Neural Networks for AI

LAB 4

Fernanda Marana (s3902064)
Floris Cornel (s2724685)

May 23, 2019

1 Questions on the Hopfield implementation

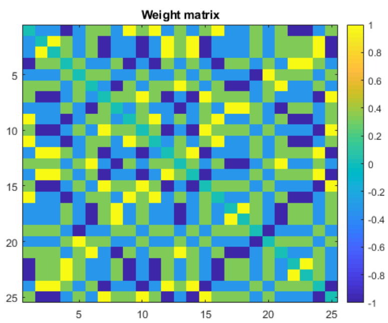

Figure 1: Weight matrix

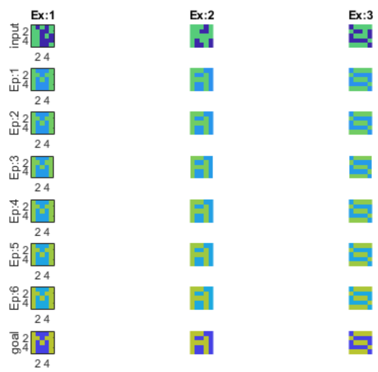

Figure 2: Correct recognition of 3 patterns.

In figure 1 you can see the normalized weight matrix. Figure 2 depicts a correct output found by the network.

1.1 Explain why repetitively applying Hebb's learning rule can be captured by the following formula

Hebb's learning rule is based on the idea that nodes that tend to have the same value over the training set are strongly correlated, and nodes with opposite values are anti-corelated. The correlated notes get more positive values and the anti-corelated nodes get more negative values. Doing this process repetitively, can be done by looking at v_i^p and v_i^p for every pattern p. The weights will be determined by the sum of $-1 * -1 = 1$ and $1 * 1 = 1$ for corelated nodes, and $-1 * 1 = -1$ and $1 * -1 = -1$ for anti-corelated nodes. So we can derive $w_{ij} = \sum_{p=1}^{P} = v_i^p * v_i^p$.

1.2 Omit the normalization of the weights and rerun the algorithm. What happens to the performance of the network? How can you explain this?

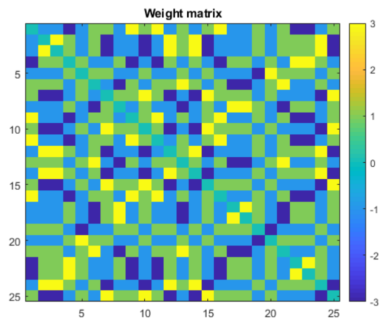

Figure 3: Weight matrix of without normalization

Figure 4: Correct recognition of 3 patterns without normalization.

What we can see from figures 3 and 4 is that when the normalization is turned off, the range of values in the weight matrix goes from 3 to -3 instead of from 1 to -1. However, it does not affect the networks ability to recognise the three patterns. This is because the weights, whether normalized or not, affect the output relatively to to other weights, so normalisation is not strictly needed.

1.3 What is the theoretical number of patterns that a Hopfield network using 25 neurons can store?

Following the equation bellow

$$\frac{N}{2 * ln(N)}$$

in which N corresponds to the number of neurons, we get the value of 3.88. So the theoretical number of patterns must be below it.

1.4 Vary $n\_examples$. How many different patterns can be stored properly by the network? Why is this different from what you have found in (1.3)?

We were unable to successfully recognise 4 different patterns with our network, confirming the $3 < 3.88 < 4$ as the maximum number of stored patterns.

2 Noise and spurious states

Figure 5: Output with deviant end states

Figure 6: Output of inverted

2.1 Explain using your own words how the network is able to recover the patterns from the noisy input. Your answer should at least involve energy and weights.

The network is able to recover the patterns from the noisy because the energy is responsible to shape the network either decreasing or staying the same upon units being updated. So under repeated updating the network will eventually converge to a state which is a local minimum in the energy function .Additionally, the weights guarantees that the energy function decreases monotonically while following the activation rules.

2.2 Occasionally, for higher noise levels (30 - 35%) some peculiar end states might occur. Explain why these states are only observed with synchronous updates and not with asynchronous updates. If you cannot find any deviant end states then you can increase the chance of observing one by using more patterns and increasing the amount of epochs.

In figure 5 we see an example of output with deviant end states due to higher noise. This mismatched recognition is because the flipping of random bits makes the input more similar to a wrong goal than its own correct goal. It does not always happen, but the larger the noise level, the larger the chance of becoming similar to the wrong goal. Very low noise levels makes it impossible because there is a minimum number of flips that need to take place in order to cross the line between one goal and another goal.

2.3 Set invert in the beginning of the code to true. Explain what happens to the input. Does anything change for the training of the network?

We can see in figure 6 that the input pattern is inverted, but the network is still capable of learning the correct shape, so the learning is not effected negatively.

2.4 Explain why the inverted patterns are also stored by the network.

Well, the network looks at the edges of certain patterns, so the difference between certain neighbours. These differences can either be large or small, but there is no indication whether a large difference should be positive or negative. As a result, the inverted pattern is just as valid and easily recognised as the regular pattern.

3 Code

```
% 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 = 60;     % 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');
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
    % Every row will represent an example
    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_examples
weights = transpose(vector_data) * vector_data;

% A hopfield neuron is not connected to itself. The diagonal of the matrix
% should be zero.
weights = weights - diag(diag(weights));

% 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 i=1:size(activation)
            if activation(i)>=0
                activation(i) = 1;
            else
                activation(i) = 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
%--------------------------------------------------------------------------
```

Listing 1: hopfield.m