Manual de uso do Toolbox SOM (Matlab)

Documento elaborado por: Nelson Albuquerque (ICA/DDEE/PUC-Rio)

Introdução ao SOMTOOLBOX

Esta instrução de usos do toolbox SOM (Matlab®) foi desenvolvida pela equipe da Prof.ª. Marley Velasco, do DEE/PUC-Rio. O aluno deve segui-la passo-a-passo para entender como a ferramenta funciona e poder realizar modificações para realização do exercício solicitado. Utilizando o arquivo de dados do próprio somtoolbox, "Iris.dat", veremos como inicializar e configurar uma rede (lattice) e, assim, prepará-la para o treinamento. Esse documento também servirá de guia para o aluno realizar o exercício proposto, cujo objetivo é, utilizando mapas de Kohonen, agrupar diferentes tipos de pacientes e, em seguida, determinar o perfil de cada grupo obtido.

O primeiro passo é instalar o toolbox no seu Matlab. Para isso deve obter os arquivos no seguinte endereço:

http://www.cis.hut.fi/projects/somtoolbox/package/ somtoolbox2_Mar_17_2005.zip

Esse toolbox vem com base de dados para exemplo (Iris.dat) e 4(quatro) tutoriais que devem ser estudados para melhor entendimento (som_demo1, som_demo2, som_demo3 e som_demo4), pois procuram mostrar o potencial dessa caixa de ferramentas do Kohonen.

O exercício está divido em duas partes: configuração do mapa e análise dos resultados.

- Configuração do Mapa:
    1. Leitura e tratamento dos dados
    2. Topologia
    3. Parametrização do treinamento durante a fase de ordenação
    4. Parametrização do treinamento durante a fase de ajuste fino
- Análise dos resultados

Leitura do arquivo de dados

O primeiro passo é ler o arquivo de dados, que deve ter uma característica própria como é apresentado no exemplo a seguir (iris.dat):

4

#n SepalL SepalW PetalL PetalW

5.1 3.5 1.4 0.2 Setosa

7.0 3.2 4.7 1.4 Versicolor

6.3 3.3 6.0 2.5 Virginica

O comando de leitura do arquivo é o seguinte:

```
sD = som_read_data('iris.data');
> data read ok
```

Observe que a execução dessa função cria uma variável ***sD*** (que é uma variável tipo '*struct*' que define toda a estrutura de dados contido no arquivo de dados), veja como é organizada a estrutura resultante da leitura do arquivo de dados <*sD* 1x1 struct>:

```
type 'som_data'
> data <150x4 double> 0.1000 7.9000
 labels <150x1 cell>
 name 'iris.data'
 comp_names <4x1 cell>
 comp_norm <4x1 cell>
 label_names [ ]
```

Isso significa que existem variáveis que seguem o seguinte padrão: *sD.XXXX*,

onde *XXXX* é uma "sub-variável", por exemplo *sD.type* guarda a informação sobre o tipo da estrutura. No exemplo citado trata-se de uma estrutura organizada para utilização pelo *Somtoolbox*. O conteúdo de cada variável composta será utilizado posteriormente pelas funções pré-definidas do *toolbox*.

Normalização dos dados

A seguir o exercício da IRIS instrui a normalização do arquivo de dados, através da função:

```
sD = som_normalize(sD,'range');
```

A estrutura *sD* possui o componente sD.Data normalizado com mínimo = 0 e máximo =1. No caso foi montado uma matriz com 500 linhas (registros lidos) e 4 colunas (as variáveis a serem analisadas.

```
data <150x4 double> 0 1
```

A função '*som_normalize*' tem a seguinte definição: '*som_normalize(sS,[method], [comps])*, que pode ser entendida como:

- sS ; é a designação da estrutura dos dados de entrada (no nosso caso é a variável sD),
- [method]; entre colchetes (significando que é opcional).

No exemplo foi indicado o método '*range*'. Existem vários métodos disponíveis: (para uma descrição mais detalhada veja SOM_NORM_VARIABLE).

MÉTODO	**DESCRIÇÃO**
'var'	A variância dos dados é normalizada para 1(um)(operação linear).
'range'	Os valores são normalizados pela amplitude dos dados entre [0,1] (operação linear).
'log'	É aplicado o LogNatural aos valores: xnew = log(x-m+1) onde m = min(x).
'logistic'	Transformação Logística ou softmax que leva todos valores entre [0,1]
'histD'	Equalização por Histograma, os valores são escalados entre [0,1].
'histC'	Equalização por histograma aproximada com operação parcialmente linear. Valores escalados entre [0,1]
'eval'	Operação definida livremente

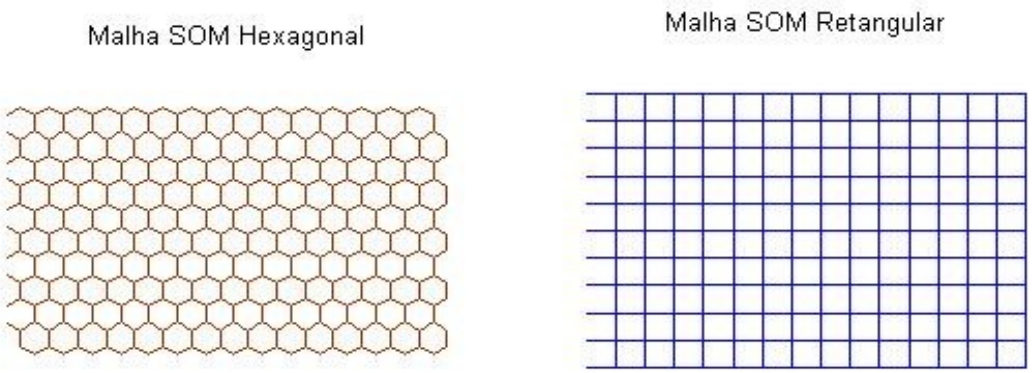
Após a execução esse comando são criadas 4(quatro) estruturas com o nome de *com_norm* <4x1 cell>. Ao investigar essas estruturas (uma para cada variável) observa-se as definições sobre o tipo de normalização, máximo, mínimo e o status que foi adotada em cada variável. A matriz definida por *sD.data* agora está normalizada.

```
data <150x4 double> 0 1
```

Topologia da Rede

O *somtoolbox* possui vários modos de montar e treinar a rede SOM. Como se sabe, inicialmente, os neurônios da topografia estabelecida precisam receber valores de forma aleatória antes de iniciar o treinamento. A seguir é apresentado os dois tipos de topologia que são adotados nesse toolbox.

```
figure1 = figure('Name','Exemplo_de_Malhas_SOM');
subplot(1,2,1)
som_cplane('hexa',[10 15],'none')
title('Malha  Hexagonal')
subplot(1,2,2)
som_cplane('rect',[10 15],'none')
title('Malha  Retangular')
```



Cada neurônio (hexagonal à esquerda e retangular à direita da figura) tem um vetor protótipo associado. Após o treinamento, os neurônios vizinhos terão vetores protótipos similares. O treinamento se dá em duas fases. Em ambas a escolha da vizinhança delimita a amplitude das comparações e ajustes (produto escalar entre o dado e o protótipo pré-definido). Na primeira fase é realizado um treinamento mais grosseiro e na segunda um ajuste fino a partir do resultado do treinamento anterior. Ao final é eleito um neurônio, entre os vizinhos avaliados, que mais se aproxima ao dado que está sendo observado pelo algoritmo.

Treinamento

No exemplo dado em aula vimos duas maneiras de se executar o treinamento; num foi utilizado uma função automática, onde apenas se informa a estrutura dos dados [som_make(sD)]. Com o uso dessa função não há necessidade de se especificar o tipo de inicialização nem especificar as etapas de treinamento embora seja possível informar os parâmetros que se deseja, forçando o seu uso. Todas as características são identificadas automaticamente e a função realiza todo o procedimento do início ao fim.

Uma outra modalidade é instruir passo-a-passo. Neste caso há necessidade de se especificar os parâmetros e as etapas são executadas com comandos / funções distintas.

Inicialmente é especificado a característica da rede e a sua dimensão. Essa função executa uma inicialização aleatória dos vetores protótipo. A função é a seguinte:

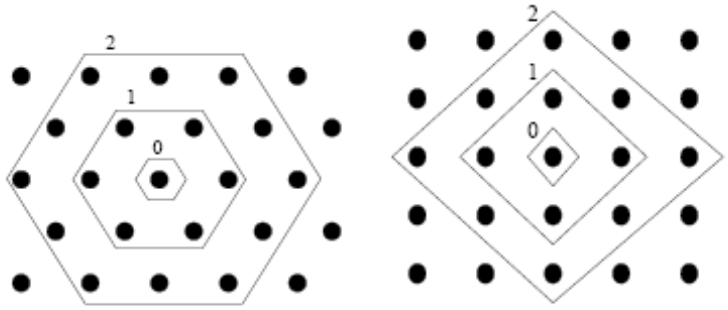
sM = som_randinit(sD,'msize', [10 10]).

Em se informa a função de treinamento (pode ser sequencial ou em batelada):

sM = som_seqtrain(sM,sD,'radius',[4 1]).

Na sequencial é necessário especificar a amplitude do treinamento, ou seja, a distância máxima de vizinhança a ser avaliada (p. ex., iniciando com a distância de 4 neurônios vizinhos e terminando com apenas um [4 1]).

A seguir apresentamos graficamente a organização da rede e a definição de vizinhança, onde podemos ver a distância de 2 vizinhos e 1 vizinho, para cada topografia utilizada.



(a) Hexagonal grid (b) Rectangular grid

Treinamento automático (default em batch)

No caso da função "*som_make(...)*", a escolha do tamanho da malha é automático, ou seja, o algoritmo escolhe uma rede em função do tamanho da matriz de dados. Em seguida inicializa a rede aleatoriamente; escolhe o método de treinamento, que no caso default é por "*batch*". Organiza cada uma das etapas de forma automática. Veja como isso ocorre e a o que aparece no "Command Window" do Matlab:

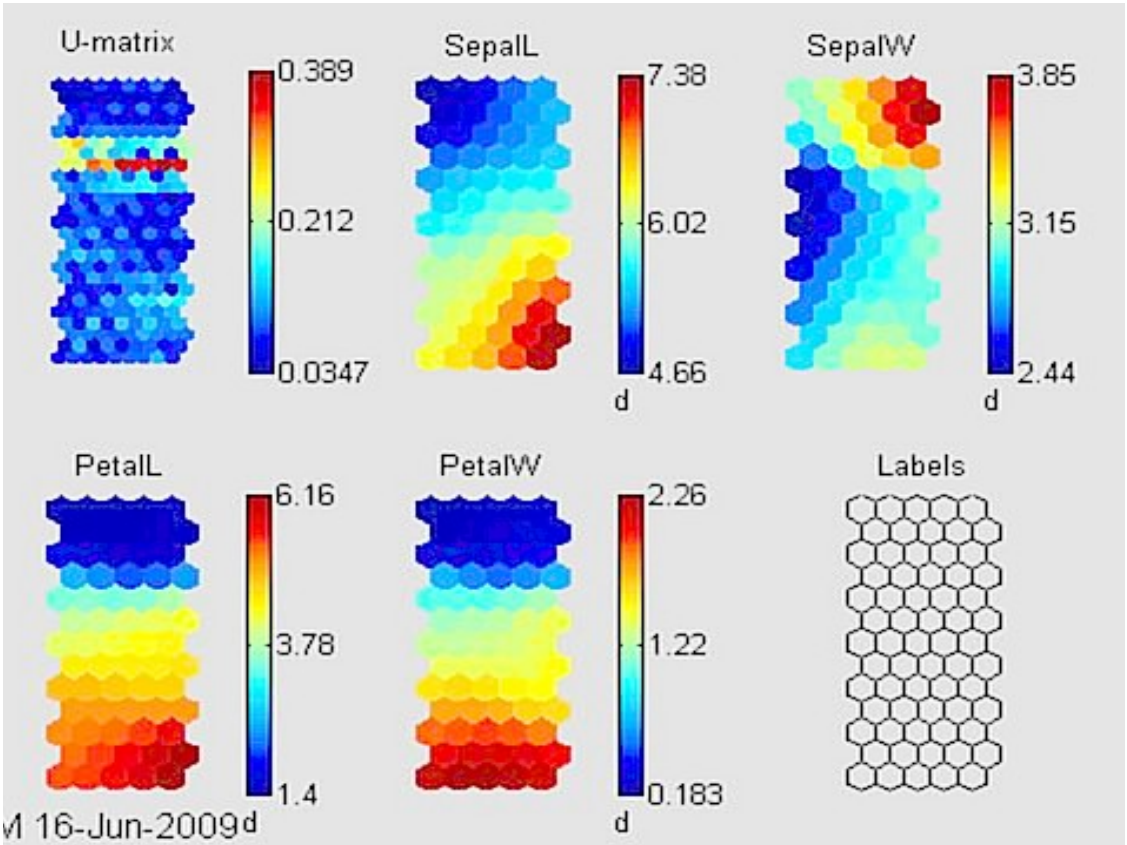
```
sM = som_make(sD);
Determining map size...
 map size [13, 5]
Initialization...
Training using batch algorithm...
Rough training phase...
```

```
Training:   0/   0 s
Training:   0/   0 s
….....
Finetuning phase...

Training:   0/   0 s
Training:   0/   0 s
….....
Training:   0/   0 s
Final quantization error: 0.093
Final topographic error:  0.027
```

A seguir observa-se um quadro com figuras obtidas da rede treinada sob a perspectiva das quatro varáveis e da matriz U.

som_show(sM,'umat','all','comp',1:4, 'empty','Labels','norm','d');

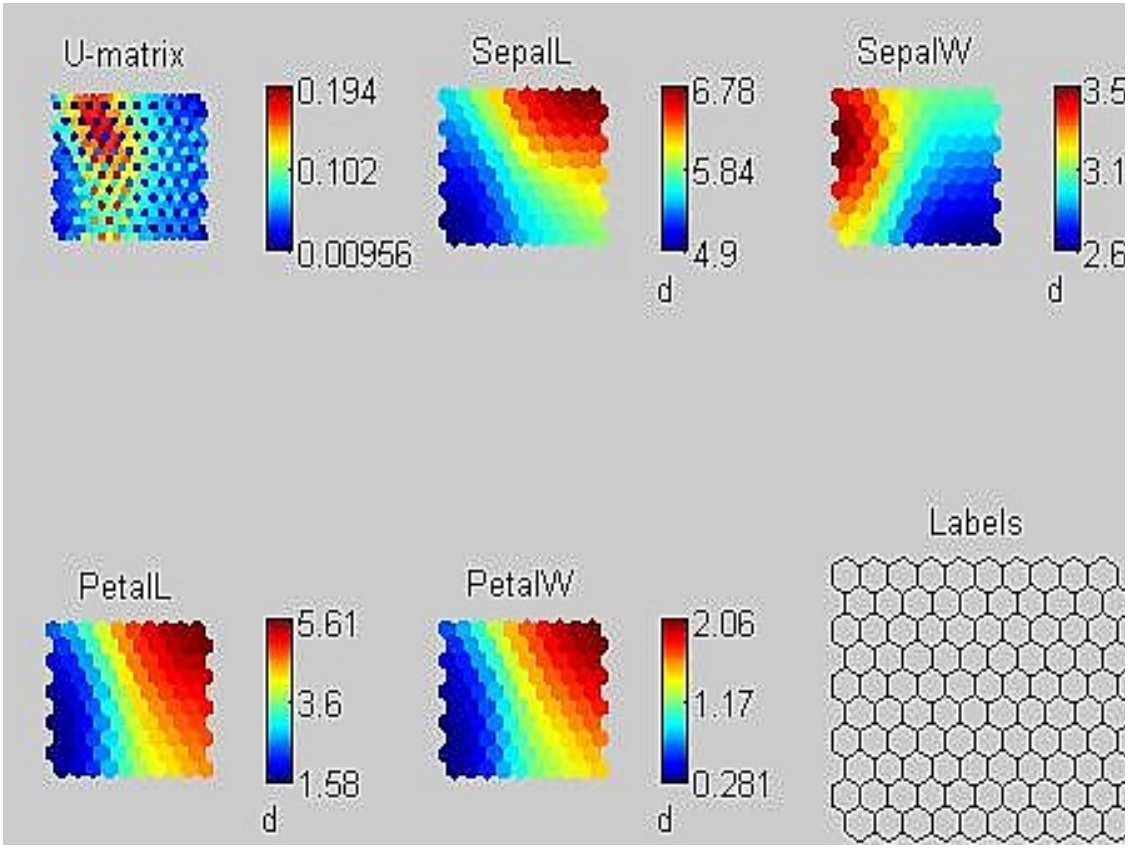


Treinamento sequencial

Na segunda modalidade de treinamento do exemplo são apresentados ao Matlab os seguintes comandos:

```
sM = som_randinit(sD,'msize', [10 10]);
[sM,sT] = som_seqtrain(sM,sD,'radius',[4 1]);
som_show(sM,'umat','all','comp',1:4, 'empty','Labels','norm','d');
```

```
Training:   0/   0 s
Training:   0/   0 s
Training:   0/   0 s
Training:   0/   0 s
Training:   0/   0 s
Training:   0/   0 s
Training:   0/   0 s
Training:   0/   0 s
Training:   0/   0 s
Training:   0/   0 s
```



Detalhando os tipos de treinamento

. Treinamento sequencial

O treinamento pode ser melhor entendido avaliando as diversas características da função "*som_seqtrain()*". Para isso deve ser observado oque a função realiza e seus principais parâmetros. Os parâmetros que não são informados são assumidos internamente com um valor default.

Para seguir o roteiro(script) da sequência adotada utilize o comando '*type som_sequetrain*' que mostra a descrição da função, o *.m script*:

```
[sM, sT] = som_seqtrain(sM, D, varargin)
```

```
SOM_SEQTRAIN Utiliza o algoritmo de treinamento sequencial para treinar uma SOM.
[sM,sT] = som_seqtrain(sM, D, [[argID,] value, ...])
sM      = som_seqtrain(sM,D);
sM      = som_seqtrain(sM,sD,'alpha_type','power','tracking',3);
[M,sT]  = som_seqtrain(M,D,'ep','trainlen',10,'inv','hexa');
```

Os argumentos de Entrada e Saída escrito entre colchetes [] são opcionais:

```
sM (struct)     armazena o mapa da estrutura e também  resultados do treinamento
(matrix)        matriz de código de uma SOM
tamanho         munits x dim  ou  msize(1) x ... x msize(k) x dim
```

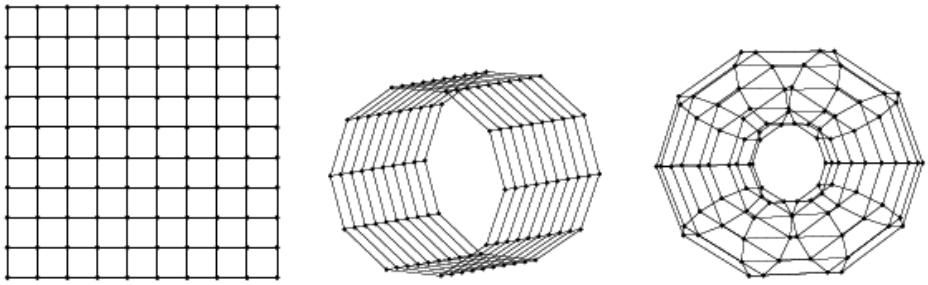
O resultado é o mapa de código treinado.

```
D       (struct)        Dados de treinamento; estrutura dos dados
        (matrix)        matriz de treinamento, tamanho é dlen x dim
[argID, (string)        Veja a seguir. Os valores ambíguos (*) podem ser dados sem o
                         argumento precedente (argID).
value]  (varies)
sT      (struct)        parâmetros de aprendizagem utilizados durante o treinamento
```

Observar que a estrutura (*sT*) é onde se guardam todos os parâmetros de treinamento que foram utilizados. A seguir são apresentados os argumentos válidos (IDs) e seus valores correspondentes. Os valores ambíguos são marcados com (*).

```
'mask'          (vector) máscara de pesquisa BMU, tamanho dim x 1
'msize'         (vector) tamanho do mapa
'radius'        (vector) raio de vizinhança,tamanho 1, 2 ou 'trainlen'
'radius_ini'    (scalar) rio para treinamento inicial
'radius_fin'    (scalar) raio para treinamento final
'alpha'         (vector) taxa de aprendizagem, tamanho do 'trainlen'
'alpha_ini'     (scalar) taxa de aprendizagem inicial
'tracking'      (scalar) nível de rastro ou trilha, 0-3
'trainlen'      (scalar) tamanho do treinamento
'trainlen_type' *(string) é o número de 'samples' ou 'epochs'
'train'        *(struct) estrutura de treinamento, parâmetros para treinamento
'sTrain','som_train '  = 'train'
'alpha_type' *(string) função da taxa de aprendizagem, 'inv', 'linear' ou 'power'
'sample_order'*(string) ordem das amostras: 'random' ou 'ordered'
'neigh'        *(string) função da vizinhança, 'gaussian', 'cutgauss', 'ep' ou
                'bubble'
'topol'        *(struct) estrutura da topologia
'som_topol','sTopo l'  = 'topol'
'lattice'      *(string) tipo de rede, 'hexa' or 'rect'
'shape'        *(string) tipo de mapa, plano=> 'sheet', cilíndrico=> 'cyl' ou toroidal
                => 'toroid'
%-------------------------------------------------------------------------
```

Observar os tipos de mapas disponíveis nesse *toolbox*:



(a) Sheet (b) Cylinder (c) Toroid

. Treinamento automático

No caso da função SOM_MAKE os parâmetros nas etapas necessárias são alocados automaticamente com valores *default* ou calculados a partir da base de dados fornecida. Ou seja, define a malha (tamanho e tipo), inicializa, faz um primeiro treinamento (*Rough phase*) e depois um segundo com ajuste fino (*Finetuning phase*).

Inicialmente se cria as estruturas e depois verifica qual foi o algoritmo escolhido (no caso default é o *batch*). A função realiza todas as etapas conforme está escrito no *script* da função som_make().

Uma maneira interessante de se aprender como essa função é realizada é acompanhando a sua execução através do módulo de inspeção chamada de *debug* onde é possível verificar como os parâmetros vã sendo incorporados. A seguir um pedaço do script da função:

```
mapsize = '';
sM = som_map_struct(dim);
sTopol = sM.topol;
munits = prod(sTopol.msize); % should be zero
mask = sM.mask;
name = sM.name;
neigh = sM.neigh;
tracking = 1;
algorithm = 'batch';
initalg = 'lininit';
training = 'default';  <=========
```

Como se pode observar o treino escolhido foi o `'default'`. O treino pode ser realizado entre um dos seguintes tipos:

```
%   'training'     (string) 'short', 'default' or 'long'
%                  (vector) size 1 x 2, first length of rough training
%                  in epochs, and then length of fine-tuning in epochs
```

Cada um dos tipos recebe uma quantidade de treinamentos diferentes. O treinamento grosseiro é feito em 5(cinco) épocas e a vizinhança inicias é igual a 2(dois). Na segunda fase observa-se que o treinamento é feito em 8(oito) épocas e a vizinhança inicial é iguala 1(um).

Vamos dar uma olhada como essa sequência é realizada no Matlab:

```
>> sM = som_make(sD)
Determining map size...
 map size [13, 5]
Initialization...
Training using batch algorithm...
Rough training phase...

Training:   0/   0 s
Training:   0/   0 s
Training:   0/   0 s
Training:   0/   0 s
Training:   0/   0 s
Finetuning phase...

Training:   0/   0 s
Training:   0/   0 s
Training:   0/   0 s
Training:   0/   0 s
Training:   0/   0 s
Training:   0/   0 s
Training:   0/   0 s
Training:   0/   0 s
Training:   0/   0 s
Training:   0/   0 s
Training:   0/   0 s
Training:   0/   0 s
Training:   0/   0 s
Training:   0/   0 s
Training:   0/   0 s
Training:   0/   0 s
Training:   0/   0 s
Final quantization error: 0.093
Final topographic error:  0.027

sM =
        type:           'som_map'
        codebook:       [65x4 double]
        topol:          [1x1 struct]
        labels:         {65x1 cell}
        neigh:          'gaussian'
        mask:           [4x1 double]
        trainhist:      [1x3 struct]      <==============
        name:           'SOM 23-Jun-2009'
        comp_names:     {4x1 cell}
        comp_norm:      {4x1 cell}
```

Veja que a variável "*sM.trainhist*" é uma estrutura que guarda as características de cada etapa. Vejamos o que ocorreu:

Variável	Primeira fase	Segunda fase	Terceira fase
type	'som_train'	'som_train'	'som_train'
algorithm	'lininit'	'batch'	'batch'
data_name	'iris.data'	'iris.data'	'iris.data'

Variável	Primeira fase	Segunda fase	Terceira fase
neigh	''	'gaussian'	'gaussian'
mask	[ ]	[1;1;1;1]	[1;1;1;1]
radius_ini	NaN	2	1
radius_fin	NaN	1	1
alpha_ini	NaN	NaN	NaN
alpha_type	''	'inv'	'inv'
trainlen	NaN	5	18
time			

Lendo a descrição da função, observamos como se processa o treinamento. Veja o texto a seguir:

```
% rough train
if tracking>0, fprintf(1,'Rough training phase...\n'); end
sTrain = som_train_struct(sMap,'dlen',dlen,'algorithm',algorithm,'phase','rough');
sTrain = som_set(sTrain,'data_name',data_name);
if isnumeric(training), sTrain.trainlen = training(1);
else
  switch training,
   case 'short', sTrain.trainlen = max(1,sTrain.trainlen/4);
   case 'long',  sTrain.trainlen = sTrain.trainlen*4;
  end
end
switch func,
 case 'seq',    sMap = som_seqtrain(sMap,D,sTrain,'tracking',tracking,'mask',mask);
 case 'sompak', sMap = som_sompaktrain(sMap,D,sTrain,'tracking',tracking,'mask',mask);
 case 'batch',  sMap = som_batchtrain(sMap,D,sTrain,'tracking',tracking,'mask',mask);
end
```

Lendo as explicações do *script* é possível verificar que podemos alterar os parâmetros pré estabelecidos e que a variável.

```
% The change training parameters, the optional arguments 'train',
% 'neigh','mask','trainlen','radius','radius_ini', 'radius_fin',
% 'alpha', 'alpha_type' and 'alpha_ini' are used.
%  sM = som_seqtrain(sM,D,'neigh','cutgauss','trainlen',10,'radius_fin',0);
%
% Another way to specify training parameters is to create a train struct:
%  sTrain = som_train_struct(sM,'dlen',size(D,1),'algorithm','seq');
%  sTrain = som_set(sTrain,'neigh','cutgauss');
%  sM = som_seqtrain(sM,D,sTrain);
```

Observamos que se pode modificar os parâmetros do treinamento diretamente na função *som_seqtrain* ou através da criação de uma variável tipo 'estrutura', por exemplo, *sTrain* com todos os parâmetros que se deseja manipular. Veja como é organizada essa função (*type som_traim_struct* e *som_set*).

Uma maneira de descobrir como sua estrutura está sendo montada é visualizar as variáveis tipo 'estrutura' durante o treinamento. Veja como fazer isso:

```
% [sM,sT]      = som_seqtrain(sM, D, [[argID,] value, ...])
%
%  sM           = som_seqtrain(sM,D);
%  sM           = som_seqtrain(sM,sD,'alpha_type','power','tracking',3);
%  [M,sT]       = som_seqtrain(M,D,'ep','trainlen',10,'inv','hexa');
%
%  Input and output arguments ([]'s are optional):
%   sM      (struct) map struct, the trained and updated map is returned
%          (matrix) codebook matrix of a self-organizing map
%                   size munits x dim or  msize(1) x ... x msize(k) x dim
%                   The trained map codebook is returned.
%   D       (struct) training data; data struct
%          (matrix) training data, size dlen x dim
%   [argID, (string) See below. The values which are unambiguous can
%    value] (varies) be given without the preceding argID.
%
%   sT      (struct) learning parameters used during the training
```

Observe que a variável estrutura ***sT*** guarda todos os parâmetros que são utilizados no treinamento. Rode seu programa e analise o que ocorre com essa variável.

Vejamos um exemplo:

Ao ler o arquivo Iris é criada uma estrutura com as principais informações do arquivo lido:

```
>> sD = som_read_data('iris.data');


data read ok
>> sD

 sD =

         type: 'som_data'
         data: [150x4 double]
       labels: {150x1 cell}
         name: 'iris.data'
    comp_names: {4x1 cell}
     comp_norm: {4x1 cell}
   label_names: []
```

A variável *sD* é uma estrutura e para ver o seu detalhamento pode abrir diretamente no Editor de Variável (dando dois cliques na variável *sD* no painel "*Workspace*".

Observe que lá você vê as variáveis que compõe a estrtura *sD* (*sD.type*, *sD.data*, *sD.labels*, *sD.name*, *sD.comp_names*, *sD.comp_norm*, *sD.label_names*). Cada uma dessas variáveis tem informações que serão utilizadas posteriormente.

Num passo seguinte, vamos normalizar a estrutura sD. Se observarmos a estrutura sD identificamos que a sD.data foi modificada e que agora a variável sD é composta de 4 estruturas, com os parâmetros utilizados para normalização. Isso é necessário para se obter o resultado na dimensão original (des-normalização).

Veja o exemplo de uma dessas estruturas (elemento 1;1 da matriz estrutura): sD.comp_norm{1,1}.type, sD.comp_norm{1,1}.method, sD.comp_norm{1,1}.params, sD.comp_norm{1,1}.status. Cada uma dessas variáveis contém a informação sobre os dados originais da spalL.

Agora vamos rodar a inicialização da rede:

Uma nova estrutura foi criada, vamos ver como ela é:

sM.type,	'som_map',
sM.codebook,	<100x4 double>,
sM.topol,	<1x1 truct>,
sM.labels,	<100x1 ell>
sM.neigh,	'gaussian',
sM.mask,	[1;1;1;1],
sM.trainhist,	<1x1 struct>
sM.name,	'SOM 23-Jun-2009'
sM.comp_names,	<4x1 cell>
sM.comp_norm,	<4x1cell>

A variável "*sM.topol*" e a variável "*sM.trainhist*" são estruturas que já tem alguns parâmetros *default* que serão utilizados caso você não os modifique. Por exemplo:

A topologia:

sM.topol.type,	'som_topol',
sM.topol.msize,	[10,10],
sM.topol.lattice,	'hexa',
sM.topol.shape,	'sheet'

Ou seja, será uma rede 'hexagonal' 'plana' com tamanho [10x10]; .

A estrutura de treinamento default é dada por:

	Antes do treinamento	**Depois do treinamento**
sM.trainhist.type	'som_train'	'som_train'
sM.trainhist.algorithm	'randinit'	'seq'
sM.trainhist.data_name	'iris.data'	'iris.data'

	Antes do treinamento	Depois do treinamento
sM.trainhist.neigh	''	'gaussian'
sM.trainhist.mask	[ ]	[1;1;1;1]
sM.trainhist.radius_ini	NaN	4
sM.trainhist.radius_fin	NaN	1
sM.trainhist.alpha_ini	NaN	0.500
sM.trainhist.alpha_type	''	'inv'
sM.trainhist.trainlen	NaN	7
sM.trainhist.time	'23-Jun-2009 09:39:37'	'23-Jun-2009 10:04:43'

Podemos observar que na inicialização não há detalhes sobre como o treinamento será realizado [NaN] nem o valor do alpha [‘’].

Na etapa seguinte é que os parâmetros são passados. Mas observe que caso não haja definição explícita, a função assumirá os valores default. Ao lado vemos como fica o treinamento. No caso do exemplo foram definido alguns parâmetros previamente

```
sM = som_seqtrain(sM,sD,'radius',[4 1]
```

Com essas informações é possível criar um script para rodar vários modelos e ter um resultado completo de sua experiência.

Observe que segundo a explicação do algoritmo de treinamento, pode ser dividido em duas fases, uma de um treinamento mais grosseiro, onde você pode informar