Neural Networks

Assignment 3

Daniël Haitink (s2525119)
Job Talle (s2455951)

March 7, 2016

1 Theory Questions

1.1 Draw a Hopfield network with 3 neurons and indicate which neurons are active.

The green nodes are active.

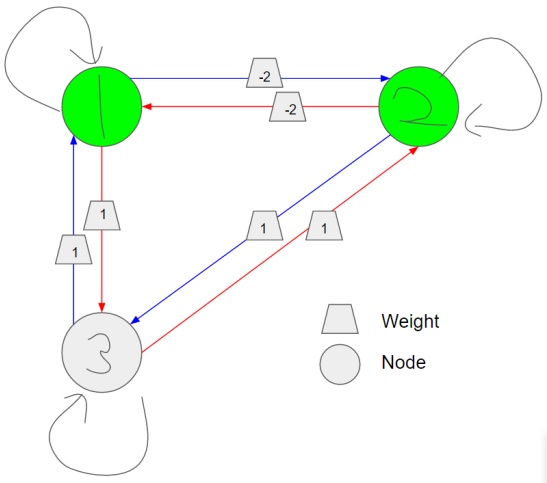

Figure 1: Hopfield network

1.2 Is the Hopfield network that you drew in an equilibrium? Explain why.

We believe that our network, showed in figure 1, is in a equilibrium, because the cumulative energy is at a local minimum.

1.3 What are two key features of the weights in a Hopfield network?

The weights are symmetric, they are equal for both directions. Also, every node is connected to every other node.

1.4 How do you compute the activation of a neuron in a Hopfield network?

It is the weighted sum of all inputs. The activation for node three in figure 1 is calculated by summing the weights of its inputs. In this case, this is 2.

1.5 Which type of activation function is used in a Hopfield network?

In the case of our implementation, a threshold of zero is used. If the activation exceeds zero, the output is one. Otherwise, it is -1.

1.6 Which kind of neurons does a Hopfield network contain when you consider your answers to (e) and (f)?

The nodes in a hopfield network are treshold logic units with a treshold of zero.

2 Implementing a Hopfield network in MATLAB

We implemented the algorithm and ran it several times using different variables for *n_ epochs* and *n_ examples*.

2.1 Examples 3

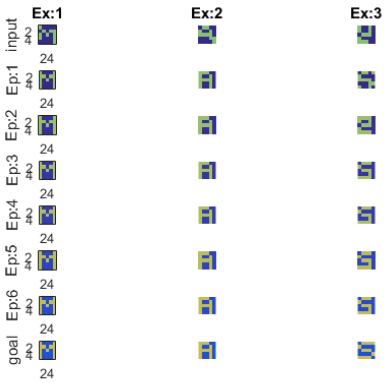

Figure 2: Hopfield output (epochs = 6, examples = 3)

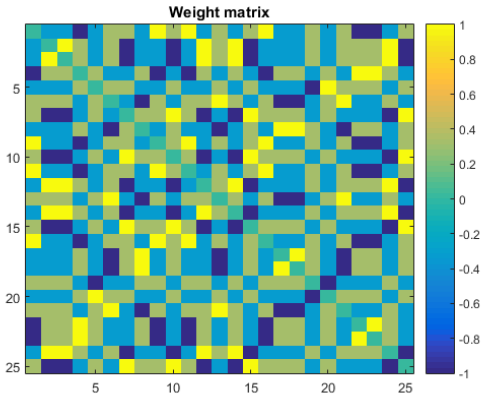

Figure 3: Hopfield weights (epochs = 6, examples = 3)

2.2 Examples 2

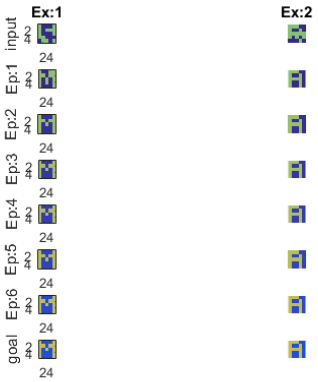

Figure 4: Hopfield output (epochs = 6, examples = 2)

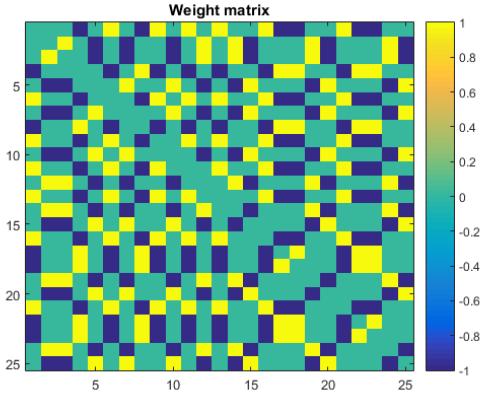

Figure 5: Hopfield weights (epochs = 6, examples = 2)

2.3 Examples 4

2.3.1 Epochs 6

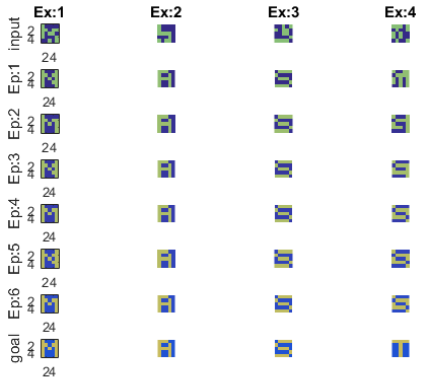

Figure 6: Hopfield output (epochs = 6, examples = 4)

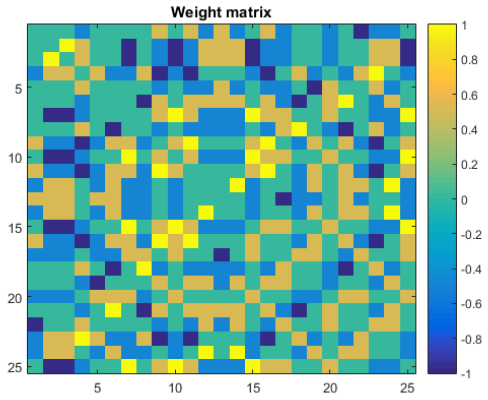

Figure 7: Hopfield weights (epochs = 6, examples = 4)

2.3.2 Epochs 15

2.4 Examples 4

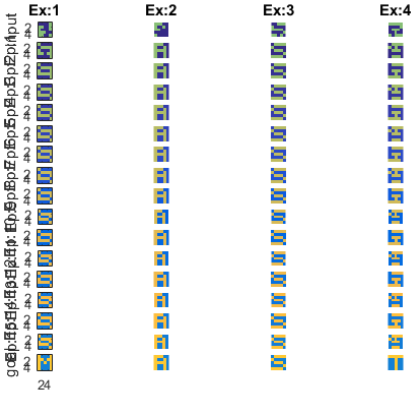

Figure 8: Hopfield output (epochs = 15, examples = 4)

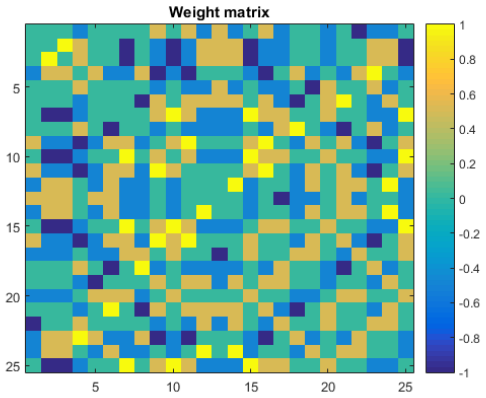

Figure 9: Hopfield weights (epochs = 15, examples = 4)

2.5 Examples 6

2.5.1 Epochs 6

Figure 10: Hopfield output (epochs = 6, examples = 6)

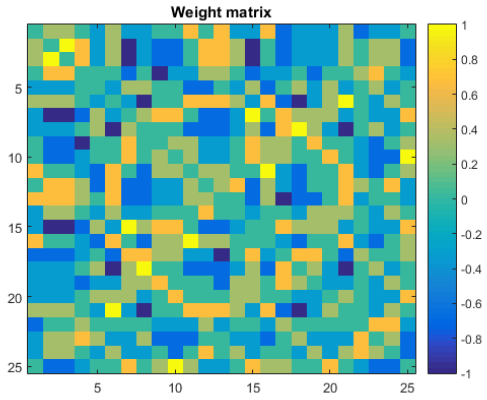

Figure 11: Hopfield weights (epochs = 6, examples = 6)

2.5.2 Epochs 20

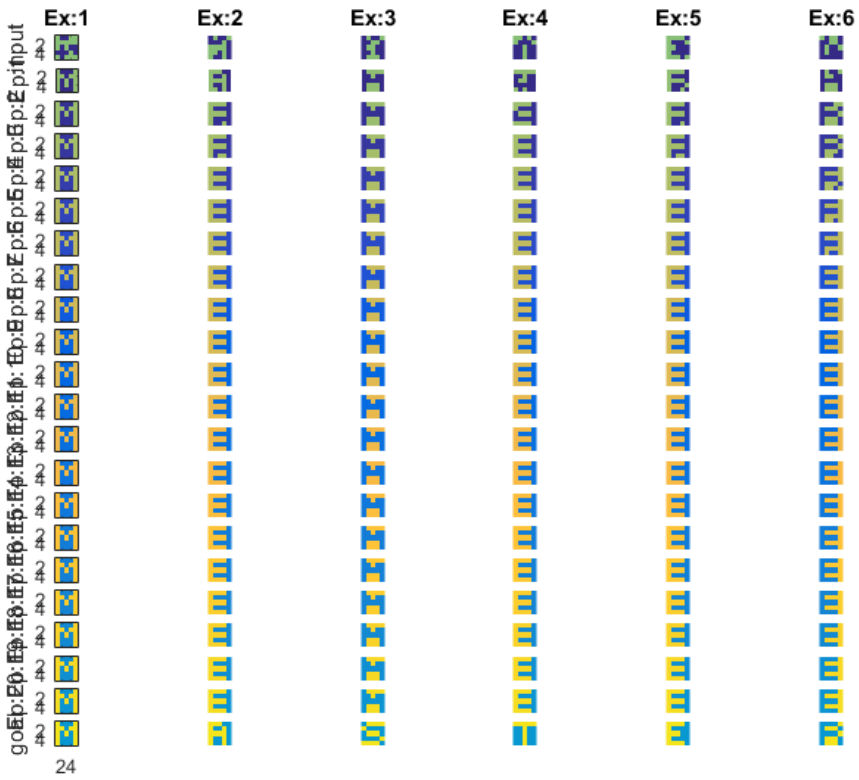

Figure 12: Hopfield output (epochs = 20, examples = 6)

3 Questions on the Hopfield implementation

3.1 Explain why repetitively applying Hebb's learning rule can be captured by the Hebbian learning formula

The formula which we need is the following:

$$w_{ij} = \sum_{p=1}^{P} v_i^p v_j^p$$

Neurons which are simultaneously activated should have stronger connections (Neurons that fire together, wire together). In the formula we multiply v_i and v_j, which reinforces the connections between each other if they fire together.

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

As shown in figure 13, it still works like it used to. We can compare this with figure 2. We see no real changes. So it seems like the performance is still the same. Even the weights figures 3 and

figure 14 look very similar. However we can see that the output of figure 13 is correct in epoch 1, whereas it takes longer when the normalisation is done.

Normalisation reduces the value of the weights, which makes the values less extreme. Therefore it will take a bit longer to get the correct result, but it is also more forgiving and can still change a bit. Without normalisation, the output can change a lot less over time.

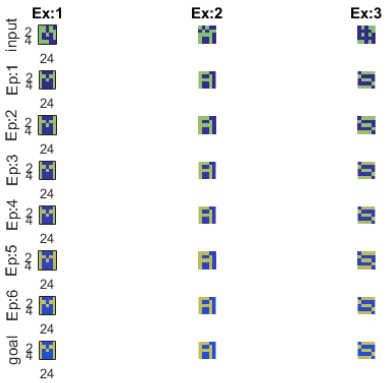

Figure 13: Hopfield output without normalisation (epochs = 6, examples = 3)

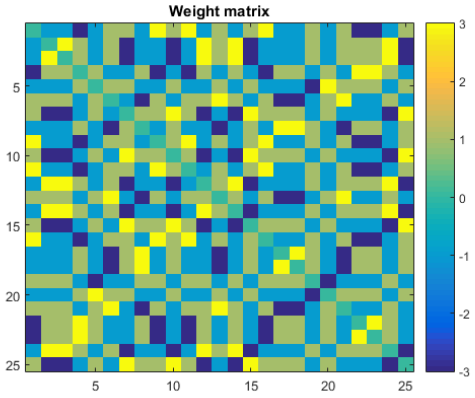

Figure 14: Hopfield weights without normalisation (epochs = 6, examples = 3)

3.3 The number of different patterns that can be stored in a Hopfield network is dependent on the number of neurons. Provide the formula with which you can determine the number of patterns that a Hopfield network can store given the number of neurons.

The function according to Hopfield is:

$$P = 0.15N$$

Where N is the number of nodes, and P is the number of patterns which can be stored.

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

If we calculate this using our formula we get:

$$P = 0.15 * 25 = 3.75$$

Therefore we can say that we can store up to 3 patterns accurately. This is also shown by our many figures above.

3.5 Vary n examples. How many different patterns can be stored properly by the network? Why is this different from what you have found in (d)?

When we run these tests with 3 examples our patterns are often correctly stored, like stated in (d). When we increase this, less than 3 patterns are stored because there is too much extra information which is stored. This creates noise, and therefore the other patterns are also incorrect (like in figure 10). This is in accordance with (d).

4 Noise and spurious states

4.1 Noise 40

Figure 15: Hopfield output with noise 40 (epochs = 6, examples = 3)

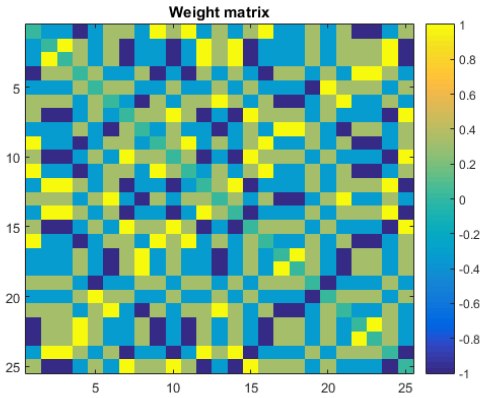

Figure 16: Hopfield weights with noise 40 (epochs = 6, examples = 3)

4.2 Noise 50

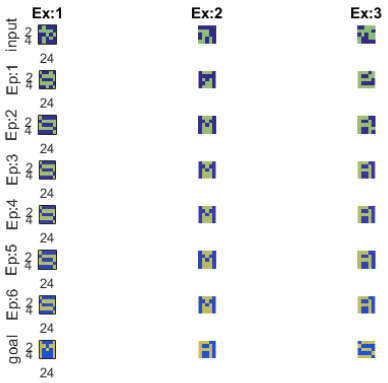

Figure 17: Hopfield output with noise 50 (epochs = 6, examples = 3)

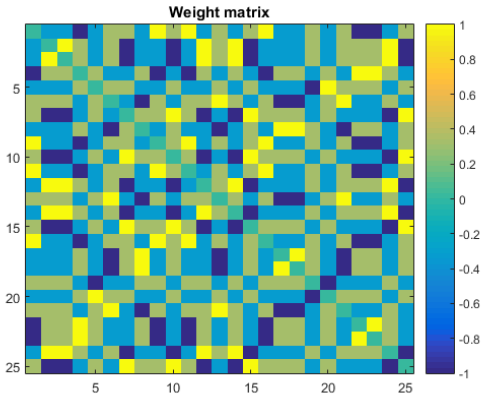

Figure 18: Hopfield weights with noise 50 (epochs = 6, examples = 3)

4.3 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.

If you have a noisy input, you have a high energy in respect to a goal state. The goal state has the lowest energy and the hopfield network strives to obtain the lowest energy state. The weights in the hopfield network will strive to the lowest energy possible and will lead the noisy input to its closest minimum.

4.4 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.

When a hopfield network is trained synchronously, multiple goal states represent multiple minima. Peculiar end states can be observed in for example figure 10. These states are visually similar to combinations of actual end states. These local minima exist when multiple actual output states are remembered by the network. If the network is trained for too many goal states, these local minima might be created from combinations of these states.

4.5 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?

The input literally inverts, thus if a pixel was white, it is now black and vice versa. Therefore the output also is the inverted goal image. Because it is just an inversion and the calculations don't get affected by inversion it still works.

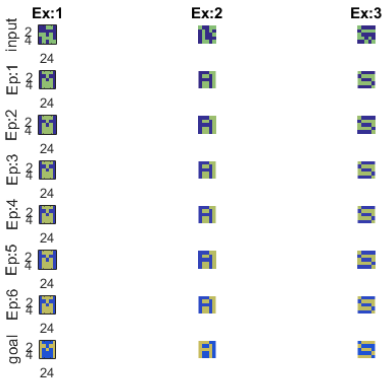

Figure 19: Hopfield output inverted (epochs = 6, examples = 3)

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

This has to do with the usage of -1 and 1. Since there is no 0, we never obtain 0 from a multiplication. Therefore, if we multiply -1 by -1, we obtain 1. And if we multiply -1 by 1 we get -1.

5 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 = 25;     % Percentage of bits that are flipped randomly
invert = false;             % Invert the input (test for spurious states)
data_size = 25;


% 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 = zeros(data_size,data_size);
```

```
52
53 for p = 1:n_examples
54    for i = 1:data_size
55       for j = 1:data_size
56           if i ~= j
57             weights(i,j) = weights(i,j) + vector_data(p,i)*vector_data(p,j);
58           end
59       end
60    end
61 end
62
63
64 % A hopfield neuron is not connected to itself. The diagonal of the matrix
65 % should be zero.
66 %weights = [];
67
68 % These lines check whether the matrix is a valid weight matrix for a
69 % Hopfield network.
70 assert(isequal(size(weights),[size_examples size_examples]), ...
71     'The matrix dimensions are invalid');
72 assert(isequal(tril(weights)',triu(weights)), ...
73     'The matrix is not symmetric');
74 assert(isempty(find(diag(weights), 1)), ...
75     'Some neurons are connected to themselves');
76
77 % Normalizing the weights
78 if normalize_weights
79     weights = weights ./ n_examples;
80 end
81
82
83 % PLOT WEIGHT MATRIX
84 figure(1)
85 imagesc(weights)
86 colorbar
87 title('Weight matrix')
88
89 % INTRODUCE NOISE
90
91 % Copy the input data
92 input = vector_data;
93
94 % Create a matrix with the same dimensions as the input data in which
95 % random_percentage elements are set to -1 and the others are set to 1.
96 % We do this by sampling from a normal distribution
97 noise_matrix = (randn(size(input)) > norminv(random_percentage));
98 noise_matrix = noise_matrix - (noise_matrix==0);
99
100 % Flip bits (* -1) using the noise_matrix
101 input = input .* noise_matrix;
102
103 % Optionally invert the input
104 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
        initial_activation = activation;
        for i = 1:data_size
            activation(i) = sum(weights(:,i).*initial_activation);
            if activation(i) < 0
                activation(i) = -1;
            else
                activation(i) = 1;
            end
        end
        % Apply the activation function

        % 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  )
```

```
158
159         % Only draw axes on the leftmost column
160         if(example == 1)
161             axis on
162             str = 'Ep:';
163             str = strcat(str,int2str(epoch));
164             ylabel(str)
165         else
166             axis off
167         end
168
169         % Make sure the plots use a square grid
170         axis square
171     end
172 end
173
174 % Finally we plot the goal vector for comparison
175 for idx = 1:n_examples
176     subplot(n_epochs + 2,n_examples,(n_epochs + 1) * n_examples + idx);
177     image(data(:,:,idx).* color + n_epochs+1  + color)
178     if(idx == 1)
179        axis on
180        ylabel('goal')
181     else
182         axis off
183     end
184     axis square
185 end
186 %--------------------------------------------------------------------------
```

Listing 1: hopfield.m