CSE2330 Introduction to Computational Neuroscience

Associative Memory Networks

Tutorial 4 **Duration**: two weeks

4.1 About this tutorial

The objective of this tutorial is to introduce the concept of Associative or Contents-Addressable Memory and study its various implementations and properties.
In addition we will study a simple model of the Alzheimer's disease.

Contents

4.1 About this tutorial . . . 1
4.2 Feed-Forward Linear Associative Memory . . . 2
4.2.1 Introductory concepts . . . 2
4.2.2 Encoding multiple memories . . . 3
4.2.3 Decoding operation . . . 3
4.2.4 A simple script for the linear associator . . . 4
4.3 Recurrent Associative Memory — Discrete Hopfield networks . . . 7
4.3.1 Structure . . . 7
4.3.2 Example of the Hopfield network behaviour for $m = 3$ . . . 8
4.4 Simple model of Alzheimer's disease . . . 9
4.4.1 Background information . . . 9
4.4.2 Specification of the fundamental memories to be stored . . . 9
4.4.3 First implementation of the model . . . 12
4.4.4 Second implementation of the model . . . 15

getting started ...

The amount of information and the number of technical terms, computational methods and concepts that you are expected to master increase quickly, therefore, you are strongly advised to come well prepared to the practical classes.
Read at least the previous prac manuals, but you cannot go wrong if you also read related lecture notes and book chapters.
I also encourage you to execute MATLAB scripts line-by-line in order to build-up your understanding of computational methods involved.

4.2 Feed-Forward Linear Associative Memory

4.2.1 Introductory concepts

At the introductory level the concept of memorizing a pattern in synaptic weights and its retrieval is based on the "read-out" property of the outer product of two vectors that we studied in prac 1. Assume that we have a **pair of column vectors**:

p-component vector	$\boldsymbol{\xi}$	representing the input pattern
m-component vector	$\mathbf{q}$	representing the desired output association with the input pattern

The pair $\{ \boldsymbol{\xi}, \mathbf{q} \}$ to be stored is called a **fundamental memory**.

Encoding a single memory

We **store or encode** this pair in a matrix $\boldsymbol{W}$ which is calculated as an outer product (column $\times$ row) of these two vectors

$$\boldsymbol{W} = \mathbf{q} \cdot \boldsymbol{\xi}^{\boldsymbol{T}} \tag{1}$$

Decoding a single memory

The **retrieval or decoding** of the store pattern is based on application of the input pattern $\mathbf{x}$ to the weight matrix $\boldsymbol{W}$. The result can be calculated as follows:

$$\mathbf{y} = \boldsymbol{W} \cdot \boldsymbol{\xi} = \mathbf{q} \cdot \boldsymbol{\xi}^{\boldsymbol{T}} \cdot \boldsymbol{\xi} = ||\boldsymbol{\xi}|| \cdot \mathbf{q} \tag{2}$$

The equation says that the decoded vector $\mathbf{y}$ for a given input pattern $\boldsymbol{\xi}$ is proportional to the encoded vector $\mathbf{q}$, the length of the input pattern $\boldsymbol{\xi}$ being the proportionality constant.

The above considerations give rise to a simple feed-forward associate memory known also as the **linear associator**. It is a single layer feed-forward network with m neurons each with p synapses as illustrated in Figure 1.

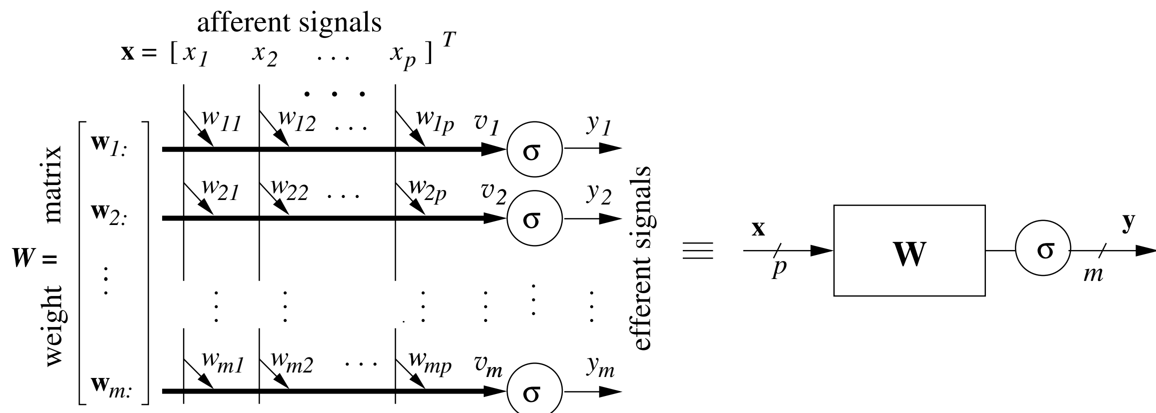


Figure 1: The structure of a feed-forward linear associator: $\mathbf{y} = \boldsymbol{\sigma}(\boldsymbol{W} \cdot \mathbf{x})$

For such a simple network to work as an associative memory, the input/output signal are **binary signals** with

$$\{0, 1\} \text{ being mapped to } \{-1, +1\}$$

4.2.2 Encoding multiple memories

Extending the introductory concepts let us assume that we would like to store/encode N **pairs** of column **vectors** (fundamental memories) arranged in the two matrices:

$\Xi = \xi(1) \ldots \xi(N)$	a matrix of p-component vectors representing the desired input patterns
$Q = \mathrm{q}(1) \ldots \mathrm{q}(N)$	a matrix of m-component vectors representing the desired output associations with

In order to **encode** the $\{\Xi, Q\}$ patterns we **sum outer products** of all pattern pairs:

$$W = \frac{1}{N} \sum_{n=1}^{N} \mathrm{q}(n) \cdot \xi^T(n) = \frac{1}{N} Q \cdot \Xi^T \tag{3}$$

The sum of the outer products can be conveniently replaced by product of two matrices consisting of the pattern vectors. The resulting $m \times p$ matrix W encodes all the desired N pattern pairs $\mathrm{x}(n), \mathrm{q}(n)$.

Note that eqn (3) can be seen as an extension of the Hebb's learning law in which we multiply afferent and efferent signals to form the synaptic weights.

4.2.3 Decoding operation

Retrieval of a pattern is equally simple and involves acting with the weight matrix on the input pattern (the key)

$$\mathrm{y} = \sigma(W \cdot \mathrm{x}) \tag{4}$$

where the function σ is the sign function:

$$y_j = \sigma(v_j) = \begin{cases} +1 & \text{if } v_j \geq 0 \\ -1 & \text{otherwise} \end{cases} \tag{5}$$

It is expected that

1. $\mathrm{x} = \xi$

   If the key (input vector) x is equal to one of the fundamental memory vectors ξ, then the decoded pattern y will be equal to the stored/encoded pattern q for the related fundamental memory.

2. $\mathrm{x} = \xi + n$

   If the key (input vector) x can be considered as one of the fundamental memory vectors ξ, corrupted by noise n then the decoded pattern y will be also equal to the stored/encoded pattern q for the related fundamental memory.

3. $\mathrm{x} \neq \xi + n$

   If the key (input vector) x is definitely different to any of the fundamental memory vectors ξ, then the decoded pattern y is a spurious pattern.

- The above expectations are difficult to satisfy in a feedforward associative memory network if the number of stored patterns N is more than a fraction of m and p.
- It means that the **memory capacity** of the feedforward associative memory network is **low** relative to the dimension of the weight matrix W.

In general, associative memories also known as content-addressable memories (CAM) are divided in two groups:

Auto-associative: In this case the desired patterns Q are identical to the input patterns X, that is, $Q = X$. Also $p = m$.

Hetero-associative: In this case the input X and stored patterns Q and are different.

4.2.4 A simple script for the linear associator

In the first example we test how the linear associator encodes and decodes binary patterns.

```
%  p4ffAM.m
%               12 September 2004
% Feed-Forward Auto-Associative Memory
clear
%  Generation of binary patterns to be stored
p = 4 ;   % number of bits. 2^p possible patterns
          % dimensionality of patterns and number of neurons
N = 5 ;   % number of patterns
% generate  pp = 2^p - 1 integer numbers
pp = 2^p - 1 ;
Xd = randperm(pp)      % generate  pp integer random numbers
Xd = Xd(1:N) ;         % taking the first N numbers
X = (dec2bin(Xd, p) - '0')'  % convert integers to binary strings
% recoding {0, 1}  to  {-1 , +1}
X = 2*X - 1     % the matrix of patterns to be stored
W = (X*X')/N ;  % storing the patterns in the weight matrix
W_ = num2str(W)
%       decoding
% Creating all possible p-bit patterns coded {-1, +1}
Xall = 2*((dec2bin(1:pp) -'0')') -1 ;
Xall_ = num2str(Xall)

% Recalling all possible binary patterns
Yb = (W*Xall >= 0) ;          % in {0, 1} form
Yb_ = num2str(Yb)
Y = 2*Yb -1 ;       % recoded to {-1, +1} form
Y_ = num2str(Y)

% recoding recalled patterns to decimals
```

```
pwrs2 = fliplr(2.^((1:p)-1))  % powers of 2
Yd = pwrs2*Yb ;    % conversion to equivalent decimal numbers
X_Yd = num2str([1:pp ;Yd], '%3d')
Xd

figure(1)
plot(1:pp, Yd, '*', Xd, Xd, 'o'),
grid on, axis([0 2^p 0 2^p]),
xlabel('input patterns'), ylabel('o-stored and *-retrieved patterns')
%  print -f1 -depsc2 p4ffAM
k = sum(Xd == Yd(Xd)) ;
sprintf('Correct recalls:  %d out of %d', k, N)
spuriousPatterns = setdiff(Yd, Xd)
```

Results can be similar to the following:

```
Xd =   2   4   6  10  11

X =  0     0     0     1     1
     0     1     1     0     0
     1     0     1     1     1
     0     0     0     0     1

X = -1    -1    -1     1     1
    -1     1     1    -1    -1
     1    -1     1     1     1
    -1    -1    -1    -1     1

W_ =   1     -0.6    0.2     0.6
     -0.6      1    -0.6    -0.2
      0.2    -0.6     1     -0.2
      0.6    -0.2   -0.2      1
Xall_ =
   -1 -1 -1 -1 -1 -1 -1  1  1  1  1  1  1  1  1
   -1 -1 -1  1  1  1  1 -1 -1 -1 -1  1  1  1  1
   -1  1  1 -1 -1  1  1 -1 -1  1  1 -1 -1  1  1
    1 -1  1 -1  1 -1  1 -1  1 -1  1 -1  1 -1  1
Yb_ =
    0  0  1  0  0  0  0  1  1  1  1  0  1  1  1
    0  0  0  1  1  1  1  0  0  0  0  1  1  1  0
    0  1  1  0  0  1  0  1  0  1  1  0  0  1  1
    1  0  1  0  1  0  1  1  1  0  1  0  1  0  1
Y_ =
   -1 -1  1 -1 -1 -1 -1  1  1  1  1 -1  1  1  1
   -1 -1 -1  1  1  1  1 -1 -1 -1 -1  1  1  1 -1
   -1  1  1 -1 -1  1 -1  1 -1  1  1 -1 -1  1  1
    1 -1  1 -1  1 -1  1  1  1 -1  1 -1  1 -1  1
```

```
 X   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
 Yd  1   2  11   4   5   6   5  11   9  10  11   4  13  14  11

Xd =    2      4      6            10  11
Correct recalls:  5 out of 5
spuriousPatterns =    1      5      9     13     14
```

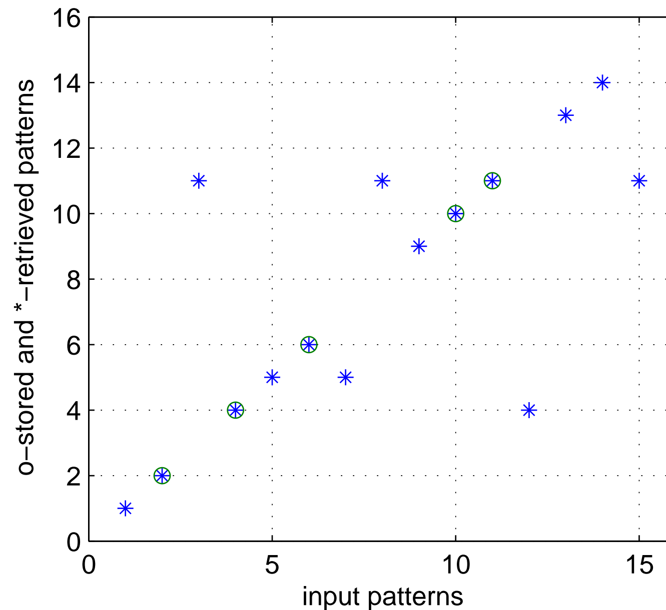


Figure 2: Encoding and decoding binary patterns in a feed-forward associative memory

Note that the pattern $\mathbf{11} = (\mathbf{1011})_\mathbf{2}$ is recalled also for patterns $\mathbf{3} = (\mathbf{0011})_\mathbf{2}$ and $\mathbf{15} = (\mathbf{1111})_\mathbf{2}$ that can be considered as a noisy version of $\mathbf{11}$ (difference on only one position), but also for $\mathbf{8} = (\mathbf{1000})_\mathbf{2}$ that differs no two positions.
There are also five spurious patterns that have not been originally encoded.

Exercise 4. 1
Run the above script for $\boldsymbol{p} = \mathbf{5}$ and different values of $\boldsymbol{N}$, ten times for each $\boldsymbol{N}$, and record the percentage of correct recalls and spurious patterns.

4.3 Recurrent Associative Memory — Discrete Hopfield networks

4.3.1 Structure

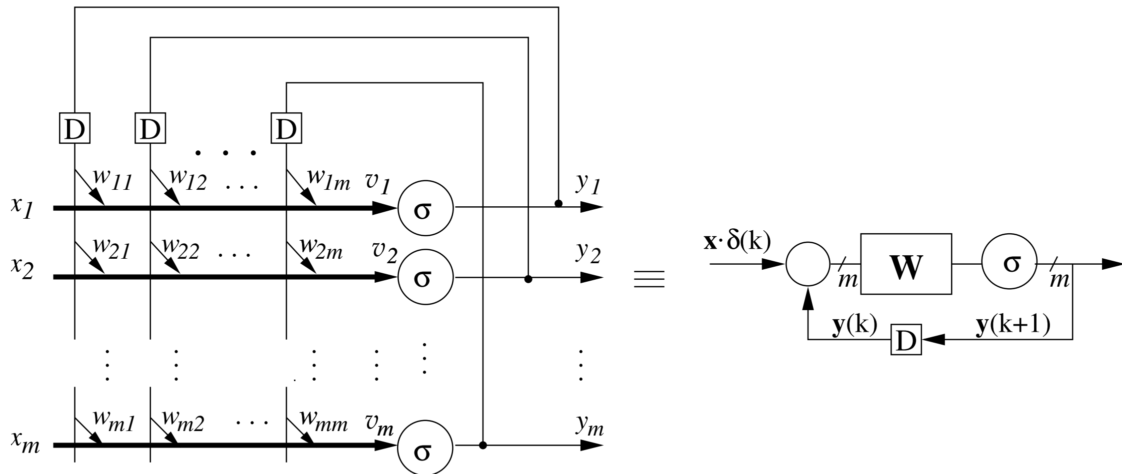


Figure 3: A dendritic and block diagram of a recurrent associative memory

- A recurrent network is built in such a way that the output signals are fed back to become the network inputs at the next time step, k
- The working of the network is described by the following expressions:

$$\mathrm{y}(k+1) = \sigma\left(W \cdot \mathrm{y}(k)\right) \; ; \quad \mathrm{y}(0) = x \text{ for } k = 0, 1, 2, \ldots \tag{6}$$

- A discrete Hopfield network is a model of an associative memory which works with binary patterns coded with $\{-1, +1\}$

  Note that if $v \in \{0, 1\}$ then $u = 2v - 1 \in \{-1, +1\}$
- The feedback signals y are often called the state signals.
- During the **storage (encoding) phase** the set of N m-dimensional fundamental memories:

$$\Xi = [\xi(1), \xi(2), \ldots, \xi(N)]$$

is stored in a matrix W in a way similar to the feedforward auto-associative memory networks, namely:

$$W = \frac{1}{m} \sum_{n=1}^{N} \xi(n) \cdot \xi(n)^T - N \cdot I_m = \frac{1}{m} \Xi \cdot \Xi^T - N \cdot I_m \tag{7}$$

By subtracting the appropriately scaled identity matrix I_m the diagonal terms of the weight matrix are made equal to zero, $(w_{jj} = 0)$. This is required for a stable behaviour of the Hopfield network.

- During the **retrieval (decoding) phase** the key vector $\mathbf{x}$ is imposed on the network as an initial state of the network

$$\mathbf{y}(0) = \mathbf{x}$$

The network then evolves towards a stable state (also called a fixed point), such that,

$$\mathbf{y}(k+1) = \mathbf{y}(k) = \mathbf{y}_s$$

It is expected that the $\mathbf{y}_s$ will be equal to the fundamental memory $\boldsymbol{\xi}$ closest to the key $\mathbf{x}$

4.3.2 Example of the Hopfield network behaviour for $m = 3$

Consider a discrete Hopfield network with three neurons as in Figure 4

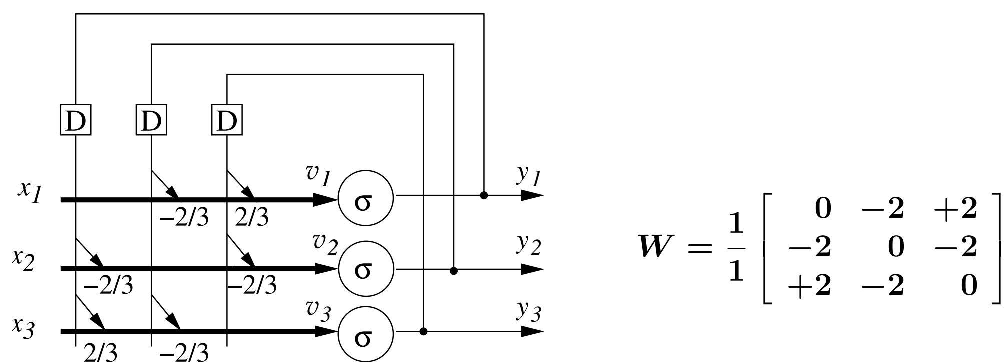


Figure 4: Example of a discrete Hopfield network with $m = 3$ neurons: its structure and the weight matrix

With $m = 3$ neurons, the network can be only in $2^3 = 8$ different states.

It can be shown (see the exercise below) that out of 8 states only two states are stable, namely: $(1, -1, 1)$ and $(-1, 1, -1)$.
In other words the network stores two fundamental memories Starting the retrieval with any of the eight possible states, the successive states are as depicted in Figure 5.

Exercise 4. 2
For the Hopfield network described in sec. 4.3.2 write a MATLAB script that generates all attractors (stable states) of the network and verify the network states presented in Figure 5

□

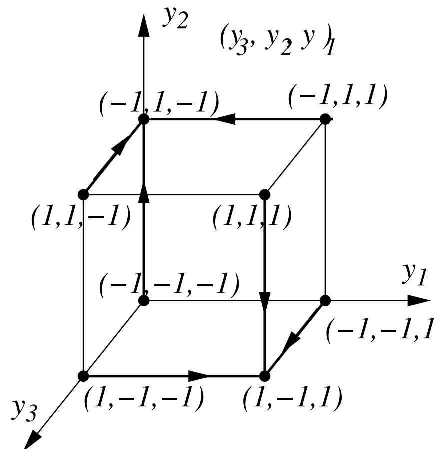


Figure 5: Evolution of states for two stable states

4.4 Simple model of Alzheimer's disease

4.4.1 Background information

Alzheimer's disease (AD) is the most common type of dementia in the elderly. The disease is characterised by a gradual loss of higher cognitive functions typically involving memory loss [1]

In our simple model of Alzheimer's disease we use a binary Hopfield recurrent neural network (as illustrated in Figure 3) as a model of human memory and the memory loss will be then modelled by gradual killing of synapses.

It must be however said that [1]

> Studies using neural network models of associative memory demonstrate very clearly that cell death cannot produce amnesia similar to that observed in AD [...] Furthermore, call death is not a consistent correlate of cognitive decline in the AD brain.

Do not be discouraged by the above statements. Our simple model was one of the first that was used to model Alzheimer's disease. The big advantage of this model is its simplicity.

4.4.2 Specification of the fundamental memories to be stored

The patterns that will be stored in synapses of the network (in the matrix W) are compositions by Tex (Swedish painter). Each composition consists of three equilateral colored triangles. The length of edges is constant and carefully measured to retain details of the painting.

Location of each triangle is specified by the (x, y) coordinates of the leftmost corner and rotation angle around that corner. The x, y resolution is 0.5mm and each coordinate is represented by an eight bit number.

Rotation angles ϕ are represented by seven bit numbers $\alpha = 90 - \phi$. The angular resolution is approximately one degree.

Each triangle is **colored** red, green or blue. The intensity for each color is given by two bits. A red triangle thus has the color vector `[3 0 0] = [1 1 0 0 0 0]`, a green triangle has the color vector `[0 3 0] = [0 0 1 1 0 0]`, a blue triangle has the color vector `[0 0 3] = [0 0 0 0 1 1]`.

Hence, each composition is coded by $m = 3(8 + 8 + 7 + 6) = 87$ bits. The network consists of m neurons, each with m synapses. In other words the weight matrix W is 87×87.

Here and there we use a complex-number notation. A complex number is a two-dimensional vector written in the form $x + jy$, where $j = \sqrt{-1}$. Do not be stunned by that: it is just a convenient way of writing a pair of coordinates x, y as a "single" number.

You can download the following script from
`http://www.csse.monash.edu.au/courseware/cse2330`

```
% trnglDefB.m
%          10 September 2004
%
clear
edg = [12 25 41];          % lengths of triangle sides in mm
vv = [0; 1; exp(j*pi/3)] ; % vertices of a unity equilateral triangle
                           % written as complex numbers
% vertices of all three triangles before rotation and translation:
vrt = vv*edg ;
m = 87 ;  % number of bits in triangle specification (number of neurons)

 %  TB set contains a variety of red, green and blue triangles
 %       x     y    deg    r  g  b
 TBd = [14   57.5   37    3  0  0
        36   55    101    0  0  3
        17.5 16.5   43    0  3  0
 %
        51   32     45    0  0  3
        25.5 14    101    0  3  0
        21   19     38    3  0  0
 %
        53  44.5   106    0  3  0
        20  10      41    3  0  0
        6   37.5    67    0  0  3
 %
        8.5 14.5   70     3  0  0
       45   46.5   80     0  3  0
       29.5 28     19     0  0  3
 %
       16    84     0     3  0  0
       21    22    60     0  0  3
       21    59.5  58     0  3  0 ]; %  15 x 6
```

```
% scans TBd row-wise and converts each number to an 8-bit string
% check first the following sequence of functions to see the structure
% Bt = dec2bin(TBd',8) , bin2dec(Bt)

Bt = (dec2bin(2*TBd',8) - '0')'; %  is is  8 x 5*3*6
B = reshape(Bt, 8*3*6, 5) ;      %  is 8*3*6 x 5
% We need only the following bits (rows of B)
k = [1:8 (1:8)+8 (1:7)+2*8 (6:7)+3*8 (6:7)+4*8 (6:7)+5*8] ;
k = [k k+48 k+2*48] ;
TB = B(k,:) ;   % is 87 x 5
% recoding each set from {0, 1}  to {-1, +1}
TB = 2*TB - 1;
```

Exercise 4. 3
Explain details of the script `trnglDefB.m`. Comment on the structure and the role of each variable

□

To see the above compositions (four first) execute the script `pltSetB.m` that you can also download from the web. To execute the script you have to type in:
`pltSetB(vrt, TB, 'can you see')`

```
% pltSetB.m
%              11 Sept 2004
function pltSetB(vrt, TB, ttl)
for k = 1:4
  Trb =  TB(:,k) ;
  Tr = reshape((Trb+1)/2, 29, 3) ;
  xx = pow2(6:-1:-1)*Tr(1:8,:) ;
  yy = pow2(6:-1:-1)*Tr((1:8)+8,:) ;
  zt = xx+j*yy  ;
  fi = pow2(6:-1:0)*Tr((1:7)+2*8,:) ;
  rot = exp(j*pi*(90-fi)/180) ;
  r = [2 1]*Tr((1:2)+23,:) ;
  g = [2 1]*Tr((1:2)+25,:) ;
  b = [2 1]*Tr((1:2)+27,:) ;
  cc = [r ; g ; b]/3 ;
  % rotation and translation of three tiangles  vrt
  z = vrt.*rot([1 1 1],:) + zt([1 1 1],:) ;
  X = real(z) ; Y = imag(z) ;
  subplot(2,2,k)
  fill(X(:,1), Y(:,1), cc(1,:), ...
       X(:,2), Y(:,2), cc(2,:), ...
       X(:,3), Y(:,3), cc(3,:))
  axis([0 80 0 80]), axis equal, grid on
  if k ==1, title(ttl), end
end
```

Exercise 4. 4
Modify the script `trnglDefB.m` into `trnglDef6.m` by adding your own three triangles.
Modify the script `pltSetB.m` into `pltSet6.m` so that you could see all six compositions.

□

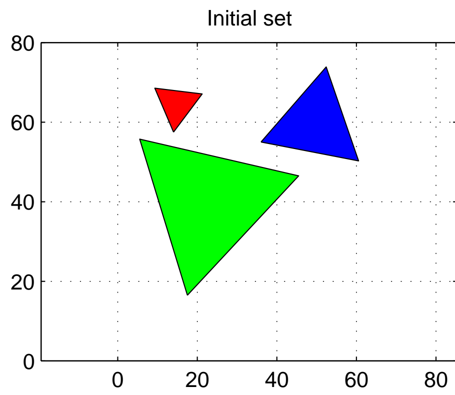
4.4.3 First implementation of the model

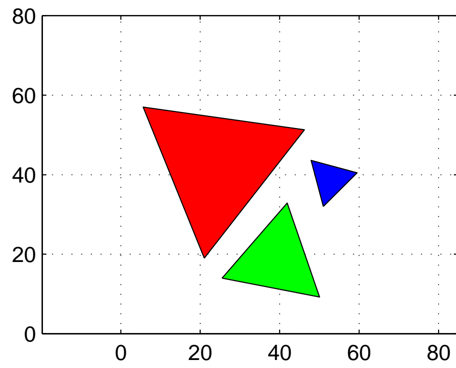
This is a demo of a simple model of the Alzheimer's disease based on the recurrent Hopfield neural network with autoassociative memory, represented by the weight matrix. The Alzheimer is modelled by "killing synapses", that is, by setting individual synaptic weights to zero.

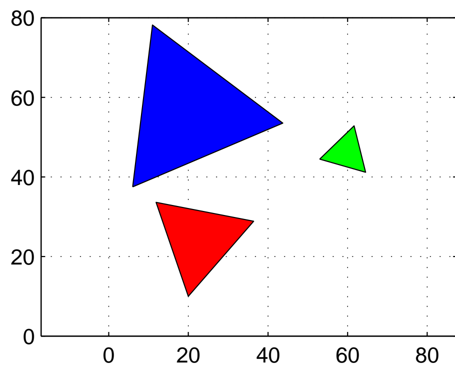
```
%  Alzh.m
%          11 September 2004
clear
trnglDefB  % definition of triangles
           % test plot of the triangle set
figure(1), clf
ttl = 'Initial set' ;
pltSetB(vrt, TB, ttl)

% matrix TB  (87 x 5) contains specification of
%       Ncomp = 5  triangle compositions, i.e,
%    five fundamental memories, each described by 87 bits
m = 87 ;
Ncomp = 5 ;
SurvF = 0.9 ;  % survival factor determines the number of killed synapses

% encoding fundamental memories
WW = (TB*TB')/Ncomp ;  %  outer products of fundamental memories

NKilledSynapses = zeros(1, Ncomp); % numbers of killed synapses
relKill = zeros(1, Ncomp) ;        % ratio of killed synapses
cntS = zeros(1, Ncomp) ;           % counters will be stored here

for k = 1:Ncomp  % composition loop
 % We now gradually destroy the memory matrix by killing a given
 % proportion of synapses. We multiply each weight by a randomly
 % generated 0 or 1
 rKill = round(SurvF*rand(size(WW)));
 W = rKill.*WW;  % a weight matrix with "killed" synapses

 % Now we want to know how many synapses we have killed.
 %   and their fraction
 NofKilledSynapses(k) = sum(sum(abs(sign(W-WW)))) ;
 relKill(k) =  NofKilledSynapses(k)/(m^2) ;

 % The Hopfield relaxation loop should converge to an attractor,
```

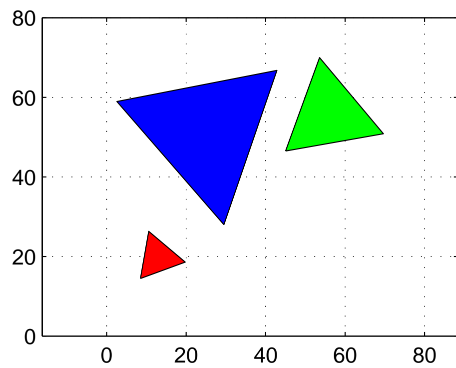
```
  % that is, should find a pattern (triangle) stored in a memory
  % matrix starting from a given initial state.
  % Naturally we may end up in a spurious attractor.
  % The memory matrix is given as W. The initial state is given
  % by a vector TB(:,k).
  % To know how long the while loop runs we also set a counter.

  cnt = 1;
  yTr = sign(W*TB(:,k));  % calculate the output pattern
                          % for the initial pattern
  dyTr = yTr;             % change in the pattern
  while (max(abs(dyTr))>0) & (cnt < 250)
      yTr22 = sign(W*yTr);
      dyTr = yTr22 - yTr ;
      yTr = yTr22 ;
      cnt = cnt+1 ;
  end

  % the attractors yTr are stored in a matrix Attr
  Y(:, k) = yTr ;

  % the counter is also stored
  cntS(k) = cnt ;

end

% Now we calculate the number of vector elements where
% the attractors and the compositions differ.
NofDiffElements = sum(abs(sign(TB(:,1:Ncomp)- Y))) ;

 figure(2), clf
 ttl = 'Final set' ;
 pltSetB(vrt, Y, ttl)  % plotting the decoded set

  ['# Killed synapses:  ',  ...
  num2str(NofKilledSynapses), sprintf('\n'),  ...
 'Fraction of killed synapses:  ',  ...
  num2str(relKill, '%1.2f '), sprintf('\n'),  ...
 '# Different Elements:  ',  ...
  num2str(NofDiffElements), sprintf('\n'),  ...
 '# Relaxation runs:  ',  num2str(cntS)]
 % print -f1 -depsc2 AlzhInit
 % print -f2 -depsc2 AlzhFinal
```

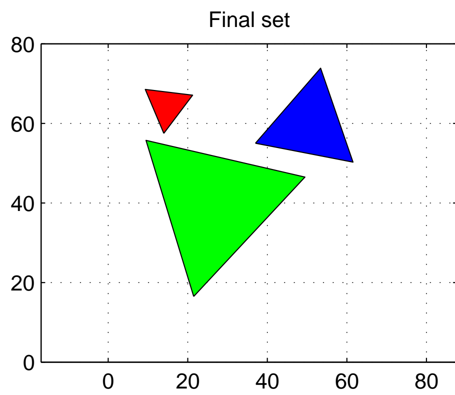
The script produces figures similar to the following:

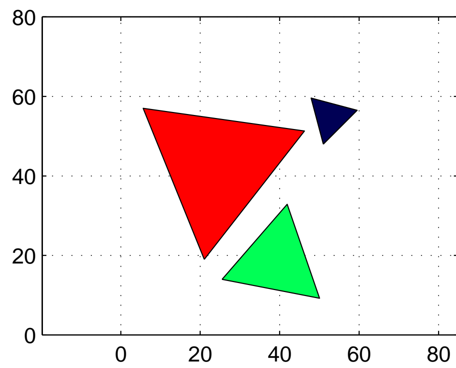

Initial set
80
60
40
20
0
0
20
40
60
80

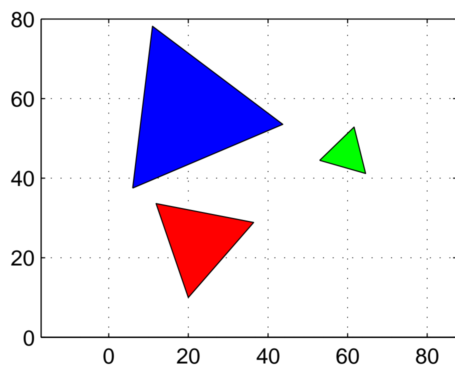

80
60
40
20
0
0
20
40
60
80

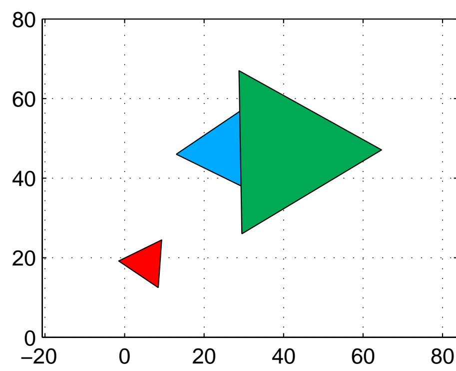

80
60
40
20
0
0
20
40
60
80


80
60
40
20
0
0
20
40
60
80


Final set
80
60
40
20
0
0
20
40
60
80


80
60
40
20
0
0
20
40
60
80


80
60
40
20
0
0
20
40
60
80


80
60
40
20
0
−20
0
20
40
60
80

Exercise 4. 5
Run the above script a number of times and record in a table: `SurvF`, `NofKilledSynapses`, `relKill`, `cntS`. Comment on the obtained results.
□

4.4.4 Second implementation of the model

Second implementation of the model follows the same principle of operation. For convenience there is a Graphical User Interface (GUI) built that makes running the script easier.

Download the following scripts: `AlzhGUI.m, AlzhGUI.fig, trnglDef.m, HopfConv.m, plotSet.m`

Invoke the main script `AlzhGUI.m` that should open new window in which you can set up parameters run a simulation.

Exercise 4. 6
Run simulations for different triangle sets and different parameters and record the results in a table as in the previous exercise.
Comment on the obtained results.
□

Written Submission

Your written submission should include results of all exercises you have performed (relevant MATLAB scripts, figures, numerical results, etc) with brief comments and explanations.
It should be in a form ready for electronic submission when requested.

References

[1] David H. Small, "Do acetylcholinesterase inhibitors boost synaptic scaling in Alzheimer's disease," *TRENDS in Neuroscience*, Vol. 27, No. 5, May 2004