o peso da conexão entre **i** e **j**. E cada u_j é dado pela *Equação III.3*.

$$u_j = \begin{cases} 0, & se \quad s_j <= 0 \\ 1, & se \quad s_j > 0 \end{cases}$$

Equação III.3

Os Perceptrons funcionam como classificadores de padrões binários (0/1). A atualização dos pesos da única camada ajustável de conexões pode ser feita por diferentes regras de aprendizado, porém todas utilizando o mesmo princípio *(Equação III.4)*. Onde w_{ji} é o peso da conexão entre **i** e **j** ; u_i é o valor da saída do elemento **i** ; u_j é a saída do elemento **j** e t_j é a saída desejada do elemento **j** ; η é uma pequena constante denominada **taxa de aprendizado**.

$$w_{ji} = w_{ji} + \eta * (t_j - u_j) * u_i ,$$

Equação III.4

Segundo esta regra, quando a saída desejada for maior do que a saída real, a taxa de aprendizado é adicionada ao peso, caso contrário, é subtraída. Se a saída real for igual à desejada, não há mudanças no peso.

III.3.2 - "Adaline"

Em 1959/60, Widrow & Hoff criam o ***Adaline***, composto de elementos processadores "adaline" (acrônimo de Adaptive linear neuron), similar aos Perceptrons. Logo em seguida criam também o ***Madaline*** (Multilayer adaline), que possui uma arquitetura onde dois ou mais elementos processadores tipo "adaline" são utilizados como entrada para um outro elemento tipo "madaline", cuja saída nada mais é do que a resposta de maior incidência entre os "adalines", criando a primeira arquitetura de 3 (três) camadas (*Figura III.12*).

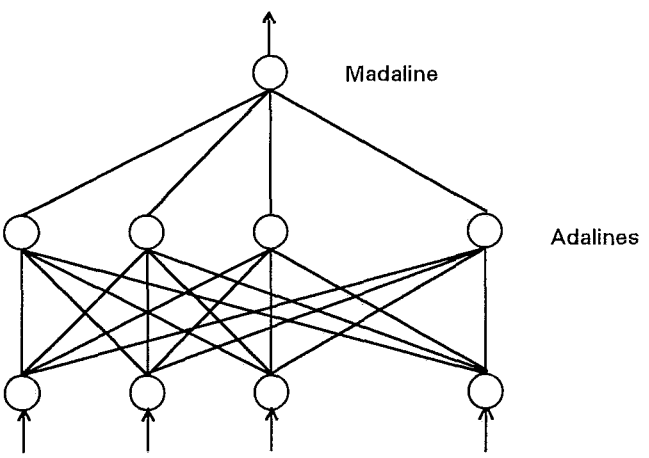


Figura III.12 - Madaline, de Widrow & Hoff

O modelo Adaline é similar ao Perceptron. Os elementos processadores também são binários, porém, variam em {-1,1}. Ou seja, embora o Adaline funcione como uma soma ponderada de suas entradas, sua saída é dada pela *(Equação III.5)*.

$$u_i = \begin{cases} -1, & se \quad s_i < 0 \\ 1, & se \quad s_j >= 0 \end{cases}$$

Equação III.5

Em 1966, Minsky publica seu livro "Perceptrons", divulgando as limitações dos Perceptrons. Cada unidade de saída só poderia classificar entradas linearmente separáveis *(Capítulo V)* e não conseguiria, nem mesmo, aprender a resolver problemas simples como a função XOR (ou-exclusivo). Poucos pesquisadores - como Werbos,

Anderson e Grossberg - continuaram as pesquisas na área, porém, sem o interesse outrora legado pelos demais pesquisadores da comunidade científica internacional.

III.3.3 - "Hopfield"

Em 1982, a partir da publicação dos trabalhos de *Hopfield (82)*, que aplicava Redes Neurais em otimização, o interesse na área tornou a crescer. Ele apresentou um novo modelo de rede no qual todos os elementos processadores estavam totalmente interconectados entre si *(Figura III.13)*.

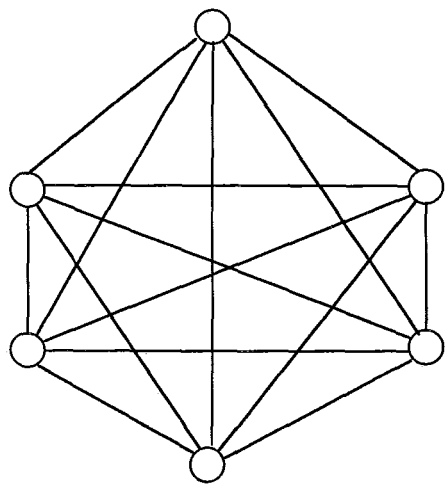


Figura III.13 - Rede de Hopfield

O modelo desenvolvido por Hopfield era essencialmente distinto de seus predecessores. Este tipo de topologia tornava a rede recursiva pois a saída de cada elemento servia como entrada para todas as demais unidades. Era necessário, então, garantir que a Rede chegasse a um estado estável na ausência de entradas externas. Para tanto, *Hopfield (86)* associou o estado da rede à uma ***função de energia*** definida pela *Equação III.6*.

$$E = -\frac{1}{2}\sum_{j}\sum_{i(i\neq j)} T_{ji}u_j u_i$$

Equação III.6

Demonstra-se que a rede de Hopfield funciona de modo a minimizar esta função de energia, até alcançar um estado onde ela seja mínima (estável). Isto é possível, desde que o procedimento de atualização somente reduza (ou mantenha) o valor desta função. Este, no entanto, não garante achar o mímino global de energia *(Hopfield,86)*, pois, avançar em direção à um estado de menor energia que o anterior pode levar a um mínimo local da mesma *(Figura III.14)*.

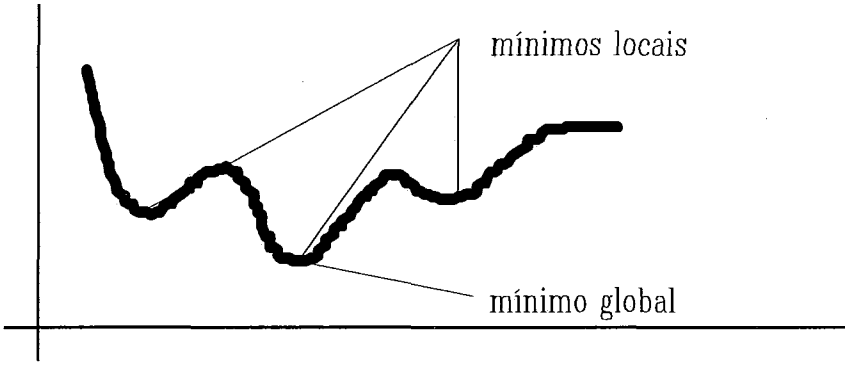


Figura III.14 - Mínimos Locais de uma Função

As Redes de Hopfield funcionam como uma memória associativa, onde cada padrão a ser memorizado deve ser escolhido à priori e associado à um mínimo da função de energia da rede. Porém, na prática, o efeito dos mínimos locais faz com que o padrão retornado não necessariamente seja o mesmo do padrão desejado e sim outro qualquer (correspondente a um mínimo local). Um outro efeito colateral ainda pior é que este mínimo local pode não corresponder a nenhum padrão desejado de memória.

III.3.4 - "Back-Propagation"

Em 1986, as pesquisas retomadas culminam quando Rumelhart, McClelland e o "PDP Research Group" redescobrem as Redes Neurais e publicam dois volumes do mais utilizado livro no assunto: Parallel Distributed Processing *(Rumelhart,86)*. Nele, foi divulgado por Rumelhart um modelo desenvolvido por Werbos em 1974, e reinventado independentemente por Parker em 1982, denominado ***Back-Propagation***. Apesar de parecer, em muito, com seu antecessor Perceptrons, difere deste por

permitir a utilização de mais de duas camadas de elementos processadores (*Figura III.15*) sendo, por isso, conhecido também como ***Perceptron multi-camadas***. Ou seja, além das camadas de entrada e saída, podem ser acrescentadas uma ou mais camadas intermediárias (ou escondidas).

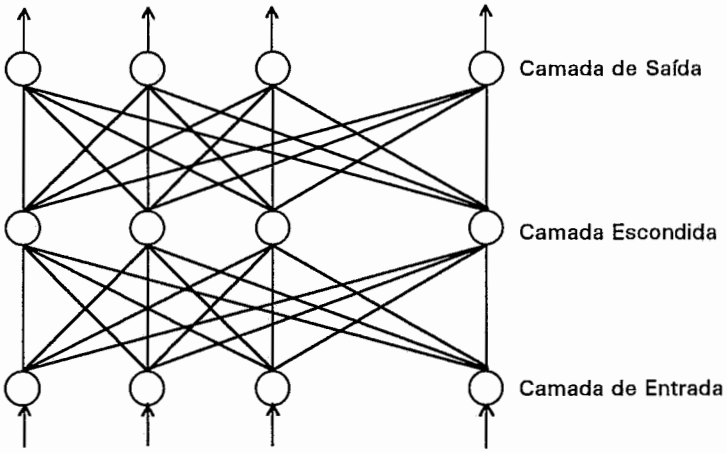


Figura III.15 -Perceptron multi-camadas

Neste modelo, os elementos processadores estão organizados em camadas, estando cada elemento de uma camada, amplamente conectado à camada subseqüente. O padrão apresentado é propagado até a camada de saída (foward-propagation), onde é calculado o erro entre a saída real e a saída desejada de cada elemento processador da camada de saída *(Equação III.7)*. Ou seja, o erro ε_j de um elemento **j** é calculado entre a diferença entre a saida desejada t_j e a calculada u_j.

$$\varepsilon_j = (t_j - u_j)$$

Equação III.7

Este erro então é propagado de volta ("backward") para os elementos conectados a **j** através dos respectivos pesos das conexões *(Equação III.8)*. Onde o erro ε_j para cada elemento **j** que não pertença à última camada pode ser calculado através da soma ponderada de cada um dos **k** erros ε_k propagados através da conexão w_{kj}.

$$\varepsilon_j = \sum_k \varepsilon_k w_{kj}$$

Equação III.8

A variação dos pesos é dada pela *Equação III.9* onde a variável η representa o coeficiente de aprendizado.

$$\Delta w_{ji} = \eta \varepsilon_j u_j$$

Equação III.9

A utilização deste algoritmo, denominado ***Backward-Error-Propagation***, permitiu treinar redes neurais que possuem camadas intermediárias, resolvendo a principal limitação dos Perceptrons. Este é, sem dúvida, o paradigma de Redes Neurais mais utilizado em todo o mundo e têm sido aplicado com sucesso nas mais diversas áreas.

Desde então, a maior parte das pesquisas na área tem se dedicado a utilizar variações dos modelos existentes em diversas aplicações. Esta tarefa foi facilitada com o advento de ferramentas (principalmente simuladores) para a experimentação destes modelos. Dentre os principais modelos que surgiram neste período podemos destacar o Brain-State-in-a-Box (BSB) de Anderson, o Self-Organization-Map (SOM) de Kohonen e a Adaptive Ressonance Theory (ART) de Grossberg.

III.4 - Diferenças entre Redes Neurais Artificiais e Biológicas

Talvez, a fascinação exercida pelas Redes Neurais deva-se, em parte, a seu relacionamento com o cérebro. É comum ouvir dizer que as Redes Neurais Artificiais explicam os mecanismos básicos e a dinâmica do cérebro humano. Porém, apesar do termo redes neurais ser proveniente das redes de células nervosas do cérebro, os presentes modelos diferem drasticamente de suas similares biológicas.

Com o propósito de uma melhor compreensão das funções cognitivas do cérebro, os pesquisadores das Redes Neurais Biológicas (experimentalistas) têm

acumulado um grande número de informações sobre as propriedades moleculares e celulares dos neurônios e de circuitos neurais do cérebro *(Rall,92)*.

Por outro lado, os pesquisadores de modelos de Redes Neurais Artificiais (modelistas) têm se esmerado na análise das propriedades decorrentes do elevado grau de paralelismo e da distribuição do processamento em elementos processadores simples *(Rumelhart,86)*. McCulloch e Pitts mostraram que tudo que pode ser computado em uma máquina de Turing pode ser computado com o auxílio de seu modelo formal de neurônio.

Infelizmente, os modelistas tradicionalmente ignoram a maior parte das informações obtidas pelos neurocientistas, substituindo a complexa estrutura do neurônio por um simples elemento processador de somas e as singulares conexões dos dendritos e axônios por uma matriz de conexões *(Shepherd)*, enquanto estes (por este mesmo motivo) subestimam os resultados obtidos pelos modelistas.

Existe, portanto, uma necessidade crítica de se reduzir a distância entre os experimentalistas e os modelistas, de modo a possibilitar a criação de redes neurais mais realistas e capazes de simular um maior número de funções cognitivas.

O cérebro é um computador extremamente sofisticado e um dos seus principais desafios é entender como ele realiza o processamento de informações. De modo a transpor a distância existente entre as teorias computacionais e os dados biológicos é necessário primeiro entender como computações elementares podem ser realizadas no "hardware" do cérebro *(Koch)*.

Os modelos de Redes Neurais consideram a existência de apenas uma conexão entre cada dois elementos processadores, o que equivale a um modelo biológico representando somente o que ocorre entre o axônio de um neurônio e o soma de outro. *Shepherd (78)* mostrou que grande parte do processamento ocorre, não no soma ou no axônio dos neurônios e sim, em seus dendritos.

Os dendritos estão normalmente compondo uma estrutura denominada ***árvore dendrítica*** devido à suas ramificações semelhantes a arbustos. *Rall* aplicou a teoria de cabos na árvore dendrítica para verificar os efeitos desta topologia.

O processo de divergência do sinal elétrico que é enviado do axônio de um neurônio para os dendritos de um outro neurônio é da ordem de apenas 10% a 20%, e não de 100% (total), como nos modelos artificiais.

O processo de convergência de diversos sinais recebidos de neurônios distintos por um único neurônio não é somente temporal, mas também é espacial, dependendo da topologia das sinapses. Muitos neurônios realizam uma soma não-linear de suas entradas, podendo haver, inclusive, processamento lógico (AND,OR,NOT) dentro da árvore dendrítica.

As árvores dendriticas não são lisas, ao contrário, apresentam pequenas estruturas da ordem de 1 a 2 micra, denominadas ***spines***. Ao que tudo indica os spines *(Rall,92)* são responsáveis por grande parte das sinapses *(Shepherd,89)* e, consequentemente, pelo processamento de informações. Hoje em dia acredita-se que os spines são responsáveis pelas memórias de longo-termo (LTM) e pela plasticidade das eficiências sinápticas.

Os estímulos externos regulam a quantidade de neurotransmissores nas sinapses, possibilitando transformar eventos externos em informações neurais. Diferentes transmissores agrupados em diferentes combinações são regulados em diferentes instantes de tempo em diferentes sinapses *(Black,91)*.

Todo modelo de Rede Neural baseia-se na premissa de que o neurônio é a unidade básica de processamento da informação. *Shepherd (78)* considera a sinapse como sendo a verdadeira unidade básica de organização dos circuitos neurais biológicos.

Um dos princípios gerais da biologia é o de que o comportamento de um organismo depende de níveis hierárquicos de organização *(Shepherd,90)*. Podemos começar pela própria sinapse onde mecanismos moleculares realizam a comunicação. As sinapses formam pequenos circuitos envolvendo algumas formas de interação entre as sinapses, denominados micro-circuitos. Estes normalmente encontram-se nos dendritos e seu comportamento é determinado pela distribuição na árvore dendrítica. O próprio neurônio representa um outro nível de organização, e as interações entre

neurônios de propriedades similares ou distintas formam circuitos locais. Estes interligam-se em circuitos interregionais, envolvendo múltiplas regiões em diferentes partes do cérebro.

Dentre todas as diferenças supra-citadas, gostaríamos de destacar aquela que consideramos a mais importante. A comunicação química entre transmissores e receptores fornece um circuito químico plástico "impresso" em um circuito topológico rígido *(Black,91)*. Em termos cognitivos, esta plasticidade confere individualidade (baseada na experiência) à semelhança (genética) dos circuitos neurais.

O futuro da neuro-computação poderá ser altamente beneficiado pelas pesquisas biológicas. Estruturas encontradas nos sistemas biológicos *(Douglas) (Shepherd,79)* podem, e devem, inspirar novas arquiteturas para modelos de Redes Neurais. Do mesmo modo que o desenvolvimento de Redes Neurais podem ser beneficiados pelos estudos biológicos, alguns modelos ou aplicações podem ilustrar e auxiliar a explicar diversas características dos sistemas biológicos. Atualmente, a anatomia e fisiologia das redes existentes em regiões específicas do cérebro é ainda imprecisa *(Levine,91)*.

Apesar de todas as divergências, experimentalistas e modelistas têm procurado reduzir suas mútuas restrições, aumentando o nível de colaboração através de foros internacionais, participações em congressos e trabalhos conjuntos. Acreditamos que os frutos desta colaboração serão colhidos em um futuro bem próximo.

Capítulo IV

IV - REDES NEURAIS EM CLASSIFICAÇÃO

IV.1 - Paradigma Classificador

Rumelhart (86) divide os paradigmas de aprendizado em 2 (dois) tipos essencialmente distintos por seus objetivos:

- Associador de Padrões
- Detector de Regularidades

Um associador de padrões é utilizado quando o objetivo é guardar padrões que possam ser recuperados no futuro, enquanto um detector de regularidades é utilizado para determinar similaridades entre os padrões apresentados.

Porém, no mesmo livro, são relacionados pelo menos 4 (quatro) paradigmas de aprendizado, que variam de acordo com o funcionamento do modelo:

- Auto-Associador
- Associador de Padrões
- Classificador
- Detector de Regularidades

O paradigma da Classificação pode ser considerado como um caso particular da associação de padrões. Neste paradigma de funcionamento de Redes Neurais, existe um conjunto fixo de categorias no qual os padrões de estímulo devem ser classificados. O objetivo é aprender a classificar os estímulos dos exemplos de modo a estar apto para classificar corretamente, no futuro, estímulos diferentes dos utilizados durante o aprendizado.

Os Perceptrons foram os primeiros a associar padrões de entrada a padrões de saída *(Capítulo III)*, porém, apesar destes realizarem uma série de classificações com sucesso, nem sempre conseguiam resolver outras aparentemente simples. Em 1966, Minsky conseguiu demonstrar que os Perceptrons somente são capazes de resolver problemas linearmente independentes, não sendo sequer capazes de aprender a resolver uma função lógica simples como a função ou-exclusivo (XOR), que pode ser representada em uma tabela de valores binários (*Tabela IV.1*), como abaixo:

A	B	XOR(A, B)
0	0	0
0	1	1
1	0	1
1	1	0

Tabela IV.1 - Valores de uma Função Lógica XOR

Se traçarmos o gráfico correspondente à função XOR da tabela acima, obteremos a *Figura IV.1*, na qual os pontos vazios correspondem a valores iguais a 1 e os pontos cheios a valores iguais a 0. Neste gráfico podemos ver que não é possível separar, com uma única reta, os pontos vazios dos pontos cheios.

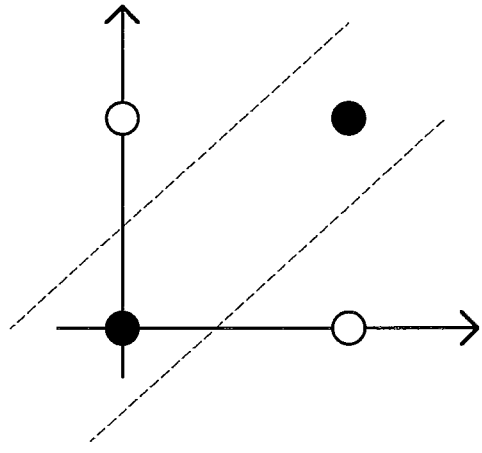


Figura IV.1 - Não-Linearidade da função lógica XOR.

Deste modo, uma rede Perceptron não é capaz de, com um único elemento, separar as duas categorias *(Figura IV.2)*, isto é, as respostas com valor igual a 0 das com valor igual a 1.

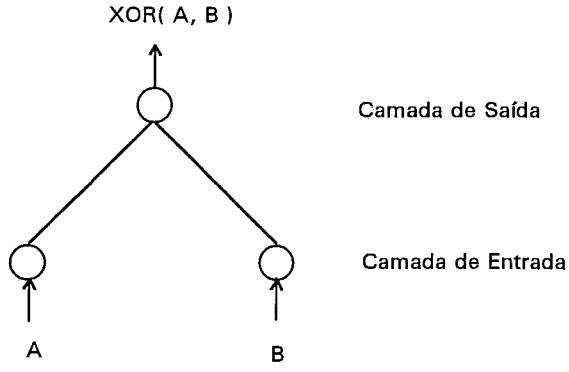


Figura IV.2 - Função XOR em Perceptrons

Em 1986, Rumelhart divulgou as redes Back-Propagation *(Capítulo III)* que, por permitirem a utilização de mais de uma camada de pesos ajustáveis, resolvem o problema da não-linearidade dos Perceptrons. É possível mostrar que uma rede deste tipo com apenas uma camada escondida de elementos processadores pode resolver qualquer função booleana, inclusive a função XOR. *Leshno (93)* prova que é possível utilizá-la para aproximar qualquer função.

Atualmente, uma das principais dificuldades tem sido o elevado tempo de treinamento de uma rede neural. Pequenas modificações no problema obrigam a refazer todo o processo de treinamento, o que se torna ainda mais crítico em problemas de maior porte. Para tentar minimizar estes e outros problemas dos modelos tradicionais, têm-se tentado construir ***redes neurais hierárquicas*** através da composição modular de redes neurais pré-treinadas. Nestes casos, os problemas são subdivididos em problemas menores e pequenos módulos de redes neurais são treinados para resolvê-los. Constrói-se a rede neural a partir da composição hierárquica destes módulos. Apesar de interessante, este "approach" causa alguns problemas de difícil resolução, como a necessidade de uma arquitetura para cada problema e a falta de interação entre os módulos *(Kim,93)*.

IV.2 - Paradigma Categorizador

Muitos modelos tradicionais de Redes Neurais têm sido utilizados com sucesso em problemas de classificação. Na realidade, a maioria destes modelos resolve um caso particular dos problemas de classificação, denominado ***Categorização***, cujo objetivo é determinar a categoria a que pertence o padrão de entrada.

Se considerarmos que um padrão pode ser representado como um ponto em um espaço multi-dimensional, podemos considerar um classificador como um dispositivo que mapeia pontos de um espaço de entrada em pontos de um espaço de saída. Em um categorizador, dado o padrão de entrada (atributos), o objetivo é determinar a categoria ou a classe deste. Para cada categoria **i** temos um conjunto $\mathbf{R_i}$ de pontos do espaço de entrada, que são mapeados nela. Assim, um categorizador é um caso particular de classificador que mapeia pontos do espaço de entrada em um único ponto de saída.

Na última década, surgiram diversos modelos de redes neurais utilizando mecanismos de aprendizado não-supervisionado, no qual não existe um "professor" externo para fornecer as respostas corretas. A rede precisa descobrir, por si própria, padrões de regularidade nas entradas e codificá-los em saídas, mostrando algum nível de ***auto-organização***. Este tipo de rede somente possui utilidade quando existe redundância nos dados de entrada, ou seja, se baseia na premissa que "redundância proporciona conhecimento". Existem, basicamente, dois tipos de aprendizado não-supervisionado: o ***aprendizado Hebbiano***, que procura medir o grau de similaridade entre as entradas ou projetar suas principais componentes; e o ***aprendizado competitivo***, que classifica os padrões de entrada em categorias distintas.

Esse tipo de rede foi inspirado em um mecanismo biológico de ***inibição lateral***, no qual cada neurônio (**S**) procura inibir (**S-I**) os demais neurônios presentes em suas laterais enquanto reforça (**S+E**) a si próprio *(Figura IV.3)*. No aprendizado competitivo, os elementos processadores de uma camada "competem" entre si para determinar um "vencedor", ou seja, o elemento processador que irá reforçar suas conexões.

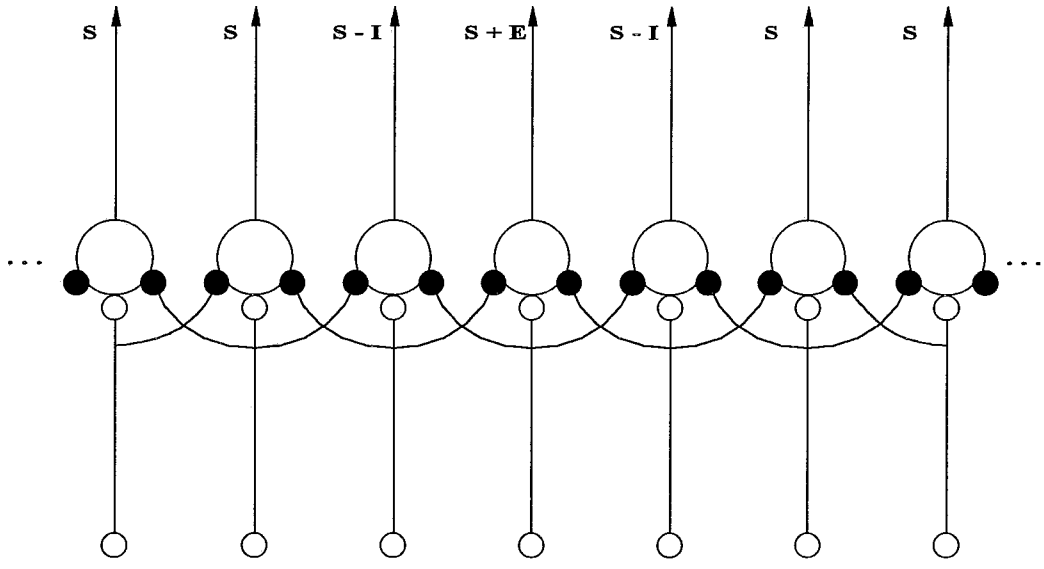


Figura IV.3 - Inibição Lateral

Uma Rede Neural utilizando aprendizado competitivo geralmente possui uma arquitetura que consiste de uma hierarquia de camadas, na qual cada elemento processador está ligado a todos os elementos das camadas adjacentes *(Figura IV.4)*. Em cada camada, as unidades dividem-se em agrupamentos ("clusters") não sobrepostos, nos quais os elementos em cada "cluster" competem entre si. Assim, cada camada proporciona um diferente nível de classificação. Este procedimento também é conhecido como "clustering".

Dentre os modelos que utilizam este tipo de aprendizado podemos destacar o Winner-Take-All (WTA), o Self-Organizing Maps (SOM) e o Adaptive Ressonance Theory (ART).

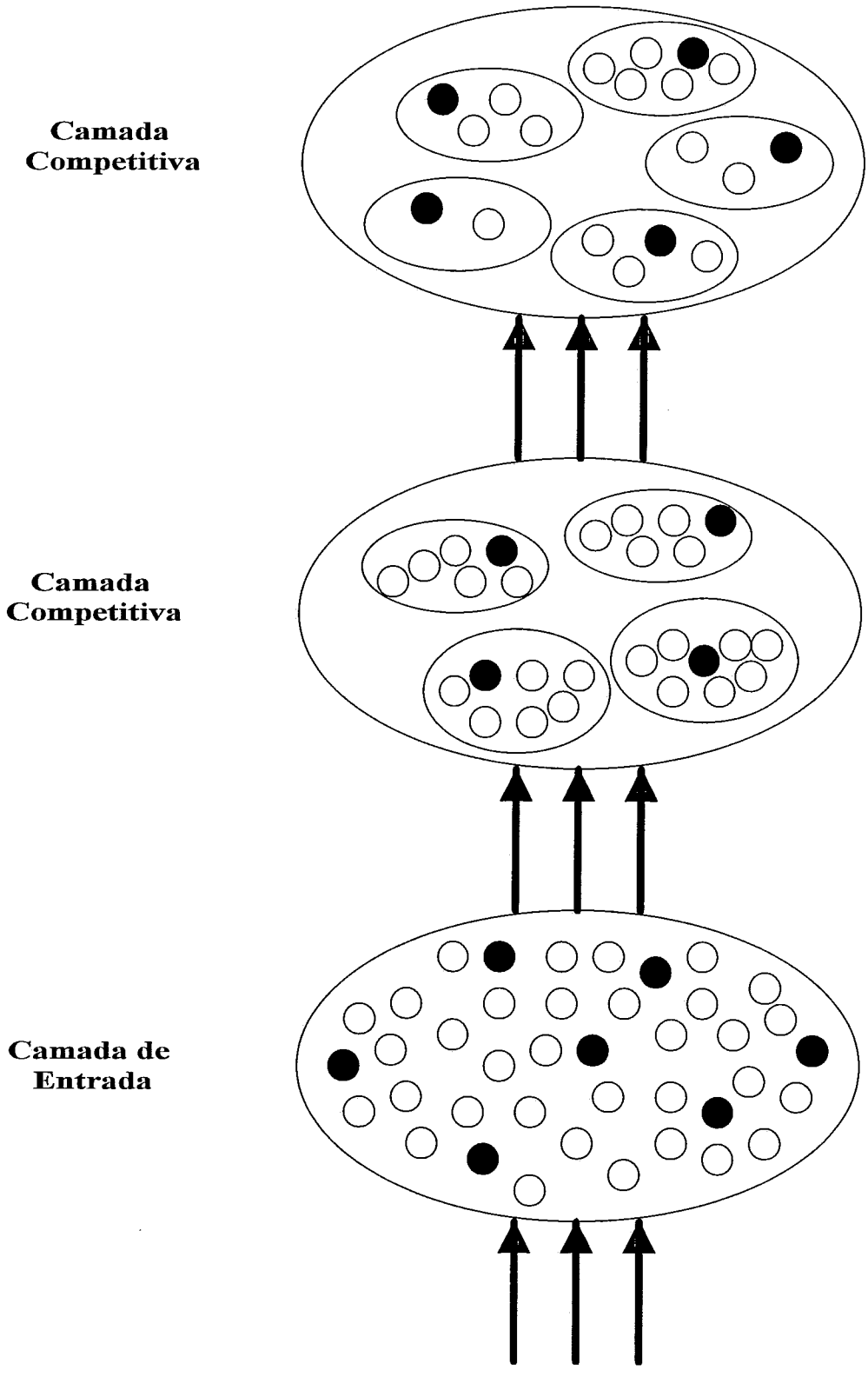


Figura IV.4 - Rede Neural com Aprendizado Competitivo (Rumelhart,86)

IV.2.1 - "Winner-Take-All" (WTA)

É a forma mais simples de aprendizado competitivo, onde os elementos da camada de saída competem entre si e apenas um conquista o direito de responder ao padrão de entrada. Cada elemento da camada de saída é responsável pela representação de uma única categoria. Assim, são necessários tantos elementos quantas categorias existentes, e caso um elemento de saída falhe, perde-se a categoria associada. Além disso, não se pode representar uma hierarquia de categorias.

Como exemplo de uma rede "winner-take-all" podemos imaginar uma rede de duas camadas. A primeira composta por elementos (i) representando a entrada (e_i), e a segunda por elementos (j) representando cada uma das categorias de saída (s_j) desejadas. Para facilitar, considera-se que tanto as entradas como as saídas sejam binárias. As duas camadas estariam totalmente conectadas através de conexões w_{ij} e a segunda camada com seus elementos totalmente interconectados através de conexões w_{jk} (onde k<>j). Supõe-se, também, que todas as conexões w_{jk} sejam iguais a 1. Deste modo, o sinal na entrada (e_j) de cada elemento de saída (s_j) seria dado por:

$$e_j = \Sigma\, e_i \,.\, w_{ij}$$

Equação IV.1

Assim, a saída fornecida por cada elemento poderia ser dada por:

$$s_i = \begin{cases} 0, & se \quad e_i < \max(e_k) \\ 1, & se \quad e_i >= \max(e_k) \end{cases}$$

Equação IV.2

Todo o aprendizado resume-se à determinação dos pesos das conexões w_{ij}. Uma das regras de aprendizado mais utilizadas neste caso pode ser interpretada como uma adaptação dos pesos das conexões entre os elementos de entrada e o elemento vencedor, de modo a diminuir o erro entre os pesos e a entrada real *(Equação IV.3)*.

$$d(w_{ij}) / dt = e_j \ (s_i - w_{ij})$$

Equação IV.3

IV.2.2 - "Self-Organizing-Map" (SOM)

O modelo SOM, de Kohonen, combinou uma camada de entrada com uma camada competitiva treinada com aprendizado não-supervisionado. Esta camada possui, tipicamente, 2 (duas) dimensões, formando uma matriz de elementos processadores. O treinamento da Rede aparenta organizar um "mapa" topológico do relacionamento entre os padrões de entrada.

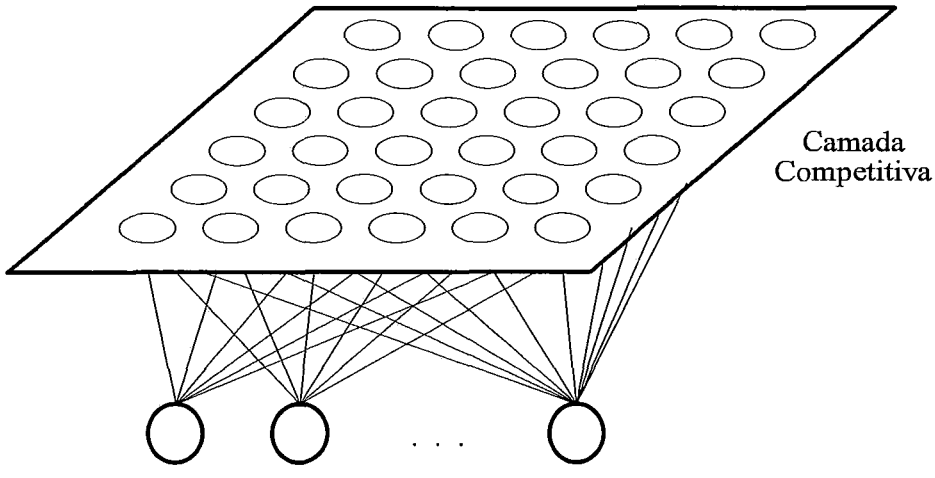


Figura IV.5 - SOM, de Kohonen

Cada um dos elementos desta camada soma suas entradas e depois estes competem entre si para determinar um único elemento processador vencedor. A unidade vencedora é a que possuir a menor distância entre o padrão desejado e o representado.

Ou seja, cada elemento **i** de entrada tem valor $\mathbf{e_i}$ e está conectado a um elemento **j** na camada de Kohonen através de um peso $\mathbf{w_{ji}}$. A distância **d** geralmente é dada pelo erro quadrático da entrada em relação ao peso *(Equação IV.4)*.

$$d = \sqrt{\sum_i (e_i - w_{ji})^2}$$

Equação IV.4

A unidade com a menor distância total ganha a competição. Os pesos são atualizados para esta unidade e as demais em sua vizinhança pelo coeficiente de aprendizado vezes a distância (e_i-w_{ij}). A determinação do tamanho da vizinhança é fundamental no desempenho da rede e, geralmente, possui um valor que decresce com o tempo.

IV.2.3 - "Adaptive Ressonance Theory" (ART)

Um dos principais objetivos da Inteligência Artificial é criar sistemas capazes de se adaptar em tempo real a mudanças inesperadas no ambiente de operação. *Grossberg (88)* apresenta um sistema com estas características e coloca o "Dilema da Estabilidade x Plasticidade":

"Como um sistema de aprendizado pode permanecer
adaptativo (plástico) em resposta a eventos significativos
e ainda assim permanecer
estável em resposta a eventos insignificantes?"

A capacidade de nos adaptarmos a novas situações sem um professor, baseados apenas em nossas experiências passadas, é denominada auto-organização. A idéia principal da Teoria da Ressonância Adaptativa (ART) é permitir que novos aprendizados sejam incorporados automaticamente à memória, não esquecendo, porém, de proteger memórias passadas (aprendidas) de serem sobrepostas pelo novo aprendizado.

As arquiteturas ART são redes neurais auto-organizáveis que proporcionam uma resposta estável em tempo real às seqüências arbitrárias de entradas. O estudo de

redes ART mostra que é possível dotar um modelo de aprendizado competitivo de um processo de aprendizado ao mesmo tempo estável e adaptativo.

Um dos maiores problemas dos paradigmas que utilizam aprendizado competitivo é que novos aprendizados podem prejudicar aprendizados antigos. Note que este problema, apesar de característico do aprendizado competitivo, também aplica-se a outros tipos de paradigmas.

Isto ocorre porque, ao treinarmos uma rede com um novo padrão, esta altera os pesos das conexões entre os elementos processadores. Nas redes ART, os padrões anteriores ficam "armazenados" em uma memória de longo termo (LTM). Isto é, existe uma outra rede interna responsável por manter os padrões anteriores.

Assim como os demais paradigmas de aprendizado competitivo, a rede ART tem uma limitada capacidade de aprendizado. Porém, diferentemente destes, ela não elimina padrões antigos em prol de novos padrões. Para entender este processo, supõe-se que um padrão e de entrada seja apresentado à rede e comparado aos padrões internos esperados k_i que representam os padrões já aprendidos. Deste modo, podem ocorrer dois eventos distintos:

1) existe um padrão interno k_i que se "aproxime" da entrada e.

2) não existe nenhum padrão interno que se aproxime da entrada.

Se não existir um padrão interno parecido, a rede considera esta entrada e como sendo um novo padrão. Caso a rede ainda não tenha esgotado sua capacidade de aprendizado, este padrão de entrada e é aprendido como um novo padrão interno k_i.

Quando o padrão de entrada se parece com um dos padrões internos k_i, estes padrões se fundem num processo de mútuo estímulo, entrando em ressonância. Somente, então, é feito o aprendizado deste novo padrão fundido k_i' no lugar de k_i. Como o aprendizado só ocorre na ressonância, este processo de aprendizado foi denominado "Ressonância Adaptativa".

Capítulo V

V - REPRESENTAÇÃO DE CONHECIMENTO EM REDES NEURAIS

"A coisa mais difícil para entender
é porque podemos entender seja lá o que for."

Albert Einstein

V.1 - Representação de Conhecimento

Inúmeras vezes, o conhecimento apresenta propriedades não desejáveis, tais como, ser volumoso, difícil de caracterizar com precisão e estar em constante mutação. Mesmo assim, ele deve ser representado de modo a capturar generalizações, ser compreensível, ser modificável, ser utilizável mesmo quando impreciso ou incompleto e ser utilizado de modo a superar seu próprio volume.

Não basta, portanto, ter conhecimento para ser inteligente. É preciso saber qual conhecimento disponível utilizar, onde buscar conhecimentos complementares e como utilizá-los eficientemente. A inteligência, portanto, está intimamente ligada à estrutura de armazenamento do conhecimento. Para que as representações sejam de algum interesse em relação ao mundo real, é preciso mapear os conhecimentos em suas respectivas representações e vice-versa *(Rich,88)*. Deste modo, uma representação eficiente do conhecimento é fator determinante do funcionamento da aquisição do conhecimento.

Embora os computadores estejam a cada dia mais rápidos, eles não são capazes de realizar de um modo rápido e eficiente tarefas que exigem a manipulação de uma enorme quantidade de conhecimento incompleto. O mais interessante é que, apesar de possuirmos uma imensa quantidade de informações guardadas em nosso cérebro,

podemos rapidamente selecionar as que nos interessam e deduzir o desejado. Em outras palavras, o que em métodos tradicionais de computação exigiria uma grande quantidade de busca, seleção e dedução de informações é rapidamente resolvido sem muito esforço pelo nosso cérebro. Quanto maior a base de conhecimento mais crítico se torna este problema, tornando necessário modelar novas formas de representação do conhecimento. As redes neurais apresentam-se, então, como uma opção muito atraente devido à sua plausibilidade neurológica, seu elevado grau de paralelismo e à sua capacidade de aprender através de exemplos.

V.2 - Representação de Conhecimento em Redes Neurais

O conhecimento pode ter diferentes origens que, presumivelmente, esgotam as fontes de conhecimento plausíveis *(Anderson,89)*:

- Nato: aquele conhecimento predeterminado (e.g. aquele com o qual se nasce).
- Racional: resultante da manipulação do conhecimento existente.
- Empírico: resultante da aquisição de novas experiências.

Em uma rede neural o conhecimento nato pode ser considerado como o paradigma escolhido, a topologia determinada para aquele problema e a seleção dos pesos iniciais (síntese) das conexões. Os conhecimentos racionais resultam da apresentação dos padrões de entrada e da posterior retirada dos padrões de saída da Rede Neural. E os empíricos são resultantes do treinamento da rede a partir da associação entre os padrões de entrada e saída do sistema.

De um modo geral, podemos afirmar que o conhecimento embutido nas Redes Neurais encontra-se nos estados atuais de saída de cada elemento processador (padrão de ativação) e nos valores de cada conexão entre eles (padrão de interconexão).

A inteligência se manifesta na capacidade de manipular o conhecimento de modo eficiente e eficaz. Representar o conhecimento significa criar um modelo para o mundo real. A interpretação de qualquer modelo envolve o mapeamento de um mundo representacional em alguma parte observável de um mundo real

(Smolensky,86).

Representar o conhecimento em Redes Neurais significa associar de algum modo conceitos externos aos seus padrões de atividade, embora estes não possuam significado algum. Ou seja, é possível atribuir aos estados de ativação dos elementos da rede responsabilidades específicas como representantes de objetos ou conceitos externos. Esta associação entre entidades externas (conceitos) e padrões de atividade pode ser denominada ***representação de conhecimento em redes neurais***.

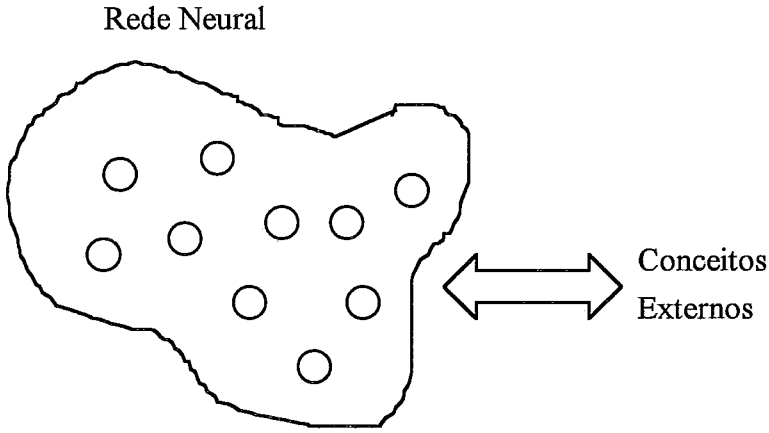


Figura V.1 - Representação de Conhecimento em Redes Neurais

A associação de um conceito externo a um padrão de atividade pode ser feita atribuindo a responsabilidade da representação de um conceito a um único elemento processador, sendo denominada ***representação localizada de conhecimento***, ou ao estado de ativação de um conjunto de elementos processadores, sendo denominada ***representação distribuída de conhecimento***.

A representação localizada é pouco plausível do ponto de vista biológico e tem como uma das principais desvantagens a baixa tolerância a falhas, porém, é de fácil implementação computacional e possui um elevado grau de inteligibilidade. A representação distribuída, por sua vez, é biologicamente mais plausível, mas possui alguns problemas como, por exemplo, no acesso à informação distribuída nos diversos elementos processadores e na representação simultânea de conceitos. Neste tipo de representação, um único elemento processador pode ser responsável pela

representação simultânea de mais de um conceito.

Alguns autores consideram a existência de uma ***representação parcialmente distribuída de conhecimento***, onde apenas um subconjunto dos neurônios da rede é responsável pelo conceito. Devemos ressaltar que estes mesmos autores consideram que na representação distribuída de conhecimento, a responsabilidade pela representação dos conceitos está distribuída através de todos os elementos da rede. Este último tipo de representação é denominado pelos demais autores como ***representação totalmente distribuída de conhecimento***, por explicitar a distribuição por todos os neurônios.

Pode-se fazer a associação entre conceitos e padrões de atividade de uma rede neural associando cada conceito ao padrão de toda a rede neural ou a apenas parte do padrão de atividade *(Carvalho,89)*. Do primeiro modo, estamos impondo a esta rede uma ***representação externa de conhecimento***. Do segundo modo, utilizamos parte do padrão de ativação para uma ***representação interna de conhecimento*** equivalente ou complementar à externa, possibilitando a criação de representações internas de um mundo exterior *(Figura V.2)*.

Apesar de ser unânime a premissa de que a inteligência se encontra nas conexões entre os elementos processadores, a representação de conhecimento é considerada apenas no padrão de ativação, esquecendo-se do padrão de interconexão.

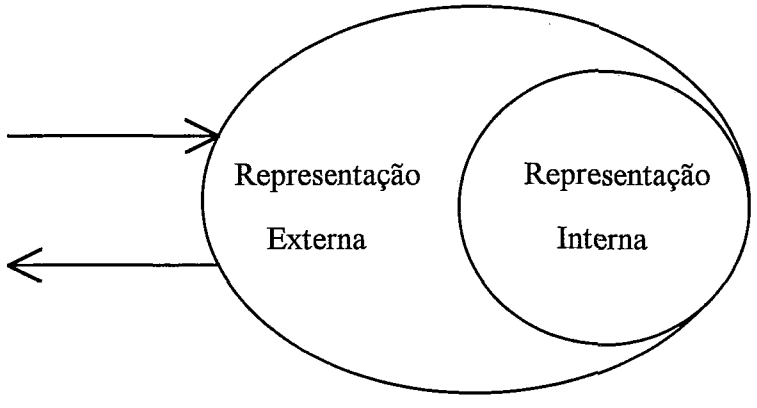


Figura V.2 - Representações Externa e Interna de Conhecimento (Carvalho,89)

Segundo *Levine (91)*, o campo da representação do conhecimento ainda carece de uma definição exata, porém, pode ser referido como a representação dos interrelacionamentos entre conceitos complexos. Ele sugere que Redes Neurais podem ser utilizadas para formar conceitos, construir relações entre estes conceitos e raciocinar inferencialmente sobre eles.

V.3 - Paradigma Simbólico-Conexionista

A Inteligência Artificial sempre pareceu seguir duas vertentes diametralmente opostas: o paradigma simbolista versus o paradigma conexionista. Atualmente, pesquisadores de ambas as áreas já admitem que sistemas híbridos compostos por ambos os paradigmas têm um desempenho melhor e uma aplicabilidade mais diversificada. Deste modo, o que antes parecia apontar para um confronto, agora sinaliza uma síntese, onde as principais características de cada paradigma sejam combinadas.

A construção de sistemas híbridos conjugando as técnicas simbólicas e conexionistas tem despertado grande interesse da comunidade internacional de Inteligência Artificial. Uma linha de pesquisa que tenta combinar o processamento simbólico convencional com as técnicas conexionistas é denominada ***Processamento Simbólico-Conexionista*** *(Ayrosa,92)* ou ***Paradigma Simbólico-Conexionista***.

Dentro desta linha, existe um certo tipo de sistema conexionista que utiliza-se somente da representação localizada de conhecimento e que, por isso, não é considerado por muitos pesquisadores como Redes Neurais.

Gallant (88) propôs um modelo denominado Linear Discriminant Networks (LDN), no qual a topologia da rede era inicialmente determinada pelo especialista e, depois, era modificada através da inclusão de nós e conexões com pesos fixos.

Kosko desenvolveu um modelo denominado Fuzzy Cognitive Map, que mistura sistemas conexionistas, fuzzy e regras de produção. Neste modelo, os elementos

processadores representam conceitos e as conexões (direcionadas) representam uma relação causal entre estes.

Rocha e *Machado*, dois brasileiros, desenvolveram um modelo denominado Combinatorial Neural Model (CNM), que utiliza uma camada intermediária com todas as combinações possíveis e depois "poda" as não utilizadas.

Na mesma época da (re)apresentação por Rumelhart do modelo Back-Propagation, *Smolensky (86)* oferecia um paradigma alternativo ao simbólico, denominado paradigma sub-simbolista, no qual utilizava correlações estatísticas em sistemas dinâmicos, ao invés da manipulação de símbolos. O ***processamento simbólico-conexionista*** procura representar distribuidamente símbolos e estruturas de símbolos em Redes Neurais.

Smolensky (86) apresenta a "Harmony Theory", que é um modelo matemático para sistemas dinâmicos capazes de realizar tarefas cognitivas. *Touretsky (90)* utiliza uma representação parcialmente distribuída em seu modelo BoltzCONS para criar e manipular objetos como listas e árvores. *Pollack (90)* mostra um modelo, denominado RAAM, que cria representações parcialmente distribuídas compactas, transformando objetos em vetores numéricos e vice-versa. *Smolensky (90)* desenvolve um método genérico para esta conversão em sua Representação por Produto Tensorial, uma representação totalmente distribuída.

Carvalho (89) apresentou em sua tese de doutorado, um método de representação de conhecimento em redes de Hopfield a partir de critérios de síntese. *Martino (90)* utilizou este método para representação do conhecimento implícito em redes neurais. A representação hierárquica do conhecimento em classes e os mecanismos de herança para a aferição dos conhecimentos implícitos dos problemas de classificação assemelha-se em muito ao modo pelo qual vemos o mundo.

Nos próximos itens mostramos detalhadamente um modelo de representação localizada de conhecimento (*Rocha* e *Machado*) e uma pesquisa em representação distribuída de conhecimento (*Carvalho*). Nesta seleção optamos por valorizar pesquisadores brasileiros que, apesar das dificuldades encontradas, conseguem

produzir trabalhos de nível internacional.

V.3.1 - "Combinatorial Neural Model"

Machado (88,90) e Rocha desenvolveram um método para extrair conhecimento de múltiplos especialistas, criando um grafo AND/OR direcionado acíclico denominado "Knowledge Graph", que podia ser utilizado como a representação formal do conhecimento desses especialistas.

Este grafo *(Figura V.2)* era composto por 3 (três) tipos distintos de nós. Os nós de evidência situavam-se no nível inferior do grafo e representavam os sintomas, sinais e resultados de testes (nos casos de diagnósticos médicos). No nível superior localizavam-se os nós de hipóteses, que representavam os possíveis diagnósticos a serem considerados. Entre eles, podiam ser encontrados os nós intermediários, representando as diferentes conjunções de evidência utilizadas pelos especialistas na obtenção das hipóteses a partir das evidências.

A partir deste trabalho, *Machado (89)* e Rocha desenvolveram um modelo de Redes Neurais para a resolução de problemas de classificação denominado ***"Combinatorial Neural Model"***.

Este modelo compõe-se de 2 (dois) tipos de elementos processadores, AND e OR, que podem assumir valores entre 0 e 1, através do cálculo dos valores mínimo e máximo, respectivamente, dentre suas entradas.

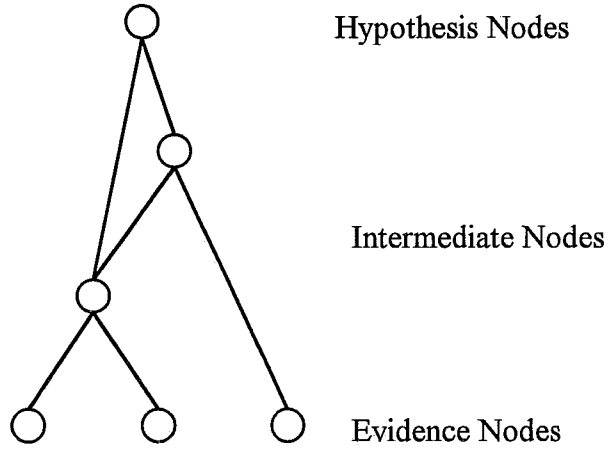


Figura V.2 - "Knowledge Graph" (Machado,88)

Os elementos processadores distribuem-se em camadas não-realimentadas (feed-foward). Embora o modelo permita a utilização de um número qualquer de camadas, é possível produzir sempre um modelo equivalente de 3 (três) camadas: uma camada de entrada, uma camada Combinatória e uma camada de saída.

Cada elemento na camada de entrada representa uma evidência na forma de uma tripla OAV (Objeto-Atributo-Valor), por exemplo, "a máquina está com o funcionamento em síncrono" (máquina-funcionamento-síncrono).

Cada elemento na camada de saída representa uma hipótese de solução do problema. Estes elementos são do tipo OR, de modo a permitir ser resultante de um ou mais elementos da camada combinatória ou de entrada.

Uma das características fundamentais do raciocínio de especialistas humanos é a associação de evidências em agrupamentos de informações. Isto é feito frequentemente utilizando o conectivo lógico E ("AND"). Baseado neste fato, eles propuseram a criação de uma camada escondida denominada ***Camada Combinatória***, que representaria as diferentes combinações dos dados de entrada.

Em primeira instância, pode-se criar uma camada Combinatória contendo TODAS as possíveis combinações de evidências de entrada, criando o que denominaram Versão Completa (ou Versão Primitiva) do modelo. Dois exemplos, para um problema com 2 evidências e 1 hipótese, e para um problema com três evidências e 1 hipótese, podem ser vistos na *Figura V.3*.

A versão completa de um modelo CNM incluiria todas as combinações de N entradas, desde a combinação destas duas a duas até a combinação de todas. Ou seja, $C(N,2)+...+C(N,N)$, o que exige 2^N-N-1 elementos processadores na camada intermediária. Isto causa uma explosão combinacional que tornaria inviável a utilização prática do modelo.

Para restringir esta explosão espacial, comparável à explosão temporal dos modelos simbolistas, eles criaram o modelo CNM de Ordem m, que limita as combinações da camada Combinacional. Este modelo possui em sua camada

intermediária apenas combinações iguais ou menores que **m**, tipicamente representado pelo número mágico 7 mais ou menos 2 de Miller.

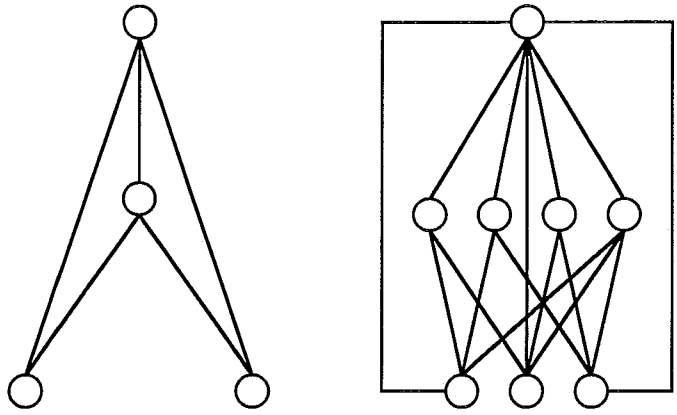


Figura V.3 - versões completas do CNM

O algoritmo de aprendizado parte do modelo de ordem m e, através de mecanismos de punição e recompensa determina novos valores para as conexões. A rede resultante, para se tornar operacional, ainda precisa passar por um processo de "poda", de modo a remover as combinações pouco significativas.

Machado (91) e Rocha criaram uma ferramenta *(Machado,92c)*, denominada Neural Expert Tool (NEXT), para facilitar o desenvolvimento de aplicações de Sistemas Especialistas Conexionistas *(Machado,92d)*. Esta ferramenta já possuía alguns conceitos de Algoritmos Genéticos que foram incorporados por *Denis (91)* em um modelo evolutivo de Rede Neural *(Machado,92a)*. *Rocha (92)* utilizou um modelo composto por 3 (três) redes evolutivas organizadas hierarquicamente para extrair conhecimento de banco de dados em Linguagem Natural. *Machado (92b)* e Ferlin embutiram no modelo um mecanismo de aprendizado incremental, de modo a preservar o conhecimento anterior durante um novo treinamento. *Machado (93)* e Rocha mostram que esté sistema pode utilizar elementos processadores "fuzzy" e explicam com detalhes os procedimentos de inferência e explanação desse Sistema Especialista.

V.3.2 - Representação de Conhecimento em Redes de Hopfield

Neste item apresentamos o estudo realizado por *Carvalho (89)* sobre a representação distribuída do conhecimento em Redes de Hopfield, utilizando métodos de síntese.

Em uma rede neuronal de Hopfield, cada padrão de atividade corresponde ao valor de uma função característica denominada energia computacional. *Hopfield (82)* demonstrou que seu modelo evolui sempre de modo a diminuir o valor desta função. Assim, podemos associar ao conjunto dos possíveis padrões de atividade de uma rede de Hopfield, uma ***superfície de energia***. Ao iniciarmos a rede em um padrão de atividade qualquer, esta evolui até um dos mínimos locais desta superfície, que denominamos padrão estável. Em outras palavras, uma rede neural de Hopfield, partindo de um padrão de atividade qualquer, convergirá para um padrão estável *(Hopfield,86)*.

Se associarmos cada padrão estável a uma informação armazenada na rede, ao excitarmos a rede com uma informação qualquer (padrão de atividade desconhecido), esta convergirá para uma informação armazenada (padrão estável). Assim, podemos recuperar uma informação total a partir de uma informação parcial, criando uma memória de acesso pelo conteúdo. Esta é uma característica da rede neuronal de Hopfield que é típica da memória humana. O ser humano é capaz de adquirir, armazenar e manipular novas informações capazes de modificar o seu comportamento, em um processo denominado ***aprendizado***. A idéia do método de ***síntese*** é gerar de uma única vez uma rede neuronal competente na execução das associações de padrões de atividade desejados. Em uma rede de Hopfield, significa determinar de forma completa o padrão de conexão W de uma rede neuronal capaz de associar o estímulo e à resposta s.

Mas, o processo de síntese em uma rede neuronal de Hopfield nada mais é do que conformar sua superfície de energia de modo que possua mínimos nos padrões de atividade desejados. Definimos conjunto estável $S(W)$ como o conjunto de todos os pontos de mínimo da superfície de energia de uma rede neuronal com padrão de

conexão W. Assim, sintetizar uma rede neuronal de Hopfield significa determinar W de modo que os padrões de atividades desejados pertençam à $S(W)$.

Normalmente existem, para cada problema, vários padrões de conexão W possíveis. Ao conjunto dos padrões de conexão possíveis denominamos sínteses viáveis. Como existem diversas soluções, desejamos chegar a uma solução ótima utilizando critérios de síntese, que variam de acordo com a utilização desejada.

Para facilitar a síntese de redes para representação de conhecimento, *Carvalho (89)* sugere que apenas parte do padrão de atividade da rede represente o conceito, liberando os demais neurônios para assumir o estado de ativação mais adequado ao método de síntese, o que o tornaria um processo de criação de representações internas de um mundo exterior.

A rede neuronal de Hopfield baseia-se em neurônios lógicos, que podem estar ativados ou desativados em um determinado padrão de atividade. Dizemos que dois padrões de atividade v e w são independentes ($v \perp w$, ou $v \diamond w$) caso nenhum neurônio da rede esteja ativado simultaneamente nas duas representações (Figura V.4a). Assim, padrões independentes são ideais para representar conceitos totalmente diferentes ou opostos. Porém, caso os conceitos apresentem alguma semelhança ou relacionamento, alguns neurônios devem estar excitados em ambas as representações, sendo os conceitos ditos dependentes (Figura V.4b).

Além disso, podemos dizer que um determinado conceito (com um padrão v) é maior do que outro conceito (com padrão w) se v possuir todos os neurônios ativos em w (Figura V.4c). A soma de dois conceitos é a ativação simultânea de seus neurônios (Figura V.4d), e a subtração de um conceito w de um conceito v, pode ser representada pela ativação dos neurônios ativos em v que não estão ativos em w (Figura V.4e). A relação de maioridade transmite a idéia que um conceito com padrão v possui algo a mais que um outro conceito w menor que v. Além disso, sempre que v estiver representado na rede, w também estará. Podemos então abstrair para uma generalização de conceitos, onde w representaria uma determinada classe de conceitos e v seria uma instância desta classe.

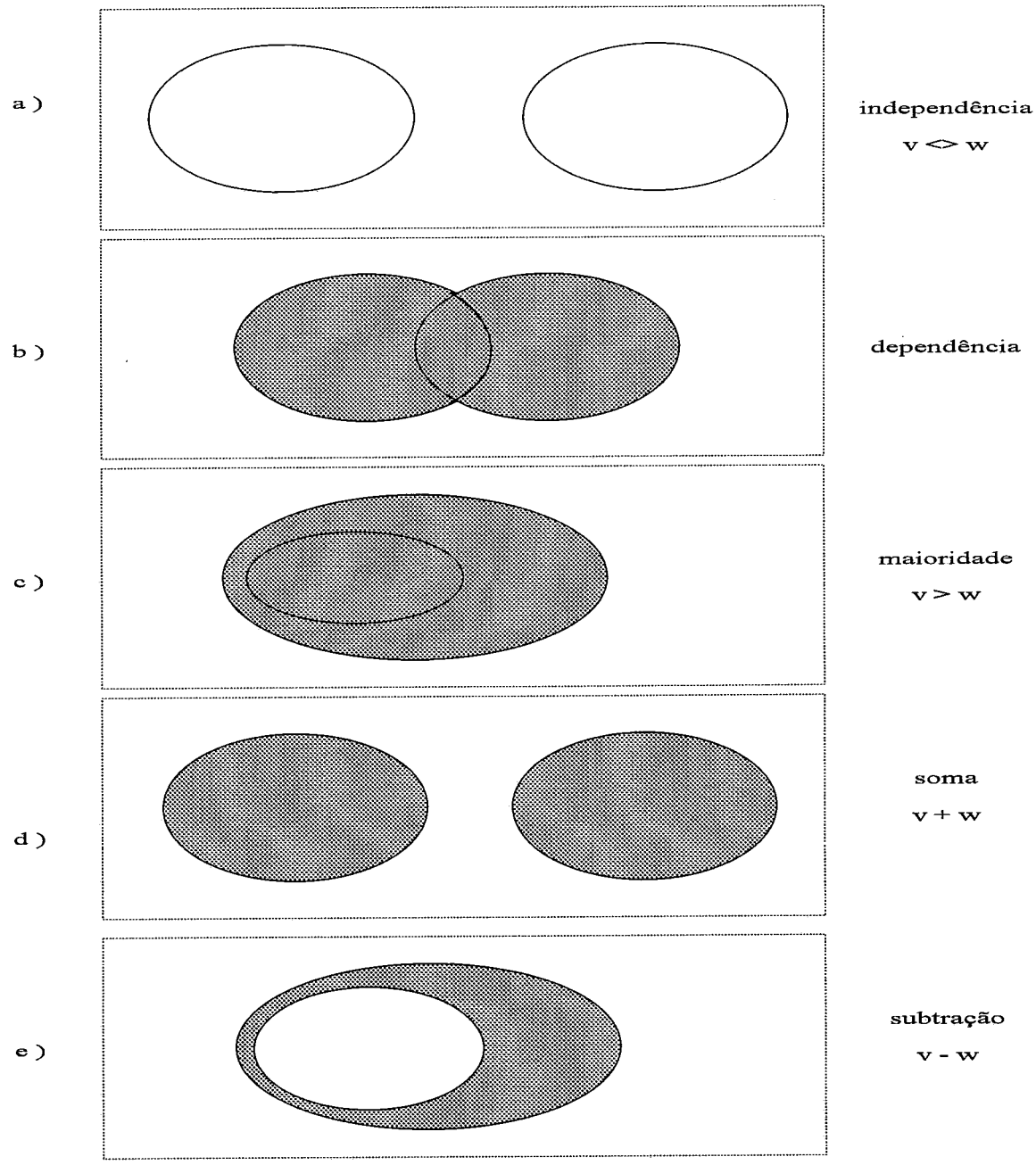


Figura V.4 - Relações entre Conceitos

Por exemplo, considere a classe CACHORRO, que possui diversas instâncias, tais como DOBERMAN, DÁLMATA e PASTOR ALEMÃO. Cada uma destas instâncias necessita de mais informações que sua respectiva classe. DÁLMATA é um CACHORRO que é todo pintado e carinhoso com crianças. Assim, para representar DÁLMATA, temos que ativar todos os neurônios correspondentes a CACHORRO e

mais alguns *(Figura V.5)*.

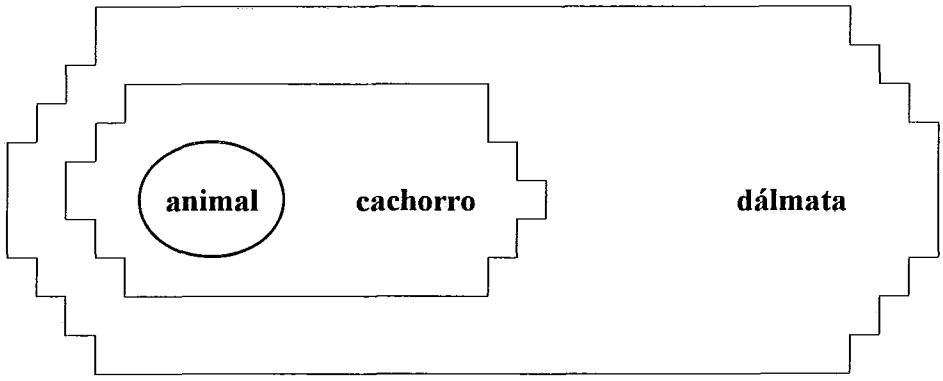


Figura V.5 - Exemplo de Hierarquia

Do mesmo modo, considere a classe CACHORRO pertencendo à classe ANIMAL que, por sua vez, possui outras sub-classes como LEÃO, BOI e TUBARÃO. CACHORRO é um ANIMAL doméstico que pode ser usado para vigiar a casa. Assim, temos que ativar todos os neurônios correspondentes a ANIMAL e mais alguns.

Em outras palavras, através da relação de maioridade, podemos obter uma generalização de conceitos, criando uma hierarquia de classes de conceitos, o que é natural para o ser humano. O processo de síntese desenvolvido por *Carvalho (89)* permite criar hierarquias semelhantes em Redes de Hopfield.

V.3.3 - Representação de Conhecimento Implícito em Redes de Hopfield

Utilizando a teoria desenvolvida por *Carvalho (89)* em sua tese de doutorado, *Martino (90)* mostrou em uma monografia como representar o conhecimento ímplicito em Redes Neurais.

A mente humana pode guardar uma imensa quantidade e variedade de informações e, apesar disso, acessá-las com grande velocidade e precisão. Algumas das informações que dispomos estão guardadas explicitamente na memória.

Ex.:

GaviãoReal ***É UM*** *Gavião*

Porém, grande parte das informações que precisamos não estão guardadas explicitamente, e têm de ser deduzidas a partir de outras informações.

Ex.

GaviãoReal ***É UM*** *Gavião*

Gavião ***É UM*** *Pássaro*

Assim podemos deduzir que:

GaviãoReal ***É UM*** *Pássaro*

Ou seja, o conhecimento desejado estava armazenado implicitamente, podendo ser "evocado" a qualquer momento. O mais interessante é que, apesar de possuirmos uma imensa quantidade de informações guardadas, podemos rapidamente selecionar as que nos interessam e deduzir o desejado. Em outras palavras, o que em métodos tradicionais de computação exigiria uma grande quantidade de busca, seleção e dedução de informações é rapidamente resolvido pelo nosso cérebro, e sem muito esforço. Note que quanto maior a base de conhecimento mais crítico se torna este problema. Deste modo, para resolver problemas de classificação é necessário algum mecanismo de representação do conhecimento implícito.

Herança é um mecanismo clássico em técnicas de Inteligência Artificial que permite que informações sobre conceitos possam ser utilizadas por especializações e instancializações deste conceito. No entanto, existem outros mecanismos que tornam-se necessários na maioria dos sistemas de classificação, tais como Múltiplas Heranças e Exceções.

a) Herança:

O conhecimento implícito depende, em muito, de modo pelo qual entendemos o mundo que nos cerca. *Mellor(88)* afirma que a mente humana percebe o mundo

como um conjunto de objetos e os relacionamentos entre eles:

Ex.

Carro ***É UM*** *Veículo*

Porém, este modelo compreende também a generalização destes objetos em uma classe de objetos.

Ex.

Chevette ***É UM*** *Carro*

Fusca ***É UM*** *Carro*

Escort ***É UM*** *Carro*

Assim, todos os objetos de uma determinada classe devem "HERDAR" todas as propriedades desta classe.

Por exemplo, se

Escort ***É UM*** *Carro*

e,

Carro ***É UM*** *Veículo*

é possível deduzir que:

Escort ***É UM*** *Veículo*

Esta informação não está explicitamente armazenada, porém está implicitamente contida, devido ao conceito de classes. Ou seja, o mecanismo de herança permite que a informação seja armazenada do modo mais genérico possível (em classes) e que seja herdada por todas as especializações e instancializações desta classe. A informação herdada não fica, normalmente, explicitamente armazenada, sendo inferida através do mecanismo de herança das classes à que pertence. A hierarquia de classes e o mecanismo de herança permitem inferir os conhecimentos implícitos já mencionados. Porém, existem outros tipos de conhecimento implícito.

b) Heranças Múltiplas:

Muitas vezes associamos um objeto a mais do que uma classe. Por exemplo, normalmente diríamos que GAVIÃO REAL é um PÁSSARO. Porém, se estivéssemos falando sobre ecologia, também poderíamos dizer que GAVIÃO REAL é uma ESPÉCIE EM EXTINÇÃO. Deste modo, uma determinada classe pode ser sub-classe de duas ou mais outras classes distintas, herdando assim as propriedades de ambas.

Por exemplo, temos que:

Pássaro ***TEM*** *DomDeVoar*

EspécieEmExtinção ***TEM*** *ProteçãoDaLei*

E:

GaviãoReal ***É UM*** *Pássaro*

GaviãoReal ***É UM*** *EspécieEmExtinção*

Assim:

GaviãoReal ***TEM*** *DomDeVoar*

GaviãoReal ***TEM*** *ProteçãoDaLei*

c) Exceções:

Muitas vezes, atribuimos uma determinada propriedade a uma classe mesmo que nem todas as suas instâncias possuam esta propriedade.

Por exemplo, sabemos que nem todas as aves voam, porém, como quase todas as aves voam, podemos dizer que uma AVE tem o DOM DE VOAR. Esta é a informação que armazenamos, independente de nem todas as aves voarem. As exceções, as aves que não voam, são armazenadas como uma informação independente.

Deste modo, uma determinada classe pode herdar todas as propriedades de outra exceto algumas em particular.

Ex.

Pinguim **É UM** *Ave*

Ave **TEM** *DomDeVoar*

Porém:

Pinguim **NÃO TEM** *DomDeVoar*

Por este ponto de vista os problemas de classificação podem ser considerados como sistemas de múltiplas especializações com exceções. Smolensky denomina este tipo de sistemas como ***Múltipla Herança com Exceções***. Eles também são conhecidos como representação do conhecimento implícito, e sistemas de raciocínio não-monotônico.

A representação hierárquica das redes neuronais de Hopfield já embutem o conhecimento implícito. Porém, é preciso analisar separadamente os casos particulares.

a) Múltiplas Heranças:

É uma expansão trivial do conceito de hierarquia em redes neuronais de Hopfield. Neste caso, a instância representada deve herdar as propriedades das classes à que pertence.

No exemplo visto:

GaviãoReal ***É UM*** *Pássaro*

GaviãoReal ***É UM*** *EspécieEmExtinção*

Assim, GaviãoReal deve herdar as propriedades de Pássaro e de EspécieEmExtinção. Ou seja, deve ativar todos os neurônios das duas classes *(Figura V.6)*. Em outras palavras, se o conceito Pássaro corresponde a um padrão de atividade v, e o conceito EspécieEmExtinção a um padrão w, então o conceito GaviãoReal deve corresponder a um padrão u, tal que ($u > v$) e ($u > w$).

Simbolicamente:

Pássaro ---> *v*

EspécieEmExtinção ---> *w*

GaviãoReal ---> *u* , onde (u >v) e (u > w)

Podemos ainda dizer que para representar o conceito GAVIÃO REAL, temos que ativar todos os neurônios ativos em PÁSSARO mais todos os ativos em ESPÉCIE EM EXTINÇÃO. Assim:

GaviãoReal ----------> *u* , *onde* *u* > *v*+*w*

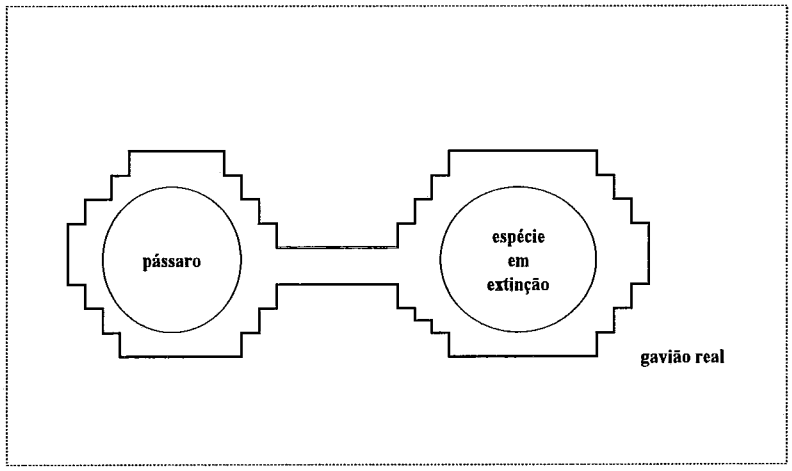


Figura V.6 - Exemplo de Múltiplas Heranças

b) Exceções:

É um conceito que requer um pouco mais de elaboração. No exemplo visto:

Ave ***TEM*** *DomDeVoar*

Pinguim ***É UM*** *Ave*

E,

Pinguim ***NÃO TEM*** *DomDeVoar*

Se considerarmos que AVE tem DOM DE VOAR e que esta propriedade dela

está representada pela ativação de alguns de seus neurônios, temos:

DomDeVoar ----> *r*

Ave ----> *s*, *onde* *s* > *r*

Agora, PINGUIM é uma AVE, e assim deve ativar todos seus neurônios, inclusive aqueles que indicam que AVE tem DOM DE VOAR.

Pinguim --------> *t*, *onde* *t* > *s*

Porém, a condição de exceção nos diz que PINGUIM NÃO TEM DOM DE VOAR. Isto sugere que os dois conceitos não são compatíveis. Assim, é preciso que dois conceitos independentes cooperem entre si para gerar um terceiro conceito. Carvalho mostra no teorema V.10 de sua tese de doutorado que isto é possível para uma rede neuronal de Hopfield.

Representaremos doravante este conceito como:

c1 + c2 => c3

Equação V.1

Porém, este terceiro conceito deve representar que, apesar de PINGUIM ser uma AVE, ele não possui o DOM DE VOAR. Para tanto, ele deve ativar todos os neurônios de AVE, exceto aqueles que indicam que esta possui o DOM DE VOAR.

Simbolicamente,

PinguimNaoTemDomDeVoar ---> *v*, onde v <> t

(e também v <> r)

e v + t => t - r

Observe que a condição de exceção, sendo satisfeita, implica em uma evolução da rede. Assim podemos concluir que é possível utilizar uma rede neuronal para a representação do conhecimento implícito.

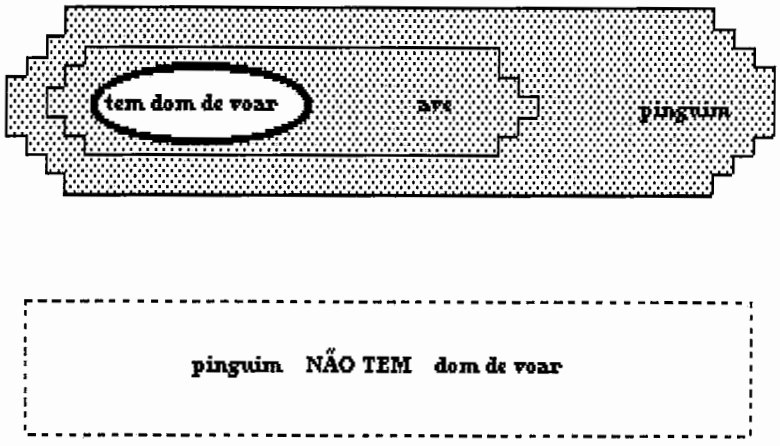


Figura V.7 - Exemplo de Exceção

Capítulo VI

VI - PROBLEMAS REAIS E DADOS INCOMPLETOS

"Data! Data! Data!", he cried impatiently, "I can't make bricks without clay."

Sherlock Holmes (Conan Doyle)

VI.1 - Aprendizado a partir de Exemplos em Redes Neurais

Aprendizado é uma transformação qualitativa no sentido de aumentar a eficiência de alguma capacidade do indivíduo *(Carvalho,89)*. Preferimos definir aprendizado como a capacidade que o ser humano possui de adquirir, armazenar e manipular o conhecimento.

Os paradigmas de aprendizado mais conhecidos são o aprendizado dedutivo e o aprendizado indutivo. O aprendizado dedutivo manipula o conhecimento existente de modo a inferir (deduzir) conclusões a partir de observações do mundo real e dos fatos conhecidos. Já o aprendizado indutivo cria novos conhecimentos (hipóteses) que permitem explicar as observações do mundo real. Ambos os paradigmas são utilizados no aprendizado humano. Alguns autores, como *Carbonell (89)*, dividem em quatro os possíveis paradigmas de aprendizado: indutivo, analítico, genético e conexionista. Porém, quaisquer outros paradigmas envolvem formas de aprendizado indutivo e dedutivo.

O aprendizado dedutivo exige que todo o conhecimento necessário esteja disponível *(Rocha,92)*. As conclusões obtidas são resultado da manipulação do conhecimento já existente, ou seja, nenhum novo conhecimento foi acrescentado. Na verdade, o aprendizado dedutivo apenas torna explícito um conhecimento que já existia mas estava implícito, enquanto o aprendizado indutivo permite criar novos

conhecimentos a partir de observações do mundo real. Ou seja, o aprendizado indutivo permite a aquisição de novos conhecimentos.

O aprendizado através de exemplos é uma forma de aprendizado indutivo. A partir de exemplos fornecidos é preciso generalizá-los induzindo regras (ou teorias) que expliquem os exemplos fornecidos. Estes exemplos podem ser obtidos através de registros anteriores, medidas atuais, bancos de dados, ou mesmo diretamente com os especialistas. A partir de exemplos diversas tarefas podem ser aprendidas, como diagnose de falhas, predição de carga e outros. *Bratko (93)* afirma que uma aplicação óbvia deste aprendizado é auxiliar na resolução do gargalo da aquisição do conhecimento. Dentre as muitas formas de aprendizado, o aprendizado através de exemplos é a mais comum e a mais bem entendida.

Muitas formalizações do aprendizado através de exemplos foram feitas, inclusive a de *Bratko (93)* que diz que o problema de aprender conceitos a partir de exemplos pode ser formalizado do seguinte modo: considere **U** como o conjunto universal de objetos; um conceito **C** pode ser formalizado como um subconjunto de objetos em **U**; aprender um conceito **C** significa reconhecer se um objeto **O** pertence ou não à **C**. Matematicamente, um exemplo pode ser representado pela tupla de conjuntos **{E, S}**, onde **E** representa um conjunto de conceitos disponíveis (conceitos de entrada) e **S** um conjunto de conceitos determinados a partir de **E** (conceitos de saída). Tanto **E** como **S** são subconjuntos de **C**. Dado um conjunto de exemplos **X**, denominado conjunto de treinamento, deve-se determinar um conjunto de regras **R** que para todo **E** pertencente a **X**, determine corretamente o **S** correspondente. Podemos dizer que existe uma relação **R** que mapeia **E** em **S**.

Na resolução de problemas de classificação, buscamos algo que resolva uma relação não-linear entre um conjunto de observações e outro de possíveis classes *(Capítulo IV)*. Assim, um problema de classificação qualquer pode ser definido como um mapeamento não-linear de N observações em M possíveis classes. Porém, ao utilizarmos uma única rede neural na resolução de um problema de classificação qualquer, estamos tentando estabelecer uma relação não-linear entre TODAS as

classes e TODAS as observações, o que normalmente não é verdade. Na maioria das aplicações existentes apenas um limitado conjunto das observações é responsável pela determinação de outro conjunto, também limitado, de classes. Isto representa um problema seríssimo se considerarmos que uma rede neural é treinada e utilizada com exemplos completos de padrões de entrada e saída de dados. Um especialista não é capaz de fornecer os exemplos de forma completa para este treinamento sendo, assim, necessário um esforço incomensurável para a construção destes exemplos.

A ausência de dados é um problema que sempre surge nas implementações reais e tem despertado o interesse dos conexionistas, como pode ser comprovado no *Anexo A*. Nele, reproduzimos uma discussão na InterNet, provocada por uma simples observação sobre a ausência de dados, que teve a participação inusitada de mais de 20 (vinte) conexionistas.

Para que possamos utilizar os mecanismos de aprendizado existentes, é necessário que exista, nos padrões de entrada e saída, um meio de representarmos que um determinado conceito não se encontra disponível ou que outro não pode ser determinado a partir daqueles disponíveis. Os dois itens a seguir mostram mecanismos para tentar representar a ausência de dados nos padrões de entrada e saída, e permitir a utilização de exemplos incompletos.

VI.2 - Dados de Entrada e Saída em Redes Neurais

Para utilizar Redes Neurais na resolução de qualquer problema (independentemente de ser ou não um problema de classificação) este deve, inevitavelmente, ser representado em forma de dados utilizados como entrada e saída da rede neural. Alguns autores afirmam que um dos maiores problemas em Redes Neurais está em como representar os dados que devem ser apresentados à rede (ou seja, as entradas e suas respectivas saídas). No caso dos problemas de classificação não existe um consenso na forma em que estes dados devem ser apresentados pelos mais diversos paradigmas.

Os dados, por sua vez, podem ser binários, ternários, discretos, numéricos ou normalizados. Assim, um mesmo conceito pode ser representado como um dado binário ou como um dado numérico. Ou seja, de acordo com a forma de representação do conceito em dados, pode-se obter diferentes representações do mesmo conceito. Podemos então constatar que, independentemente do paradigma de Redes Neurais utilizado, a representação dos dados a serem apresentados como exemplos à rede determina a capacidade de aprendizado da mesma. Deve ficar claro que não nos referimos à ordem ou mesmo ao número de repetições, mas sim à codificação dos dados, ou seja, nos referimos à estruturação do conhecimento. Deste modo, antes de decidir-se por um paradigma deve-se, primeiramente, determinar a forma de representação adequada para os conceitos desejados. Este problema deve ser resolvido antes de definir-se qual o paradigma de Redes Neurais a ser utilizado.

Normalmente, uma Rede Neural possui pelo menos duas camadas de Elementos Processadores amplamente interconectados, sendo uma camada para a entrada e outra para a saída de dados. Tipicamente, pode existir uma ou mais camadas intermediárias entre elas. Para que a rede possa aprender através dos exemplos fornecidos, os dados devem ser representados de modo a poderem ser utilizados como entrada e saída desta. Deve-se ressaltar que em todas as implementações de Redes Neurais por nós conhecidas, os dados de entrada são apresentados simultaneamente, de modo a compor um padrão de entrada e retirados também simultaneamente, de modo a compor um padrão de saída. Os dados normalmente são apresentados na camada de entrada, se propagam através das camadas intermediárias e são retirados na camada de saída.

Como já visto, quando os dados de saída são diferentes dos dados de entrada, a rede é denominada hetero-associativa, caso contrário, é denominada auto-associativa. Embora estejamos considerando apenas redes hetero-associativas, os conceitos apresentados também são válidos para redes auto-associativas.

A especificação dos dados para uma Rede Neural é tão significativa que *Stein (92)* afirma que a decisão mais importante em uma modelagem é escolher o conteúdo e

as fontes de dados para o modelo. O modelista precisa determinar quais são as variáveis mais usadas na prática e de que modo são utilizadas.

Segundo *Bratko (93)* os exemplos para o aprendizado são descritos em termos de atributos, que podem ser simbólicos ou numéricos. Um atributo simbólico tem um conjunto não-ordenado de valores, enquanto um numérico possui um conjunto ordenado de valores.

Expandimos um pouco as idéias acima, de modo a incluir algumas das representações de dados mais usuais:

a) Dados Binários:

São dados que podem ser representados por 2 (dois) valores. Normalmente, utiliza-se 0 e 1 (como na álgebra de Boole), mas também podem ser utilizados conjuntos de 2 elementos quaisquer (e.g. {FALSO, VERDADEIRO}, como na lógica). Esta representação é trivial tanto nas simulações em computador quanto nas implementações em hardware. Este tipo de dado é muito utilizado para representar estados antagônicos, como em um disjuntor {ABERTO, FECHADO}.

b) Dados Ternários:

São dados que podem ser representados por 3 (três) valores. Normalmente, os valores utilizados são -1, 0 e 1, mas também podem ser utilizados um conjunto de 3 elementos (e.g. {FALSO, DESCONHECIDO, VERDADEIRO} como na lógica de 3 estados). Pode ser utilizado, por exemplo, para representar o estado de uma seccionadora {ABERTO, TRÂNSITO, FECHADO}.

c) Dados N-ários (Discretos):

São dados que podem ser representados por um número finito de valores. Normalmente, são simulados em computador como valores inteiros a partir de 0, ou por um conjunto com um número finito de elementos (e.g. {OUTONO, INVERNO, PRIMAVERA, VERÃO} ou {SEGUNDA, TERÇA, QUARTA, QUINTA, SEXTA,

SÁBADO, DOMINGO}). Pode ser utilizado, por exemplo, para representar estados de funcionamento {NORMAL, ALARME, EMERGÊNCIA, RESTAURAÇÃO}.

d) Dados Contínuos:

São dados que podem ser representados por valores inteiros ou reais. Como as redes neurais são simuladas em computadores ou executadas em um "hardware" específico, os valores têm um limite superior e um limite inferior. Além disso, as próprias variáveis de entrada e saída possuem um mínimo e um máximo, estando limitadas entre estes. Muitas vezes se quer representar estes valores normalizados no intervalo de 0 a 1, de modo que 0 sempre represente o valor mínimo e 1 o máximo. São utilizados para representar valores de TENSÕES, CORRENTES, TEMPERATURAS e outras medidas analógicas.

Resumindo, as Redes Neurais são treinadas a partir de exemplos de padrões de entrada e saída de dados. Os dados dos exemplos que podem ser apresentados às redes neurais podem ter domínios que variam *(Tabela VI.1)*.

Representação	Domínio	Descrição
binária	{0,1}	base 2 ou booleana {F,T}
ternária	{-1,0,1}	base 3 ou 3-State {F,U,T}
discreta	{0,1,2,...,n}	valores contínuos em $\aleph^+$ (base N)
numérica	$[-\infty,+\infty]$	valores contínuos em $\Re$
limitada	[mín.,máx.]	valores contínuos até os limites
normalizada	[0,1]	valores contínuos no intervalo

Tabela VI.1 - Representação de Dados em Redes Neurais

VI.3 - Representação de Conceitos em Redes Neurais

Qualquer sistema utilizando técnicas de Inteligência Artificial precisa, de algum modo, representar o conhecimento envolvido. No caso de Redes Neurais, é fundamental determinar pelo menos a forma em que os exemplos devem ser apresentados à rede. Isto significa criar um modelo do ambiente, o que não é tarefa

trivial (nem mesmo para um especialista). Algumas formas de representar este ambiente estão descritas abaixo com o intuito de mostrar a importância da representação do conhecimento nos problemas de classificação.

VI.3.1 - Conceitos Simples:

Quando o conceito (a característica ou a classe) que desejamos representar pode ser expresso por uma frase, podemos dizer que este é um conceito simples. Uma outra forma de visualizar conceitos simples é considerá-los como uma tripla O-A-V (Objeto-Atributo-Valor).

a) Como um dado binário:

Pode-se facilmente representar um conceito simples como um dado binário. Por exemplo, quando deseja-se representar que um disjuntor está aberto, basta criar uma frase que resuma este conceito tal como "disjuntor 742 aberto". Este fato pode ser representado com domínio {0,1} ou {falso, verdadeiro}.

Usualmente, utiliza-se o valor 0 para representar que o fato é falso e o valor 1 para representar que é verdadeiro. Assim, o fato só terá valor 1 (ou verdadeiro) quando o disjuntor estiver aberto, caso contrário, possuirá valor 0 (ou falso). Pode-se dizer que o disjuntor está aberto quando existe uma ***evidência positiva*** do fato, e que não está quando existe uma ***evidência negativa*** do mesmo.

Porém, às vezes, as informações referentes a um conceito não se encontram disponíveis. Por exemplo, não existe informação se o disjuntor 742 está, ou não, aberto. Pode-se dizer que existe uma ***ausência de evidência*** do fato. Em uma rede neural este problema é crítico, uma vez que é preciso utilizar algum valor no padrão, mesmo que o valor de um conceito de entrada não esteja disponível para a rede neural e mesmo que o valor de um conceito de saída, para um determinado padrão de entrada, seja desconhecido.

Em relação ao treinamento da rede, é necessário que o especialista seja capaz de determinar os valores de cada uma das saídas a partir dos padrões de entrada. Ou

seja, para cada exemplo de entrada ele deve poder determinar a evidência positiva ou negativa de cada um dos conceitos de saída. Em se tratando da utilização da mesma, é necessário que todos os valores de entrada estejam presentes. Ou seja, mesmo que o valor de algum conceito de entrada não esteja disponível, é preciso apresentar algum valor para a rede neural.

Normalmente, redes com este tipo de representação são ensinadas com valores "falso" como default. Assim, não sabemos se existe alguma evidência de que o conceito é falso (***evidência negativa***) ou se esta informação não encontra-se disponível (***ausência de evidência***). No caso exemplo, se "disjuntor 742 aberto" for falso, não sabemos se é porque não está realmente aberto (ou seja, está fechado) ou porque não existe nenhuma informação sobre o estado do disjuntor. Apesar disto, esta forma de representação é uma das mais utilizadas em redes neurais.

Deste modo, caso não exista nenhuma evidência de que o conceito é verdadeiro, o conceito deve ser falso. No exemplo, caso não exista nenhuma evidência de que "disjuntor 742 aberto" seja verdadeiro, então considera-se que seja falso (disjuntor fechado) *(Figura VI.1)*.

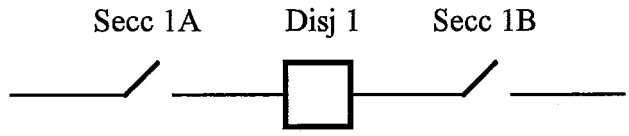


Figura VI.1 - Disjuntor isolado por Seccionadoras

Para que se possa fechar um disjuntor é preciso que as seccionadoras que o isolam estejam fechadas. No exemplo, para fechar o disjuntor 1 é preciso que tanto a seccionadora 1A quanto a seccionadora 1B estejam fechadas *(Tabela VI.2)*.

Secc.1A Aberta	Secc.1B Aberta	Fechar Disj.01
0	0	1
1	0	0
0	1	0

Tabela VI.2 - Exemplo de Conceitos como Dados Binários

Vamos supor agora que o dado referente ao conceito "Secc.1B Aberta" não esteja disponível. A rede neural tem que ser alimentada com algum valor, então, continuemos a supor que este valor default é 0 (é o valor normalmente utilizado). Deste modo, a rede pode decidir erroneamente por fechar o Disjuntor 01. Em exemplos simples como este é possível evitar erros tão grosseiros, porém, em exemplos mais complexos, estes erros podem ocorrer sem que se perceba.

b) Como dois dados binários:

Como vimos no item anterior, muitas vezes é preciso distinguir entre a evidência negativa e a ausência da evidência. Para contornar estas dificuldades muitas redes neurais representam não só o conceito mas também a negação deste. Por exemplo, podemos representar não somente "gerador 01 ligado" mas também "gerador 01 não ligado" (desligado). Ou seja, estamos representando separadamente tanto a evidência positiva quanto a evidência negativa do conceito. Quando ambas as evidências (positiva e negativa) forem falsas, ficará clara a ausência de evidências.

Normalmente, este tipo de representação utiliza os valores "falso" como default quando o conceito não encontra-se disponível. Com isto, estamos sempre ensinando à rede que o default é a ausência de evidência do conceito. Este "aproach" parece ser sempre mais adequado que o primeiro.

No exemplo anterior, supondo que não tivéssemos informação sobre a seccionadora 1B, ou seja, não possuíssemos os valores dos conceitos "Secc.1B Aberta" e "Secc.1B Fechada", e que o valor default de ambos fosse 0, mesmo assim a rede neural não optaria por fechar o disjuntor *(Tabela VI.3)*.

Secc.1A Aberta	Secc.1A Fechada	Secc.1B Aberta	Secc.1B Fechada	Fechar Disj.01
0	1	0	1	1
1	0	0	1	0
0	1	1	0	0

Tabela VI.3 - Exemplo de Conceitos como Dados Binários Duplos

Note, entretanto, que com isto duplicamos todas as entradas e saídas da rede, além de permitirmos a ocorrência da evidência positiva simultaneamente com a evidência negativa do mesmo conceito. Novamente utilizando o exemplo anterior, podemos ter dois estados antagônicos de um mesmo conceito, como "disjuntor aberto" e "disjuntor fechado", representados simultaneamente.

c) Como um dado ternário:

Resumindo, para cada conceito precisamos saber se existe alguma evidência positiva ou alguma evidência negativa do mesmo. Se nenhuma evidência existir (tanto positiva quanto negativa) podemos dizer que existe uma ausência de evidências. Assim, de acordo com um especialista, para um conceito qualquer pode existir a evidência positiva, a evidência negativa ou a ausência de evidências. Deste modo, um conceito qualquer pode ser diretamente verdadeiro, falso, ou não-disponível (ou seja, ter domínio {verdadeiro, não-disponível, falso}). Com isto, poderemos representar não só a evidência negativa do conceito, mas também as situações de ausência de evidência, nas quais o conhecimento não está disponível ou não é conhecido.

No exemplo anterior poderíamos ter o conceito "disjuntor" com o seguinte domínio {aberto, não-disponível, fechado}, eliminando a possibilidade de termos os dois conceitos representados simultaneamente *(Tabela VI.4)*. Esta forma de representação é claramente mais adequada a problemas de classificação.

Secc.1A Aberta	Secc.1B Aberta	Fechar Disj.01
-1	-1	1
1	-1	-1
-1	1	-1

Tabela VI.4 - Exemplo de Conceitos como Dados Ternários

Na ausência de evidência do conceito "Secc.1B Aberta" este teria valor 0, o que não levaria a rede a tomar a decisão errada.

d) Como um dado numérico:

Muitas vezes desejamos expressar o *grau de confiança* que possuímos em relação a um conceito. Por exemplo, "a previsão meteorológica disse que existe uma probabilidade de 60% de chover" não quer dizer que irá chover 60%, mas sim que temos uma confiança de 60% de que vá chover.

Para representar o grau de confiança em um conceito simples, utiliza-se uma variável numérica normalizada entre 0 e 1, onde 1 representa a total confiança em um conceito e 0 a ausência de confiança no mesmo. Deste modo, existem os mesmos problemas de quando se representa somente a evidência positiva, ou seja, é preciso definir valores "default" para o treinamento.

e) Exemplo:

Um disjuntor é um equipamento dos Sistemas Elétricos de Potência que pode possuir dois estados antagônicos {ABERTO, FECHADO}. Este mesmo equipamento pode ser representado de modos distintos em uma rede neural *(Tabela VI.5)*.

DADO	CONCEITO	VALOR	ESTADO
Binário	Disjuntor	0	Aberto
		1	Fechado
Binário+1	Disjuntor	0	Aberto
		1	Fechado
	Medida	0	Inconsistente
		1	Consistente
Binário Duplo	Disjuntor Aberto	0	Não-Aberto
		1	Aberto
	Disjuntor Fechado	0	Não-Fechado
		1	Fechado
Ternário	Disjuntor	-1	Aberto
		0	Desconhecido
		1	Fechado

Tabela VI.5 - Representações de um Disjuntor

VI.3.2 - Conceitos Compostos:

Nem sempre os conceitos a serem representados são simples como "disjuntor 742 aberto". Alguns conceitos parecem possuir toda uma gama de valores, podendo ser melhor representados por um conjunto finito de elementos (ou instâncias). Estes conceitos normalmente podem ser interpretados como o valor ou o estado do atributo de um objeto. Por exemplo: o conceito "cor" pode ser representado pelo conjunto {preto, vermelho, verde, azul, branco}; o estado de uma seccionadora pode ser {aberto, fechado, em-trânsito}; o estado de funcionamento do sistema pode ser {normal, alarme, emergência, restauração}. Conceitos simples podem ser interpretados como um caso particular de conceitos compostos com apenas dois elementos ("verdadeiro" e "falso") antagônicos.

a) Como um dado discreto:

Para representar um conceito deste tipo pode-se, a princípio, utilizar um conjunto discreto de elementos (valores), tais como {aberto, em-trânsito, fechado} sugeridos no exemplo acima. Assim, a rede é treinada de modo que algum dos elementos do domínio seja o "default". Existem diversas variações desta forma de representação baseadas no número de elementos deste conjunto.

Para representar também a evidência negativa de conceitos deste tipo, pode-se dobrar o conjunto discreto de elementos (valores). Assim, o conjunto de cores acima seria acrescido de {não-preto, não vermelho, não verde, não azul, não branco}. Ainda assim, a rede neural pode tomar uma decisão errada baseada nos defaults.

Para representar que o conceito não encontra-se disponível (ausência da evidência) pode ser acrescentada mais uma instância ao domínio ("não-disponível") no exemplo anterior {preto, vermelho, verde, azul, branco, não-disponível}. Assim, ao se treinar a rede, poder-se-ia usar o elemento "não-disponível" como default.

Por exemplo, uma seccionadora é um equipamento de manobra que possui 3 (três) estados {Aberta, Trânsito, Fechada}. Para representar este equipamento como um dado discreto podemos utilizar diferentes quantidades de elementos *(Tabela VI.6)*.

No. Elementos	CONCEITO	VALOR	ESTADO
N	Seccionadora	0	Aberto
		1	Trânsito
		2	Fechado
N + 1	Seccionadora	0	Não Disponível
		1	Aberto
		2	Trânsito
		3	Fechado
2*N	Seccionadora	0	Aberta
		1	Não Aberta
		2	Trânsito
		3	Não Trânsito
		4	Fechada
		5	Não Fechada
2*N + 1	Seccionadora	0	Não Disponível
		1	Aberta
		2	Não Aberta
		3	Trânsito
		4	Não Trânsito
		5	Fechada
		6	Não Fechada

Tabela VI.6 - Seccionadora como um Dado Discreto

b) Como um conjunto de conceitos simples:

Esta forma simples de representar conceitos compostos não contempla os casos em que estes domínios aparecem simultaneamente (múltiplas evidências). Por exemplo, é importante para o especialista que ele possa representar que um objeto é, ao mesmo tempo, verde, vermelho e branco.

Assim, podemos considerar que um conceito composto nada mais é do que um conjunto de conceitos simples (no caso exemplo, o conceito composto "cor" é um conjunto de conceitos simples "preto", "vermelho", "verde", "azul", "branco"), e que o conjunto de um ou mais conceitos simples compõe um conceito composto.

Se cada conceito simples for um dado binário, poderemos representar um conceito que seja ao mesmo tempo "verde", "vermelho" e "branco". Ou seja, a representação permite diferentes instâncias simultâneas de um mesmo conceito.

Porém, deste modo, não poderíamos representar a ausência da evidência de alguns conceitos. Para isto, devemos representar o conceito composto como o seguinte conjunto de conceitos simples: "preto", "não preto", "vermelho", "não vermelho", "verde", "não verde", "azul", "não azul", "branco", "não branco". Assim, passamos a representar a ausência da evidência como default.

Pode-se ver que, novamente, duplicam-se as entradas e saídas em prol da ausência da evidência. E mais uma vez o problema pode ser resolvido do mesmo modo, ou seja, representando cada conceito simples como {falso, não-disponível, verdadeiro}.

DADOS	CONCEITO	VALOR	ESTADO
Binários	Seccionadora Aberta	0	Não Aberta
		1	Aberta
	Seccionadora em Trânsito	0	Não Trânsito
		1	Trânsito
	Seccionadora Fechada	0	Não Fechada
		1	Fechada
Binários+1	Seccionadora Aberta	0	Não Aberta
		1	Aberta
	Seccionadora em Trânsito	0	Não Trânsito
		1	Trânsito
	Seccionadora Fechada	0	Não Fechada
		1	Trânsito
	Validade	0	Não Válido
		1	Válido
Ternários	Seccionadora Aberta	-1	Não Aberta
		0	Não Disponível
		1	Aberta
	Seccionadora em Trânsito	-1	Não Trânsito
		0	Não Disponível
		1	Trânsito
	Seccionadora Fechada	-1	Não Fechada
		0	Não Disponível
		1	Fechada
Codificada	Seccionadora	00	Aberta
		01	Trânsito
		10	Fechada

Tabela VI.7 - Seccionadora como um conjunto de conceitos simples

No exemplo anterior, uma seccionadora poderia ser representada por diferentes conjuntos de conceitos simples *(Tabela VI.7)*.

Uma outra forma de representar conceitos compostos como dados binários é codificando-os em um algarismo binário. Por exemplo, transformando um elemento de índice 3 (três) em um código binário (11) de tamanho 2 (dois).

VI.3.3 - Conceitos Numéricos:

No mundo real, muitas vezes as observações são medições realizadas através de instrumentos (tais como termômetros, etc...). Por exemplo, para concluir que uma pessoa está com febre podemos medir a temperatura dela através de um termômetro. A partir do valor medido podemos chegar à conclusão desejada: se a temperatura for maior que 37^{o} então a pessoa tem febre. Mas, quanto é 37 ? Como, mais geralmente, definimos os números ?

a) Como um dado numérico:

Deste modo, por que não representar a medida pelo próprio valor? Isto equivale a representar a evidência positiva de um conceito simples. À primeira vista parece ser suficiente utilizar o valor da medida como entrada para o problema. Porém, do mesmo modo que na representação da evidência positiva de conceitos simples, a rede deve ser treinada com valores default no caso de ausência de evidência.

b) Como um conceito composto:

É muito comum que uma medida seja representada não apenas por seu valor mas por faixas de medida, ou seja, regiões onde conceitualmente a medida embora com valores distintos tenha o mesmo significado. Por exemplo, uma medida de corrente através de um transdutor pode ter as seguintes faixas {ovf-negativo, baixa, normal, alta, ovf-positivo}. A principal diferença entre um conceito numérico composto e um conceito composto é a ordenação dos elementos.

c) Como um conjunto de conceitos simples:

Pode-se utilizar um conceito simples para representar o fato. Criamos um conceito para observação "temperatura maior que 37 graus" e utilizamos este conceito para concluir um outro conceito "pessoa tem febre".

Note, no entanto, que esta pode não ser a única conclusão a que se pode chegar a partir do valor medido. No exemplo, podemos querer concluir também que se a temperatura for maior que $38,5^{o}$ então a pessoa tem febre alta e deve procurar um médico. Podemos criar um novo conceito "temperatura maior que 38,5 graus" e utilizar este conceito para concluir o conceito "pessoa tem febre alta e deve procurar médico".

O raciocínio acima pode ser utilizado para cada uma das conclusões que se queira tirar a partir da medida. Assim, a medida deve ser pré-processada de modo a extrair os conceitos de observação desejados, transformando um conceito numérico em múltiplos conceitos simples.

Esta forma de representar os valores numéricos tem um inconveniente: é preciso saber previamente qual o pré-processamento necessário, sendo indicado apenas para problemas conhecidos. Porém, no caso dos problemas desconhecidos, não existe nenhuma indicação do pré-processamento necessário.

A codificação de um número como um padrão de dados binários (zeros e uns) em computadores pode ser feita de modos distintos: como caracteres representando dígitos, como um conjunto de caracteres formando palavras, como inteiros com diferentes quantidades de bits e como variáveis de ponto flutuante *(Anderson,90)*.

VI.4 - Conclusões

Os dados podem ser utilizados em Problemas de Classificação para representar conceitos *(Tabela VI.8)*.

Estes conceitos tem que ser representados como dados de exemplos de padrões a partir das representações existentes dos dados para as redes neurais *(Tabela VI.9)*.

CONCEITOS	INTERPRETAÇÃO
Simples	elemento de um conjunto
Compostos	conjunto não-ordenado de elementos
Numéricos	conjunto ordenado de elementos

Tabela VI.8 - Conceitos em Problemas de Classificação

Dentre as diferentes formas de representação, as que permitem representar a ausência de evidência dos conceitos proporcionam os melhores resultados em termos de generalização por serem mais adequadas ao raciocínio não-monotônico. Devemos também sempre que possível representar conceitos compostos como um conjunto de conceitos simples como os acima, de modo a poder representar simultaneamente evidências positivas ou negativas.

Conceitos	**Representação**	**Capacidade**	**Observações**
Simples	binária	evidência positiva ou evidência negativa	a mais comum, escolha do default
	binária dupla	evidência positiva e evidência negativa	duplica E/S, ausência como default
	ternária	ausência de evidência	ausência como default, melhor
	normalizada	grau de confiança	escolha do default
Compostos	discreta	evidência positiva	escolha do default
	discreta dupla	evidência negativa	ausência como default
	discreta + 1	ausência de evidência	ausência como default
como Simples	binária	evidências positivas simultâneas	escolha do default
	binária dupla	evidências positivas e negativas simultâneas	mesmo conceito pode ter evidência positiva e negativa simultânea
	ternárias	evidências positivas ou negativas simultâneas	mais adequado
Numéricos	numérica / normalizada	evidência positiva	escolha do default
como Compostos	discreta	"crisp sets"	-
como Simples	normalizada	"fuzzy sets"	-

Tabela VI.9 - Representação de Conceitos como Dados em Redes Neurais

Já no caso de conceitos numéricos, se o problema for conhecido também podemos considerar as variáveis numéricas como um conceito composto ou um conjunto de conceitos simples. Assim, podemos também concluir que o modo mais adequado de representar conceitos para as redes neurais é representar cada variável do ambiente como conceitos simples onde o default é a ausência da evidência (conceito não-disponível), sejam eles binários duplos ou ternários. Estes resultados podem ser melhor compreendidos no *Anexo B*, que implementa diversas formas de representação no exemplo do "Disjuntor Isolado por Seccionadoras".

Os dados relativos aos testes do exemplo do "Disjuntor Isolado por Seccionadoras" foram realizados no NevProp ("Nevada Propagation"), da Universidade de Nevada, utilizando um PC-AT 386 com co-processador, e podem ser encontrados no Anexo B.

Infelizmente, acreditamos que a solução deste problema se encontra no mecanismo de aprendizado, e que estas formas de representação somente representam um paliativo, uma vez que elas estão embutidas no próprio padrão de dados.

De todas as formas de representar a ausência de evidências, a mais fácil de ser utilizada pelos especialistas é a representação em dados ternários. Esta forma permite ao especialista representar a evidência positiva de um conceito como 1, a evidência negativa do mesmo como -1 e a ausência de evidência como 0.

É preciso destacar que, apesar das múltiplas formas de representação dos dados, as redes neurais utilizam elementos processadores similares, ou são digitais (binários) ou analógicos (contínuos). São raros os processadores ternários ou n-ários. As regras de aprendizado geralmente também independem da forma dos dados necessários, são geralmente funções numéricas ou, muito raramente, lógicas.

Acreditamos que o problema do aprendizado a partir de exemplos incompletos só será resolvido com utilização de um modelo ternário de elemento processador *(Figura VI.2)* com um mecanismo de aprendizado que explore estas características.

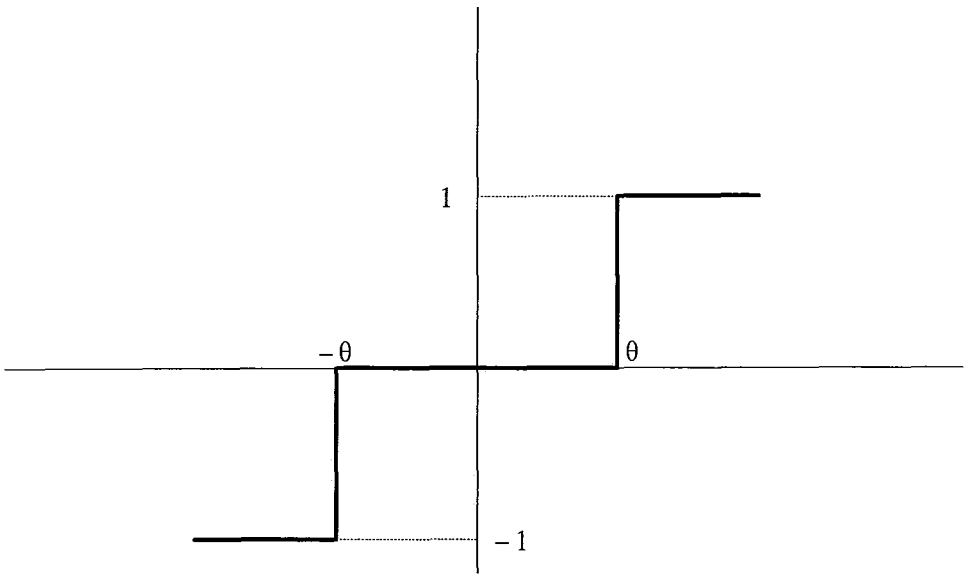


Figura VI.2 - Função de Transferência de um EP ternário.

Capítulo VII

VII - UMA ARQUITETURA PARA PROBLEMAS DE CLASSIFICAÇÃO

"The present contains nothing more then the past,
and what is found in the effect was already in the cause."

Henri Bergson, (em *"L'Évolution Créatrice"*)

VII.1 - Representação de Problemas em Redes Neurais

Um especialista consegue descrever facilmente os conceitos externos envolvidos em um problema, porém, apresenta imensa dificuldade em explicar como os utiliza de modo a resolver o problema, ou seja, como realiza o mapeamento entre eles.

Como vimos no *Capítulo V*, a representação localizada de conhecimento é mais intuitiva para um especialista, porém obriga-o a explicar as relações entre os conceitos envolvidos, o que não acontece na representação distribuída. Esta, por sua vez, é de difícil compreensão e, normalmente, obriga o desenvolvimento de um aplicativo que mapeie o conhecimento do especialista na representação distribuída e vice-versa.

O modelo de redes neurais conhecido como Back-Propagation é o mais utilizado em todo o mundo. Segundo seus próprios usuários, a principal razão de seu sucesso é a facilidade de utilização. O modelo (como a maioria dos demais modelos) aprende através de exemplos de padrões de entrada e saída, que são os conceitos externos que devem ser representados na rede. A partir destes exemplos a rede altera o valor dos pesos das conexões entre seus elementos processadores, de modo a otimizar o mapeamento entre estes conceitos.

Utilizando os conceitos apresentados no *Capítulo V* podemos concluir que uma rede Back-Propagation utiliza uma forma de representação externa do conhecimento

para representar os conceitos envolvidos em um problema. Isto facilita o entendimento pelo especialista, de modo que ele consegue representar os exemplos sem dificuldade. Normalmente, esta representação é localizada (mais intuitiva para um especialista), de modo que cada componente do padrão represente um conceito distinto. A partir destes exemplos, o mecanismo de aprendizado determina o peso de cada uma das conexões entre os elementos da rede neural, ou seja, determina uma representação interna para os mapeamentos entre os conceitos. Esta representação é distribuída, uma vez que é induzida isoladamente por cada elemento processador.

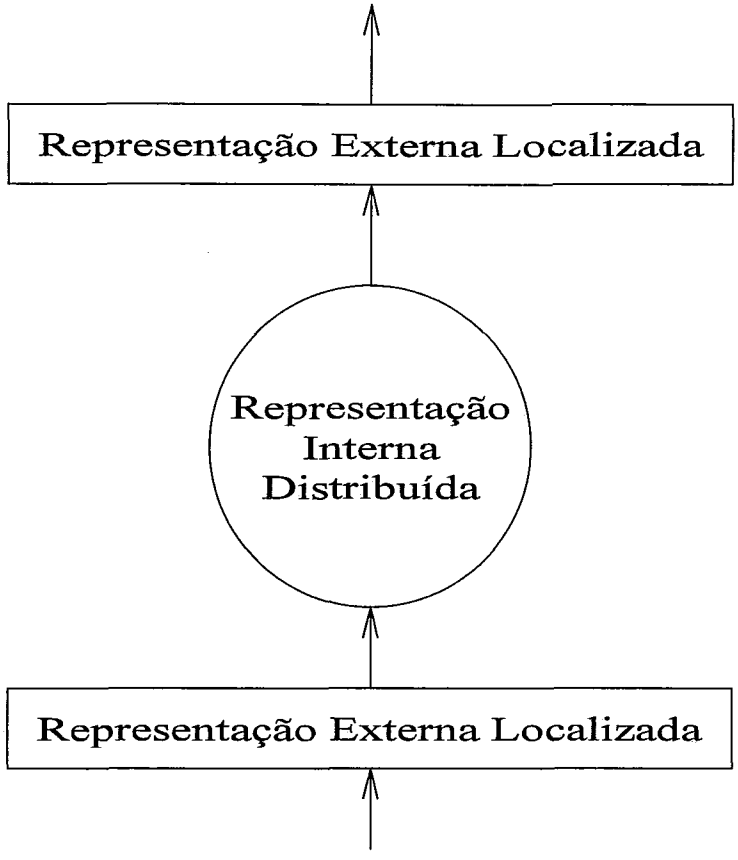


Figura VII.1 - Representação de Conhecimento (Back-Propagation)

Creditamos o sucesso da Back-Propagation a esta capacidade de representar externamente os conceitos e permitir induzir uma representação interna do mapeamento entre estes (*Figura VII.1*). Assim, para representar um problema em redes neurais deve ser utilizado algum "approach" similar *(Tabela VII.1)*.

Conceitos	Mapeamentos
Repr. Localizada	Repr. Distribuída
Repr. Externa	Repr. Interna
Padrão de Ativação	+Padrão de Interconexão
Dados de E/S	Aprendizado

Tabela VII.1 - Representação de Problemas em Redes Neurais

VII.2 - Redes Neurais em Classificação

Tradicionalmente, os modelos de redes neurais são aplicados à problemas de classificação do mesmo modo que à associação entre padrões *(Capítulo IV)*, sem o cuidado de analizar as características particulares inerentes a esses problemas.

É preciso, primeiro, considerar que o especialista é quem determina quais são os conceitos significativos e fornece os exemplos adequados. Ele é a pessoa mais capacitada na resolução do problema em questão. Portanto, a rede neural a ser utilizada deve possuir um mecanismo de aprendizado supervisionado.

Este aprendizado deve ser feito a partir de exemplos *(Equação VII.1)*, onde o símbolo "⇒" representa uma dependência conceitual dos conceitos de saída **S** em relação aos conceitos de entrada **E**.

$$\mathbf{E} \Rightarrow \mathbf{S}$$

Equação VII.1

No paradigma classificador, os padrões de saída são diferentes dos padrões de entrada, indicando a utilização de redes hetero-associativas em sua resolução.

Como vimos no *Capítulo IV*, a maior parte da literatura existente sobre classificadores trata de um caso particular denominado categorizadores, onde os conceitos de entrada são totalmente independentes dos conceitos de saída. Porém, um especialista muitas vezes se utiliza de ***conceitos intermediários*** (conceitos que às vezes servem de entrada e outras vezes servem de saída) na resolução dos problemas

que envolvem classificação. Ou seja, o especialista recorre a exemplos *(Equação VII.2)*, onde **I** representa os conceitos intermediários.

$$\mathbf{(E+I)\Rightarrow(I+S)}$$

Equação VII.2

Neste tipo de exemplos, os conceitos intermediários podem ser utilizados tanto como entrada quanto como saída. Um conceito intermediário pode, inclusive, ser utilizado para obter outro conceito intermediário, obrigando a rede a ser recursiva.

Os conceitos intermediários devem ser utilizados como entrada somente durante o aprendizado da rede, pois durante sua utilização apenas os conceitos de entrada estarão disponíveis. Isto implica em diferentes conjuntos de conceitos durante o treinamento e o funcionamento normal da rede (lembrança ou "recall"). Devemos poder treinar a rede com exemplos iguais à *Equação VII.2* e utilizá-la para obter padrões de saída **S** e padrões intermediários **I** a partir de padrões de entrada **E** *(Equação VII.3)*.

$$\mathbf{E\Rightarrow(I+S)}$$

Equação VII.3

Finalmente, até mesmo os exemplos a serem utilizados durante o treinamento não são "completos". Ou seja, o especialista não fornece exemplos em que TODAS as entradas resultem em TODAS as saídas, pelo contrário, ele procura selecionar exemplos gerais e, a partir destes, se especializa em exemplos mais detalhados. Esta característica requer que exista alguma forma de apresentar exemplos incompletos, normalmente, através da representação da ausência de evidência destes conceitos nos dados que compõem os padrões de entrada e saída da rede neural *(Equação VII.4)*. Este requisito engloba o anterior, proporcionando um funcionamento idêntico à *Equação VII.3*.

$$(\mathbf{E}+\{\})\Rightarrow(\mathbf{I}+\mathbf{S})$$

(Equação VII.4)

As restrições *(Tabela VII.2)* combinadas praticamente eliminam quaisquer modelos atualmente existentes. Embora estes requisitos sejam essenciais, é possível levantar as conseqüências do afrouxamento de cada uma destas restrições e sugerir mecanismos que minimizem essas conseqüências.

PROBLEMAS DE CLASSIFICAÇÃO	REDES NEURAIS
exemplos fornecidos por especialistas	aprendizado supervisionado
observações <> conclusões	hetero-associativa
conceitos intermediários	recursiva
exemplos incompletos	??????

Tabela VII.2 - Redes Neurais em problemas de Classificação

Em uma rede hetero-associativa é preciso que os conceitos intermediários estejam presentes tanto nos padrões de entrada como nos padrões de saída. Caso a rede seja auto-associativa é preciso definir somente quais são os conceitos existentes no universo do problema (**E+I+S**). Porém deste modo a relação de dependência entre os conceitos (qual o conjunto de conceitos que determina os demais) não é explicitada. É preciso garantir que os padrões (**E+I+S**) possam ser recuperados a partir somente dos conceitos de entrada (**E**).

Para processar conceitos em redes sem realimentação ("feed-foward") é preciso apresentá-los como padrões de entrada e, após o processamento, retirar os padrões de saída resultantes. Nessas redes direcionadas, o processamento é realizado num intervalo de tempo determinístico proporcional ao número de camadas do modelo. Se imaginarmos uma rede direcionada como um dígrafo acíclico, o tempo de processamento é proporcional à profundidade do dígrafo. Porém, se tivermos conceitos intermediários não é possível garantir que, em uma única etapa de

processamento, os conceitos de entrada originem os conceitos de saída desejados, principalmente se existirem conceitos intermediários dependentes de outros conceitos intermediários.

É possível utilizar redes direcionadas, mesmo com conceitos intermediários. Para que se obtenha os conceitos de saída desejados é preciso reapresentar os conceitos intermediários obtidos nesta etapa em uma nova etapa de processamento, repetindo este processamento iterativo até que o padrão de saída não mais se modifique. À principio, nada garante que em algum instante o padrão de saída não mais se modificará, porém é possível embutir no aprendizado mecanismos que garantam a convergência do padrão de saída.

A utilização do processamento iterativo se assemelha à utilização de redes recursivas. Em uma rede com realimentação, os conceitos devem estar associados a estados estáveis na rede, de tal modo que após a apresentação dos padrões de entrada, mais um determinado intervalo de tempo, a rede convirja para um padrão de saída estável.

Finalmente, se a rede neural não puder utilizar exemplos incompletos, ou seja, tiver que utilizar exemplos completos, torna a ser necessário permitir diferentes padrões para treinamento e utilização. Uma alternativa é treinar com exemplos iguais à *Equação VII.4* e testar com exemplos iguais à *Equação VII.3*.

VII.2 - Arquitetura Proposta

A maioria dos modelos conhecidos possuem aprendizado supervisionado e são hetero-associativas, porém não apresentam uma topologia recursiva (são redes "feed-forward"). Um exemplo típico é a rede Back-Propagation, atualmente a rede mais utilizada em aplicações práticas no mundo.

Os modelos recursivos são menos utilizados, e sua aplicação prática é ainda mais restrita. *Hopfield (86)* foi um dos pioneiros em sua utilização, desenvolvendo uma rede totalmente recursiva, a ***Hopfield***, que funciona como uma memória auto-

associativa. *Hinton (86)* criou um tipo de rede que extendia as Redes de Hopfield para incluir unidades de processamento escondidas, denominada ***máquina de Boltzmann***, pois a probabilidade dos estados do sistema era dada por uma distribuição de Boltzmann da estatística.

As redes recursivas apresentam sérios problemas de convergência e estabilização. Uma abordagem recente utiliza redes com a maioria das conexões em uma direção ("feed-foward") e algumas, cuidadosamente selecionadas, na outra direção ("feed-back"). Estes modelos são denominados ***parcialmente recursivos***, e em grande parte destes as conexões de retorno ("feed-back") têm peso fixo, permitindo a utilização de variações do algoritmo Back-Propagation.

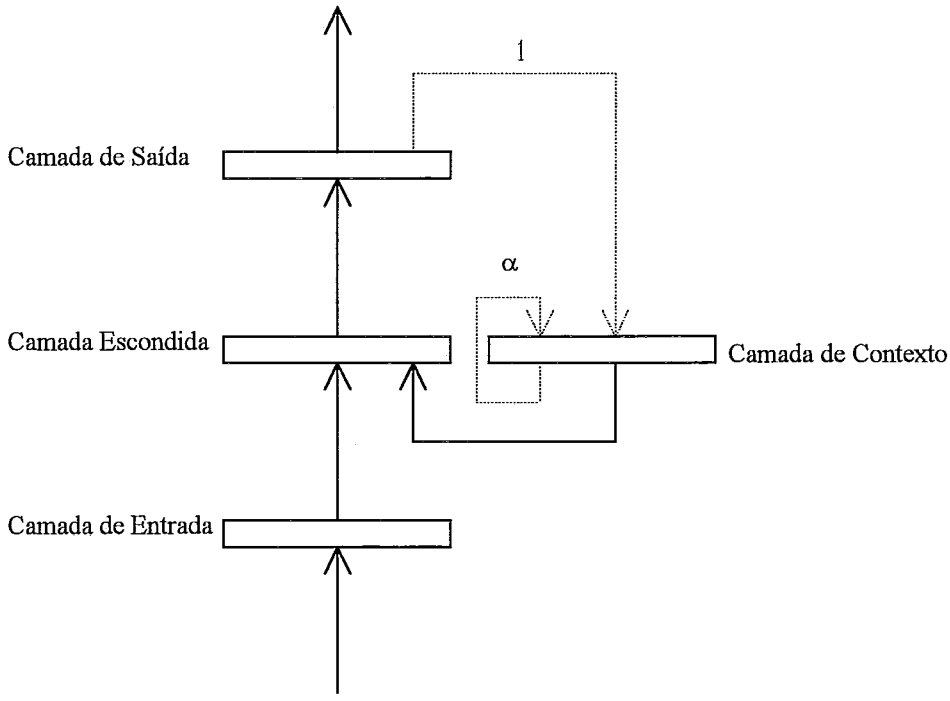


Figura VII.2 - Rede de Jordan

Jordan (89) criou uma rede parcialmente recursiva *(Figura VII.2)*, na qual uma camada extra, denominada ***camada de contexto***, copiava o padrão de ativação da camada de saída no instante anterior.

Além disso, esta camada era realimentada por si mesma, de modo a acumular os traços dos valores passados. A regra de atualização dos elementos processadores da

camada de contexto era dada pela *Equação VII.4*. Onde C_i representa o estado dos elementos da camada de contexto e O_i representa o estado dos elementos da camada de saída.

$$C_i(t+1) = \alpha C_i(t) + O_i(t)$$

Equação VII.4

As conexões recursivas permitiam que a camada escondida utilizasse suas próprias respostas anteriores de modo a orientar seu comportamento subseqüente, criando uma memória na rede. *Jordan (89)* mostrou que, com uma entrada fixa, sua rede podia ser treinada para gerar uma seqüência de saídas, além de poder ser treinada para reconhecer e distinguir diferentes seqüências de entradas. *Cleeremans (89)* mostrou que uma rede deste tipo era capaz de aprender a emular um Autômato de Estados Finitos. *Elman (90)* propôs uma rede similar, porém mais simples, na qual a camada de contexto copiava os valores da camada escondida ao invés da camada de saída *(Figura VII.3)*.

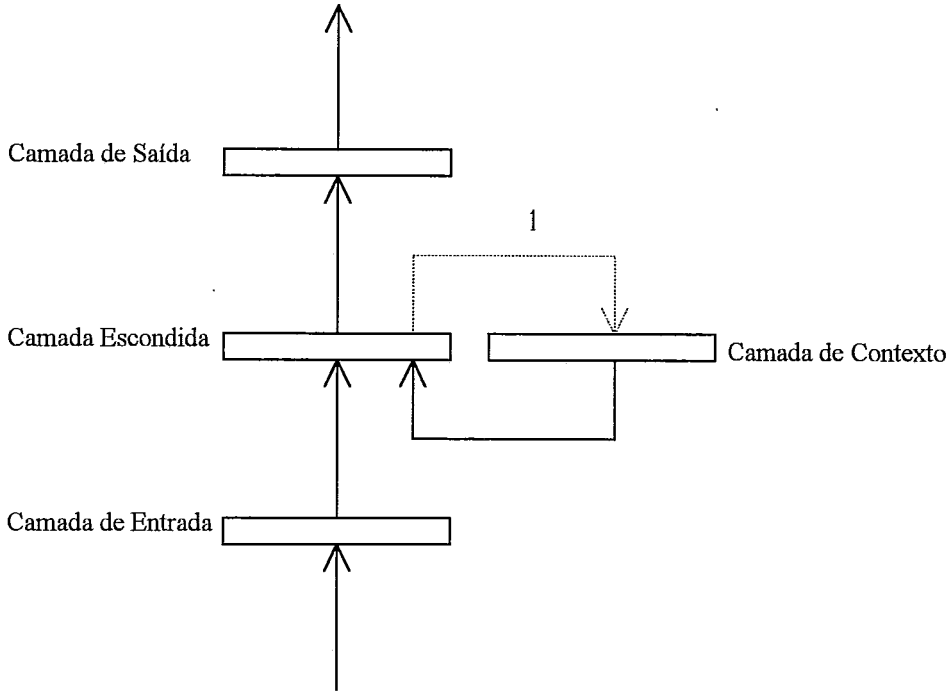


Figura VII.3 - Rede de Elman

Nesta arquitetura, a camada de contexto armazena o estado anterior da camada intermediária, provendo a rede de memória. A camada escondida, por sua vez, deve mapear tanto as entradas quanto o estado anterior nas saídas desejadas. Os padrões internos são salvos como contexto e este, por sua vez, é utilizado na determinação do próximo padrão interno. Deste modo, os padrões internos são sensíveis ao contexto.

Baseado nas arquiteturas de Jordan e Elman, desenvolvemos uma arquitetura *(Figura VII.4)* de redes neurais adequada à resolução de problemas que envolvem Classificação.

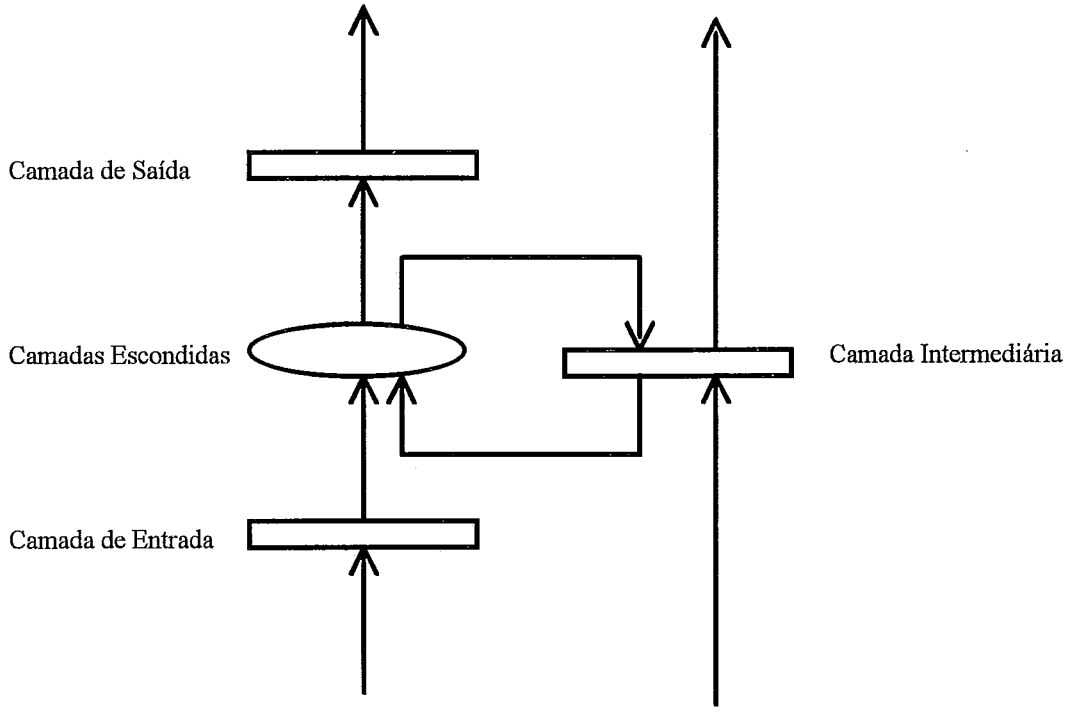


Figura VII.4 - Rede Neural para Problemas de Classificação

A arquitetura sugerida, além de prover a rede de memória, permite treiná-la com o contexto desejado. Este contexto é aproveitado durante a determinação do próximo contexto e, assim sucessivamente.

Esta arquitetura permite resolver 3 (três) dos 4 (quatro) requisitos necessários, ou seja, aprendizado supervisionado, hetero-associatividade e recursividade. É preciso, portanto, resolver ainda o problema do aprendizado a partir de exemplos incompletos.

Esta não é uma limitação relativa apenas à problemas de classificação, mas comum a todas as aplicações em Redes Neurais.

VII.3 - Um Algoritmo para a Arquitetura Sugerida

As redes desenvolvidas por Jordan e Elman possuem somente conexões recursivas fixas e permitem a utilização do algoritmo de aprendizado "Back-Propagation" *(Rumelhart,86)*. Distintamente destas, a arquitetura proposta envolve conexões recursivas adaptáveis, não sendo possível a utilização deste algoritmo.

Pineda (89), *Almeida (87)* e *Rohwer (87)* independentemente mostraram que o algoritmo Back-Propagation podia ser extendido para Redes Neurais arbitrárias. Estes algoritmos modificados são usualmente chamados de ***Back-Propagation Recursivo***.

A dificuldade deste algoritmo é garantir a convergência para estados estáveis, assim como em qualquer rede recursiva. Várias regras dinâmicas *(Equação VII.5)* podem ser impostas para a rede de modo a convergir para os pontos fixos corretos.

$$\tau U_i' = - U_i + g(h_i)$$

Equação VII.5

Onde U_i representa o estado de ativação de uma unidade, g( ) representa a função de transferência e h_i representa a combinação das entradas de U_i.

Isto requer calcular uma matriz inversa *(Rohwer,87)*, em uma operação não-local, em cada passo do aprendizado para calcular a variação dos pesos. Porém, *Pineda(89)* mostrou que uma versão modificada da própria rede *(Figura VII.5a)* poderia realizar este cálculo mais rapidamente (e localmente).

A topologia da rede de erros *(Figura VII.5b)* necessária é similar à rede original, a regra dinâmica é idêntica à *Equação VII.5*, e o procedimento completo pode ser dividido em 4 (quatro) etapas:

1. relaxar a rede original com a *Equação VII.5* ;

2. comparar com a saída desejada para obter os erros ;
3. relaxar a rede de erros com a *Equação VII.5* ;
4. atualizar os pesos.

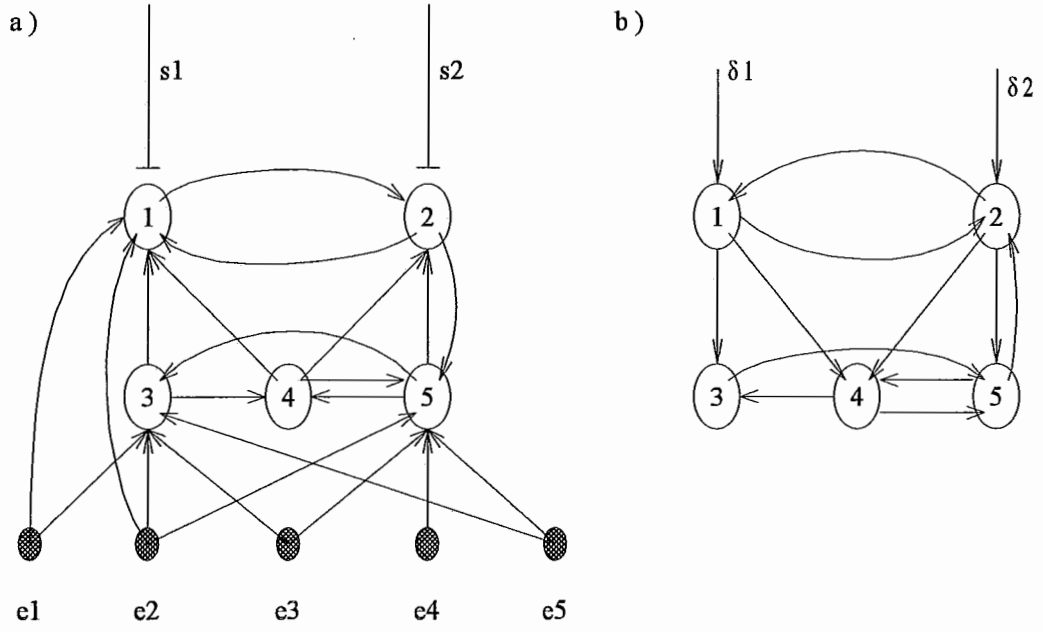


Figura VII.5 - Exemplo de Rede de Propagação de Erros Associada

O procedimento acima permite expandir o algoritmo Back-Propagation para qualquer rede, inclusive as recursivas, sem realizar operações que não sejam locais (como inversões de matrizes). O Back-Propagation pode ser, então, considerado como um caso particular do Back-Propagation Recursivo.

No mesmo trabalho em que criticou os Perceptrons, *Minsky (69)* mostrou que para qualquer rede neural recursiva existia uma rede neural "feed-foward" correspondente *(Figura VII.6)*, com o mesmo funcionamento (em um intervalo finito de tempo).

Rumelhart (86d) mostrou como fazer esta conversão e qual a forma correta da regra de aprendizado necessária para a rede recursiva. A rede recursiva é discretizada em um intervalo finito de tempo. A rede sem realimentação equivalente pode ser

treinada através de uma versão ligeiramente diferente do algoritmo Back-Propagation, conhecido como ***Back-Propagation Through Time (BPTT)***. De um modo geral, o procedimento consiste em: apresentar o padrão de entrada para o sistema; permitir que ele rode por um determinado número de iterações; comparar o resultado com a saída desejada; retropropagar os erros obtidos pelo mesmo número de iterações; e atualizar os pesos.

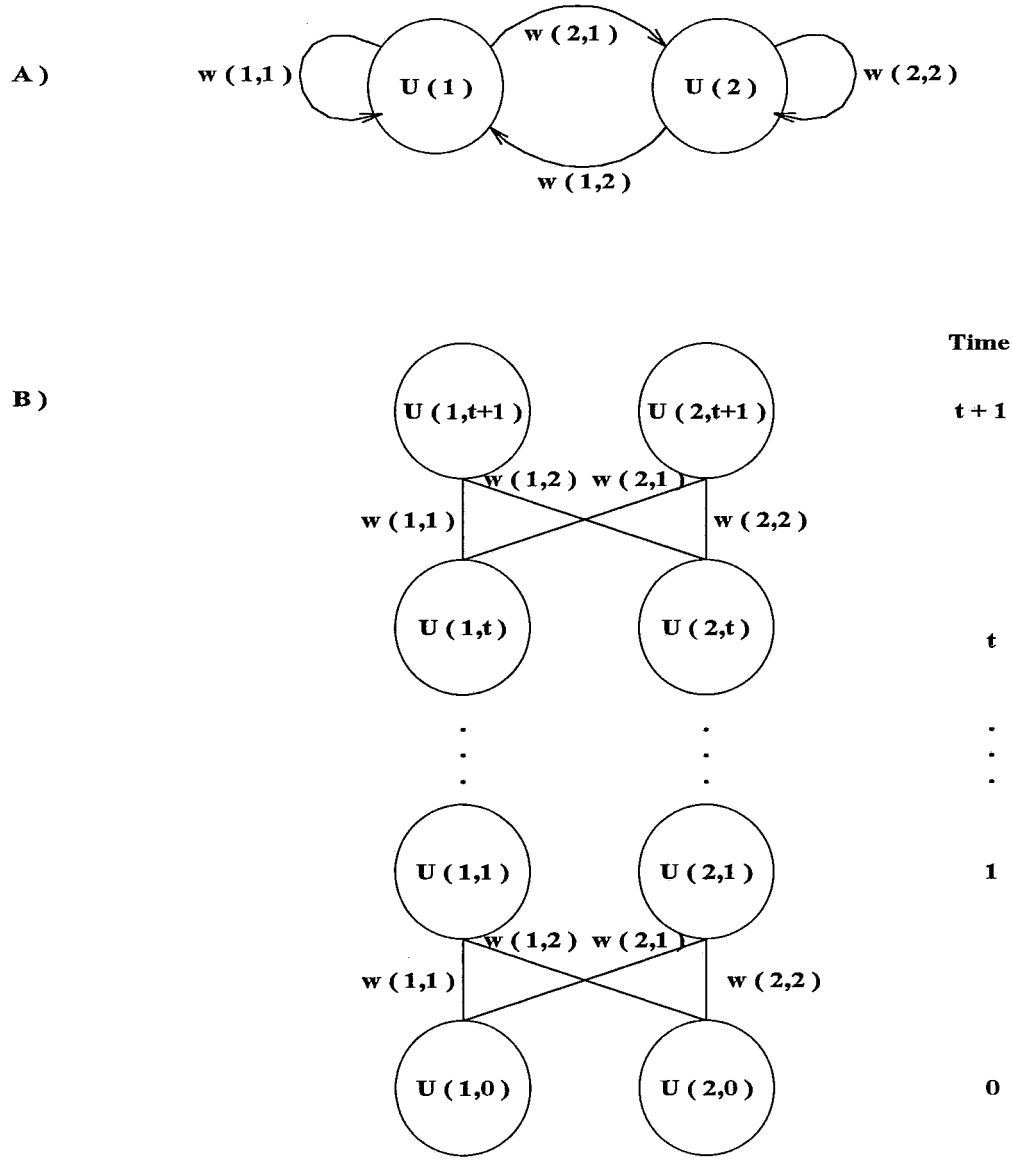


Figura VII.6 - Exemplo de Redes Equivalentes no Tempo

Para um intervalo **t** igual a {1, 2, ..., T}, existirão **T** unidades para cada unidade original, de modo que cada unidade **u(i,t)** representará o estado da unidade $\mathbf{u_i}$ no instante **t**. Este algoritmo é especialmente útil para a especificação de seqüências de entrada e saída. As entradas e saídas especificadas para uma unidade **i** em um instante **t** devem ser aplicadas diretamente na unidade **u(i,t)**. Uma complicação é a restrição de que TODAS as cópias de cada conexão **wij** sejam idênticas, uma vez que o algoritmo Back-Propagation normalmente produz diferentes incrementos **Δwij** para cada uma das cópias. A solução usual consiste em somar os incrementos individuais e, então, modificar cada uma das cópias pelo total.

O principal problema desse "approach" é a necessidade de elevados recursos computacionais, devido à duplicação das unidades. Para seqüências compridas ou de tamanho indeterminado, o custo se torna impraticável. Apesar disso, esta restrição se aplica somente à fase de treinamento, pois, uma vez treinada a rede discretizada no tempo, a rede recursiva original com os pesos obtidos pode ser utilizada.

Zipser (89) criou uma regra de aprendizado para redes recursivas genéricas sem duplicar as unidades. A regra pode, inclusive, ser utilizada "on-line". Ou seja, aprendendo enquanto os padrões são sequencialmente apresentados. Deste modo, ele é capaz de lidar com seqüências de tamanho indeterminado. Porém, este algoritmo requer a manutenção, para cada intervalo de tempo, de N^4 derivadas. Este algoritmo foi denominado ***Real-Time Recurrent Learning (RTRL)***.

Pearlmutter(89) criou um algoritmo para treinar uma rede recursiva genérica em intervalos contínuos de tempo. Este algoritmo, denominado ***Time-Dependent Recurrent Back-Propagation***, pode ser visto tanto como uma extensão do RTRL para intervalos contínuos quanto como uma extensão do Back-Propagation Recursivo para seqüências dinâmicas.

Como o problema em questão não exige intervalos contínuos de tempo, nem aprender durante a apresentação das entradas, selecionamos o Back-Propagation Through Time para ser utilizado na arquitetura sugerida. Uma de suas maiores dificuldades está em determinar o número de iterações, de modo que seja

suficientemente grande para garantir a resposta desejada e que não torne o procedimento inviável. De modo a limitar o número destas iterações, *Zipser (90)* criou uma variação deste algoritmo denominada ***Truncated Back-Propagation Through Time***, que utilizamos para testar a arquitetura proposta. No caso dos problemas de Classificação este número pode ser determinado através da altura da árvore de hierarquia ou da profundidade do grafo correspondente. A regra de atualização discretizada no tempo passa a ser dada pela *Equação VII.6.*

$$U_i(t+1) = g\ (\ \Sigma\ w_{ij}\ U_j(t) + \varepsilon_i(t)\ )$$

Equação VII.6

Assim como toda rede recursiva, a arquitetura proposta pode ser igualmente discretizada no tempo *(Figura VII.7)*.

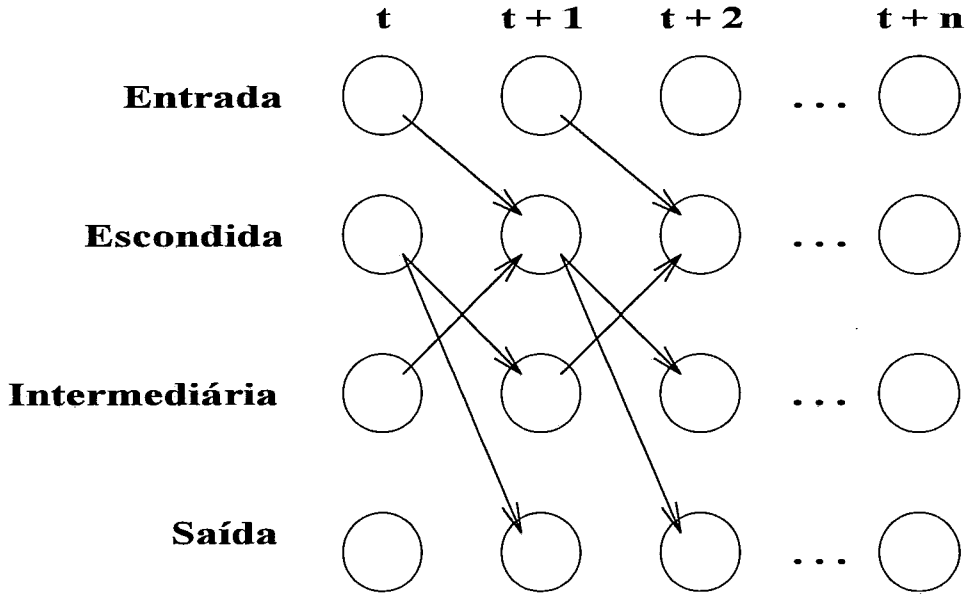


Figura VII.7 - Discretização da Arquitetura Sugerida no Tempo

O algoritmo sugerido minimiza o erro entre os padrões obtidos e os desejados. Deste modo, ele necessita que estes exemplos sejam completos, violando um dos

requisitos para utilização em problemas de Classificação. Conforme visto no início deste capítulo, é preciso treiná-la com padrões iguais à *Equação VII.4* e utilizá-la com padrões iguais à *Equação VII.3 (Figura VII.8)*.

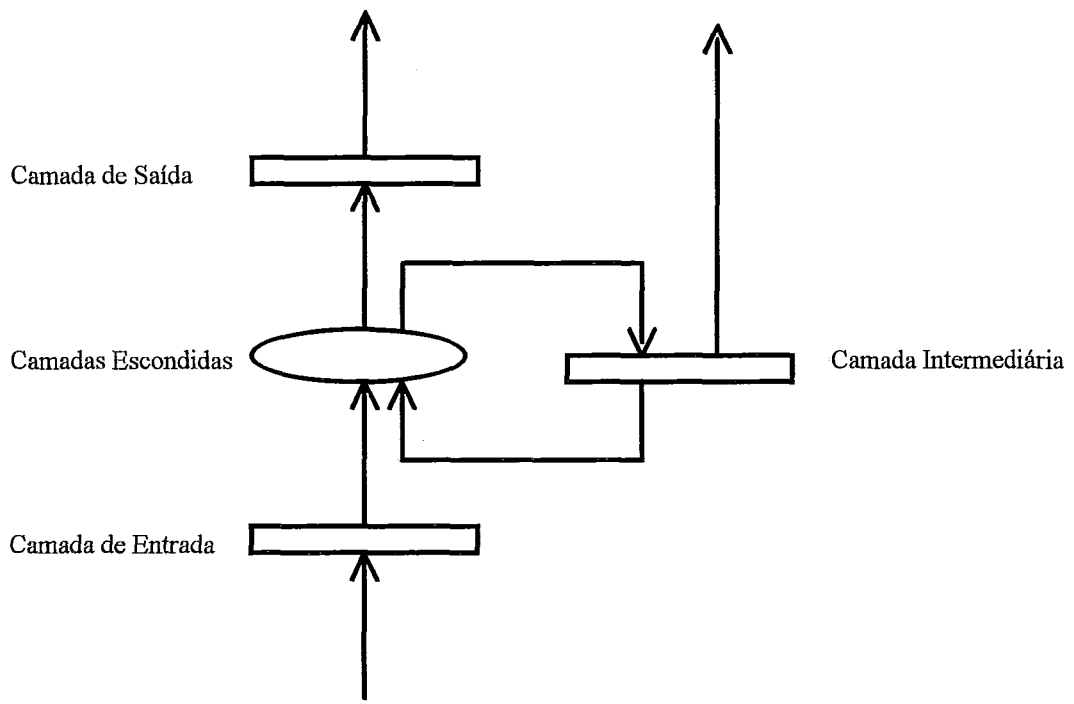


Figura VII.8 - Atualização ("recall") na Arquitetura Proposta

A arquitetura apresentada foi testada em dois exemplos descritos a seguir. Ambos os testes foram realizados no SNNS v3.1 ("Stuttgart Neural Network Simulator"), da Universidade de Stuttgart, utilizando uma "workstation" da SUN. Sua descrição e resultados podem ser encontrados, respectivamente, nos Anexos C e D.

VII.4 - Implementação e Testes

VII.4.1 - Detecção de Falhas em Transformadores

O primeiro exemplo *(Figura VII.9)* soluciona o problema apresentado no *Capítulo II*, e pode ser induzido a partir da tabela apresentada pelo IEC *(Tabela II.2)*.

Como cada especialista tem sua própria experiência, *Tomsovic (93)* utilizou lógica Fuzzy para integrar diferentes métodos de diagnósticos em transformadores.

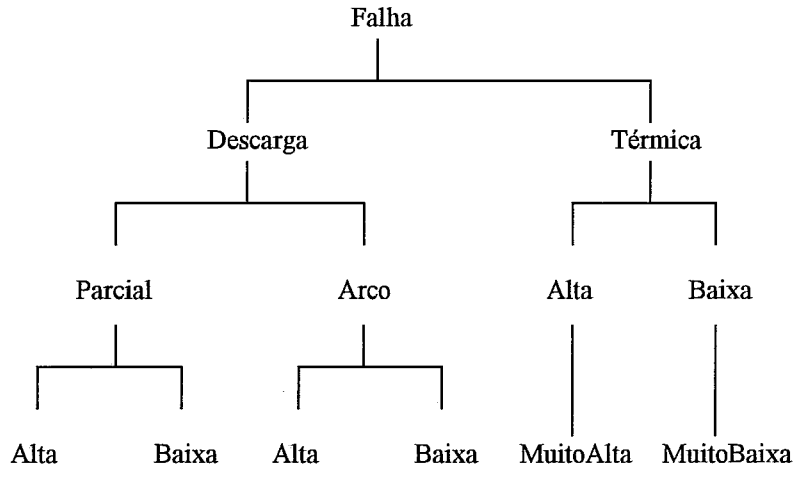


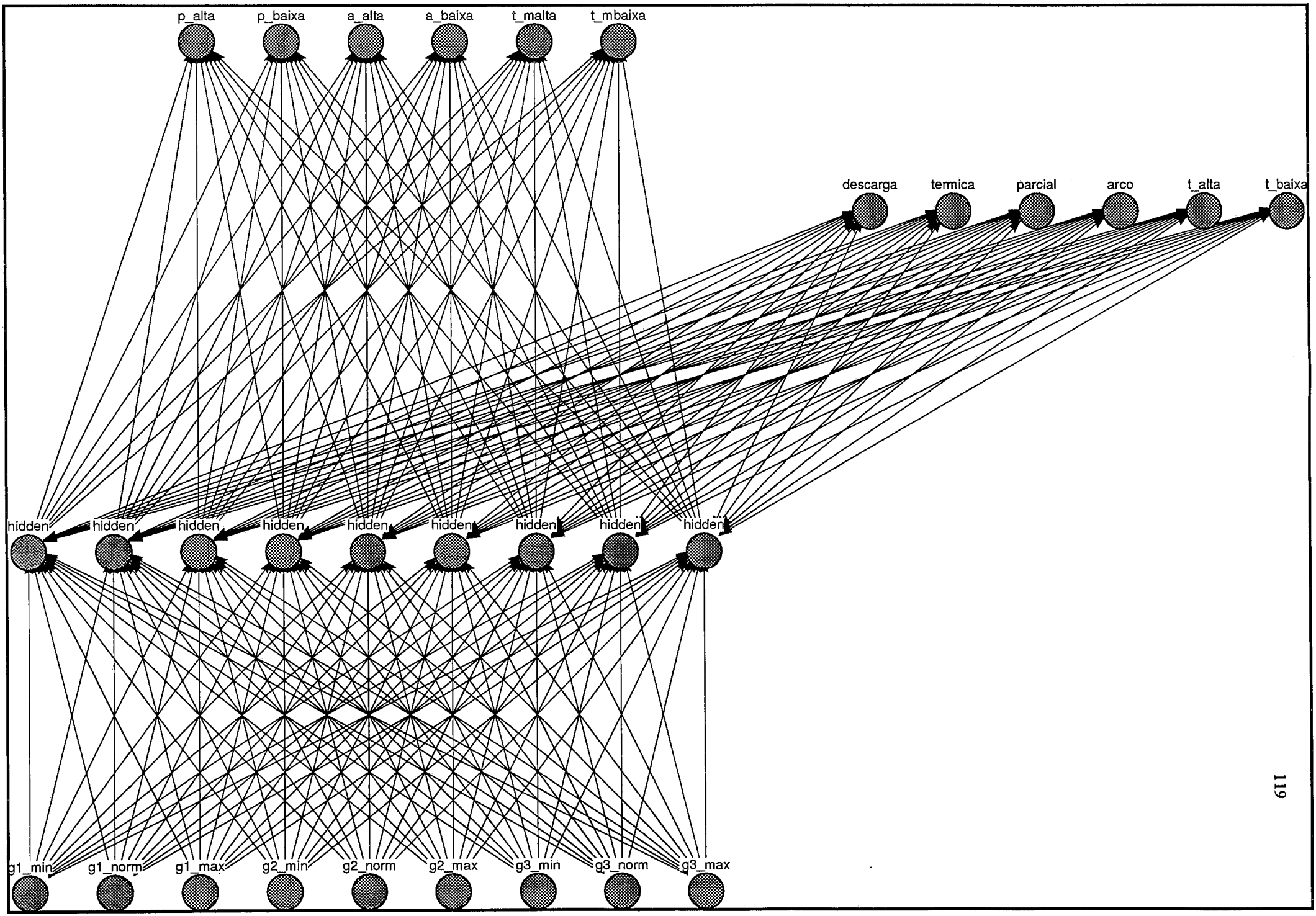
Figura VII.9 - Classificação de Falhas em Transformadores (Tomsovic,93).

O problema consiste em associar as falhas em transformadores à relações entre os gases dissolvidos coletados, como vimos no *Capítulo II - item 5.2.* A tabela *(Tabela II.2)* apresentada utiliza 3 (três) relações entre gases para determinar as possíveis falhas (Acetileno/Etileno, Metano/Hidrogênio e Etileno/Etano). Cada uma destas relações entre gases pode estar em três faixas distintas (mínimo, normal, máximo), de acordo com seu valor *(Tabela VII.3)*.

RELAÇÃO	Mínimo	Normal	Máximo
Acetileno/Etileno	< 0.1	0.1 - 3.0	> 3.0
Metano/Hidrogênio	< 0.1	0.1 - 1.0	> 1.0
Etileno/Etano	< 1.0	1.0 - 3.0	> 3.0

Tabela VII.3 - Faixas de Relações entre Gases Dissolvidos

Para implementação na arquitetura proposta associamos cada faixa a um elemento da camada de entrada. Os elementos de saída representam cada uma das possíveis falhas. E os diagnósticos intermediários são representados por elementos intermediários *(Figura VII.10)*.


p_alta
p_baixa
a_alta
a_baixa
t_malta
t_mbaixa
descarga
termica
parcial
arco
t_alta
t_baixa
hidden
hidden
hidden
hidden
hidden
hidden
hidden
hidden
hidden
g1_min
g1_norm
g1_max
g2_min
g2_norm
g2_max
g3_min
g3_norm
g3_max

A partir da *Tabela VII.3* juntamente com a *Tabela II.2* podemos obter os exemplos necessários ao treinamento *(Tabela VII.4)*.

	1	2	3	4	5	6	7	8	9	10	11
Rel.Gás 1 (mínimo)	1	0	0	0	0	0	0	1	1	1	1
Rel.Gás 1 (normal)	0	1	1	0	1	0	1	0	0	0	0
Rel.Gás 1 (máximo)	0	0	0	1	0	1	0	0	0	0	0
Rel.Gás 2 (mínimo)	1	1	0	0	0	0	0	0	0	0	0
Rel.Gás 2 (normal)	0	0	1	1	1	1	1	1	0	0	0
Rel.Gás 2 (máximo)	0	0	0	0	0	0	0	0	1	1	1
Rel.Gás 3 (mínimo)	1	1	1	1	0	0	0	0	1	0	0
Rel.Gás 3 (normal)	0	0	0	0	1	1	0	1	0	1	0
Rel.Gás 3 (máximo)	0	0	0	0	0	0	1	0	0	0	1
Descarga	1	1	1	1	1	1	1	0	0	0	0
Térmica	0	0	0	0	0	0	0	1	1	1	1
Parcial	1	1	0	0	0	0	0	0	0	0	0
Arco	0	0	1	1	1	1	1	0	0	0	0
Térmica Baixa	0	0	0	0	0	0	0	1	1	0	0
Térmica Alta	0	0	0	0	0	0	0	0	0	1	1
Parcial Baixa	1	0	0	0	0	0	0	0	0	0	0
Parcial Alta	0	1	0	0	0	0	0	0	0	0	0
Arco Baixa	0	0	1	1	1	1	0	0	0	0	0
Arco Alta	0	0	0	0	0	0	1	0	0	0	0
Térmica Muito Baixa	0	0	0	0	0	0	0	1	0	0	0
Térmica Muito Alta	0	0	0	0	0	0	0	0	0	0	1

Tabela VII.4 - Exemplos de Treinamento (Transformadores)

O Anexo D contém toda a documentação do procedimento de treinamento e testes realizado para o exemplo. Os testes mostram que a rede foi capaz de aprender os exemplos fornecidos e abstrair para os casos não abrangidos pelo critério do IEC *(Tabela VII.5)*.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
Rel.Gás 1 (mínimo)	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Rel.Gás 1 (normal)	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
Rel.Gás 1 (máximo)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
Rel.Gás 2 (mínimo)	1	1	1	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
Rel.Gás 2 (normal)	0	0	0	1	1	1	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	1	1	1	0	0	0
Rel.Gás 2 (máximo)	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	1	1	1
Rel.Gás 3 (mínimo)	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0
Rel.Gás 3 (normal)	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0
Rel.Gás 3 (máximo)	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1
Descarga	1	1	1	1	0	0	0	0	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	0	1
Térmica	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0
Parcial	1	1	1	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0.5	1	1	1	0	0	0	0	0	0.5
Arco	0	0	0	0.5	0	0	0	0	0	0	0	0	1	1	1	0.5	0	0.5	0	0	0	1	1	1	0.5	0	0
Térmica Baixa	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Térmica Alta	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.5	0
Parcial Baixa	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Parcial Alta	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
Arco Baixa	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	1	0	0	0	0
Arco Alta	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0
Térmica Muito Baixa	0	0	0	0	1	0.5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Térmica Muito Alta	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabela VII.5 - Exemplos de Teste (Transformadores)

VII.4.2 - *Telfer (91)*

Como exemplo das idéias apresentadas utilizaremos uma variação do exemplo fornecido por *Telfer (91)* sobre a força aérea em um artigo publicado no periódico *Neural Networks*. Especialistas espontaneamente levantaram a classificação abaixo para determinação de diferentes aeronaves *(Figura VII.11)*.

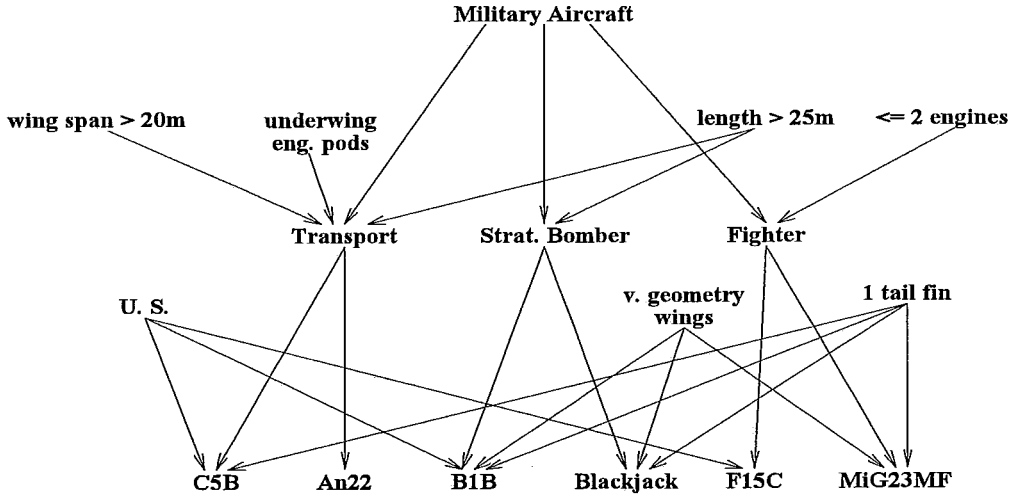
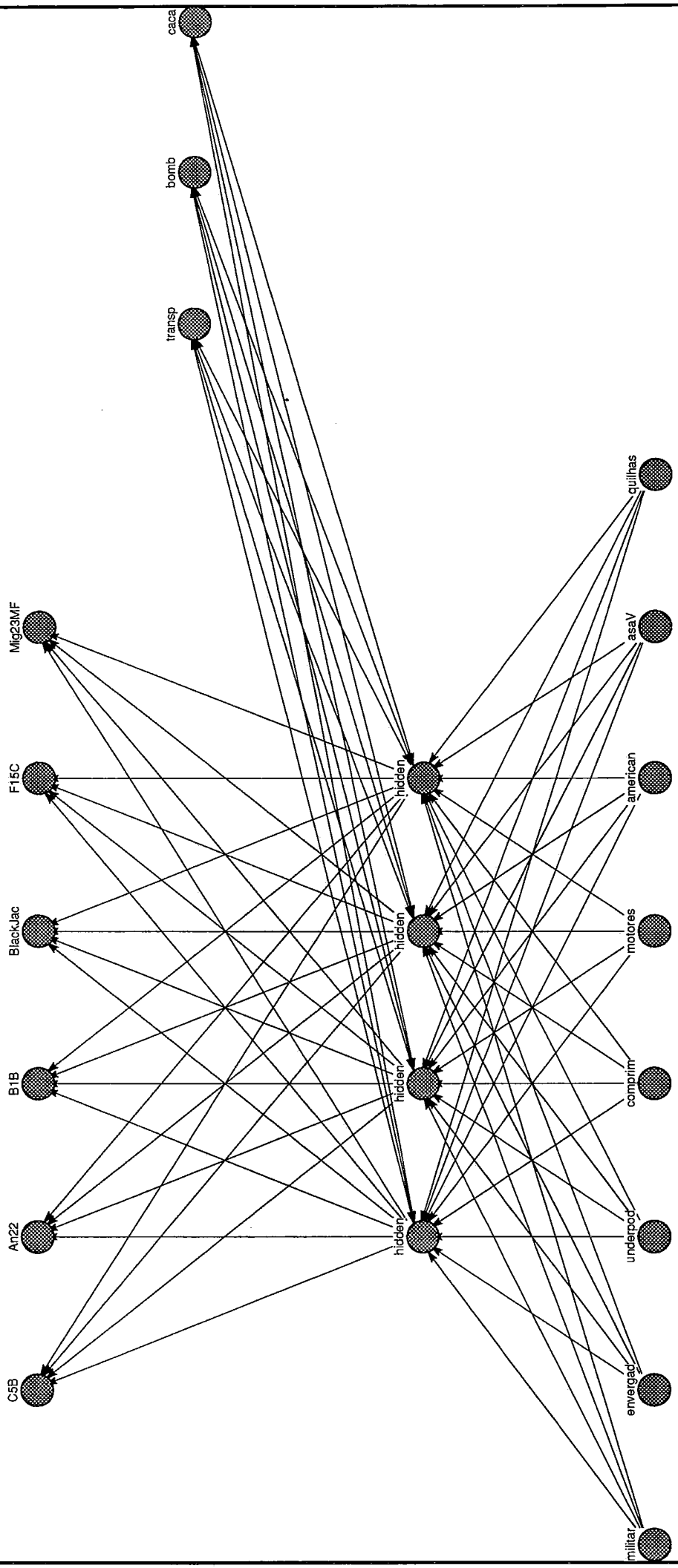


Figura VII.11 - Hierarquia do Exemplo (Telfer,91)

Nesta classificação podemos ver que foram utilizados conceitos intermediários (Transporte, Bombardeiro e Caça) a partir dos conceitos de entrada (Militar, Envergadura>20m, TanqueNasAsas, Comprimento>25m, Motores<3, Americano, Quilhas=1, AsasEmV) para determinar os conceitos de saída (C5B, An22, B1B, BlackJack, F15C e Mig23MF).

Para implementação deste exemplo na arquitetura proposta, associamos cada conceito a um elemento processador. Cada elemento da camada de entrada representa um conceito de entrada; cada elemento da camada de saída representa um conceito de saída; e cada elemento da camada intermediária representa um conceito intermediário.

Assim, a rede neural construída *(Figura VII.12)* tem 8 (oito) elementos de entrada, 6 (seis) elementos de saída e 3 (três) elementos intermediários.


caca
bomb
transp
Mig23MF
F15C
BlackJac
B1B
An22
C5B
hidden
hidden
hidden
hidden
hidden
quilhas
asaV
american
motores
comprim
underpod
envergad
militar

Para treinar uma rede neural normal (e.g. Back-Propagation) é preciso levantar os exemplos completos do mapeamento entre os conceitos de entrada e saída. Para treinar a arquitetura proposta é necessário incluir também os conceitos intermediários nos exemplos. Estes exemplos *(Tabela VII.6)* podem ser obtidos diretamente da hierarquia de conceitos *(Figura VII.10)*.

	1	2	3	4	5	6
Militar	1	1	1	1	1	1
Envergadura>20m	1	1	1	1	0	0
Tanque nas Asas	1	1	0	0	0	0
Comprimento>25m	1	1	1	1	0	0
Motores<3	0	0	0	0	1	1
Americano	1	0	1	0	1	0
Asas em V	0	0	1	1	0	1
Quilhas=1	1	0	1	1	0	1
Transporte	1	1	0	0	0	0
Bombardeiro	0	0	1	1	0	0
Caça	0	0	0	0	1	1
C5B	1	0	0	0	0	0
An22	0	1	0	0	0	0
B1B	0	0	1	0	0	0
BlackJack	0	0	0	1	0	0
F15C	0	0	0	0	1	0
Mig23MF	0	0	0	0	0	1

Tabela VII.6 - Exemplos de Treinamento

Para ensinar a rede foram utilizados os 6 (seis) exemplos fornecidos. Para testá-la foram utilizados os mesmos 6 (seis) exemplos mais 3 (três) outros *(Tabela VII.7)*, que servem para determinar se a rede aprendeu corretamente a hierarquia. Escolhemos para representar os conceitos não-disponíveis o valor 0.5 (valor médio entre 0 e 1).

	7	8	9
Militar	1	1	1
Envergadura>20m	1	0	0
Tanque nas Asas	1	1	0
Comprimento>25m	1	1	0
Motores<3	0	0	1
Americano	0.5	0.5	0.5
Asas em V	0.5	0.5	0.5
Quilhas=1	0.5	0.5	0.5
Transporte	1	0	0
Bombardeiro	0	1	0
Caça	0	0	1
C5B	0.5	0	0
An22	0	0	0
B1B	0	0	0
BlackJack	0	0.5	0
F15C	0	0	0.5
Mig23MF	0	0	0.5

Tabela VII.7 - Exemplos de Teste

A descrição da rede, os padrões de treinamento, o relatório do treinamento, os resultados do treinamento, os padrões de teste, o relatório dos testes e os resultados dos testes, padronizados pelo SNNS v3.1 podem ser encontrados no Anexo C.

Capítulo VIII

VIII - EPÍLOGO

VIII.1 - Conclusões

A caracterização de parte das aplicações práticas de Inteligência Artificial nos Sistemas Elétricos de Potência como problemas de Classificação permitiu concentrar esforços no estudo de Redes Neurais para Classificação.

Depois, o estudo do paradigma conexionista mostrou que as Redes Neurais não se limitam à Back-Propagation e que cada modelo possui características peculiares tornando-o mais adequado à resolução de uma classe de problemas.

Na resolução de quaisquer problemas reais é particularmente interessante, para o especialista, que os conceitos envolvidos apresentem uma representação externa localizada e que a rede neural seja capaz de induzir uma representação interna distribuída do mapeamento entre eles, como ocorre nas redes Back-Propagation. Creditamos o grande sucesso da Back-Propagation a esta capacidade.

Uma outra conclusão interessante é que a maioria absoluta dos modelos para Classificação preocupam-se, na realidade, com Categorização. Para resolver problemas reais de Classificação é necessária, então, a construção de Redes Neurais Hierárquicas. Este tipo de rede envolve a criação de uma topologia particular para cada problema. A arquitetura proposta permite a utilização de uma topologia única para a resolução de um problema qualquer.

A conclusão mais negativa da tese foi a de que uma das principais limitações da aplicação de Redes Neurais em problemas reais é a necessidade de padrões completos de entrada e saída. O processo de aquisição do conhecimento tem sido dificultado pela necessidade de um entendimento completo do problema em questão *(Coleman,92)*.

A conclusão mais positiva é que este gargalo pode ser suplantado através de novos mecanismos de aprendizado. Dentre as diferentes formas de representação dos conceitos como dados, acreditamos que a representação ternária é a mais promissora para futura utilização em exemplos incompletos.

Para que uma Rede Neural seja adequada à resolução de problemas de classificação é preciso que satisfaça um conjunto de requisitos: possuir aprendizado supervisionado, ser hetero-associativa e ser recursiva. Poucos modelos atuais satisfazem estes requisitos e, entre estes, os mais promissores são aqueles que possuem uma topologia parcialmente recursiva.

Os modelos com esta topologia provêm a Rede Neural de memória *(Capítulo VII)*. A arquitetura proposta acrescenta a esta topologia a possibilidade de determinar externamente a memória desejada.

Os testes realizados demonstraram a viabilidade de utilização da arquitetura proposta, em conjunto com o algoritmo "Back-Propagation Through Time", na resolução de problemas de Classificação.

A principal conclusão desta tese é que não devemos nos limitar a aplicar as tecnologias existentes em nossos problemas . É preciso entendê-las em sua essência, de modo a conseguir explorar ao máximo suas características.

VIII.2 - Sugestões

Durante o desenvolvimento deste trabalho surgiram interessantes sugestões para sua complementação:

- Embora os dados possam ser binários, ternários, discretos ou contínuos, os modelos existentes de elementos processadores somente são binários ou contínuos. A capacidade de representação da ausência de evidência apresentada pelos dados ternários motiva o desenvolvimento de um modelo ternário de elemento processador *(Capítulo VI)*.

- Para que este elemento processador ternário *(Figura VI.2)* pudesse ser efetivamente utilizado seria necessário desenvolver um algoritmo de aprendizado capaz de explorar suas características.
- O desenvolvimento de um modelo de Rede Neural utilizando a arquitetura sugerida e o algoritmo proposto acima é uma óbvia continuação deste trabalho.
- *Sandewall (86)* diz que é relativamente fácil obter sistemas de múltipla herança sem exceções e sistemas simples com exceções, porém combinar ambos se torna extremamente complicado. Sistemas de múltipla herança com exceções nada mais são que problemas de classificação. Para testar o modelo acima proposto sugerimos verificar como a rede responderia aos tipos de heranca relatados por Sandewall.
- O Centro de Pesquisas de Energia Elétrica (CEPEL) faz parte do Grupo Eletrobrás, que é o orgão governamental responsável pelo sistema elétrico de potência do Brasil. Um dos principais projetos desenvolvidos atualmente pelo CEPEL é um sistema digital de supervisão e controle para usinas e subestações de energia elétrica, que é um sistema distribuído em que alguns dos diversos processadores estão reunidos, constituindo um Centro de Supervisão e Controle, denominado Sistema Portável de Supervisão e Controle (SPSC). Os seus processadores independentes mantêm consistentes um banco de dados replicado para cada processador. Os aplicativos são responsáveis pelo processamento destes dados e pela Interface Homem-Máquina do sistema. Como vimos, para que se possa utilizar efetivamente a Inteligência Artificial na resolução de problemas do setor elétrico, é necessário que os projetos a serem desenvolvidos se integrem aos sistemas digitalizados de supervisão e controle já existentes e em funcionamento. Assim, nossa principal sugestão é o desenvolvimento de um sistema prático utilizando a teoria desenvolvida nesta tese, preferencialmente através da implementação um aplicativo que utilize técnicas de Inteligência Artificial para auxiliar o operador na resolução dos problemas apresentados, como a Detecção de

Falhas em Hidrogeradores. Este sistema poderia ser um sistema híbrido utilizando a arquitetura proposta como base de conhecimento.

VIII.3 - Perspectivas

A utilização de Redes Neurais em Representação do Conhecimento ainda é um campo pouco explorado. A maior parte das aplicações existentes utiliza somente representações localizadas de conhecimento, cuja topologia depende do problema em questão. Só recentemente as pesquisas com representações distribuídas de conhecimento começam a ter seus resultados mais divulgados e discutidos entre os pesquisadores da área. Ambas as formas de representação possuem seus defensores e contestadores, no entanto, acreditamos na potencialidade da utilização racional de ambas em um problema específico. O desenvolvimento do paradigma simbólico-conexionista trará grandes avanços para outras áreas de pesquisas, como Aprendizado de Máquina ("Machine Learning"), Sistemas Especialistas e Redes Neurais.

A tarefa mais difícil, mais demorada e mais cara na construção de um sistema especialista é construir, depurar e atualizar sua base de conhecimento *(Coleman,92)*. Com o advento dos pacotes comerciais de sistemas especialistas ("shells"), pode-se até dizer que esta é a única tarefa na construção de um sistema especialista, que motivou a criação da função de ***Engenheiro de Conhecimento***, cuja atribuição é justamente construir e depurar as bases de conhecimento a partir de entrevistas com o especialista.

As Redes Neurais podem ser utilizadas como bases de conhecimento de Sistemas Especialistas, dando origem aos sistemas conhecidos como ***Sistemas Especialistas Conexionistas***. As redes neurais da base de conhecimento podem ser construídas de exemplos de treinamento através de técnicas de aprendizado de máquina. Uma máquina de inferência de sistema especialista pode ser utilizada para interpretar estas bases de conhecimento, provavelmente utilizando Lógica Nebulosa.

Esperamos que, a partir do estudo teórico apresentado, possa ser desenvolvido um modelo prático para a utilização de Redes Neurais em Classificação e, um sistema

para a aquisição e manutenção do Conhecimento de especialistas nos Sistemas Elétricos de Potência baseado neste modelo.

Para aquisição de conhecimento a rede deve satisfazer ao requisito de adaptabilidade, de modo a tornar possível a sua utilização em problemas similares e ao de preservação do conhecimento, de modo a tornar possível a adaptação a modificações no problema sem perda do conhecimento adquirido. Tais requisitos somente podem ser alcançados através da utilização de redes neurais plásticas e evolutivas.

Finalmente, para que um sistema baseado em Redes Neurais pudesse realmente aquisitar e manter o conhecimento de um especialista, ele deveria ser capaz de não somente satisfazer os requisitos descritos, mas também de interagir com o próprio especialista de modo a adaptar continuamente seu funcionamento às necessidades do mesmo.

GLOSSÁRIO

Adaline (Adaptive Linear Neuron)**:** modelo de rede neural similar ao Perceptron, que utiliza um elemento processador cuja função de transferência é linear.

algoritmo de aprendizado: algoritmo responsável pelo aprendizado em uma rede neural.

aprendizado: mecanismo pelo qual a rede neural se adapta de modo a poder responder melhor a futuros estímulos.

aprendizado competitivo: mecanismo de aprendizado no qual os elementos processadores de uma mesma camada competem entre si para conquistar o direito de atualizarem o peso de suas conexões.

aprendizado Hebbiano: mecanismo de aprendizado no qual o elemento processador altera o peso de suas conexões baseado somente nas suas próprias entradas e saídas.

aprendizado não-supervisionado: mecanismo de aprendizado no qual a rede neural é capaz de se adaptar sem o auxilio de nenhum parâmetro externo.

aprendizado por reforço: mecanismo de aprendizado no qual apenas uma medida da adequação dos resultados obtidos é fornecida como parâmetro externo.

aprendizado supervisionado: mecanismo de aprendizado no qual os resultados a serem obtidos são fornecidos integramente para a rede neural.

árvore dendrítica: o mesmo que *dendritos*.

ausência de evidência: ocorre quando não existem evidências de que um fato seja verdadeiro ou falso.

auto-organização: capacidade que uma rede neural possui de determinar por si própria os pesos das conexões entre os elementos processadores através de um mecanismo de aprendizado não-supervisionado.

axônio: prolongamento da membrana plasmática de um neurônio responsável pela emissão dos impulsos elétricos, geralmente único para cada neurônio.

Back-Error Propagation: algoritmo de aprendizado utilizado no modelo Back-Propagation.

Back-Propagation: modelo de rede neural mais utilizado atualmente. Similar ao Perceptron, porém, sem o limite de duas camadas. Também como sinônimo do algoritmo *Back-Error Propagation.*

Back-Propagation Recursivo: generalização do algoritmo de aprendizado *Back-Error Propagation* para redes recursivas.

Back-Propagation Through Time: algoritmo de aprendizado para redes recursivas, que generaliza o *Back-Propagation* através de um processo de discretização no tempo.

camada: agrupamento de elementos processadores similares de uma rede neural. Normalmente existem pelo menos duas camadas em uma rede neural, uma para a apresentação dos dados de entrada e outra para obtenção dos dados de saída após o processamento.

camada de entrada: camada de elementos processadores na qual são apresentados os dados de entrada para uma rede neural.

camada de contexto: camada de elementos processadores na qual é armazenado o contexto de funcionamento da rede neural. Geralmente é uma cópia da camada de saída ou da camada escondida.

camada de saída: camada de elementos processadores na qual são obtidos os dados de saída resultantes do processamento de uma rede neural.

camada escondida: camada que não é utilizada para entrada nem saída de dados.

camada intermediária: camada que não é nem camada de entrada nem camada de saída de uma rede neural. Também é usada para denominar camadas que não sejam somente de entrada ou saída.

categorização: caso particular da classificação cujo objetivo é determinar a categoria a que pertence o padrão de entrada.

classificação: processo pelo qual novos conceitos são adicionados a uma taxonomia.

CNM (Combinatorial Neural Model): modelo de rede neural que utiliza representação localizada de conhecimento.

conexões: cada uma das ligações entre dois elementos processadores.

conhecimento: toda a informação necessária a resolução de um problema.

dendritos: prolongamentos em forma de árvore da membrana plasmática de um neurônio, também referido como árvore dendrítica, reponsável pela captação dos impulsos elétricos.

dígrafo: grafo direcionado, ou seja, grafo no qual as arestas possuem sentido, conectando um nó de origem à um nó de destino.

dígrafo acíclico: dígrafo que não possue ciclos (caminho circular de arestas).

elementos processadores: cada um dos elementos que compõem uma Rede Neural.

engenheiro de conhecimento: profissional responsável pela criação e manutenção das bases de conhecimento dos sistemas especialistas.

especialistas: pessoas capazes de tomar decisões a partir da lógica, heurística, experiência e até mesmo intuição.

estado de ativação da rede neural: conjunto dos valores de saída dos elementos processadores de uma rede neural em um determinado instante de tempo.

estado de ativação do elemento processador: valor de saída de um elemento processador em um determinado instante de tempo.

evidência negativa: evidência de que um fato é falso.

evidência positiva: evidência de que um fato é verdadeiro.

freqüências notáveis: conjunto das freqüências características de um equipamento.

função de ativação da rede neural: o mesmo que *estado de ativação da rede neural.*

função de saída: o mesmo que *função de transferência.*

função de transferência: cada elemento processador transfere para sua única saída o valor resultante da aplicação desta função sobre a combinação de suas entradas.

função degrau: função de transferência onde a saída recebe valor 1 caso a combinação (soma) seja maior que 0, e 0 em caso-contrário.

função linear: função de transferência onde a saída é proporcional à combinação (soma), ou seja, f(x)=ax+b.

função sigmoidal: função de transferência onde a saída é dada por uma sigmóide da combinação (soma), ou seja, $f(x)=1/(1+e^{(-x)})$.

função threshold: função de transferência onde a saída recebe valor 1 caso a combinação (soma) seja maior que um determinado valor (threshold), e 0 em caso-contrário.

grafo: forma de representação composta por nós (vértices) e arestas conectando estes nós. Graficamente, pode ser visualizado como um conjunto de pontos (vértices) ligados por semi-retas (arestas). Matematicamente, como uma tupla {V,A}, onde V representa um conjunto de elementos e A um conjunto de pares de elementos (v,w) de V.

Hipótese do Sistema de Símbolos Físicos: um sistema de símbolos físicos possui os meios necessários e suficientes para a ação inteligente geral.

Hopfield: modelo de rede neural, desenvolvido por Hopfield, no qual todos os elementos processadores estão amplamente conectados entre si.

inibição lateral: uma forma de aprendizado competitivo no qual o elemento processador vencedor inibe os elementos processadores adjacentes na camada.

inteligência artificial: estudo de como fazer os computadores realizarem tarefas em que, no momento, as pessoas são melhores.

lembrança (recall)**:** processo de utilização de uma rede neural.

Madaline (Multi-Layer Adalines): modelo de rede neural composto por um conjunto de elementos processadores Adaline conectados a um único elemento processador de saída.

Máquina de Boltzmann: modelo de rede neural recursiva baseada na rede de Hopfield, porém, com elementos processadores escondidos.

Multi-Layer Perceptrons: o mesmo que *Back-Propagation*.

neurônios: células altamente especializadas encontradas no cérebro, capazes de receber e transmitir impulsos elétricos.

otimização: problemas que tem por objetivo achar valores ótimos para um conjunto de variáveis.

padrão de ativação: conjunto dos estados da saída de cada um dos elementos processadores de uma redes neural.

padrão de interconexão: é o modo pelo qual cada um dos elementos processadores de uma Rede Neural se encontram ligados pelas suas conexões.

paradigma conexionista: paradigma que procura entender e emular as propriedades decorrentes do alto grau de paralelismo e conectividade.

paradigma simbolista: paradigma que procura entender e emular as propriedades decorrentes do processamento de símbolos.

paradigma simbólico-conexionista: linha de pesquisa que tenta combinar o paradigma simbolista com o paradigma conexionista.

PDP (parallel distributed processing): o mesmo que *processamento distribuído paralelo*.

Perceptrons: primeiro modelo de redes neurais. Modelo de duas camadas que utiliza elementos processadores cuja função de transferência é uma função threshold.

Perceptrons multi-camadas: o mesmo que *Back-Propagation*.

processamento distribuído paralelo: tipo de processamento no qual diversos processadores distribuídos realizam seu processamento independentemente dos demais e paralelamente. As redes neurais utilizam esta forma de processamento.

processamento simbólico conexionista: o mesmo que *paradigma simbólico-conexionista*.

peso: grau de conectividade entre dois elementos processadores, representado por um valor numérico em cada conexão.

Real-Time Recurrent Learning: generalização do algoritmo *Back Propagation Through Time* para aprendizado a cada intervalo de tempo.

rede neural: modelo que utiliza o paradigma conexionista para resolver um determinado tipo de problemas. Uma rede neural se compõe de um elevado número de elementos processadores amplamente conectados.

rede neural auto-associativa: rede neural em que os padrões de entrada são iguais aos padrões de saída.

rede neural com realimentação: rede neural cujo grafo apresenta ciclos.

rede neural dinâmica: rede neural em que os pesos das conexões podem variar.

rede neural estática: rede neural em que os pesos das conexões não variam.

rede neural hetero-associativa: rede neural em que os padrões de entrada são diferentes dos padrões de saída.

rede neural hierárquica: rede neural cuja topologia é uma organização hierárquica de sub-redes.

rede neural parcialmente recursiva: rede neural recursiva cujas conexões recursivas são cuidadosamente determinadas.

rede neural plástica: rede neural em que a topologia da rede pode variar.

rede neural recursiva: o mesmo que *rede neural com realimentação*.

rede neural rígida: rede neural em que a topologia da rede não varia.

rede neural sem realimentação: rede neural cujo grafo não apresenta ciclos.

rede semântica: grafo no qual os vértices representam objetos, conceitos e eventos, e os arcos representam relações entre os conceitos.

representação de conhecimento em redes neurais: associação de padrões de ativação dos neurônios da rede neural a conceitos externos.

representação externa de conhecimento: parte da representação de conhecimento que pode ser determinada externamente.

representação interna de conhecimento: parte da representação de conhecimento que pode ser determinada externamente.

representação localizada de conhecimento: associação de um conceito externo ao estado de ativação de um único neurônio.

representação distribuída de conhecimento: associação de um conceito externo ao estado de ativação de um conjunto de neurônios.

representação parcialmente distribuída de conhecimento: associação de um conceito externo ao estado de ativação de um sub-conjunto dos neurônios da rede.

representação totalmente distribuída de conhecimento: associação de um conceito externo ao estado de ativação de todos os neurônios da rede neural.

sinapse: aposição da membrana plasmática de dois neurônios, formando uma conexão pontual orientada de um neurônio para o outro, através da qual são realizadas as trocas de informação.

síntese: processo de inicialização do padrão de interconexão de uma rede neural.

sistemas elétricos de potência: sistemas para geração, transmissão e distribuição de energia elétrica.

sistemas especialistas: "framework" para o desenvolvimento de sistemas utilizando regras de produção.

sistemas especialistas conexionistas: sistemas especialistas que possuem redes neurais como base de conhecimento.

soma: corpo celular de um neurônio, onde se encontram a maioria de seus orgânulos.

spines: pequenas estruturas encontradas nas árvores dendríticas.

taxa de aprendizado: fator de atualização dos pesos das conexões em uma regra de aprendizado.

teste: processo de verificação do funcionamento correto de uma rede neural.

Time-Dependent Recurrent Back-Propagation: algoritmo de aprendizado para aprendizado "on-line" em intervalos contínuos de tempo.

topologia: o mesmo que *padrão de interconexão.*

treinamento: processo interativo de modificação do padrão de interconexão de uma rede neural, capacitando-a na resolução de um determinado problema.

unidades de processamento: o mesmo que *elementos processadores.*

REFERÊNCIAS BIBLIOGRÁFICAS

Almeida 87 A Learning Rule for Asynchronous Perceptrons with Feedback in a Combinatorial Environment
IEEE First ICNN, San Diego, 87

Anderson 89 A Theory of the Origins of Human Knowledge
J.R. Anderson
Artificial Intelligence, vol.40, no.1-3, September 1989

Anderson 90 Data Representation in Neural Networks
James A. Anderson
AI Expert, June 1990

Ayrosa 92 Representação do Conhecimento em Sistemas Conexionistas: Tópicos em Análise
Pedro Paulo da Silva Ayrosa
Tese M.Sc. - COPPE, UFRJ, Brasil, 1992

Bratko 93 Machine Learning in Artificial Intelligence
Ivan Bratko
Artificial Intelligence in Engeneering, 8 (1993) pp.159-164

Carbonell 89 Introduction: Paradigms for Machine Learning
J.G. Carbonell
Artificial Intelligence, vol.40, no.1-3, September 1989

Cardador 90 Representação de Conhecimento: Modelos Clássicos e Conexionistas
Cardador, D.M.
Tese de Mestrado - IME, 1990

Carvalho 89 Síntese de Redes Neuronais com Aplicações à Representação do Conhecimento e à Otimização
Luís Alfredo V. de Carvalho
Tese D.Sc. - COPPE - Rio de Janeiro, RJ - Brasil - 1989

Casanova - Programação em Lógica e a Linguagem Prolog
M.A. Casanova, F.A.C. Giorno & A.L. Furtado
Ed. Edgar Blücher Ltda.

Chan 90 Using Neural Network to Interpret Multiple Alarms
Edward H. P. Chan
IEEE Computer Applications in Power, April 1990

Chan 91 Interpretive Power System Alarm Processing Test Results
E.H.Chan & N.S.Markushevish & R.Adapa
First International Forum on Applications of Neural Networks to Power Systems - 1991

Chow 91 Application of Learning Theory to a Single Phase Induction Motor Incipient Fault Dectector Neural Network
M.Chow, G.L.Bilbro & S.O.Yee
IEEE 1991

Chow 92 Recognizing Animal-Caused Faults in Power Systems using Artificial Neural Networks
M.Chow, S.O.Yee & L.S.Taylor
IEEE 92 SM 505-8 PWRD

Cleeremans 89 Finite State Automata and Simple Recurrent Networks
A. Cleeremans, D. Servan-Schreiber & J.L. McClelland
Neural Computation, 1

Coleman 92 Neural Networks in Knowledge Acquisition
Kevin G.Coleman & Susan Watenpool
AI Expert, January 1992

Daniels 91 Power Quality Monitoring Using Neural Networks
R.F. Daniels
IEEE 1991

Dayhoff 90 Neural Network Architectures - An Introduction
Judith E. Dayhoff
Van Nostrand Reinhold - 1990

Denis 91 O Modelo Conexionista Evolutivo
F.A.R.M.Denis & R.J.Machado
IBM - Relatório Técnico CCR128
Centro Científico Rio - Rio de Janeiro, Setembro, 1991

de Kleer 90 Characterizing Diagnoses
J.de Kleer, A.K.Mackworth & R.Reiter
AAAI - 1990

de Kleer 92 Characterizing Diagnoses and Systems
J. de Kleer, A. K. Mackworth & R. Reiter
Artificial Intelligence, 56 pp.197-222, 1992

do Coutto 91 Um Sistema Especialista para Diagnóstico de Hidrogeradores
M.H.T.A.do Coutto, G.F.Guidacci da Silveira & H.R.T. de Azevedo
XI SNPTEE Outubro 1991

Elman 90 Finding Structure in Time
Jeffrey L. Elman
Cognitive Science, vol.14, no.2, pp.179-211

Fahlman 81 Representação Implicit Knowledge
Scott E. Fahlman
Parallel Models of Associative Memory
G.E.Hinton & J.A.Anderson
Ed. Lawrence Erlbaum Associates, 1981

Feigenbaum - The Handbook of Artificial Intelligence
Avron Barr & Edward A. Feigenbaum
HeurisTech Press - Stanford, California

Feser 91 Application of Neural Networks in Numerical Busbar Protections Systems (NBPS)
K.Feser & U.Braun
IEEE 1991

Fisher 89 Models of Incremental Concept Formation
John H. Gennari, Pat Langley & Doug Fisher
Artificial Intelligence, Vol.40, No.1-3, September 1989

Fukuyama 91 An Application of Artificial Neural Network to Dynamic Economic Load Dispatching
Y.Fukuyama & Y.Ueki
IEEE 1991

Gallant 88 Connectionist Expert Systems
S.I. Gallant
Communications of the ACM, 24 pp.152-169, 1988

Gingrich - Modeling Human Operators using Neural Networks
C.G. Gingrich, D.R. Kuespert & T.J. McAvoy
Chemical Engeneering Department - University of Maryland
(Unpublished Paper)

Grossberg 88 The ART of Adaptive Pattern Recognition by a Self-Organizing Neural Network
Gail A. Carpenter & Stephen Grossberg
IEEE Computer, March 1988 pp.77-88

Halmos 73 Teoria Ingênua dos Conjuntos
Paul R. Halmos
Editora Polígono - 1973

Hertz 90 Introduction to the Theory of Neural Computation
John Hertz, Anders Krogh & Richard Palmer
Volume I
Addison-Wesley Publishing Company

Hinton 86 Learning and Relearning in Boltzmann Machines
G.E.Hinton & T.J.Sejnowski
Parallel Distributed Processing - vol.1
David E. Rumelhart & James L. McClelland
The MIT Press - Cambridge, Massachusetts - 1986

Hinton 81 Parallel Models of Associative Memory
Geoffrey E. Hinton & James A. Anderson
Lawrence Erlbaum Associates, New Jersey - 1981

Hopfield 82 Neural Networks and Physical Systems with Emergent Collective Computacional Abilities
J.J.Hopfield
Proc. Natl. Acad. Sci. USA - Vol.79, pp.2554-2558, April 82

Hopfield 86 Computing with Neural Circuits: A Model
John J. Hopfield & David W. Tank
Science - Vol.233, pp.625-633, August 86

Hopfield 82 Neural Networks and Physical Systems with Emergent Collective Computacional Abilities
John J. Hopfield
Proceedings of Natural Academic Sciences, vol.79, April 1982

Hopfield 86 Computing with Neural Circuits: A Model
John J. Hopfield & David W. Tank
Science, vol.233, pp.625-633, August 1986

Horty 90 A Skeptical Theory of Inheritance in Non-Monotonic Semantic Networks
John F. Horty, Richmond H. Thompson & David Touretzky
Artificial Inteligence, vol.42, no.2-3, pp.311-348, March 1990

Jarayama 91 Neural Net Based Correction of Power System Distortion caused by Switching Power Supplies
B.Jarayama & K.Ashenayi & M.O.Durham & R.D.Strattan
IEEE 1991

Jordan 89 Generic Constraints on Underspecified Target Trajectories
Michael I. Jordan
IJCNN'89, pp.217-225

Kim 93 Artificial Neural-Network based Feeder Reconfiguration for Loss Reduction in Distribution Systems
H.Kim, Y.Ko & K.Jung
IEEE Transactions on Power Delivery - vol.8, no.3, July 1993

Koch - Biophysics of Computation: Toward the Mechanisms Underlying Information Processing in Single Neurons
Christof Koch

Konolige 92 Abduction versus Closure in Causal Theories
Kurt Konolige
Artificial Inteligence, no.53 pg.255-272

Kosko 87 Adaptive Inference in Fuzzy Knowledge Networks
Bart Kosko
First IJCNN, San Diego 1987

Kraft 91 A Hybrid Neural Network and Expert System for Monitoring Fossil Fuel Power Plants
T.Kraft, A.Brandon et all.
IEEE 1991

Latimer 77 Design of a Dispatcher Training System
J.R.Latimer, R.D.Masiello
Power Industry Computer Applications Conference 1977

Leshno 93 Multilayer Feedforward Networks With a Nonpolynomial Activation Function Can Approximate Any Function
M.Leshno, V.Y.Lin, A.Pinkus & S.Schocken
Neural Networks, vol.6, pp.861-867,1993

Levine 91 Introduction to Neural & Cognitive Modeling
Daniel S. Levine
Lawrence Erlbaum Associates, Inc. - 1991

Liu 92 A Progress Report on Practical Use of Expert Systems in Planning and Operation
C.C. Liu & al.
CIGRÉ 1992 Session - Task Force 38.03.10

Lubkeman 91 Unsupervised Learning Strategies for the Detection and Classification of Transient Phenomena on Electric Power Distribution Systems
D.L.Lubkeman & C.D.Fallon & A.A.Girgis
IEEE 1991

Lukaszewicz 90 Non-Monotonic Reasoning
Formalization of Commonsense Reasoning
Witold Lucaszewicz
Ed. Ellis Horwood, 1990

Machado 88 Calculating the Mean Knowledge Representation from Multiple Experts
Ricardo J. Machado & Armando F. da Rocha
IBM Technical Report CCR062
Centro Científico Rio, IBM, Junho 1988

Machado 89 Handling Knowledge in High Order Neural Networks:
The Combinatorial Neural Model
R.J. Machado & A.F. da Rocha
IBM Technical Report CCR076
Centro Científico Rio, IBM, Maio 1989

Machado 90 Handling Knowledge in High Order Neural Networks:
The Combinatorial Neural Model
R.J. Machado, A.F. da Rocha & B.F.Leão
Multiperson Decision Making Using Fuzzy Sets and Possibility Theory
Kluwer Academic Publishers - Netherlands, 1990

Machado 91 NEXT - The Neural Expert Tool
R.J. Machado, V.H.A.Duarte, F.A.R.M.Denis, A.F. da Rocha & M.P.Ramos
IBM Technical Report CCR120
Centro Científico Rio, IBM, Maio 1991

Machado 92a Evolutive Fuzzy Neural Networks
R.J. Machado & A.F. da Rocha
Proceedings of the IEEE International Conference on Fuzzy Systems, San Diego, March 1992

Machado 92b Incremental Learning in Fuzzy Neural Networks
R.J.Machado, C.Ferlin, A.F. da Rocha & G.J.Erthal
Proceedings of the IEEE International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, Palma de Malorca, July 1992

Machado 92c NEXTool: An Enviroment for Connectionist Expert Systems
R.J.Machado, C.Ferlin & A.F. da Rocha
Third Annual Symposium of the International Association of Knowledge Engineers, Washington, November, 1992

Machado 92d
A Hybrid Architecture for Fuzzy Connectionist Expert Systems
R.J.Machado & A.F. da Rocha
Hybrid Architectures for Intelligent Systems
A.Kandel & G.Langhola
CRC Press, 1992

Machado 93
Inference, Inquiry and Explanation in Expert Systems by means of Fuzzy Neural Networks
R.J.Machado & A.F. da Rocha
Proceedings of the Second IEEE International Conference on Fuzzy Systems, San Francisco, March 1993

Manna 74
Mathematical Theory of Computation
Zohar Manna
McGrawHill, 1974

Martino 90
Representação de Conhecimento Implícito em Redes Neuronais
Marcello Baptista de Martino
Monografia de Redes Neuronais - COPPE/UFRJ - Agosto 1990

Mellor 88
Object Oriented Systems Analysis
(Modeling the World in Data)
Stephen J. Mellor / Sally Shlaer
Yourdon Press,1988

Minsky 89
A Sociedade da Mente
Marvin Minsky
Ed. Francisco Alves, 1989

Neily 91
Joint VAR Controller Implemented in an Artificial Neural Network Enviroment
G.Neily , R.Barone , G.Josin & D.Charney
IEEE 1991

Newell 76
Computer Science as Empirical Inquiry: Symbols and Search
A. Newell & H.A. Simon
Communications of the ACM, Vol.19, No.3, March 1976

Novosel [07] 91
Identification of Power System Emergency Actions using Neural Networks
D. Novosel & R.L.King
IEEE 1991

Nute 88
Defeasible Reasoning: A Philosophical Analysis in Prolog
Donald Nute
Aspects of Artificial Intelligence, pp.251-288
James H. Fetzer
Ed. Kluwer Academic Publishers, 1988

Nute 88 Defeasible Reasoning and Decision Support Systems
Donald Nute
Decision Support Systems, 4 (1988)

Nute 91 Basic Defeasible Logic
Donald Nute
(to appear in)
Intensional Logics for Programming
Farinas-del-Cerro & Pentonnen
Oxford, 1991

Pearlmutter 89 Learning State Space Trajectories in Recurrent Neural Networks
B.A. Pearlmutter
IEEE ICNN' 89, vol.II, pp.365-372

Peng 92 An Adaptive Neural Network Aproach to One-Week-Ahead Load Forecasting
T.M.Peng, N.F.Hubele & G.G.Karady
IEEE 92 SM 407-7 PWRD

Peng 92 An Adaptive Neural Network Approach to One-Week Ahead Load Forecasting
T.M. Peng, N.F. Hubele & G.G. Karady
IEEE/PES Summer Meeting - July 1992

Pessoa 90 Aprendizado Não-Supervisionado em Redes Neuronais
Luiz Adauto F. C. Pessoa
Tese M.Sc., COPPE-UFRJ, Julho 1990

Pineda 89 Recurrent Back-Propagation and the Dynamical Approach to Adaptive Neural Computation
F.J. Pineda
Neural Computation, vol.1, pp.161-172

Poole 87 Theorist: A Logical Reasoning System for Defaults and Diagnosis
D.Poole, R.Goebel & R.Aleliunas
The Knowledge Frontier: Essays in the Representation of Knowledge
N.Cercone & G.McCalla (Eds.)
Springer-Varlag, New York, 1987

Poole 88 Representing Knowledge for Logic-based Diagnosis
David Poole
Proceedings of the International Conference on Fifth Generation Computer Systems, 1988 - pp.1282-1290

Poole 88 A Logical Framework for Default Reasoning
David Poole
Artificial Intelligence, 36 pp.27-47, 1988

Poole - Normality and Faults in Logic-Based Diagnosis
David Poole

Rall 92 Cable Theory for Dendritic Neurons
Wilfrid Rall

Ramani 91 Fish Identification from Sonar Echoes - Preprocessing and Parallel Networks
N.Ramani & W.G.Hanson & P.H.Patrick & H.Anderson
IEEE 1991

Reiter 87 A Theory of Diagnosis from First Principles
Raymond Reiter
Artificial Intelligence, 32 pp.57-95, 1987

Reiter - Non-Monotonic Reasoning
Raymond Reiter
(to appear in)
Annual Reviews of Computer Science

Rich 88 Inteligência Artificial
Elaine Rich
McGraw-Hill - São Paulo - 1988

Ripper 89 Medições de Vibrações de Eixo em Unidade Hidrogeradora
A.P.Ripper , E.Freire & L.P.Nascimento
3ºERLAC - Foz do Iguaçu - Brasil -1989

Rocha 92 A Neural Net for Extracting Knowledge from Natural Language Data Bases
A.F.Rocha, J.R.Guilherme, A.M.K.Miyadahira & M.S.Koizumi
IEEE Transactions on Neural Networks, vol.3, no.1, January 1992

Rohwer 87 Training Time-Dependence in Neural Networks
R. Rohwer & B. Forrest
IEEE First ICNN, San Diego, 1987, vol. II, pp.701-708

Ronne-Hansen 91 Neural Network as a Tool for Unit Commitment
P.Ronne-Hansen & J.Ronne-Hansen
IEEE 1991

Rumelhart 86 Parallel Distributed Processing
Explorations in the Microstructure of Cognition
David E. Rumelhart & James L. McClelland
The MIT Press - Cambridge, Massachusetts - 1986

Rumelhart 86a Feature Discovery by Competitive Learning
D.E.Rumelhart & D.Zipser
Parallel Distributed Processing - Vol.1
David E. Rumelhart & James L. McClelland
The MIT Press - Cambridge, Massachusetts - 1986

Saitoh 91 Application of Neural Network Based Fuzzy Control to Power System Generator
Kensuke Saitoh & Shinichi Iwamoto
91TH0374-9/91/0000-0144$01.00@1991 IEEE

Sakagushi 87 Prospects of Expert Systems in Power System Operations
T.Sakagushi et al.
9th. Power Systems Computation Conference - September 1987

Sandewall 86 Non-Monotonic Inference Rules for Multiple Inheritance with Exceptions
Erik Sandewall
Proceedings of the IEEE, vol.74, no.10, October 1986

Santos 79 A Experiência do CEPEL no Diagnóstico de Falhas em Transformadores por Análise Cromatográfica de Gases Dissolvidos
J. C. dos Santos & C.L.C. Sobral Vieira
IV SNPTEE, Recife, 1979

Sasaki 91 A Solution of Generation Expansion Problem by means of Neural Network
H.Sasaki , J.Kubokawa , M.Watanabe , R.Yokoyama & R.Tanabe
IEEE 91

Schulte 87 Artificial Intelligence Solutions to Power System Operating Problems
R.P.Schulte, S.L.Larsen, G.B.Sheble, J.N.Wrubel & B.F.Wollenberg
IEEE Transactions on Power Systems, Vol. PWRS-2, No.4, November 1987

Shepherd 78 Microcircuits in the Nervous System
Gordon M. Shepherd
Scientific American, vol.238, no.2, February 1978

Shepherd 90 Introduction to Synaptic Circuits
Gordon M. Shepherd and Christof Koch
The Synaptic Organization of the Brain
Oxford University Press, 1990

Shepherd - The Significance of Real Neuron Architectures for Neural Network Simulations
Gordon M. Shepherd

Shepherd 89 Apical Dendritic Spines of Cortical Pyramidal Cells: Remarks on their Possible Roles in Higher Brain Functions, Including Memory
Gordon M. Shepherd
Synapses, Circuits, and the Beggining of Memory
Gary Lynch

Smolensky 86 Neural and Conceptual Interpretation of PDP Models
P. Smolensky
Parallel Distributed Processing, vol.2, chapter 22

Stein 92 Resolving Ambiguity in Non-Monotonic Inheritance Hierarchies
Lynn Andrea Stein
Artificial Intelligence, 55 (1992) pp.259-310

Sugeno - Fuzzy Systems Theory and Its Applications
T. Terano, K. Asai & M. Sugeno
Academic Press Inc. - Boston

Sun 94 Ron Sun
Bibliography of Connectionist Models with Symbolic Processing
(Internet avaiable archive)

Sundholm 83 Systems of Deduction
Göran Sundholm
Handbook of Philosophical Logic, vol.I, pp.133-188
D. Gibbay & F. Guenthner
Ed. D.Reidel Publishing Company, 1983

Susumago 86 Development of a Large-Scale Dispatcher Training Simulator and Training Results
I.Susumago, M.Suzuki, K.Miyama, T. Tsuji, K.Dan, A.Yamanishi
IEEE Transactions on Power Systems, Vol. PWRS-1, No.2, May 1986

Tamura 89 An International Survey of the Present Status and the Perspective of Expert Systems on Power System Analysis and Technique
Y. Tamura
CIGRÉ - Symposium Bournemouth 1989

Telfer 91 Neural Closure Associative Processor
Brian Telfer & David Casasent
Neural Networks, vol.4, pp.589-598, 1991

Thomas 91 Dynamical Implications of Using Neural Networks as Controllers
R.J.Thomas & E.Sakk
IEEE 1991

Tokyo - The Study of Artificial Intelligence for Power Systems
The Tokyo Eletric Power Co., Inc.

Tomsovic 87 An Expert System as a Dispatchers' Aid for the Isolation of Line Section Faults
K.Tomsovic, C. Liu, P.Ackerman & S.Pope
IEEE Transactions on Power Delivery, vol.2, no.3, July 1987

Tomsovic 93 A Fuzzy Information Approach to Integrating Different Transformer Diagnostic Methods
K.Tomsovic, M.Tapper & T.Ingvarsson
IEEE Transactions on Power Delivery, vol.8, no.3, July 1993

Vadari 91 A Hybrid Artificial Neural Network / Artificial Intelligenge Approach for Voltage Stability Enhancement
S.V.Vadari & S.S.Venkata
IEEE 1991

Valiquette 90 An Expert System Based Diagnosis and Advisor Tool for Teaching Power System Operation Strategies
B.Valiquette, G.L.Torres & D.Mukhedvar
IEEE 90 SM 271-7 PWRS

Wasserman 89 Neural Computing: Theory and Practice
Philip D. Wasserman
Van Nostrand Reinhold, New York, 1989

Wildeberger 92 Overview of Neural Network Projects at the Eletric Power Research Institute
A. Martin Wildeberger
SIMTEC'92 / WNN'92 Houston, Texas, November 1992

Wollenberg 87 Artificial Intelligence in Power Systems Operations
Bruce F. Wollenberg & Toshiaki Sakagushi
Proceedings of the IEEE, vol.75, no.12, December 1987

Woods 91 Understanding Subsumption and Taxonomy:
A Framework for Progress
W.A. Woods
Principles of Semantic Networks:
Explorations in the Representation of Knowledge
John F. Sowa, ed.
Morgan Kaufmann Publishers, Inc. - 1991

Wu 91 On-Line Training of Neural Network Model and Controller for Turbogenerators
Q.H.Wu & B.W.Hogg & G.W. Irwin
IEEE 1991

Wu 92 A Neural Network Regulator for Turbogenerators
Q.H.Wu & B.W.Hogg & G.W. Irwin
IEEE Transactions on Neural Networks, vol.3, no.1, January 1992

Zadeh 87 Fuzzy Sets and Applications
L. A. Zadeh
John Wiley & Sons - New York

Zipser 89 Encoding Sequential Structure:
Experience with the Real-Time Recurrent Learning Algorithm
D. Zipser & A.W. Smith
IEEE ICNN'87, vol.I, pp.645-648

Anexo A

Este anexo transcreve uma discussão ocorrida entre conexionistas, pela Internet, sobre a utilização de redes neurais com dados incompletos.

```
From prechelt@ira.uka.de Mon Mar 21 16:30:46 1994
Return-Path: <prechelt@ira.uka.de>
Received: from irafs1.ira.uka.de by cos.ufrj.br (4.1/SMI-4.1)
        id AA04127; Mon, 21 Mar 94 16:29:32 EST
Received: from irafs2.ira.uka.de (actually irafs2) by irafs1 with SMTP (PP);
          Mon, 21 Mar 1994 16:51:59 +0000
Received: from i41s25.ira.uka.de by irafs2.ira.uka.de with SMTP (PP)
          id <09534-0@irafs2.ira.uka.de>; Mon, 21 Mar 1994 16:55:17 +0100
To: martino@rio.cos.ufrj.br (Marcello Baptista de Martino)
Subject: Re: Incomplete data
In-Reply-To: Your message of "Mon, 21 Mar 1994 12:47:46 EST."
<9403211547.AA02451@cos.ufrj.br>
Date: Mon, 21 Mar 1994 16:55:23 +0100
From: Lutz Prechelt <prechelt@ira.uka.de>
Message-Id: <"irafs2.ira.536:21.02.94.15.55.19"@ira.uka.de>
Status: R


Subject: SUMMARY: encoding missing values


A few days ago, I posted some thoughts about how to represent missing
input values to a neural network and asked for comments and further ideas.
This message is a summary of the replies I received (some in my personal mail
some in connectionists).  I show the most significant comments and ideas
and append versions of the messages that are trimmed to the most important parts
(in case somebody wants to keep this discussion in his/her archive)

This was my original message:

------------------------------------------------------------------------


Subject: Encoding missing values
Date: Wed, 02 Feb 1994 09:58:56 +0100
From: Lutz Prechelt <prechelt@ira.uka.de>

I am currently thinking about the problem of how to encode data with
attributes for which some of the values are missing in the data set for
neural network training and use.

An example of such data is the 'heart-disease' dataset from the UCI machine
learning database (anonymous FTP on "ics.uci.edu" [128.195.1.1], directory
"/pub/machine-learning-databases"). There are 920 records altogether with
14 attributes each. Only 299 of the records are complete, the others have one
or several missing attribute values.  11% of all values are missing.

I consider only networks that handle arbitrary numbers of real-valued inputs
here (e.g. all backpropagation-suited network types etc). I do NOT consider
missing output values. In this setting, I can think of several ways how to
encode such missing values that might be reasonable and depend on
the kind of attribute and how it was encoded in the first place:

1. Nominal attributes (that have n different possible values)
  1.1 encoded "1-of-n", i.e., one network input per possible value,
      the relevant one being 1 all others 0.
      This encoding is very general, but has the disadvantage of producing
      networks with very many connections.
      Missing values can either be represented as 'all zero' or by simply
      treating 'is missing' as just another possible input value, resulting
      in a "1-of-(n+1)" encoding.
  1.2 encoded binary, i.e.,  log2(n) inputs being used like the bits in a
    binary representation of the numbers 0...n-1 (or 1...n).
      Missing values can either be represented as just another possible input
      value (probably all-bits-zero is best) or by adding an additional network
      input which is 1 for 'is missing' and 0 for 'is present'. The original
      inputs should probably be all zero in the 'is missing' case.

2. continuous attributes (or attributes treated as continuous)
  2.1 encoded as a single network input, perhaps using some monotone
      transformation to force the values into a certain distribution.
      Missing values are either encoded as a kind of 'best guess' (e.g. the
      average of the non-missing values for this attribute) or by using
      an additional network input being 0 for 'missing' and 1 for 'present'
      (or vice versa) and setting the original attribute input either to 0
      or to the 'best guess'. (The 'best guess' variant also applies to
      nominal attributes above)

3. binary attributes (truth values)
  3.1 encoded by one input:  0=false  1=true   or vice versa
      Treat like (2.1)
```

```
  3.2 encoded by one input:  -1=false 1=true   or vice versa
      In this case we may act as for (3.1) or may just use 0 to
      indicate 'missing'.
  3.3 treat like nominal attribute with 2 possible values

4. ordinal attributes (having n different possible values, which are ordered)
  4.1 treat either like continuous or like nominal attribute.
    If (1.2) is chosen, a Gray-Code should be used.
    Continuous representation is risky unless a 'sensible' quantification
    of the possible values is available.

So far to my considerations. Now to my questions.

a) Can you think of other encoding methods that seem reasonable ?  Which ?

b) Do you have experience with some of these methods that is worth sharing ?

c) Have you compared any of the alternatives directly ?

------------------------------------------------------------------------

SUMMARY:

For a), the following ideas were mentioned:
1. use statistical techniques to compute replacement values from the rest
   of the data set
2. use a Boltzman machine to do this for you
3. use an autoencoder feed forward network to do this for you
4. randomize on the missing values (correct in the Bayesian sense)

For b), some experience was reported. I don't know how to summarize
that nicely, so I just don't summarize at all.

For c), no explicit quantitative results were given directly.

Some replies suggest that data is not always missing randomly.
The biases are often known and should be taken into account (e.g.
medical tests are not carried out (resulting in missing data) for
moreless healthy persons more often than for ill persons).

Many replies contained references to published work on this area,
from NN, machine learning, and mathematical statistics.
To ease searching for these references in the replies below, I have
marked them with the string ##REF## (if you have a 'grep' program that
extracts whole paragraphs, you can get them all out with one command).

Thanks to all who answered.
These are the trimmed versions of the replies:

------------------------------------------------------------------------

From: tgd@research.CS.ORST.EDU (Tom Dietterich)

[...for nominal attributes:]
An alternative here is to encode them as bit-strings in a
error-correcting code, so that the hamming distance between any two
bit strings is constant.  This would probably be better than a dense
binary encoding.  The cost in additional inputs is small.  I haven't
tried this though.  My guess is that distributed representations at
the input are a bad idea.

One must always determine WHY the value is missing.  In the heart
disease data, I believe the values were not measured because other
features were believed to be sufficient in each case.  In such cases,
the network should learn to down-weight the importance of the feature
(which can be accomplished by randomizing it---see below).

In other cases, it may be more appropriate to treat a missing value as
a separate value for the feature, e.g., in survey research, where a
subject chooses not to answer a question.

[...for continuous attributes:]
Ross Quinlan suggests encoding missing values as the mean observed
output value when the value is missing.  He has tried this in his
regression tree work.

Another obvious approach is to randomize the missing values--on each
presentation of the training example, choose a different, random,
value for each missing input feature.  This is the "right thing to do"
in the bayesian sense.
```

[...for binary attributes:]
I'm skeptical of the -1,0,1 encoding, but I think there is more
research to be done here.

[...for ordinal attributes:]
I would treat them as continuous.

From: shavlik@cs.wisc.edu (Jude W. Shavlik)

We looked at some of the methods you talked about in
the following article in the journal Machine Learning.

##REF##
%T Symbolic and Neural Network Learning Algorithms: An Experimental Comparison
%A J. W. Shavlik
%A R. J. Mooney
%A G. G. Towell
%J Machine Learning
%V 6
%N 2
%P 111-143
%D 1991

From: hertz@nordita.dk (John Hertz)

It seems to me that the most natural way to handle missing data is to
leave them out. You can do this if you work with a recurrent network
(fx Boltzmann machine) where the inputs are fed in by clamping the
input units to the given input values and the rest of the net relaxes
to a fixed point, after which the output is read off the output units.
If some of the input values are missing, the corresponding input units
are just left unclamped, free to relax to values most consistent with
the known inputs.

I have meant for a long time to try this on some medical prognosis data
I was working on, but I never got around to it, so I would be happy to
hear how it works if you try it.

From: jozo@sequoia.WPI.EDU (Jozo Dujmovic)

In the case of clustering benchmark programs I frequently have
the the problem of estimation of missing data. A relatively
simple SW that implements a heuristic algorithm generates
estimates having the average error of 8%. NN will somehow
"implicitly estimate" the missing data. The two approaches
might even be in some sense equivalent (?).

Jozo

[I suspect that they are not: When you generate values for the
missing items and put them in the training set, the network loses the
information that this data is only estimated. Since estimations are
not as reliable as true input data, the network will weigh inputs that
have lots of generated values as less important. If it gets the 'is
missing' information explicitly, it can discriminate true values from
estimations instead.]

From: guy@cs.uq.oz.au

A final year student of mine worked on the problem of dealing with missing
inputs, without much success. However, the student as not very good, so take
the following opinions with a pinch of salt.

We (very tentatively) came to the conclusion that if the inputs were
redundant, the problem was easy; if the missing input contained vital
information, the problem was pretty much impossible.

We used the heart disease data. I don't recommend it for the missing inputs
problem. All of the inputs are very good indicators of the correct result,
so missing inputs were not important.

Apparently there is a large literature in statistics on dealing with missing inputs.

Anthony Adams (University of Tasmania) has published a technical report on this. His email address is "A.Adams@cs.utas.edu.au".

##REF##
@techreport{kn:Vamplew-91,
   author = "P. Vamplew and A. Adams",
   address = {Hobart, Tasmania, Australia},
   institution = {Department of Computer Science, University of Tasmania},
   number = {R1-4},
   title = {Real World Problems in Backpropagation: Missing Values and Generalisability},
   year = {1991}
}

From: Mike Southcott <mlsouth@cssip.levels.unisa.edu.au>

##REF##
I wrote a paper for the Australian conference on neural networks in 1993.
``Classification of Incomplete Data using neural networks''
Southcott, Bogner.
.
I have stuck it on our anonymous ftp machine,
just ftp to ftp.cssip.edu.au and you'll find two versions of the paper in pub/users/michael

They are both from the Macintosh, so my postscript viewer (Ghostview) had problems viewing them, but they printed out OK.

From: Eric Saund <saund@parc.xerox.com>

I have done some work on unsupervised learning of mulitple cause clusters in binary data, for which an appropriate encoding scheme is -1 = FALSE, 1 = TRUE, and 0 = NO DATA. This has worked well for me, but my paradigm is not your standard feedforward network and uses a different activiation function from the standard weighted sum followed by sigmoid squashing.
I presented the paper on this work at NIPS:

##REF##
Saund, Eric; 1994; "Unsupervised Learning of Mixtures of Multiple Causes in Binary Data," in Advances in Neural Information Processing Systems -6-, Cowan, J., Tesauro, G, and Alspector, J., eds. Morgan Kaufmann, San Francisco.

From: Thierry.Denoeux@hds.univ-compiegne.fr

In a recent mailing, Lutz Prechelt mentioned the interesting problem of how to encode attributes with missing values as inputs to a neural network.
I have recently been faced to that problem while applying neural nets to rainfall prediction using weather radar images. The problem was to classify pairs of "echoes" -- defined as groups of connected pixels with reflectivity above some threshold -- taken from successive images as corresponding to the same rain cell or not. Each pair of echoes was discribed by a list of attributes. Some of these attributes, refering to the past of a sequence, were not defined for some instances. To encode these attributes with potentially missing values, we applied two different methods actually suggested by Lutz:

- the replacement of the missing value by a "best-guess" value
- the addition of a binary input indicating whether the corresponding attribute was present or absent.

Significantly better results were obtained by the second method.

This work was presented at ICANN'93 last september:
##REF##
X. Ding, T. Denoeux & F. Helloco (1993). Tracking rain cells in radar images using multilayer neural networks. In Proc. of ICANN'93, Springer-Verlag, p. 962-967.

From: "N. Karunanithi" <karun@faline.bellcore.com>

[...for nominal attributes:]
Both methods have the problem of poor scalability. If the number of missing values increases then the number of additional inputs will increase linearly in 1.1 and logarithmically in 1.2.
In fact, 1-of-n encoding may be a poor choice if (1) the number of input features is large and (2) such an expanded dimensional representation does not become a (semi) linearly separable problem. Even if it becomes a linearly separable problem, the overall complexity of the network can sometimes be very high.

[...for continuous attributes:]
This representation requires GUESS. A nominal transformation may not be a proper representation in some cases. Assume that the output values range over a large numerical interval. For example, from 0.0 to 10,000.0. If you use a simple scaling like dividing by 10,000.0 to make it between 0.0 and 1.0, this will result in poor accuracy of prediction. If the attribute is on the input side, then on theory the scaling is unnecessary because the input layer weights will scale accordingly. However, in practice I had lot of problem with this approach. Maybe a log tranformation before scaling may not be a bad choice.
If you use a closed scaling you may have problem whenever a future value exceeds the maximum value of the numerical intervel. For example, assume that the attribute is time, say in miliseconds. Any future time from the point of reference can exceed the limit. Hence any closed scaling will not work properly.

[...for ordinal attributes:]
I have compared Binary Encoding (1.2), Gray-Coded representation and straighforward scaling. Colsed scaling seems to do a good job. I have also compared open scaling and closed scaling and did find significant improvement in prediction accuracy.

###REF###
N. Karunanithi, D. Whitley and Y. K. Malaiya,
"Prediction of Software Reliability Using Connectionist Models",
IEEE Trans. Software Eng., July 1992, pp 563-574.
.
N. Karunanithi and Y. K. Malaiya, "The Scaling Problem in Neural Networks for Software Reliability Prediction", Proc. IEEE Int. Symposium on Rel. Eng., Oct. 1992, pp. 776-82.

I have not found a simple solution that is general. I think representation in general and the missing information in specific are open problems within connectionist research. I am not sure we will have a magic bullet for all problems. The best approach is to come up with a specific solution for a given problem.

From: Bill Skaggs <bill@nsma.arizona.edu>

There is at least one kind of network that has no problem (in principle) with missing inputs, namely a Boltzmann machine. You just refrain from clamping the input node whose value is missing, and treat it like an output node or hidden unit.

This may seem to be irrelevant to anything other than Boltzmann machines, but I think it could be argued that nothing very much simpler is capable of dealing with the problem. When you ask a network to handle missing inputs, you are in effect asking it to do pattern completion on the input layer, and for this a Boltzmann machine or some other sort of attractor network would seem to be required.

From: "Scott E. Fahlman" <sef+@cs.cmu.edu>

[Follow-up to Bill Skaggs:]
Good point, but perhaps in need of clarification for some readers:

There are two ways of training a Boltzmann machine. In one (the original form), there is no distinction between input and output units. During training we alternate between an instruction phase, in which all of the externally visible units are clamped to some pattern, and a normalization phase, in which the whole network is allow to run free. The idea is to modify the weights so that, when running free, the external units assume the various pattern values in the training set in their proper frequencies.

If only some subset of the externally visible units are clamped to certain values, the net will produce compatible completions in the other units, again with frequencies that match this part of the training set.

A net trained in this way will (in principle -- it might take a *very* long time for anything complicated) do what you suggest: Complete an "input" pattern and produce a compatible output at the same time. This works even if the input is *totally* missing.

I believe it was Geoff Hinton who realized that a Boltzmann machine could be trained more efficiently if you do make a distinction between input and output units, and don't waste any of the training effort learning to reconstruct the input. In this model, the instruction phase clamps both input and output units to some pattern, while the normalization phase clamps only the input units. Since the input units are correct in both cases, all of the networks learning power (such as it is) goes into producing correct patterns on the output units. A net trained in this way will not do input-completion.

I bring this up because I think many people will only have seen the latter kind of Boltzmann training, and will therefore misunderstand your observation.

By the way, one alternative method I have seen proposed for reconstructing missing input values is to first train an auto-encoder (with some degree of bottleneck to get generalization) on the training set, and then feed the output of this auto-encoder into the classification net. The auto-encoder should be able to replace any missing values with some degree of accuracy. I haven't played with this myself, but it does sound plausible. If anyone can point to a good study of this method, please post it here or send me E-mail.

From: "David G. Stork" <stork@cache.crc.ricoh.com>

##REF##
There is a provably optimal method for performing classification with missing inputs, described in Chapter 2 of "Pattern Classification and Scene Analysis" (2nd ed.) by R. O. Duda, P. E. Hart and D. G. Stork, which avoids the ad-hoc heuristics that have been described by others. Those interested in obtaining Chapter two via ftp should contact me.

From: Wray Buntine <wray@ptolemy-ethernet.arc.nasa.gov>

This missing value problem is of course shared amongst all the learning communities, artificial intelligence, statistics, pattern recognition, etc., not just neural networks.

A classic study in this area, which includes most suggestions I've read here so far, is

##REF##
```
@inproceedings{quinlan:ml6,
        AUTHOR = "J.R. Quinlan",
        TITLE = "Unknown Attribute Values in Induction",
        YEAR = 1989,
        BOOKTITLE = "Proceedings of the Sixth International
                        Machine Learning Workshop",
        PUBLISHER = "Morgan Kaufmann",
        ADDRESS = "Cornell, New York"}
```

The most frequently cited methods I've seen, and they're so common amongst the different communities its hard to lay credit:

1) replace missings by their some best guess
2) fracture the example into a set of fractional examples
   each with the missing value filled in somehow
3) call the missing value another input value

3 is a good thing to do if they are "informative" missing, i.e. if someone leaves the entry "telephone number" blank in a questionaire, then maybe they don't have a telephone, but probably not good otherwise unless you have loads of data and don't mind all the extra example types generated (as already mentioned)

1 is a quick and dirty hack at 2. How good depends on your application.

```
2 is an approximation to the "correct" approach for handling
"non-informative" missing values according to the standard
"mixture model".  The mathematics for this is general and applies
to virtually any learning algorithm trees, feed-forward nets,
linear regression, whatever.  We do it for feed-forward nets in

##REF##
@article{buntine.weigend:bbp,
        AUTHOR = "W.L. Buntine and A.S. Weigend",
        TITLE =  "Bayesian Back-Propagation",
        JOURNAL = "Complex Systems",
        Volume = 5,
        PAGES = "603--643",
        Number = 1,
        YEAR = "1991" }
and see Tresp, Ahmad & Neuneier in NIPS'94 for an implementation.
But no doubt someone probably published the general idea back in
the 50's.

I certainly wouldn't call missing values an open problem.
Rather, "efficient implementations of the standard approaches"
is, in some cases, an open problem.

------------------------------------------------------------------------

From: Volker Tresp <Volker.Tresp@zfe.siemens.de>

In general, the solution to the missing-data problem depends on
the missing-data mechanism. For example, if you sample the income
of a population and rich people tend to refuse the answer the mean
of your sample is biased. To obtain an unbiased solution
you would have to take into account the missing-data mechanism.

The missing-data mechanism can be ignored if it is independent of
the input and the output (in the example: the likelihood that a
person refuses to answer is independent of the person's income).
Most approaches assume that the missing-data mechanism can be ignored.

There exist a number of ad hoc solutions to the missing-data problem
but it is also possible to approach the problem from a statistical point
of view. In our paper (which will be published in the upcoming
NIPS-volume and which will be available on neuroprose
shortly) we discuss a systematic likelihood-based approach.
NN-regression  can be framed as a maximum likelihood learning problem
if we assume the standard signal plus Gaussian noise model

P(x, y) =  P(x) P(y|x)     \propto P(x) exp(-1/(2 \sigma^2) (y - NN(x))^2).


By deriving the probability density function for  a pattern with missing
features  we can formulate a likelihood function including patterns
with complete and incomplete features.

The solution  requires an  integration over the missing input.
In practice, the  integral  is  approximated  using a numerical approximation.
For networks of Gaussian basis functions,  it is possible to obtain
closed-form solutions (by extending the EM algorithm).

Our paper also discusses why and when ad hoc solutions --such as substituting
the mean for an unknown input--  are  harmful. For example,
if the mapping is approximately linear substituting the mean might work
quite well. In general, although, it introduces bias.


Training with missing and noisy input data is described in:

##REF##
``Training Neural Networks with Deficient Data,''
V. Tresp, S. Ahmad and R. Neuneier, in Cowan, J. D., Tesauro, G.,
and Alspector, J. (eds.), {\em  Advances in Neural Information Processing Systems 6},
Morgan Kaufmann,  1994.
.
A related paper by Zoubin Ghahramani and Michael Jordan will also appear
in the  upcoming NIPS-volume.
.
Recall with missing and noisy data is discussed in (available in neuroprose
as ahmad.missing.ps.Z):
.
``Some Solutions to the Missing Feature Problem in Vision,''
```

S. Ahmad and V. Tresp, in
{\em Advances in Neural Information Processing Systems 5,}
S. J. Hanson, J. D. Cowan, and C. L. Giles eds.,
San Mateo, CA, Morgan Kaufman, 1993.

From: Subhash Kak <kak@gate.ee.lsu.edu>

Missing values in feedback networks raise interesting questions:
Should these values be considered "don't know" values or should
these be generated in some "most likelihood" fashion? These issues
are discussed in the following paper:

##REF##
S.C. Kak, "Feedback neural networks: new characteristics and a
generalization", Circuits, Systems, Signal Processing, vol. 12,
no. 2, 1993, pp. 263-278.

From: Zoubin Ghahramani <zoubin@psyche.mit.edu>

I have also been looking into the issue of encoding and learning from
missing values in a neural network. The issue of handling missing
values has been addressed extensively in the statistics literature for
obvious reasons. To learn despite the missing values the data has to
be filled in, or the missing values integrated over. The basic
question is how to fill in the missing data. There are many different
methods for doing this in stats (mean imputation, regression
imputation, Bayesian methods, EM, etc.). For good reviews see (Little
and Rubin 1987; Little, 1992).

I do not in general recommend encoding "missing" as yet another value
to be learned over. Missing means something in a statistical sense --
that the input could be any of the values with some probability
distribution. You could, for example, augment the original data
filling in different values for the missing data points according to a
prior distribution. Then the training would assign different weights
to the artificially filled-in data points depending on how well they
predict the output (their posterior probability). This is essentially
the method proposed by Buntine and Weigand (1991). Other approaches
have been proposed by Tresp et al. (1993) and Ahmad and Tresp (1993).

I have just written a paper on the topic of learning from incomplete
data. In this paper I bring a statistical algorithm for learning from
incomplete data, called EM, into the framework of nonlinear function
approximation and classification with missing values. This approach
fits the data iteratively with a mixture model and uses that same
mixture model to effectively fill in any missing input or output
values at each step.

You can obtain the preprint by
        ftp psyche.mit.edu
        login: anonymous
        cd pub
        get zoubin.nips93.ps
To obtain code for the algorithm please contact me directly.

##REF##
Ahmad, S and Tresp, V (1993) "Some Solutions to the Missing Feature
Problem in Vision." In Hanson, S.J., Cowan, J.D., and Giles, C.L.,
editors, Advances in Neural Information Processing Systems 5. Morgan
Kaufmann Publishers, San Mateo, CA.
.
Buntine, WL, and Weigand, AS (1991) "Bayesian back-propagation." Complex
Systems. Vol 5 no 6 pp 603-43
.
Ghahramani, Z and Jordan MI (1994) "Supervised learning from
incomplete data via an EM approach" To appear in Cowan, J.D., Tesauro,
G., and Alspector,J. (eds.). Advances in Neural Information Processing
Systems 6. Morgan Kaufmann Publishers, San Francisco, CA, 1994.
.
Little, RJA (1992) "Regression With Missing X's: A Review." Journal of the
American Statistical Association. Volume 87, Number 420. pp.
1227-1237
.
Little, RJA. and Rubin, DB (1987). Statistical Analysis with Missing
Data. Wiley, New York.
.

Tresp, V, Hollatz J, Ahmad S (1993) "Network structuring and training using rule-based knowledge." In Hanson, S.J., Cowan, J.D., and Giles, C.~L., editors, Advances in Neural Information Processing Systems 5. Morgan Kaufmann Publishers, San Mateo, CA.

From: "Luis B. Almeida" <lba@ilusion.inesc.pt>

[replying to Bill Skaggs:]
The same effect, of trying to guess the missing inputs, can also be obtained with a recurrent multilayer perceptron, trained with recurrent backprop. This is the reason why the pattern completion results that I described in my 1987 ICNN paper (ref. below) were rather good.

##REF##
L. B. Almeida, "A learning rule for asynchronous perceptrons with feedback in a combinatorial environment", Proc IEEE First International Conference on Neural Networks, San Diego, Ca., 1987.

From: Michael Jordan <jordan@psyche.mit.edu>

The above is a nice observation that is worth emphasizing; I agree with all of it except the comment about being irrelevant to anything else. The Boltzmann machine is actually relevant to everything else. What the Boltzmann algorithm is doing with the missing value is essentially the same as what the EM algorithm for mixtures (that Ghahramani and Tresp referred to) is doing, and epitomizes the general case of an iterative "filling in" algorithm. The Boltzmann machine learning algorithm is a generalized EM (GEM) algorithm. During the E step the system computes the conditional correlation function for the nodes under the Boltzmann distribution, where the conditioning variables are the known data (the values of the clamped units) and the current values of the parameters (weights). This "fills in" the relevant statistic (the correlation function) and allows it to be used in the generalized M step (the contrastive Hebb rule).

Moreover, despite the fancy terminology, these algorithms are nothing more (nor less) than maximum likelihood estimation, where the likelihood function is the likelihood of the parameters *given the data that was actually observed*. By "filling in" missing data, you're not adding new information to the problem; rather, you're allowing yourself to use all the information that is in those components of the data vector that aren't missing. (EM theory provides the justification for that statement). E.g., if only one component of an input vector is missing, it's obviously wasteful to neglect what the other components of the input vector are telling you. And, indeed, if you neglect the whole vector, you will not end up with maximum likelihood estimates for the weights (nor in general will you get maximum likelihood estimates if you fill in a value with the unconditional mean of that variable).

"Filling in" is not the only way to compute ML estimates for missing data problems, but its virtue is that it allows the use of the same learning algorithms as would be used for complete data (without incurring any bias, if the filling in is done correctly). The only downside is that even if the complete-data algorithm is one-pass (which the Boltzmann algorithm and mixture fitting are not) the "filling-in" approach is generally iterative, because the parameter estimates depend on the filled-in values which in turn depend on the parameter estimates.
On the other hand, there are so-called "monotone" patterns of missing data for which the filling-in approach is not necessarily iterative. This monotone case might be of interest, because it is relevant for problems involving feedforward networks in which the input vectors are complete but some of the outputs are missing. (Note that even if all the output values for a case are missing, a ML algorithm will not throw the case out; there is statistical structure in the input vector that the algorithm must not neglect).

(See Ghahramani's message for references; particularly the Little and Rubin book).

That's it.

  Lutz

Lutz Prechelt (email: prechelt@ira.uka.de) | Whenever you
Institut fuer Programmstrukturen und Datenorganisation | complicate things,
Universitaet Karlsruhe; 76128 Karlsruhe; Germany | they get
(Voice: ++49/721/608-4068, FAX: ++49/721/694092) | less simple.

Anexo B

Este anexo contém os arquivos da simulação do problema do "Disjuntor Isolado por Seccionadoras" utilizando o Nevada Propagation. A decisão de abrir ou não o disjuntor está sempre representada como um dado binário. O estado das seccionadoras é representado de modos distintos em 7 (sete) exemplos:

EXEMPLO	ELEMENTO	DADO
bin	estado secc1	binário
	estado secc2	binário
bin+1	estado secc1	binário
	validade secc1	binário
	estado secc1	binário
	validade secc1	binário
bin2	secc1 aberta	binário
	secc1 fechada	binário
	secc2 aberta	binário
	secc2 fechada	binário
tern	estado secc1	ternário
	estado secc2	ternário
disc	secc1 aberta	binário
	secc1 fechada	binário
	secc1 trânsito	binário
	secc2 aberta	binário
	secc2 fechada	binário
	secc2 trânsito	binário
disc+1	secc1 aberta	binário
	secc1 fechada	binário
	secc1 trânsito	binário
	validade secc1	binário
	secc2 aberta	binário
	secc2 fechada	binário
	secc2 trânsito	binário
	validade secc2	binário
disc3	secc1 aberta	ternário
	secc1 fechada	ternário
	secc1 trânsito	ternário
	secc2 aberta	ternário
	secc2 fechada	ternário
	secc2 trânsito	ternário

Cada um dos exemplos acima contém um relatório da simulação, a descrição da rede, os padrões de teste e treinamento, e os resultados esperados e obtidos dos testes.

```
########################## Welcome to NevProp ##########################
NevProp started on Wed Mar 23 20:55:37 1994

Parameters read from file "bin.net"...

TRAINING set patterns: 3
TESTING set patterns:  3

InputUnits: 2 -- HiddenUnits: 5 -- OutputUnits: 1

SEED for initial random weights=764470537; Using lrand48(),srand48().

IBMorDOS 0              UseQuickProp 1          EpochWiseUpdate 1
BestByCindex 0          MinEpochs 200           BeyondBestEpoch 3
WtRange 2               HyperErr 1              SigmoidPrimeOffset 0.4
Epsilon 0.01            SplitEpsilon 1          Momentum 0.1
Decay -0.001            ScoreThreshold 0.1
MaxFactor 1.75          ModeSwitchThreshold 0
*------------------------------------------------------------------*
Epoch 0:
 TRAINING:   0.00 %correct ;  RMSErr=0.54882 ;  c index=0.50000
  TESTING:   0.00 %correct ;  RMSErr=0.47847
*------------------------------------------------------------------*
Epoch 5: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING:   0.00 %correct ;  RMSErr=0.54117 ;  c index=0.50000
  TESTING:   0.00 %correct ;  RMSErr=0.46906
*------------------------------------------------------------------*
Epoch 10: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING:   0.00 %correct ;  RMSErr=0.40695 ;  c index=1.00000
  TESTING:   0.00 %correct ;  RMSErr=0.37824
*------------------------------------------------------------------*
Epoch 15: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.02129 ;  c index=1.00000
  TESTING:  66.67 %correct ;  RMSErr=0.56439
*------------------------------------------------------------------*
Epoch 20: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00386 ;  c index=1.00000
  TESTING:  66.67 %correct ;  RMSErr=0.57478
*------------------------------------------------------------------*
Epoch 25: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00437 ;  c index=1.00000
  TESTING:  66.67 %correct ;  RMSErr=0.57410
*------------------------------------------------------------------*
Epoch 30: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00385 ;  c index=1.00000
  TESTING:  66.67 %correct ;  RMSErr=0.57451
*------------------------------------------------------------------*
Epoch 35: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00384 ;  c index=1.00000
  TESTING:  66.67 %correct ;  RMSErr=0.57461
*------------------------------------------------------------------*
Epoch 40: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00406 ;  c index=1.00000
  TESTING:  66.67 %correct ;  RMSErr=0.57439
*------------------------------------------------------------------*
Epoch 45: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00374 ;  c index=1.00000
  TESTING:  66.67 %correct ;  RMSErr=0.57463
*------------------------------------------------------------------*
Epoch 50: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00390 ;  c index=1.00000
  TESTING:  66.67 %correct ;  RMSErr=0.57435
*------------------------------------------------------------------*
Epoch 55: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00384 ;  c index=1.00000
  TESTING:  66.67 %correct ;  RMSErr=0.57455
*------------------------------------------------------------------*
Epoch 60: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00363 ;  c index=1.00000
  TESTING:  66.67 %correct ;  RMSErr=0.57471
*------------------------------------------------------------------*
Epoch 65: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00367 ;  c index=1.00000
  TESTING:  66.67 %correct ;  RMSErr=0.57459
*------------------------------------------------------------------*
Epoch 70: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00376 ;  c index=1.00000
  TESTING:  66.67 %correct ;  RMSErr=0.57453
*------------------------------------------------------------------*
Epoch 75: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
```

```
 TRAINING: 100.00 %correct ;  RMSErr=0.00375 ;  c index=1.00000
  TESTING:  66.67 %correct ;  RMSErr=0.57458
*-------------------------------------------------------------------*
Epoch 80: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00377 ;  c index=1.00000
  TESTING:  66.67 %correct ;  RMSErr=0.57458
*-------------------------------------------------------------------*
Epoch 85: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00360 ;  c index=1.00000
  TESTING:  66.67 %correct ;  RMSErr=0.57479
*-------------------------------------------------------------------*
Epoch 90: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00373 ;  c index=1.00000
  TESTING:  66.67 %correct ;  RMSErr=0.57462
*-------------------------------------------------------------------*
Epoch 95: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00368 ;  c index=1.00000
  TESTING:  66.67 %correct ;  RMSErr=0.57465
*-------------------------------------------------------------------*
Epoch 100: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00374 ;  c index=1.00000
  TESTING:  66.67 %correct ;  RMSErr=0.57459
   ... Predictions saved to "bin-2-5-1.prd"

   ... Weights saved to "bin-2-5-1.wts"
*********************************************************************
NevProp completed on Wed Mar 23 20:55:37 1994


#
IBMorDOS 0       UseQuiqProp 1       EpochWiseUpdate 1
BestByCindex 0   MinEpochs 200       BeyondBestEpoch 3.0
WeightRange 2.0  HyperErr 1          SigmoidPrimeoffset 0.4
Epsilon 0.01     SplitEpsilon 1      Momentum 0.1
WeightDecay -0.001 ScoreThreshold 0.10
QPMaxFactor 1.75 QPModeSwitchTreshold 0.0
Ninputs 2 Nhidden 5 Noutputs 1
Outputunittype 1
connectcalls  3
1 2    3 7
1 2    8 8
3 7    8 8
Ntrainingpatterns 3 NTestingpatterns 3
-0.5 -0.5  0.5
 0.5 -0.5 -0.5
-0.5  0.5 -0.5
-0.5 -0.5 -0.5
-0.5  0.5 -0.5
 0.5 -0.5 -0.5
SEQU PRED1 TRUE1
1 0.495205 -0.500000
2 -0.496917 -0.500000
3 -0.496924 -0.500000
```

```
########################## Welcome to NevProp ##########################
NevProp started on Wed Mar 23 18:26:39 1994

Parameters read from file "bin+1.net"...

TRAINING set patterns: 5
TESTING set patterns:  10

InputUnits: 4 -- HiddenUnits: 5 -- OutputUnits: 2

SEED for initial random weights=764461599; Using lrand48(),srand48().

IBMorDOS 0              UseQuickProp 1          EpochWiseUpdate 1
BestByCindex 0          MinEpochs 200           BeyondBestEpoch 3
WtRange 2               HyperErr 1              SigmoidPrimeOffset 0.4
Epsilon 0.01            SplitEpsilon 1          Momentum 0.1
Decay -0.001            ScoreThreshold 0.1
MaxFactor 1.75          ModeSwitchThreshold 0
*------------------------------------------------------------------*
Epoch 0:
 TRAINING:    0.00 %correct ;  RMSErr=0.51921
  TESTING:    5.00 %correct ;  RMSErr=0.56191
*------------------------------------------------------------------*
Epoch 5: Did QuickProp on 100.00 %, GradDesc on    0.00 % of weights.
 TRAINING:    0.00 %correct ;  RMSErr=0.48493
  TESTING:    5.00 %correct ;  RMSErr=0.51705
*------------------------------------------------------------------*
Epoch 10: Did QuickProp on 100.00 %, GradDesc on    0.00 % of weights.
 TRAINING:   20.00 %correct ;  RMSErr=0.33435
  TESTING:   35.00 %correct ;  RMSErr=0.32699
*------------------------------------------------------------------*
Epoch 15: Did QuickProp on 100.00 %, GradDesc on    0.00 % of weights.
 TRAINING:   90.00 %correct ;  RMSErr=0.04960
  TESTING:   80.00 %correct ;  RMSErr=0.08073
*------------------------------------------------------------------*
Epoch 20: Did QuickProp on 100.00 %, GradDesc on    0.00 % of weights.
 TRAINING:  100.00 %correct ;  RMSErr=0.01035
  TESTING:  100.00 %correct ;  RMSErr=0.03388
*------------------------------------------------------------------*
Epoch 25: Did QuickProp on 100.00 %, GradDesc on    0.00 % of weights.
 TRAINING:  100.00 %correct ;  RMSErr=0.00774
  TESTING:  100.00 %correct ;  RMSErr=0.02327
*------------------------------------------------------------------*
Epoch 30: Did QuickProp on 100.00 %, GradDesc on    0.00 % of weights.
 TRAINING:  100.00 %correct ;  RMSErr=0.00586
  TESTING:  100.00 %correct ;  RMSErr=0.02005
*------------------------------------------------------------------*
Epoch 35: Did QuickProp on 100.00 %, GradDesc on    0.00 % of weights.
 TRAINING:  100.00 %correct ;  RMSErr=0.00498
  TESTING:  100.00 %correct ;  RMSErr=0.02396
*------------------------------------------------------------------*
Epoch 40: Did QuickProp on 100.00 %, GradDesc on    0.00 % of weights.
 TRAINING:  100.00 %correct ;  RMSErr=0.00446
  TESTING:  100.00 %correct ;  RMSErr=0.02606
*------------------------------------------------------------------*
Epoch 45: Did QuickProp on 100.00 %, GradDesc on    0.00 % of weights.
 TRAINING:  100.00 %correct ;  RMSErr=0.00443
  TESTING:  100.00 %correct ;  RMSErr=0.01910
*------------------------------------------------------------------*
Epoch 50: Did QuickProp on 100.00 %, GradDesc on    0.00 % of weights.
 TRAINING:  100.00 %correct ;  RMSErr=0.00405
  TESTING:  100.00 %correct ;  RMSErr=0.02601
*------------------------------------------------------------------*
Epoch 55: Did QuickProp on 100.00 %, GradDesc on    0.00 % of weights.
 TRAINING:  100.00 %correct ;  RMSErr=0.00383
  TESTING:  100.00 %correct ;  RMSErr=0.02861
*------------------------------------------------------------------*
Epoch 60: Did QuickProp on 100.00 %, GradDesc on    0.00 % of weights.
 TRAINING:  100.00 %correct ;  RMSErr=0.00345
  TESTING:  100.00 %correct ;  RMSErr=0.02301
*------------------------------------------------------------------*
Epoch 65: Did QuickProp on 100.00 %, GradDesc on    0.00 % of weights.
 TRAINING:  100.00 %correct ;  RMSErr=0.00358
  TESTING:  100.00 %correct ;  RMSErr=0.02495
*------------------------------------------------------------------*
Epoch 70: Did QuickProp on 100.00 %, GradDesc on    0.00 % of weights.
 TRAINING:  100.00 %correct ;  RMSErr=0.00342
  TESTING:  100.00 %correct ;  RMSErr=0.02290
*------------------------------------------------------------------*
Epoch 75: Did QuickProp on 100.00 %, GradDesc on    0.00 % of weights.
```

```
 TRAINING: 100.00 %correct ;  RMSErr=0.00342
  TESTING: 100.00 %correct ;  RMSErr=0.02349
*-------------------------------------------------------------------*
Epoch 80: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00350
  TESTING: 100.00 %correct ;  RMSErr=0.02233
*-------------------------------------------------------------------*
Epoch 85: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00346
  TESTING: 100.00 %correct ;  RMSErr=0.02351
*-------------------------------------------------------------------*
Epoch 90: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00345
  TESTING: 100.00 %correct ;  RMSErr=0.02719
*-------------------------------------------------------------------*
Epoch 95: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00337
  TESTING: 100.00 %correct ;  RMSErr=0.03040
*-------------------------------------------------------------------*
Epoch 100: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00338
  TESTING: 100.00 %correct ;  RMSErr=0.03154
   ... Predictions saved to "bin+1-4-5-2.prd"

   ... Weights saved to "bin+1-4-5-2.wts"
*********************************************************************
NevProp completed on Wed Mar 23 18:26:39 1994

#
IBMorDOS 0       UseQuiqProp 1       EpochWiseUpdate 1
BestByCindex 0   MinEpochs 200       BeyondBestEpoch 3.0
WeightRange 2.0  HyperErr 1          SigmoidPrimeoffset 0.4
Epsilon 0.01     SplitEpsilon 1      Momentum 0.1
WeightDecay -0.001 ScoreThreshold 0.10
QPMaxFactor 1.75 QPModeSwitchTreshold 0.0
Ninputs 4 Nhidden 5 Noutputs 2
Outputunittype 0
connectcalls  3
1 4    5  9
1 4   10 11
5 9   10 11
Ntrainingpatterns 5 NTestingpatterns 10
 0 0 0 1 0 1
 0 1 0 0 0 1
 0 0 0 0 1 0
 1 0 0 0 0 0
 0 0 1 0 0 0
 0 1 0 1 0 1
 0 0 0 0 1 0
 1 0 0 0 0 0
 0 0 1 0 0 0
 0 0 0 1 0 1
 1 0 0 1 0 1
 0 0 1 1 0 1
 0 1 0 0 0 1
 1 1 0 0 0 1
 0 1 1 0 0 1
SEQU PRED1 PRED2 TRUE1 TRUE2
1 0.000357 0.999840 0.000000 1.000000
2 0.991928 0.002591 1.000000 0.000000
3 0.002352 0.001483 0.000000 0.000000
4 0.002305 0.001489 0.000000 0.000000
5 0.001710 0.996694 0.000000 1.000000
6 0.000029 0.927555 0.000000 1.000000
7 0.000026 0.927024 0.000000 1.000000
8 0.001680 0.996730 0.000000 1.000000
9 0.000028 0.932635 0.000000 1.000000
10 0.000025 0.931634 0.000000 1.000000
```

```
########################## Welcome to NevProp ##########################
NevProp started on Tue Mar 22 17:41:17 1994

Parameters read from file "bin2.net"...

TRAINING set patterns: 3
TESTING set patterns:  7

InputUnits: 4 -- HiddenUnits: 5 -- OutputUnits: 2

SEED for initial random weights=764372477; Using lrand48(),srand48().

IBMorDOS 0              UseQuickProp 1          EpochWiseUpdate 1
BestByCindex 0          MinEpochs 200           BeyondBestEpoch 3
WtRange 2               HyperErr 1              SigmoidPrimeOffset 0.4
Epsilon 0.01            SplitEpsilon 1          Momentum 0.1
Decay -0.001            ScoreThreshold 0.1
MaxFactor 1.75          ModeSwitchThreshold 0
*------------------------------------------------------------------*
Epoch 0:
 TRAINING:  50.00 %correct ;  RMSErr=0.54323
  TESTING:  35.71 %correct ;  RMSErr=0.56196
*------------------------------------------------------------------*
Epoch 1: Did QuickProp on   0.00 %, GradDesc on 100.00 % of weights.
 TRAINING:  50.00 %correct ;  RMSErr=0.54256
  TESTING:  35.71 %correct ;  RMSErr=0.56130
*------------------------------------------------------------------*
Epoch 2: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING:  50.00 %correct ;  RMSErr=0.54065
  TESTING:  35.71 %correct ;  RMSErr=0.55948
*------------------------------------------------------------------*
Epoch 3: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING:  33.33 %correct ;  RMSErr=0.53634
  TESTING:  28.57 %correct ;  RMSErr=0.55550
*------------------------------------------------------------------*
Epoch 4: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING:  33.33 %correct ;  RMSErr=0.52701
  TESTING:  28.57 %correct ;  RMSErr=0.54731
*------------------------------------------------------------------*
Epoch 5: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING:  33.33 %correct ;  RMSErr=0.50683
  TESTING:  28.57 %correct ;  RMSErr=0.53069
*------------------------------------------------------------------*
Epoch 6: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING:  33.33 %correct ;  RMSErr=0.46841
  TESTING:  21.43 %correct ;  RMSErr=0.49967
*------------------------------------------------------------------*
Epoch 7: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING:  16.67 %correct ;  RMSErr=0.43041
  TESTING:  14.29 %correct ;  RMSErr=0.46058
*------------------------------------------------------------------*
Epoch 8: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING:   0.00 %correct ;  RMSErr=0.39138
  TESTING:   7.14 %correct ;  RMSErr=0.43196
*------------------------------------------------------------------*
Epoch 9: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING:   0.00 %correct ;  RMSErr=0.34785
  TESTING:   7.14 %correct ;  RMSErr=0.41385
*------------------------------------------------------------------*
Epoch 10: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING:   0.00 %correct ;  RMSErr=0.30065
  TESTING:   7.14 %correct ;  RMSErr=0.40245
*------------------------------------------------------------------*
Epoch 11: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING:   0.00 %correct ;  RMSErr=0.25435
  TESTING:   7.14 %correct ;  RMSErr=0.40474
*------------------------------------------------------------------*
Epoch 12: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING:   0.00 %correct ;  RMSErr=0.21924
  TESTING:   7.14 %correct ;  RMSErr=0.41704
*------------------------------------------------------------------*
Epoch 13: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING:  33.33 %correct ;  RMSErr=0.19628
  TESTING:  28.57 %correct ;  RMSErr=0.43632
*------------------------------------------------------------------*
Epoch 14: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING:  33.33 %correct ;  RMSErr=0.16024
  TESTING:  28.57 %correct ;  RMSErr=0.45442
*------------------------------------------------------------------*
Epoch 15: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
```

```
 TRAINING:  50.00 %correct ;  RMSErr=0.12218
  TESTING:  35.71 %correct ;  RMSErr=0.47012
*-------------------------------------------------------------------*
Epoch 16: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING:  83.33 %correct ;  RMSErr=0.07859
  TESTING:  50.00 %correct ;  RMSErr=0.48272
*-------------------------------------------------------------------*
Epoch 17: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.04152
  TESTING:  57.14 %correct ;  RMSErr=0.48194
*-------------------------------------------------------------------*
Epoch 18: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.02562
  TESTING:  57.14 %correct ;  RMSErr=0.48658
*-------------------------------------------------------------------*
Epoch 19: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.01704
  TESTING:  57.14 %correct ;  RMSErr=0.48204
*-------------------------------------------------------------------*
Epoch 20: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.01069
  TESTING:  64.29 %correct ;  RMSErr=0.47639
   ... Predictions saved to "bin2-4-5-2.prd"

   ... Weights saved to "bin2-4-5-2.wts"
*********************************************************************
NevProp completed on Tue Mar 22 17:41:18 1994

#
IBMorDOS 0       UseQuiqProp 1       EpochWiseUpdate 1
BestByCindex 0   MinEpochs 200       BeyondBestEpoch 3.0
WeightRange 2.0  HyperErr 1          SigmoidPrimeoffset 0.4
Epsilon 0.01     SplitEpsilon 1      Momentum 0.1
WeightDecay -0.001 ScoreThreshold 0.10
QPMaxFactor 1.75 QPModeSwitchTreshold 0.0
Ninputs 4 Nhidden 5 Noutputs 2
Outputunittype 0
connectcalls  3
1 4    5  9
1 4   10 11
5 9   10 11
Ntrainingpatterns 3 NTestingpatterns  7
 0 1 0 1 0 1
 1 0 0 1 1 0
 0 1 1 0 1 0
 0 1 0 1 0 1
 1 0 0 1 1 0
 0 1 1 0 1 0
 1 0 0 0 0 0
 0 1 0 0 0 0
 0 0 1 0 0 0
 0 0 0 1 0 0
SEQU PRED1 PRED2 TRUE1 TRUE2
1 0.020319 0.986728 0.000000 1.000000
2 0.991968 0.003015 1.000000 0.000000
3 0.997479 0.004017 1.000000 0.000000
4 0.999304 0.001706 0.000000 0.000000
5 0.318063 0.967873 0.000000 0.000000
6 0.997590 0.000139 0.000000 0.000000
7 0.098715 0.367565 0.000000 0.000000
```

```
########################## Welcome to NevProp ##########################
NevProp started on Tue Mar 22 17:46:18 1994

Parameters read from file "tern.net"...

TRAINING set patterns: 3
TESTING set patterns:  5

InputUnits: 2 -- HiddenUnits: 5 -- OutputUnits: 1

SEED for initial random weights=764372778; Using lrand48(),srand48().

IBMorDOS 0              UseQuickProp 1          EpochWiseUpdate 1
BestByCindex 0          MinEpochs 200           BeyondBestEpoch 3
WtRange 2               HyperErr 1              SigmoidPrimeOffset 0.4
Epsilon 0.01            SplitEpsilon 1          Momentum 0.1
Decay -0.001            ScoreThreshold 0.1
MaxFactor 1.75          ModeSwitchThreshold 0
*------------------------------------------------------------------*
Epoch 0:
 TRAINING:   0.00 %correct ;  RMSErr=0.56472 ;  c index=0.50000
  TESTING:   0.00 %correct ;  RMSErr=0.44946
*------------------------------------------------------------------*
Epoch 5: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING:   0.00 %correct ;  RMSErr=0.55303 ;  c index=0.50000
  TESTING:   0.00 %correct ;  RMSErr=0.43870
*------------------------------------------------------------------*
Epoch 10: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING:   0.00 %correct ;  RMSErr=0.31200 ;  c index=1.00000
  TESTING:  20.00 %correct ;  RMSErr=0.25222
*------------------------------------------------------------------*
Epoch 15: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.07019 ;  c index=1.00000
  TESTING:  80.00 %correct ;  RMSErr=0.20108
*------------------------------------------------------------------*
Epoch 20: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00646 ;  c index=1.00000
  TESTING:  80.00 %correct ;  RMSErr=0.07943
*------------------------------------------------------------------*
Epoch 25: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00659 ;  c index=1.00000
  TESTING: 100.00 %correct ;  RMSErr=0.04623
*------------------------------------------------------------------*
Epoch 30: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00555 ;  c index=1.00000
  TESTING:  60.00 %correct ;  RMSErr=0.10326
*------------------------------------------------------------------*
Epoch 35: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00484 ;  c index=1.00000
  TESTING: 100.00 %correct ;  RMSErr=0.04288
*------------------------------------------------------------------*
Epoch 40: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00430 ;  c index=1.00000
  TESTING: 100.00 %correct ;  RMSErr=0.05894
*------------------------------------------------------------------*
Epoch 45: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00403 ;  c index=1.00000
  TESTING: 100.00 %correct ;  RMSErr=0.02992
*------------------------------------------------------------------*
Epoch 50: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00392 ;  c index=1.00000
  TESTING:  80.00 %correct ;  RMSErr=0.05806
*------------------------------------------------------------------*
Epoch 55: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00385 ;  c index=1.00000
  TESTING:  80.00 %correct ;  RMSErr=0.06731
*------------------------------------------------------------------*
Epoch 60: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00372 ;  c index=1.00000
  TESTING:  80.00 %correct ;  RMSErr=0.08108
*------------------------------------------------------------------*
Epoch 65: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00367 ;  c index=1.00000
  TESTING: 100.00 %correct ;  RMSErr=0.05946
*------------------------------------------------------------------*
Epoch 70: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00376 ;  c index=1.00000
  TESTING:  80.00 %correct ;  RMSErr=0.06482
*------------------------------------------------------------------*
Epoch 75: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
```

```
 TRAINING: 100.00 %correct ;  RMSErr=0.00396 ;  c index=1.00000
  TESTING:  60.00 %correct ;  RMSErr=0.07936
*-------------------------------------------------------------------*
Epoch 80: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00377 ;  c index=1.00000
  TESTING:  80.00 %correct ;  RMSErr=0.06322
*-------------------------------------------------------------------*
Epoch 85: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00381 ;  c index=1.00000
  TESTING: 100.00 %correct ;  RMSErr=0.06200
*-------------------------------------------------------------------*
Epoch 90: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00383 ;  c index=1.00000
  TESTING: 100.00 %correct ;  RMSErr=0.05494
*-------------------------------------------------------------------*
Epoch 95: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00381 ;  c index=1.00000
  TESTING: 100.00 %correct ;  RMSErr=0.04765
*-------------------------------------------------------------------*
Epoch 100: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00373 ;  c index=1.00000
  TESTING: 100.00 %correct ;  RMSErr=0.05011
   ... Predictions saved to "tern-2-5-1.prd"

   ... Weights saved to "tern-2-5-1.wts"
*********************************************************************
NevProp completed on Tue Mar 22 17:46:19 1994

#
IBMorDOS 0       UseQuiqProp 1       EpochWiseUpdate 1
BestByCindex 0   MinEpochs 200       BeyondBestEpoch 3.0
WeightRange 2.0  HyperErr 1          SigmoidPrimeoffset 0.4
Epsilon 0.01     SplitEpsilon 1      Momentum 0.1
WeightDecay -0.001 ScoreThreshold 0.10
QPMaxFactor 1.75 QPModeSwitchTreshold 0.0
Ninputs 2 Nhidden 5 Noutputs 1
Outputunittype 1
connectcalls  3
1 2    3 7
1 2    8 8
3 7    8 8
Ntrainingpatterns 3 NTestingpatterns 5
-0.5 -0.5  0.5
 0.5 -0.5 -0.5
-0.5  0.5 -0.5
-0.5 -0.5  0.5
-0.5  0.5 -0.5
 0.5 -0.5 -0.5
-0.5  0.0  0.0
 0.0 -0.5  0.0
SEQU PRED1 TRUE1
1 0.495173 0.500000
2 -0.496965 -0.500000
3 -0.496948 -0.500000
4 -0.081471 0.000000
5 -0.076639 0.000000
```

```
########################## Welcome to NevProp ##########################
NevProp started on Wed Mar 23 20:16:01 1994

Parameters read from file "disc.net"...

TRAINING set patterns: 10
TESTING set patterns:  13

InputUnits: 6 -- HiddenUnits: 5 -- OutputUnits: 1

SEED for initial random weights=764468161; Using lrand48(),srand48().

IBMorDOS 0              UseQuickProp 1          EpochWiseUpdate 1
BestByCindex 0          MinEpochs 200           BeyondBestEpoch 3
WtRange 2               HyperErr 1              SigmoidPrimeOffset 0.4
Epsilon 0.01            SplitEpsilon 1          Momentum 0.1
Decay -0.001            ScoreThreshold 0.1
MaxFactor 1.75          ModeSwitchThreshold 0
*-------------------------------------------------------------------*
Epoch 0:
 TRAINING:  50.00 %correct ;  RMSErr=0.32469 ;  c index=0.66667
  TESTING:  46.15 %correct ;  RMSErr=0.30096 ;  c index=0.66667
*-------------------------------------------------------------------*
Epoch 5: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING:  50.00 %correct ;  RMSErr=0.31818 ;  c index=0.77778
  TESTING:  46.15 %correct ;  RMSErr=0.29539 ;  c index=0.75000
*-------------------------------------------------------------------*
Epoch 10: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING:  60.00 %correct ;  RMSErr=0.18909 ;  c index=1.00000
  TESTING:  61.54 %correct ;  RMSErr=0.19240 ;  c index=1.00000
*-------------------------------------------------------------------*
Epoch 15: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.02279 ;  c index=1.00000
  TESTING:  92.31 %correct ;  RMSErr=0.13216 ;  c index=1.00000
*-------------------------------------------------------------------*
Epoch 20: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00364 ;  c index=1.00000
  TESTING:  92.31 %correct ;  RMSErr=0.16956 ;  c index=1.00000
*-------------------------------------------------------------------*
Epoch 25: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00211 ;  c index=1.00000
  TESTING:  92.31 %correct ;  RMSErr=0.16445 ;  c index=1.00000
*-------------------------------------------------------------------*
Epoch 30: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00353 ;  c index=1.00000
  TESTING:  92.31 %correct ;  RMSErr=0.09975 ;  c index=1.00000
*-------------------------------------------------------------------*
Epoch 35: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00190 ;  c index=1.00000
  TESTING:  92.31 %correct ;  RMSErr=0.16677 ;  c index=1.00000
*-------------------------------------------------------------------*
Epoch 40: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00173 ;  c index=1.00000
  TESTING:  92.31 %correct ;  RMSErr=0.11823 ;  c index=1.00000
*-------------------------------------------------------------------*
Epoch 45: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00179 ;  c index=1.00000
  TESTING:  92.31 %correct ;  RMSErr=0.09784 ;  c index=1.00000
*-------------------------------------------------------------------*
Epoch 50: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00177 ;  c index=1.00000
  TESTING:  92.31 %correct ;  RMSErr=0.08185 ;  c index=1.00000
*-------------------------------------------------------------------*
Epoch 55: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00169 ;  c index=1.00000
  TESTING:  92.31 %correct ;  RMSErr=0.06857 ;  c index=1.00000
*-------------------------------------------------------------------*
Epoch 60: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00155 ;  c index=1.00000
  TESTING:  92.31 %correct ;  RMSErr=0.07675 ;  c index=1.00000
*-------------------------------------------------------------------*
Epoch 65: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00156 ;  c index=1.00000
  TESTING:  92.31 %correct ;  RMSErr=0.05845 ;  c index=1.00000
*-------------------------------------------------------------------*
Epoch 70: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00162 ;  c index=1.00000
  TESTING:  92.31 %correct ;  RMSErr=0.04813 ;  c index=1.00000
*-------------------------------------------------------------------*
Epoch 75: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
```

```
 TRAINING: 100.00 %correct ;  RMSErr=0.00169 ;  c index=1.00000
  TESTING:  92.31 %correct ;  RMSErr=0.05142 ;  c index=1.00000
*-------------------------------------------------------------------*
Epoch 80: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00166 ;  c index=1.00000
  TESTING:  92.31 %correct ;  RMSErr=0.05944 ;  c index=1.00000
*-------------------------------------------------------------------*
Epoch 85: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00158 ;  c index=1.00000
  TESTING:  92.31 %correct ;  RMSErr=0.07777 ;  c index=1.00000
*-------------------------------------------------------------------*
Epoch 90: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00158 ;  c index=1.00000
  TESTING:  92.31 %correct ;  RMSErr=0.07817 ;  c index=1.00000
*-------------------------------------------------------------------*
Epoch 95: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00167 ;  c index=1.00000
  TESTING:  92.31 %correct ;  RMSErr=0.07679 ;  c index=1.00000
*-------------------------------------------------------------------*
Epoch 100: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00165 ;  c index=1.00000
  TESTING:  92.31 %correct ;  RMSErr=0.07912 ;  c index=1.00000
   ... Predictions saved to "disc-6-5-1.prd"

   ... Weights saved to "disc-6-5-1.wts"
*********************************************************************
NevProp completed on Wed Mar 23 20:16:02 1994

#
IBMorDOS 0       UseQuiqProp 1       EpochWiseUpdate 1
BestByCindex 0   MinEpochs 200       BeyondBestEpoch 3.0
WeightRange 2.0  HyperErr 1          SigmoidPrimeoffset 0.4
Epsilon 0.01     SplitEpsilon 1      Momentum 0.1
WeightDecay -0.001 ScoreThreshold 0.10
QPMaxFactor 1.75 QPModeSwitchTreshold 0.0
Ninputs 6 Nhidden 5 Noutputs 1
Outputunittype 0
connectcalls  3
1 6    7 11
1 6   12 12
7 11  12 12
Ntrainingpatterns 10 NTestingpatterns 13
 0 0 0 0 0 0
 0 0 1 0 0 1 1
 0 0 1 0 1 0 0
 0 0 1 1 0 0 0
 0 1 0 0 0 1 0
 0 1 0 0 1 0 0
 0 1 0 1 0 0 0
 1 0 0 0 0 1 0
 1 0 0 0 1 0 0
 1 0 0 1 0 0 0
 0 0 0 0 0 0 0
 0 0 1 0 0 0 0
 0 0 1 0 0 1 1
 0 0 1 0 1 0 0
 0 0 1 1 0 0 0
 0 1 0 0 0 0 0
 0 1 0 0 0 1 0
 0 1 0 0 1 0 0
 0 1 0 1 0 0 0
 1 0 0 0 0 0 0
 1 0 0 0 0 1 0
 1 0 0 0 1 0 0
 1 0 0 1 0 0 0
SEQU PRED1 TRUE1
1 0.000333 0.000000
2 0.285220 0.000000
3 0.995437 1.000000
4 0.001251 0.000000
5 0.001247 0.000000
6 0.000024 0.000000
7 0.001234 0.000000
8 0.000006 0.000000
9 0.000006 0.000000
10 0.000024 0.000000
11 0.001232 0.000000
12 0.000006 0.000000
13 0.000006 0.000000
```

```
######################### Welcome to NevProp #########################
NevProp started on Wed Mar 23 21:16:02 1994

Parameters read from file "disc+1.net"...

TRAINING set patterns: 10
TESTING set patterns:  12

InputUnits: 8 -- HiddenUnits: 5 -- OutputUnits: 1

SEED for initial random weights=764471762; Using lrand48(),srand48().

IBMorDOS 0              UseQuickProp 1          EpochWiseUpdate 1
BestByCindex 0          MinEpochs 200           BeyondBestEpoch 3
WtRange 2               HyperErr 1              SigmoidPrimeOffset 0.4
Epsilon 0.01            SplitEpsilon 1          Momentum 0.1
Decay -0.001            ScoreThreshold 0.2
MaxFactor 1.75          ModeSwitchThreshold 0
*-----------------------------------------------------------------*
Epoch 0:
 TRAINING:  30.00 %correct ;  RMSErr=0.39652 ;  c index=0.88889
  TESTING:  25.00 %correct ;  RMSErr=0.41598 ;  c index=0.81818
*-----------------------------------------------------------------*
Epoch 5: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING:  50.00 %correct ;  RMSErr=0.33250 ;  c index=0.88889
  TESTING:  41.67 %correct ;  RMSErr=0.34689 ;  c index=0.81818
*-----------------------------------------------------------------*
Epoch 10: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING:  90.00 %correct ;  RMSErr=0.20677 ;  c index=1.00000
  TESTING:  91.67 %correct ;  RMSErr=0.19741 ;  c index=1.00000
*-----------------------------------------------------------------*
Epoch 15: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.02235 ;  c index=1.00000
  TESTING: 100.00 %correct ;  RMSErr=0.04373 ;  c index=1.00000
*-----------------------------------------------------------------*
Epoch 20: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00362 ;  c index=1.00000
  TESTING:  91.67 %correct ;  RMSErr=0.06939 ;  c index=1.00000
*-----------------------------------------------------------------*
Epoch 25: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00172 ;  c index=1.00000
  TESTING: 100.00 %correct ;  RMSErr=0.03918 ;  c index=1.00000
*-----------------------------------------------------------------*
Epoch 30: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00253 ;  c index=1.00000
  TESTING: 100.00 %correct ;  RMSErr=0.04354 ;  c index=1.00000
*-----------------------------------------------------------------*
Epoch 35: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00229 ;  c index=1.00000
  TESTING:  91.67 %correct ;  RMSErr=0.07898 ;  c index=1.00000
*-----------------------------------------------------------------*
Epoch 40: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00197 ;  c index=1.00000
  TESTING:  91.67 %correct ;  RMSErr=0.10806 ;  c index=1.00000
*-----------------------------------------------------------------*
Epoch 45: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00170 ;  c index=1.00000
  TESTING:  91.67 %correct ;  RMSErr=0.06268 ;  c index=1.00000
*-----------------------------------------------------------------*
Epoch 50: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00196 ;  c index=1.00000
  TESTING: 100.00 %correct ;  RMSErr=0.05009 ;  c index=1.00000
*-----------------------------------------------------------------*
Epoch 55: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00181 ;  c index=1.00000
  TESTING: 100.00 %correct ;  RMSErr=0.03653 ;  c index=1.00000
*-----------------------------------------------------------------*
Epoch 60: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00169 ;  c index=1.00000
  TESTING: 100.00 %correct ;  RMSErr=0.03549 ;  c index=1.00000
*-----------------------------------------------------------------*
Epoch 65: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00184 ;  c index=1.00000
  TESTING: 100.00 %correct ;  RMSErr=0.03418 ;  c index=1.00000
*-----------------------------------------------------------------*
Epoch 70: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00182 ;  c index=1.00000
  TESTING: 100.00 %correct ;  RMSErr=0.03299 ;  c index=1.00000
*-----------------------------------------------------------------*
Epoch 75: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
```

```
 TRAINING: 100.00 %correct ;  RMSErr=0.00172 ;  c index=1.00000
  TESTING: 100.00 %correct ;  RMSErr=0.03289 ;  c index=1.00000
*-------------------------------------------------------------------*
Epoch 80: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00184 ;  c index=1.00000
  TESTING: 100.00 %correct ;  RMSErr=0.02341 ;  c index=1.00000
*-------------------------------------------------------------------*
Epoch 85: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00176 ;  c index=1.00000
  TESTING: 100.00 %correct ;  RMSErr=0.02053 ;  c index=1.00000
*-------------------------------------------------------------------*
Epoch 90: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00175 ;  c index=1.00000
  TESTING: 100.00 %correct ;  RMSErr=0.02066 ;  c index=1.00000
*-------------------------------------------------------------------*
Epoch 95: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00179 ;  c index=1.00000
  TESTING: 100.00 %correct ;  RMSErr=0.01892 ;  c index=1.00000
*-------------------------------------------------------------------*
Epoch 100: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00174 ;  c index=1.00000
  TESTING: 100.00 %correct ;  RMSErr=0.01972 ;  c index=1.00000
   ... Predictions saved to "disc+1-8-5-1.prd"

   ... Weights saved to "disc+1-8-5-1.wts"
*********************************************************************
NevProp completed on Wed Mar 23 21:16:03 1994

#
IBMorDOS 0       UseQuiqProp 1       EpochWiseUpdate 1
BestByCindex 0   MinEpochs 200       BeyondBestEpoch 3.0
WeightRange 2.0  HyperErr 1          SigmoidPrimeoffset 0.4
Epsilon 0.01     SplitEpsilon 1      Momentum 0.1
WeightDecay -0.001 ScoreThreshold 0.20
QPMaxFactor 1.75 QPModeSwitchTreshold 0.0
Ninputs 8 Nhidden 5 Noutputs 1
Outputunittype 0
connectcalls  3
1 8    9 13
1 8   14 14
9 13  14 14
Ntrainingpatterns 10 NTestingpatterns 12
 0 0 0 1 0 0 0 1 0
 0 0 1 0 0 0 1 0 1
 0 0 1 0 0 1 0 0 0
 0 0 1 0 1 0 0 0 0
 0 1 0 0 0 0 1 0 0
 0 1 0 0 0 1 0 0 0
 0 1 0 0 1 0 0 0 0
 1 0 0 0 0 0 1 0 0
 1 0 0 0 0 1 0 0 0
 1 0 0 0 1 0 0 0 0
 0 0 0 1 0 0 0 1 0
 0 0 1 0 0 0 1 0 1
 0 0 1 0 0 1 0 0 0
 0 0 1 0 1 0 0 0 0
 0 1 0 0 0 0 1 0 0
 0 1 0 0 0 1 0 0 0
 0 1 0 0 1 0 0 0 0
 1 0 0 0 0 0 1 0 0
 1 0 0 0 0 1 0 0 0
 1 0 0 0 1 0 0 0 0
 0 0 1 0 0 0 0 1 0
 0 0 0 1 0 0 1 0 0
SEQU PRED1 TRUE1
1 0.000524 0.000000
2 0.995672 1.000000
3 0.001682 0.000000
4 0.001675 0.000000
5 0.001676 0.000000
6 0.000045 0.000000
7 0.000044 0.000000
8 0.001659 0.000000
9 0.000045 0.000000
10 0.000045 0.000000
11 0.038970 0.000000
12 0.055838 0.000000
```

```
########################## Welcome to NevProp ##########################
NevProp started on Wed Mar 23 20:52:52 1994

Parameters read from file "disc3.net"...

TRAINING set patterns: 10
TESTING set patterns:  12

InputUnits: 6 -- HiddenUnits: 5 -- OutputUnits: 1

SEED for initial random weights=764470372; Using lrand48(),srand48().

IBMorDOS 0              UseQuickProp 1          EpochWiseUpdate 1
BestByCindex 0          MinEpochs 200           BeyondBestEpoch 3
WtRange 2               HyperErr 1              SigmoidPrimeOffset 0.4
Epsilon 0.01            SplitEpsilon 1          Momentum 0.1
Decay -0.001            ScoreThreshold 0.1
MaxFactor 1.75          ModeSwitchThreshold 0
*------------------------------------------------------------------*
Epoch 0:
 TRAINING:    0.00 %correct ;  RMSErr=0.70418 ;  c index=0.66667
  TESTING:    0.00 %correct ;  RMSErr=0.71918 ;  c index=0.63636
*------------------------------------------------------------------*
Epoch 5: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING:    0.00 %correct ;  RMSErr=0.62091 ;  c index=0.66667
  TESTING:    0.00 %correct ;  RMSErr=0.63417 ;  c index=0.63636
*------------------------------------------------------------------*
Epoch 10: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING:   70.00 %correct ;  RMSErr=0.14902 ;  c index=1.00000
  TESTING:   58.33 %correct ;  RMSErr=0.17137 ;  c index=1.00000
*------------------------------------------------------------------*
Epoch 15: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00806 ;  c index=1.00000
  TESTING:   83.33 %correct ;  RMSErr=0.09468 ;  c index=1.00000
*------------------------------------------------------------------*
Epoch 20: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00184 ;  c index=1.00000
  TESTING:   83.33 %correct ;  RMSErr=0.07015 ;  c index=1.00000
*------------------------------------------------------------------*
Epoch 25: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00162 ;  c index=1.00000
  TESTING:   83.33 %correct ;  RMSErr=0.11486 ;  c index=1.00000
*------------------------------------------------------------------*
Epoch 30: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00165 ;  c index=1.00000
  TESTING:   83.33 %correct ;  RMSErr=0.08960 ;  c index=1.00000
*------------------------------------------------------------------*
Epoch 35: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00191 ;  c index=1.00000
  TESTING:   83.33 %correct ;  RMSErr=0.05637 ;  c index=1.00000
*------------------------------------------------------------------*
Epoch 40: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00171 ;  c index=1.00000
  TESTING:   83.33 %correct ;  RMSErr=0.07256 ;  c index=1.00000
*------------------------------------------------------------------*
Epoch 45: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00167 ;  c index=1.00000
  TESTING:   83.33 %correct ;  RMSErr=0.06584 ;  c index=1.00000
*------------------------------------------------------------------*
Epoch 50: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00167 ;  c index=1.00000
  TESTING:   83.33 %correct ;  RMSErr=0.05617 ;  c index=1.00000
*------------------------------------------------------------------*
Epoch 55: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00154 ;  c index=1.00000
  TESTING:   83.33 %correct ;  RMSErr=0.05362 ;  c index=1.00000
*------------------------------------------------------------------*
Epoch 60: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00166 ;  c index=1.00000
  TESTING:   83.33 %correct ;  RMSErr=0.04555 ;  c index=1.00000
*------------------------------------------------------------------*
Epoch 65: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00159 ;  c index=1.00000
  TESTING:   83.33 %correct ;  RMSErr=0.04586 ;  c index=1.00000
*------------------------------------------------------------------*
Epoch 70: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00164 ;  c index=1.00000
  TESTING:   83.33 %correct ;  RMSErr=0.04371 ;  c index=1.00000
*------------------------------------------------------------------*
Epoch 75: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
```

```
 TRAINING: 100.00 %correct ;  RMSErr=0.00160 ;  c index=1.00000
  TESTING:  91.67 %correct ;  RMSErr=0.04076 ;  c index=1.00000
*-------------------------------------------------------------------*
Epoch 80: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00156 ;  c index=1.00000
  TESTING:  91.67 %correct ;  RMSErr=0.03888 ;  c index=1.00000
*-------------------------------------------------------------------*
Epoch 85: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00159 ;  c index=1.00000
  TESTING:  91.67 %correct ;  RMSErr=0.03968 ;  c index=1.00000
*-------------------------------------------------------------------*
Epoch 90: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00156 ;  c index=1.00000
  TESTING:  91.67 %correct ;  RMSErr=0.04169 ;  c index=1.00000
*-------------------------------------------------------------------*
Epoch 95: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00157 ;  c index=1.00000
  TESTING:  91.67 %correct ;  RMSErr=0.04092 ;  c index=1.00000
*-------------------------------------------------------------------*
Epoch 100: Did QuickProp on 100.00 %, GradDesc on   0.00 % of weights.
 TRAINING: 100.00 %correct ;  RMSErr=0.00159 ;  c index=1.00000
  TESTING: 100.00 %correct ;  RMSErr=0.03856 ;  c index=1.00000
   ... Predictions saved to "disc3-6-5-1.prd"

   ... Weights saved to "disc3-6-5-1.wts"
*********************************************************************
NevProp completed on Wed Mar 23 20:52:53 1994

#
IBMorDOS 0        UseQuiqProp 1        EpochWiseUpdate 1
BestByCindex 0    MinEpochs 200        BeyondBestEpoch 3.0
WeightRange 2.0   HyperErr 1           SigmoidPrimeoffset 0.4
Epsilon 0.01      SplitEpsilon 1       Momentum 0.1
WeightDecay -0.001 ScoreThreshold 0.10
QPMaxFactor 1.75 QPModeSwitchTreshold 0.0
Ninputs 6 Nhidden 5 Noutputs 1
Outputunittype 1
connectcalls  3
1 6    7 11
1 6   12 12
7 11  12 12
Ntrainingpatterns 10 NTestingpatterns 12
  0.0  0.0  0.0  0.0  0.0  0.0 -0.5
 -0.5 -0.5  0.5 -0.5 -0.5  0.5  0.5
 -0.5 -0.5  0.5 -0.5  0.5 -0.5 -0.5
 -0.5 -0.5  0.5  0.5 -0.5 -0.5 -0.5
 -0.5  0.5 -0.5 -0.5 -0.5  0.5 -0.5
 -0.5  0.5 -0.5 -0.5  0.5 -0.5 -0.5
 -0.5  0.5 -0.5  0.5 -0.5 -0.5 -0.5
  0.5 -0.5 -0.5 -0.5 -0.5  0.5 -0.5
  0.5 -0.5 -0.5 -0.5  0.5 -0.5 -0.5
  0.5 -0.5 -0.5  0.5 -0.5 -0.5 -0.5
  0.0  0.0  0.0  0.0  0.0  0.0 -0.5
 -0.5 -0.5  0.5 -0.5 -0.5  0.5  0.5
 -0.5 -0.5  0.5 -0.5  0.5 -0.5 -0.5
 -0.5 -0.5  0.5  0.5 -0.5 -0.5 -0.5
 -0.5  0.5 -0.5 -0.5 -0.5  0.5 -0.5
 -0.5  0.5 -0.5 -0.5  0.5 -0.5 -0.5
 -0.5  0.5 -0.5  0.5 -0.5 -0.5 -0.5
  0.5 -0.5 -0.5 -0.5 -0.5  0.5 -0.5
  0.5 -0.5 -0.5 -0.5  0.5 -0.5 -0.5
  0.5 -0.5 -0.5  0.5 -0.5 -0.5 -0.5
 -0.5 -0.5  0.5  0.0  0.0  0.0 -0.5
  0.0  0.0  0.0 -0.5 -0.5  0.5 -0.5
SEQU PRED1 TRUE1
1 -0.499669 -0.500000
2 0.495911 0.500000
3 -0.498523 -0.500000
4 -0.498570 -0.500000
5 -0.498550 -0.500000
6 -0.499979 -0.500000
7 -0.499977 -0.500000
8 -0.498565 -0.500000
9 -0.499980 -0.500000
10 -0.499978 -0.500000
11 -0.408318 -0.500000
12 -0.403001 -0.500000
```

Anexo C

Este anexo contém os arquivos da simulação do exemplo fornecido por *Telfer(91)* utilizando o SNNS v3.1, e são descritos na seguinte ordem:

ARQUIVO	DESCRIÇÃO
air.cfb	descrição do treinamento
air.net	descrição da rede
air.pat	padrões de treinamento
air.log	relatório do treinamento
air.res	resultado do treinamento
air.trn	descrição da rede treinada
air.xcfb	descrição do teste
air.xnet	descrição da rede
air.xpat	padrões de teste
air.xlog	relatório do teste
air.xres	resultado do teste

```
#
Type: SNNSBATCH_1
#
# This is the style for an snnsbat configuration file, type 1.
#
# The following keyword-value combinations may be supplied in any order.
# If a key is given twice, the second appearence is taken.
# Keys that are not required for a special run may be omitted.
# If a key is ommited but required, a default value is assumed.
# The lines may be separated with comments.
#
# Please note the mandatory file type specification at the begining and
# the colon following the key.
#
#
#                                              Defaults:
#
#     NoOfLearnParam:       <int>                 0
#     LearnParam:           [ <float> ]*          0.0
#     NoOfInitParam:        <int>                 0
#     InitParam:            [ <float> ]*          0.0
#     NetworkFile:          <string>              ""
#     TrainedNetworkFile:   <string>              ""
#     LearnPatternFile:     <string>              ""
#     TestPatternFile:      <string>              ""
#     ResultFile:           <string>              ""
#     InitFunction:         <string>              ""
#     MaxLearnCycles:       <int>                 0
#     MaxErrorToStop:       <float>               0.0
#     Shuffle:              [ YES | NO ]          NO
#     ResultMinMaxPattern:  <int><int>            0 0
#     ResultIncludeInput:   [ YES | NO ]          NO
#     ResultIncludeOutput:  [ YES | NO ]          YES
#
###############################################################################
NetworkFile: air.net
#
InitFunction: Randomize_Weights
NoOfInitParam: 2
InitParam: -0.5 0.5
#
LearnPatternFile: air.pat
NoOfLearnParam: 3
LearnParam: 0.005 0.9 2
MaxLearnCycles: 5000
MaxErrorToStop: 0.1
Shuffle: NO
#
TrainedNetworkFile: air.trn
TestPatternFile: air.pat
ResultFile: air.res
ResultIncludeInput: YES
ResultIncludeOutput: YES
```

```
SNNS network definition file V1.4-3D
generated at Wed Feb 24 23:50:08 1993

network name : aircraft
source files :
no. of units : 21
no. of connections : 83
no. of unit types : 2
no. of site types : 2


learning function : BPTT
update function   : BPTT_Order


type definition section :

name           | act func      | out func      |sites
---------------|---------------|---------------|---------
outType        | Act_Logistic | Out_Identity |
LongeroutType | Act_Logistic | Out_Identity |
---------------|---------------|---------------|---------


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 |                | militar  |  0.00000 |  0.00000 | i  |  2, 9, 0 |||
  2 |                | envergad |  0.00000 |  0.00000 | i  |  4, 9, 0 |||
  3 |                | underpod |  0.00000 |  0.00000 | i  |  6, 9, 0 |||
  4 |                | comprim  |  0.00000 |  0.00000 | i  |  8, 9, 0 |||
  5 |                | motores  |  0.00000 |  0.00000 | i  | 10, 9, 0 |||
  6 |                | american |  0.00000 |  0.00000 | i  | 12, 9, 0 |||
  7 |                | asaV     |  0.00000 |  0.00000 | i  | 14, 9, 0 |||
  8 |                | quilhas  |  0.00000 |  0.00000 | i  | 16, 9, 0 |||
  9 |                | transp   |  0.00000 |  0.00000 | d  | 18, 3, 0 |||
 10 |                | bomb     |  0.00000 |  0.00000 | d  | 20, 3, 0 |||
 11 |                | caca     |  0.00000 |  0.00000 | d  | 22, 3, 0 |||
 12 |                | C5B      |  0.00000 |  0.00000 | o  |  4, 1, 0 |||
 13 |                | An22     |  0.00000 |  0.00000 | o  |  6, 1, 0 |||
 14 |                | B1B      |  0.00000 |  0.00000 | o  |  8, 1, 0 |||
 15 |                | BlackJac |  0.00000 |  0.00000 | o  | 10, 1, 0 |||
 16 |                | F15C     |  0.00000 |  0.00000 | o  | 12, 1, 0 |||
 17 |                | Mig23MF  |  0.00000 |  0.00000 | o  | 14, 1, 0 |||
 18 |                | hidden   |  0.00000 |  0.00000 | h  |  6, 6, 0 |||
 19 |                | hidden   |  0.00000 |  0.00000 | h  |  8, 6, 0 |||
 20 |                | hidden   |  0.00000 |  0.00000 | h  | 10, 6, 0 |||
 21 |                | hidden   |  0.00000 |  0.00000 | h  | 12, 6, 0 |||
----|----------------|----------|-----------|-----------|----|----------|----------|-----
-----|---------


connection definition section :

target | site     | source:weight
-------|----------|-------------------------------------------------------------------------
---------------------------------------
     9 |          | 18: 0.00000, 19: 0.00000, 20: 0.00000, 21:0.00000
    10 |          | 18: 0.00000, 19: 0.00000, 20: 0.00000, 21:0.00000
    11 |          | 18: 0.00000, 19: 0.00000, 20: 0.00000, 21:0.00000
    12 |          | 18: 0.00000, 19: 0.00000, 20: 0.00000, 21:0.00000
    13 |          | 18: 0.00000, 19: 0.00000, 20: 0.00000, 21:0.00000
    14 |          | 18: 0.00000, 19: 0.00000, 20: 0.00000, 21:0.00000
    15 |          | 18: 0.00000, 19: 0.00000, 20: 0.00000, 21:0.00000
    16 |          | 18: 0.00000, 19: 0.00000, 20: 0.00000, 21:0.00000
    17 |          | 18: 0.00000, 19: 0.00000, 20: 0.00000, 21:0.00000
    18 |          | 1: 0.00000, 2: 0.00000, 3: 0.00000, 4:0.00000,
5:0.00000, 6:0.00000, 7:0.00000, 8:00000,  9: 0.00000, 10:0.00000,
11:0.0000
```

```
    19 |         | 1: 0.00000, 2: 0.00000, 3: 0.00000, 4:0.00000,
5:0.00000, 6:0.00000, 7:0.00000, 8:00000,  9: 0.00000, 10:0.00000,
11:0.0000
    20 |         | 1: 0.00000, 2: 0.00000, 3: 0.00000, 4:0.00000,
5:0.00000, 6:0.00000, 7:0.00000, 8:00000,  9: 0.00000, 10:0.00000,
11:0.0000
    21 |         | 1: 0.00000, 2: 0.00000, 3: 0.00000, 4:0.00000,
5:0.00000, 6:0.00000, 7:0.00000, 8:00000,  9: 0.00000, 10:0.00000,
11:0.0000
-------|---------|-----------------------------------------------------------
---------------------------------------
```

```
SNNS pattern definition file V1.4
generated at Tue Dec  3 20:36:50 1991

No. of patterns     :  43
No. of input units  : 11
No. of output units :  9

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0
1 1 1 1 0 1 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0
1 1 1 1 0 1 0 1 1 0 0 1 0 0 1 0 0 0 0 0 0
1 1 1 1 0 1 0 1 1 0 0 1 0 0 1 0 0 0 0 0 0
1 1 1 1 0 1 0 1 1 0 0 1 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
1 1 1 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0
1 1 1 1 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0
1 1 1 1 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0
1 1 1 1 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 0 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 0 1 0 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0
1 1 0 1 0 1 1 1 0 1 0 0 1 0 0 0 0 0 0 0 0
1 1 0 1 0 1 1 1 0 1 0 0 1 0 0 0 1 0 0 0 0
1 1 0 1 0 1 1 1 0 1 0 0 1 0 0 0 1 0 0 0 0
1 1 0 1 0 1 1 1 0 1 0 0 1 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 0 1 0 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0
1 1 0 1 0 0 1 1 0 1 0 0 1 0 0 0 0 0 0 0 0
1 1 0 1 0 0 1 1 0 1 0 0 1 0 0 0 0 1 0 0 0
1 1 0 1 0 0 1 1 0 1 0 0 1 0 0 0 0 1 0 0 0
1 1 0 1 0 0 1 1 0 1 0 0 1 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
1 0 0 0 1 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0
1 0 0 0 1 1 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0
1 0 0 0 1 1 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0
1 0 0 0 1 1 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 1 0 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0
1 0 0 0 1 0 1 1 0 0 1 0 0 1 0 0 0 0 0 0 0
1 0 0 0 1 0 1 1 0 0 1 0 0 1 0 0 0 0 0 0 1
1 0 0 0 1 0 1 1 0 0 1 0 0 1 0 0 0 0 0 0 1
1 0 0 0 1 0 1 1 0 0 1 0 0 1 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
SNNS 3D-Kernel V3.0002 Batchlearning Program
Configuration file: 'air.cfb'
Log file          : 'air.log'



######## SNNS batch execution run. Loop 1 ########
Networkfile 'air.net' loaded.
Patternfile 'air.pat' loaded.
No. of patterns: 43

No. of cycles: 5000

Max. network error to stop: 0.100000

Patterns are in order

Network name             : aircraft
No. of units             : 21
No. of sites             : 0
No. of links             : 80

Learning Function        : BPTT
Learning Parameter #1    : 0.005000
Learning Parameter #2    : 0.900000
Learning Parameter #3    : 2.000000


Test pattern file : 'air.pat'

Init Function: Randomize_Weights
Init Parameter #1       : -0.500000
Init Parameter #2       : 0.500000


Result File              : 'air.res'
Result File Start Pattern: 0
Result File End Pattern  : 0
Result File Input Pattern included
Result File Output Pattern included


******************************************************

SNNS 3D-Kernel V3.0002 Batchlearning started at Tue May 10 16:50:38 1994


Network initialized with
Randomize_Weights -0.50 0.50

Cycle: 0
Learning function value(s): [1]: 80.7412

Cycle: 50
Learning function value(s): [1]: 24.0895

Cycle: 100
Learning function value(s): [1]: 15.2483

Cycle: 150
Learning function value(s): [1]: 12.8561

Cycle: 200
Learning function value(s): [1]: 11.5254

Cycle: 250
Learning function value(s): [1]: 10.3976

Cycle: 300
Learning function value(s): [1]: 9.22941

Cycle: 350
Learning function value(s): [1]: 8.15422

Cycle: 400
Learning function value(s): [1]: 7.28355

Cycle: 450
Learning function value(s): [1]: 6.59614
```

```
Cycle: 500
Learning function value(s): [1]: 6.03562

Cycle: 550
Learning function value(s): [1]: 5.52398

Cycle: 600
Learning function value(s): [1]: 4.94466

Cycle: 650
Learning function value(s): [1]: 4.28501

Cycle: 700
Learning function value(s): [1]: 3.6521

Cycle: 750
Learning function value(s): [1]: 3.12214

Cycle: 800
Learning function value(s): [1]: 2.70065

Cycle: 850
Learning function value(s): [1]: 2.36433

Cycle: 900
Learning function value(s): [1]: 2.09287

Cycle: 950
Learning function value(s): [1]: 1.87118

Cycle: 1000
Learning function value(s): [1]: 1.6879

Cycle: 1050
Learning function value(s): [1]: 1.53453

Cycle: 1100
Learning function value(s): [1]: 1.40474

Cycle: 1150
Learning function value(s): [1]: 1.29374

Cycle: 1200
Learning function value(s): [1]: 1.1979

Cycle: 1250
Learning function value(s): [1]: 1.11445

Cycle: 1300
Learning function value(s): [1]: 1.0412

Cycle: 1350
Learning function value(s): [1]: 0.976468

Cycle: 1400
Learning function value(s): [1]: 0.918889

Cycle: 1450
Learning function value(s): [1]: 0.86738

Cycle: 1500
Learning function value(s): [1]: 0.821054

Cycle: 1550
Learning function value(s): [1]: 0.77919

Cycle: 1600
Learning function value(s): [1]: 0.741188

Cycle: 1650
Learning function value(s): [1]: 0.706556

Cycle: 1700
Learning function value(s): [1]: 0.67487

Cycle: 1750
Learning function value(s): [1]: 0.645781

Cycle: 1800
```

```
Learning function value(s): [1]: 0.618991

Cycle: 1850
Learning function value(s): [1]: 0.594238

Cycle: 1900
Learning function value(s): [1]: 0.571315

Cycle: 1950
Learning function value(s): [1]: 0.550024

Cycle: 2000
Learning function value(s): [1]: 0.5302

Cycle: 2050
Learning function value(s): [1]: 0.511703

Cycle: 2100
Learning function value(s): [1]: 0.494407

Cycle: 2150
Learning function value(s): [1]: 0.478199

Cycle: 2200
Learning function value(s): [1]: 0.462977

Cycle: 2250
Learning function value(s): [1]: 0.448665

Cycle: 2300
Learning function value(s): [1]: 0.435182

Cycle: 2350
Learning function value(s): [1]: 0.422454

Cycle: 2400
Learning function value(s): [1]: 0.410423

Cycle: 2450
Learning function value(s): [1]: 0.399043

Cycle: 2500
Learning function value(s): [1]: 0.388257

Cycle: 2550
Learning function value(s): [1]: 0.378014

Cycle: 2600
Learning function value(s): [1]: 0.368284

Cycle: 2650
Learning function value(s): [1]: 0.359031

Cycle: 2700
Learning function value(s): [1]: 0.350216

Cycle: 2750
Learning function value(s): [1]: 0.341811

Cycle: 2800
Learning function value(s): [1]: 0.333788

Cycle: 2850
Learning function value(s): [1]: 0.326123

Cycle: 2900
Learning function value(s): [1]: 0.318794

Cycle: 2950
Learning function value(s): [1]: 0.311778

Cycle: 3000
Learning function value(s): [1]: 0.305054

Cycle: 3050
Learning function value(s): [1]: 0.298606

Cycle: 3100
Learning function value(s): [1]: 0.292426
```

```
Cycle: 3150
Learning function value(s): [1]: 0.286487

Cycle: 3200
Learning function value(s): [1]: 0.280779

Cycle: 3250
Learning function value(s): [1]: 0.275285

Cycle: 3300
Learning function value(s): [1]: 0.269997

Cycle: 3350
Learning function value(s): [1]: 0.264905

Cycle: 3400
Learning function value(s): [1]: 0.259998

Cycle: 3450
Learning function value(s): [1]: 0.255266

Cycle: 3500
Learning function value(s): [1]: 0.250695

Cycle: 3550
Learning function value(s): [1]: 0.246287

Cycle: 3600
Learning function value(s): [1]: 0.242026

Cycle: 3650
Learning function value(s): [1]: 0.237906

Cycle: 3700
Learning function value(s): [1]: 0.233923

Cycle: 3750
Learning function value(s): [1]: 0.23007

Cycle: 3800
Learning function value(s): [1]: 0.226341

Cycle: 3850
Learning function value(s): [1]: 0.222725

Cycle: 3900
Learning function value(s): [1]: 0.219219

Cycle: 3950
Learning function value(s): [1]: 0.215819

Cycle: 4000
Learning function value(s): [1]: 0.212523

Cycle: 4050
Learning function value(s): [1]: 0.209325

Cycle: 4100
Learning function value(s): [1]: 0.20622

Cycle: 4150
Learning function value(s): [1]: 0.203202

Cycle: 4200
Learning function value(s): [1]: 0.20027

Cycle: 4250
Learning function value(s): [1]: 0.197422

Cycle: 4300
Learning function value(s): [1]: 0.194653

Cycle: 4350
Learning function value(s): [1]: 0.191959

Cycle: 4400
Learning function value(s): [1]: 0.189337

Cycle: 4450
Learning function value(s): [1]: 0.186785
```

```
Cycle: 4500
Learning function value(s): [1]: 0.184299

Cycle: 4550
Learning function value(s): [1]: 0.181877

Cycle: 4600
Learning function value(s): [1]: 0.179518

Cycle: 4650
Learning function value(s): [1]: 0.177218

Cycle: 4700
Learning function value(s): [1]: 0.174976

Cycle: 4750
Learning function value(s): [1]: 0.172789

Cycle: 4800
Learning function value(s): [1]: 0.170655

Cycle: 4850
Learning function value(s): [1]: 0.168572

Cycle: 4900
Learning function value(s): [1]: 0.166538

Cycle: 4950
Learning function value(s): [1]: 0.164553

Result file saved.

Network saved to air.trn.

SNNS 3D-Kernel V3.0002 Batchlearning terminated at Tue May 10 16:52:25 1994
System: ULTRIX  Node: amazonas  Machine: RISC
**********************************************************

----   STATISTICS  ----
No. of learned cycles: 5000
No. of units updated : 4515000
No. of sites updated : 0
No. of links updated : 17200000
CPU Time used: 40.26 seconds
User time: 107 seconds
No. of connection updates per second (CUPS): 4.272231e+05
```

```
SNNS result file V1.4-3D
generated at Tue May 10 16:52:25 1994

No. of patterns     : 43
No. of input units  : 11
No. of output units : 9
startpattern        : 1
endpattern          : 43
input patterns included
teaching output included
#1
0 0 0 0 0 0 0 0 0 0 0
0
0 0 0 0 0 0 0 0 0
0.00886 0.0083 0.00756 0.00186 0.01798 0.00877 0.01673 0.01762 0.00449
#2
1 1 1 1 0 1 0 1 0 0 0
0
0 0 0 0 0 0 0 0 0
0.0222 0.01647 0.01873 0.00001 0.00056 0.00006 0.01187 0.00004 0.01524
#3
1 1 1 1 0 1 0 1 0 0 0
0
1 0 0 0 0 0 0 0 0
0.96557 0.02017 0.00292 0.02784 0.01524 0.00004 0.00008 0.00015 0
#4
1 1 1 1 0 1 0 1 1 0 0
0
1 0 0 0 0 0 0 0 0
0.96806 0.01906 0.00302 0.03371 0.01648 0.00005 0.00007 0.00017 0
#5
1 1 1 1 0 1 0 1 1 0 0
0
1 0 0 1 0 0 0 0 0
0.98954 0.00734 0.0273 0.93735 0.02683 0.00051 0 0.02423 0
#6
1 1 1 1 0 1 0 1 1 0 0
0
1 0 0 1 0 0 0 0 0
0.98951 0.00735 0.02745 0.93771 0.02671 0.00051 0 0.02435 0
#7
1 1 1 1 0 1 0 1 1 0 0
0
1 0 0 1 0 0 0 0 0
0.98917 0.00747 0.0296 0.94232 0.0251 0.00055 0 0.026 0
#8
0 0 0 0 0 0 0 0 0 0 0
0
0 0 0 0 0 0 0 0 0
0.00886 0.0083 0.00756 0.00186 0.01798 0.00877 0.01673 0.01762 0.00449
#9
1 1 1 1 0 0 0 0 0 0 0
0
0 0 0 0 0 0 0 0 0
0.0222 0.01647 0.01873 0.00001 0.00056 0.00006 0.01187 0.00004 0.01524
#10
1 1 1 1 0 0 0 0 0 0 0
0
1 0 0 0 0 0 0 0 0
0.95971 0.02115 0.00012 0.00005 0.0465 0 0.01995 0 0.00001
#11
1 1 1 1 0 0 0 0 1 0 0
0
1 0 0 0 0 0 0 0 0
0.96184 0.02018 0.00012 0.00005 0.04985 0 0.01906 0 0.00001
#12
1 1 1 1 0 0 0 0 1 0 0
0
1 0 0 0 1 0 0 0 0
0.99971 0.00081 0.00004 0.04078 0.94514 0 0.00066 0.00028 0
#13
1 1 1 1 0 0 0 0 1 0 0
0
1 0 0 0 1 0 0 0 0
0.99971 0.00081 0.00004 0.04108 0.94547 0 0.00066 0.00029 0
#14
1 1 1 1 0 0 0 0 1 0 0
0
1 0 0 0 1 0 0 0 0
0.99974 0.00076 0.00004 0.04648 0.95064 0 0.00061 0.00032 0
```

```
#15
0 0 0 0 0 0 0 0 0 0
0
0 0 0 0 0 0 0 0 0
0.00886 0.0083 0.00756 0.00186 0.01798 0.00877 0.01673 0.01762 0.00449
#16
1 1 0 1 0 1 1 1 0 0
0
0 0 0 0 0 0 0 0 0
0.0222 0.01647 0.01873 0.00001 0.00056 0.00006 0.01187 0.00004 0.01524
#17
1 1 0 1 0 1 1 1 0 0
0
0 1 0 0 0 0 0 0 0
0.01144 0.95148 0.04614 0.00736 0 0.05668 0.00034 0.00001 0.00056
#18
1 1 0 1 0 1 1 1 0 1
0
0 1 0 0 0 0 0 0 0
0.01135 0.95202 0.0459 0.0074 0 0.0575 0.00034 0.00001 0.00056
#19
1 1 0 1 0 1 1 1 0 1
0
0 1 0 0 0 1 0 0 0
0.00033 0.9997 0.00429 0.01546 0 0.94245 0.02468 0 0.00113
#20
1 1 0 1 0 1 1 1 0 1
0
0 1 0 0 0 1 0 0 0
0.00033 0.9997 0.00428 0.01545 0 0.94246 0.02469 0 0.00113
#21
1 1 0 1 0 1 1 1 0 1
0
0 1 0 0 0 1 0 0 0
0.00034 0.99971 0.00402 0.01499 0 0.94215 0.02663 0 0.00111
#22
0 0 0 0 0 0 0 0 0 0
0
0 0 0 0 0 0 0 0 0
0.00886 0.0083 0.00756 0.00186 0.01798 0.00877 0.01673 0.01762 0.00449
#23
1 1 0 1 0 0 1 1 0 0
0
0 0 0 0 0 0 0 0 0
0.0222 0.01647 0.01873 0.00001 0.00056 0.00006 0.01187 0.00004 0.01524
#24
1 1 0 1 0 0 1 1 0 0
0
0 1 0 0 0 0 0 0 0
0.02157 0.97061 0.00116 0.00085 0.00002 0.02011 0.02752 0 0.00028
#25
1 1 0 1 0 0 1 1 0 1
0
0 1 0 0 0 0 0 0 0
0.02067 0.97148 0.00121 0.00089 0.00002 0.02143 0.02643 0 0.00028
#26
1 1 0 1 0 0 1 1 0 1
0
0 1 0 0 0 0 1 0 0
0.01818 0.99678 0.00001 0.00009 0.00016 0.03051 0.95507 0 0.00012
#27
1 1 0 1 0 0 1 1 0 1
0
0 1 0 0 0 0 1 0 0
0.01821 0.99678 0.00001 0.00009 0.00016 0.03045 0.95519 0 0.00012
#28
1 1 0 1 0 0 1 1 0 1
0
0 1 0 0 0 0 1 0 0
0.01862 0.99675 0.00001 0.00009 0.00017 0.02945 0.95731 0 0.00012
#29
0 0 0 0 0 0 0 0 0 0
0
0 0 0 0 0 0 0 0 0
0.00886 0.0083 0.00756 0.00186 0.01798 0.00877 0.01673 0.01762 0.00449
#30
1 0 0 0 1 1 0 0 0 0
0
0 0 0 0 0 0 0 0 0
0.0222 0.01647 0.01873 0.00001 0.00056 0.00006 0.01187 0.00004 0.01524
```

```
#31
1 0 0 0 1 1 0 0 0 0
0
0 0 1 0 0 0 0 0 0
0.01252 0.00408 0.9707 0.00356 0.00001 0.00103 0 0.02606 0.01988
#32
1 0 0 0 1 1 0 0 0 0
1
0 0 1 0 0 0 0 0 0
0.01324 0.00366 0.97173 0.00398 0.00001 0.00102 0 0.03067 0.019
#33
1 0 0 0 1 1 0 0 0 0
1
0 0 1 0 0 0 0 1 0
0.01811 0.00004 0.99831 0.04007 0.00014 0.00074 0 0.9505 0.03551
#34
1 0 0 0 1 1 0 0 0 0
1
0 0 1 0 0 0 0 1 0
0.01819 0.00004 0.99831 0.04032 0.00014 0.00074 0 0.95077 0.03535
#35
1 0 0 0 1 1 0 0 0 0
1
0 0 1 0 0 0 0 1 0
0.01924 0.00004 0.9983 0.04381 0.00016 0.00074 0 0.95474 0.03337
#36
0 0 0 0 0 0 0 0 0 0
0
0 0 0 0 0 0 0 0 0
0.00886 0.0083 0.00756 0.00186 0.01798 0.00877 0.01673 0.01762 0.00449
#37
1 0 0 0 1 0 1 1 0 0
0
0 0 0 0 0 0 0 0 0
0.0222 0.01647 0.01873 0.00001 0.00056 0.00006 0.01187 0.00004 0.01524
#38
1 0 0 0 1 0 1 1 0 0
0
0 0 1 0 0 0 0 0 0
0.00303 0.0422 0.94133 0.00029 0 0.00135 0.00001 0.00074 0.06303
#39
1 0 0 0 1 0 1 1 0 0
1
0 0 1 0 0 0 0 0 0
0.00289 0.04069 0.94464 0.00028 0 0.00135 0.00001 0.00078 0.0674
#40
1 0 0 0 1 0 1 1 0 0
1
0 0 1 0 0 0 0 0 1
0.00008 0.00221 0.99936 0.00002 0 0.00112 0 0.02697 0.93167
#41
1 0 0 0 1 0 1 1 0 0
1
0 0 1 0 0 0 0 0 1
0.00008 0.0022 0.99937 0.00002 0 0.00112 0 0.02706 0.93188
#42
1 0 0 0 1 0 1 1 0 0
1
0 0 1 0 0 0 0 0 1
0.00008 0.00209 0.9994 0.00002 0 0.0011 0 0.0285 0.93484
#43
0 0 0 0 0 0 0 0 0 0
0
0 0 0 0 0 0 0 0 0
0.00886 0.0083 0.00756 0.00186 0.01798 0.00877 0.01673 0.01762 0.00449
```

```
SNNS network definition file V1.4-3D
generated at Tue May 10 16:52:25 1994

network name : aircraft
source files :
no. of units : 21
no. of connections : 80
no. of unit types : 2
no. of site types : 0


learning function : BPTT
update function   : BPTT_Order


type definition section :

name           | act func      | out func      |sites
---------------|---------------|---------------|-------
LongeroutType  | Act_Logistic  | Out_Identity  |
outType        | Act_Logistic  | Out_Identity  |
---------------|---------------|---------------|-------


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 |                | militar  |  0.00000  | -0.10354  | i  |  2, 9, 0 |||
  2 |                | envergad |  0.00000  |  0.34049  | i  |  4, 9, 0 |||
  3 |                | underpod |  0.00000  | -0.14666  | i  |  6, 9, 0 |||
  4 |                | comprim  |  0.00000  | -0.05342  | i  |  8, 9, 0 |||
  5 |                | motores  |  0.00000  | -0.18131  | i  | 10, 9, 0 |||
  6 |                | american |  0.00000  |  0.38643  | i  | 12, 9, 0 |||
  7 |                | asaV     |  0.00000  | -0.48442  | i  | 14, 9, 0 |||
  8 |                | quilhas  |  0.00000  |  0.08409  | i  | 16, 9, 0 |||
  9 |                | transp   |  0.00886  | -4.71763  | d  | 18, 3, 0 |||
 10 |                | bomb     |  0.00830  | -4.78326  | d  | 20, 3, 0 |||
 11 |                | caca     |  0.00756  | -4.87701  | d  | 22, 3, 0 |||
 12 |                | C5B      |  0.00186  | -6.28453  | o  |  4, 1, 0 |||
 13 |                | An22     |  0.01798  | -4.00061  | o  |  6, 1, 0 |||
 14 |                | B1B      |  0.00877  | -4.72806  | o  |  8, 1, 0 |||
 15 |                | BlackJac |  0.01673  | -4.07356  | o  | 10, 1, 0 |||
 16 |                | F15C     |  0.01762  | -4.02102  | o  | 12, 1, 0 |||
 17 |                | Mig23MF  |  0.00449  | -5.40194  | o  | 14, 1, 0 |||
 18 |                | hidden   |  0.65742  |  0.65180  | h  |  6, 6, 0 |||
 19 |                | hidden   |  0.12202  | -1.97341  | h  |  8, 6, 0 |||
 20 |                | hidden   |  0.38792  | -0.45604  | h  | 10, 6, 0 |||
 21 |                | hidden   |  0.85009  |  1.73529  | h  | 12, 6, 0 |||
----|----------------|----------|-----------|-----------|----|----------|----------|-----
-----|-------


connection definition section :

target | site | source:weight
-------|------|-------------------------------------------------------------------------------
-----------------------------------------
     9 |      | 21:-6.35405, 20: 7.45999, 19:-4.59916, 18: 6.08601
    10 |      | 21: 4.39134, 20: 5.98453, 19: 2.74206, 18:-8.66353
    11 |      | 21: 1.30228, 20:-9.21195, 19: 7.43935, 18: 3.76802
    12 |      | 21:-8.91382, 20: 5.45196, 19: 5.75453, 18:-1.51457
    13 |      | 21:-7.70814, 20: 3.64231, 19:-7.81844, 18: 3.96674
    14 |      | 21: 0.38932, 20: 0.35001, 19: 7.11877, 18:-9.65961
    15 |      | 21: 4.48923, 20: 3.33412, 19:-7.47876, 18:-6.91356
    16 |      | 21:-7.47107, 20:-7.20968, 19: 4.47338, 18: 3.60914
    17 |      | 21: 7.00766, 20:-10.86415, 19: 2.54363, 18:-1.24620
    18 |      | 11: 0.68974, 10:-4.83402,  9: 1.88720,  8:-0.32182,  7:-1.30749,  6:
0.78406,  5: 2.84468,  4:-0.95625,  3: 2.71335,
                2:-0.71279,  1: 2.34287
```

```
    19 |      | 11: 1.91214, 10:-2.85536,  9: 2.29039,  8: 1.63304,  7: 2.99048,  6:
5.97851,  5: 1.96370,  4:-1.30265,  3:-3.24367,
                2:-1.32466,  1: 0.30990
    20 |      | 11:-4.04649, 10: 0.76459,  9: 0.95041,  8: 0.99810,  7: 0.70983,  6:
1.22526,  5:-1.24531,  4: 1.87435,  3: 0.47182,
                2: 1.45162,  1: 0.07065
    21 |      | 11:-2.52193, 10: 1.69767,  9:-4.22713,  8: 0.82103,  7: 3.57226,  6:-
2.55554,  5: 0.42443,  4: 0.18837,  3:-1.86768,
                2: 0.57690,  1: 1.03175
-------|------|-------------------------------------------------------------------------
-----------------------------------------
```

```
#
Type: SNNSBATCH_1
#
# This is the style for an snnsbat configuration file, type 1.
#
# The following keyword-value combinations may be supplied in any order.
# If a key is given twice, the second appearence is taken.
# Keys that are not required for a special run may be omitted.
# If a key is ommited but required, a default value is assumed.
# The lines may be separated with comments.
#
# Please note the mandatory file type specification at the begining and
# the colon following the key.
#
#
#                                              Defaults:
#
#     NoOfLearnParam:       <int>                 0
#     LearnParam:           { <float> }*          0.0
#     NoOfInitParam:        <int>                 0
#     InitParam:            [ <float> ]*          0.0
#     NetworkFile:          <string>              ""
#     TrainedNetworkFile:   <string>              ""
#     LearnPatternFile:     <string>              ""
#     TestPatternFile:      <string>              ""
#     ResultFile:           <string>              ""
#     InitFunction:         <string>              ""
#     MaxLearnCycles:       <int>                 0
#     MaxErrorToStop:       <float>               0.0
#     Shuffle:              [ YES | NO ]          NO
#     ResultMinMaxPattern: <int><int>             0 0
#     ResultIncludeInput:   [ YES | NO ]          NO
#     ResultIncludeOutput: [ YES | NO ]           YES
#
#############################################################################
NetworkFile: air.xnet
#
TestPatternFile: air.xpat
ResultFile: air.xres
ResultIncludeInput: YES
ResultIncludeOutput: YES
```

```
SNNS network definition file V1.4-3D
generated at Tue May 10 16:52:25 1994

network name : aircraft
source files :
no. of units : 21
no. of connections : 80
no. of unit types : 2
no. of site types : 0


learning function : BPTT
update function   : BPTT_Order


type definition section :

name           | act func     | out func     |sites
---------------|--------------|--------------|-------
LongeroutType  | Act_Logistic | Out_Identity |
outType        | Act_Logistic | Out_Identity |
---------------|--------------|--------------|-------


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 |                | militar  |  0.00000 | -0.10354 | i  |  2, 9, 0 |||
  2 |                | envergad |  0.00000 |  0.34049 | i  |  4, 9, 0 |||
  3 |                | underpod |  0.00000 | -0.14666 | i  |  6, 9, 0 |||
  4 |                | comprim  |  0.00000 | -0.05342 | i  |  8, 9, 0 |||
  5 |                | motores  |  0.00000 | -0.18131 | i  | 10, 9, 0 |||
  6 |                | american |  0.00000 |  0.38643 | i  | 12, 9, 0 |||
  7 |                | asaV     |  0.00000 | -0.48442 | i  | 14, 9, 0 |||
  8 |                | quilhas  |  0.00000 |  0.08409 | i  | 16, 9, 0 |||
  9 |                | transp   |  0.00886 | -4.71763 | o  | 18, 3, 0 |||
 10 |                | bomb     |  0.00830 | -4.78326 | o  | 20, 3, 0 |||
 11 |                | caca     |  0.00756 | -4.87701 | o  | 22, 3, 0 |||
 12 |                | C5B      |  0.00186 | -6.28453 | o  |  4, 1, 0 |||
 13 |                | An22     |  0.01798 | -4.00061 | o  |  6, 1, 0 |||
 14 |                | B1B      |  0.00877 | -4.72806 | o  |  8, 1, 0 |||
 15 |                | BlackJac |  0.01673 | -4.07356 | o  | 10, 1, 0 |||
 16 |                | F15C     |  0.01762 | -4.02102 | o  | 12, 1, 0 |||
 17 |                | Mig23MF  |  0.00449 | -5.40194 | o  | 14, 1, 0 |||
 18 |                | hidden   |  0.65742 |  0.65180 | h  |  6, 6, 0 |||
 19 |                | hidden   |  0.12202 | -1.97341 | h  |  8, 6, 0 |||
 20 |                | hidden   |  0.38792 | -0.45604 | h  | 10, 6, 0 |||
 21 |                | hidden   |  0.85009 |  1.73529 | h  | 12, 6, 0 |||
----|----------------|----------|----------|----------|----|----------|----------|-----
-----|-------


connection definition section :

target | site | source:weight
-------|------|-------------------------------------------------------------------------
-------------------------------------------
     9 |      | 21:-6.35405, 20: 7.45999, 19:-4.59916, 18: 6.08601
    10 |      | 21: 4.39134, 20: 5.98453, 19: 2.74206, 18:-8.66353
    11 |      | 21: 1.30228, 20:-9.21195, 19: 7.43935, 18: 3.76802
    12 |      | 21:-8.91382, 20: 5.45196, 19: 5.75453, 18:-1.51457
    13 |      | 21:-7.70814, 20: 3.64231, 19:-7.81844, 18: 3.96674
    14 |      | 21: 0.38932, 20: 0.35001, 19: 7.11877, 18:-9.65961
    15 |      | 21: 4.48923, 20: 3.33412, 19:-7.47876, 18:-6.91356
    16 |      | 21:-7.47107, 20:-7.20968, 19: 4.47338, 18: 3.60914
    17 |      | 21: 7.00766, 20:-10.86415, 19: 2.54363, 18:-1.24620
    18 |      | 11: 0.68974, 10:-4.83402,  9: 1.88720,  8:-0.32182,  7:-1.30749,  6:
0.78406,  5: 2.84468,  4:-0.95625,  3: 2.71335,
                2:-0.71279,  1: 2.34287
```

```
    19 |      | 11: 1.91214, 10:-2.85536,  9: 2.29039,  8: 1.63304,  7: 2.99048,  6:
5.97851,  5: 1.96370,  4:-1.30265,  3:-3.24367,
                2:-1.32466,  1: 0.30990
    20 |      | 11:-4.04649, 10: 0.76459,  9: 0.95041,  8: 0.99810,  7: 0.70983,  6:
1.22526,  5:-1.24531,  4: 1.87435,  3: 0.47182,
                2: 1.45162,  1: 0.07065
    21 |      | 11:-2.52193, 10: 1.69767,  9:-4.22713,  8: 0.82103,  7: 3.57226,  6:-
2.55554,  5: 0.42443,  4: 0.18837,  3:-1.86768,
                2: 0.57690,  1: 1.03175
-------|------|-------------------------------------------------------------------------
-----------------------------------------
```

```
SNNS pattern definition file V1.4
generated at Tue Dec  3 20:36:50 1991

No. of patterns     :  64
No. of input units  :  8
No. of output units :  9

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 0 1 0 1 0 0 0 0 0 0 0 0 0
1 1 1 1 0 1 0 1 1 0 0 0 0 0 0 0 0
1 1 1 1 0 1 0 1 1 0 0 0 0 0 0 0 0
1 1 1 1 0 1 0 1 1 0 0 1 0 0 0 0 0
1 1 1 1 0 1 0 1 1 0 0 1 0 0 0 0 0
1 1 1 1 0 1 0 1 1 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0
1 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0
1 1 1 1 0 0 0 0 1 0 0 0 1 0 0 0 0
1 1 1 1 0 0 0 0 1 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 0 1 0 1 1 1 0 0 0 0 0 0 0 0 0
1 1 0 1 0 1 1 1 0 1 0 0 0 0 0 0 0
1 1 0 1 0 1 1 1 0 1 0 0 0 0 0 0 0
1 1 0 1 0 1 1 1 0 1 0 0 0 1 0 0 0
1 1 0 1 0 1 1 1 0 1 0 0 0 1 0 0 0
1 1 0 1 0 1 1 1 0 1 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0
1 1 0 1 0 0 1 1 0 1 0 0 0 0 0 0 0
1 1 0 1 0 0 1 1 0 1 0 0 0 0 0 0 0
1 1 0 1 0 0 1 1 0 1 0 0 0 0 1 0 0
1 1 0 1 0 0 1 1 0 1 0 0 0 0 1 0 0
1 1 0 1 0 0 1 1 0 1 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 1 1 0 0 0 0 1 0 0 0 0 0 0
1 0 0 0 1 1 0 0 0 0 1 0 0 0 0 0 0
1 0 0 0 1 1 0 0 0 0 1 0 0 0 0 1 0
1 0 0 0 1 1 0 0 0 0 1 0 0 0 0 1 0
1 0 0 0 1 1 0 0 0 0 1 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0
1 0 0 0 1 0 1 1 0 0 1 0 0 0 0 0 0
1 0 0 0 1 0 1 1 0 0 1 0 0 0 0 0 1
1 0 0 0 1 0 1 1 0 0 1 0 0 0 0 0 1
1 0 0 0 1 0 1 1 0 0 1 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0
1 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0
1 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0
1 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0
1 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0
1 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0
1 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0
1 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0
1 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0
1 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0
1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0
1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0
1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0
1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0
1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
SNNS 3D-Kernel V3.0002 Batchlearning Program
Configuration file: 'air.xcfb'
Log file          : 'air.xlog'



######## SNNS batch execution run. Loop 1 ########
Networkfile 'air.xnet' loaded.
No. of patterns: 0

No. of cycles: 0

Max. network error to stop: 0.000000

Patterns are in order

Network name             : aircraft
No. of units             : 21
No. of sites             : 0
No. of links             : 80

Learning Function        : BPTT


Test pattern file : 'air.xpat'

No Initialization


Result File              : 'air.xres'
Result File Start Pattern: 0
Result File End Pattern  : 0
Result File Input Pattern included
Result File Output Pattern included


*********************************************************

SNNS 3D-Kernel V3.0002 Result file computation started at Wed May 11 12:03:47 1994



Test Pattern File 'air.xpat' loaded.
No. of test patterns: 64


Result file saved.

Trained Network was not saved

SNNS 3D-Kernel V3.0002 Batchlearning terminated at Wed May 11 12:03:47 1994
System: ULTRIX  Node: amazonas  Machine: RISC
*********************************************************

----   STATISTICS  ----
No. of learned cycles: 0
No. of units updated : 0
No. of sites updated : 0
No. of links updated : 0
CPU Time used: 0.06 seconds
User time: 0 seconds
No. of connection updates per second (CUPS): 0.000000e+00
```

```
SNNS result file V1.4-3D
generated at Wed May 11 12:03:47 1994

No. of patterns     : 64
No. of input units  : 8
No. of output units : 9
startpattern        : 1
endpattern          : 64
input patterns included
teaching output included
#1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0.00886 0.0083 0.00756 0.00186 0.01798 0.00877 0.01673 0.01762 0.00449
#2
1 1 1 1 0 1 0 1
0 0 0 0 0 0 0 0 0
0.0222 0.01647 0.01873 0.00001 0.00056 0.00006 0.01187 0.00004 0.01524
#3
1 1 1 1 0 1 0 1
1 0 0 0 0 0 0 0 0
0.96557 0.02017 0.00292 0.02784 0.01524 0.00004 0.00008 0.00015 0
#4
1 1 1 1 0 1 0 1
1 0 0 0 0 0 0 0 0
0.96806 0.01906 0.00302 0.03371 0.01648 0.00005 0.00007 0.00017 0
#5
1 1 1 1 0 1 0 1
1 0 0 1 0 0 0 0 0
0.98954 0.00734 0.0273 0.93735 0.02683 0.00051 0 0.02423 0
#6
1 1 1 1 0 1 0 1
1 0 0 1 0 0 0 0 0
0.98951 0.00735 0.02745 0.93771 0.02671 0.00051 0 0.02435 0
#7
1 1 1 1 0 1 0 1
1 0 0 1 0 0 0 0 0
0.98917 0.00747 0.0296 0.94232 0.0251 0.00055 0 0.026 0
#8
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0.00886 0.0083 0.00756 0.00186 0.01798 0.00877 0.01673 0.01762 0.00449
#9
1 1 1 1 0 0 0 0
0 0 0 0 0 0 0 0 0
0.0222 0.01647 0.01873 0.00001 0.00056 0.00006 0.01187 0.00004 0.01524
#10
1 1 1 1 0 0 0 0
1 0 0 0 0 0 0 0 0
0.95971 0.02115 0.00012 0.00005 0.0465 0 0.01995 0 0.00001
#11
1 1 1 1 0 0 0 0
1 0 0 0 0 0 0 0 0
0.96184 0.02018 0.00012 0.00005 0.04985 0 0.01906 0 0.00001
#12
1 1 1 1 0 0 0 0
1 0 0 0 1 0 0 0 0
0.99971 0.00081 0.00004 0.04078 0.94514 0 0.00066 0.00028 0
#13
1 1 1 1 0 0 0 0
1 0 0 0 1 0 0 0 0
0.99971 0.00081 0.00004 0.04108 0.94547 0 0.00066 0.00029 0
#14
1 1 1 1 0 0 0 0
1 0 0 0 1 0 0 0 0
0.99974 0.00076 0.00004 0.04648 0.95064 0 0.00061 0.00032 0
#15
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0.00886 0.0083 0.00756 0.00186 0.01798 0.00877 0.01673 0.01762 0.00449
#16
1 1 0 1 0 1 1 1
0 0 0 0 0 0 0 0 0
0.0222 0.01647 0.01873 0.00001 0.00056 0.00006 0.01187 0.00004 0.01524
#17
1 1 0 1 0 1 1 1
0 1 0 0 0 0 0 0 0
0.01144 0.95148 0.04614 0.00736 0 0.05667 0.00034 0.00001 0.00056
#18
1 1 0 1 0 1 1 1
```

```
0 1 0 0 0 0 0 0
0.01135 0.95202 0.0459 0.0074 0 0.0575 0.00034 0.00001 0.00056
#19
1 1 0 1 0 1 1 1
0 1 0 0 0 1 0 0 0
0.00033 0.9997 0.00429 0.01546 0 0.94245 0.02468 0 0.00113
#20
1 1 0 1 0 1 1 1
0 1 0 0 0 1 0 0 0
0.00033 0.9997 0.00428 0.01545 0 0.94246 0.02469 0 0.00113
#21
1 1 0 1 0 1 1 1
0 1 0 0 0 1 0 0 0
0.00034 0.99971 0.00402 0.01499 0 0.94215 0.02663 0 0.00111
#22
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0.00886 0.0083 0.00756 0.00186 0.01798 0.00877 0.01673 0.01762 0.00449
#23
1 1 0 1 0 0 1 1
0 0 0 0 0 0 0 0 0
0.0222 0.01647 0.01873 0.00001 0.00056 0.00006 0.01187 0.00004 0.01524
#24
1 1 0 1 0 0 1 1
0 1 0 0 0 0 0 0 0
0.02157 0.9706 0.00116 0.00085 0.00002 0.02011 0.02752 0 0.00028
#25
1 1 0 1 0 0 1 1
0 1 0 0 0 0 0 0 0
0.02067 0.97148 0.00121 0.00089 0.00002 0.02143 0.02642 0 0.00028
#26
1 1 0 1 0 0 1 1
0 1 0 0 0 0 1 0 0
0.01818 0.99678 0.00001 0.00009 0.00016 0.03051 0.95507 0 0.00012
#27
1 1 0 1 0 0 1 1
0 1 0 0 0 0 1 0 0
0.01821 0.99678 0.00001 0.00009 0.00016 0.03044 0.95519 0 0.00012
#28
1 1 0 1 0 0 1 1
0 1 0 0 0 0 1 0 0
0.01862 0.99675 0.00001 0.00009 0.00017 0.02945 0.95731 0 0.00012
#29
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0.00886 0.0083 0.00756 0.00186 0.01798 0.00877 0.01673 0.01762 0.00449
#30
1 0 0 0 1 1 0 0
0 0 0 0 0 0 0 0 0
0.0222 0.01647 0.01873 0.00001 0.00056 0.00006 0.01187 0.00004 0.01524
#31
1 0 0 0 1 1 0 0
0 0 1 0 0 0 0 0 0
0.01252 0.00408 0.9707 0.00356 0.00001 0.00103 0 0.02606 0.01988
#32
1 0 0 0 1 1 0 0
0 0 1 0 0 0 0 0 0
0.01324 0.00366 0.97173 0.00398 0.00001 0.00102 0 0.03067 0.019
#33
1 0 0 0 1 1 0 0
0 0 1 0 0 0 0 1 0
0.01811 0.00004 0.99831 0.04007 0.00014 0.00074 0 0.9505 0.03551
#34
1 0 0 0 1 1 0 0
0 0 1 0 0 0 0 1 0
0.01819 0.00004 0.99831 0.04032 0.00014 0.00074 0 0.95077 0.03535
#35
1 0 0 0 1 1 0 0
0 0 1 0 0 0 0 1 0
0.01924 0.00004 0.9983 0.04381 0.00016 0.00074 0 0.95474 0.03337
#36
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0.00886 0.0083 0.00756 0.00186 0.01798 0.00877 0.01673 0.01762 0.00449
#37
1 0 0 0 1 0 1 1
0 0 0 0 0 0 0 0 0
0.0222 0.01647 0.01873 0.00001 0.00056 0.00006 0.01187 0.00004 0.01524
#38
1 0 0 0 1 0 1 1
```

```
0 0 1 0 0 0 0 0 0
0.00303 0.0422 0.94133 0.00029 0 0.00135 0.00001 0.00074 0.06304
#39
1 0 0 0 1 0 1 1
0 0 1 0 0 0 0 0 0
0.00289 0.04069 0.94464 0.00028 0 0.00135 0.00001 0.00078 0.0674
#40
1 0 0 0 1 0 1 1
0 0 1 0 0 0 0 0 1
0.00008 0.00221 0.99936 0.00002 0 0.00112 0 0.02697 0.93167
#41
1 0 0 0 1 0 1 1
0 0 1 0 0 0 0 0 1
0.00008 0.0022 0.99937 0.00002 0 0.00112 0 0.02706 0.93188
#42
1 0 0 0 1 0 1 1
0 0 1 0 0 0 0 0 1
0.00008 0.00209 0.9994 0.00002 0 0.0011 0 0.0285 0.93485
#43
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0.00886 0.0083 0.00756 0.00186 0.01798 0.00877 0.01673 0.01762 0.00449
#44
1 1 1 1 0 0 0 0
1 0 0 0 0 0 0 0 0
0.0222 0.01647 0.01873 0.00001 0.00056 0.00006 0.01187 0.00004 0.01524
#45
1 1 1 1 0 0 0 0
1 0 0 0 0 0 0 0 0
0.95971 0.02115 0.00012 0.00005 0.0465 0 0.01995 0 0.00001
#46
1 1 1 1 0 0 0 0
1 0 0 0 0 0 0 0 0
0.96184 0.02018 0.00012 0.00005 0.04985 0 0.01906 0 0.00001
#47
1 1 1 1 0 0 0 0
1 0 0 0 0 0 0 0 0
0.99971 0.00081 0.00004 0.04078 0.94514 0 0.00066 0.00028 0
#48
1 1 1 1 0 0 0 0
1 0 0 0 0 0 0 0 0
0.99971 0.00081 0.00004 0.04108 0.94547 0 0.00066 0.00029 0
#49
1 1 1 1 0 0 0 0
1 0 0 0 0 0 0 0 0
0.99974 0.00076 0.00004 0.04648 0.95064 0 0.00061 0.00032 0
#50
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0.00886 0.0083 0.00756 0.00186 0.01798 0.00877 0.01673 0.01762 0.00449
#51
1 1 0 1 0 0 0 0
0 1 0 0 0 0 0 0 0
0.0222 0.01647 0.01873 0.00001 0.00056 0.00006 0.01187 0.00004 0.01524
#52
1 1 0 1 0 0 0 0
0 1 0 0 0 0 0 0 0
0.71581 0.16469 0.00009 0.00002 0.00664 0.00001 0.10942 0 0.00005
#53
1 1 0 1 0 0 0 0
0 1 0 0 0 0 0 0 0
0.71483 0.16384 0.00009 0.00002 0.00666 0.00001 0.10855 0 0.00005
#54
1 1 0 1 0 0 0 0
0 1 0 0 0 0 0 0 0
0.96415 0.03485 0.00008 0.00029 0.0675 0.00001 0.01718 0 0.00001
#55
1 1 0 1 0 0 0 0
0 1 0 0 0 0 0 0 0
0.96407 0.03485 0.00008 0.00029 0.06727 0.00001 0.01719 0 0.00001
#56
1 1 0 1 0 0 0 0
0 1 0 0 0 0 0 0 0
0.99591 0.00543 0.00012 0.00549 0.40404 0 0.00161 0.00004 0
#57
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0.00886 0.0083 0.00756 0.00186 0.01798 0.00877 0.01673 0.01762 0.00449
#58
1 0 0 0 1 0 0 0
```

```
0 0 1 0 0 0 0 0 0
0.0222 0.01647 0.01873 0.00001 0.00056 0.00006 0.01187 0.00004 0.01524
#59
1 0 0 0 1 0 0 0
0 0 1 0 0 0 0 0 0
0.00204 0.00128 0.9495 0.00001 0.00001 0.00005 0.00003 0.00209 0.44713
#60
1 0 0 0 1 0 0 0
0 0 1 0 0 0 0 0 0
0.00196 0.00126 0.95322 0.00001 0.00001 0.00006 0.00003 0.00226 0.45602
#61
1 0 0 0 1 0 0 0
0 0 1 0 0 0 0 0 0
0.00084 0.00036 0.99831 0.00016 0 0.00043 0 0.16875 0.59849
#62
1 0 0 0 1 0 0 0
0 0 1 0 0 0 0 0 0
0.00084 0.00035 0.99832 0.00016 0 0.00043 0 0.1713 0.59577
#63
1 0 0 0 1 0 0 0
0 0 1 0 0 0 0 0 0
0.00094 0.00032 0.99837 0.00021 0.00001 0.00045 0 0.20516 0.56106
#64
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0.00886 0.0083 0.00756 0.00186 0.01798 0.00877 0.01673 0.01762 0.00449
```

Anexo D

Este anexo contém os arquivos da simulação do exemplo da Detecção de Falhas em Transformadores utilizando o SNNS v3.1.

ARQUIVO	DESCRIÇÃO
trans.cfb	descrição do treinamento
trans.net	descrição da rede
trans.pat	padrões de treinamento
trans.log	relatório do treinamento
trans.res	resultado do treinamento
trans.trn	descrição da rede treinada
trans.xcfb	descrição do teste
trans.xnet	descrição da rede
trans.xpat	padrões de teste
trans.xlog	relatório do teste
trans.xres	resultado do teste

```
#
Type: SNNSBATCH_1
#
# This is the style for an snnsbat configuration file, type 1.
#
# The following keyword-value combinations may be supplied in any order.
# If a key is given twice, the second appearence is taken.
# Keys that are not required for a special run may be omitted.
# If a key is ommited but required, a default value is assumed.
# The lines may be separated with comments.
#
# Please note the mandatory file type specification at the begining and
# the colon following the key.
#
#
#                                              Defaults:
#
#     NoOfLearnParam:       <int>                 0
#     LearnParam:           [ <float> ]*          0.0
#     NoOfInitParam:        <int>                 0
#     InitParam:            [ <float> ]*          0.0
#     NetworkFile:          <string>              ""
#     TrainedNetworkFile:   <string>              ""
#     LearnPatternFile:     <string>              ""
#     TestPatternFile:      <string>              ""
#     ResultFile:           <string>              ""
#     InitFunction:         <string>              ""
#     MaxLearnCycles:       <int>                 0
#     MaxErrorToStop:       <float>               0.0
#     Shuffle:              [ YES | NO ]          NO
#     ResultMinMaxPattern: <int><int>             0 0
#     ResultIncludeInput:   [ YES | NO ]          NO
#     ResultIncludeOutput: | YES | NO ]           YES
#
#########################################################################
NetworkFile: trans.net
#
InitFunction: Randomize_Weights
NoOfInitParam: 2
InitParam: -0.1 0.1
#
LearnPatternFile: trans.pat
NoOfLearnParam: 3
LearnParam: 0.005 0.9 2
MaxLearnCycles: 5000
MaxErrorToStop: 0.1
Shuffle: NO
#
TrainedNetworkFile: trans.trn
TestPatternFile: trans.pat
ResultFile: trans.res
ResultIncludeInput: YES
ResultIncludeOutput: YES
```

```
SNNS network definition file V1.4-3D
generated at Sat Apr 16 15:03:26 1994

network name : trans
source files :
no. of units : 30
no. of connections : 243
no. of unit types : 2
no. of site types : 0


learning function : BPTT
update function   : BPTT_Order


type definition section :

name           | act func     | out func     |sites
---------------|--------------|--------------|-------
LongeroutType | Act_Logistic | Out_Identity |
outType        | Act_Logistic | Out_Identity |
---------------|--------------|--------------|-------


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 |               | g1_min   |  0.00000 |  0.00000 | i  |  1,11, 0 |||
  2 |               | g1_norm  |  0.00000 |  0.00000 | i  |  2,11, 0 |||
  3 |               | g1_max   |  0.00000 |  0.00000 | i  |  3,11, 0 |||
  4 |               | g2_min   |  0.00000 |  0.00000 | i  |  4,11, 0 |||
  5 |               | g2_norm  |  0.00000 |  0.00000 | i  |  5,11, 0 |||
  6 |               | g2_max   |  0.00000 |  0.00000 | i  |  6,11, 0 |||
  7 |               | g3_min   |  0.00000 |  0.00000 | i  |  7,11, 0 |||
  8 |               | g3_norm  |  0.00000 |  0.00000 | i  |  8,11, 0 |||
  9 |               | g3_max   |  0.00000 |  0.00000 | i  |  9,11, 0 |||
 10 |               | descarga |  0.00000 |  0.00000 | d  | 11, 3, 0 |||
 11 |               | termica  |  0.00000 |  0.00000 | d  | 12, 3, 0 |||
 12 |               | parcial  |  0.00000 |  0.00000 | d  | 13, 3, 0 |||
 13 |               | arco     |  0.00000 |  0.00000 | d  | 14, 3, 0 |||
 14 |               | t_alta   |  0.00000 |  0.00000 | d  | 15, 3, 0 |||
 15 |               | t_baixa  |  0.00000 |  0.00000 | d  | 16, 3, 0 |||
 16 |               | p_alta   |  0.00000 |  0.00000 | o  |  3, 1, 0 |||
 17 |               | p_baixa  |  0.00000 |  0.00000 | o  |  4, 1, 0 |||
 18 |               | a_alta   |  0.00000 |  0.00000 | o  |  5, 1, 0 |||
 19 |               | a_baixa  |  0.00000 |  0.00000 | o  |  6, 1, 0 |||
 20 |               | t_malta  |  0.00000 |  0.00000 | o  |  7, 1, 0 |||
 21 |               | t_mbaixa |  0.00000 |  0.00000 | o  |  8, 1, 0 |||
 22 |               | hidden   |  0.00000 |  0.00000 | h  |  1, 7, 0 |||
 23 |               | hidden   |  0.00000 |  0.00000 | h  |  2, 7, 0 |||
 24 |               | hidden   |  0.00000 |  0.00000 | h  |  3, 7, 0 |||
 25 |               | hidden   |  0.00000 |  0.00000 | h  |  4, 7, 0 |||
 26 |               | hidden   |  0.00000 |  0.00000 | h  |  5, 7, 0 |||
 27 |               | hidden   |  0.00000 |  0.00000 | h  |  6, 7, 0 |||
 28 |               | hidden   |  0.00000 |  0.00000 | h  |  7, 7, 0 |||
 29 |               | hidden   |  0.00000 |  0.00000 | h  |  8, 7, 0 |||
 30 |               | hidden   |  0.00000 |  0.00000 | h  |  9, 7, 0 |||
----|---------------|----------|----------|----------|----|----------|----------|-----
-----|-------


connection definition section :

target | site | source:weight
-------|------|-------------------------------------------------------------------------
-----------------------------------------
    10 |      | 22: 0.00000, 23: 0.00000, 24: 0.00000, 25: 0.00000, 26: 0.00000, 27:
0.00000, 28: 0.00000, 29: 0.00000, 30: 0.00000
```

```
    11 |       | 22: 0.00000, 23: 0.00000, 24: 0.00000, 25: 0.00000, 26: 0.00000, 27:
0.00000, 28: 0.00000, 29: 0.00000, 30: 0.00000
    12 |       | 22: 0.00000, 23: 0.00000, 24: 0.00000, 25: 0.00000, 26: 0.00000, 27:
0.00000, 28: 0.00000, 29: 0.00000, 30: 0.00000
    13 |       | 22: 0.00000, 23: 0.00000, 24: 0.00000, 25: 0.00000, 26: 0.00000, 27:
0.00000, 28: 0.00000, 29: 0.00000, 30: 0.00000
    14 |       | 22: 0.00000, 23: 0.00000, 24: 0.00000, 25: 0.00000, 26: 0.00000, 27:
0.00000, 28: 0.00000, 29: 0.00000, 30: 0.00000
    15 |       | 22: 0.00000, 23: 0.00000, 24: 0.00000, 25: 0.00000, 26: 0.00000, 27:
0.00000, 28: 0.00000, 29: 0.00000, 30: 0.00000
    16 |       | 22: 0.00000, 23: 0.00000, 24: 0.00000, 25: 0.00000, 26: 0.00000, 27:
0.00000, 28: 0.00000, 29: 0.00000, 30: 0.00000
    17 |       | 22: 0.00000, 23: 0.00000, 24: 0.00000, 25: 0.00000, 26: 0.00000, 27:
0.00000, 28: 0.00000, 29: 0.00000, 30: 0.00000
    18 |       | 22: 0.00000, 23: 0.00000, 24: 0.00000, 25: 0.00000, 26: 0.00000, 27:
0.00000, 28: 0.00000, 29: 0.00000, 30: 0.00000
    19 |       | 22: 0.00000, 23: 0.00000, 24: 0.00000, 25: 0.00000, 26: 0.00000, 27:
0.00000, 28: 0.00000, 29: 0.00000, 30: 0.00000
    20 |       | 22: 0.00000, 23: 0.00000, 24: 0.00000, 25: 0.00000, 26: 0.00000, 27:
0.00000, 28: 0.00000, 29: 0.00000, 30: 0.00000
    21 |       | 22: 0.00000, 23: 0.00000, 24: 0.00000, 25: 0.00000, 26: 0.00000, 27:
0.00000, 28: 0.00000, 29: 0.00000, 30: 0.00000
    22 |       | 10: 0.00000, 11: 0.00000, 12: 0.00000, 13: 0.00000, 14: 0.00000, 15:
0.00000,  1: 0.00000,  2: 0.00000,  3: 0.00000,
                 4: 0.00000,  5: 0.00000,  6: 0.00000,  7: 0.00000,  8: 0.00000,  9:
0.00000
    23 |       | 10: 0.00000, 11: 0.00000, 12: 0.00000, 13: 0.00000, 14: 0.00000, 15:
0.00000,  1: 0.00000,  2: 0.00000,  3: 0.00000,
                 4: 0.00000,  5: 0.00000,  6: 0.00000,  7: 0.00000,  8: 0.00000,  9:
0.00000
    24 |       | 10: 0.00000, 11: 0.00000, 12: 0.00000, 13: 0.00000, 14: 0.00000, 15:
0.00000,  1: 0.00000,  2: 0.00000,  3: 0.00000,
                 4: 0.00000,  5: 0.00000,  6: 0.00000,  7: 0.00000,  8: 0.00000,  9:
0.00000
    25 |       | 10: 0.00000, 11: 0.00000, 12: 0.00000, 13: 0.00000, 14: 0.00000, 15:
0.00000,  1: 0.00000,  2: 0.00000,  3: 0.00000,
                 4: 0.00000,  5: 0.00000,  6: 0.00000,  7: 0.00000,  8: 0.00000,  9:
0.00000
    26 |       | 10: 0.00000, 11: 0.00000, 12: 0.00000, 13: 0.00000, 14: 0.00000, 15:
0.00000,  1: 0.00000,  2: 0.00000,  3: 0.00000,
                 4: 0.00000,  5: 0.00000,  6: 0.00000,  7: 0.00000,  8: 0.00000,  9:
0.00000
    27 |       | 10: 0.00000, 11: 0.00000, 12: 0.00000, 13: 0.00000, 14: 0.00000, 15:
0.00000,  1: 0.00000,  2: 0.00000,  3: 0.00000,
                 4: 0.00000,  5: 0.00000,  6: 0.00000,  7: 0.00000,  8: 0.00000,  9:
0.00000
    28 |       | 10: 0.00000, 11: 0.00000, 12: 0.00000, 13: 0.00000, 14: 0.00000, 15:
0.00000,  1: 0.00000,  2: 0.00000,  3: 0.00000,
                 4: 0.00000,  5: 0.00000,  6: 0.00000,  7: 0.00000,  8: 0.00000,  9:
0.00000
    29 |       | 10: 0.00000, 11: 0.00000, 12: 0.00000, 13: 0.00000, 14: 0.00000, 15:
0.00000,  1: 0.00000,  2: 0.00000,  3: 0.00000,
                 4: 0.00000,  5: 0.00000,  6: 0.00000,  7: 0.00000,  8: 0.00000,  9:
0.00000
    30 |       | 10: 0.00000, 11: 0.00000, 12: 0.00000, 13: 0.00000, 14: 0.00000, 15:
0.00000,  1: 0.00000,  2: 0.00000,  3: 0.00000,
                 4: 0.00000,  5: 0.00000,  6: 0.00000,  7: 0.00000,  8: 0.00000,  9:
0.00000
-------|------|-------------------------------------------------------------------------
-----------------------------------------
```

```
SNNS pattern definition file V1.4
generated at Tue Dec  3 20:36:50 1991

No. of patterns     : 89
No. of input units  : 15
No. of output units : 12

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0
1 0 0 1 0 0 1 0 0 1 0 1 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0
1 0 0 1 0 0 1 0 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0
1 0 0 1 0 0 1 0 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 1 0 1 0 0 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 1 0 1 0 0 1 0 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0
0 1 0 1 0 0 1 0 0 1 0 1 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0
0 1 0 1 0 0 1 0 0 1 0 1 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 1 0 1 0 0 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0
0 1 0 0 1 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0
0 1 0 0 1 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 1 0 0
0 1 0 0 1 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 1 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 1 0 0 1 0 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0
0 1 0 0 1 0 0 1 0 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0 0
0 1 0 0 1 0 0 1 0 1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 1 0 0 1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 1 0 0 1 0 0 1 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0
0 0 1 0 1 0 0 1 0 0 1 0 0 1 0 1 0 0 1 0 0 0 0 0 1 0 0
0 0 1 0 1 0 0 1 0 0 1 0 0 1 0 1 0 0 1 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 1 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 1 0 0 1 0 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0
0 1 0 0 1 0 0 1 0 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0 0
0 1 0 0 1 0 0 1 0 1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 1 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 1 0 0 1 0 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0
0 0 1 0 1 0 0 1 0 1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 1 0 0
0 0 1 0 1 0 0 1 0 1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 1 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 1 0 0 0 1 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0
0 1 0 0 1 0 0 0 1 1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0
0 1 0 0 1 0 0 0 1 1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
1 0 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0
1 0 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 1 0
1 0 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 1 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 1 1 0 0 0 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0
1 0 0 0 0 1 1 0 0 0 1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0
1 0 0 0 0 1 1 0 0 0 1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0
1 0 0 0 0 1 1 0 0 0 1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0
1 0 0 0 0 1 0 1 0 0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0
1 0 0 0 0 1 0 1 0 0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0
1 0 0 0 0 1 0 1 0 0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0
1 0 0 0 0 1 0 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0
1 0 0 0 0 1 0 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 1
1 0 0 0 0 1 0 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

```
SNNS 3D-Kernel V3.0002 Batchlearning Program
Configuration file: 'trans.cfb'
Log file          : 'trans.log'



######## SNNS batch execution run. Loop 1 ########
Networkfile 'trans.net' loaded.
Patternfile 'trans.pat' loaded.
No. of patterns: 89

No. of cycles: 5000

Max. network error to stop: 0.100000

Patterns are in order

Network name             : trans
No. of units             : 30
No. of sites             : 0
No. of links             : 243

Learning Function        : BPTT
Learning Parameter #1    : 0.005000
Learning Parameter #2    : 0.900000
Learning Parameter #3    : 2.000000


Test pattern file : 'trans.pat'

Init Function: Randomize_Weights
Init Parameter #1       : -0.100000
Init Parameter #2       : 0.100000


Result File              : 'trans.res'
Result File Start Pattern: 0
Result File End Pattern  : 0
Result File Input Pattern included
Result File Output Pattern included


*******************************************************

SNNS 3D-Kernel V3.0002 Batchlearning started at Wed May 11 13:54:30 1994


Network initialized with
Randomize_Weights -0.10 0.10

Cycle: 0
Learning function value(s): [1]: 176.805

Cycle: 50
Learning function value(s): [1]: 44.1475

Cycle: 100
Learning function value(s): [1]: 28.1618

Cycle: 150
Learning function value(s): [1]: 22.5745

Cycle: 200
Learning function value(s): [1]: 15.6632

Cycle: 250
Learning function value(s): [1]: 12.1789

Cycle: 300
Learning function value(s): [1]: 10.3704

Cycle: 350
Learning function value(s): [1]: 9.28607

Cycle: 400
Learning function value(s): [1]: 8.5241

Cycle: 450
Learning function value(s): [1]: 7.91572
```

```
Cycle: 500
Learning function value(s): [1]: 7.41083

Cycle: 550
Learning function value(s): [1]: 6.98054

Cycle: 600
Learning function value(s): [1]: 6.58317

Cycle: 650
Learning function value(s): [1]: 6.16548

Cycle: 700
Learning function value(s): [1]: 5.68422

Cycle: 750
Learning function value(s): [1]: 5.16219

Cycle: 800
Learning function value(s): [1]: 4.64869

Cycle: 850
Learning function value(s): [1]: 4.17298

Cycle: 900
Learning function value(s): [1]: 3.75704

Cycle: 950
Learning function value(s): [1]: 3.40465

Cycle: 1000
Learning function value(s): [1]: 3.10379

Cycle: 1050
Learning function value(s): [1]: 2.83577

Cycle: 1100
Learning function value(s): [1]: 2.57688

Cycle: 1150
Learning function value(s): [1]: 2.30443

Cycle: 1200
Learning function value(s): [1]: 2.03209

Cycle: 1250
Learning function value(s): [1]: 1.79579

Cycle: 1300
Learning function value(s): [1]: 1.60423

Cycle: 1350
Learning function value(s): [1]: 1.4496

Cycle: 1400
Learning function value(s): [1]: 1.32245

Cycle: 1450
Learning function value(s): [1]: 1.21565

Cycle: 1500
Learning function value(s): [1]: 1.12438

Cycle: 1550
Learning function value(s): [1]: 1.04531

Cycle: 1600
Learning function value(s): [1]: 0.976079

Cycle: 1650
Learning function value(s): [1]: 0.914927

Cycle: 1700
Learning function value(s): [1]: 0.860511

Cycle: 1750
Learning function value(s): [1]: 0.811787

Cycle: 1800
```

```
Learning function value(s): [1]: 0.767918

Cycle: 1850
Learning function value(s): [1]: 0.728226

Cycle: 1900
Learning function value(s): [1]: 0.692154

Cycle: 1950
Learning function value(s): [1]: 0.659236

Cycle: 2000
Learning function value(s): [1]: 0.629097

Cycle: 2050
Learning function value(s): [1]: 0.601402

Cycle: 2100
Learning function value(s): [1]: 0.57588

Cycle: 2150
Learning function value(s): [1]: 0.552288

Cycle: 2200
Learning function value(s): [1]: 0.530424

Cycle: 2250
Learning function value(s): [1]: 0.510109

Cycle: 2300
Learning function value(s): [1]: 0.4912

Cycle: 2350
Learning function value(s): [1]: 0.473549

Cycle: 2400
Learning function value(s): [1]: 0.45704

Cycle: 2450
Learning function value(s): [1]: 0.441574

Cycle: 2500
Learning function value(s): [1]: 0.427057

Cycle: 2550
Learning function value(s): [1]: 0.413403

Cycle: 2600
Learning function value(s): [1]: 0.400545

Cycle: 2650
Learning function value(s): [1]: 0.388419

Cycle: 2700
Learning function value(s): [1]: 0.376957

Cycle: 2750
Learning function value(s): [1]: 0.366112

Cycle: 2800
Learning function value(s): [1]: 0.35584

Cycle: 2850
Learning function value(s): [1]: 0.346094

Cycle: 2900
Learning function value(s): [1]: 0.33684

Cycle: 2950
Learning function value(s): [1]: 0.32804

Cycle: 3000
Learning function value(s): [1]: 0.319662

Cycle: 3050
Learning function value(s): [1]: 0.311682

Cycle: 3100
Learning function value(s): [1]: 0.30407
```

```
Cycle: 3150
Learning function value(s): [1]: 0.296799

Cycle: 3200
Learning function value(s): [1]: 0.28985

Cycle: 3250
Learning function value(s): [1]: 0.283203

Cycle: 3300
Learning function value(s): [1]: 0.276844

Cycle: 3350
Learning function value(s): [1]: 0.270747

Cycle: 3400
Learning function value(s): [1]: 0.264895

Cycle: 3450
Learning function value(s): [1]: 0.259281

Cycle: 3500
Learning function value(s): [1]: 0.25389

Cycle: 3550
Learning function value(s): [1]: 0.248708

Cycle: 3600
Learning function value(s): [1]: 0.243724

Cycle: 3650
Learning function value(s): [1]: 0.238926

Cycle: 3700
Learning function value(s): [1]: 0.234305

Cycle: 3750
Learning function value(s): [1]: 0.229852

Cycle: 3800
Learning function value(s): [1]: 0.225557

Cycle: 3850
Learning function value(s): [1]: 0.221413

Cycle: 3900
Learning function value(s): [1]: 0.217412

Cycle: 3950
Learning function value(s): [1]: 0.213548

Cycle: 4000
Learning function value(s): [1]: 0.209813

Cycle: 4050
Learning function value(s): [1]: 0.206202

Cycle: 4100
Learning function value(s): [1]: 0.202706

Cycle: 4150
Learning function value(s): [1]: 0.199319

Cycle: 4200
Learning function value(s): [1]: 0.19604

Cycle: 4250
Learning function value(s): [1]: 0.192863

Cycle: 4300
Learning function value(s): [1]: 0.189783

Cycle: 4350
Learning function value(s): [1]: 0.186799

Cycle: 4400
Learning function value(s): [1]: 0.183904

Cycle: 4450
Learning function value(s): [1]: 0.181092
```

```
Cycle: 4500
Learning function value(s): [1]: 0.178361

Cycle: 4550
Learning function value(s): [1]: 0.175707

Cycle: 4600
Learning function value(s): [1]: 0.173129

Cycle: 4650
Learning function value(s): [1]: 0.170623

Cycle: 4700
Learning function value(s): [1]: 0.168187

Cycle: 4750
Learning function value(s): [1]: 0.165818

Cycle: 4800
Learning function value(s): [1]: 0.16351

Cycle: 4850
Learning function value(s): [1]: 0.161263

Cycle: 4900
Learning function value(s): [1]: 0.159074

Cycle: 4950
Learning function value(s): [1]: 0.15694

Result file saved.

Network saved to trans.trn.

SNNS 3D-Kernel V3.0002 Batchlearning terminated at Wed May 11 14:11:32 1994
System: ULTRIX  Node: amazonas  Machine: RISC
******************************************************

----   STATISTICS  ----
No. of learned cycles: 5000
No. of units updated : 13350000
No. of sites updated : 0
No. of links updated : 108135000
CPU Time used: 265.72 seconds
User time: 1022 seconds
No. of connection updates per second (CUPS): 4.069509e+05
```

```
SNNS result file V1.4-3D
generated at Wed May 11 14:11:31 1994

No. of patterns     : 89
No. of input units  : 15
No. of output units : 12
startpattern        : 1
endpattern          : 89
input patterns included
teaching output included
#1
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0
0.00462 0.01314 0.00928 0.005 0.00954 0.01759 0.00066 0.01194 0.00962 0.00407
0.00841 0.00862
#2
1 0 0 1 0 0 1 0 0 0
0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0
0.01184 0.01081 0.00004 0.00425 0.00046 0.0039 0.00004 0.00002 0.0001 0.00116
0.00001 0.00016
#3
1 0 0 1 0 0 1 0 0 0
0 0 0 0 0
1 0 0 0 0 0 0 0 0 0
0 0
0.9691 0.02838 0.03586 0.00001 0.00003 0.0022 0.00158 0.00009 0 0.00003
0 0.00002
#4
1 0 0 1 0 0 1 0 0 1
0 0 0 0 0
1 0 0 0 0 0 0 0 0 0
0 0
0.97152 0.02817 0.03743 0.00001 0.00003 0.00199 0.00155 0.00009 0 0.00003
0 0.00002
#5
1 0 0 1 0 0 1 0 0 1
0 0 0 0 0
1 0 1 0 0 0 0 0 0 0
0 0
0.99904 0.00672 0.95731 0.00079 0 0.00012 0.03336 0.00137 0.00001 0.00003
0 0
#6
1 0 0 1 0 0 1 0 0 1
0 1 0 0 0
1 0 1 0 0 0 0 0 0 0
0 0
0.99904 0.00671 0.95787 0.00079 0 0.00012 0.03374 0.00138 0.00001 0.00003
0 0
#7
1 0 0 1 0 0 1 0 0 1
0 1 0 0 0
1 0 1 0 0 0 1 0 0 0
0 0
0.99884 0.00336 0.99936 0.00011 0 0.00675 0.93783 0.03982 0 0.00003
0 0.00032
#8
1 0 0 1 0 0 1 0 0 1
0 1 0 0 0
1 0 1 0 0 0 1 0 0 0
0 0
0.99884 0.00336 0.99936 0.00011 0 0.00676 0.93787 0.03985 0 0.00003
0 0.00033
#9
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0
0.00462 0.01314 0.00928 0.005 0.00954 0.01759 0.00066 0.01194 0.00962 0.00407
0.00841 0.00862
#10
0 1 0 1 0 0 1 0 0 0
0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0
0.01184 0.01081 0.00004 0.00425 0.00046 0.0039 0.00004 0.00002 0.0001 0.00116
0.00001 0.00016
```

```
#11
0 1 0 1 0 0 1 0 0 0
0 0 0 0 0
1 0 0 0 0 0 0 0 0 0
0 0
0.9882 0.00844 0.02302 0.00002 0.00006 0.00036 0.00037 0.0002 0.00001 0.00016
0 0.00001
#12
0 1 0 1 0 0 1 0 0 1
0 0 0 0 0
1 0 0 0 0 0 0 0 0 0
0 0
0.98894 0.00847 0.02355 0.00002 0.00006 0.00033 0.00037 0.0002 0.00001 0.00017
0 0.00001
#13
0 1 0 1 0 0 1 0 0 1
0 0 0 0 0
1 0 1 0 0 0 0 0 0 0
0 0
0.99984 0.00025 0.96504 0.02351 0 0 0.00108 0.02528 0.00007 0.00235
0 0
#14
0 1 0 1 0 0 1 0 0 1
0 1 0 0 0
1 0 1 0 0 0 0 0 0 0
0 0
0.99984 0.00025 0.96529 0.0236 0 0 0.00108 0.02545 0.00007 0.00236
0 0
#15
0 1 0 1 0 0 1 0 0 1
0 1 0 0 0
1 0 1 0 0 0 0 1 0 0
0 0
0.99995 0.00001 0.99992 0.02841 0 0 0.04605 0.94486 0.00007 0.02636
0 0.00002
#16
0 1 0 1 0 0 1 0 0 1
0 1 0 0 0
1 0 1 0 0 0 0 1 0 0
0 0
0.99995 0.00001 0.99992 0.02841 0 0 0.04612 0.94494 0.00007 0.02637
0 0.00002
#17
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0
0.00462 0.01314 0.00928 0.005 0.00954 0.01759 0.00066 0.01194 0.00962 0.00407
0.00841 0.00862
#18
0 1 0 0 1 0 1 0 0 0
0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0
0.01184 0.01081 0.00004 0.00425 0.00046 0.0039 0.00004 0.00002 0.0001 0.00116
0.00001 0.00016
#19
0 1 0 0 1 0 1 0 0 0
0 0 0 0 0
1 0 0 0 0 0 0 0 0 0
0 0
0.99684 0.00955 0.00001 0.01515 0.01814 0.00001 0.00002 0 0.00033 0.00188
0.00005 0
#20
0 1 0 0 1 0 1 0 0 1
0 0 0 0 0
1 0 0 0 0 0 0 0 0 0
0 0
0.99694 0.00904 0.00001 0.01693 0.01687 0.00001 0.00002 0 0.00034 0.00196
0.00005 0
#21
0 1 0 0 1 0 1 0 0 1
0 0 0 0 0
1 0 0 1 0 0 0 0 0 0
0 0
0.99986 0.00032 0.00128 0.98058 0.00044 0 0.00007 0.00004 0.00559 0.00675
0.00001 0
#22
0 1 0 0 1 0 1 0 0 1
0 0 1 0 0
```

```
1 0 0 1 0 0 0 0 0 0
0 0
0.99986 0.00031 0.00128 0.98113 0.00044 0 0.00007 0.00004 0.00569 0.00678
0.00001 0
#23
0 1 0 0 1 0 1 0 0 1
0 0 1 0 0
1 0 0 1 0 0 0 0 1 0
0 0
0.99999 0 0.00107 1 0.00018 0 0.00001 0.00104 0.98129 0.02192
0.00003 0
#24
0 1 0 0 1 0 1 0 0 1
0 0 1 0 0
1 0 0 1 0 0 0 0 1 0
0 0
0.99999 0 0.00108 1 0.00018 0 0.00001 0.00105 0.98133 0.02197
0.00003 0
#25
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0
0.00462 0.01314 0.00928 0.005 0.00954 0.01759 0.00066 0.01194 0.00962 0.00407
0.00841 0.00862
#26
0 0 1 0 1 0 1 0 0 0
0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0
0.01184 0.01081 0.00004 0.00425 0.00046 0.0039 0.00004 0.00002 0.0001 0.00116
0.00001 0.00016
#27
0 0 1 0 1 0 1 0 0 0
0 0 0 0 0
1 0 0 0 0 0 0 0 0 0
0 0
0.99719 0.01007 0.00002 0.01743 0.01786 0.00001 0.00002 0 0.00037 0.00161
0.00005 0
#28
0 0 1 0 1 0 1 0 0 1
0 0 0 0 0
1 0 0 0 0 0 0 0 0 0
0 0
0.99729 0.00952 0.00002 0.01956 0.01655 0.00001 0.00002 0 0.00038 0.00168
0.00005 0
#29
0 0 1 0 1 0 1 0 0 1
0 0 0 0 0
1 0 0 1 0 0 0 0 0 0
0 0
0.99987 0.00036 0.00148 0.9824 0.00044 0 0.00009 0.00003 0.00625 0.00498
0.00001 0
#30
0 0 1 0 1 0 1 0 0 1
0 0 1 0 0
1 0 0 1 0 0 0 0 0 0
0 0
0.99987 0.00035 0.00147 0.98297 0.00044 0 0.00009 0.00003 0.00638 0.005
0.00001 0
#31
0 0 1 0 1 0 1 0 0 1
0 0 1 0 0
1 0 0 1 0 0 0 0 1 0
0 0
0.99999 0 0.00085 1 0.0002 0 0.00001 0.00062 0.97732 0.01311
0.00003 0
#32
0 0 1 0 1 0 1 0 0 1
0 0 1 0 0
1 0 0 1 0 0 0 0 1 0
0 0
0.99999 0 0.00085 1 0.0002 0 0.00001 0.00062 0.97737 0.01314
0.00003 0
#33
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0
0.00462 0.01314 0.00928 0.005 0.00954 0.01759 0.00066 0.01194 0.00962 0.00407
```

```
0.00841 0.00862
#34
0 1 0 0 1 0 0 1 0 0
0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0
0.01184 0.01081 0.00004 0.00425 0.00046 0.0039 0.00004 0.00002 0.0001 0.00116
0.00001 0.00016
#35
0 1 0 0 1 0 0 1 0 0
0 0 0 0 0
1 0 0 0 0 0 0 0 0 0
0 0
0.98358 0.00378 0.00009 0.01328 0.00027 0.00004 0.00006 0.00001 0.00058 0.00011
0.00001 0
#36
0 1 0 0 1 0 0 1 0 1
0 0 0 0 0
1 0 0 0 0 0 0 0 0 0
0 0
0.98478 0.00337 0.00009 0.01525 0.00026 0.00004 0.00006 0.00001 0.00061 0.00012
0.00001 0
#37
0 1 0 0 1 0 0 1 0 1
0 0 0 0 0
1 0 0 1 0 0 0 0 0 0
0 0
0.99962 0.00003 0.01033 0.97802 0.00001 0 0.00037 0.00011 0.01475 0.00027
0 0
#38
0 1 0 0 1 0 0 1 0 1
0 0 1 0 0
1 0 0 1 0 0 0 0 0 0
0 0
0.99962 0.00003 0.01019 0.97896 0.00001 0 0.00037 0.00011 0.01525 0.00028
0 0
#39
0 1 0 0 1 0 0 1 0 1
0 0 1 0 0
1 0 0 1 0 0 0 0 1 0
0 0
0.99999 0 0.00111 1 0.00017 0 0.00001 0.00107 0.98433 0.01953
0.00003 0
#40
0 1 0 0 1 0 0 1 0 1
0 0 1 0 0
1 0 0 1 0 0 0 0 1 0
0 0
0.99999 0 0.00111 1 0.00017 0 0.00001 0.00107 0.98437 0.01961
0.00003 0
#41
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0
0.00462 0.01314 0.00928 0.005 0.00954 0.01759 0.00066 0.01194 0.00962 0.00407
0.00841 0.00862
#42
0 0 1 0 1 0 0 1 0 0
0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0
0.01184 0.01081 0.00004 0.00425 0.00046 0.0039 0.00004 0.00002 0.0001 0.00116
0.00001 0.00016
#43
0 0 1 0 1 0 0 1 0 0
0 0 0 0 0
1 0 0 0 0 0 0 0 0 0
0 0
0.98209 0.00499 0.00011 0.01656 0.00033 0.00004 0.00007 0.00001 0.00069 0.0001
0.00001 0
#44
0 0 1 0 1 0 0 1 0 1
0 0 0 0 0
1 0 0 0 0 0 0 0 0 0
0 0
0.98363 0.0044 0.0001 0.01922 0.00032 0.00004 0.00007 0.00001 0.00073 0.00011
0.00001 0
#45
0 0 1 0 1 0 0 1 0 1
```

```
0 0 0 0 0
1 0 0 1 0 0 0 0 0 0
0 0
0.99964 0.00004 0.01147 0.98083 0.00001 0 0.00053 0.0001 0.01741 0.0002
0 0
#46
0 0 1 0 1 0 0 1 0 1
0 0 1 0 0
1 0 0 1 0 0 0 0 0 0
0 0
0.99964 0.00004 0.01121 0.9819 0.00001 0 0.00052 0.0001 0.01819 0.00021
0 0
#47
0 0 1 0 1 0 0 1 0 1
0 0 1 0 0
1 0 0 1 0 0 0 0 1 0
0 0
0.99999 0 0.00089 1 0.00019 0 0.00001 0.00065 0.98051 0.01209
0.00003 0
#48
0 0 1 0 1 0 0 1 0 1
0 0 1 0 0
1 0 0 1 0 0 0 0 1 0
0 0
0.99999 0 0.00089 1 0.00019 0 0.00001 0.00065 0.98057 0.01215
0.00003 0
#49
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0
0.00462 0.01314 0.00928 0.005 0.00954 0.01759 0.00066 0.01194 0.00962 0.00407
0.00841 0.00862
#50
0 1 0 0 1 0 0 0 1 0
0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0
0.01184 0.01081 0.00004 0.00425 0.00046 0.0039 0.00004 0.00002 0.0001 0.00116
0.00001 0.00016
#51
0 1 0 0 1 0 0 0 1 0
0 0 0 0 0
1 0 0 0 0 0 0 0 0 0
0 0
0.98823 0.00106 0.00008 0.01862 0.00011 0.00003 0.00002 0.00002 0.00007 0.0035
0 0
#52
0 1 0 0 1 0 0 0 1 1
0 0 0 0 0
1 0 0 0 0 0 0 0 0 0
0 0
0.98913 0.00101 0.00008 0.02161 0.00011 0.00003 0.00002 0.00002 0.00007 0.0037
0 0
#53
0 1 0 0 1 0 0 0 1 1
0 0 0 0 0
1 0 0 1 0 0 0 0 0 0
0 0
0.99975 0.00003 0.022 0.96582 0 0 0.00007 0.00075 0.00078 0.03803
0 0
#54
0 1 0 0 1 0 0 0 1 1
0 0 1 0 0
1 0 0 1 0 0 0 0 0 0
0 0
0.99976 0.00003 0.02179 0.96696 0 0 0.00006 0.00076 0.00079 0.03918
0 0
#55
0 1 0 0 1 0 0 0 1 1
0 0 1 0 0
1 0 0 1 0 0 0 0 0 1
0 0
0.99999 0 0.01876 0.9999 0.00004 0 0 0.02141 0.02785 0.94336
0 0
#56
0 1 0 0 1 0 0 0 1 1
0 0 1 0 0
1 0 0 1 0 0 0 0 0 1
0 0
```

```
0.99999 0 0.01877 0.9999 0.00004 0 0 0.02145 0.02792 0.94346
0 0
#57
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0
0.00462 0.01314 0.00928 0.005 0.00954 0.01759 0.00066 0.01194 0.00962 0.00407
0.00841 0.00862
#58
1 0 0 0 1 0 0 1 0 0
0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0
0.01184 0.01081 0.00004 0.00425 0.00046 0.0039 0.00004 0.00002 0.0001 0.00116
0.00001 0.00016
#59
1 0 0 0 1 0 0 1 0 0
0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 0
0.02391 0.9814 0.00028 0.00037 0.02522 0.00645 0.00006 0 0.00012 0.00003
0.00007 0.00001
#60
1 0 0 0 1 0 0 1 0 0
0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 0
0.02585 0.97984 0.00029 0.0004 0.02405 0.00566 0.00006 0 0.00012 0.00003
0.00007 0.00001
#61
1 0 0 0 1 0 0 1 0 0
1 0 0 0 0
0 1 0 0 1 0 0 0 0 0
0 0
0.00768 0.99856 0 0.00064 0.96046 0.02067 0.00001 0 0.00031 0.0002
0.04763 0.00005
#62
1 0 0 0 1 0 0 1 0 0
1 0 0 1 0
0 1 0 0 1 0 0 0 0 0
0 0
0.00774 0.99856 0 0.00064 0.95943 0.02056 0.00001 0 0.00031 0.0002
0.04639 0.00005
#63
1 0 0 0 1 0 0 1 0 0
1 0 0 1 0
0 1 0 0 1 0 0 0 0 0
1 0
0.00239 0.99749 0 0.00757 0.99988 0.00921 0 0 0.00496 0.00032
0.94229 0.00007
#64
1 0 0 0 1 0 0 1 0 0
1 0 0 1 0
0 1 0 0 1 0 0 0 0 0
1 0
0.00239 0.99749 0 0.00757 0.99987 0.00921 0 0 0.00496 0.00032
0.94218 0.00007
#65
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0
0.00462 0.01314 0.00928 0.005 0.00954 0.01759 0.00066 0.01194 0.00962 0.00407
0.00841 0.00862
#66
1 0 0 0 0 1 1 0 0 0
0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0
0.01184 0.01081 0.00004 0.00425 0.00046 0.0039 0.00004 0.00002 0.0001 0.00116
0.00001 0.00016
#67
1 0 0 0 0 1 1 0 0 0
0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 0
0.01731 0.97926 0.00061 0.00002 0.02117 0.00455 0.00008 0.00001 0.00001 0.00012
0 0.00001
#68
```

```
1 0 0 0 0 1 1 0 0 0
1 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 0
0.01896 0.97934 0.00062 0.00002 0.0213 0.00412 0.00008 0.00001 0.00001 0.00013
0 0.00001
#69
1 0 0 0 0 1 1 0 0 0
1 0 0 0 0
0 1 0 0 1 0 0 0 0 0
0 0
0.00295 0.99396 0 0.00004 0.96657 0.0147 0.00001 0 0.00001 0.00155
0.00237 0.0002
#70
1 0 0 0 0 1 1 0 0 0
1 0 0 1 0
0 1 0 0 1 0 0 0 0 0
0 0
0.00296 0.99396 0 0.00004 0.96663 0.01461 0.00001 0 0.00001 0.00155
0.00237 0.0002
#71
1 0 0 0 0 1 1 0 0 0
1 0 0 1 0
0 1 0 0 1 0 0 0 0 0
0 0
0.00378 0.98939 0 0.00053 0.99793 0.00173 0 0 0.00024 0.00095
0.03141 0.00004
#72
1 0 0 0 0 1 1 0 0 0
1 0 0 1 0
0 1 0 0 1 0 0 0 0 0
0 0
0.00378 0.98939 0 0.00053 0.99793 0.00173 0 0 0.00024 0.00095
0.0314 0.00004
#73
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0
0.00462 0.01314 0.00928 0.005 0.00954 0.01759 0.00066 0.01194 0.00962 0.00407
0.00841 0.00862
#74
1 0 0 0 0 1 0 1 0 0
0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0
0.01184 0.01081 0.00004 0.00425 0.00046 0.0039 0.00004 0.00002 0.0001 0.00116
0.00001 0.00016
#75
1 0 0 0 0 1 0 1 0 0
0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 0
0.01082 0.97872 0.00591 0.00001 0.00093 0.0361 0.00134 0.00003 0.00001 0.00002
0 0.00004
#76
1 0 0 0 0 1 0 1 0 0
1 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 0
0.01138 0.97798 0.00573 0.00001 0.00096 0.02971 0.00117 0.00003 0.00001 0.00002
0 0.00004
#77
1 0 0 0 0 1 0 1 0 0
1 0 0 0 0
0 1 0 0 0 1 0 0 0 0
0 0
0.0009 0.99554 0.00157 0 0.00619 0.95662 0.00625 0.00001 0 0.00004
0.00014 0.01632
#78
1 0 0 0 0 1 0 1 0 0
1 0 0 0 1
0 1 0 0 0 1 0 0 0 0
0 0
0.00092 0.99557 0.00153 0 0.00639 0.95397 0.00586 0.00001 0 0.00004
0.00014 0.01541
#79
1 0 0 0 0 1 0 1 0 0
1 0 0 0 1
0 1 0 0 0 1 0 0 0 0
```

```
0 0
0.00034 0.98272 0.001 0.00001 0.00821 0.98292 0.03709 0.00001 0.00003 0.00001
0.0009 0.04431
#80
1 0 0 0 0 1 0 1 0 0
1 0 0 0 1
0 1 0 0 0 1 0 0 0 0
0 0
0.00034 0.98278 0.001 0 0.00819 0.98296 0.03708 0.00001 0.00003 0.00001
0.0009 0.0444
#81
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0
0.00462 0.01314 0.00928 0.005 0.00954 0.01759 0.00066 0.01194 0.00962 0.00407
0.00841 0.00862
#82
1 0 0 0 0 1 0 0 1 0
0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0
0.01184 0.01081 0.00004 0.00425 0.00046 0.0039 0.00004 0.00002 0.0001 0.00116
0.00001 0.00016
#83
1 0 0 0 0 1 0 0 1 0
0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 0
0.0088 0.97824 0.00678 0.00001 0.00106 0.0292 0.00036 0.00003 0 0.0001
0 0.00006
#84
1 0 0 0 0 1 0 0 1 0
1 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 0
0.00937 0.97779 0.00683 0.00001 0.00105 0.02552 0.00033 0.00003 0 0.0001
0 0.00006
#85
1 0 0 0 0 1 0 0 1 0
1 0 0 0 0
0 1 0 0 0 1 0 0 0 0
0 0
0.0003 0.9945 0.00068 0 0.02557 0.95978 0.00083 0.00001 0 0.00029
0.00008 0.04617
#86
1 0 0 0 0 1 0 0 1 0
1 0 0 0 1
0 1 0 0 0 1 0 0 0 0
0 0
0.0003 0.99445 0.00066 0 0.02613 0.95714 0.0008 0.00001 0 0.00029
0.00008 0.04358
#87
1 0 0 0 0 1 0 0 1 0
1 0 0 0 1
0 1 0 0 0 1 0 0 0 0
0 1
0.00005 0.99832 0.00223 0 0.00987 0.99992 0.01852 0.00002 0 0.00037
0.00077 0.93479
#88
1 0 0 0 0 1 0 0 1 0
1 0 0 0 1
0 1 0 0 0 1 0 0 0 0
0 1
0.00005 0.99832 0.00223 0 0.00986 0.99992 0.01852 0.00002 0 0.00037
0.00077 0.9347
#89
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0
0.00462 0.01314 0.00928 0.005 0.00954 0.01759 0.00066 0.01194 0.00962 0.00407
0.00841 0.00862
```

```
SNNS network definition file V1.4-3D
generated at Wed May 11 14:11:32 1994

network name : trans
source files :
no. of units : 30
no. of connections : 243
no. of unit types : 2
no. of site types : 0


learning function : BPTT
update function   : BPTT_Order


type definition section :

name           | act func     | out func     |sites
---------------|--------------|--------------|-------
outType        | Act_Logistic | Out_Identity |
LongeroutType | Act_Logistic | Out_Identity |
---------------|--------------|--------------|-------


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 |                | g1_min   |  0.00000 | -0.02071 | i  |  1,11, 0 |||
  2 |                | g1_norm  |  0.00000 |  0.06810 | i  |  2,11, 0 |||
  3 |                | g1_max   |  0.00000 | -0.02933 | i  |  3,11, 0 |||
  4 |                | g2_min   |  0.00000 | -0.01068 | i  |  4,11, 0 |||
  5 |                | g2_norm  |  0.00000 | -0.03626 | i  |  5,11, 0 |||
  6 |                | g2_max   |  0.00000 |  0.07729 | i  |  6,11, 0 |||
  7 |                | g3_min   |  0.00000 | -0.09688 | i  |  7,11, 0 |||
  8 |                | g3_norm  |  0.00000 |  0.01682 | i  |  8,11, 0 |||
  9 |                | g3_max   |  0.00000 | -0.06813 | i  |  9,11, 0 |||
 10 |                | descarga |  0.00462 | -5.37349 | d  | 11, 3, 0 |||
 11 |                | termica  |  0.01314 | -4.31856 | d  | 12, 3, 0 |||
 12 |                | parcial  |  0.00928 | -4.67067 | d  | 13, 3, 0 |||
 13 |                | arco     |  0.00500 | -5.29382 | d  | 14, 3, 0 |||
 14 |                | t_alta   |  0.00954 | -4.64278 | d  | 15, 3, 0 |||
 15 |                | t_baixa  |  0.01759 | -4.02247 | d  | 16, 3, 0 |||
 16 |                | p_alta   |  0.00066 | -7.32891 | o  |  3, 1, 0 |||
 17 |                | p_baixa  |  0.01194 | -4.41551 | o  |  4, 1, 0 |||
 18 |                | a_alta   |  0.00962 | -4.63429 | o  |  5, 1, 0 |||
 19 |                | a_baixa  |  0.00407 | -5.49886 | o  |  6, 1, 0 |||
 20 |                | t_malta  |  0.00841 | -4.76984 | o  |  7, 1, 0 |||
 21 |                | t_mbaixa |  0.00862 | -4.74458 | o  |  8, 1, 0 |||
 22 |                | hidden   |  0.71441 |  0.91692 | h  |  1, 7, 0 |||
 23 |                | hidden   |  0.27309 | -0.97899 | h  |  2, 7, 0 |||
 24 |                | hidden   |  0.08796 | -2.33875 | h  |  3, 7, 0 |||
 25 |                | hidden   |  0.04029 | -3.17058 | h  |  4, 7, 0 |||
 26 |                | hidden   |  0.76233 |  1.16549 | h  |  5, 7, 0 |||
 27 |                | hidden   |  0.78496 |  1.29483 | h  |  6, 7, 0 |||
 28 |                | hidden   |  0.27827 | -0.95308 | h  |  7, 7, 0 |||
 29 |                | hidden   |  0.79137 |  1.33319 | h  |  8, 7, 0 |||
 30 |                | hidden   |  0.89609 |  2.15453 | h  |  9, 7, 0 |||
----|----------------|----------|----------|----------|----|----------|----------|-----
-----|-------


connection definition section :

target | site | source:weight
-------|------|-------------------------------------------------------------------------
-----------------------------------------
    10 |      | 30: 1.42808, 29:-3.48931, 28: 8.74071, 27:-0.99198, 26: 0.45575, 25:
2.20118, 24: 3.06294, 23: 3.65907, 22:-1.29836
```

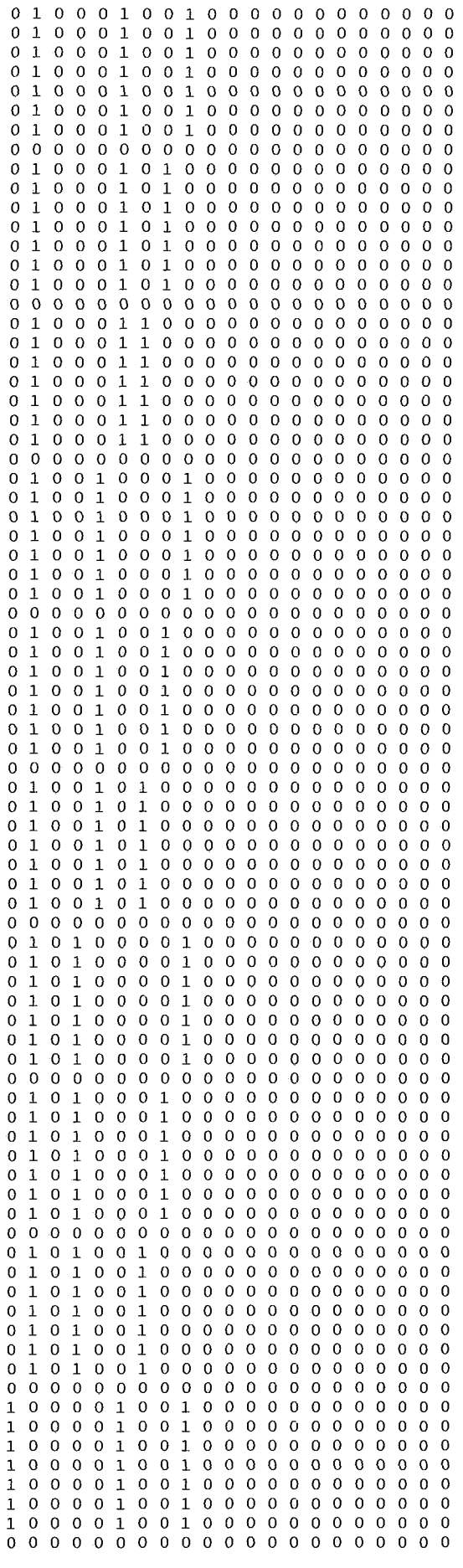
```
    11 |      | 30:-3.26839, 29: 7.08536, 28:-7.87624, 27:-5.02877, 26: 0.66437, 25:
2.06600, 24:-1.48632, 23: 0.68117, 22: 3.66394
    12 |      | 30: 1.79572, 29:-3.73916, 28: 2.54469, 27:-5.28918, 26:-4.18211, 25:-
5.31037, 24: 6.93359, 23: 2.63371, 22: 2.07860
    13 |      | 30: 0.30475, 29:-5.47287, 28: 0.19862, 27: 7.01558, 26: 2.49288, 25:
4.18176, 24: 6.17540, 23: 1.18382, 22:-6.44205
    14 |      | 30:-2.90144, 29: 1.46715, 28:-4.86938, 27:-2.70570, 26: 4.78179, 25:
8.01497, 24:-2.96219, 23:-3.85128, 22:-1.09413
    15 |      | 30:-5.11072, 29: 7.65412, 28:-2.34838, 27:-1.86691, 26:-5.00249, 25:-
3.78115, 24:-2.95943, 23:-4.08317, 22: 6.24689
    16 |      | 30: 1.48955, 29: 4.06021, 28: 2.22616, 27:-2.66694, 26:-7.30459, 25:-
2.92342, 24: 2.81781, 23: 0.69067, 22:-0.77016
    17 |      | 30: 1.73125, 29:-7.94633, 28: 1.83709, 27:-1.24391, 26:-3.56000, 25:-
2.79963, 24: 2.51309, 23: 1.81608, 22: 1.30738
    18 |      | 30: 0.96044, 29:-3.59969, 28: 0.43985, 27: 2.44746, 26: 1.62241, 25:
2.54191, 24: 1.31446, 23: 0.91634, 22:-8.88117
    19 |      | 30:-1.36657, 29:-7.62487, 28:-1.05995, 27: 2.74282, 26: 0.56891, 25:
3.78760, 24: 0.48533, 23:-1.02439, 22: 5.31189
    20 |      | 30:-6.26630, 29: 0.01244, 28:-1.02004, 27:-0.61846, 26: 2.55814, 25:
5.88009, 24:-1.21599, 23:-5.02471, 22:-1.93972
    21 |      | 30:-4.10231, 29: 1.57190, 28:-0.77837, 27:-0.63540, 26:-5.01000, 25:-
1.78275, 24:-0.94210, 23:-5.12535, 22: 6.30075
    22 |      |  9: 5.94561,  8:-1.90848,  7:-0.72882,  6: 1.04554,  5: 0.17069,  4:
2.23948,  3: 1.16043,  2: 1.26153,  1: 0.87138,
              15:-1.56439, 14:-1.47032, 13:-4.45900, 12: 2.15895, 11:-0.31729, 10:-
0.38552
    23 |      |  9: 0.63495,  8: 2.44445,  7: 0.20638,  6: 1.01982,  5: 1.37010,  4:
0.94624,  3: 1.10379,  2: 1.08654,  1: 1.00391,
              15:-1.19088, 14:-3.27153, 13: 0.93066, 12: 0.52691, 11:-3.03542, 10:
2.34262
    24 |      |  9: 0.59202,  8: 0.05368,  7:-0.09829,  6:-0.12391,  5: 0.30366,  4:
0.28080,  3: 0.21541,  2:-0.13660,  1: 0.48665,
              15:-1.15688, 14:-1.13295, 13: 2.40948, 12: 2.12822, 11:-2.50384, 10:
4.42857
    25 |      |  9:-0.75861,  8:-1.32091,  7: 2.79784,  6:-0.16260,  5: 3.22552,  4:-
2.20408,  3: 0.76022,  2: 0.71715,  1:-0.51594,
              15:-4.64724, 14: 6.56003, 13: 4.28517, 12:-5.25473, 11: 2.39476, 10:
0.12447
    26 |      |  9: 0.58727,  8:-0.58309,  7: 2.69110,  6: 0.25428,  5: 3.32289,  4:-
0.87437,  3: 0.45596,  2: 2.30285,  1:-0.02211,
              15:-7.16347, 14: 3.92278, 13: 2.17916, 12:-5.22355, 11:-1.68126, 10:-
0.98829
    27 |      |  9:-0.22440,  8:-1.36079,  7:-1.81410,  6:-0.97849,  5: 0.67623,  4:-
2.99401,  3:-0.38783,  2:-0.30820,  1:-2.81819,
              15:-0.85116, 14:-0.66284, 13: 5.01561, 12:-2.58799, 11:-1.89169, 10:
1.18344
    28 |      |  9: 0.43561,  8:-0.27374,  7: 2.59164,  6:-2.13568,  5: 1.09557,  4:
4.01289,  3: 2.39704,  2: 2.73318,  1:-2.47069,
              15:-0.54309, 14:-1.11108, 13: 0.63996, 12: 0.55711, 11:-2.94868, 10:
4.36887
    29 |      |  9: 0.15500,  8: 1.44741,  7: 1.46524,  6: 1.42111,  5: 1.87876,  4:-
0.20713,  3:-0.70053,  2:-0.98619,  1: 4.92881,
              15: 0.51522, 14: 0.88002, 13:-2.85838, 12:-2.47888, 11: 2.30033, 10:-
1.51715
    30 |      |  9: 0.45561,  8:-1.87454,  7: 3.08741,  6: 1.18547,  5:-0.58761,  4:
0.87375,  3: 0.92517,  2: 1.01549,  1:-0.46045,
              15:-4.46197, 14: 0.00274, 13: 0.48102, 12: 0.34162, 11:-3.07592, 10:
1.80794
-------|------|-------------------------------------------------------------------
-------------------------------------------
```

```
#
Type: SNNSBATCH_1
#
# This is the style for an snnsbat configuration file, type 1.
#
# The following keyword-value combinations may be supplied in any order.
# If a key is given twice, the second appearence is taken.
# Keys that are not required for a special run may be omitted.
# If a key is ommited but required, a default value is assumed.
# The lines may be separated with comments.
#
# Please note the mandatory file type specification at the begining and
# the colon following the key.
#
#
#                                              Defaults:
#
#     NoOfLearnParam:       <int>                0
#     LearnParam:           [ <float> ]*         0.0
#     NoOfInitParam:        <int>                0
#     InitParam:            [ <float> ]*         0.0
#     NetworkFile:          <string>             ""
#     TrainedNetworkFile:   <string>             ""
#     LearnPatternFile:     <string>             ""
#     TestPatternFile:      <string>             ""
#     ResultFile:           <string>             ""
#     InitFunction:         <string>             ""
#     MaxLearnCycles:       <int>                0
#     MaxErrorToStop:       <float>              0.0
#     Shuffle:              [ YES | NO ]         NO
#     ResultMinMaxPattern:  <int><int>           0 0
#     ResultIncludeInput:   [ YES | NO ]         NO
#     ResultIncludeOutput:  [ YES | NO ]         YES
#
#########################################################################
NetworkFile: trans.xnet
#
#
TestPatternFile: trans.xpat
ResultFile: trans.xres
ResultIncludeInput: YES
ResultIncludeOutput: NO
```

```
SNNS network definition file V1.4-3D
generated at Wed May 11 14:11:32 1994

network name : trans
source files :
no. of units : 30
no. of connections : 243
no. of unit types : 2
no. of site types : 0


learning function : BPTT
update function   : BPTT_Order


type definition section :

name           | act func     | out func     |sites
---------------|--------------|--------------|-------
outType        | Act_Logistic | Out_Identity |
LongeroutType | Act_Logistic | Out_Identity |
---------------|--------------|--------------|-------


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 |               | g1_min   |  0.00000 | -0.02071 | i  |  1,11, 0 |||
  2 |               | g1_norm  |  0.00000 |  0.06810 | i  |  2,11, 0 |||
  3 |               | g1_max   |  0.00000 | -0.02933 | i  |  3,11, 0 |||
  4 |               | g2_min   |  0.00000 | -0.01068 | i  |  4,11, 0 |||
  5 |               | g2_norm  |  0.00000 | -0.03626 | i  |  5,11, 0 |||
  6 |               | g2_max   |  0.00000 |  0.07729 | i  |  6,11, 0 |||
  7 |               | g3_min   |  0.00000 | -0.09688 | i  |  7,11, 0 |||
  8 |               | g3_norm  |  0.00000 |  0.01682 | i  |  8,11, 0 |||
  9 |               | g3_max   |  0.00000 | -0.06813 | i  |  9,11, 0 |||
 10 |               | descarga |  0.00462 | -5.37349 | o  | 11, 3, 0 |||
 11 |               | termica  |  0.01314 | -4.31856 | o  | 12, 3, 0 |||
 12 |               | parcial  |  0.00928 | -4.67067 | o  | 13, 3, 0 |||
 13 |               | arco     |  0.00500 | -5.29382 | o  | 14, 3, 0 |||
 14 |               | t_alta   |  0.00954 | -4.64278 | o  | 15, 3, 0 |||
 15 |               | t_baixa  |  0.01759 | -4.02247 | o  | 16, 3, 0 |||
 16 |               | p_alta   |  0.00066 | -7.32891 | o  |  3, 1, 0 |||
 17 |               | p_baixa  |  0.01194 | -4.41551 | o  |  4, 1, 0 |||
 18 |               | a_alta   |  0.00962 | -4.63429 | o  |  5, 1, 0 |||
 19 |               | a_baixa  |  0.00407 | -5.49886 | o  |  6, 1, 0 |||
 20 |               | t_malta  |  0.00841 | -4.76984 | o  |  7, 1, 0 |||
 21 |               | t_mbaixa |  0.00862 | -4.74458 | o  |  8, 1, 0 |||
 22 |               | hidden   |  0.71441 |  0.91692 | h  |  1, 7, 0 |||
 23 |               | hidden   |  0.27309 | -0.97899 | h  |  2, 7, 0 |||
 24 |               | hidden   |  0.08796 | -2.33875 | h  |  3, 7, 0 |||
 25 |               | hidden   |  0.04029 | -3.17058 | h  |  4, 7, 0 |||
 26 |               | hidden   |  0.76233 |  1.16549 | h  |  5, 7, 0 |||
 27 |               | hidden   |  0.78496 |  1.29483 | h  |  6, 7, 0 |||
 28 |               | hidden   |  0.27827 | -0.95308 | h  |  7, 7, 0 |||
 29 |               | hidden   |  0.79137 |  1.33319 | h  |  8, 7, 0 |||
 30 |               | hidden   |  0.89609 |  2.15453 | h  |  9, 7, 0 |||
----|---------------|----------|----------|----------|----|----------|----------|-----
-----|-------


connection definition section :

target | site | source:weight
-------|------|-------------------------------------------------------------------------
-----------------------------------------
    10 |      | 30: 1.42808, 29:-3.48931, 28: 8.74071, 27:-0.99198, 26: 0.45575, 25:
2.20118, 24: 3.06294, 23: 3.65907, 22:-1.29836
```

```
    11 |       | 30:-3.26839, 29: 7.08536, 28:-7.87624, 27:-5.02877, 26: 0.66437, 25:
2.06600, 24:-1.48632, 23: 0.68117, 22: 3.66394
    12 |       | 30: 1.79572, 29:-3.73916, 28: 2.54469, 27:-5.28918, 26:-4.18211, 25:-
5.31037, 24: 6.93359, 23: 2.63371, 22: 2.07860
    13 |       | 30: 0.30475, 29:-5.47287, 28: 0.19862, 27: 7.01558, 26: 2.49288, 25:
4.18176, 24: 6.17540, 23: 1.18382, 22:-6.44205
    14 |       | 30:-2.90144, 29: 1.46715, 28:-4.86938, 27:-2.70570, 26: 4.78179, 25:
8.01497, 24:-2.96219, 23:-3.85128, 22:-1.09413
    15 |       | 30:-5.11072, 29: 7.65412, 28:-2.34838, 27:-1.86691, 26:-5.00249, 25:-
3.78115, 24:-2.95943, 23:-4.08317, 22: 6.24689
    16 |       | 30: 1.48955, 29: 4.06021, 28: 2.22616, 27:-2.66694, 26:-7.30459, 25:-
2.92342, 24: 2.81781, 23: 0.69067, 22:-0.77016
    17 |       | 30: 1.73125, 29:-7.94633, 28: 1.83709, 27:-1.24391, 26:-3.56000, 25:-
2.79963, 24: 2.51309, 23: 1.81608, 22: 1.30738
    18 |       | 30: 0.96044, 29:-3.59969, 28: 0.43985, 27: 2.44746, 26: 1.62241, 25:
2.54191, 24: 1.31446, 23: 0.91634, 22:-8.88117
    19 |       | 30:-1.36657, 29:-7.62487, 28:-1.05995, 27: 2.74282, 26: 0.56891, 25:
3.78760, 24: 0.48533, 23:-1.02439, 22: 5.31189
    20 |       | 30:-6.26630, 29: 0.01244, 28:-1.02004, 27:-0.61846, 26: 2.55814, 25:
5.88009, 24:-1.21599, 23:-5.02471, 22:-1.93972
    21 |       | 30:-4.10231, 29: 1.57190, 28:-0.77837, 27:-0.63540, 26:-5.01000, 25:-
1.78275, 24:-0.94210, 23:-5.12535, 22: 6.30075
    22 |       |  9: 5.94561,  8:-1.90848,  7:-0.72882,  6: 1.04554,  5: 0.17069,  4:
2.23948,  3: 1.16043,  2: 1.26153,  1: 0.87138,
                 15:-1.56439, 14:-1.47032, 13:-4.45900, 12: 2.15895, 11:-0.31729, 10:-
0.38552
    23 |       |  9: 0.63495,  8: 2.44445,  7: 0.20638,  6: 1.01982,  5: 1.37010,  4:
0.94624,  3: 1.10379,  2: 1.08654,  1: 1.00391,
                 15:-1.19088, 14:-3.27153, 13: 0.93066, 12: 0.52691, 11:-3.03542, 10:
2.34262
    24 |       |  9: 0.59202,  8: 0.05368,  7:-0.09829,  6:-0.12391,  5: 0.30366,  4:
0.28080,  3: 0.21541,  2:-0.13660,  1: 0.48665,
                 15:-1.15688, 14:-1.13295, 13: 2.40948, 12: 2.12822, 11:-2.50384, 10:
4.42857
    25 |       |  9:-0.75861,  8:-1.32091,  7: 2.79784,  6:-0.16260,  5: 3.22552,  4:-
2.20408,  3: 0.76022,  2: 0.71715,  1:-0.51594,
                 15:-4.64724, 14: 6.56003, 13: 4.28517, 12:-5.25473, 11: 2.39476, 10:
0.12447
    26 |       |  9: 0.58727,  8:-0.58309,  7: 2.69110,  6: 0.25428,  5: 3.32289,  4:-
0.87437,  3: 0.45596,  2: 2.30285,  1:-0.02211,
                 15:-7.16347, 14: 3.92278, 13: 2.17916, 12:-5.22355, 11:-1.68126, 10:-
0.98829
    27 |       |  9:-0.22440,  8:-1.36079,  7:-1.81410,  6:-0.97849,  5: 0.67623,  4:-
2.99401,  3:-0.38783,  2:-0.30820,  1:-2.81819,
                 15:-0.85116, 14:-0.66284, 13: 5.01561, 12:-2.58799, 11:-1.89169, 10:
1.18344
    28 |       |  9: 0.43561,  8:-0.27374,  7: 2.59164,  6:-2.13568,  5: 1.09557,  4:
4.01289,  3: 2.39704,  2: 2.73318,  1:-2.47069,
                 15:-0.54309, 14:-1.11108, 13: 0.63996, 12: 0.55711, 11:-2.94868, 10:
4.36887
    29 |       |  9: 0.15500,  8: 1.44741,  7: 1.46524,  6: 1.42111,  5: 1.87876,  4:-
0.20713,  3:-0.70053,  2:-0.98619,  1: 4.92881,
                 15: 0.51522, 14: 0.88002, 13:-2.85838, 12:-2.47888, 11: 2.30033, 10:-
1.51715
    30 |       |  9: 0.45561,  8:-1.87454,  7: 3.08741,  6: 1.18547,  5:-0.58761,  4:
0.87375,  3: 0.92517,  2: 1.01549,  1:-0.46045,
                 15:-4.46197, 14: 0.00274, 13: 0.48102, 12: 0.34162, 11:-3.07592, 10:
1.80794
-------|------|-------------------------------------------------------------------
-------------------------------------------
```

```
SNNS pattern definition file V1.4
generated at Tue Dec  3 20:36:50 1991

No. of patterns     : 217
No. of input units  :   9
No. of output units : 12

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```



1	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

```
SNNS 3D-Kernel V3.0002 Batchlearning Program
Configuration file: 'trans.xcfb'
Log file          : 'trans.xlog'



######## SNNS batch execution run. Loop 1 ########
Networkfile 'trans.xnet' loaded.
No. of patterns: 0

No. of cycles: 0

Max. network error to stop: 0.000000

Patterns are in order

Network name             : trans
No. of units             : 30
No. of sites             : 0
No. of links             : 243

Learning Function        : BPTT


Test pattern file : 'trans.xpat'

No Initialization


Result File              : 'trans.xres'
Result File Start Pattern: 0
Result File End Pattern  : 0
Result File Input Pattern included


*******************************************************

SNNS 3D-Kernel V3.0002 Result file computation started at Wed May 11 14:20:18 1994



Test Pattern File 'trans.xpat' loaded.
No. of test patterns: 217


Result file saved.

Trained Network was not saved

SNNS 3D-Kernel V3.0002 Batchlearning terminated at Wed May 11 14:20:18 1994
System: ULTRIX  Node: amazonas  Machine: RISC
*******************************************************

----   STATISTICS  ----
No. of learned cycles: 0
No. of units updated : 0
No. of sites updated : 0
No. of links updated : 0
CPU Time used: 0.20 seconds
User time: 0 seconds
No. of connection updates per second (CUPS): 0.000000e+00
```

```
SNNS result file V1.4-3D
generated at Wed May 11 14:20:18 1994

No. of patterns     : 217
No. of input units  : 9
No. of output units : 12
startpattern        : 1
endpattern          : 217
input patterns included
#1
0 0 0 0 0 0 0 0 0
0.00462 0.01314 0.00928 0.005 0.00954 0.01759 0.00066 0.01194 0.00962 0.00407
0.00841 0.00862
#2
0 0 1 0 0 1 0 0 1
0.01184 0.01081 0.00004 0.00425 0.00046 0.0039 0.00004 0.00002 0.0001 0.00116
0.00001 0.00016
#3
0 0 1 0 0 1 0 0 1
0.27011 0.12302 0.00259 0.00013 0.00007 0.00178 0.00015 0.00006 0.00001 0.00038
0 0.00002
#4
0 0 1 0 0 1 0 0 1
0.30144 0.10672 0.00256 0.00015 0.00006 0.00152 0.00014 0.00006 0.00001 0.00039
0 0.00002
#5
0 0 1 0 0 1 0 0 1
0.81477 0.01708 0.01405 0.00048 0.00001 0.00056 0.0004 0.00017 0.00001 0.00039
0 0.00001
#6
0 0 1 0 0 1 0 0 1
0.88325 0.01011 0.02019 0.00073 0.00001 0.00039 0.00046 0.00022 0.00001 0.00042
0 0.00001
#7
0 0 1 0 0 1 0 0 1
0.99877 0.00007 0.62239 0.12815 0 0.00001 0.00194 0.00516 0.00006 0.00183
0 0
#8
0 0 1 0 0 1 0 0 1
0.99906 0.00004 0.69649 0.1871 0 0.00001 0.00205 0.00736 0.00007 0.00234
0 0
#9
0 0 0 0 0 0 0 0 0
0.00462 0.01314 0.00928 0.005 0.00954 0.01759 0.00066 0.01194 0.00962 0.00407
0.00841 0.00862
#10
0 0 1 0 0 1 0 1 0
0.01184 0.01081 0.00004 0.00425 0.00046 0.0039 0.00004 0.00002 0.0001 0.00116
0.00001 0.00016
#11
0 0 1 0 0 1 0 1 0
0.11555 0.59403 0.00443 0.00004 0.00016 0.00378 0.00078 0.00004 0.00001 0.00003
0 0.00001
#12
0 0 1 0 0 1 0 1 0
0.12778 0.56483 0.00422 0.00004 0.00016 0.00319 0.00069 0.00004 0.00002 0.00003
0 0.00001
#13
0 0 1 0 0 1 0 1 0
0.01415 0.95331 0.00293 0.00001 0.00088 0.07999 0.00196 0.00003 0.00001 0.00003
0 0.00014
#14
0 0 1 0 0 1 0 1 0
0.0172 0.94276 0.00323 0.00001 0.00072 0.06263 0.00197 0.00003 0.00001 0.00003
0 0.00011
#15
0 0 1 0 0 1 0 1 0
0.00176 0.99189 0.00133 0 0.00547 0.84349 0.00455 0.00002 0 0.00005
0.00005 0.00601
#16
0 0 1 0 0 1 0 1 0
0.00197 0.99161 0.00126 0 0.0057 0.79733 0.00378 0.00001 0 0.00006
0.00005 0.00452
#17
0 0 0 0 0 0 0 0 0
0.00462 0.01314 0.00928 0.005 0.00954 0.01759 0.00066 0.01194 0.00962 0.00407
0.00841 0.00862
#18
0 0 1 0 0 1 1 0 0
0.01184 0.01081 0.00004 0.00425 0.00046 0.0039 0.00004 0.00002 0.0001 0.00116
```

```
0.00001 0.00016
#19
0 0 1 0 0 1 1 0 0
0.97634 0.06779 0.00056 0.00014 0.00303 0.00016 0.00012 0.00001 0.00002 0.0003
0.00001 0
#20
0 0 1 0 0 1 1 0 0
0.97892 0.065 0.00055 0.00016 0.00302 0.00014 0.00011 0.00001 0.00002 0.00032
0.00001 0
#21
0 0 1 0 0 1 1 0 0
0.99972 0.00245 0.08835 0.14164 0.00008 0 0.00069 0.0002 0.00023 0.00097
0 0
#22
0 0 1 0 0 1 1 0 0
0.99973 0.00239 0.08983 0.14483 0.00008 0 0.00069 0.0002 0.00023 0.00098
0 0
#23
0 0 1 0 0 1 1 0 0
0.99984 0.00042 0.09363 0.63238 0.00004 0 0.00042 0.00043 0.00082 0.00235
0 0
#24
0 0 1 0 0 1 1 0 0
0.99984 0.00041 0.09247 0.6446 0.00004 0 0.00041 0.00044 0.00085 0.0024
0 0
#25
0 0 0 0 0 0 0 0 0
0.00462 0.01314 0.00928 0.005 0.00954 0.01759 0.00066 0.01194 0.00962 0.00407
0.00841 0.00862
#26
0 0 1 0 1 0 0 0 1
0.01184 0.01081 0.00004 0.00425 0.00046 0.0039 0.00004 0.00002 0.0001 0.00116
0.00001 0.00016
#27
0 0 1 0 1 0 0 0 1
0.98841 0.00138 0.00011 0.02215 0.00011 0.00003 0.00003 0.00002 0.00007 0.00316
0 0
#28
0 0 1 0 1 0 0 0 1
0.98939 0.0013 0.00011 0.02599 0.00011 0.00003 0.00003 0.00002 0.00007 0.00335
0 0
#29
0 0 1 0 1 0 0 0 1
0.99972 0.00004 0.0196 0.95902 0 0 0.00009 0.0005 0.00066 0.02655
0 0
#30
0 0 1 0 1 0 0 0 1
0.99973 0.00004 0.01931 0.96065 0 0 0.00009 0.00051 0.00068 0.02752
0 0
#31
0 0 1 0 1 0 0 0 1
0.99999 0 0.01645 0.99988 0.00004 0 0 0.01597 0.0248 0.92515
0 0
#32
0 0 1 0 1 0 0 0 1
0.99999 0 0.01647 0.99988 0.00004 0 0 0.01603 0.0249 0.92538
0 0
#33
0 0 0 0 0 0 0 0 0
0.00462 0.01314 0.00928 0.005 0.00954 0.01759 0.00066 0.01194 0.00962 0.00407
0.00841 0.00862
#34
0 0 1 0 1 0 0 1 0
0.01184 0.01081 0.00004 0.00425 0.00046 0.0039 0.00004 0.00002 0.0001 0.00116
0.00001 0.00016
#35
0 0 1 0 1 0 0 1 0
0.98209 0.00499 0.00011 0.01656 0.00033 0.00004 0.00007 0.00001 0.00069 0.0001
0.00001 0
#36
0 0 1 0 1 0 0 1 0
0.98362 0.0044 0.0001 0.01923 0.00032 0.00004 0.00007 0.00001 0.00073 0.00011
0.00001 0
#37
0 0 1 0 1 0 0 1 0
0.99964 0.00004 0.01147 0.98084 0.00001 0 0.00053 0.0001 0.01741 0.0002
0 0
#38
0 0 1 0 1 0 0 1 0
0.99964 0.00004 0.01121 0.9819 0.00001 0 0.00052 0.0001 0.01819 0.00021
```

```
0 0
#39
0 0 1 0 1 0 0 1 0
0.99999 0 0.00089 1 0.00019 0 0.00001 0.00065 0.98051 0.01209
0.00003 0
#40
0 0 1 0 1 0 0 1 0
0.99999 0 0.00089 1 0.00019 0 0.00001 0.00065 0.98057 0.01215
0.00003 0
#41
0 0 0 0 0 0 0 0 0
0.00462 0.01314 0.00928 0.005 0.00954 0.01759 0.00066 0.01194 0.00962 0.00407
0.00841 0.00862
#42
0 0 1 0 1 0 1 0 0
0.01184 0.01081 0.00004 0.00425 0.00046 0.0039 0.00004 0.00002 0.0001 0.00116
0.00001 0.00016
#43
0 0 1 0 1 0 1 0 0
0.99719 0.01007 0.00002 0.01743 0.01786 0.00001 0.00002 0 0.00037 0.00161
0.00005 0
#44
0 0 1 0 1 0 1 0 0
0.99729 0.00952 0.00002 0.01956 0.01655 0.00001 0.00002 0 0.00038 0.00168
0.00005 0
#45
0 0 1 0 1 0 1 0 0
0.99987 0.00036 0.00148 0.9824 0.00044 0 0.00009 0.00003 0.00625 0.00498
0.00001 0
#46
0 0 1 0 1 0 1 0 0
0.99987 0.00035 0.00147 0.98297 0.00044 0 0.00009 0.00003 0.00638 0.005
0.00001 0
#47
0 0 1 0 1 0 1 0 0
0.99999 0 0.00085 1 0.0002 0 0.00001 0.00062 0.97732 0.01311
0.00003 0
#48
0 0 1 0 1 0 1 0 0
0.99999 0 0.00085 1 0.0002 0 0.00001 0.00062 0.97737 0.01314
0.00003 0
#49
0 0 0 0 0 0 0 0 0
0.00462 0.01314 0.00928 0.005 0.00954 0.01759 0.00066 0.01194 0.00962 0.00407
0.00841 0.00862
#50
0 0 1 1 0 0 0 0 1
0.01184 0.01081 0.00004 0.00425 0.00046 0.0039 0.00004 0.00002 0.0001 0.00116
0.00001 0.00016
#51
0 0 1 1 0 0 0 0 1
0.99464 0.00105 0.34645 0.00005 0 0.00024 0.0016 0.00432 0.00001 0.00053
0 0.00002
#52
0 0 1 1 0 0 0 0 1
0.99498 0.00102 0.34224 0.00006 0 0.0002 0.00141 0.00422 0.00001 0.00054
0 0.00002
#53
0 0 1 1 0 0 0 0 1
0.99993 0.00001 0.9999 0.02886 0 0 0.11437 0.93997 0.00005 0.02161
0 0.00004
#54
0 0 1 1 0 0 0 0 1
0.99993 0.00001 0.9999 0.02912 0 0 0.11203 0.93863 0.00005 0.02152
0 0.00004
#55
0 0 1 1 0 0 0 0 1
0.99996 0 0.99998 0.02058 0 0.00001 0.28712 0.98858 0.00005 0.03602
0 0.00009
#56
0 0 1 1 0 0 0 0 1
0.99996 0 0.99998 0.02059 0 0.00001 0.28708 0.98858 0.00005 0.03603
0 0.00009
#57
0 0 0 0 0 0 0 0 0
0.00462 0.01314 0.00928 0.005 0.00954 0.01759 0.00066 0.01194 0.00962 0.00407
0.00841 0.00862
#58
0 0 1 1 0 0 0 1 0
0.01184 0.01081 0.00004 0.00425 0.00046 0.0039 0.00004 0.00002 0.0001 0.00116
```

```
0.00001 0.00016
#59
0 0 1 1 0 0 0 1 0
0.98948 0.00748 0.34395 0.00001 0 0.00425 0.02458 0.00153 0 0.00004
0 0.00006
#60
0 0 1 1 0 0 0 1 0
0.98998 0.00729 0.33117 0.00001 0 0.00349 0.02076 0.00144 0 0.00004
0 0.00005
#61
0 0 1 1 0 0 0 1 0
0.99985 0.00004 0.99987 0.00454 0 0.00005 0.50565 0.77337 0.00002 0.00191
0 0.00008
#62
0 0 1 1 0 0 0 1 0
0.99984 0.00004 0.99986 0.00448 0 0.00005 0.50338 0.75924 0.00002 0.00181
0 0.00008
#63
0 0 1 1 0 0 0 1 0
0.99995 0.00001 0.99998 0.01275 0 0.00001 0.37318 0.97941 0.00004 0.01799
0 0.00009
#64
0 0 1 1 0 0 0 1 0
0.99995 0.00001 0.99998 0.01275 0 0.00001 0.3732 0.97941 0.00004 0.01799
0 0.00009
#65
0 0 0 0 0 0 0 0 0
0.00462 0.01314 0.00928 0.005 0.00954 0.01759 0.00066 0.01194 0.00962 0.00407
0.00841 0.00862
#66
0 0 1 1 0 0 1 0 0
0.01184 0.01081 0.00004 0.00425 0.00046 0.0039 0.00004 0.00002 0.0001 0.00116
0.00001 0.00016
#67
0 0 1 1 0 0 1 0 0
0.98808 0.01026 0.02819 0.00002 0.00005 0.00048 0.00058 0.00018 0.00001 0.00012
0 0.00001
#68
0 0 1 1 0 0 1 0 0
0.98888 0.01032 0.02867 0.00002 0.00005 0.00043 0.00057 0.00018 0.00001 0.00013
0 0.00001
#69
0 0 1 1 0 0 1 0 0
0.99981 0.00038 0.9734 0.01608 0 0 0.00273 0.02079 0.00005 0.00133
0 0
#70
0 0 1 1 0 0 1 0 0
0.99981 0.00038 0.97358 0.01613 0 0 0.00273 0.02094 0.00005 0.00133
0 0
#71
0 0 1 1 0 0 1 0 0
0.99994 0.00001 0.99997 0.01329 0 0.00001 0.27204 0.97322 0.00004 0.01882
0 0.00007
#72
0 0 1 1 0 0 1 0 0
0.99994 0.00001 0.99997 0.01329 0 0.00001 0.27208 0.97324 0.00004 0.01883
0 0.00007
#73
0 0 0 0 0 0 0 0 0
0.00462 0.01314 0.00928 0.005 0.00954 0.01759 0.00066 0.01194 0.00962 0.00407
0.00841 0.00862
#74
0 1 0 0 0 1 0 0 1
0.01184 0.01081 0.00004 0.00425 0.00046 0.0039 0.00004 0.00002 0.0001 0.00116
0.00001 0.00016
#75
0 1 0 0 0 1 0 0 1
0.42869 0.05639 0.00177 0.00016 0.00007 0.00089 0.00008 0.00006 0.00001 0.00047
0 0.00001
#76
0 1 0 0 0 1 0 0 1
0.46545 0.04817 0.00178 0.00019 0.00006 0.00079 0.00008 0.00006 0.00001 0.00048
0 0.00001
#77
0 1 0 0 0 1 0 0 1
0.98492 0.00091 0.03523 0.00442 0 0.00005 0.00025 0.00047 0.00002 0.00081
0 0
#78
0 1 0 0 0 1 0 0 1
0.98961 0.00061 0.04983 0.00699 0 0.00004 0.00028 0.0006 0.00003 0.00092
```

```
0 0
#79
0 1 0 0 0 1 0 0 1
0.99949 0.00001 0.64288 0.47629 0 0 0.00043 0.01126 0.00016 0.00673
0 0
#80
0 1 0 0 0 1 0 0 1
0.99952 0.00001 0.67034 0.48929 0 0 0.00043 0.01284 0.00017 0.00726
0 0
#81
0 0 0 0 0 0 0 0 0
0.00462 0.01314 0.00928 0.005 0.00954 0.01759 0.00066 0.01194 0.00962 0.00407
0.00841 0.00862
#82
0 1 0 0 0 1 0 1 0
0.01184 0.01081 0.00004 0.00425 0.00046 0.0039 0.00004 0.00002 0.0001 0.00116
0.00001 0.00016
#83
0 1 0 0 0 1 0 1 0
0.19691 0.47558 0.002 0.00006 0.00028 0.00129 0.00019 0.00003 0.00002 0.00004
0 0
#84
0 1 0 0 0 1 0 1 0
0.21598 0.4408 0.00201 0.00006 0.00026 0.00117 0.00019 0.00003 0.00002 0.00004
0 0
#85
0 1 0 0 0 1 0 1 0
0.07062 0.8311 0.00137 0.00003 0.00111 0.00485 0.00022 0.00002 0.00001 0.00004
0 0.00001
#86
0 1 0 0 0 1 0 1 0
0.10099 0.76334 0.00165 0.00003 0.00078 0.00363 0.00024 0.00002 0.00002 0.00003
0 0.00001
#87
0 1 0 0 0 1 0 1 0
0.00606 0.98701 0.0003 0.00001 0.01968 0.07594 0.00012 0 0.00001 0.00007
0.00003 0.00011
#88
0 1 0 0 0 1 0 1 0
0.00926 0.98008 0.00041 0.00001 0.01103 0.043 0.00014 0.00001 0.00001 0.00006
0.00001 0.00007
#89
0 0 0 0 0 0 0 0 0
0.00462 0.01314 0.00928 0.005 0.00954 0.01759 0.00066 0.01194 0.00962 0.00407
0.00841 0.00862
#90
0 1 0 0 0 1 1 0 0
0.01184 0.01081 0.00004 0.00425 0.00046 0.0039 0.00004 0.00002 0.0001 0.00116
0.00001 0.00016
#91
0 1 0 0 0 1 1 0 0
0.98073 0.04947 0.00052 0.00013 0.00256 0.00015 0.00011 0.00001 0.00002 0.00032
0.00001 0
#92
0 1 0 0 0 1 1 0 0
0.9826 0.04779 0.00051 0.00014 0.00256 0.00014 0.0001 0.00001 0.00002 0.00034
0.00001 0
#93
0 1 0 0 0 1 1 0 0
0.99971 0.0018 0.07111 0.1463 0.00007 0 0.00045 0.00023 0.00024 0.00131
0 0
#94
0 1 0 0 0 1 1 0 0
0.99971 0.00177 0.07208 0.14895 0.00007 0 0.00045 0.00023 0.00024 0.00132
0 0
#95
0 1 0 0 0 1 1 0 0
0.99985 0.00028 0.06431 0.72353 0.00004 0 0.00023 0.00048 0.00105 0.00399
0 0
#96
0 1 0 0 0 1 1 0 0
0.99985 0.00028 0.06356 0.73314 0.00004 0 0.00022 0.00048 0.00108 0.00407
0 0
#97
0 0 0 0 0 0 0 0 0
0.00462 0.01314 0.00928 0.005 0.00954 0.01759 0.00066 0.01194 0.00962 0.00407
0.00841 0.00862
#98
0 1 0 0 1 0 0 0 1
0.01184 0.01081 0.00004 0.00425 0.00046 0.0039 0.00004 0.00002 0.0001 0.00116
```

```
0.00001 0.00016
#99
0 1 0 0 1 0 0 0 1
0.98823 0.00106 0.00008 0.01862 0.00011 0.00003 0.00002 0.00002 0.00007 0.0035
0 0
#100
0 1 0 0 1 0 0 0 1
0.98913 0.00101 0.00008 0.02161 0.00011 0.00003 0.00002 0.00002 0.00007 0.0037
0 0
#101
0 1 0 0 1 0 0 0 1
0.99975 0.00003 0.022 0.96582 0 0 0.00007 0.00075 0.00078 0.03803
0 0
#102
0 1 0 0 1 0 0 0 1
0.99976 0.00003 0.02179 0.96696 0 0 0.00006 0.00076 0.00079 0.03918
0 0
#103
0 1 0 0 1 0 0 0 1
0.99999 0 0.01876 0.9999 0.00004 0 0 0.02141 0.02785 0.94336
0 0
#104
0 1 0 0 1 0 0 0 1
0.99999 0 0.01877 0.9999 0.00004 0 0 0.02145 0.02792 0.94346
0 0
#105
0 0 0 0 0 0 0 0 0
0.00462 0.01314 0.00928 0.005 0.00954 0.01759 0.00066 0.01194 0.00962 0.00407
0.00841 0.00862
#106
0 1 0 0 1 0 0 1 0
0.01184 0.01081 0.00004 0.00425 0.00046 0.0039 0.00004 0.00002 0.0001 0.00116
0.00001 0.00016
#107
0 1 0 0 1 0 0 1 0
0.98358 0.00378 0.00009 0.01328 0.00027 0.00004 0.00006 0.00001 0.00058 0.00011
0.00001 0
#108
0 1 0 0 1 0 0 1 0
0.98478 0.00337 0.00009 0.01525 0.00026 0.00004 0.00006 0.00001 0.00061 0.00012
0.00001 0
#109
0 1 0 0 1 0 0 1 0
0.99962 0.00003 0.01033 0.97802 0.00001 0 0.00037 0.00011 0.01475 0.00027
0 0
#110
0 1 0 0 1 0 0 1 0
0.99962 0.00003 0.01019 0.97896 0.00001 0 0.00037 0.00011 0.01525 0.00028
0 0
#111
0 1 0 0 1 0 0 1 0
0.99999 0 0.00111 1 0.00017 0 0.00001 0.00107 0.98433 0.01953
0.00003 0
#112
0 1 0 0 1 0 0 1 0
0.99999 0 0.00111 1 0.00017 0 0.00001 0.00107 0.98437 0.01961
0.00003 0
#113
0 0 0 0 0 0 0 0 0
0.00462 0.01314 0.00928 0.005 0.00954 0.01759 0.00066 0.01194 0.00962 0.00407
0.00841 0.00862
#114
0 1 0 0 1 0 1 0 0
0.01184 0.01081 0.00004 0.00425 0.00046 0.0039 0.00004 0.00002 0.0001 0.00116
0.00001 0.00016
#115
0 1 0 0 1 0 1 0 0
0.99684 0.00955 0.00001 0.01515 0.01814 0.00001 0.00002 0 0.00033 0.00188
0.00005 0
#116
0 1 0 0 1 0 1 0 0
0.99694 0.00904 0.00001 0.01693 0.01687 0.00001 0.00002 0 0.00034 0.00196
0.00005 0
#117
0 1 0 0 1 0 1 0 0
0.99986 0.00032 0.00128 0.98058 0.00044 0 0.00007 0.00004 0.00559 0.00675
0.00001 0
#118
0 1 0 0 1 0 1 0 0
0.99986 0.00031 0.00128 0.98113 0.00044 0 0.00007 0.00004 0.00569 0.00678
```

```
0.00001 0
#119
0 1 0 0 1 0 1 0 0
0.99999 0 0.00107 1 0.00018 0 0.00001 0.00104 0.98129 0.02192
0.00003 0
#120
0 1 0 0 1 0 1 0 0
0.99999 0 0.00108 1 0.00018 0 0.00001 0.00105 0.98133 0.02197
0.00003 0
#121
0 0 0 0 0 0 0 0 0
0.00462 0.01314 0.00928 0.005 0.00954 0.01759 0.00066 0.01194 0.00962 0.00407
0.00841 0.00862
#122
0 1 0 1 0 0 0 0 1
0.01184 0.01081 0.00004 0.00425 0.00046 0.0039 0.00004 0.00002 0.0001 0.00116
0.00001 0.00016
#123
0 1 0 1 0 0 0 0 1
0.99538 0.00076 0.17028 0.00009 0.00001 0.00007 0.00026 0.00328 0.00001 0.00098
0 0.00001
#124
0 1 0 1 0 0 0 0 1
0.99563 0.00073 0.17444 0.0001 0.00001 0.00006 0.00025 0.00335 0.00001 0.001
0 0.00001
#125
0 1 0 1 0 0 0 0 1
0.99994 0.00001 0.99778 0.1443 0 0 0.00138 0.58049 0.00015 0.0311
0 0
#126
0 1 0 1 0 0 0 0 1
0.99994 0.00001 0.99785 0.1431 0 0 0.00142 0.58726 0.00015 0.03125
0 0
#127
0 1 0 1 0 0 0 0 1
0.99996 0 0.99998 0.02978 0 0 0.19345 0.98715 0.00006 0.04355
0 0.00006
#128
0 1 0 1 0 0 0 0 1
0.99996 0 0.99998 0.0297 0 0 0.19381 0.98716 0.00006 0.04349
0 0.00006
#129
0 0 0 0 0 0 0 0 0
0.00462 0.01314 0.00928 0.005 0.00954 0.01759 0.00066 0.01194 0.00962 0.00407
0.00841 0.00862
#130
0 1 0 1 0 0 0 1 0
0.01184 0.01081 0.00004 0.00425 0.00046 0.0039 0.00004 0.00002 0.0001 0.00116
0.00001 0.00016
#131
0 1 0 1 0 0 0 1 0
0.99129 0.00756 0.09429 0.00001 0.00001 0.00059 0.00139 0.00052 0.00001 0.00007
0 0.00001
#132
0 1 0 1 0 0 0 1 0
0.99161 0.00731 0.09378 0.00002 0.00001 0.00052 0.0013 0.00052 0.00001 0.00007
0 0.00001
#133
0 1 0 1 0 0 0 1 0
0.99982 0.00008 0.99677 0.01424 0 0 0.01854 0.1659 0.00006 0.00134
0 0
#134
0 1 0 1 0 0 0 1 0
0.99982 0.00008 0.99675 0.0143 0 0 0.01844 0.16545 0.00006 0.00134
0 0
#135
0 1 0 1 0 0 0 1 0
0.99995 0.00001 0.99998 0.01487 0 0.00001 0.32538 0.98223 0.00004 0.02203
0 0.00009
#136
0 1 0 1 0 0 0 1 0
0.99995 0.00001 0.99998 0.01488 0 0.00001 0.32536 0.98223 0.00004 0.02203
0 0.00009
#137
0 0 0 0 0 0 0 0 0
0.00462 0.01314 0.00928 0.005 0.00954 0.01759 0.00066 0.01194 0.00962 0.00407
0.00841 0.00862
#138
0 1 0 1 0 0 1 0 0
0.01184 0.01081 0.00004 0.00425 0.00046 0.0039 0.00004 0.00002 0.0001 0.00116
```

```
0.00001 0.00016
#139
0 1 0 1 0 0 1 0 0
0.9882 0.00844 0.02302 0.00002 0.00006 0.00036 0.00037 0.0002 0.00001 0.00016
0 0.00001
#140
0 1 0 1 0 0 1 0 0
0.98894 0.00847 0.02355 0.00002 0.00006 0.00033 0.00037 0.0002 0.00001 0.00017
0 0.00001
#141
0 1 0 1 0 0 1 0 0
0.99984 0.00025 0.96504 0.02351 0 0 0.00108 0.02528 0.00007 0.00235
0 0
#142
0 1 0 1 0 0 1 0 0
0.99984 0.00025 0.96529 0.02361 0 0 0.00108 0.02545 0.00007 0.00236
0 0
#143
0 1 0 1 0 0 1 0 0
0.99995 0.00001 0.99992 0.02841 0 0 0.04605 0.94486 0.00007 0.02636
0 0.00002
#144
0 1 0 1 0 0 1 0 0
0.99995 0.00001 0.99992 0.02841 0 0 0.04612 0.94494 0.00007 0.02637
0 0.00002
#145
0 0 0 0 0 0 0 0 0
0.00462 0.01314 0.00928 0.005 0.00954 0.01759 0.00066 0.01194 0.00962 0.00407
0.00841 0.00862
#146
1 0 0 0 0 1 0 0 1
0.01184 0.01081 0.00004 0.00425 0.00046 0.0039 0.00004 0.00002 0.0001 0.00116
0.00001 0.00016
#147
1 0 0 0 0 1 0 0 1
0.0088 0.97824 0.00678 0.00001 0.00106 0.0292 0.00036 0.00003 0 0.0001
0 0.00006
#148
1 0 0 0 0 1 0 0 1
0.00937 0.97779 0.00683 0.00001 0.00105 0.02552 0.00033 0.00003 0 0.0001
0 0.00006
#149
1 0 0 0 0 1 0 0 1
0.0003 0.9945 0.00068 0 0.02557 0.95978 0.00083 0.00001 0 0.00029
0.00008 0.04617
#150
1 0 0 0 0 1 0 0 1
0.0003 0.99445 0.00066 0 0.02613 0.95714 0.0008 0.00001 0 0.00029
0.00008 0.04358
#151
1 0 0 0 0 1 0 0 1
0.00005 0.99832 0.00223 0 0.00987 0.99992 0.01852 0.00002 0 0.00037
0.00077 0.93479
#152
1 0 0 0 0 1 0 0 1
0.00005 0.99832 0.00223 0 0.00986 0.99992 0.01852 0.00002 0 0.00037
0.00077 0.9347
#153
0 0 0 0 0 0 0 0 0
0.00462 0.01314 0.00928 0.005 0.00954 0.01759 0.00066 0.01194 0.00962 0.00407
0.00841 0.00862
#154
1 0 0 0 0 1 0 1 0
0.01184 0.01081 0.00004 0.00425 0.00046 0.0039 0.00004 0.00002 0.0001 0.00116
0.00001 0.00016
#155
1 0 0 0 0 1 0 1 0
0.01082 0.97872 0.00591 0.00001 0.00093 0.0361 0.00134 0.00003 0.00001 0.00002
0 0.00004
#156
1 0 0 0 0 1 0 1 0
0.01138 0.97798 0.00573 0.00001 0.00096 0.02971 0.00117 0.00003 0.00001 0.00002
0 0.00004
#157
1 0 0 0 0 1 0 1 0
0.0009 0.99554 0.00157 0 0.00619 0.95662 0.00625 0.00001 0 0.00004
0.00014 0.01632
#158
1 0 0 0 0 1 0 1 0
0.00092 0.99557 0.00153 0 0.00639 0.95397 0.00586 0.00001 0 0.00004
```

0.00014 0.01541
#159
1 0 0 0 0 1 0 1 0
0.00034 0.98272 0.001 0.00001 0.00821 0.98292 0.03709 0.00001 0.00003 0.00001
0.0009 0.04431
#160
1 0 0 0 0 1 0 1 0
0.00034 0.98278 0.001 0 0.00819 0.98296 0.03708 0.00001 0.00003 0.00001
0.0009 0.0444
#161
0 0 0 0 0 0 0 0 0
0.00462 0.01314 0.00928 0.005 0.00954 0.01759 0.00066 0.01194 0.00962 0.00407
0.00841 0.00862
#162
1 0 0 0 0 1 1 0 0
0.01184 0.01081 0.00004 0.00425 0.00046 0.0039 0.00004 0.00002 0.0001 0.00116
0.00001 0.00016
#163
1 0 0 0 0 1 1 0 0
0.01731 0.97926 0.00061 0.00002 0.02117 0.00455 0.00008 0.00001 0.00001 0.00012
0 0.00001
#164
1 0 0 0 0 1 1 0 0
0.01896 0.97934 0.00062 0.00002 0.0213 0.00412 0.00008 0.00001 0.00001 0.00013
0 0.00001
#165
1 0 0 0 0 1 1 0 0
0.00295 0.99396 0 0.00004 0.96657 0.0147 0.00001 0 0.00001 0.00155
0.00237 0.0002
#166
1 0 0 0 0 1 1 0 0
0.00296 0.99396 0 0.00004 0.96663 0.01461 0.00001 0 0.00001 0.00155
0.00237 0.0002
#167
1 0 0 0 0 1 1 0 0
0.00378 0.98939 0 0.00053 0.99793 0.00173 0 0 0.00024 0.00095
0.03141 0.00004
#168
1 0 0 0 0 1 1 0 0
0.00378 0.98939 0 0.00053 0.99793 0.00173 0 0 0.00024 0.00095
0.0314 0.00004
#169
0 0 0 0 0 0 0 0 0
0.00462 0.01314 0.00928 0.005 0.00954 0.01759 0.00066 0.01194 0.00962 0.00407
0.00841 0.00862
#170
1 0 0 0 1 0 0 0 1
0.01184 0.01081 0.00004 0.00425 0.00046 0.0039 0.00004 0.00002 0.0001 0.00116
0.00001 0.00016
#171
1 0 0 0 1 0 0 0 1
0.04172 0.94479 0.00092 0.00012 0.00387 0.00525 0.00006 0.00001 0 0.00044
0 0.00003
#172
1 0 0 0 1 0 0 0 1
0.04705 0.93805 0.00093 0.00014 0.0036 0.0044 0.00006 0.00001 0 0.00046
0 0.00002
#173
1 0 0 0 1 0 0 0 1
0.00158 0.99954 0 0.00002 0.98378 0.39657 0 0 0 0.00724
0.05114 0.00353
#174
1 0 0 0 1 0 0 0 1
0.00164 0.99952 0 0.00003 0.98253 0.38203 0 0 0 0.00709
0.04684 0.00331
#175
1 0 0 0 1 0 0 0 1
0.00061 0.99986 0 0.00003 0.99973 0.70267 0 0 0 0.02927
0.80281 0.02107
#176
1 0 0 0 1 0 0 0 1
0.00061 0.99986 0 0.00003 0.99973 0.69858 0 0 0 0.02925
0.80086 0.02073
#177
0 0 0 0 0 0 0 0 0
0.00462 0.01314 0.00928 0.005 0.00954 0.01759 0.00066 0.01194 0.00962 0.00407
0.00841 0.00862
#178
1 0 0 0 1 0 0 1 0
0.01184 0.01081 0.00004 0.00425 0.00046 0.0039 0.00004 0.00002 0.0001 0.00116

```
0.00001 0.00016
#179
1 0 0 0 1 0 0 1 0
0.02391 0.9814 0.00028 0.00037 0.02522 0.00645 0.00006 0 0.00012 0.00003
0.00007 0.00001
#180
1 0 0 0 1 0 0 1 0
0.02585 0.97984 0.00029 0.0004 0.02405 0.00566 0.00006 0 0.00012 0.00003
0.00007 0.00001
#181
1 0 0 0 1 0 0 1 0
0.00768 0.99856 0 0.00064 0.96046 0.02067 0.00001 0 0.00031 0.0002
0.04763 0.00005
#182
1 0 0 0 1 0 0 1 0
0.00774 0.99856 0 0.00064 0.95943 0.02056 0.00001 0 0.00031 0.0002
0.04639 0.00005
#183
1 0 0 0 1 0 0 1 0
0.00239 0.99749 0 0.00757 0.99988 0.00921 0 0 0.00496 0.00032
0.94229 0.00007
#184
1 0 0 0 1 0 0 1 0
0.00239 0.99749 0 0.00757 0.99987 0.00921 0 0 0.00496 0.00032
0.94218 0.00007
#185
0 0 0 0 0 0 0 0 0
0.00462 0.01314 0.00928 0.005 0.00954 0.01759 0.00066 0.01194 0.00962 0.00407
0.00841 0.00862
#186
1 0 0 0 1 0 1 0 0
0.01184 0.01081 0.00004 0.00425 0.00046 0.0039 0.00004 0.00002 0.0001 0.00116
0.00001 0.00016
#187
1 0 0 0 1 0 1 0 0
0.91061 0.62501 0.00006 0.00138 0.21142 0.00003 0.00003 0 0.00015 0.00052
0.00008 0
#188
1 0 0 0 1 0 1 0 0
0.923 0.60101 0.00007 0.00148 0.19539 0.00003 0.00004 0 0.00016 0.00052
0.00008 0
#189
1 0 0 0 1 0 1 0 0
0.99904 0.03479 0.00162 0.09878 0.03719 0 0.00026 0.00001 0.00179 0.00026
0.0001 0
#190
1 0 0 0 1 0 1 0 0
0.99928 0.03076 0.00222 0.11828 0.02829 0 0.00029 0.00001 0.00183 0.00026
0.00008 0
#191
1 0 0 0 1 0 1 0 0
0.9999 0.004 0.01261 0.77998 0.00181 0 0.00057 0.00003 0.00545 0.00031
0.00001 0
#192
1 0 0 0 1 0 1 0 0
0.99991 0.00335 0.01116 0.82526 0.00172 0 0.00055 0.00003 0.00674 0.0003
0.00001 0
#193
0 0 0 0 0 0 0 0 0
0.00462 0.01314 0.00928 0.005 0.00954 0.01759 0.00066 0.01194 0.00962 0.00407
0.00841 0.00862
#194
1 0 0 1 0 0 0 0 1
0.01184 0.01081 0.00004 0.00425 0.00046 0.0039 0.00004 0.00002 0.0001 0.00116
0.00001 0.00016
#195
1 0 0 1 0 0 0 0 1
0.84809 0.13349 0.17257 0 0.00001 0.01439 0.01005 0.00025 0 0.00004
0 0.0001
#196
1 0 0 1 0 0 0 0 1
0.86768 0.12425 0.17483 0 0.00001 0.01157 0.009 0.00025 0 0.00004
0 0.00009
#197
1 0 0 1 0 0 0 0 1
0.99837 0.00509 0.99759 0.00012 0 0.00437 0.76938 0.01496 0 0.00003
0 0.00017
#198
1 0 0 1 0 0 0 0 1
0.99843 0.00496 0.99771 0.00012 0 0.00417 0.77198 0.01533 0 0.00003
```

```
0 0.00017
#199
1 0 0 1 0 0 0 0 1
0.99907 0.00208 0.9996 0.00015 0 0.00499 0.93823 0.07912 0 0.00004
0 0.00037
#200
1 0 0 1 0 0 0 0 1
0.99907 0.00208 0.9996 0.00015 0 0.00499 0.93821 0.07917 0 0.00004
0 0.00037
#201
0 0 0 0 0 0 0 0 0
0.00462 0.01314 0.00928 0.005 0.00954 0.01759 0.00066 0.01194 0.00962 0.00407
0.00841 0.00862
#202
1 0 0 1 0 0 0 1 0
0.01184 0.01081 0.00004 0.00425 0.00046 0.0039 0.00004 0.00002 0.0001 0.00116
0.00001 0.00016
#203
1 0 0 1 0 0 0 1 0
0.53688 0.51409 0.14111 0 0.00001 0.13485 0.03288 0.00027 0 0.00003
0 0.00045
#204
1 0 0 1 0 0 0 1 0
0.57852 0.479 0.14214 0 0.00001 0.10696 0.02944 0.00027 0 0.00003
0 0.00036
#205
1 0 0 1 0 0 0 1 0
0.88363 0.21663 0.79535 0.00001 0 0.48539 0.48977 0.00154 0 0.00002
0 0.00469
#206
1 0 0 1 0 0 0 1 0
0.94492 0.11759 0.89596 0.00001 0 0.30315 0.58595 0.00233 0 0.00002
0 0.00293
#207
1 0 0 1 0 0 0 1 0
0.99646 0.02403 0.9976 0.00009 0 0.12449 0.89706 0.01275 0 0.00003
0 0.00301
#208
1 0 0 1 0 0 0 1 0
0.9981 0.00846 0.99897 0.0001 0 0.02943 0.93529 0.02538 0 0.00003
0 0.00097
#209
0 0 0 0 0 0 0 0 0
0.00462 0.01314 0.00928 0.005 0.00954 0.01759 0.00066 0.01194 0.00962 0.00407
0.00841 0.00862
#210
1 0 0 1 0 0 1 0 0
0.01184 0.01081 0.00004 0.00425 0.00046 0.0039 0.00004 0.00002 0.0001 0.00116
0.00001 0.00016
#211
1 0 0 1 0 0 1 0 0
0.9691 0.02838 0.03586 0.00001 0.00003 0.0022 0.00158 0.00009 0 0.00003
0 0.00002
#212
1 0 0 1 0 0 1 0 0
0.97152 0.02817 0.03743 0.00001 0.00003 0.00199 0.00155 0.00009 0 0.00003
0 0.00002
#213
1 0 0 1 0 0 1 0 0
0.99904 0.00672 0.95731 0.00079 0 0.00012 0.03336 0.00137 0.00001 0.00003
0 0
#214
1 0 0 1 0 0 1 0 0
0.99904 0.00671 0.95787 0.00079 0 0.00012 0.03374 0.00138 0.00001 0.00003
0 0
#215
1 0 0 1 0 0 1 0 0
0.99884 0.00336 0.99936 0.00011 0 0.00675 0.93783 0.03982 0 0.00003
0 0.00032
#216
1 0 0 1 0 0 1 0 0
0.99884 0.00336 0.99936 0.00011 0 0.00676 0.93787 0.03985 0 0.00003
0 0.00033
#217
0 0 0 0 0 0 0 0 0
0.00462 0.01314 0.00928 0.005 0.00954 0.01759 0.00066 0.01194 0.00962 0.00407
0.00841 0.00862
```