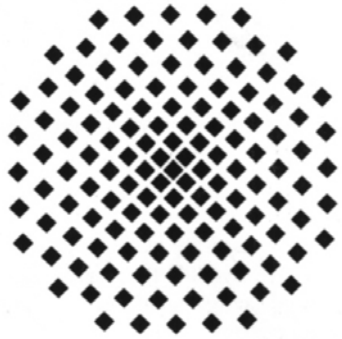
UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas de São Carlos
Laboratório de Inteligência Computacional
LABIC



SNNS

Stuttgart Neural Network Simulator

Manual de Referência Rápida

Helio Diniz
Elson F. Mendes Filho
André C. P. L. F. de Carvalho

Indíce:

1. Introdução **01**

2. Terminologia **02**
- **2.1. Blocos de uma rede neural** **02**
  - 2.1.1. Unidades 02
  - 2.1.2. Conections (Links) 03
  - 2.1.3. Sites 03
- **2.2. Modos de Atualização** **03**
- **2.3. Aprendizagem em Redes Neurais** **03**
- **2.4.Generalização** **04**

3. XGUI - Graphical User Interface **05**
- **3.1. Criando uma rede neural** **06**
  - 3.1.1. Edição 08
  - 3.1.1.1. Camadas de Neurônios 08
  - 3.1.1.2. Ligações 08
  - 3.1.2. Criação 09
- **3.2. Salvando e Carregando uma rede neural e seus dados** **09**
- **3.3. Visualizando uma rede neural** **11**
  - 3.3.1. Setup Panel 12
- **3.4. Operando uma rede neural** **13**
- **3.5. Visualizando erro de aprendizagem** **19**
- **3.6. Visualizando os pesos de uma rede neural** **20**
- **3.7. Visualizando as unidades de uma rede neural** **21**
- **3.8. Otimizando as ligações de uma rede neural** **22**
- **3.9. Analisando Projeção em neurônios de uma rede neural** **23**
- **3.10. Imprimindo uma rede neural** **25**
- **3.11. Obtendo informação sobre o SNNS** **26**
- **3.12. Outras funcionalidades da XGUI** **26**

4. Dados de uma rede neural **27**
- **4.1. Convenções** **27**
- **4.2. Símbolos Terminais** **27**
- **4.3. Gramática** **28**
- **4.4. Exemplo** **28**

5. Operando o Simulador por batch **29**
- **5.1. Chamando o Interpretador batch** **29**
- **5.2. Linguagem do Interpretador** **30**
  - 5.2.1. Tipos de Dados 30
  - 5.2.2. Variáveis 30
  - 5.2.3. Expressões e Operadores 31
  - 5.2.4. Função Print 31
  - 5.2.5. Estrturas de Controle 32
  - 5.2.6. Chamadas de Funções do SNNS 32
    - 5.2.6.1 Chamadas de Funções para selecionar parâmetros do SNNS 32
    - 5.2.6.2. Chamadas de Funções relacionadas a redes 33
    - 5.2.6.3. Chamadas de Funções relacionadas a padrões 33
    - 5.2.6.4. Funções Especiais 34
    - 5.2.7. Exemplo 34

6. Transformando uma rede neural em programa "C" **35**
6.1. Fluxo do Programa **35**
6.2. Incluindo uma rede compilada em sua aplicação **35**
6.3 Funções de Ativação **36**
6.4 Mensagens de Erro **36**
7. Conclusão **37**
APÊNDICE A - Parâmetros das Funções de Aprendizagem **38**
APÊNDICE B - Exemplo de Aplicação **42**
APÊNDICE C - Parâmetros das Funções de Aprendizagem **50**

1. INTRODUÇÃO

A manipulação de uma rede neural, do inicio da sua aprendizagem até o final dos testes, é uma tarefa árdua e dispendiosa. Uma boa tentativa de diminuir os esforços deste trabalho é a utilização de um ambiente simples e flexível para manutenção e treinamento de redes neurais, fornecendo uma maneira adequada de visualizar sua performance na aprendizagem, validação e teste.

O SNNS v4.1 (Stuttgart Neural Network Simulator) é um pacote de software de uso compartilhado criado na Universidade de Stuttgart, que satisfaz boa parte destas exigências. O objetivo deste manual de referência rápida é apenas fornecer uma visão básica dos recursos do SNNS, ilustrando as atividades mais comuns que um usuário pode realizar com o simulador.

O primeiro passo é enfocar a terminologia empregada pelo SNNS. Os elementos básicos de uma rede neural (neurônios, conexões, e sites) e os conceitos de aprendizagem, atualização e generalização são abordados de acordo com o ponto de vista do simulador.

Em seguida, este manual enfoca a utilização da interface gráfica do simulador conhecida como XGUI. Neste ponto, são discutidas as atividades de criação, operação (aprendizagem, validação e teste), visualização (erros de aprendizagem e validação, pesos e unidades em particular), impressão, otimização das conexões, etc de uma rede neural. A manutenção dos arquivos suportado pelo simulador também é ilustrada. Algumas dicas sobre o funcionamento do recurso de Help do SNNS são ainda fornecidas.

O próximo tópico diz respeito ao formato dos arquivos de padrões utilizados no simulador. São feitas algumas considerações sobre a gramática de arquivos de padrões SNNS, ilustrando suas convenções e seus símbolos terminais. É também apresentado um exemplo de arquivo de padrões para treinar uma rede para resolver o problema XOR.

O manual termina ilustrando a utilização do simulador em batch através do BATCHMAN e a geração de redes em código C. Os aspectos da linguagem batch referentes a tipos de dados, variáveis, expressões, operadores, estruturas de controle, função de impressão, e chamada de funções do SNNS são apresentados detalhadamente, seguidos de exemplos. Os aspectos referentes a integração de uma rede neural treinada em um programa “C” são também discutidos.

Há também apêndices extras onde são descritos suscintamente todos os parâmetros das regras de aprendizagem empregadas pelo simulador, um exemplo

completo de uma aplicação prática de redes neurais, feito através do Batchman e endereços na internet com informações úteis sobre o SNNS.

2. TERMINOLOGIA

Para estabelecer uma via de comunicação comum, será apresentada, a seguir, a terminologia básica utilizada pelo SNNS.

2.1. BLOCOS DE UMA REDE NEURAL

2.1.1. Unidades: neurônios da rede neural podendo ser de três tipos:

- *unidades de entrada*: recebem entrada externa da rede;
- *unidades de saída*: representam a saída da rede;
- *unidades escondidas ou intermediárias:* recebem tal nome pois não são visíveis fora da rede.

Cada unidade no SNNS possui um conjunto de atributos que são apresentados a seguir:

- *no*: número de um neurônio, determinando a ordem na qual ele é armazenado no simulator kernel;
- *name*: nome para uma unidade fornecido pelo usuário. Tem que começar com uma letra e não pode conter brancos ou caracteres especiais;
- *io-type* ou *io*: tipo de cada unidade dentro da rede (entrada, saída, escondida, dual, especial, especial de entrada, especial escondida ou especial de saída). O tipo dual é para unidades que são tanto de entrada quanto de saída. O tipo especial é para unidades cujos pesos não são alterados durante a aprendizagem.

Os últimos três tipos permitem a correlação de uma unidade a várias camadas na rede.

- *activation*: valor de ativação da unidade;
- *bias*:complementar do threshold de uma unidade;
- *ativation function ou actFunc*: um valor de ativação é calculado a partir das entradas multiplicadas pelo pesos de suas ligações à unidade, e a partir do valor anterior de ativação e do bias. Quando sites são usados, a entrada da rede é calculada a partir dos valores dos sites. As funções de ativação são funções C relativamente simples ligadas ao simulator kernel.
- *output function ou outFunc*: calcula o valor de saída de cada unidade a partir da sua ativação atual. Geralmente é a função Identidade (SNNS: Out_identify) assumida como default pelo simulador.
- *f-type*: tipo de funcionalidade das unidades.
- *position*: Cada unidade possui uma posição associada, que consiste de 3 coordenadas inteiras x, y, e z em um espaço 3D. Em um espaço 2D, apenas as coordenadas x e y são utilizadas.
- *subnet no*: Cada unidade é atribuída a uma subnet montando redes estruturadas que podem ser visualizadas mais claramente do que em uma representação 2D.

- *layers*: As unidades podem ser visualizadas em 2D com até 8 camadas. Camadas podem ser visualizadas seletivamente.
- *frozen*: este atributo determina que a saída ou ativação está congelada, ou seja, não se altera durante a simulação.

2.1.2. Conections (Links): ligação de uma unidade fonte para uma outra unidade destino e sempre única para essas unidades. Não são permitidas conexões redundantes. Cada conexão tem um peso associado. Este peso representa o efeito da saída da unidade anterior na unidade sucessora. O peso pode ser negativo (inibitório) ou positivo (excitatório) e consiste de um valor real de ponto flutuante com 9 dígitos de precisão.

2.1.3. Sites: Uma unidade com sites não possui suas unidades de entrada diretamente conectadas aos estímulos de entrada. Todas as ligações conduzem a diferentes sites, onde os sinais de saída ponderados que chegam da unidade precedente são processados por diferentes funções sites. Estas funções sites podem ser definidas pelo usuário e seus resultados são representados por valores sites. A função de ativação toma então o valor de cada site como entrada da rede. São permitidas conexões de uma mesma unidade destino para diferentes sites. Também são permitidas múltiplas conexões de uma mesma unidade para diferentes sites de entrada.

2.2. MODOS DE ATUALIZAÇÃO

São os métodos utilizados para obter os valores de ativação dos neurônios de uma rede. O SNNS possui 5 modos de atualização. O primeiro deles é assíncrono sendo os demais síncronos. Uma atualização síncrona é aquela cujas unidades conseguem acessar as novas saídas de seus predecessores apenas se estes foram ativados anteriormente.

- *síncrono*: os estados de ativação de todas unidades são alteradas simultaneamente.
- *permutação randômica*: os estados de ativação das unidades são alterado sequencialmente. A ordem é randômica, mas apenas uma unidade é selecionada em cada passo.
- *random*: a ordem de atualização é definida pelo gerador de números randômico não garantindo que todas unidade serão visitadas em um instante de tempo t.
- *serial*: a ordem é determinada pelo número interno da unidade.
- *topological*: o kernel ordena as unidades pela sua topologia, fazendo a propagação natural da atividade da entrada para saída.

Há mais 12 modos de atualização no SNNS: CPN(counterpropagation), Time Delay(redes Time Delay), ART1_Stable, ART2_Syncronous e ARTMAP_Syncronous(modelos de redes para teoria de ressonância adaptativa), CC(Cascade Correlation) e RCC(Recursive Cascade Correlation), BPTT(redes recursivas treinadas com Backpropagation Through Time), RM_Synchronous(redes de memória auto-associativa).

2.3. APRENDIZAGEM EM REDES NEURAIS

A aprendizagem consiste da modificação dos pesos da rede para adapta-la a um conjunto de padrões. As formas de aprendizagem geralmente são baseadas na regra Hebbiana. A regra de Hebb determina que a ligação entre duas unidades é forte se as duas estão ativas ao mesmo tempo. O simulador suporta várias regras de aprendizagem que podem ser utilizadas nos modos *online*, *offline* ou *batch*.

2.4. GENERALIZAÇÃO

É a capacidade da rede classificar dados que nunca antes vistos. Para melhorar a generalização, os dados que alimentam uma rede são geralmente divididos em: **conjunto de treinamento** (usado durante a fase de treinamento), **conjunto de validação** (usado para determinar a performance da rede com padrões que não foram utilizados durante treinamento) e **conjunto de teste** (checagem da performance global da rede).

O treinamento deve parar quando for obtido um erro mínimo de validação. Para tal, o SNNS pode realizar um ciclo de validação a cada ciclo de treinamento.

3. XGUI - GRAPHICAL USER INTERFACE

A XGUI é uma interface gráfica que permite interação do usuário com o SNNS em sistemas operacionais baseados em Unix. Ela consiste dos seguintes arquivos:

- *snns* ou *xgui*: programa simulador SNNS (XGUI e simulator kernel juntos);
- *default.cfg*: configuração default;
- *help.hdoc*: textos de ajuda utilizados pelo XGUI.

O SNNS utiliza as seguintes extensões para seus arquivos:

.net arquivos de rede (unidades e ligações)
.pat arquivos de padrões
.cfg arquivos de configurações
.txt arquivos texto (log files)
.res arquivos de resultados (ativações de unidades)

sendo iniciado pelo comando:

snns [<netfile>.net] [<pattern>.pat] [<config>.cfg] [options] <RETURN>

onde as opções válidas são:

-font <name>: fonte para simulador
-dfont <name>: fonte para display
-mono: branco & preto em um monitor colorido
-help: tela de help para explicar as opções

Ex.: snns -font 7x13bold <RETURN>

OBS.: as fontes disponíveis podem ser detectadas através do programa xfontsel (não é parte da distribuição).

Há várias janelas diferentes do XGUI para diferentes funções. Mesmo assim, elas possuem em comum:

- uma barra de título.
- um botão no canto esquerdo superior para manipulação da janela.
- um botão DONE que fecha a janela.
- e, as vezes, um botão CANCEL que cancela o acionamento da janela.

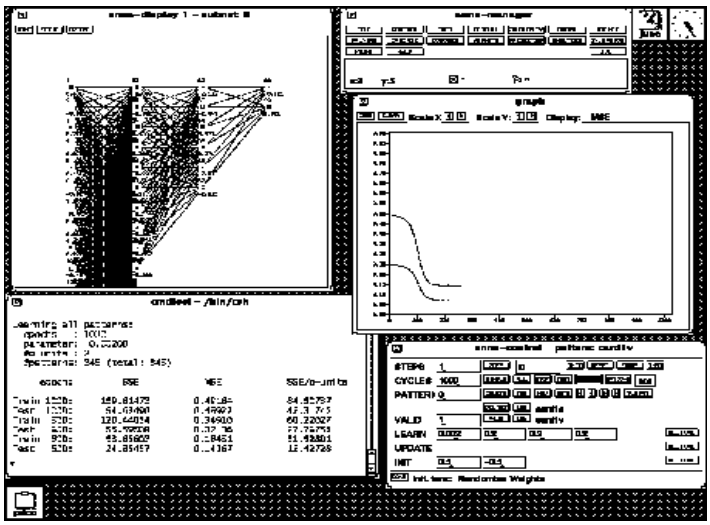


Figura 1 - As várias janelas do XGUI.

O SNNS controla todas as operações sobre uma rede neural através da janela principal MANAGER PANEL apresentada na Figura 2. Os campos desta janela serão apresentados ordenadamente de acordo com as atividades necessárias para a construção uma rede utilizando o simulador.

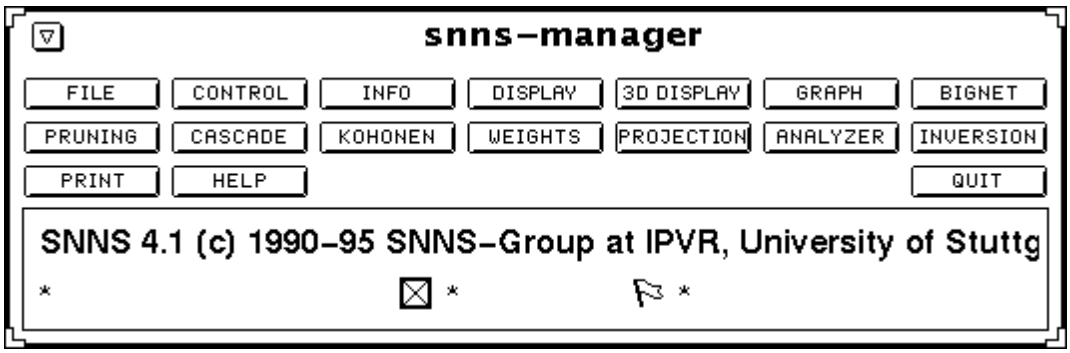


Figura 2 - MANAGER PANEL.

3.1. CRIANDO UMA REDE NEURAL

A primeira operação a ser vista será a edição e criação de uma rede neural. Para isto será utilizado o BIGNET do MANAGER PANEL. Uma vez clicado este campo, será apresentado o menu da figura 3, com todas arquitetura suportadas pelo SNNS.

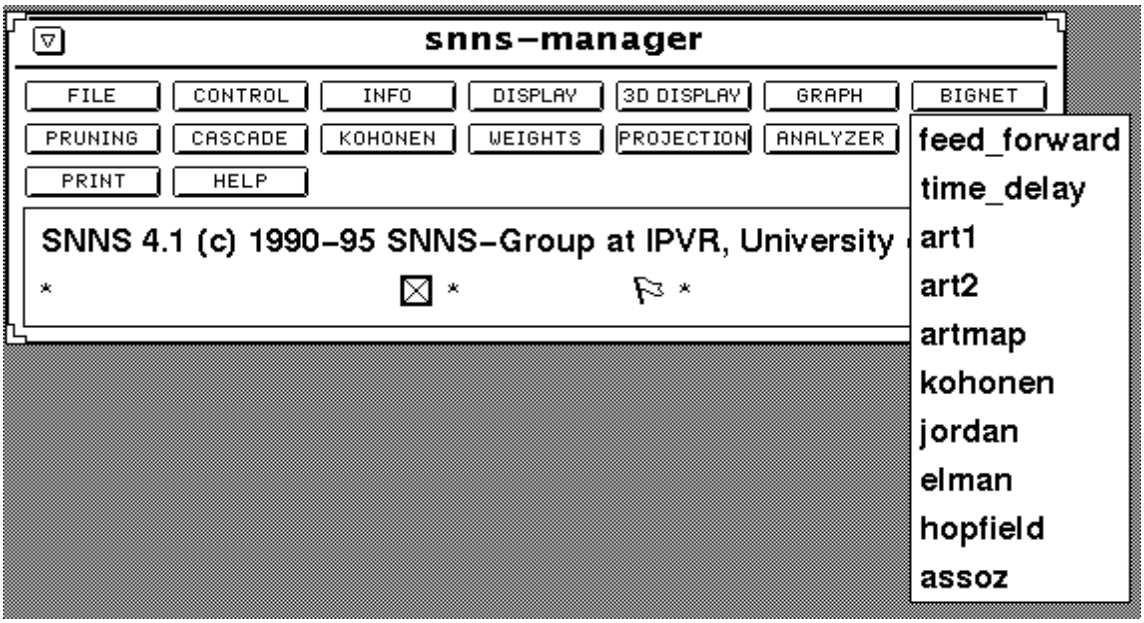


Figura 3 - Menu do BIGNET.

As telas do BIGNET variam de acordo com as característica da topologia escolhida. Será apresentado somente o exemplo para uma rede feedforward, por ser a mais comum, ilustrado na figura 4. Detalhes do BIGNET para as demais arquiteturas são encontrados no manual original.

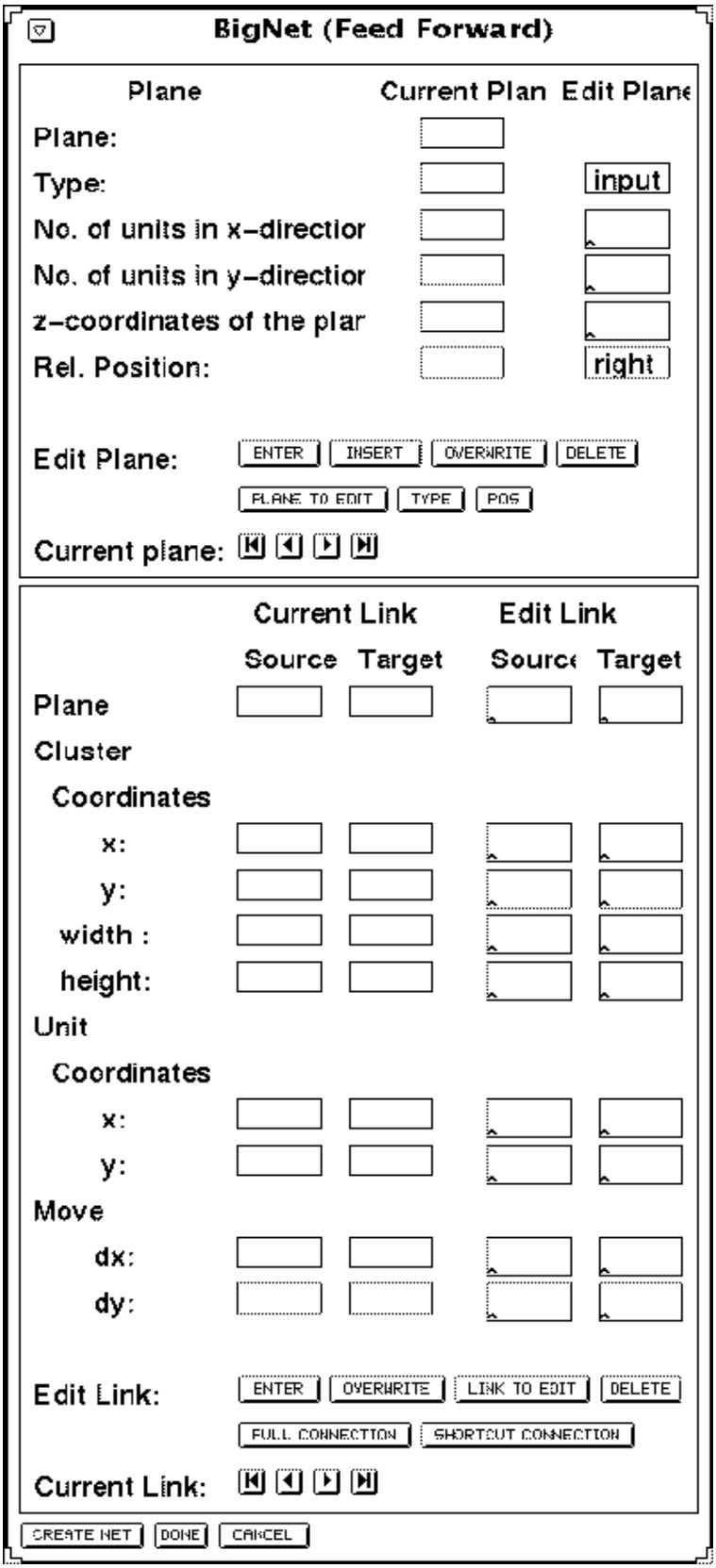


Figura 4 - BIGNET de redes feedforward.

Para uma arquitetura *feedforward*, a edição da rede neural no BIGNET é dividida em duas partes: uma para manipulação das camadas de neurônios da rede e outra para manipulação das ligações entre as camadas da rede. Uma vez editada a rede, só resta cria-la na memória.

3.1.1. Edição

3.1.1.1 Camadas de Neurônios

No BIGNET, as camadas de neurônios são chamadas de **planos** que são arrays bidimensionais de unidades. Cada plano possui coordenadas e a unidade na extremidade esquerda superior do plano tem coordenadas (1,1). Um grupo de unidades em um plano com formato quadrado é chamado **cluster,** cuja posição é determinada pela coordenadas de sua extremidade superior esquerda e suas expansões nas direções x e y.

Uma camada de neurônio será editada na coluna **Edit Plane**. Cada plano possui:

- uma identificação **Plane**.
- um tipo **Type** que pode ser *input*, *output* e *hidden*.
- o número de neurônios na direção x (**No. of units in x-direction**).
- o número de neurônios na direção y (**No. of units in y-direction**).
- uma coordenada para visualização em 3D (**z-coordenates of the plane**).
- e sua posição relativa a última camada **Rel. Position** que pode ser *right*, *below* ou *left.*
- um botão ENTER que insere o plano no final lista de planos.
- um botão INSERT que inclui o plano editado na frente do plano corrente.
- um botão OVERWRITE que altera o plano atual (**Current Plane**) pelos valores de **Edit Plane**.
- um botão PLANE TO EDIT que inicia a alteração de um plano.
- um botão TYPE que determina o tipo do plano.
- um botão POS que determina a posição relativa do plano.
- um botão DELETE que apaga o plano corrente.
- setas para navegar pela lista de planos.

OBS.: a coordenada z ainda não foi implementada.

3.1.1.2 Ligações

O editor de ligações possui uma coluna para ligações visualizadas em um dado instante (**Current Link**) e outra para editá-las (**Edit Link**). As ligações sempre conduzem de uma fonte para um destino. Para gerar uma conexão completa (de cada plano *i* para seu sucessor *i+1*) basta pressionar o botão FULL CONECTION depois da definição dos planos.

Se forem desejadas ligações mais complicadas, pode-se editá-las diretamente através do SHORT CONNECTION. O SNNS possui 9 diferentes combinações para criação de ligações:

todas unidades de um plano todas unidades de um plano

Ligações de	todas unidades de um cluster uma unidade simples	para	todas unidades de um cluster uma unidade simples

Em qualquer combinação, os planos fonte e destino devem ser preenchidos em **Plane**, mesmo se a fonte ou destino for um cluster ou unidade simples. A edição da ligações consiste em preencher os campos para o planos, cluster ou unidades da maneira desejada. O campo **Move** determina quantos passos um cluster ou unidade será movida na direção x ou y dentro de um plano depois que o cluster ou unidade é conectada com um destino ou fonte. Além disso, o **Link Editor** possui:

- Um botão ENTER que insere a ligação no final da lista de ligações.
- Um botão OVERWRITE que altera a ligação atual (**Current Link**) pelos valores de **Edit Link**.
- Um botão LINK TO EDIT que inicia a alteração de uma ligação.
- Um botão DELETE que apaga uma ligação.
- Setas para navegar pela lista de ligações.

3.1.2. Criação

A criação da rede na memória é feita pressionando o botão CREATE NET. Os pesos das ligações são inicializados com o valor 0.5 e a rede é armazenada no arquivo default **SNNS_NET.net**. Se já existir uma rede na memória, ela será substituída pela nova. Se a rede usada possuir ligações redundantes, aparecerá a mensagem de erro "Invalid Target". Neste caso, a ligações devem ser revisadas e a redundância consertada.

3.2. SALVANDO E CARREGANDO UMA REDE NEURAL E SEUS DADOS

O SNNS utiliza o **File Browser** para salvar e carregar redes, padrões, configurações (número, locação e dimensão do display bem como seus valores de setup e nome das camadas), e índices de todas janelas de texto. O **File Browser** é invocado através do botão FILE no MANAGER PANEL. O **File Browser**, como ilustrado na figura 5, consiste de:

- Uma linha topo que mostra o diretório corrente.
- Uma janela de arquivos e diretórios dentro do diretório corrente.
- Uma outra linha que mostra o arquivo corrente.
- Os botões de seleção do tipo de arquivo: NET (**Networks -** extensão *.net*), PAT (**Patterns -** extensão *.pat*), RES (**Result Files -** extensão *.res*), CFG (**Config -** extensão *.cfg*) e TXT (**Log File -** extensão *.txt*).
- Os botões SAVE e LOAD para salvar ou carregar um arquivo.

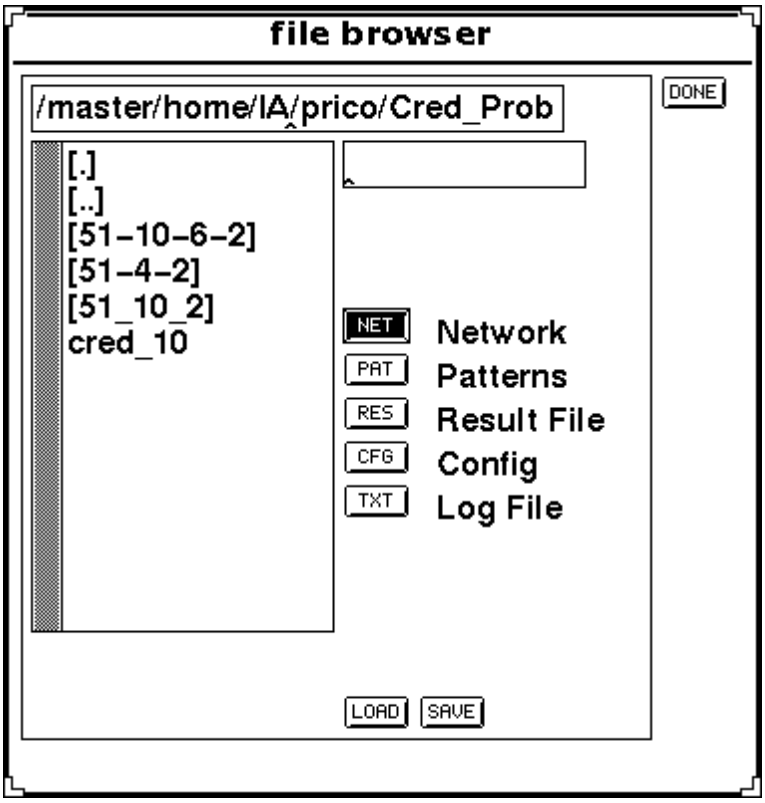


Figura 5 - file browser.

OBS.:

- Se houver uma rede na memória e for carregado um arquivo de rede, esta rede será substituída. Antes da substituição haverá um pedido de confirmação.
- Uma configuração contêm a locação e tamanho de todo o display com seus parâmetros e nome de várias camadas. É uma informação independente da rede e por isso pode ser armazenada separadamente.
- Um arquivo de resultado contêm as ativações de todas a unidade de saída e só pode ser salvo. Estas ativações são obtidas executando um passo da propagação forward. Depois de pressionar o botão SAVE, aparecerá um menu para o usuário selecionar quais padrões devem ser testados além da saída de teste.
- Mensagens que documentam a simulação podem ser armazenada no **log file** (.TXT). Este arquivo contêm operações de arquivo, definições do conjunto de valores, bem como os valores de aprendizagem (ciclos, parâmetros e erros).

3.3. VISUALIZANDO UMA REDE NEURAL

A visualização da topologia de uma rede neural, com as ativações de suas unidades e pesos da conexões, é feita através do **2D Display** ou simplesmente **Display**. Esta ferramenta é acionada através do botão DISPLAY do MANAGER PANEL, como ilustrado na figura 6:

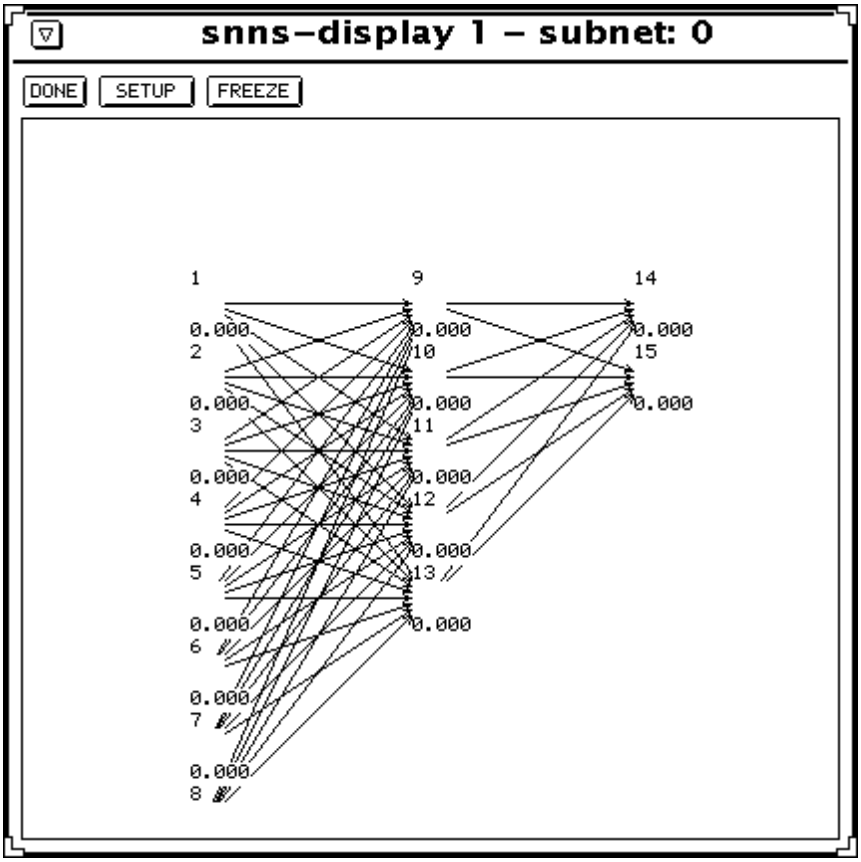


Figura 6 - DISPLAY de uma rede 8 - 5 - 2.

Que consiste de:

- Um botão FREEZE que "congela" a rede não permitindo sua atualização e nem a aceitação comandos do editor.
- Um botão SETUP para configurar a visualização das unidades, ligações e camadas.
- Uma janela de visualização da rede neural na memória em um dado instante. Cada unidade é uma caixa de tamanho proporcional à sua ativação localizada em uma posição do grid. A identificação de uma unidade (nome, número e valor-z) é posicionada sobre ela. As informações da ativação (ativação, ativação inicial, bias e saída) são posicionadas abaixo das unidades.

3.3.1 Setup Panel

Alterações do tipo de display podem ser realizadas no **Setup Panel** mostrado abaixo:

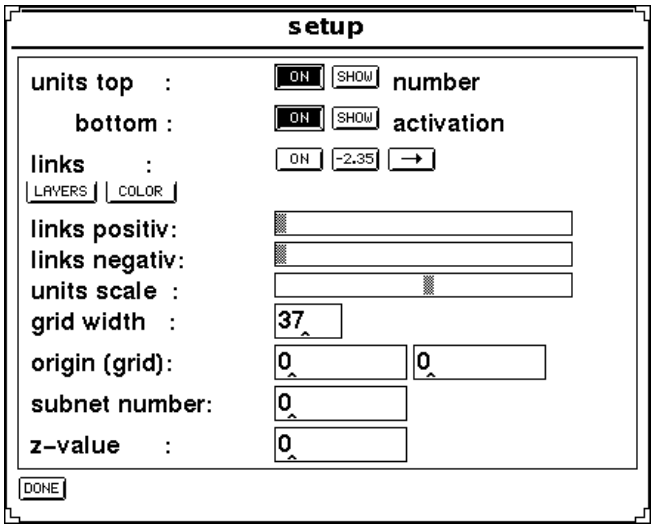


Figura 7 - SETUP PANEL.

Que consiste de:

- Duas linhas UNIT TOP e BOTTOM, que controlam o display de informações das unidades: identificação das unidades e ativação respectivamente. Há um botão ON que ativa as informações selecionadas pelo botão SHOW em cada uma das linhas. O nome da unidade, número da unidade ou seu valor-z podem ser mostrados acima dela. A ativação, ativação inicial, bias ou saída da unidade podem ser mostrada abaixo.
- Uma linha LINKS, que controla o display de informações das ligações. Um botão ON determina o desenho de todos as ligações. Um botão [-2.35] mostra os pesos das ligações no centro das linhas que as representam. Um botão → mostras as ligações como setas apontando para a unidade destino.
- Uma linha com os botões LAYERS e COLOR, para controlar a visualização da camadas e as cores do **2D-Display**. O botão LAYERS invoca um menu para selecionar a visualização de até oito camadas diferentes em uma mesma tela. No SNNS, camadas são vistas como transparências permitindo uma visualização seletiva das unidades e ligações. O botão COLOR, em um monitor colorido, abre uma janela de edição das cores dos elementos de um rede neural.
- Três barras de rolagem LINKS POSITIV, LINK NEGATIV e LINK SCALE, para seleção dos parâmetros de visualização das ligações.
- Uma barra de rolagem UNITS SCALE, que determina o parâmetro fator de escala para o tamanho das caixas das unidades. O intervalo válido é entre 0.0 e 2.0. A unidade com ativação 1 será desenhada com o tamanho do limite superior do fator de escala.
- Um campo texto GRID WIDTH que determina a distância entre as unidades no grid. O valor default é de 37 pixels.
- Dois campos ORIGIN (GRID), que determinam a posição inicial da janela **Graph**.

- Um campo SUBNET NUMBER, que ajusta o número da subnet a ser visualizada na janela. Valores entre -32736 e 32735 são válidos.

3.4. OPERANDO UMA REDE NEURAL

Todas as operações possíveis sobre as redes são realizadas através do **Control Panel**. O **Control Panel** é acionado através do botão CONTROL do MANAGER PANEL. Através dele controla-se a inicialização dos pesos da rede, o aprendizado, a atualização da rede, etc. A figura 8 abaixo mostra a janela do **Control Panel**.

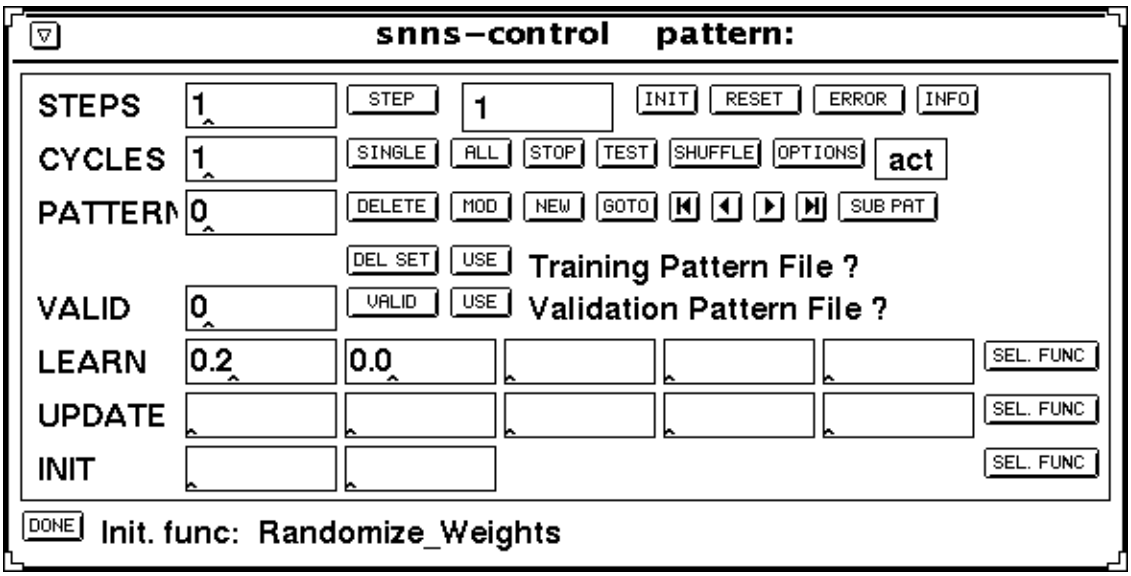


Figura 8 - CONTROL PANEL.

Que consiste de:

I. Uma linha que controla atualização e inicialização da rede além dos resultados estatísticos. Esta linha é formada por:

- ♦ Um campo de texto STEPS, que especifica o número de passos de atualização da rede. Em redes comuns (Ex.: **Topological_Order**), um passo é suficiente. Em redes recursivas, ou com outros modos de atualização, mais de um passo pode ser necessário.
- ♦ Um botão STEP, que executa o número de passos especificado no campo de texto STEPS.
- ♦ Um campo de texto COUNT, que registra execuções já realizadas.
- ♦ Um botão INIT, que inicializa a rede com valores de acordo com a função e parâmetros dados na última linha do panel.
- ♦ Um botão RESET, que reinicializa o contador, e atribui uma ativação inicial às unidades.
- ♦ Um botão ERROR, que imprime várias estatísticas utilizando modelos lineares. As estatísticas fornecem informação sobre *n* (tamanho da amostra), *p* (número de parâmetros a ser estimado - Ex. pesos), *SSE* (somatória do erros ao quadrado), e

TSS (o total da somatória dos quadrado corrigido pela raiz da variável independente).

- Um botão INFO, que fornece informações sobre a rede.

II. Uma linha que controla aprendizagem, constituída por:

- Um campo de texto CYCLES, que especifica o número de ciclos de aprendizagem
- Um botão SINGLE, que provoca o treinamento da rede com um único padrão para o número de ciclos definido em CYCLES. A janela de texto relata o erro a cada CYCLES/10 ciclos. O erro é o somatório da diferença quadrática entre a saída real e a desejada da rede sobre todos as unidades de saída. O erro médio por unidade de saída é dado abaixo de **ave**.
- Um botão ALL, que provoca o treinamento da rede com todos os padrões para o número de ciclos definido em CYCLES. O erro é o somatório da diferença quadrática entre a saída real e a desejada da rede sobre todas as unidades de saída somada sobre o número de padrões apresentados. O erro médio por unidade de saída é dado abaixo de **ave**.
- Um botão STOP, que interrompe o ciclo de treinamento.
- Um botão TEST, que permite testar a performance da rede com todas os padrões utilizados no treinamento. Os valores de ativação das unidades de entrada e saída são copiados na rede. Depois, o número de passos de atualização especificados em STEPS são executados.
- Um botão SHUFFLE para otimizar aprendizagem, fazendo vários padrões serem apresentados em diferente ordem em ciclos distintos. Uma seqüência randômica de padrões é criada automaticamente quando este botão é pressionado.
- Um botão OPTIONS, que oferece um menu com as opções: *jog weight* (altera todos os pesos automaticamente), *edit f-types* (edita/cria f-types) e *edit sites* (edita/cria sites). Se *jog weights* é selecionado, uma janela é apresentada com o alcance (low limits .. high limits) da distorção randômica a ser acrescentada em todas ligações da rede.
- Um botão ACT, que especifica as alterações para os valores de ativação das unidades de saída quando um padrão é aplicado com TEST. Este botão fornece três opções: *None* (unidade de saída permanece inalterada), *Out* (os valores de saída são computados e selecionados, ativações permanecem inalteradas) e *Act* (os valores de ativação são selecionados).

III. Duas linhas relacionadas com a manipulação dos padrões e são constituídas por:

- Um campo texto PATTERN, que mostra o número do padrão corrente.
- Um botão DELETE, que elimina o padrão mostrado por PATTERN do arquivo de padrões.
- Um botão MOD, que modifica no lugar o padrão mostrado por PATTERN.
- Um botão NEW, que define um novo padrão que é adicionado atrás dos padrões existentes. Valores de entrada e saída são definidos como acima.
- Um botão GOTO, que avança até o padrão cujo número é mostrado por PATTERN.

- ♦ Botões de setas para navegar através de todos padrões carregados, bem como ir diretamente para o primeiro ou último deles.
- ♦ Um botão SUB-PAT, que abre um painel para manipulação de sub-padrões. Este botão é inativo quando o conjunto de padrões tem tamanho variável.
- ♦ Um botão DEL-SET, que abre o menu de conjunto de padrões. O conjunto de padrões da entrada selecionado é removido somente da memória principal.
- ♦ Um botão USE, que abre um menu de padrões carregados. O conjunto de padrões selecionados passa a ser o conjunto corrente.
- ♦ Um campo CURRENT PATTERN SET que mostra o nome do conjunto de padrões de treinamento usado no momento. Quando não há nenhum, é mostrado a mensagem "**Training Pattern File ?**".

IV. Uma linha relacionada com a validação do conjunto de padrões durante o treinamento. Ela é constituída por:

- ♦ Um botão VALID, que fornece o número de ciclos no qual o processo de treinamento é interrompido, realizando a validação. O valor zero inibe a validação. O erro de validação é impresso e plotado no **Graph Display**.
- ♦ Um botão USE, que abre um menu de conjuntos de padrões carregados. O conjunto de padrões selecionados passa a ser o conjunto corrente.
- ♦ Um campo VALIDATON PATTERN SET, que mostra o nome do conjunto de padrões de validação usado no momento. Quando não há nenhum, é mostrado a mensagem "**Validation Pattern File ?**".

V. Uma linha que fornece informações sobre as funções de aprendizagem e é constituída por:

- ♦ Campos texto LEARN, que especificam os parâmetros das funções de aprendizagem. O número de parâmetros válidos e seus significados depende da função de aprendizagem utilizada.
- ♦ Um botão SEL FUNC, que invoca um menu para selecionar a função de aprendizagem (procedimento de treinamento). As funções pré-definidas no SNNS são apresentadas na Tabela 1.

Função	Descrição
ART1	Algoritmo de aprendizagem ART1
ART2	Algoritmo de aprendizagem ART2
ARTMAP	Algoritmo de aprendizagem ARTMAP (todos os modelos ART por Carpenter&Grossberg)
BBPTT	Batch-Backpropagation para redes recursivas
BPTT	Backpropagation para rede recursiva
Backpercolation	Back percolation 1 (Mark Jurik)
BackpropBatch	Backpropagation para treinamento batch
BackpropMomentum	Backpropagation com termo momento
BackpropWeightDecay	Backpropagation com Weight Decay
CC	Algoritmo Cascade Correlation
Cunterpropagation	Counterpropagation (Robert Hetch-Nielsen)
Dynamic_LVQ	Algoritmo LVQ com alocação de unidade dinâmica
Hebbian	Regra de aprendizagem Hebbian
JE_PP	Backpropagation para redes Jordan-Elman
JE_PP_Momentum	BackpropMomentum para redes Jordan-Elman
JE_Quickprop	Quickprop para redes Jordan-Elman
JE_Rprop	Rprop para redes Jordan-Elman
Kohonen	Mapas Self Organizing de Kohonen
Monte-Carlo	Aprendizagem Monte Carlo
Pruning FeedForward	Algoritmos de Pruning
QPTT	Quickprop para redes recursivas
Quickprop	Quickprop (Scott Fahlman)
RCC	Cascade Correlation para redes recursivas
RM-delta	Delta Rule de Rumelhart-McClelland
RadialBasisLearning	Radial BasisFunctions modificadas
RBF-DDA	Aprendizagem Resilient Propagation
SimAnn_SS_error	Simulated Annealing com cálculo SSE
SimAnn_WTA_error	Simulated Annealing com cálculo WWTA
Std_Backpropagation	“Vanilla” Backpropagation
TimeDelayBackprop	Backpropagation para TDNNs (Alex Waibel)

Tabela 1 - Funções de aprendizagem fornecidas pelo SNNS.

I. Uma linha que fornece informações sobre as funções de atualização e é constituída por:

- Campos texto UPDATE, que especificam os parâmetros das funções de atualização. O número de parâmetros válidos e seus significados depende da função de atualização utilizada.
- Um botão SEL FUNC, que invoca um menu para selecionar a função de aprendizagem. As funções pré-definidas no SNNS são definidas na Tabela 2.

Função	Rede
ART1_Stable	ART1
ART1_Synchronous	ART1
ART2_Stable	ART2
ART2_Synchronous	ART2
ARTMAP_Stable	ARTMAP
ARTMAP_synchronous	ART1
Auto_Synchronous	Memória Auto associativa
BAM_Order	Bidirecional Associative Memory
BPTT_Order	Backpropagation-Through-Time
CC_Order	Cascade-Correlation
Cunterpropagation	Counterpropagation
Dynamic_LVQ	Dynamic-Learning-Vector-Quantization
Hopfield_Fixed_Act	Hopfield
Hopfield_Synchronous	Hopfield
JE_PP_Order	Jordan-Elman
JE_PP_Special	Jordan-Elman
Kohonen_Order	Self-Organizing Maps (SOMS)
RCC_Order	Cascade-Correlation recursivas
Random_Order	qualquer
Random_Permutation	qualquer
Serial_Order	qualquer
TimeDelay_Order	Time-Delay
Topological_Order	qualquer

Tabela 2 - Funções de atualização.

- Uma linha que fornece informações sobre as funções de inicialização e é constituída por:
  - ♦ Campos texto INIT que especificam os parâmetros das funções da inicialização. O número de parâmetros válidos e seus significados depende da função de inicialização utilizada.
  - ♦ Um botão SEL FUNC que invoca um menu para selecionar a função de inicialização. As funções pré-definidas no SNNS são definidos na Tabela 3.

Função	Rede
ART1_Weights	ART1
ART2_Weights	ART2
ARTMAP_Weights	ARTMAP
CC_Weights	Cascade-Correlation
ClippHebb	Memória Associativa
CPN_Rand_Pat	Counterpropagation
CPN_Weights_v3.2	Counterpropagation
CPN_Weights_v3.3	Counterpropagation
DLVQ_Weights	Dynamic Learning Vector Quantization
Hebb	Memória Associativa
Hebb_Fixed_Act	Memória Associativa
JE_Weights	Jordan-Elman
Kohonen_Rand_Pat	Self Organizing Maps (SOMS)
Kohonen_Const	Self Organizing Maps (SOMS)
Kohonen_Weights_v3.2	Self Organizing Maps (SOMS)
Monte-Carlo	Aprendizagem Monte Carlo
Pruning FeedForward	Algoritmos de Pruning
Pseudoniv	Memória Associativa
Randomize_Weights	Qualquer rede, exceto da familha ART
Random_Weights_Perc	Backpercolation
Quickprop	Quickprop (Scott Fahlman)
RBF_Weights	Radial Basis Function (RBFs)
RBF_Weights_Kohonen	Radial Basis Function (RBFs)
RBF_Weights_Redo	Radial Basis Function (RBFs)
RCC_Weights	Cascade-Correlation recursivas
RM_Random_Weights	Memória Associativa

Tabela 3 - Funções de inicialização.

OBS.: maiores detalhes sobre os parâmetros das funções de atualização e inicialização são encontrados em seções especiais do manual original. O apêndice A extra descreve os parâmetros das regras de aprendizagem.

3.5. VISUALIZANDO ERRO DE APRENDIZAGEM

A visualização do desenvolvimento erro de treinamento da rede é feita através da ferramenta **Graph**. Esta ferramenta é acionada pelo botão GRAPH do MANAGER PANEL. **Graph** desenha tal desenvolvimento somente durante o tempo em que ela não está fechada. A figura 9 abaixo mostra esta janela.

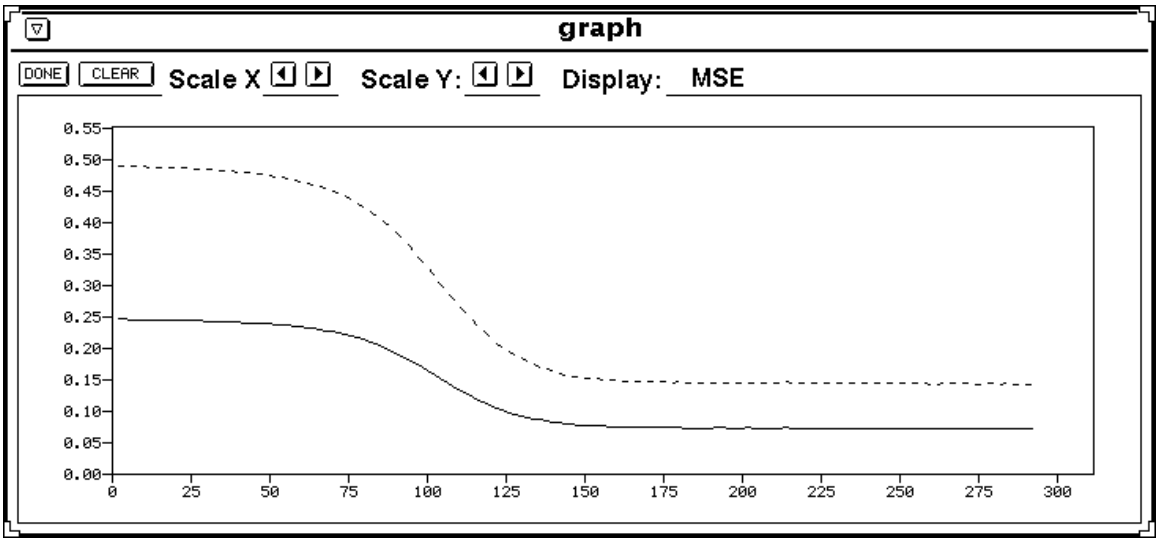


Figura 9 - GRAPH.

Que consiste de:

- Um botão CLEAR que limpa a tela da janela do gráfico e zera o contador de ciclos.
- Botões setas SCALE X e SCALE Y para controlar a dimensão das escalas no gráfico.
- Um botão SSE que abre um menu para selecionar o valores a serem plotados. As possíveis escolhas são: *SSE*, *MSE*, e *SSE/out* (*SSE* dividida pelo número de unidades de saída).

OBS.:

- A curva de erro da rede é plotada até a rede ser inicializada ou uma nova rede for carregada zerando o contador de ciclos.
- Enquanto o simulador estiver executando uma simulação, todos os botões da janela estarão bloqueados.
- Quando a validação for ativada no **Control Panel**, duas curvas serão desenhadas simultaneamente no **Graph**: uma para conjunto de treinamento e outra para conjunto de validação. Em um monitor colorido, o erro de validação será plotado como uma linha vermelha. Em um monitor branco e preto, será plotado como uma linha pontilhada.

3.6. VISUALIZANDO OS PESOS DE UMA REDE NEURAL

A visualização dos pesos de um rede neural ocorre no SNNS através da opção **Weight Display**. A janela, mostrada na figura 10 abaixo, é acionada através do botão WEIGHT do MANAGER PANEL:

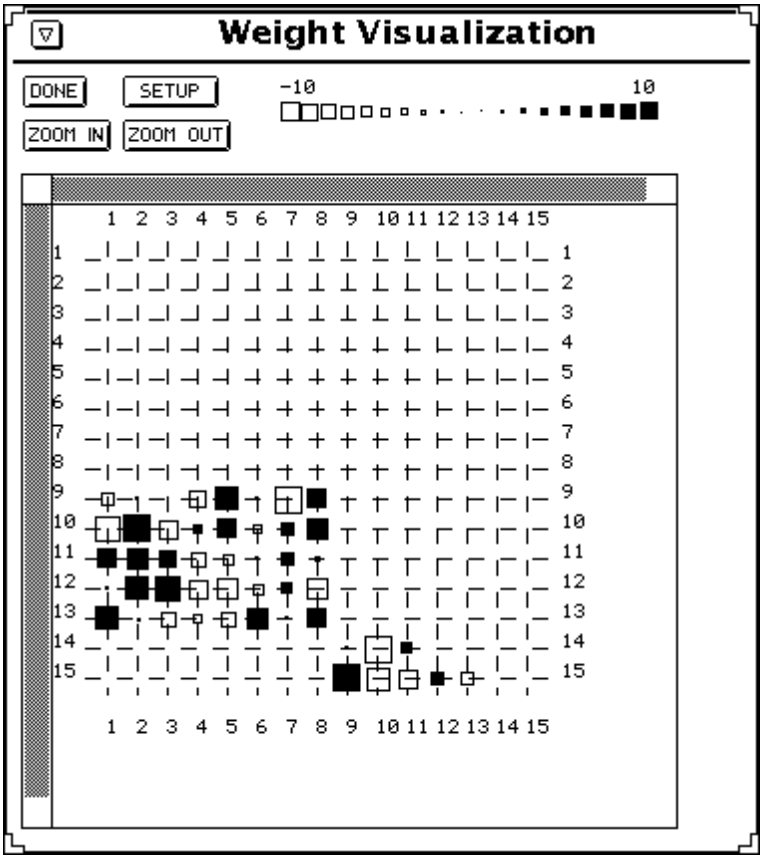


Figura 10 - WEIGHT DISPLAY.

Que engloba:

- Um botão SETUP para configuração da visualização dos pesos.
- Dois botões ZOOM IN e ZOOM OUT que aumentam ou diminuem o quadrado do pesos em um píxel por cada lado.
- Uma janela de visualização dos pesos. Em uma tela preto e branco, o pesos são representados como quadrados com alteração de tamanho no sistema Hinton. No sistema Hinton, o tamanho do quadrado corresponde ao valor absoluto do peso da ligação correlacionada e quadrados preenchidos representam ligações negativas. Em uma tela colorida, os quadrados têm tamanho fixo e um código de cor representando a intensidade da ligação. Ligações negativas são representadas por tons de vermelho, e positivas por tons de verde. Ligação nula é representada por branco.
- Uma barra de rolagem que permite navegar pela janela de pesos.

3.7. VISUALIZANDO UNIDADES DE UMA REDE NEURAL

O **Info Panel** mostra os dados de duas unidades e a ligação entre elas. Esta ferramenta é acionada através do botão INFO do MANAGER PANEL. Abaixo é mostrado o **Info Panel**:

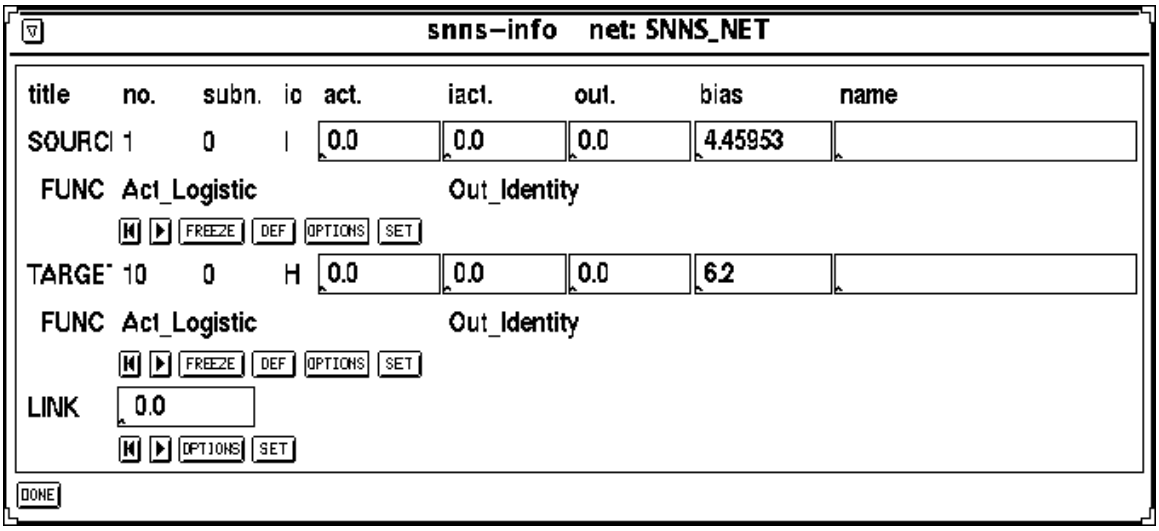


Figura 11 - INFO PANEL.

Que consiste de:

- Duas linhas SOURCE e TARGET que fornecem informações sobre as unidades fonte e destino. Essas informações são mostradas através dos campos: SUBN. (número da subnet), NO. (número da unidade) , ACT. (ativação), IACT. (ativação inicial), OUT (saída), BIAS (valor bias) e NAME (nome da unidade).
- Duas linhas que informam as funções de ativação e de saída usadas nas unidades fonte e destino.
- Dois botões de seta < para as unidades fonte e destino. Para unidade destino, este botão seleciona a primeira unidade destino para uma dada unidade fonte. Para unidade fonte, este botão seleciona a primeira unidade fonte para uma dada unidade destino.
- Dois botões de seta > um para a unidade fonte e outro para a unidade de destino. Para unidade destino, este botão seleciona a próxima unidade destino para uma dada unidade fonte. Para unidade fonte, este botão seleciona a próxima unidade fonte para dada unidade destino.
- Um botão FREEZE, que congela uma unidade fonte ou destino. As alterações tornam-se ativas somente depois de acionar SET.
- Um botão DEF, que atribui a unidade default aos valores exibidos de TARGET e SOURCE.
- Um botão OPTIONS que abre um menu com as opções: *change io-type* (altera o valor io-type), *change f-type* (altera f-type), *display activation function* (gráfico da função de ativação), *change activation function* (altera a função de ativação), *display output function* (gráfico da função de saída), *change output function* (altera a função de saída), *assign layers* (atribui a unidade a uma camada), *list all sources* (lista todos os predecessores), *list all targets* (lista todos os sucessores).

- Um botão SET para cada fonte e destino que, apenas depois de clicado, seleciona os atributos de uma unidade correspondente para um valor específico.
- Um campo LINK que mostra o valor do pesos da ligação entre SOURCE e TARGET.
- Um botão de seta < que seleciona a primeira ligação da unidade TARGET.
- Um botão de seta > que seleciona a próxima ligação da unidade TARGET.
- Um botão OPTIONS que abre menu com as opções: *list current site of TARGET* (lista de todas as ligações do site corrente), *list all sites of TARGET* (lista de todos os sites de um TARGET), *list all links from SOURCE* (lista de todas as ligações começando na SOURCE), *delete site* (apaga o site visualizado) e *add site* (adiciona o novo site para o TARGET).
- Um botão SET que, apenas após clicado, seleciona os pesos das ligações.

3.8. OTIMIZANDO AS LIGAÇÕES DE UMA REDE NEURAL

É possível otimizar as ligações de uma rede neural no SNNS através das quatro funções de pruning disponíveis. Esta ferramenta é acionada através do botão PRUNING do MANAGER PANEL. Há toda uma teoria relacionada com esses algoritmos que não será enfocada por este manual. Apenas será visto o uso de um dos algoritmos para rede Feedforward. O algoritmo pode ser adequado às necessidades do usuário através dos parâmetros do PRUNING PANEL. O PRUNING PANEL é mostrado a seguir:

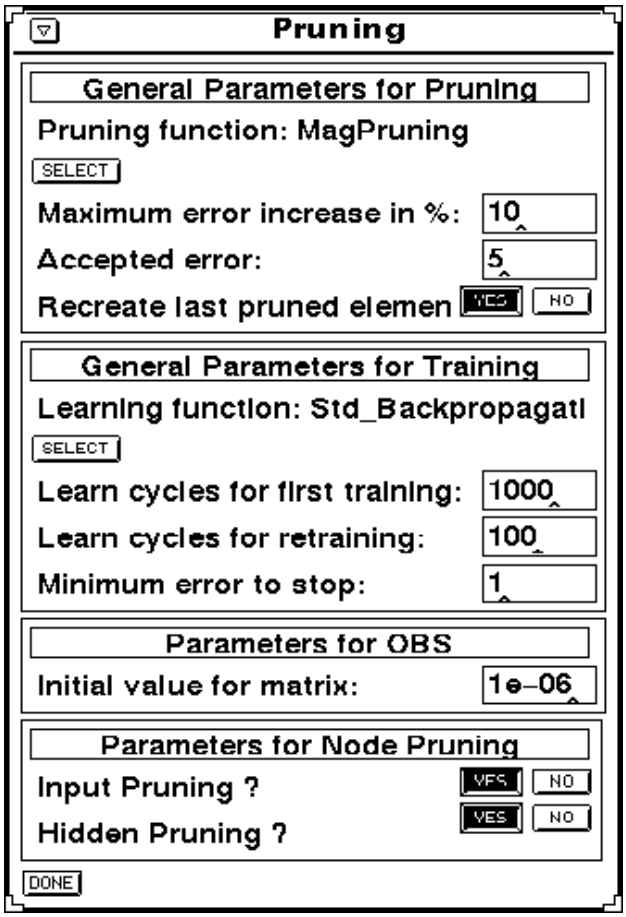


Figura 12 - PRUNING PANEL.

E consiste de:

I. Um bloco para controlar a função de pruning. Este bloco possui:
- ♦ Um botão SELECT para escolher a função de pruning mostrada em PRUNING FUNCTION
- ♦ Dois campos texto MAXIMUM ERROR INCREASE IN % e ACCEPTED ERROR, que são os critério de parada do pruning.
- ♦ Dois botões RECREATE LAST PRUNED ELEMENT ? que possibilita restaurar o estado da rede antes do último pruning (YES) ou não (NO).

II. Um bloco que permite selecionar a função de aprendizagem. Este bloco possui:
- ♦ Um botão SELECT para escolher a função de aprendizagem mostrada em LEARNING FUNCTION
- ♦ Um campo LEARN CYCLE FOR FIRST TRAINING que determina a quantidade de ciclos para o primeiro treinamento.
- ♦ Um campo LEARN CYCLE FOR FIRST RETRAINING que determina a quantidade de ciclos para o segundo treinamento separadamente.
- ♦ Um campo MINIMUM ERROR TO STOP que bloqueia o treinamento quando o erro absoluto cai abaixo dele.

III. Um bloco para manipular os parâmetros para OBS e para os nós que sofrerão pruning. Esse bloco é formado por:
- ♦ Um campo INITIAL VALUE FOR MATRIX que determina o valor inicial da diagonal principal da Hesse-Matrix.
- ♦ Dois botões INPUT PRUNING ? que seleciona (YES) ou não (NO) os neurônios da camada de entrada para sofrer pruning.
- ♦ Dois botões HIDDEN PRUNING ? que seleciona (YES) ou não (NO) os neurônios da camada de escondida para sofrer pruning.

IV. Um botão DONE que inicia o pruning.

3.9. ANALISANDO PROJEÇÃO EM NEURÔNIOS DE UMA REDE NEURAL

O **Projection Panel** é uma ferramenta de análise que permite mostrar como a saída de uma unidade (camada escondida ou de saída) depende de duas unidades de entrada. Isto é feito através de uma projeção dos eixos dos vetores das duas entradas. Esta ferramenta é acionada através do botão PROJECTION do MANAGER PANEL. O **Projection Panel** é bastante parecido com **Weight Panel**. A figura 13 mostra a janela do Unit Activation Projection, e abaixo segue-se uma descrição de tal ferramenta:

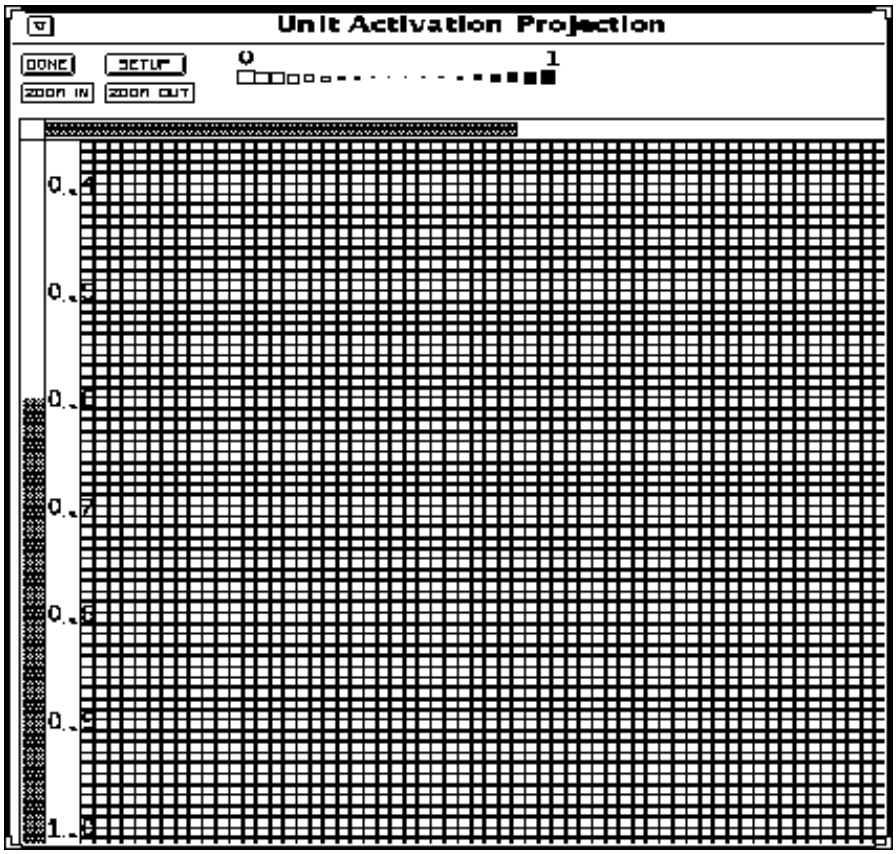


Figura 13 - PROJECTION PANEL.

Que consiste de:

- Um botão SETUP, onde duas unidades devem ser especificadas cujas entradas são variadas sobre o valor de alcance de entrada referentes às coordenas X e Y respectivamente. A terceira unidade a ser especificada é aquela cujo valor de saída determina a cor de ponteiros para os valores de coordenadas X e Y respectivamente.
- Dois botões ZOOM IN e ZOOM OUT que aumentam ou diminuem o quadrado por um píxel em cada lado.
- Uma janela de visualização.
- Um barra de rolagem que permite navegar pela janela.

3.10. IMPRIMINDO UMA REDE NEURAL

Um **2D-Display** pode ser impresso através do **Print Panel**. Esta ferramenta é acionada através do PRINT do botão MANAGER PANEL. O dispositivo de saída pode ser uma impressora ou arquivo. O **Printer Panel** é mostrado a seguir:

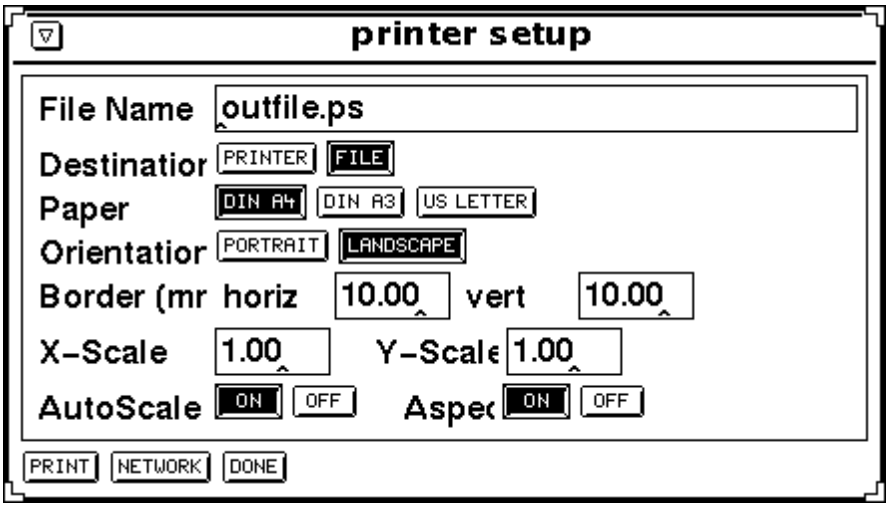


Figura 14 - PRINTER PANEL.

E consiste de:

- Um campo texto COMMAND LINE, onde aparece o comando dado para impressão.
- Uma linha DESTINATION, com os botões PRINTER e FILE, para escolher o destino da impressão.
- Uma linha PAPER com os botões DIN A4, DIN A5, e US LETTER para selecionar o formato do papel.
- Uma linha ORIENTATION com os botões PORTRAIT e LANDSCAPE para selecionar a orientação da visualização no papel.
- Uma linha BORDER (mm) com dois campos HORIZ e VERT para selecionar o tamanho das bordas horizontais e verticais na página em milímetros.
- Uma linha com os campos X-SCALE e Y-SCALE para determinar o fator de escala nas direções X e Y respectivamente. Válidos apenas se o AUTOESCALE está “OFF”.
- Um campo AUTOSCALE, que ligado (botão ON), posiciona a rede no papel com o maior tamanho possível.
- Um campo ASPECT, que ligado(ON), posiciona a rede nas direções X e Y uniformemente.
- Um botão PRINT que inicia a impressão
- Um botão NETWORK que abre um Panel de configuração da rede. Isto permite especificar várias opções para controlar a maneira com que a rede é impressa.

3.11. OBTENDO INFORMAÇÃO SOBRE O SNNS

Várias janelas de help podem ser abertas mostrando diferentes auxílios sobre diferentes partes do simulador. Para tal, basta acionar o botão HELP do MANAGER PANEL. Feito isso, aparecerá a janela **Help** mostrada a seguir:

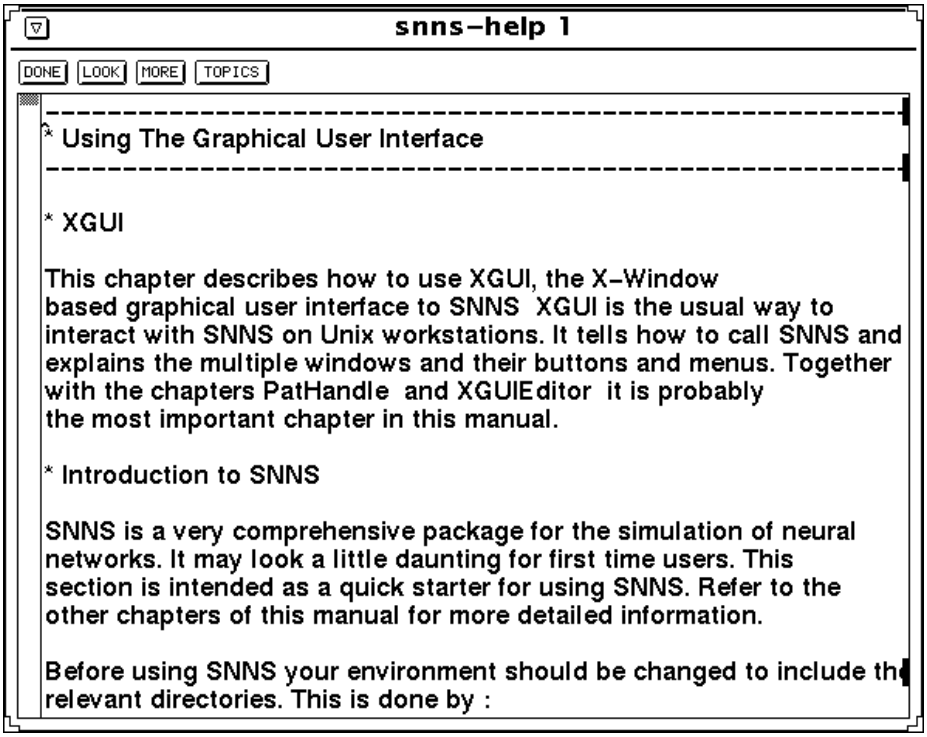


Figura 15 - HELP WINDOW.

Que consiste de:

- Um botão LOOK, que faz o SNNS buscar a primeira ocorrência de uma palavra à partir do início do texto.
- Um botão MORE, que faz o SNNS buscar a primeira ocorrência de uma palavra à partir da última posição visitada pelo chamado de uma função de help.
- Um botão TOPICS, que fornece uma lista de palavras chaves relevantes.
- Uma janela de texto, onde aparece o texto de help corrente.
- Uma barra de rolagem para navegar ao longo do documento.

3.12. OUTRAS FUNCIONALIDADES DA XGUI

O XGUI possui outras opções que não serão apresentadas com detalhes, pois o objetivo deste manual é apenas dar um visão geral das principais funções do simulador. Informações detalhadas sobre essas opções são encontradas no manual original. Mesmo assim, segue uma lista delas:

CASCADE: painel para controle das fases de aprendizagem da rede Cascade Correlation.
KOHONEN: extensão do **Control Panel** para redes de Kohonen.
ANALYZER: para supervisionar redes recursivas e outras.

INVERSION: display para controlar a ferramenta de análise de redes com método de inversão

4. DADOS DE UMA REDE NEURAL

O SNNS consegue manipular conjuntos de dados de tamanhos fixos e variados. Este manual apenas apresentará os símbolos terminais e a gramática para editar dados de tamanho fixo, além de um exemplo. Maiores detalhes sobre os tipos de dados são encontrados no manual original.

4.1. CONVENÇÕES:

Os elementos léxicos da gramática - para dados utilizados no SNNS - são expressões regulares que seguem a convenção abaixo:

- todos os terminais são postos entre aspas.
- elementos de conjuntos são colocados entre colchetes. Dentro dos colchetes, caracteres definem os limites do conjunto. Ex.: dígito é definido como [“0” - “9”]
- caracteres podem ser combinados em grupos utilizando parênteses ( ).
- x* significa que o caracter x ocorre zero ou mais vezes.
- x+ significa que o caracter x ocorre pelo menos de uma vez.
- x{n} significa que x pode ser omitido.
- x | y significa que x ou y tem que ocorrer.
- *, + e {}bind strongest, ? é a segunda, | bind weakest.
- grupos ou classes de caracteres são tratados com um caracter simples em relação à prioridade.

A gramática é listada em uma forma especial de EBNF:

- partes entre colchetes são facultativas
- | separa alternativas
- {x} significa que ocorre zero ou mais vezes
- CSTRING é qualquer coisa reconhecido como string na linguagem de programação C.

4.2. SÍMBOLO TERMINAIS:

WHITE	{“ ” | “\t”}
FREE	{^”\n”} // algo além do EOL
COMMENT	“#”FREE “\n”
L_BRACKET	“[”
R_BRACKET	“]”
V_NUMBER	[Vv]{INT}+“.”{INT}+
INT	[“0”- “9”]
NUMBER	[-+]?{{INT}+ | {INT}+{EXP} | {INT}+ “.” ({EXP})? | {INT}* “.”{INT}+({EXP})?}
EXP	[Ee][-+]?{INT}+
VERSION_HEADER	“SNNS pattern definition file”
GENERATED_AT	“generated at {FREE}*
NO_OF_PATTERN	“No. of patterns” {WHITE}* “:”
NO_OF_INPUT	“No. of input units” {WHITE}* “:”
NO_OF_OUTPUT	“No. of output units” {WHITE}* “:”
NO_OF_IDIM	“No. of variable input dimension” {WHITE}* “:”
NO_OF_VAR_ODIM	“No. of variable output dimension” {WHITE}* “:”
MAXIMUM_IDIM	“Maximum input dimension” {WHITE}* “:”
MAXIMUM_ODIM	“Maximum output dimension” {WHITE}* “:”

4.3. GRAMÁTICA:

```
pattern file      ::= header_pattern_list
header            ::= VERSION_HEADER_ V_NUMBER GENERATED_AT
                      NO_OF_PATTERN NUMBER i_head [o_head] [vi_head] [vo_head]
i_head            ::= [Comment] NO_OF_INPUT NUMBER
o_head            ::= [Comment] NO_OF_OUTPUT NUMBER
vi_head           ::= [Comment] NO_OF_VAR_IDIM NUMBER actual_dim
vo_head           ::= [Comment] NO_OF_VAR_ODIM NUMBER actual_dim
actual_dim        ::= (L_BRACKET actual_dim_rest R_BRACKET) |
                      (L_BRACKET R_BRACKET)
actual_dim_rest   ::= dim_entry | (actual_dim_rest dim_entry)
dim_entry         ::= INT
pattern_list      ::= pattern | (pattern_list pattern)
pattern           ::= pattern_start pattern_body
pattern_start     ::= [Comment] [actual_dim]
pattern_body      ::= patterrn_entry | (pattern_body pattern_entry)
pattern_entry     ::= NUMBER
```

4.4. EXEMPLO:

O exemplo apresentado é o conjunto de padrões para uma rede neural resolver o problema XOR. Este exemplo foi tirado do SNNS e são utilizados quatro padrões para um rede com duas unidade de entrada e uma de saída:

```
SNNS pattern definition file V3.2
generated at Mon Apr 25 15:58:23 1994

No. of  patterns : 4
No. of input units : 2
No. of output units : 1

# Input pattern 1:
0 0
# Output pattern 1:
0
# Input pattern 2:
0 1
# Output pattern 2:
1
# Input pattern 3:
1 0
# Output pattern 3:
1
# Input pattern 4:
1 1
# Output pattern 4:
1
```

5. OPERANDO O SIMULADOR POR BATCH

Devido ao fato de que treinamento de uma rede neural requer muito tempo de CPU, o SNNS oferece a ferramenta **batchman** para desempenhar esta tarefa no módulo batch. Esta nova linguagem substitui o velho **snnsbat** da versão 4.0. A nova linguagem suporta todas as funções necessárias para treinar e testar redes neurais. Todos os aspectos não gráficos oferecidos pelo XGUI podem ser conseguido com o auxilio da linguagem.

A nova linguagem possui várias vantagens:

- permite ao usuário obter os resultados desejados sem gastar muito tempo aprendendo estruturas estatísticas.
- permite a utilização de muitos operadores.
- as variáveis não tem que ser declaradas antes de usadas.
- apresenta mensagens de erros.

5.1. CHAMANDO O INTERPRETADOR BATCH

O interpretador pode ser usado em modo interativo ou como um arquivo contendo o programa batch. Quando usado por um arquivo, nenhuma entrada do teclado é necessária.

O modo interativo pode ser iniciado da seguinte maneira:

/Unix>batchman

produzindo:

SNNS Batch Interpreter V1.0 Type batchman -h for help
No input file specified, reading input from stdin
batchman>

e podendo receber as instruções do usuário através do teclado. Quando uma entrada estiver completa, o interpretador pode começar a trabalhar com Crtl-D. Crtrl-C interrompe simulador.

Se o usuário decidir usar o modo por arquivo, deve-se ser incluído a opção -f e o nome do arquivo junto com o nome do interpretador:

/Unix> batchman -f myprog.bat

fazendo o interpretador executar as instruções do programa contido em myprog.bat.

A saída padrão é geralmente a tela. Contudo, com a opção -l na linha de comando, a saída pode ser direcionada para um arquivo texto tanto no modo interativo:

/Unix> batchman -l logfile

quanto no modo de arquivo:

/Unix> batchman -f myprog.bat -l logfile

As outras opções possíveis são:

- -p: programa deve ser analisado sintaticamente mas não executado.
- -q: nenhuma mensagem deve ser mostrada exceto as causadas pela função print().
- -s: nenhuma warning deve ser mostrada.

- -h: mensagens de auxílio devem ser mostradas.

5.2. LINGUAGEM DO INTERPRETADOR

Um programa batch não possui estrutura pré-definida. Não há seção de declaração das variáveis. Todas as instruções são especificadas no programa de acordo com a sua ordem de execução. Instruções simples em uma linha não precisam ser finalizadas por um ponto-vírgula. O caracter de fim de linha (Ctrl-D) separa duas diferentes instruções em diferentes linhas. Palavras chaves que determinam o fim de um bloco (endwhile, endif, endfor, until e else), não precisam terminar em ponto e vírgula. Múltiplos pontos e vírgulas são possíveis entre duas instruções. Entretanto, se houver mais que duas instruções em um linha, o ponto e vírgula é necessário. Comentários, no código fonte, começam com o caracter # sendo válidos até o fim da linha.

Em um programa batch podem haver: *tipos de dados*, *variáveis*, *operadores e expressões*, *estruturas de controle* e *funções do SNNS*. Cada um desses elementos será discutido a seguir:

5.2.1 Tipos de Dados:

Os tipos de dados reconhecidos pelo interpretador são:

- Números Inteiros
- Números Reais de ponto flutuante
- Tipo Booleano (TRUE e FALSE)
- Strings (limitadas por aspas duplas)

5.2.2 Variáveis:

As variáveis são criadas pelo interpretador quando são usadas pela primeira vez. O nome de uma variável consiste de uma letra ou underscore seguido de letras, dígitos ou underscores. O interpretador faz distinção entre letras maiúsculas e minúsculas. O tipo de uma variável é determinado no momento de sua criação.

Ex.: a = 5;	tipo inteiro, valor 5
file_name = “first.net”	tipo sting, valor “first.net”

Existem algumas variáveis de sistema pré-definidas, as quais o usuário pode apenas ler os seus valores. São elas:

- **SSE**: somatório da diferença quadrática de todas as unidades de saída.
- **MSE**: SSE dividido pelo número de padrões.
- **SSEPU**: SSE dividido pelo número de unidades de saída da rede.
- **CYCLES**: número atual de ciclos de treinamento.

- **PAT**: número do padrão no conjunto de padrões corrente.
- **EXIT_CODE**: status de saída após uma execução.

5.2.3 Expressões e Operadores

Uma expressão é uma fórmula matemática que pode ser atribuída a uma variável. Elas são avaliadas da esquerda para direita. Seus tipos são determinados durante o tempo de execução. Os operadores que podem aparecer em uma expressão são apresentados a seguir, ordenados por prioridade de cima para baixo:

Operador	Função
+,-	Sinal para número
not, !	Negação lógica de valores Booleanos
sqtr	Raiz quadrada
ln	Logaritmo Natural na base e
log	Logaritmo na base 10
**,^	Função de exponenciação
*	Multiplicação
/	Divisão
div	Divisão truncada
mod,%	Resultado depois da divisão pelo mesmo número
+	Adição
-	Subtração
<	Menor
<=,=<	Menor ou igual
>	Maior
>=,=>	Maior ou igual
==	Igual
<>,|=	Diferente
and,&&	Operador conjuntivo “E”
or,||	Operador disjuntivo “OU”

Tabela 4 - Operadore da Linguagem Batch

5.2.4 Função Print

A função print é uma função do interpretador que direciona o seu argumento para a saída padrão, caso nenhum arquivo não seja especificado. Trata-se de uma função com múltiplos argumentos. Se *print* for chamada sem argumentos, uma linha em branco será produzida na saída.

Ex.:	Instrução	Saída
	print(5)	5
	print(3*4)	12

print(“Este texto e:”,1,2,3) Estes texto e:1 2 3

5.2.5 Estruturas de Controle

As estruturas suportadas pela linguagem do interpretador são: *if-then*, *if-then-else*, *for*, *while*, e *repeat*. A utilização das estruturas de controle é semelhante a das linguagens de programação comuns. Suas sintaxes são apresentadas abaixo:

if EXPRESSÃO *then* BLOCO *endif*
if EXPRESSÃO *then* BLOCO *else* BLOCO *endif*
for ATRIBUIÇÃO *to* EXPRESSÃO *do* BLOCO *endfor*
while EXPRESSÃO *do* BLOCO *endwhile*
repeat BLOCO *until* EXPRESSÃO

onde,

BLOCO: seqüência de instrução;
ATRIBUIÇÃO: comando de atribuição;
EXPRESSÃO: qualquer tipo de expressão suportada pelo interpretador.

5.2.6 Chamadas de Função do SNNS

As chamadas de função do SNNS controlam o SNNS kernel e são de quatro categorias: *para seleção de parâmetros do SNNS*, *referentes a redes neurais*, *referentes a padrões* e *funções especiais*. Abaixo segue uma lista de todas a funções, descrevendo suas sintaxes:

5.2.6.1 Chamadas de funções para selecionar parâmetros do SNNS:

- **setInitFunc(function name, parameters ...):** seleciona a função de inicialização e seus parâmetros onde:
  function name: uma das funções de inicialização descritas no XGUI;
  parameters: variam de acordo com a função de inicialização.
- **setLearnFunc(function name, parameters ...):** seleciona a função de aprendizagem e seus parâmetros onde:
  function name: um dos algoritmos de aprendizagem descritos no XGUI;
  parameters: variam de acordo com o algoritmo de aprendizagem.
- **setUpdateFunc(function name, parameters ...):** seleciona a função de atualização e seus parâmetros onde:
  function name: um dos algoritmos de atualização descritos no XGUI;
  parameters: variam de acordo com a função de atualização.
- **setPruningFunc(function name 1, function name 2, parameters):** seleciona a função de pruning e seus parâmetros onde:
  function name 1: uma das funções pruning descritas no XGUI;
  function name 2: uma das funções de aprendizagem descritas no XGUI;

parameters: variam.

- **setSubPattern(InputSize 1, InputStep 1, OutputSize 1, OutputStep 1):** define o esquema de acionamento de subpadrões onde:

  InputSize 1, InputStep 1, OutputSize 1, OutputStep 1: as primeiras dimensões dos subpadrões. Podem haver outras dimensões que necessitam de quatro dimensões para serem definidas.

- **setShuffle(mode):** altera o modo Shuffle onde:

  mode: valor Booleano - TRUE ativa a função Shuffle e FALSE desativa.

- **setSubShuffle(mode):** altera o modo Shuffle de um subpadrão onde:

  mode: valor Booleano - TRUE ativa a função Shuffle e FALSE desativa.

5.2.6.2 Chamadas de função relacionadas às redes

- **loadNet(file_name)**: carrega uma rede onde:

  file_name: arquivo Unix limitado entre aspas “ ”.
  OBS.: Esta chamada de função zera CYCLES.

- **saveNet(file_name, star, end, inclIn, inclOut, filemode)**: salva uma rede onde:

  file_name: arquivo Unix limitado entre aspas “”.
  start: número do primeiro padrão a ser manipulado;
  end: número do último padrão a ser manipulado;
  inclIn: decide se os padrões de entrada são salvos em um result file (TRUE) ou não (FALSE);
  inclOut: decide se os padrões de saídas são salvos em um result file (TRUE) ou não (FALSE);
  file_mode: decide se o result file será criado (TRUE) ou adicionado (FALSE).

- **initNet()**: inicializa uma rede.
- **trainNet()**: treina uma rede.
- **testNet()**: testa uma rede.

5.2.6.3 Chamadas de funções relacionadas a padrões

- **loadPattern(file_name):** carrega um arquivo de padrões onde:
  file_name: arquivo Unix limitado por aspas “”.
  OBS.: faz PAT receber o número de padrões do último arquivo usado.
- **savePattern(file_name):** salva um arquivo de padrões onde:
  file_name: arquivo Unix limitado por aspas “”.
  OBS.: faz PAT receber o número de padrões do último arquivo usado.
- **delPattern(file_name):** apaga um arquivo de padrões onde:
  file_name: arquivo Unix limitado por aspas “”.
  OBS.: faz PAT receber o número de padrões do último arquivo usado.

- **setPattern(file_name):** seleciona um arquivo de padrões como corrente onde,
  file_name: arquivo Unix limitado por aspas “”.

5.2.6.4 Funções especiais

- **pruneNet():** inicia o procedimento de pruning para um rede.
- **execute(instruction, variable 1, variable 2,1 ...):** executa alguma comando da shell do Unix ou programa, onde:
  instruction: instrução ou comando Unix;
  variable 1, variable 2, ...: saídas do programa ou comando.
- **exit(state):** termina o batchman, onde:
  state: inteiro com valor no intervalo de -128 até 127 que determina ausência de erro.
- **setseed(seed):** seleciona um limite para um gerador de números aleatórios.
  seed: limite para gerador de números aleatórios.

5.2.7 Exemplo

Agora será apresentado um típico programa para treinar uma rede:

```
loadNet(“encoder.net”)
loadPattern(“encoder.pat”)
setInitFunc(“Randomize_Weights”,1.0,-1.0)
InitNet()
while SSE > 6.9 and CYCLES < 1000 do
  if (CYCLES mod 10 =0) then
    print(“cycles = ”, CYCLES, “ SSE = ”, SSE)
  endif
  trainNet();
endwhile
saveResult(“encorder.res”, 1, PAT, TRUE, TRUE, “create”)
saveNet(“encoder.trained.net”)
print(“Cycles trained: ”, CYCLES)
print(“Training stopped at error: ”, SSE)
```

6. TRANSFORMANDO UMA REDE NEURAL EM PROGRAMA "C"

É possível compilar um arquivo de rede SNNS (<network>) em uma fonte "C" executável (<C-filename>) através da linha de comando Snns2c. A sintaxe do Snns2c é apresentada a seguir:

snns2c <network> [<C-filename> [<function-name>]]

A rede agora pode ser chamada através do procedimento <function-name>. A rede gerada em "C" não pode mais ser treinada. O objetivo desta ferramenta é integrar uma rede treinada pelo SNNS a uma dada aplicação. O programa gerado também mostra como usar a interface do SNNS kernel para carregar e alterar uma rede em um outro formato. Todos os dados e todas as funções do SNNS - exceto a função de ativação - são salvos como funções "C".

O Snns2c também suporta site e os seguintes tipos de rede: redes feed_forward treinada com Backpropagation e variações, Radial Basis Function, redes de Elman-Jordan parcialmente recorrente, TDNN, DLVQ, BPTT, QPTT, BBPTT, e redes Counterpropagation.

6.1. FLUXO DO PROGRAMA

Os passos da compilação indicados para o usuário são:

1. loading net... o arquivo de rede treinada é carregado.
2. dividing net into layers... todas as unidades são agrupadas em camadas. As unidades de uma camada tem que ter o mesmo tipo e função de ativação.
3. sorting layers... camadas são ordenadas topologicamente.
4. writing net... seleciona as funções de ativação e as escreve para o arquivo fonte "C".

6.2. INCLUINDO UMA REDE COMPILADA EM SUA APLICAÇÃO

Toda rede gerada pode ser chamada como uma função "C" cuja forma é:

int *function_name*(float *in, float *out, int init)

onde:

in e out: ponteiro para o vetor de entrada e saída da rede;
init: flag necessário para alguns tipos de rede

O código gerado pode ser compilado separadamente. Para usar a rede é necessário incluir um arquivo header que também é gerado pelo Snns2c com o mesmo nome da *function_name*. Este arquivo contem os protótipos das funções geradas e um record contendo um número de unidades de entrada e saída.

6.3. FUNÇÕES DE ATIVAÇÃO

Funções de Ativação suportada pelo Snns2c:

Act_Logistic	Act_StepFunc
Act_Elliot	Act_Identify
Act_BSB	Act_IdentifyPlusBias
Act_TanH	Act_RBF_Gaussian
Act_TanHPlusBias	Act_RBF_MuliQuadratic
Act_TanH_Xdiv2	Act_RBF_ThinPlateSpline
Act_Perceptron	Act_TD_Logistic
Act_Signum	Act_TD_Elliot
Act_Signum0	

Tabela 5 - Funções de ativação suportadas pelo Snns2c.

Funções de Ativação incluídas pelo usuário: O arquivo "tools/source/function.h" contêm dois arrays: um com nome das funções (ACT_FUNC_NAMES) e outro para macros que representam funções (ACT_FUNCTIONS). O modo mais fácil do usuário incluir sua própria função de ativação é escrever as duas entradas necessárias na primeira posição do array. Depois disso, a constante "ActRbfNumber" deveria ser adicionada. Se uma nova Radial Basis function for incluída, as entradas seriam adicionadas no final sem incrementar ActRbfNumber. Uma string vazia "" , a última entrada do array Act_FUNC_NAMES, o flag de fim de array.

6.4. Mensagens de Erro

As possíveis mensagens de erro durante a compilação são:
not enough memory: uma locação de memória dinâmica falhou.
can't load file: o arquivo de rede SNNS não foi encontrado.
can't open file: o mesmo significado da mensagem acima ou disco cheio
wrong parameter: tipo ou número de parâmetro errado.
net contains illegal cycles: a rede contém ciclos ilegais.
can't find the function *atcfunc*: a função de ativação não é suportada.
net is not a CounterPropagation network: redes Counterpropagation necessita de uma arquitetura especial: uma entrada, uma saída, uma camada escondida que é completamente conectada.
net is not Time Delay Neural Network: As redes SNNS TDDNNs têm uma arquitetura especial que deve ser editada somente pelo BIGNET.
not supported network type: tipo de rede não suportado pelo Snns2c
unspecified Error: erro indeterminado.
unknown error code: *number*: erro do simulador.

7. CONCLUSÃO

O SNNS é uma ferramenta eficiente para auxiliar a criação, treinamento e manutenção de redes neurais. Possui também facilidades para manipulação de arquivos de dados, de resultados, etc. O simulador funciona muito bem neste contexto. Contudo, quando se deseja implementar algo mais sofisticado com redes neurais, os recursos XGUI não são suficientes. Passa a ser necessário o uso de outras ferramentas presentes no pacote do SNNS.

A criação e edição de redes no simulador é muito simples e rápida, possibilitando trabalhar com um grande número de arquiteturas distintas. Também é possível utilizar algoritmo de pruning para otimizar redes treinadas.

O SNNS possui um grande número de algoritmos de aprendizagem. Entre esses algoritmos, estão presentes diversas variações de Backpropagation. É fácil acompanhar a evolução tanto do erro de aprendizagem quanto de validação, através de gráficos, durante este processo.

A manutenção de redes e arquivos de dados, resultados, etc... é uma atividade extremamente simples no SNNS. É possível gravar as redes ensinadas para arquivos e mais tarde reutilizá-las. Para auxiliar o monitoramento das redes, existem arquivos que recebem todas as informações geradas durante o processo de aprendizagem.

Por outro lado, é impossível realizar simulações muito sofisticadas utilizando apenas a interface gráfica do simulador. Opções como aprendizagem com parâmetros variáveis, critério de parada por validação, etc. são possíveis apenas através da utilização de ferramentas com interface orientada por comando como o **Batchman.**

APÊNDICE A

Parâmetros das Funções de Aprendizagem

A seguir serão apresentados os parâmetros utilizados pelas funções de aprendizado fornecidas pelo SNNS. Serão analisadas em maiores detalhes as funções mais utilizadas:

- **Std_Backpropagation, BackpropBatch e TimeDelayBackprop**

1. η: parâmetro de aprendizado, específica o tamanho do passo do gradiente descendente.

Valores típicos: 0.1 ... 1.0;

2. d_{max}: a diferença máxima ($d_j = t_j - o_j$) entre o valor de aprendizado t_j e uma saída o_j de uma unidade de saída, que é tolerada. Se valores acima de 0.9 devem ser considerados como 1 e valores abaixo de 0.1 como 0, então d_{max} deve ser 0.1. O que previne o overtraining da rede.
   Valores típicos: 0, 0.1 ou 0.2.

- **BackpropMomentum**

1. η: parâmetro de aprendizado, especifica o tamanho do passo do gradiente descendente.

Valores típicos: 0.1 ... 1.0;

2. μ: termo momentum, especifica a quantidade de mudanças de pesos anteriores (relativo a 1) que é adicionada a mudança atual.

Valores típicos: 0 ... 1.0;

3. c: valor de eliminação de superfícies planas, uma constante que é adicionada a derivada da função de ativação qie permite à rede ultrapassar superfícies planas na superfície do erro.

Valores típicos: 0 ... 0.25, o mais amplamente utilizado é 0.1;

4. d_{max}: a diferença máxima ($d_j = t_j - o_j$) entre o valor de aprendizado t_j e uma saída o_j de uma unidade de saída, que é tolerada. Vide acima.
   Valores típicos: 0, 0.1 ou 0.2.

- **Quickprop**

1. η: parâmetro de aprendizado, especifica o tamanho do passo do gradiente descendente.

Valores típicos, para o Quickprop: 0.1 ... 0.3;

2. μ: parâmetro de crescimento máximo, especifica a quantidade máxima de mudanças de pesos (relativo a 1) que é adicionada a mudança atual.

Valores típicos: 1.75 ... 2.25;

3. ν: termo de decadência de pesos para a diminuíção dos pesos.

Valor típico: 0.0001;

4. ***d_{max}***: a diferença máxima ($d_j = t_j - o_j$) entre o valor de aprendizado t_j e uma saída o_j de uma unidade de saída, que é tolerada. Vide acima.
   Valores típicos: 0, 0.1 ou 0.2.

- **RPROP (Resalient Propagation)**

1. ***$delta_0$***: valores iniciais para todos os Δ_{ij}.
O valor padrão é 0.1;

2. ***$delta_{max}$***: o limite superior para atualização dos valores de Δ_{ij}.
O valor padrão é 50.0;

3. α : a diminuição de pesos determina a relação entre o erro de saída e a redução no tamanho dos pesos.
   Importante: este parâmetro denota um expoente.

- **Kohonen**

1. ***h(0)***: Adaptação de altura. A adaptação de altura inicial pode variar entre 0 e 1. Ela determina a força global de adaptação;

2. ***r(0)***: Adaptação de raio. A adaptação de raio inicial é o raio de vizinhaça da unidade vencedora. Todas as unidades dentro deste raio são adaptadas. Os valores devem estar entre 1 e o tamanho do mapa;

3. ***mult_H***: Fator de diminuição. A adaptação de altura diminuí monotonicamente depois da apresentação de todo padrão de aprendizado. Esta diminuição é controlada pelo fator de diminuição mult_H: h(t+1) := h(t) . mult_H;

4. ***mult_R***: Fator de diminuição. A adaptação de raio também diminuí monotonicamente depois da apresentação de todo padrão de aprendizado. Esta segunda diminuição é controlada pelo fator de diminuição mult_R: r(t+1) := r(t) * mult_R;

5. ***h***: Tamanho horizontal. Como a representação interna da rede não permite determinar a 2ª dimensional do grid, o tamanho horizontal nas unidades deve ser fornecido para a função de aprendizagem. É o mesmo valor utilizado na criação da rede.

- **BackPropWeightDecay**

1. η: taxa de aprendizagem;
2. μ: termo momento;
3. ***d_{max}*** : limitante superior do erro de aprendizagem $d_i=t_i-o_i$.

- **BackPropThroughTime(BPTT), BatchBackpropThroughTime(BBPTT)**

1. η: parâmetro de aprendizado;
2. μ: termo momentum;
3. ***backstep***: número de passo para trás do backprop no tempo.

- **QuickpropThroughTime**

1. η: parâmetro de aprendizado;
2. μ: termo momentum;
3. ν: termo de diminuição de peso;
4. ***backstep***: número de passo para trás do quickprop no tempo.

- **CounterPropagation**

1. α: parâmetro de aprendizado da camada de Kohonen;
2. β: parâmetro de aprendizado da camada de Grossberg;
3. θ: threshold de uma unidade.

- **BackPercolation 1**

1. λ: visualização do erro global;
2. θ: threshold de uma unidade;
3. d_{max+}: limitante superior do erro de aprendizagem $d_i=t_i-o_i$.

- **Dynamic Learning Vector Quantization (DLVQ)**

1. $\eta+$: taxa de aprendizagem que descreve o tamanho do μ_a que é mais próximo ao padrão x_a;
2. $\eta-$: taxa de aprendizagem que descreve o tamanho do μ_b que é mais próximo ao padrão x_b;
3. ***número*** de ciclos que se deseja treinar a rede antes dos vetores serem calculados.

- **RadialBasisLearning**

1. ***centers***: determina taxa de aprendizagem η_1, usada para modificação do centro dos vetores;
2. ***bias(p)***: determina taxa de aprendizagem η_2, usada para modificação dos parâmetros da função base;
3. ***delta max***: se o erro é menor do que o erro máximo permitido, os pesos correspondentes não serão alterados;
4. μ: termo momento.

- **RadialBasisLearning with Dynamic Decay Adjust**

1. $\theta+$: threshold positivo;
2. $\theta-$: threshold negativo;
3. ***número*** máximo de unidades RBF mostrada em uma linha.

- **ART1**

1. ρ: parâmetro de vigilância.

- **ART2**

1. ρ: parâmetro de vigilância;
2. ***influência*** do nível mais baixo em F1 exercida pelo nível médio;
3. ***influência*** do nível médio em F1 exercida pelo nível mais alto;
4. ***parte*** do comprimento do vetor p usada para computar o erro;
5. θ: threshold para a função de saída f das unidades x_i e q_i.

- **ARTMAP**

1. ρa: parâmetro de vigilância para sub-rede ARTa;
2. ρb: parâmetro de vigilância para sub-rede ARTb;
3. ρar: parâmetro de vigilância para o controle de reset da rede ART.

- **Backprop em Cascade Correlation (CC) e Recursive Cascade Correlation (RCC)**

1. η_1: parâmetro de aprendizado que especifica a dimensão do gradiente descendente para minimizar o erro da rede;
2. μ_1: termo momentum que especifica a quantidade de alteração do peso antigo que é adicionada à alteração atual;
3. c: valor de eliminação de superfícies planas, uma constante que é adicionada a derivada da função de ativação que permite à rede ultrapassar superfícies planas na superfície de erros;
4. η_2: parâmetro de aprendizado que especifica a dimensão do gradiente descendente para maximizar a covariância;
5. μ_2: termo momentum que especifica a quantidade de alteração do peso antigo que é adicionada a alteração atual.

- **RPROP em Cascade Correlation (CC) e Recursive Cascade Correlation (RCC)**

1. η_1-: especifica o fator pela qual o valor de atualização Δ_{ij} é diminuído quando se minimiza o erro da rede;
2. η_1+: especifica o fator pela qual o valor de atualização Δ_{ij} é aumentado quando se minimiza o erro da rede;
3. η_2-: especifica o fator pela qual o valor de atualização Δ_{ij} é diminuído quando se maximiza a covariância;
4. η_2+: especifica o fator pela qual o valor de atualização Δ_{ij} é aumentado quando se maximiza a covariância.

- **Quickprop em Cascade Correlation (CC) e Recursive Cascade Correlation (RCC)**

1. η_1-: parâmetro de aprendizado quando se minimiza o erro da rede;
2. μ_1: parâmetro de crescimento máximo (tipo de termo momento dinâmico);
3. η_2: taxa de aprendizagem quando se maximiza a covariância;
4. μ_2: parâmetro de crescimento máximo (tipo de termo momento dinâmico).

- **RM_delta (Rumelhart e McClelland's delta rule)**

1. η: parâmetro de aprendizado;
2. ***Ncycles***: número de ciclos de atualização, que especifica quanto tempo um padrão é propagado na rede antes da regra de aprendizagem ser aplicada.

- **Hebbian Learning**

1. η: parâmetro de aprendizado;

2. ***Wmax***: valor absoluto máximo de peso permitido na rede;
3. ***count***: quantidade de tempo que a rede é atualizada antes de calcular o erro.

APÊNDICE B

Exemplo de Aplicação

A seguir será apresentado um exemplo de aplicação de uma rede neural MLP para um problema prático, serão ilustrados todos os passos do desenvolvimento dessa aplicação: criação dos conjuntos de dados, da rede neural, do programa Batchman e de programas auxiliares, treinamento da rede e geração de arquivos resultantes reportando o andamento do treinamento e estatísticas de performance da rede.

O primeiro passo é a obtenção ou criação dos conjuntos de dados necessários para o treinamento da rede neural. Estes dados podem ser coletados de bases de dados existentes, estes devem então ser normalizados e convertidos para o formato do simulador. Geralmente estes dados são separados em três conjuntos: treinamento, validação e teste.

Construídos os conjuntos de dados, passa-se para a fase de construção da rede neural, no SNNS existem duas ferramentas para esta tarefa: a BigNet da interface XGUI e a BigNet da interface kernel. A rede a ser construída deve conter na camada de entrada um número de unidades igual ao número de atributos dos padrões de entrada; de forma semelhante, a camada de saída deve conter um número de unidades igual ao número de atributos dos padrões de saída; já para as camadas intermediárias, tanto o número de unidades quanto o número de camadas é variável, não existem metodologias claras para essa determinação, podem feitos vários testes gerando e testando cada rede. Recentemente, estão sendo utilizados diversos métodos de busca e otimização, entre eles Algoritmos Genéticos.

O próximo passo é a definição de uma estratégia de aprendizado, e construção de um programa Batchman, como o ilustrado abaixo. Este programa é organizado em três partes principais:

1. *Definição dos parâmetros e funções de treinamento*

Nesta primeira parte são definidas todos os parâmetros do processo de aprendizado tais como: a versão do processo, a rede a ser treinada, os arquivos dos conjuntos de dados, o número de passos a cada validação, o número de passos a cada impressão dos erros de treinamento, o número de ciclos de treinamento, a função de aprendizado utilizada e seus parâmetros, e a função de inicialização dos pesos da rede e os parâmetros dessa função. Todas as estas definições são colocadas nesta primeira parte de modo a facilitar modificações nos parâmetros. São utilizados vários *prints* para mostrar no arquivo resultante (.log) quais foram os parâmetros utilizados, na determinada versão do treinamento.

2. *Treinamento*

Nesta parte é definida a estratégia de aprendizado. Aqui foi utilizada a seguinte estratégia: treinar por um grande número de ciclos, e gravar o estado da rede no ponto em que forem apresentados os menores erros no conjunto de validação, nesse ponto a rede terá grande capacidade de generalização. O processo de aprendizado é realizado da seguinte maneira: Primeiro é feita a inicialização da rede, a cada ciclo a rede é

treinada, a cada passo de validação a rede é testada para o conjunto de validação e se o erro apresentado for o menor até o momento a rede é gravada e a cada passo de impressão é mostrado o erro quadrático médio no conjunto de treinamento e validação.

3. *Definição dos parâmetros e funções de treinamento*

Nesta última parte são feitos os testes com a rede gerada. A rede é testada para os conjuntos de treinamento, validação e um outro conjunto ainda não visto até então, o conjunto de teste, os erros quadráticos médios para cada dos conjuntos é reportado no arquivo resultante (.log); os arquivos de resultados (.res) para cada um dos conjuntos são gerados estes arquivos 'ascii' contém os padrões de saída as respectivas saídas da rede; em seguida esses arquivos são analisados através da ferramenta analyze do SNNS que gera as estatísticas de classificação dos padrões (totais e por classe), esta saída é passada ao comando analyze.gawk, ilustrado posteriormente, que passará estas estatísticas em forma de parâmetros; assim as estatísticas para cada conjunto de dados são reportadas no arquivo resultante.

Veja o Programa Batchman abaixo:

```
############# Batchman Program generated by: BatchGen #################
##############################################################################
#####   autor: Elson Felix Mendes Filho    #    e-mail: prico@icmsc.sc.usp.br   #####
#####################      LabIC - ICMSC - USP      #################
##############################################################################

# definition of the version:
version := "g1"

# definition of the topology:
topol := "10-20-5"

# definition of the data sets:
set_tre := "../Data_bck/treina.pat"
set_val := "../Data_bck/valida.pat"
set_tes := "../Data_bck/testa.pat"

# Load the untrained net:
loadNet("38-10-20-5-2nt.net")

# Load data sets used in the training:
loadPattern(set_tre)
loadPattern(set_val)
setPattern(set_tre)

# definition of the process parameters:
val_step := 10
print_step := 50
end_cycle := 200

# definition of the training parameters:
neta := 0.0025
mi := 0.0025
```

```
c := 0.1
dmax := 0.1

# definition of the Learning Function:
setLearnFunc("BackpropMomentum",neta,mi,c,dmax)
setUpdateFunc("Topological_Order")
setShuffle(TRUE)

# definition the Initializing Function:
setInitFunc("Randomize_Weights",0.5,-0.5)

# Shows the initial time, training parameters and functions:
print("############# Batchman Training generated by: BatchGen #################")
print("##########################################################################")
print("#####   autor: Elson Felix Mendes Filho    ##    e-mail: prico@icmsc.sc.usp.br   ######")
print("##########################################################################")
print(" Training version: ", version)
print(" ")
execute("date", dweek, month, dmonth, hour, est, year)
print("   Date: ", month, " ", dmonth," ", year)
print("   Initial training time: ", hour)
print(" ")
print(" Neural Network:")
print("   Topology: 38-", topol,"-2" )
print("   BackpropMomentum")
print("      neta: ",neta, "  mi: ",mi,"  c: ",c," dmax: ",dmax)
print("   Initialization: Randomize_Weights [-0.5, 0.5]")
print("   Cycles: ", end_cycle)
print("   Data sets:")
print("      Training:   ",set_tre)
print("      Validation: ",set_val)
print("      Test:       ",set_tes)
print("*******************************************************************************")
print(" Errors during training (MSE) : ")

# Initializes the training and process parameters:
initNet()
saved = FALSE
best_val := 1
best_cycle := 0

# Begins the training:
while CYCLES < end_cycle do
  trainNet()                  # Trains the net.
  errorl := MSE               # Computes the error on the training set.
  if CYCLES mod val_step == 0 then  # makes the validation at validation step
            setPattern(set_val)
            testNet()
            errorv:=MSE
            setPattern(set_tre)
            if errorv < best_val then        # If this is the smallest error
                  best_cycle := CYCLES             # save the net and cycle
                  best_val := errorv
                  best_tre := errorl
                  saveNet("38-10-20-5-2t")         # save the net
                  saved := TRUE
```

```
            endif
  endif
  if CYCLES mod print_step == 0 then  # prints on the print step
            print(" ")
            print("   Cycle = ", CYCLES, "  -   training = ",errorl,"   validation = ",errorv)
  endif
endwhile          #end of training

# Shows the time of the end of the training:
print(" ")
print("*******************************************************************************")
execute("date", dweek, month, dmonth, hour, est, year)
print("   Final training time: ", hour)
print("*******************************************************************************")

# Loads the net in the best point and saves the result files:
loadNet("38-10-20-5-2t")

# Training
setPattern(set_tre)
testNet()
arq_res_tre := "tre"+version+".res"
saveResult(arq_res_tre,1,PAT,FALSE,TRUE,"create")
delPattern(set_tre)

# validation
setPattern(set_val)
testNet()
arq_res_val := "val"+version+".res"
saveResult(arq_res_val,1,PAT,FALSE,TRUE,"create")
delPattern(set_val)

# test
loadPattern(set_tes)
setPattern(set_tes)
testNet()
best_tes = MSE
arq_res_tes := "tes"+version+".res"
saveResult(arq_res_tes,1,PAT,FALSE,TRUE,"create")
delPattern(set_tes)

# Prints the results in the best point
print("Results in the best point: ")
print("------------------------- ")
print(" Cycle = ", best_cycle, "   training = ",best_tre,"   validation = ",best_val,"   test = ",best_tes)
print(" ")

# Prints the net performance statistics in each data set:
print("Performance: ")
print("------------")
print(" ")

print(" Training set: ")
command := "analyze -sc -e 402040 -l 0.4 -h 0.6 -i tre"+version+".res | ../analyze.gawk"
execute(command, np, pw, nw, pr, nr, pu, nu, nw1, nr1, nu1, nw2, nr2, nu2)
print("  Number of patterns: ",np)
```

```
print("  Wrong:   ",pw," ( ",nw," patterns )")
print("  Right:   ",pr," ( ",nr," patterns )")
print("  Unknown: ",pu," ( ",nu," patterns )")
print(" ")
print("  Class 1: ")
print("   Wrong:   ",nw1," patterns")
print("   Right:   ",nr1," patterns ")
print("   Unknown: ",nu1," patterns ")
print(" ")
print("  Class 2: ")
print("   Wrong:   ",nw2," patterns")
print("   Right:   ",nr2," patterns ")
print("   Unknown: ",nu2," patterns ")
print(" ")
print("------------------------------------------------------------------------------ ")
print(" ")

print(" Validation set: ")
command := "analyze -sc -e 402040 -l 0.4 -h 0.6 -i val"+version+".res | ../analyze.gawk"
execute(command, np, pw, nw, pr, nr, pu, nu, nw1, nr1, nu1, nw2, nr2, nu2)
print("  Number of patterns: ",np)
print("  Wrong:   ",pw," ( ",nw," patterns )")
print("  Right:   ",pr," ( ",nr," patterns )")
print("  Unknown: ",pu," ( ",nu," patterns )")
print(" ")
print("  Class 1: ")
print("   Wrong:   ",nw1," patterns")
print("   Right:   ",nr1," patterns ")
print("   Unknown: ",nu1," patterns ")
print(" ")
print("  Class 2: ")
print("   Wrong:   ",nw2," patterns")
print("   Right:   ",nr2," patterns ")
print("   Unknown: ",nu2," patterns ")
print(" ")
print("------------------------------------------------------------------------------ ")
print(" ")

print(" Test set: ")
command := "analyze -sc -e 402040 -l 0.4 -h 0.6 -i tes"+version+".res | ../analyze.gawk"
print("  Number of patterns: ",np)
print("  Wrong:   ",pw," ( ",nw," patterns )")
print("  Right:   ",pr," ( ",nr," patterns )")
print("  Unknown: ",pu," ( ",nu," patterns )")
print(" ")
print("  Class 1: ")
print("   Wrong:   ",nw1," patterns")
print("   Right:   ",nr1," patterns ")
print("   Unknown: ",nu1," patterns ")
print(" ")
print("  Class 2: ")
print("   Wrong:   ",nw2," patterns")
print("   Right:   ",nr2," patterns ")
print("   Unknown: ",nu2," patterns ")
print(" ")
```

```
# Deleting result files:
command = "rm *.res"
execute(command)

# Shows the time of end of the process:
print("*******************************************************************************")
execute("date", one, two, three, hour)
print("Process end time: ", hour)
```

OBS: Como este programa consome muito tempo, a melhor coisa a fazer é rodá-lo como *background job* em horários de baixa utilização do sistema. Para tanto, utilize o comando:

```
echo 'batchman -f -q <seu arquivo bat> -l <seu arquivo log>'| at <hora:minuto>
```

por exemplo:

```
echo 'batchman -q -f progG1.bat -l G1log -q' | at <04:20>
```

Veja abaixo um arquivo resultante do programa Batchman acima:

```
############# Batchman Training generated by: BatchGen ################
#########################################################################
#####   autor: Elson Felix Mendes Filho    ##    e-mail: prico@icmsc.sc.usp.br   #####
#########################################################################
 Training version: g1

   Date: Jan 28 1997
   Initial training time: 15:38:17

 Neural Network:
   Topology: 38-10-2
   BackpropMomentum
      neta: 0.025  mi: 0.025  c: 0.1 dmax: 0.1
   Initialization: Randomize_Weights  [-0.5, 0.5]
   Cycles: 200
 Data sets:
   training:   ../Data_bck/treina.pat
   validation: ../Data_bck/valida.pat
   test:       ../Data_bck/testa.pat

*******************************************************************************
 Errors during training (MSE) :

   Cycle = 50  -   training = 0.0860726   validation = 0.0833399

   Cycle = 100  -   training = 0.0659207   validation = 0.04439

   Cycle = 150  -   training = 0.046919   validation = 0.0336403

   Cycle = 200  -   training = 0.0424512   validation = 0.184802

*******************************************************************************
   Final training time: 15:40:48
*******************************************************************************
```

Results in the best point:

Cycle = 190 training = 0.0483413 validation = 0.0112886 test = 0.0110834

Performance:

Training set:
Number of patterns: 354
Wrong: 3.11 (11 patterns)
Right: 96.89 (343 patterns)
Unknown.: 0 (0 patterns)

Class 1:
Wrong: 0 patterns
Right: 177 patterns
Unknown.: 0 patterns

Class 2:
Wrong: 11 patterns
Right: 166 patterns
Unknown.: 0 patterns

Validation set:
Number of patterns: 563
Wrong: 0.36 (2 patterns)
Right: 99.29 (559 patterns)
Unknown.: 0.36 (2 patterns)

Class 1:
Wrong: 0 patterns
Right: 539 patterns
Unknown.: 2 patterns

Class 2:
Wrong: 2 patterns
Right: 20 patterns
Unknown.: 0 patterns

Test set:
Number of patterns: 563
Wrong: 0.53 (3 patterns)
Right: 99.29 (559 patterns)
Unknown.: 0.18 (1 patterns)

Class 1:
Wrong: 0 patterns
Right: 540 patterns
Unknown.: 1 patterns

Class 2:
Wrong: 3 patterns
Right: 19 patterns
Unknown.: 0 patterns

****
Process end time: 15:41:07

A ferramenta analyze, pode ser utilizada, como no programa Batchman acima, para analisar arquivos de resultados (.res) criados pelo SNNS. O arquivo de resultados a ser analisado deve conter a saída desejada e a saída da rede. Esta ferramenta pode ser utilizada na linha de comando ou dentro de um programa Batchman. A sintaxe do analyze é apresentada a seguir:

analyze [- opções]

É possível utilizar, em qualquer ordem, as seguintes opções:

-w número de padrões classificados erradamente;
-r número de padrões classificados corretamente;
-u número de padrões não classificados;
-a o mesmo que -w -r -u;
-v saída prolixa. Cada padrão é apresentado com o resultado de sua classificação.
-s informações estatísticas contendo padrões classificados erradamente, classificados corretamente, não classificados e o erro da rede.
-c o mesmo que -s, mas são mostradas as estatísticas para cada unidade de saída (classe);
-i <arquivo> nome do arquivo de resultados a ser analisado;
-o <arquivo> nome do arquivo a ser produzido pelo analyze;
-e <função> define o nome da função de analise. Estas funções podem ser: 402040, WTA e band;
-l <real> primeiro parâmetro da função de análise;
-h <real> segundo parâmetro da função de análise.

Como pode ser verificado esta ferramenta apresenta sua saída no dispositivo de saída padrão ou em arquivos de saída. Se for desejado incluir essa saída em arquivos resultantes de programas Batchman é necessário passar essa saída como parâmetro ao programa Batchman, isso pode ser feito através de simples programas awk. O exemplo abaixo, mostra como utilizar o analyze para obter estatísticas, gerais e por classe, dentro de um trecho de programa Batchman e os programas awk necessários a passagem dos parâmetros: np, pw, nw, pr, nr, pu, nu, nw1, nr1, nu1, nw2, nr2 e nu2.

trecho Batchman:

```
setPattern(set_val)
testNet()
arq_res = "val.res"
saveResult(arq_res,1,PAT,FALSE,TRUE,"create")
command = "analyze -sc -e 402040 -l 0.4 -h 0.6 -i va.res | analyze.gawk"
execute(command, np, pw, nw, pr, nr, pu, nu, nw1, nr1, nu1, nw2, nr2, nu2)
```

analyze.gawk

```
tr -d "\n" | awk -f ana.awk
```

ana.awk

{print $3, $7, $10, $14, $17, $21, $24, $39, $46, $53, $66, $73, $80}

APÊNDICE C

Snns na Internet

Neste apêndice serão mostrados alguns endereços da internet onde se podem encontrar informações úteis sobre o Snns.

Manuais do SNNS versão 4.1

ftp://ftp.informatik.uni-stuttgart.de/pub/SNNS/SNNSv4.1.Manual.ps.Z
The SNNS User Manual (Zipped ps file 1.5 MB) - Manual original.

ftp://ftp.informatik.uni-stuttgart.de/pub/SNNS/SNNS-4.1-Hypertext-Manual.tar.gz
HTML Manual (Gzipped tarfile, 2.5 MB) - Versão hipertexto do manual original.

ftp://xavante.icmsc.sc.usp.br/pub/snns/man_snns.zip
SNNS Manual de Referência Básica elaborado por membros do LABIC - ICMSC - USP. (.zip (.doc - Word 6.0) em Português).

Lista de discusão sobre o SNNS

Existe uma lista de discusão sobre o SNNS, onde vários pesquisadores trocam informações, dicas e dúvidas sobre o simulador. Você pode se juntar a esta lista mandando um mail para:

listserv@informatik.uni-stuttgart.de
com o corpo:
subscribe snns "Seu nome completo"

as mensagens da lista virão de: snns@informatik.uni-stuttgart.de

Como obter o SNNS

O simulador pode ser obtido via anonymous ftp do host:
ftp.informatik.uni-stuttgart.de (129.69.211.2)
no subdiretorio /pub/SNNS.
os arquivos: SNNSv4.1.tar.Z (2.36 MB) ou SNNSv4.1.tar.gz (1.42 MB)

Home Page do SNNS

Existe uma página oficial sobre o simulador com muitas informações sobre o SNNS, além de uma versão do Manual em html. Esta página pode ser encontrada em vários sites.

http://www.informatik.uni-stuttgart.de/ipvr/bv/projekte/snns/snns.html
http://www-ra.informatik.uni-tuebingen.de/SNNSinfo/snns.html
http://www-ti.informatik.uni-tuebingen.de/~goeppert/SNNSinfo/snns.html
http://paris.mgu.bg/~sasho/SNNSinfo/snns.html
http://fangorn.ci.tuwien.ac.at/docs/services/SNNSinfo/snns.html
http://www.automat.uni-essen.de/html_info/snns.html