Character recognition with Hopfield networks

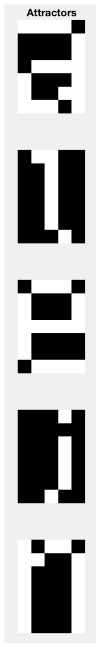
Alejandro Rodríguez Salamanca - r0650814 - Erasmus

Data preparation

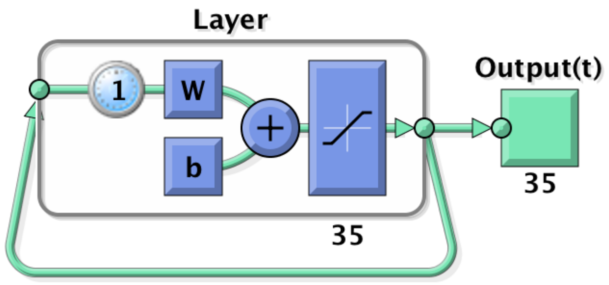
Before starting to solve the tasks, the dataset to use must be constructed. For this exercise, we are asked to pre-pend the lowercase characters of our first and last name to the set of capital characters given in `prprob`. As my name is Alejandro Rodriguez, the characters I have to pre-pend are: `a, l, e, j, n, d, r, o, i, g, u, z`. Once this is done, it's time to solve the three proposed problems.

Task 1: Retrieve the first 5 characters.

The first 5 characters of the alphabet are `a, l, e, j` and `n`, this is, the first five characters that appear in my first name and last name. After loading this characters, the pixels whose value is 0 must be changed to -1. This has to be done because Hopfield networks normally have units that take on values of 1 or -1 as convention.

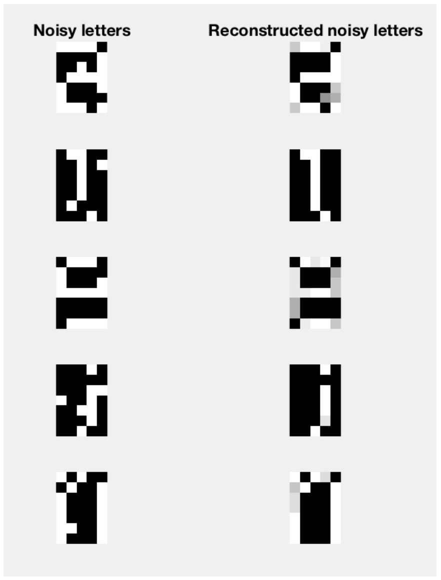


With this five characters as attractor states, a Hopfield neural network is created. Then, three random pixels of each character are inverted (from 1 to -1 and vice versa). The objective of this inversion is to check if the network is able to reconstruct the distorsioned characters.

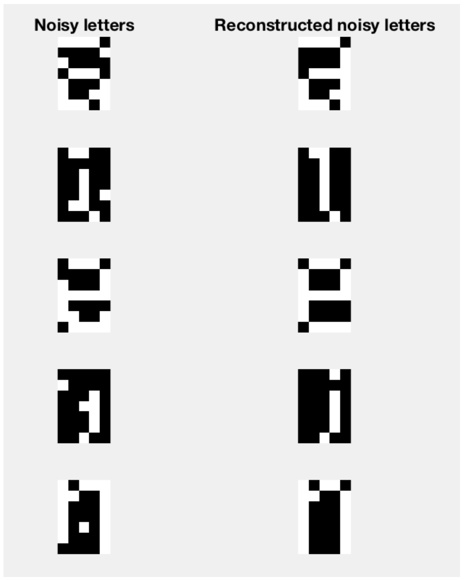


Executing the neural network with the noisy characters as input shows the following results:

- With a small number of steps: it can be seen that the characters are being attracted by the attractor states, but they still contain some noise.



- When the number of steps is increased: with just 5 steps, the characters are perfectly reconstructed.



Sporious patterns are local energy minima that are created during training, in addition to the intended target patterns. They are activity patterns that have not been explicitly embedded in the synaptic matrix, but are nonetheless stable. They are in other words "unwanted" attractor states that, by virtue of a finite overlap with the "wanted" attractor states, come about as a local minimum in the energy function. They can be composed of various combinations of the original patterns or simply the negation of any pattern in the original pattern set.

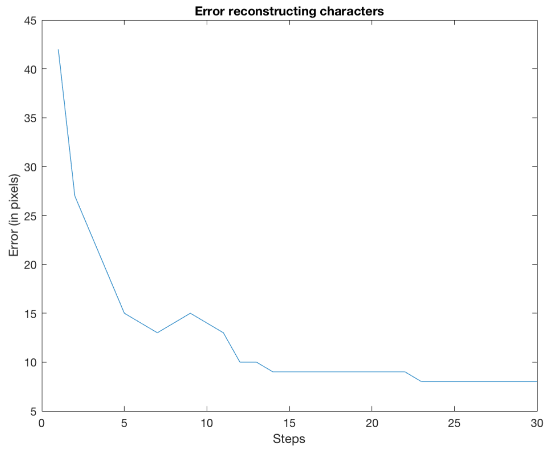
There are three different types of spurious states:

1. For each stored pattern, its negation is also an attractor
2. Any linear combination of an odd number of stored patterns. give rise to the so-called mixture states. Note that Spurious patterns that have an even number of states cannot exist, since they might sum up to zero.
3. for large p, we get local minima that are not correlated to any linear combination of stored patterns.

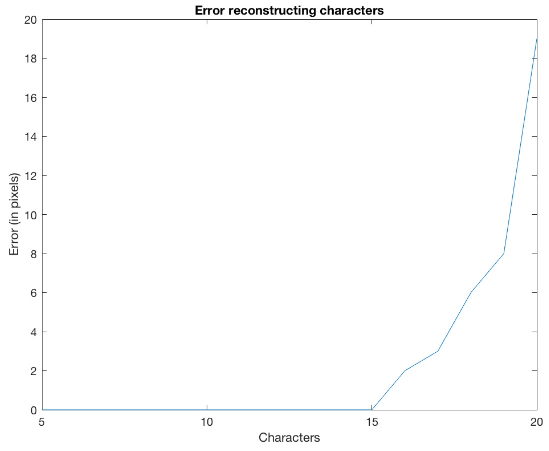
In this case, the existence of spurious patters is not noticeable. According to Hebb rule, a Hopfield network can store up to $0.138N$ uncorrelated patterns, being N the number of neurons. Replacing N by its value in this problem, 35, gives that the uncorrelated patterns that can be stored is almost 5, which is the number of charaters used in this exercise.

Task 2: Critical loading capacity

A Hopfield neural network exceed the loading capacity when the number of learned patterns over the number of units p/N is greater than the critical capacity $\alpha \approx 0.138$. First, a number $p = 20$ characters is selected, and the error in the reconstruction is calculated as a function of the number of steps used to reconstruct the character.



The purpose of this plot is to see how the approximation becomes more accurate when the number of steps is increased. Now, let's see what happens when the number of characters varies while keeping the number of steps fixed at 15.



The critical capacity of a Hopfield network, as stated before, is p/N, where p is the set of patterns to be memorized and N the number of neurons. The number of neurons is determined by the number of pixels in each image. The dimension of each image is 7×5, which means that the number of neurons of the network is 35.

Now, the critical capacity can be expressed as $p/35$.

It can be seen in the previous plot that when the number of patterns p has values below 15, there is no error in the reconstruction, but when it is increased, an error in the reconstruction appears.

The loading capacity can be expressed then as $15/35 = 0.428$, which is much higher than the one obtained using Hebb-rule.

This is an interesting phenomena. Nevertheless, it has an explanation. It could happen that the spurious states and the real attractors overlap, which means that the reconstruction of the character will be correct even if the pixel is reconstructed by an spurious state. Hebb rule states that it can be stored $0.138N$ uncorrelated patterns, which in this case would be $0.138 \times 35 \approx 5$ patterns, as aforementioned. It can be then concluded that the charaters are not uncorrelated patterns.

Task 3: Retrieve 25 characters

As it can be seen previously, Hopfield networks works well when the number of stored patterns is small, but when it increases, spurious states appears and the images cannot be reconstructed. For this reason, a different approach should be used.

As we already know from previous assignments, a feedforward network with one hidden layer and enough neurons in the hidden layers, can fit any finite input-output mapping problem. But some changes should be done to use the data with a feedforward network.

When using a feedforward network, a set of targets and data is needed. The data corresponds to the digits previously used, but it is neccessary to assign a class to each of the digits. It can be done creating an indentity matrix whose dimensions are the number of digits, 25 in this case.

Different architectures and parameters where used, but in some cases the network where not able to identify correctly the characters. There are different techniques to improve the performance and accuracy of a neural network, but the simplest and most effective one is to generate more training data. As we are trying to idenfify noisy characters, is it possible to generate a set of noisy characters and append them to the original ones. Now, our network not only learns with the original characters, but also with noisy characters similar to the ones it will have to clasify later.

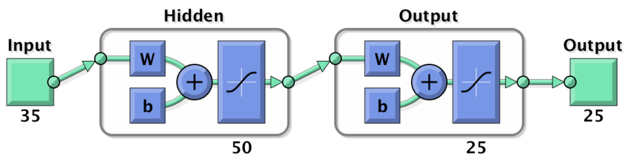
As expected, the network learnt to clasify better the digits, but the results showed where not perfect. It happened that depending on which pixels were inverted, two different characters can become almost equal, as happens with G and C.

At this point, the obvious idea is to generate more noisy characters and add them to our trainig data set. It could be an option, but we take the risk that our network does not learn to generalize.

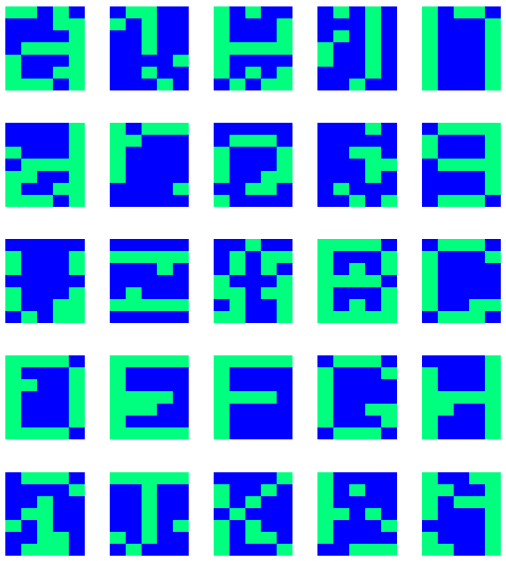
Can we do anything else to improve the accuracy of a neural network? Fortunately, yes, we can. Two common techniques to improve generalization and avoid overfitting are **retraining the neural network** and **using multiple neural networks**.

The approach used consists of use multiple neural networks and average their outputs. This technique is really useful when the error in the classification is caused by noisy data or a small dataset, which is exactly this case.

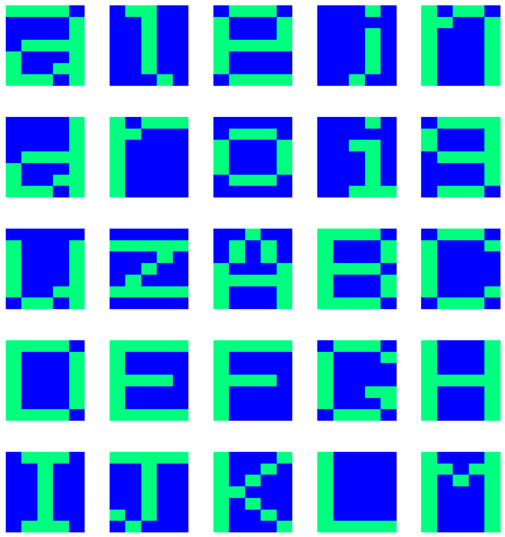
To solve this problem, ten feedfordward neural networks have been used, with 50 neurons in the hidden layer, `tansig` transfer function for the output layer, and bayesian regularization backpropagation for training.



The distorsioned data set of the first 25 characters can be seen in the following image:



And finally, it is perfectly reconstructed:



Conclusion

After solving the reconstruction of noisy digits problem using two different approaches, it's time to discuss the advantages and drawbacks of each one.

Using Hopfield networks to reconstruct noisy characters is a good idea when the number of patters to store in the network, this is, the number of characters is small. Once the number of stored patterns grows, spurious states appers, and the accuracy of the network decreases. In fact, as it have been seen before, the maximum theoretical number of patters the network can store is $N\frac{2}{\times}lnN$, being N the number of neurons. However, Hopfield networks are fast when training, so if training speed is a bottleneck in our problem, using this type of network can be the solution.

The second approach, feedfordward networks, is different from the first one in the sense that this network does not reconstruct the character, but identifies which one has been received as input. We have seen that this networks can fit any finite input-output mapping problem, which means that if the trainig data set is good enough and the number of neurons in the hidden layer is high, we can *reconstruct* any set of characters, independently of its size, and if the training data set is small, other techniques can be used, as previously seen in this assignment. The main problem with this type of network is that it needs a lot of time for training, unlike Hopfield networks.

It can be concluded that feedforward netwoks are more suitable for pattern recognition than Hopfield networks if the main concern is accuracy in the classification.

Appendix

gen_noisy_chars.m

```
function[noisy_alphabet] = gen_noisy_chars(alphabet)

    noisy_alphabet = alphabet;
    for i = 1:size(noisy_alphabet, 2)
        %get three random numbers between 1 and 35 (number of px in each
        %imgage
        rands = randperm(length(noisy_alphabet),3);
        for j = 1:3
            if noisy_alphabet(rands(j), i) == 1
                noisy_alphabet(rands(j), i) = -1;
            else
                noisy_alphabet(rands(j), i) = 1;
            end
        end
    end
end
```

get_alphabet.m

```
function [alphabet, target] = get_alphabet()

lettera =  [1 1 1 1 0 ...
            0 0 0 0 1 ...
            0 0 0 0 1 ...
            0 1 1 1 1 ...
            1 0 0 0 1 ...
            1 0 0 1 1 ...
            1 1 1 0 1 ]';

letterl =  [0 1 1 0 0 ...
            0 0 1 0 0 ...
            0 0 1 0 0 ...
            0 0 1 0 0 ...
            0 0 1 0 0 ...
            0 0 1 0 0 ...
            0 0 0 1 0 ]';

lettere =  [0 1 1 1 0 ...
            1 0 0 0 1 ...
            1 0 0 0 1 ...
            1 1 1 1 1 ...
            1 0 0 0 0 ...
            1 0 0 0 0 ...
            0 1 1 1 1 ]';

letterj =  [0 0 0 1 0 ...
            0 0 0 0 0 ...
            0 0 0 1 0 ...
            0 0 0 1 0 ...
            0 0 0 1 0 ...
            0 0 0 1 0 ...
```

```
                 0 0 1 0 0 ]';

letter n =   [1  0 1 1 0 ...
              1 1 0 0 1 ...
              1 0 0 0 1 ...
              1 0 0 0 1 ...
              1 0 0 0 1 ...
              1 0 0 0 1 ...
              1 0 0 0 1 ]';

letterd =   [0  0 0 0 1 ...
              0 0 0 0 1 ...
              0 0 0 0 1 ...
              0 1 1 1 1 ...
              1 0 0 0 1 ...
              1 0 0 1 1 ...
              1 1 1 0 1 ]';

letterr =   [1  0 1 1 1 ...
              1 1 0 0 0 ...
              1 0 0 0 0 ...
              1 0 0 0 0 ...
              1 0 0 0 0 ...
              1 0 0 0 0 ...
              1 0 0 0 0 ]';

lettero =   [0  0 0 0 0 ...
              0 1 1 1 0 ...
              1 0 0 0 1 ...
              1 0 0 0 1 ...
              1 0 0 0 1 ...
              0 1 1 1 0 ...
              0 0 0 0 0 ]';

letteri =   [0  0 0 1 0 ...
              0 0 0 0 0 ...
              0 0 1 1 0 ...
              0 0 0 1 0 ...
              0 0 0 1 0 ...
              0 0 0 1 0 ...
              0 0 1 1 1 ]';

letterg =   [0  1 1 1 1 ...
              1 0 0 0 1 ...
              1 0 0 0 1 ...
              0 1 1 1 1 ...
              0 0 0 0 1 ...
              0 0 0 0 1 ...
              0 1 1 1 0 ]';

letteru =   [0  0 0 0 0 ...
              1 0 0 0 1 ...
              1 0 0 0 1 ...
              1 0 0 0 1 ...
              1 0 0 0 1 ...
              1 0 0 1 1 ...
```

```
                0 1 1 0 1 ]';

letterz =  [0 0 0 0 0 ...
            1 1 1 1 1 ...
            0 0 0 1 0 ...
            0 0 1 0 0 ...
            0 1 0 0 0 ...
            1 1 1 1 1 ...
            0 0 0 0 0 ]';


letterA =  [0 0 1 0 0 ...
            0 1 0 1 0 ...
            0 1 0 1 0 ...
            1 0 0 0 1 ...
            1 1 1 1 1 ...
            1 0 0 0 1 ...
            1 0 0 0 1 ]';

letterB =  [1 1 1 1 0 ...
            1 0 0 0 1 ...
            1 0 0 0 1 ...
            1 1 1 1 0 ...
            1 0 0 0 1 ...
            1 0 0 0 1 ...
            1 1 1 1 0 ]';

letterC =  [0 1 1 1 0 ...
            1 0 0 0 1 ...
            1 0 0 0 0 ...
            1 0 0 0 0 ...
            1 0 0 0 0 ...
            1 0 0 0 1 ...
            0 1 1 1 0 ]';

letterD  = [1 1 1 1 0 ...
            1 0 0 0 1 ...
            1 0 0 0 1 ...
            1 0 0 0 1 ...
            1 0 0 0 1 ...
            1 0 0 0 1 ...
            1 1 1 1 0 ]';

letterE  = [1 1 1 1 1 ...
            1 0 0 0 0 ...
            1 0 0 0 0 ...
            1 1 1 1 0 ...
            1 0 0 0 0 ...
            1 0 0 0 0 ...
            1 1 1 1 1 ]';

letterF =  [1 1 1 1 1 ...
            1 0 0 0 0 ...
            1 0 0 0 0 ...
            1 1 1 1 0 ...
            1 0 0 0 0 ...
```

```
                 1 0 0 0 0 ...
                 1 0 0 0 0 ]';

letterG =   [0 1 1 1 0 ...
                 1 0 0 0 1 ...
                 1 0 0 0 0 ...
                 1 0 0 0 0 ...
                 1 0 0 1 1 ...
                 1 0 0 0 1 ...
                 0 1 1 1 0 ]';

letterH =   [1 0 0 0 1 ...
                 1 0 0 0 1 ...
                 1 0 0 0 1 ...
                 1 1 1 1 1 ...
                 1 0 0 0 1 ...
                 1 0 0 0 1 ...
                 1 0 0 0 1 ]';

letterI =   [0 1 1 1 0 ...
                 0 0 1 0 0 ...
                 0 0 1 0 0 ...
                 0 0 1 0 0 ...
                 0 0 1 0 0 ...
                 0 0 1 0 0 ...
                 0 1 1 1 0 ]';

letterJ =   [1 1 1 1 1 ...
                 0 0 1 0 0 ...
                 0 0 1 0 0 ...
                 0 0 1 0 0 ...
                 0 0 1 0 0 ...
                 1 0 1 0 0 ...
                 0 1 0 0 0 ]';

letterK =   [1 0 0 0 1 ...
                 1 0 0 1 0 ...
                 1 0 1 0 0 ...
                 1 1 0 0 0 ...
                 1 0 1 0 0 ...
                 1 0 0 1 0 ...
                 1 0 0 0 1 ]';

letterL =   [1 0 0 0 0 ...
                 1 0 0 0 0 ...
                 1 0 0 0 0 ...
                 1 0 0 0 0 ...
                 1 0 0 0 0 ...
                 1 0 0 0 0 ...
                 1 1 1 1 1 ]';

letterM =   [1 0 0 0 1 ...
                 1 1 0 1 1 ...
                 1 0 1 0 1 ...
                 1 0 0 0 1 ...
                 1 0 0 0 1 ...
```

```
                1 0 0 0 1 ...
                1 0 0 0 1 ]';

letterN =   [1 0 0 0 1 ...
             1 1 0 0 1 ...
             1 1 0 0 1 ...
             1 0 1 0 1 ...
             1 0 0 1 1 ...
             1 0 0 1 1 ...
             1 0 0 0 1 ]';

letterO =   [0 1 1 1 0 ...
             1 0 0 0 1 ...
             1 0 0 0 1 ...
             1 0 0 0 1 ...
             1 0 0 0 1 ...
             1 0 0 0 1 ...
             0 1 1 1 0 ]';

letterP =   [1 1 1 1 0 ...
             1 0 0 0 1 ...
             1 0 0 0 1 ...
             1 1 1 1 0 ...
             1 0 0 0 0 ...
             1 0 0 0 0 ...
             1 0 0 0 0 ]';

letterQ =   [0 1 1 1 0 ...
             1 0 0 0 1 ...
             1 0 0 0 1 ...
             1 0 0 0 1 ...
             1 0 1 0 1 ...
             1 0 0 1 0 ...
             0 1 1 0 1 ]';

letterR =   [1 1 1 1 0 ...
             1 0 0 0 1 ...
             1 0 0 0 1 ...
             1 1 1 1 0 ...
             1 0 1 0 0 ...
             1 0 0 1 0 ...
             1 0 0 0 1 ]';

letterS =   [0 1 1 1 0 ...
             1 0 0 0 1 ...
             0 1 0 0 0 ...
             0 0 1 0 0 ...
             0 0 0 1 0 ...
             1 0 0 0 1 ...
             0 1 1 1 0 ]';

letterT =   [1 1 1 1 1 ...
             0 0 1 0 0 ...
             0 0 1 0 0 ...
             0 0 1 0 0 ...
             0 0 1 0 0 ...
             0 0 1 0 0 ...
             0 0 1 0 0 ]';

letterU =   [1 0 0 0 1 ...
             1 0 0 0 1 ...
             1 0 0 0 1 ...
             1 0 0 0 1 ...
             1 0 0 0 1 ...
             1 0 0 0 1 ...
             0 1 1 1 0 ]';

letterV =   [1 0 0 0 1 ...
             1 0 0 0 1 ...
             1 0 0 0 1 ...
             1 0 0 0 1 ...
             1 0 0 0 1 ...
             0 1 0 1 0 ...
             0 0 1 0 0 ]';

letterW =   [1 0 0 0 1 ...
             1 0 0 0 1 ...
             1 0 0 0 1 ...
             1 0 0 0 1 ...
             1 0 1 0 1 ...
             1 1 0 1 1 ...
             1 0 0 0 1 ]';

letterX =   [1 0 0 0 1 ...
             1 0 0 0 1 ...
             0 1 0 1 0 ...
             0 0 1 0 0 ...
             0 1 0 1 0 ...
             1 0 0 0 1 ...
             1 0 0 0 1 ]';

letterY =   [1 0 0 0 1 ...
             1 0 0 0 1 ...
             0 1 0 1 0 ...
             0 0 1 0 0 ...
             0 0 1 0 0 ...
             0 0 1 0 0 ...
             0 0 1 0 0 ]';

letterZ =   [1 1 1 1 1 ...
             0 0 0 0 1 ...
             0 0 0 1 0 ...
             0 0 1 0 0 ...
             0 1 0 0 0 ...
             1 0 0 0 0 ...
             1 1 1 1 1 ]';

alphabet = [lettera, letterl, lettere, letterj, lettern, letterd, letterr, lettero, ...
             letteri, letterg, letteru, letterz, letterA,letterB,letterC,letterD,...
             letterE,letterF,letterG,letterH,...
             letterI,letterJ,letterK,letterL,letterM,letterN,letterO,letterP,...
```

```
            letterQ, letterR, letterS, letterT, letterU, letterV, letterW, letterX, ...
            letterY, letterZ];


alphabet(alphabet == 0) = -1;

target = eye(length(alphabet));

end
```

show_chars.m

```
function show_chars(num_letters, chars, desc, ind)

if ind == 1
    figure;
end

subplot(num_letters, 3, ind);

    for i = 1:num_letters
        letter = chars(i,:);
        letter = reshape(letter,5,7)';

        subplot(5,5,(i));
        colormap('winter')
        imagesc(letter)
        hold on
        axis off
    end
end
```

ex2_1.m

```
% Build a Hopfield neural network that is capable of retrieving the first 5
% of these characters. Distort each character by inverting three randomly
% chosen pixels (so you change 1 to -1 and viceversa) and check if the network
% is able to recall these distorted patterns. Discuss the existence of spurious
    patterns.
[alphabet, ~] = get_alphabet();

a = alphabet(:, 1)';
l = alphabet(:, 2)';
e = alphabet(:, 3)';
j = alphabet(:, 4)';
n = alphabet(:, 5)';

num_letters = 5;
index_dig = [1,2,3,4,5];
T = [a;l;e;j;n]';
%Create network
net = newhop(T);

%Check if digits are attractors
```

```
[Y,~,~] = sim(net,num_letters,[],T);

show_chars(num_letters, Y','Attractors', 1);

Xn = gen_noisy_chars(T);

%Show noisy digits:
noisy_a = Xn(:, 1)';
noisy_l = Xn(:, 2)';
noisy_e = Xn(:, 3)';
noisy_j = Xn(:, 4)';
noisy_n = Xn(:, 5)';

Tn = [noisy_a; noisy_l; noisy_e; noisy_j; noisy_n];

subplot(num_letters,3,2);
show_chars(num_letters, Tn, 'Noisy letters', 2);

num_steps = 8;

Tn = Tn';
[Yn,~,~] = sim(net,{num_letters num_steps},{},Tn);
Yn = Yn{1,num_steps};
Yn = Yn';

subplot(num_letters,3,3);

show_chars(num_letters, Yn, 'Reconstructed noisy letters', 3);
```

ex2_2.m

```
% In this part you must determine the critical loading capacity of the
% network. Choose a number P, and store P characters from your collection
% in a Hopfield neural network. Distort these characters by inverting three
% randomly chosen pixels and try to retrieve them. Do this by iterating the
% network for a chosen number of iterations and clipping (rounding) the
% pixels of the final state to +1 and -1. Calculate the error for several
% values of P and plot this error as a function of P. The error is the total
% number of wrong pixels over all the retrieved characters. Estimate the
% critical loading capacity. Compare your estimated capacity to the
% theoretical loading capacity of the Hopfield neural network that uses the
% Hebb-rule for uncorrelated patterns. Discuss the existence of spurious patterns.

[alphabet, ~] = get_alphabet();

letters = 5:20;
error = [];
for num_letters = letters
    index_dig = 1:num_letters;

    T = [];
    for i=1:num_letters
        T = vertcat(T, alphabet(:, i)');
    end
    T = T';
    %Create network
    net = newhop(T);

    %Check if digits are attractors
    [Y,~,~] = sim(net,num_letters,[],T);

    %show_chars(num_letters, Y', 'Attractors', 1);

    Xn = gen_noisy_chars(T);

    %Show noisy digits:
    Tn = [];
    for i=1:num_letters
        Tn = vertcat(Tn, Xn(:, i)');
    end

    %show_chars(num_letters, Tn, 'Noisy letters', 2);

    Tn = Tn';
    step = 30;
    T = T';

        [Yn,~,~] = sim(net,{num_letters step},{},Tn);
        Yn = Yn{1,step};
        Yn = Yn';

        %show_chars(num_letters, Yn, 'Reconstructed noisy letters', 3);

        Yn(Yn < 0) = -1;
        Yn(Yn >= 0) = 1;

        diffs = sum(sum((T == Yn) == 0));
        error = [error, diffs];
end

view(net);
figure;
plot(letters, error);
title('Error reconstructing characters');
xlabel('Characters');
ylabel('Error (in pixels)');
```

ex2_3.m

```
% What options would you have when you need to store 25 characters with the
% matlab routines in such a way that characters with three random inversions
% can be retrieved perfectly? Demonstrate a method using the first 25
% characters of your collection as a basis.

num_letters = 25;
error = [];

% get original alphabet and targets for training
[alphabet, target] = get_alphabet();
T = target(1:num_letters, 1:num_letters);
P = alphabet(:, 1:num_letters)';

```

```
% get noisy alphabet and targets for training
[noisy_alphabet, noisy_target] = get_alphabet();
noisy_alphabet = gen_noisy_chars(noisy_alphabet')';
noisy_T = noisy_target(1:num_letters, 1:num_letters);

P = vertcat(P, noisy_alphabet(:, 1:num_letters)');
T = vertcat(T, noisy_T);

%Create network
numNN = 10;
numN = [50];
nets = cell(1, numNN);
trainAlg = 'trainbr';
net = feedforwardnet(numN, trainAlg);
net.layers{2}.transferFcn = 'tansig';
net.trainParam.epochs = 5000;

for i = 1:numNN
  fprintf('Training %d/%d\n', i, numNN)
  nets{i} = train(net, P', T');
end

Pn = gen_noisy_chars(alphabet');
Pn = Pn(1:num_letters, :);

Y_total = 0;
for i = 1:numNN
    neti = nets{i};
    Y = sim(neti, Pn');
    Y_total = Y_total + Y;
end

Y_avg = Y_total / numNN;
Y_avg = compet(Y_avg);

output = [];
for i = 1:size(Y_avg,2)
     index = find(Y_avg(:,i));
     output(:,i) = alphabet(:,index);
end

show_chars(num_letters, alphabet(:, 1:num_letters)', 'Original letters', 1);
show_chars(num_letters, Pn, 'Noisy letters', 2);
show_chars(num_letters, output', 'Reconstructed noisy letters', 3);

length(find(P(1:num_letters, :) ~= output'))
```