Neural Networks Lab 3

Dekel Viner(S2612925),Luis Knigge(S2560224)

September 3, 2019

1 Questions on the Hopfield implementation

a) When Hebb's rule is applied repetitively what occurs is that two or more neurons that are fired together repetitively will have the strength of their bonds increase to the point that the firing of one neuron will lead to the firing of the other. To put it in terms of artificial neurons we can say that the weights between the neurons will be as such that the firing of one neuron will lead to the firing of the other. When using the formula provided in the question we set the weights as such that the weight between two neurons will be set so that the firing of one neuron will lead to the desired output the neurons connected to it.

b) When we omit the normalization which simply divides the weights obtained by the weight function by 3 as the weight function sum three connections. However the network still works because the threshold is set at zero. any activation value being above 0 with normalized weight would simply become a larger value above 0 without normalization, similarly it goes for numbers below zero. Meaning that the final activation output will remain the same.

c) for m equals the maximum number of patterns that can be stored and N equals the number of nodes.
$m < f(N) = N/(2ln(N))$

d) $m < f(25) = 25/(2ln(25))$
$m < f(25) = 3.88$
$m = 3$

e) When varying the n_examples we can see that that for n_examples equals 3 the network works perfectly fine and is able identify the letters. However when we set the n_examples to 4 or higher the network fails as we expected.

2 Noise and sprouts

a) The Hopfield network is able to recover the original pictures because the weights that we computed represent the patterns as a state that has a local minimum of energy. By updating the network the energy goes down and there for to a state which was stored in the weights before.

b) The noise level effects the synchronous updating a lot more than the asynchronous updating because the synchronous updating does a lot less steps then the asynchronous updating. The latter method updates each node on its own while the other method does them all at once. This means that the noise is not so much visible anymore in the asynchronous updating since it has done more updating steps which decrease the effect of noise.

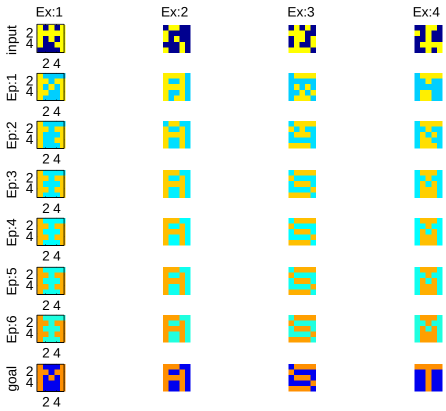
c) The colors of the input pictures seem to be inverted but the learned weights are still the same. This results in the output being inverted as well.

d) The Hopfield network stores the inverted version of the original pattern as well because the weights are just signaling for each pixel if it is supposed to be different or the same in comparison to all the other pixels. Consider a picture with just two pixels and the pixel at position (1,1) is turned on in the pattern and pixel (1,2) is turned of. The weights between them are -1 then. This means if you have an input where the first pixel is turned of, the weight indicates that the other pixel is different from the first one and is therefore turned on. This results in the inverted picture to the original picture.

3 Code

hopfield.m

```
% Clear workspace and close all previous windows
clear all;
close all;

% Initializing data and parameters
% PARAMETERS
n_examples = 3;              % The number of examples(0 <
    n_examples < 7)
n_epochs = 6;                % The number of epochs
normalize_weights = true;    % Normalization bool

random_percentage = 25;      % Percentage of bits that are flipped
    randomly
invert = false;              % Invert the input (test for spurious
    states)


% Do not change these lines
dim_x = 5;                   % Dimensions of examples
dim_y = 5;

```




```
% Compute size of examples
size_examples = dim_x * dim_y;

% Convert percentage to fraction
random_percentage = random_percentage/100;

% Set color for network plots
color = 20;

% The data is stored in .dat files. They have to be located in the same
% directory as this source file
data = importdata('M.dat');
```

```matlab
data(:,:,2) = importdata('A.dat');
data(:,:,3) = importdata('S.dat');
data(:,:,4) = importdata('T.dat');
data(:,:,5) = importdata('E.dat');
data(:,:,6) = importdata('R.dat');

% Convert data matrices into row vectors. Store all vectors in a matrix
vector_data = zeros(n_examples,size_examples);
for idx = 1:n_examples
    vector_data(idx,:) = reshape(data(:,:,idx)',1,size_examples);
end


% TRAINING THE NETWORK
% The network is trained using one-shot Hebbian learning

% The result should be a matrix dimensions: size_examples * size_examples
% Each entry should contain a sum of n_exampless
weights = vector_data' * vector_data;

% A hopfield neuron is not connected to itself. The diagonal of the matrix
% should be zero.
for x = 1:length(vector_data)
    weights(x,x) = 0;
end

% These lines check whether the matrix is a valid weight matrix for a
% Hopfield network.
assert(isequal(size(weights),[size_examples size_examples]), ...
    'The matrix dimensions are invalid');
assert(isequal(tril(weights)',triu(weights)), ...
    'The matrix is not symmetric');
assert(isempty(find(diag(weights), 1)), ...
    'Some neurons are connected to themselves');

% Normalizing the weights
if normalize_weights
    weights = weights ./ n_examples;
end


% PLOT WEIGHT MATRIX
figure(1)
imagesc(weights)
colorbar
```

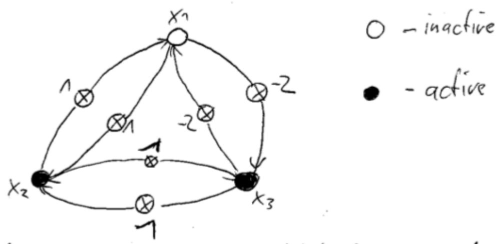
```
title('Weight matrix')

% INTRODUCE NOISE

% Copy the input data
input = vector_data;

% Create a matrix with the same dimensions as the input data in which
% random_percentage elements are set to -1 and the others are set to 1.
% We do this by sampling from a normal distribution
noise_matrix = (randn(size(input)) > norminv(random_percentage));
noise_matrix = noise_matrix - (noise_matrix==0);

% Flip bits (* -1) using the noise_matrix
input = input .* noise_matrix;

% Optionally invert the input
if invert
    input = -1 .* input;
end

% PLOTTING INPUT PATTERNS
figure(2)
for example = 1:n_examples
    subplot(n_epochs + 2,n_examples,example)
    test = reshape((input(example,:)),dim_x, dim_y)';
    image(test .* color + color)
    str = 'Ex:';
    str = strcat(str,int2str(example));
    title(str)
    if(example == 1)
        axis on
        ylabel('input')
    else
        axis off
    end
    axis square
end

% UPDATING THE NETWORK
% Feed the network with all of the acquired inputs. Update the network and
% plot the activation after each epoch.
for example = 1:n_examples

    % The initial activation is the row vector of the current example.
```

```
    activation = input(example,:)';

    for epoch = 1:n_epochs
        % Compute the new activation
        activation = weights * activation;

        % Apply the activation function
        for x = 1:length(activation)
            if(activation(x) >= 0)
                activation(x) = 1;
            else
                activation(x) = 0;
            end
        end
        % PLOTTING THE ACTIVATION

        % Reshape the activation such that we get a 5x5 matrix
        output = reshape(activation, dim_x, dim_y)';

        % Compute the index of where to plot
        idx = epoch * n_examples + example;

        % Create the plot
        subplot(n_epochs + 2,n_examples,idx)
        image(output .* color + epoch   + color   )

        % Only draw axes on the leftmost column
        if(example == 1)
            axis on
            str = 'Ep:';
            str = strcat(str,int2str(epoch));
            ylabel(str)
        else
            axis off
        end

        % Make sure the plots use a square grid
        axis square
    end
end

% Finally we plot the goal vector for comparison
for idx = 1:n_examples
    subplot(n_epochs + 2,n_examples,(n_epochs + 1) * n_examples + idx);
    image(data(:,:,idx).* color + n_epochs+1  + color)
    if(idx == 1)
       axis on
       ylabel('goal')
```

```
    else
        axis off
    end
    axis square
end
%
    -------------------------------------------------------------------------------------------
```

4 Theory Questions

a)



b) This network is in an equilibrium and also a stable state because no matter what node you fire it will go back to this state.

c) The 2 key features of weights in a Hopfield are:

1. The weights between 2 nodes are the same in both directions ($w_{ij} = w_{ji}$)
2. The weight of a node to it self is always zero ($w_{ii} = 0$)

d) To compute the activation for a single node you use the following formula:

$$a_i = \sum_{j \in J} w_{ij} x_j$$

e) The type of the activation function is a step function with the formula:

$$Y = \begin{cases} 1 & \text{if } a \geq 0 \\ 0 & \text{if } a < 0 \end{cases}$$