Instituto de Ciências Matemáticas de São Carlos

ISSN - 0103-2569

UTILIZAÇÃO DO *SOMPAK*
(SELF-ORGANIZING MAP PACKAGE)

ROSELI A. FRANCELIN
NELSON F. PONCE DE LEON

Nº 43

RELATÓRIOS TÉCNICOS DO ICMSC

São Carlos
Out./1996

Utilização do SOMPAK
(Self-Organizing Map Package)

Roseli A . Francelin
USP - ICMSC - SCE
13560 - São Carlos - SP - Brasil

Nelson F. Ponce de Leon
USP - ICMSC - SCE
13560 - São Carlos - SP - Brasil

Sumário

O presente trabalho apresenta um estudo do pacote SOMPAK, que implementa os algoritmos da rede de Kohonen. É realizada uma breve apresentação deste modelo e em seguida são mostrados os principais procedimentos para se utilizar o pacote. Alguns exemplos são apresentados para ilustrar a utilização dos principais comandos.

1 INTRODUÇÃO

O objetivo deste trabalho é descrever os principais procedimentos necessários para se utilizar o SOMPAK (versão 3.1).

Inicialmente, serão fornecidas algumas noções de redes neurais artificiais, e do modelo de Kohonen, que é implementado neste pacote.

Em seguida serão apresentadas algumas das características deste modelo, e como ele é implementado neste pacote, juntamente com uma breve apresentação do método LVQ, que pode ser usado em conjunto com o SOM.

No capítulo seguinte, serão descritos todos os programas componentes deste pacote, bem como detalhes de sua utilização.

Por fim, será apresentado um exemplo da utilização dos programas, descrevendo todos os passos necessários para se desenvolver uma aplicação.

Deve-se ressaltar que este trabalho serve como uma introdução à utilização do SOMPAK, e não esgota todas as possibilidades do pacote.

1.1 Redes Neurais

Redes neurais artificiais são modelos que tentam obter da máquina um desempenho que "imite" o comportamento humano, através de interconexões de elementos computacionais simples, que "simulam" o sistema nervoso biológico humano. A estrutura básica deste sistema é o neurônio, que desempenha o papel de difusor de impulsos.

O modelo dos neurônios do sistema nervoso foi copiado para uma estrutura computacional, onde cada neurônio se torna um processador e a cada informação trocada entre esses processadores está associado um peso.

Pode-se definir então, redes neurais como sendo um sistema de interconexão de processadores, baseado no sistema nervoso, que atua após um estímulo externo.

Redes neurais têm encontrado uma grande aplicação nas áreas de reconhecimento de padrões e imagens, processamento de sinais, controle, automação, otimização, etc.

Modelos de redes neurais são especificados pela topologia da rede, características dos nós (neurônios), regras de treinamento ou aprendizado. Estas regras, baseadas em valores iniciais dos pesos, indicam como os pesos deverão ser modificados para melhorar o desempenho da rede.

1.2 Self-Organizing maps

Self-Organizing Maps (SOM) correspondem à um método de treinamento não-supervisionado de redes neurais, adaptando o estado interno da rede para modelar as características encontradas nos dados de treinamento. Este método foi desenvolvido por Teuvo Kohonen (professor da Universidade de Helsinki) com o intuito de simular as características de aprendizagem da mente humana, onde em muitos casos é também realizada uma aprendizagem não-supervisionada.

2 VISÃO GERAL DO SOM

Existem muitas versões do SOM. A filosofia básica, no entanto, é muito simples e efetiva, e está implementada pelos procedimentos contidos neste pacote.

O SOM define um mapeamento do espaço de entrada de dados $\Re^n$ em um mapa bi-dimensional de nós. Para cada nó i é associado um vetor $m_i \in \Re^n$. A topografia do mapa pode ser definida como retangular ou hexagonal neste pacote, sendo que o último é mais eficiente

O mapa e a posição do nó vencedor (imagem da entrada) neste mapa poderão então ser visualizados graficamente. Para uma análise mais detalhada, cada plano componente do mapa (os valores numéricos dos componentes dos vetores m_i) pode também ser mostrado separadamente no mesmo formato que o mapa, utilizando-se uma escala de cinzas para ilustrar os valores dos componentes. São fornecidos programas para converter o arquivo do mapa resultante para o formato "encapsulated postcript".

Pode-se dizer que o SOM é a "projeção não-linear" da função densidade dos dados de entrada multi-dimensional em um display bi-dimensional. Seja $x \in \Re^n$, um vetor de dados de entrada. Ele pode ser comparado com todos os vetores m_i, utilizando-se qualquer medida, em várias aplicações, a menor das distâncias Euclidianas é calculada para se definir o nó vencedor, representado por c:

$$\|x - m_c\| = \min_i \{\|x - m_i\|\}, \text{ ou}$$
$$c = \arg\min_i \{\|x - m_i\|\}. \qquad (1)$$

O vetor x é então mapeado no nó c.

Durante a fase de aprendizado, os nós topograficamente próximos no mapa, até uma certa distância, irão se ativar mutuamente para um aprendizado da mesma entrada. Os valores dos componentes de m_i se tornam significativos após um processo de aprendizagem, que procura um limite de convergência após uma sequência de iterações. Aqui, os valores iniciais de $m_i(0)$ podem ser arbitrários (por exemplo randômicos):

$$m_i(t+1) = m_i(t) + h_{ci}(t)[x(t) - m_i(t)], \qquad (2)$$

onde t é a variável discreta representando o tempo, e $h_{ci}(t)$ é o núcleo de vizinhança; uma função definida sobre os pontos do grid. O comprimento médio, e a forma de $h_{ci}(t)$, irão definir a "firmeza" da "superfície elástica" a ser formada.

O próximo passo a ser executado é a calibragem do mapa, com o objetivo de tornar possível a localização de imagens de diferentes itens de entrada. Em muitas aplicações práticas, é conhecido como uma particular entrada deve ser interpretada. Colocando-se um número de conjunto de dados, já analisados anteriormente, e verificando-se onde cada entrada melhor se encaixa no mapa de acordo com a Eq. (1), o mapa ou pelo menos um subconjunto de seus nós podem ser rotulados para delinear um "sistema de coordenadas" de acordo com a interpretação anterior. Assumindo-se que o mapa é contínuo sobre alguma "superfície elástica" hipotética, se torna possível a interpretação de dados desconhecidos por meio de interpolação e extrapolação relativa aos pontos calibrados.

2.1 Learning Vector Quantization (LVQ)

A despeito do fato de o SOM ser um paradigma de aprendizagem não-supervisionado, ele faz também, uso de uma técnica de aprendizagem supervisionada, chamada Learning Vector Quantization. Esta técnica realiza uma melhoria na precisão de reconhecimento em um

mapa já treinado, para otimizar sua performance em diversas circunstâncias. Uma situação típica é quando se deseja adicionar novos vetores de treinamento para melhorar a performance de determinadas vizinhanças encontradas no mapa.

Isto é conseguido selecionando-se vetores de treinamento (x) com classificação conhecidas, e apresentando-os para que a rede examine casos de classificação errônea.. É executada uma comparação para se encontrar o nó vencedor (n_W). O vetor de pesos dos nós vencedores serão então modificados de acordo com o seguinte critério.

Para uma entrada classificada corretamente:

$$n_W(t + 1) = n_W(t) + \eta(t)[x(t) - n_W(t)]$$

Para uma entrada classificada incorretamente:

$$n_W(t + 1) = n_W(t) - \eta(t)[x(t) - n_W(t)]$$

O termo $\eta(t)$ é a taxa de aprendizado, similar a utilizada no treinamento pelo SOM.

O pacote LVQ-PAK implementa os algorítmos necessários para a utilização desta técnica, e pode ser utilizado em conjunto com o SOM-PAK para a obtenção de uma melhor performance. Uma breve descrição dos principais programas contidos no pacote LVQ-PAK é apresentada no Apêndice B.

3 FORMATO DOS ARQUIVOS

Todos os arquivos de dados (vetores de entrada e mapas) são armazenados em arquivos ASCII para facilitar sua edição. Os arquivos que contêm dados de treinamento e dados de teste são similares, e podem ser usados alternadamente.

Os arquivos de dados e mapas possuem um formato similar ao usado no pacote *LVQ_PAK*, portanto os mesmos arquivos podem ser usados em ambos.

3.1 Formato dos Arquivos de Dados

Os arquivos de dados são armazenados no formato ASCII como uma lista de entradas, cada linha sendo reservada para uma amostra vetorial.

A primeira linha do arquivo é reservada para o reconhecimento do status das entradas; nesta versão, é usada para a definição dos seguintes itens (devendo aparecer na ordem indicada):

- Dimensão dos vetores (inteiro, compulsório).
- Dimensão dos vetores (inteiro, compulsório)
- Topologia, *hexa* ou *rect* (string, opcional, case-sensitive).
- Dimensão do mapa na direção *x* (inteiro, opcional).
- Dimensão do mapa na direção *y* (inteiro, opcional).
- Tipo de vizinhança, *bubble* ou *gaussian* (string, opcional, case-sensitive).

Nos arquivos de dados, os itens opcionais são ignorados.

As linhas seguintes consistem de *n* números de ponto-flutuante seguidos por um rótulo de classe (opcional, pode ser qualquer string) e dois qualificadores (opcional, veja abaixo) que determinam o uso de cada entrada nos programas de treinamento. Os arquivos de dados também podem conter linhas de comentários que começam com '#' e serão ignoradas.

Se alguns componentes dos vetores de dados forem desconhecidos (em razão de erro de leitura dos dados, ou algum outro), estes componentes deverão ser marcados com '*x*' (substituindo o valor numérico). Por exemplo, um trecho de um arquivo de dados de dimensão 5 poderia ser o seguinte:

```
1.1  2.0  0.5  4.0  5.5
1.3  6.0   x   2.9   x
1.9  1.5  0.1  0.3   x
```

Quando as distâncias aos vetores são calculadas para a escolha do nó vencedor, e na modificação dos vetores de pesos, os componentes marcados com *x* são ignorados.

Exemplo de arquivo de dados: Considere um arquivo de dados hipotético *exam.dat* que representa tons de cores num formato com 3 componentes. Este arquivo contem 4 amostras, cada uma representada por um vetor de dados tridimensional. (A dimensão dos vetores é dada na primeira linha.) Os rótulos podem ser cadeias de caracteres (strings); neste caso 'amarelo' e 'vermelho' são os nomes das classes.

exam.dat:

```
3
# Primeiro as entradas para amarelo
181.0  196.0  17.0  amarelo
251.0  217.0  49.0  amarelo
# Agora as entradas para vermelho
248.0  119.0 110.0  vermelho
213.0   64.0  87.0  vermelho
```

Cada linha de dados pode ter dois qualificadores opcionais que determinam o uso de cada entrada durante o treinamento. Os qualificadores são usados no seguinte formato

código=valor, sem o uso de espaços entre as partes do qualificador. Os qualificadores opcionais são os seguintes:

- Fator de Intensificação: por exemplo *weight=3*.
  A taxa de treinamento para o correspondente vetor de entrada é multiplicada por esse parâmetro, de modo que os vetores de pesos são atualizados como se esse vetor de entrada fosse repetido 3 vezes durante o treinamento.
- Qualificador fixed-point: por exemplo *fixed=2,5*.
  A unidade do mapa definida pelas coordenadas do parâmetro ($x = 2, y = 5$) são selecionadas ao invés da unidade vencedora durante o treinamento. (Mais a frente, é mostrada a definição de coordenadas sobre o mapa.) Se muitas entradas forem forçadas a lugares determinados, uma orientação direcionada é obtida no mapa.

Os qualificadores opcionais não são usados como default; veja a definição dos parâmetros *-fixed* e *-weights*.

3.2 Formato do arquivo de mapa

Os arquivos de mapa são produzidos pelos programas do SOM_PAK, e o usuário não precisa examiná-lo pessoalmente.

Os vetores de pesos são armazenados no formato ASCII. O formato das entradas é similar aos usados nos arquivos de dados de entrada, exceto que os itens opcionais na primeira linha dos arquivos de dados (topologia, dimensões nos eixos x e y, e tipo de vizinhança) agora são compulsórios. Nos arquivos de mapa é possível incluir vários rótulos para cada entrada.

Exemplo: O arquivo de mapa *code.cod* contém um mapa de vetores tridimensionais, com 3x2 unidades. Este mapa corresponde aos vetores de treinamento do arquivo *exam.dat*.

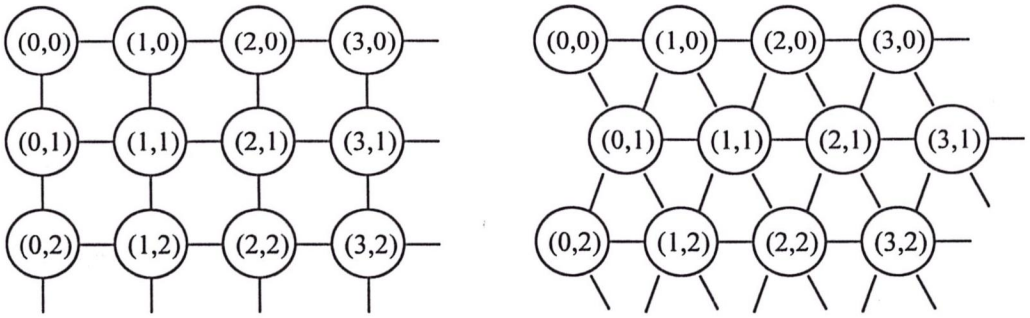
code.cod:

```
3 hexa 3 2 buble
191.105  199.014  21.6269
215.389  156.693  63.8977
242.999  111.141  106.704
241.070  214.011  44.4638
231.183  140.824  67.8754
217.914  71.7228  90.2189
```

Os vetores de pesos (relativos a cada neurônio do mapa) são armazenados no arquivo de mapa na seguinte ordem:

1 Unidade com coordenada (0,0).
2 Unidade com coordenada (1,0).
...
n Unidade com coordenada (*n*-1,0).
n+1 Unidade com coordenada (0,1).
...
nm A última unidade, com coordenada (*n*-1,*m*-1).

onde *n* e *m* são as coordenadas nos eixos *x* e *y*, respectivamente.



Retangular Hexagonal

Na figura acima são mostradas as posições ocupadas pelas unidades em duas estruturas topológicas possíveis. A distância entre duas unidades no mapa é computada como a distância Euclidiana na topologia (bi-dimensional) do mapa.

4 APLICATIVOS DESTE PACOTE

4.1 O programa de busca vfind

O método mais fácil para se utilizar o SOM_PAK é executar o programa *vfind*, que procura por bons mapas, repetindo automaticamente diferentes inicializações e procedimentos de treinamento. O critério para a caracterização de bons mapas é o erro quantizado.

O programa *vfind* pergunta por todos os argumentos necessários interativamente. O usuário precisa apenas executar o programa sem nenhum parâmetro (podem ser fornecidos os seguintes parâmetros: parâmetro verboso (*-v*), tipo de função para a taxa de aprendizagem (*-alpha_type*), tipo de erro quantizado (*-qetype*) e os parâmetros qualificadores (*-fixed* e *-weights*)).

4.2 Usando os programas diretamente

Normalmente, os sub-programas contidos neste pacote são executados diretamente do console usando linhas de comando. O usuário deve tomar cuidado para que os programas sejam executados na ordem correta: primeiramente a inicialização, depois treinamento, e finalmente os testes; e para que os parâmetros sejam fornecidos corretamente (a correspondência entre arquivos de entrada e saída de programas subsequentes é particularmente importante).

Cada programa requer alguns parâmetros: nomes de arquivo, parâmetros de aprendizado, formato dos mapas, etc. Todos eles devem ser fornecidos ao programa logo no início; os programas não são interativos no sentido que eles não pedem pelos parâmetros durante sua execução (Uma lista com a descrição de todos os parâmetros é fornecida no Apêndice A).

4.3 Programas de inicialização

Os programas de inicialização inicializam os vetores de pesos.

- *randinit* - Este programa inicializa os vetores de pesos com valores randômicos. Os componentes dos vetores recebem valores randômicos que são distribuídos uniformemente na área dos componentes dos vetores de dados correspondentes. O tamanho do mapa é dado pela definição da dimensão no eixo x (*-xdim*) e no eixo y (*-ydim*) do mapa. A topologia do mapa é definida com a opção (*-topol*) e pode ser hexagonal (*hexa*) ou retangular (*rect*). A função de vizinhança é definida com a opção (*-neigh*) e pode ser função step (*bubble*) ou Gausiana (*gaussian*).

> randinit -xdim 16 -ydim 12 -din file.dat -cout file.cod -neigh bubble -topol hexa

- *lininit* - Este programa inicializa os vetores de pesos de uma maneira ordenada sobre o espaço bi-dimensional abrangido pelos dois eingenvectors principais dos vetores de dados de entrada.

> lininit -xdim 16 -ydim 12 -din file.dat -cout file.cod -neigh bubble -topos hexa

4.4 Programas de treinamento

vsom - Este programa treina os vetores de pesos usando o algoritmo self-organizing map. O tipo de topologia e a função de vizinhança definidos na fase de inicialização são usados durante o treinamento. O programa procura a unidade vencedora para cada vetor de amostra e atualiza as unidades de sua vizinhança, de acordo com a função de vizinhança selecionada.

O valor inicial da taxa de aprendizado é definido e irá decrescer linearmente a zero até o final do treinamento. O valor inicial do raio de vizinhança também é definido e irá decrescer linearmente a um durante o treinamento (no final, apenas os vizinhos mais próximos serão treinados). Se aos parâmetros qualificadores (*-fixed* e *-weight*) forem fornecidos valores maiores que zero, as definições correspondentes no arquivo de vetores de padrões serão usadas. A função da taxa de aprendizado α pode ser definida usando-se a opção *-alpha_type*. As opções possíveis são *linear* e *inverse_t*. A função linear é definida como $\alpha(t) = \alpha(0)(1.0 - t/rlen)$ e a função do tipo inverse-time como $\alpha(t) = \alpha(0)C/(C + t)$ para computar $\alpha(t)$ para a iteração t. Neste pacote a constante C é definida $C = rlen/100.0$.

> vsom -din file.dat -cin file1.cod -cout file2.cod -rlen 10000 -alpha 0.03 -radius 10 [-fixed 1] [-weights 1] [-alpha_type linear] [-snapinterval 200] [-snapfile file.snap]

Note-se que pode se controlar a colocação de dados em determinada unidade do mapa, alternando-se ciclos de treinamento 'fixados' e 'não-fixados'.

4.5 Programa de avaliação de erro

qerror - O erro quadrático é avaliado. Para cada vetor de amostra, a unidade vencedora no mapa é procurada e a média dos erros quantizados é retornada.

> qerror -din file.dat -cin file.cod [-qetype 1] [-radius 2]

É possível computar o erro quadrático ponderado $\Sigma\, h_{ci}\|x - m_i\|^2$ para cada amostra de entrada e calcular a média sobre os arquivos de dados. Se à opção *-qetype* for fornecido um valor maior que zero, então o erro quadrático ponderado será utilizado. A opção *-radius* pode ser usada para se definir o raio de vizinhança para o cálculo. O valor default é 1.0.

4.6 Programas de monitoração

visual - Este programa gera uma lista de coordenadas correspondentes às unidades vencedoras do mapa para cada amostra de dados do arquivo de dados. Ele também fornece os erros quantizados individuais e os rótulos das classes das unidades vencedoras, se estes tiverem sido definidos. O programa irá armazenar os pontos da imagem tri-dimensional (valor das coordenadas e o erro quantizado) de uma maneira similar ao armazenamento das entradas de dados. Se um vetor de entrada for constituído de apenas componentes ausentes, o programa irá ignorar o vetor. Se a opção -noskip for fornecida, o programa irá indicar a existência desta linha salvando a linha *'-1 -1 -1.0 EMPTY_LINE'* como o resultado.

> *visual -din file.dat -cin file.cod -dout file.vis [-noskip 1]*

planes - Este programa gera um código "encapsulated postscript" (eps) de um plano componente selecionado (especificado pelo parâmetro *-plane*) do mapa, criando uma imagem dos valores dos componentes usando tons de cinza. Se o parâmetro fornecido for zero, então todos os planos serão convertidos. Se o arquivo de dados de entrada também for fornecido, a trajetória formada pelas unidades vencedoras também será convertida em um arquivo separado. O arquivo eps é nomeado utilizando-se a base do nome do arquivo de saída, e adicionando-se *_px.eps* (onde x é substituído pelo índice do plano , começando por um) a ele. O arquivo de trajetória é nomeado adicionando-se *_tr.eps* à base do nome do arquivo. Se a opção *-ps* for fornecida, será gerado um código postscript e os arquivos produzidos serão nomeados com *.ps* ao invés de *.eps*.
No seguinte exemplo, um arquivo chamado *file_p1.eps* é gerado contendo a imagem do plano. Se a opção *-din* for fornecida, outro arquivo *file_tr.eps* será gerado contendo a trajetória. Se a opção *-ps* for fornecida então o arquivo produzido será nomeado *file_p1.ps*.

> *planes -cin file.cod [-plane 1] [-din file.dat] [-ps 1]*

umat - Este programa gera um código encapsulated postscript (eps) para se visualizar as distâncias entre vetores de pesos de unidades vizinhas do mapa usando-se tons de cinza. O arquivo eps é nomeado utilizando-se a base do nome do mapa, e adicionando-se *.eps* a ele.
Se a opção *-average* for fornecida, os tons de cinza da imagem serão filtrados espacialmente pela average, e se a opção *-median* for fornecida, será usado median filtering. Se a opção *-ps* for fornecida, um código postscript será gerado, e a terminação *.ps* será usada no nome do arquivo.
No exemplo seguinte um arquivo chamado *file.eps* é gerado contendo a imagem.

> *umat -cin file.cod [-average 1] [-median 1] [-ps 1]*

4.7 Outros programas

vcal - Este programa rotula as unidades do mapa de acordo com as amostras do arquivo de dados de entrada. A unidade vencedora do mapa correspondente a cada vetor de dados é procurada. As unidades do mapa são então rotuladas de acordo com a maioria de rótulos que 'encontraram' uma determinada unidade do mapa. As unidades que não foram encontradas nenhuma vez continuam sem rótulo. Dada a opção *-numlabs* pode-se selecionar o número máximo de rótulos armazenados par cada vetor de pesos. O valor default é um.

> vcal -din file.dat -cin file.cod -cout file.cod [-numlabs 2]

5 EXEMPLO DE APLICAÇÃO

O exemplo fornecido nesta seção demonstra o uso direto dos programas do pacote, através de linhas de comando. É indicado como uma introdução à aplicação deste pacote, e pode ser muito útil se estudado em detalhe. (Este exemplo também pode ser executado automaticamente através do comando *make example*.)

Os dados usados neste exemplo descrevem medições obtidas de um dispositivo real. Eles estão contidos no pacote e consistem de quatro conjuntos de dados: *ex.dat*, *ex_fts.dat*, *ex_ndy.dat* e *ex_fdy.dat*. O primeiro arquivo (*ex.dat*) contém 3840 amostras vetoriais de medições do estado do dispositivo e é usado para o treinamento do mapa, enquanto o segundo arquivo (*ex_fts.dat*) contém 246 exemplos de estados de falha do dispositivo e é usado para a calibragem do mapa. Os outros dois arquivos contêm amostras coletadas durante 24 horas de operação do dispositivo. No *ex_ndy.dat* as amostras são do dispositivo operando normalmente e no *ex_fdy.dat* de um dispositivo em superaquecimento. Estes arquivos de amostras são usados com o intuito de demonstrar como o mapa pode ser usado para monitoração de dispositivos.

Cada vetor destes arquivos tem dimensão 5.

Abaixo, o mapa é inicializado é treinado, o erro quantizado é avaliado e o mapa resultante é visualizado.

5.1 Primeiro estágio: Inicialização do mapa

Os vetores de pesos do mapa são primeiramente inicializados por tentativas. O tipo de grid do mapa e a função de vizinhança usados nos procedimentos de treinamento também são definidos na inicialização.

Neste exemplo, o mapa é inicializado usando-se números randômicos (a semente para o gerador de números aleatórios é especificada pelo parâmetro *-rand*). O tipo de grid é selecionado como hexagonal (*hexa*) e o tipo da função de vizinhança é a função step (*bubble*). O tamanho do mapa aqui é de 12 por 8 unidades.

> randinit -din ex.dat -cout ex.cod -xdim 12 -ydim 8 -topol hexa -neigh bubble -rand 123

Agora o mapa está inicializado.

5.2 Segundo estágio: Treinamento do mapa

O mapa é treinado pelo algoritmo self-organizing map através do programa *vsom*.

O treinamento é feito em duas fases. A primeira delas é a fase de ordenação, durante a qual os vetores de pesos das unidades do mapa são ordenados. Durante a segunda fase os valores dos vetores de pesos são ajustados para obter melhor precisão.

No início o raio de vizinhança é tomado como quase o diâmetro do mapa e decresce a um durante o treinamento, enquanto a taxa de aprendizagem decresce a zero. (Com a inicialização através do *lininit*, a primeira fase (ordenação) pode ser ignorada e somente a segunda fase de treinamento é necessária.)

> vsom -din ex.dat -cin ex.cod -cout ex.cod -rlen 1000 -alpha 0.05 -radius 10

Durante a segunda fase os vetores de pesos de cada unidade convergem para seus valores 'corretos'. A segunda fase é normalmente mais longa que a primeira. A taxa de aprendizagem é portanto menor. O raio de vizinhança é também menor na média: no início as unidades com distância de até três são cobertas. Neste exemplo, o tempo de treinamento da segunda fase é dez vezes mais longo que na primeira fase.

> vsom -din ex.dat -cin ex.cod -cout ex.cod -rlen 10000 -alpha 0.02 -radius 3

Depois destas duas fases de treinamento, o mapa está pronto para ser testado e usado em aplicações de monitoração.

5.3 Terceiro estágio: Avaliação do erro quantizado

Quando os nós no mapa tiverem sido treinadas para seus valores finais, o erro quantizado resultante pode ser avaliado. O arquivo de treinamento é utilizado para este propósito. O programa *qerror* é utilizado para avaliar o erro quantizado resultante.

> qerror -din ex.dat -cin ex.cod

Este programa computa o erro quantizado sobre todas as amostras do arquivo de dados. O erro quantizado médio com o conjunto de treinamento neste exemplo é esperado para estar em torno de 3.5.

5.4 Quarto estágio: Visualização do mapa

O mapa treinado pode agora ser usado para visualização de amostras de dados. Neste pacote existem programas para visualização, que constróem uma imagem do mapa (um plano selecionado dele) e traçam a trajetória da unidade vencedora pelo tempo.

Antes da visualização, as unidades do mapa são calibradas usando-se amostras de dados de entrada conhecidos. O arquivo de amostras *ex_fts.dat* contém amostras rotuladas do estado de um dispositivo em superaquecimento.

> *vcal -din ex_fts.dat -cin ex.cod -cout ex.cod*

Depois da calibragem algumas das unidades do mapa terão rótulos mostrando uma área do mapa correspondente a estados de falha.

O programa *visual* gera uma lista de coordenadas correspondente a todas as unidades vencedoras do mapa para cada amostra de dados do arquivo de dados. Ele também retorna os erros quantizados e os rótulos de classe das unidades vencedoras, se estes foram definidos. A lista de coordenadas pode então ser processada para vários dispositivos gráficos.

O arquivo de dados *ex_ndy.dat* contém amostras coletadas durante 24 horas de um dispositivo operando normalmente. O arquivo *ex_fdy.dat* contém amostras coletadas durante 24 horas de um dispositivo que apresentou problemas de superaquecimento durante o dia.

> *visual -din ex_ndy.dat -cin ex.cod -dout ex.nvs*

> *visual -din ex_fdy.dat -cin ex.cod -dout ex.fvs*

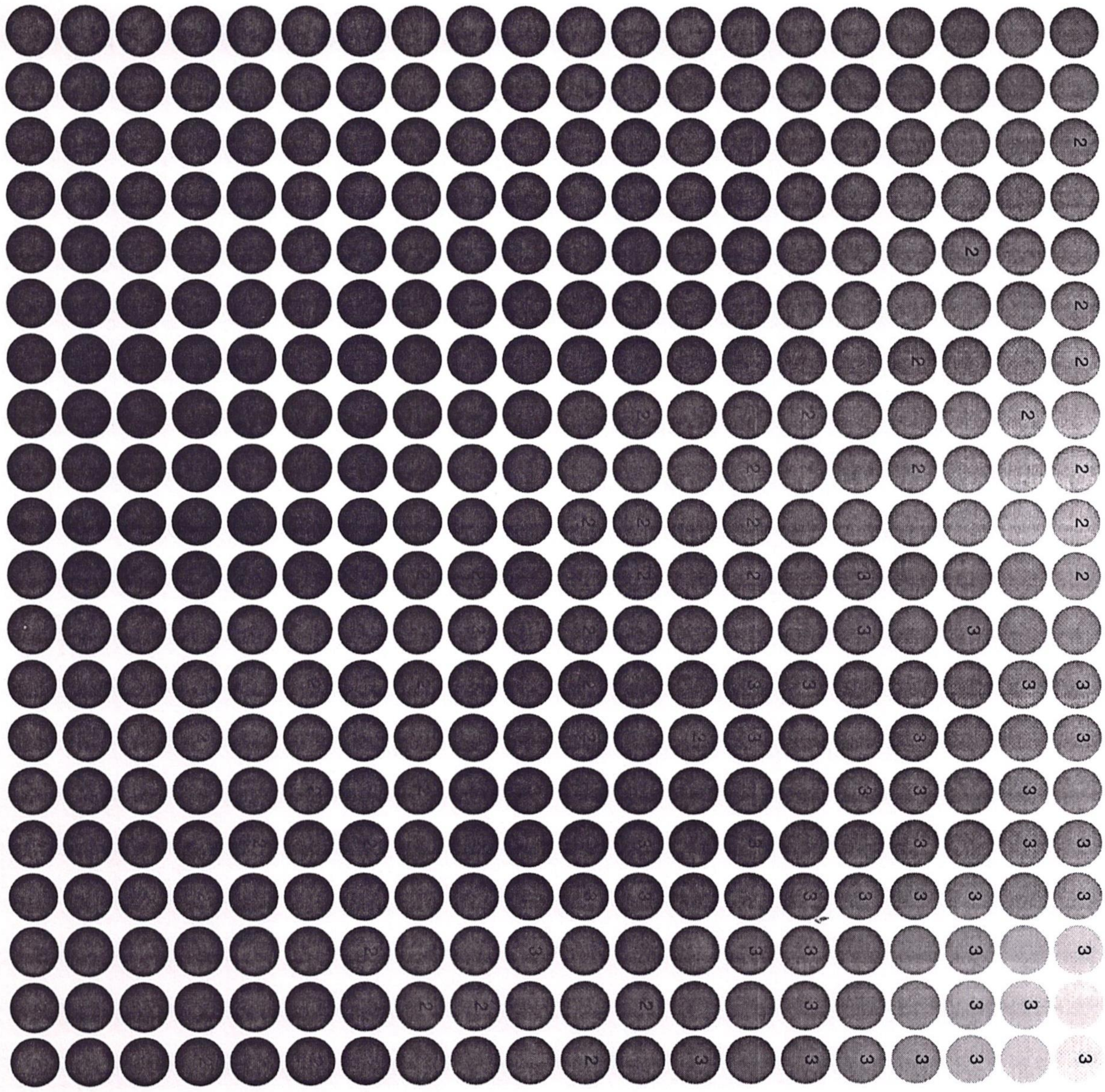
O programa *visual* armazena os pontos da imagem tridimensional (valor das coordenadas das respostas e dos erros quantizados) de um modo similar ao armazenamento das entradas de dados.

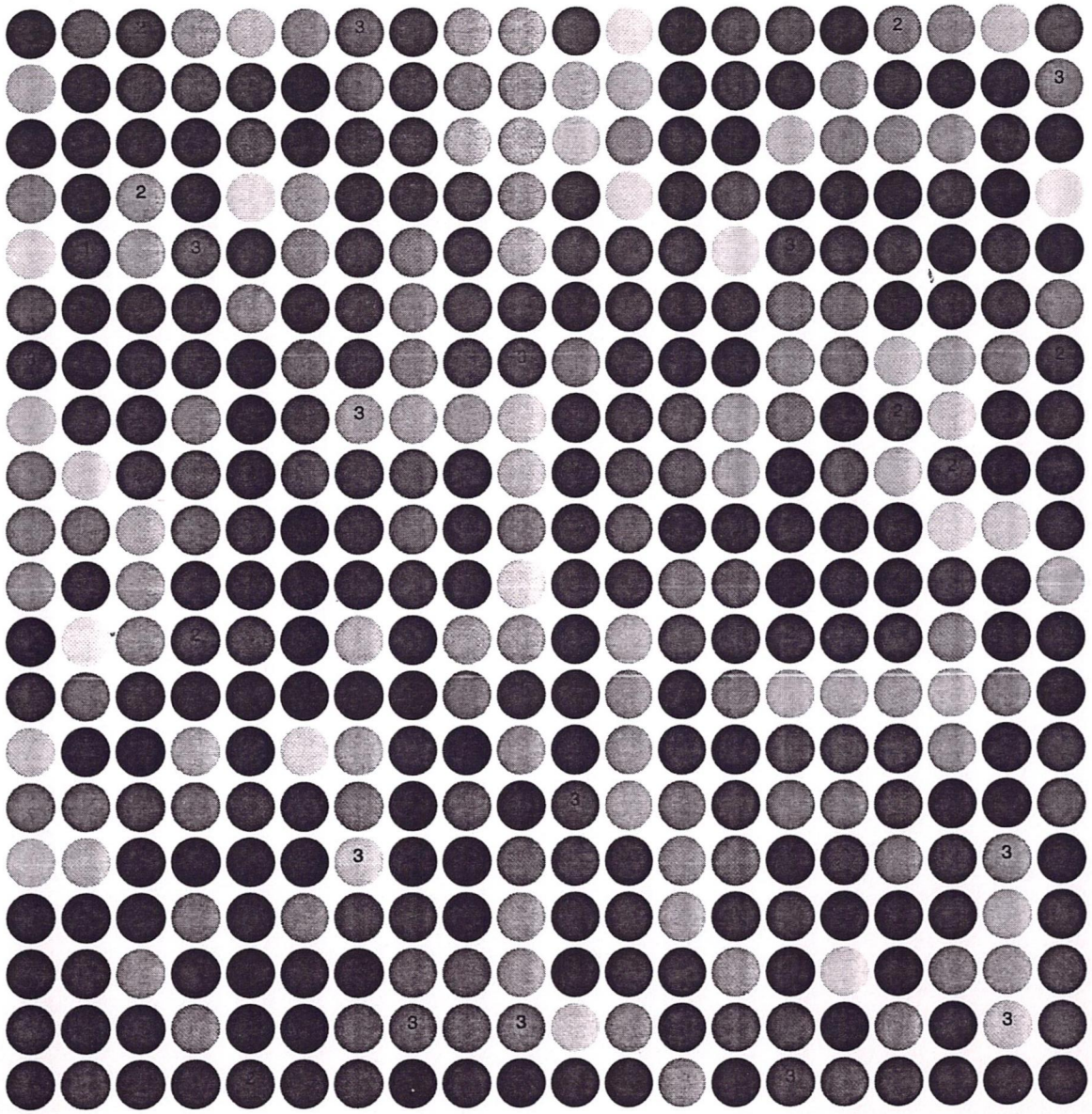
Este pacote inclui também os programas *planes* para converter os planos do mapa para imagens encapsulated postscript (eps) e *umat* para computar a chamada matriz de visualização dos vetores de pesos do SOM e convertê-los para imagem encapsulated postscript (eps).

6 EXEMPLO ILUSTRATIVO

Como exemplo de utilização deste pacote, foram realizados testes sobre a base de dados citada por Fisher (1936), conhecida como *Medidas sobre três variedades da planta Íris*.

As figuras seguintes representam o estado da rede antes e após o treinamento, respectivamente. Os números nas unidades, representam os rótulos armazenados, indicando o tipo de variedade da planta; e o tom de cinza, representa o valor dos vetores de pesos das respectivas unidades.





CONCLUSÃO

Este trabalho apresentou os conceitos básicos de utilização do pacote SOMPAK , que implementa os algoritmos da Rede de Kohonen.

Foram apresentados conceitos básicos de redes neurais, e do modelo de rede aqui utilizado. Em seguida, foram descritos os aspectos técnicos da ferramenta, como formato de arquivos, sintaxe dos principais programas e um exemplo de utilização dos mesmos.

Acreditamos que o presente trabalho será de grande utilização para usuários iniciantes do pacote, mostrando exemplos de utilização, bem como conceitos teóricos fundamentais.

Apêndice A - PARÂMETROS DOS PROGRAMAS

Vários programas necessitam de diversos parâmetros. Todos os parâmetros que são necessários aos programas deste pacote estão listados abaixo. Os parâmetros podem ser fornecidos em qualquer ordem na linha de comando.

-din Nome do arquivo de dados de entrada.

-dout Nome do arquivo de dados de saída.

-cin Nome do arquivo do qual são lidos os vetores de pesos.

-cout Nome do arquivo no qual serão armazenados os vetores de pesos.

-rlen Comprimento de execução (número de iterações) no treinamento.

-alpha Taxa de aprendizado inicial. Decresce linearmente a zero durante o treinamento.

radius Raio inicial da área de treinamento no algoritmo som. Decresce linearmente até atingir o valor 1 durante o treinamento.

-xdim Número de unidades na direção x.

-ydim Numero de unidades na direção y.

-topol Tipo de topologia usado no mapa. As opções possíveis são grid hexagonal (*hexa*) e grid retangular (*rect*).

-neigh Tipo de função de vizinhança usado. As opções possíveis são função step (*bubble*) e Gausiana (*gaussian*).

-plane O plano componente dos vetores de pesos que é mostrado nas rotinas de conversão.

-fixed Define se os qualificadores fixed-points serão usados nos programas de treinamento. O valor 1 significa que os qualificadores fixed-points serão considerados. O valor default é 0.

-weights Define se os qualificadores de intensificação serão usados nos programas de treinamento. O valor 1 significa que serão considerados. O valor default é 0.

-alpha_type Tipo da função da taxa de aprendizado (no *vsom* e *vfind*). As opções possíveis são função linear (*linear*, o default) e a função do tipo inverse-time (*inverse_t*). A função linear é definida como $\alpha(t) = \alpha(0)(1.0 - t/rlen)$ e a função do tipo inverse-time como $\alpha(t) = \alpha(0)C/(C + t)$ para computar $\alpha(t)$ para a iteração t. Neste pacote a constante C e definida $C = rlen/100.0$.

-qetype Tipo de Função de quantização do erro (usado no *qerror* e *vfind*). Se um valor maior que zero é fornecido então a função de quantização ponderada será usada.

-version Fornece o número da versão do SOM_PAK.

Além destes, é sempre possível se fornecer o parâmetro *-v n* (parâmetro verboso), que define quanto diagnóstico de saída o programa irá gerar. O valor de *n* pode variar de 0 em diante, onde valores altos irão gerar mais saída; o valor default é 1.

-v parâmetro verboso, define o nível de saída.

Na maioria dos programas é possível fornecer o parâmetro *-help 1*, que lista os parâmetros necessários e opcionais para o programa .

-help Fornece uma listagem descrevendo os parâmetros necessários e opcionais.

Nos programas de inicialização e treinamento o gerador de números aleatórios é usado para selecionar a ordem das amostras de treinamento, etc. O parâmetro *-rand* define se uma nova semente será fornecida para o gerador de números aleatórios; quando um número diferente de zero é fornecido, o número será usado como a semente, caso contrário, a semente será lida do relógio do sistema. O valor default é zero.

-rand Parâmetro que define se uma nova semente será definida para o gerador de números aleatórios.

Alguns exemplos do uso dos parâmetros:

> randinit -xdim 12 -ydim 8 -din exam 1.dat -cout code1.map -topo. hexa -neigh bubble

Um programa de inicialização foi chamado acima para criar um mapa de 12 por 8 unidades. As entradas usadas na inicialização foram lidas do arquivo exam1.dat e o mapa foi armazenado no arquivo *code1.map*. A topologia do mapa foi selecionada como hexagonal e a função de vizinhança é a função step.

> vsom -din exam1.dat -cin code1.map -cout code1.map -rlen 10000 -alpha 0.05 -radius 10

Foi chamado um programa de treinamento. As entradas para o treinamento foram lidas do arquivo *exam1.dat*; o mapa a ser treinado foi lido do arquivo *code1.map* e os vetores de pesos treinados foram regravados no mesmo arquivo. O treinamento foi definido para durar 1000 iterações, mas se o número de entradas no arquivo for menor, o arquivo será reutilizado o número necessário de vezes. A taxa de aprendizado inicial foi definida em 0.05 e o raio inicial de aprendizagem "bubble" foi definido como 10.

> qerror -din exam2.dat -cin code1.map

O erro quantizado relativo aos vetores de pesos armazenados no arquivo de mapa code1.cod foi testado usando o arquivo de dados de teste *exam2.dat*.

Apêndice B - DESCRIÇÃO DO LVQ-PAK

Neste apêndice é apresentada breve descrição dos principais programas contidos no pacote LVQ-PAK. Uma descrição mais detalhada destes programas, bem como uma apresentação teórica da técnica, pode ser encontrada na documentação contida junto com os arquivos do pacote.

O Programa de Interface *lvq_run*

O método mais fácil de se utilizar os programas do *lvq-pak* é executá-los através do programa de interface *lvq_run*, já que não são necessárias linhas de comando separadas. O programa lvq_run interage com o usuário perguntando pelos parâmetros necessários, e toma o cuidado de executar os demais subprogramas na ordem correta.

Usando os Programas Diretamente

Também é possível executar cada um dos subprogramas contidos neste pacote separadamente, e diretamente do console, usando-se linhas de comando. O usuário, no entanto, deve tomar cuidado para que os programas sejam executados na ordem correta: primeiramente uma inicialização adequada, então o treinamento, e em seguida os testes; e para que os parâmetros corretos sejam fornecidos (a correspondência entre os arquivos de entrada e saída entre os programas subsequêntes é de particular importância).

Cada programa necessita de diversos parâmetros: nomes de arquivos, parâmetros de aprendizado, tamanho dos vetores de pesos, etc. Todos estes devem ser fornecidos aos programas em seu início, já que os programas não são interativos durante a execução.

Programas de Inicialização

Os programas de inicialização inicializam os vetores de pesos. O número total de vetores de pesos é fornecido como parâmetro (*-noc*).

Eveninit - Este programa seleciona um número de vetores de pesos igual ao número de classes e define seus valores iniciais. Entradas classificadas erroneamente são rejeitadas

automaticamente pelo cheque *knn-classification* (sendo que k pode ser definido pelo parâmetro *-knn*; seu valor default é 5).

> eveninit -noc 200 -din file.dat -cout file.cod [-knn 7]

Propinit - Este programa define o número de vetores de pesos para cada classe, em proporção a probabilidade das classes.

> propinit - noc 200 -dim file.dat -cout file.cod [-knn 7]

Balance - Este programa ajusta os números dos vetores de pesos armazenados no arquivo definido pelo parâmetro *-cin*, usando dados de treinamento armazenados no arquivo *-din,* de modo que as medias das distâncias mais curtas entre os vetores de pesos em todas as classes sejam equalizados.

> balance -din file.dat -cin code1.cod -cout code2.cod {-knn 7]

Programas de Treinamento

olvq1 - Este é o algorítmo LVQ1 de taxa de aprendizado otimizada, recomendado para o algorítmo de aprendizagem principal. Ele deve ser sempre precedido por um programa de inicialização. O número de passos no treinamento é definido pelo parâmetro *-rlen.*

> olvq1 -din file.dat -cin file1.cod -cout file2.cod -rlen 10000 [-alpha_type inverse_t] [-snapinterval 1000] [-snapfile file.snap]

lvq1 - O algorítmo LVQ original. Pode ser usado (com valores para -alpha baixos) para um estágio adicional de ajuste, durante o aprendizado.

> lvq1 -din file.dat -cin file1.cod -cout file2.cod -alpha 0.05 -rlen 40000 [-alpha_type inverse_t] [-snapinterval 1000] [-snapfile file.snap]

lvq2 - Versão LVQ2.1 dos algorítmos LVQ. Pode ser usado (com valores para -alpha baixos) para um estágio adicional de ajuste, durante o aprendizado. A largura relativa da 'janela' em que os dados de treinamento devem ficar é definido pelo parâmetro *-win.*

> Lvq2 -din file.dat -cin file1.cod -cout file2.cod -alpha 0.05 -rlen 40000 -win 0.3 [-alpha_type inverse_t] [-snapinterval 1000] [-snapfile file.snap]

lvq3 - Versão LVQ3 dos algorítmos LVQ. Pode ser usado (com valores para -alpha baixos) para um estágio adicional de ajuste, durante o aprendizado. O parâmetro de taxa de aprendizagem relativa *-epsilon* é usado quando todos os vetores de pesos mais

próximos pertencem à mesma classe. A largura relativa da 'janela' em que os dados de treinamento devem ficar é definido pelo parâmetro *-win*.

> *Lvq3 -din file.dat -cin file1.cod -cout file2.cod -alpha 0.05 -epsilon -rlen 40000 -win 0.3 [-alpha_type inverse_t] [-snapinterval 1000] [-snapfile file.snap]*

Programa de Precisão do Reconhecimento

accuracy - A precisão no reconhecimento é avaliada. Opcionalmente este programa cria um arquivo contendo as informações da classificação, necessário no teste da significância estatística da diferença entre dois classificadores, usando-se o programa *mcnemar*.

> *accuracy -din file.dat -cin file1.cod [-cfout file.cfo]*

Programa de Classificação

classify - A classificação de vetores de dados desconhecidos é encontrada. Opcionalmente, este programa cria um arquivo de classificação que contém apenas os rótulos dos vetores classificados.

> *classify -din file.dat -cin file1.cod -dout file.cla [-cfout file.cfo]*

Programas de Monitoração

showlabs - Mostra os rótulos das classes e os números de entradas em cada classe de um dado arquivo.

> *showlabs. -cin file1.cod*

mindist - Mostra as médias das menores distâncias entre os vetores de pesos de cada classe, e os desvios padrão das entradas em cada classe no arquivo de dados correspondente (se fornecido).

> *mindist. -cin file1.cod [-din data.dat]*

stddev - Mostra as médias das menores distâncias entre os vetores de dados de cada classe, e os desvios padrão das entradas em cada classe.
> *stddev. -din file1.cod*

mcnemar - Computa a significância estatística da diferença entre resultados na classificação de dois qualificadores que foram testados com os mesmos dados. Como entrada, são necessários dois arquivos de informação de classificação criados pelo *accuracy*.

> *mcnemar file1.cfo file2.cfo*