Neural Networks

Assignment 2

Daniël Haitink (s2525119)
Job Talle (s2455951)

March 2, 2016

1 Theory Questions

1.1

When a multilayer neural network does something well (or poorly), it is hard to determine which parts did well or bad. It's a challenge to assign credit to the individual neurons in a MLP.

1.2

The two factors that determine the error of a neuron in a hidden layer after the forward pass has finished are the node's influence on the output nodes and how the output of the output node affects the error. We can call those the input and the sigmoid slope.

1.3

The mathematical definition of the sigmoid function is stated below:

$$S(t) = \frac{1}{1 + e^{-t}}$$

Where **t** is a variable, and **e** is a constant (just the mathematical e).

The derivative is as follows:

$$S(t) = \frac{e^t}{(e^t + 1)^2}$$

1.4

This is impractical because the neural network has to alter the weights during training, and if those weights are initialised at a high number it is hard to correct these weights if they are not correct.

1.5

1.5.1

The mean network error can be used as a criteria, the disadvantage is that it assumes the net can decrease its mean error below a certain value.

1.5.2

The network can stop learning if the change of the error becomes very small. The disadvantage is that the network can get stuck in a local minimum.

1.5.3

If the output nodes of the network are closest to the boolean they should give, for each situation, then the network can stop training. The disadvantage of this method is that the network might be unable to reach this quality.

1.6

By implementing momentum of weight change, which means that the weights do not change in one cycle, but get a change speed which persists for a few cycles. The persistence of the momentum is determined by a number between zero and one. One means that the momentum will not change, ever. Zero means that it will lose its momentum instantly.

1.7

If the network can classify images outside of the trainingset it has generalised.

1.8

If the network can classify the trainingset well, but cases outside of the trainingset poorly, it might be overfitted. The weights are fitted too well to the trainingset, which leaves no room for deviations.

1.9

Network pruning means changing the internal structure and possibly connections between neurons/input/output to improve the performance of the neural network.

Two ways of network pruning are:

1.9.1

Incremental pruning, which means that neurons are added to the network and the network is retrained at each increment. At a certain point the network's performance will be assumed to be optimal.

1.9.2

Selective pruning, means that after a network is trained, insignificant weights will be pruned. Therefore the network can focus on training significant weights.

2 MLP on (digital) paper

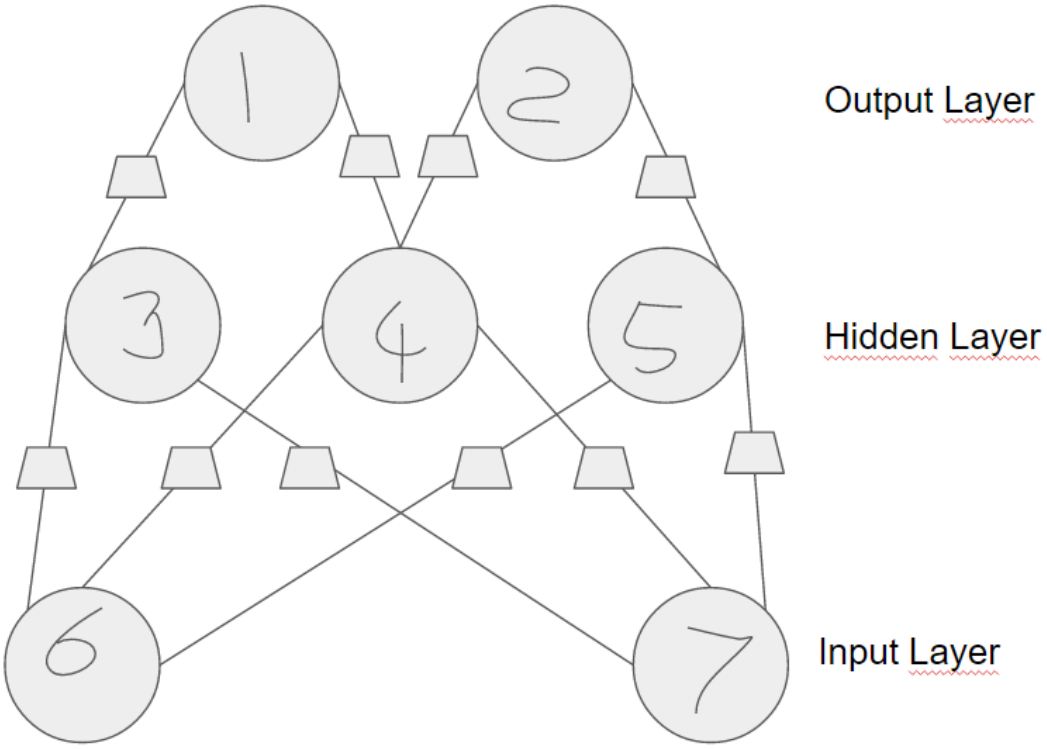

Figure 1: MLP on paper

2.1 Why is this considered a two-layer network?

There are only two layers of weights in this network, an input signal passes through merely two connections. The input nodes don't process information and therefore they do not count.

2.2 What are the dimensions of the weight matrix?

The weights matrix is a 2x3 matrix. There are 2 input (and output) nodes, and three nodes in the hidden layer.

2.3 How many weights does the network count?

There are 10 weights in total, distributed over two layers. The weights in a single column are all connected to the same hidden node.

3 Implementing in MATLAB

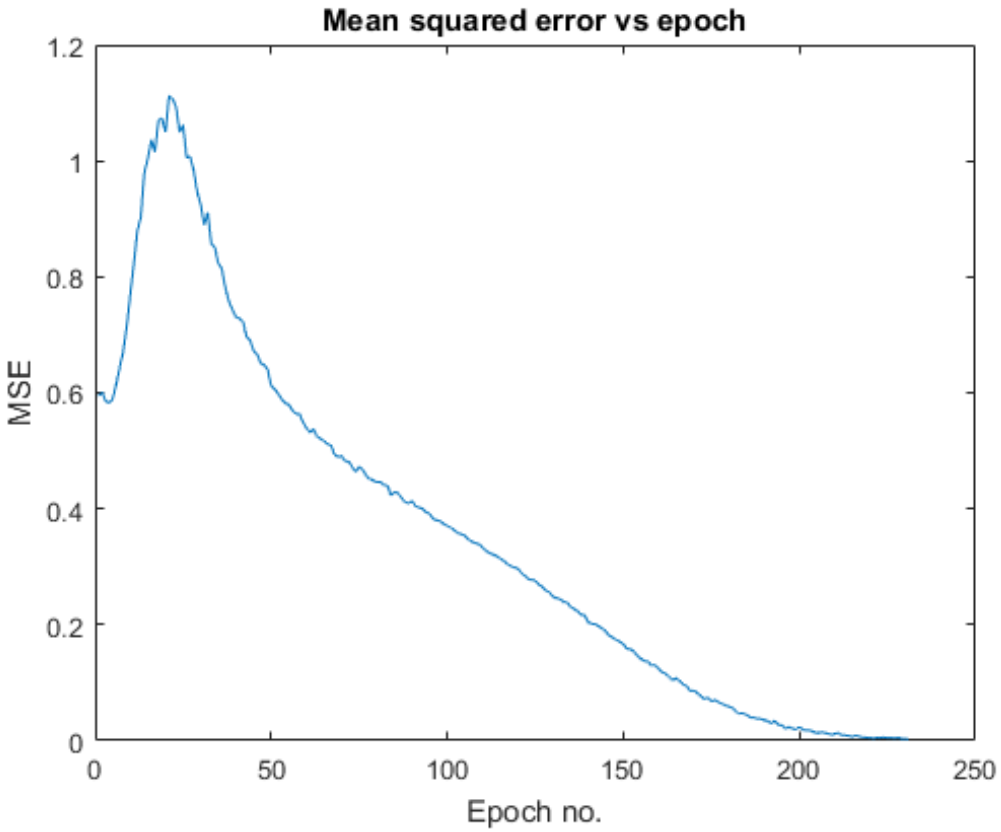

Figure 2: Error of the MLP

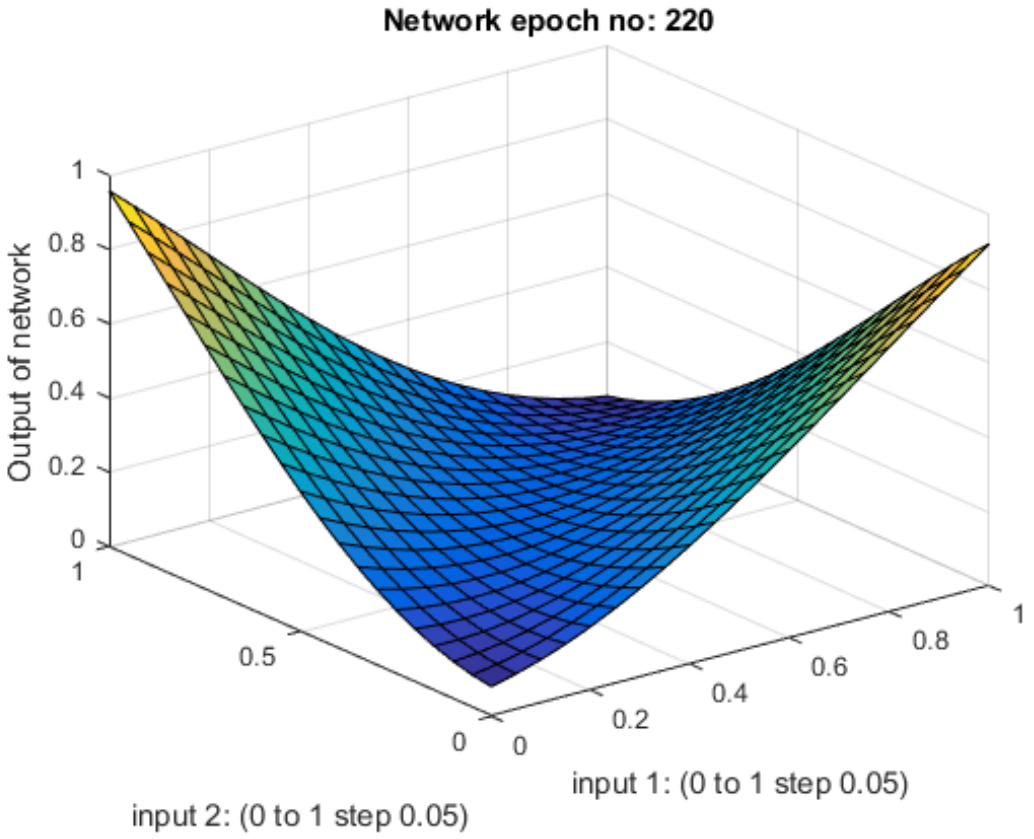

Figure 3: MLP output

4 Testing MLP

4.1 Is it guaranteed that the network finds a solution. Why so?

In the XOR case, this is guaranteed because there are no local minimums which can prevent the network from finding the solution. In other cases it is not per se.

4.2 About how many epochs are needed to find a solution?

About 500-600 epochs are needed to find the solution, this can also be seen in the figure 2.

4.3 Set the noise_level to 0.5. Explain what happens.

The noise makes the error very distorted after about 500 epochs, and it varies wildly between 0 and up to 0.6. The output is also worse than the first since it is less well trained on the data.

5 Implementing in MATLAB

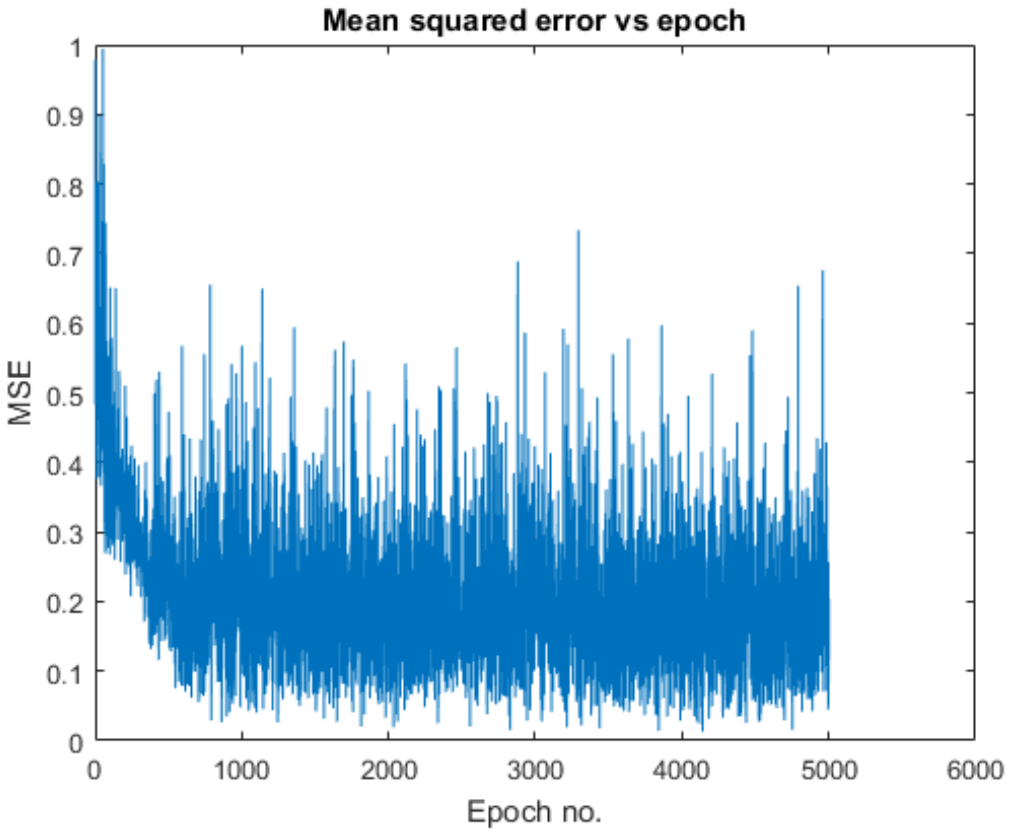

Figure 4: Error of the MLP with noise of 0.5

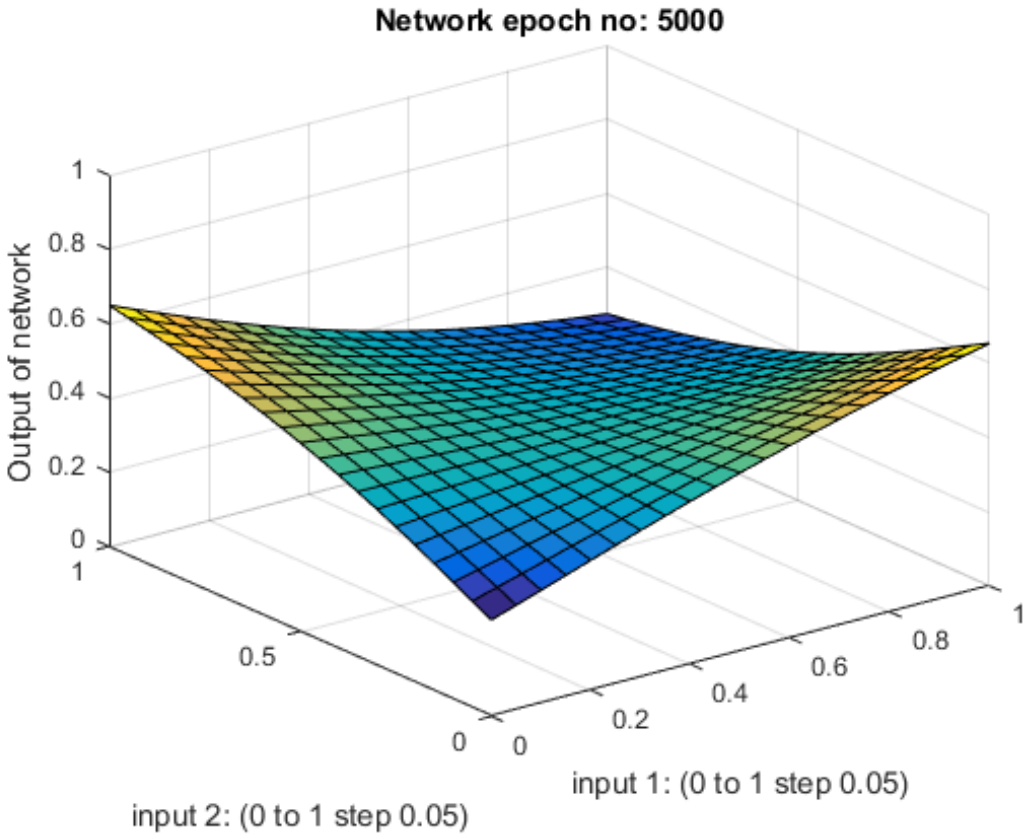

Figure 5: MLP output with noise of 0.5

5.1 Set the noise_level to 0.2. Change the weight_spread to 5. What can you observe? Explain your results using the delta-rule.

The weights are initialised too high and have to be set to normal values. Because it has to descent back to an error of close to 0, and there is quite a bit of noise, the neural network creates a suboptimal form. This has to do with the delta rule, which causes big delta's of the weights.

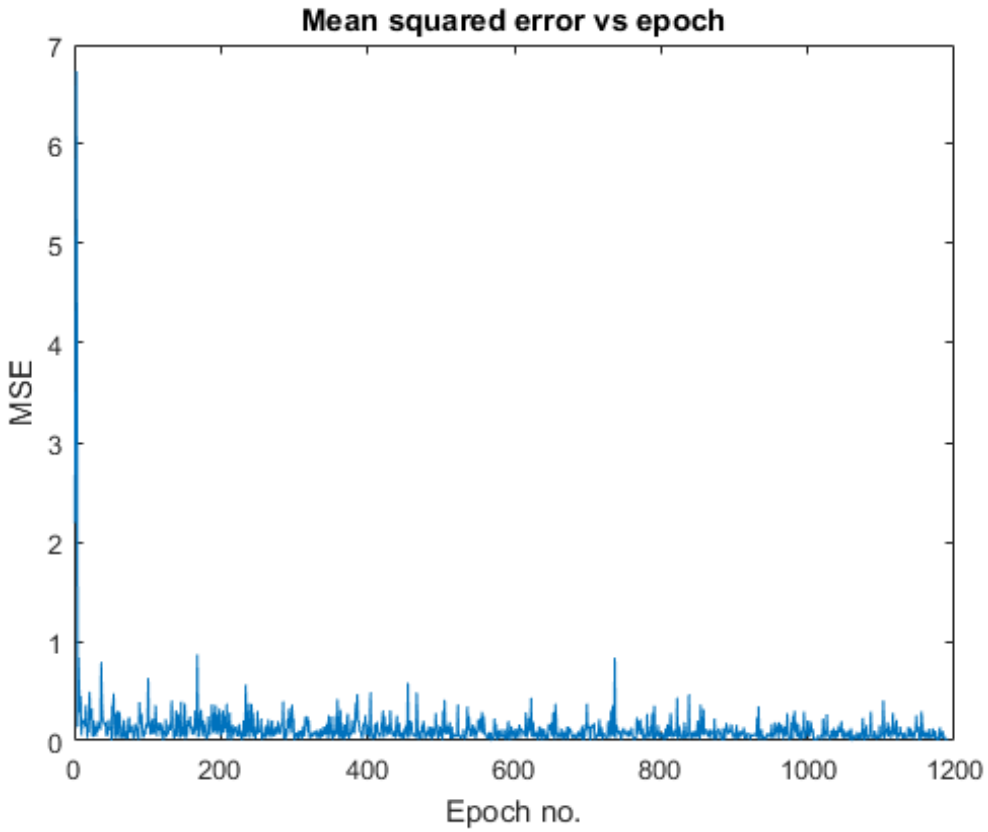

Figure 6: Error of the MLP with noise of 0.2

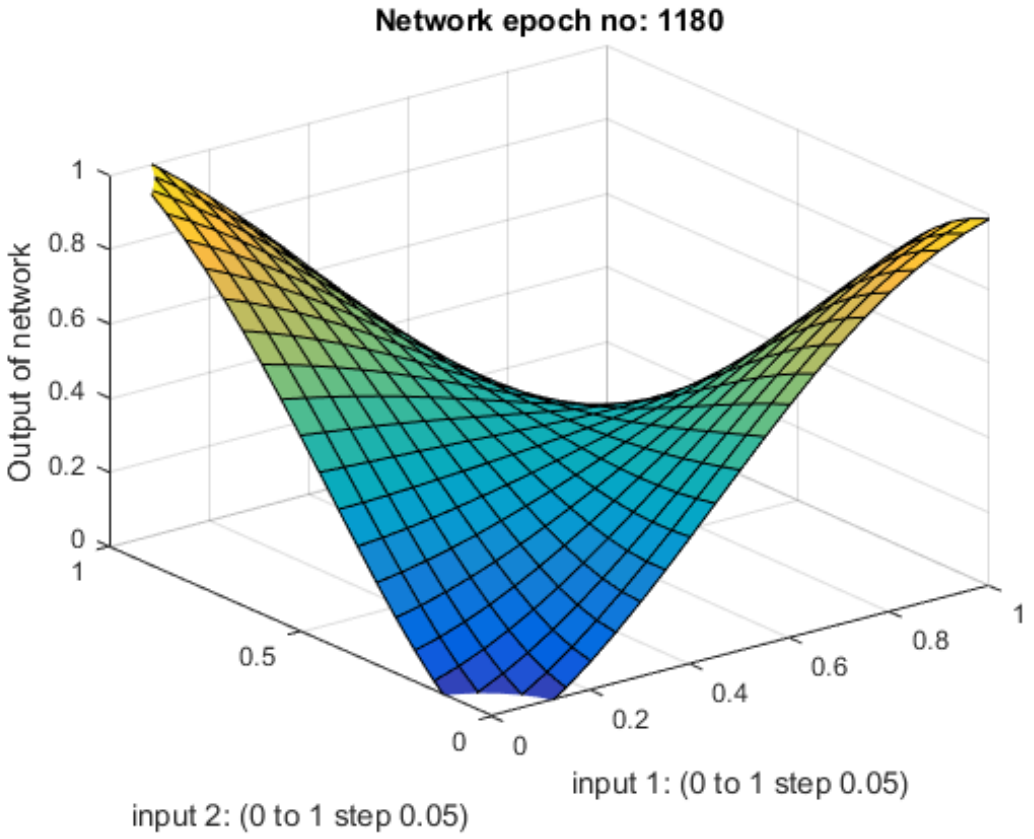

Figure 7: MLP output with noise of 0.2

5.2 Set the noise_level to 1%. Leave the weight_spread at 5. There are two qualitatively different solutions to the XOR problem. What are these two? Include a figure of both solutions.

The first image is shown in figure 3 and the new one is shown in figure 8.

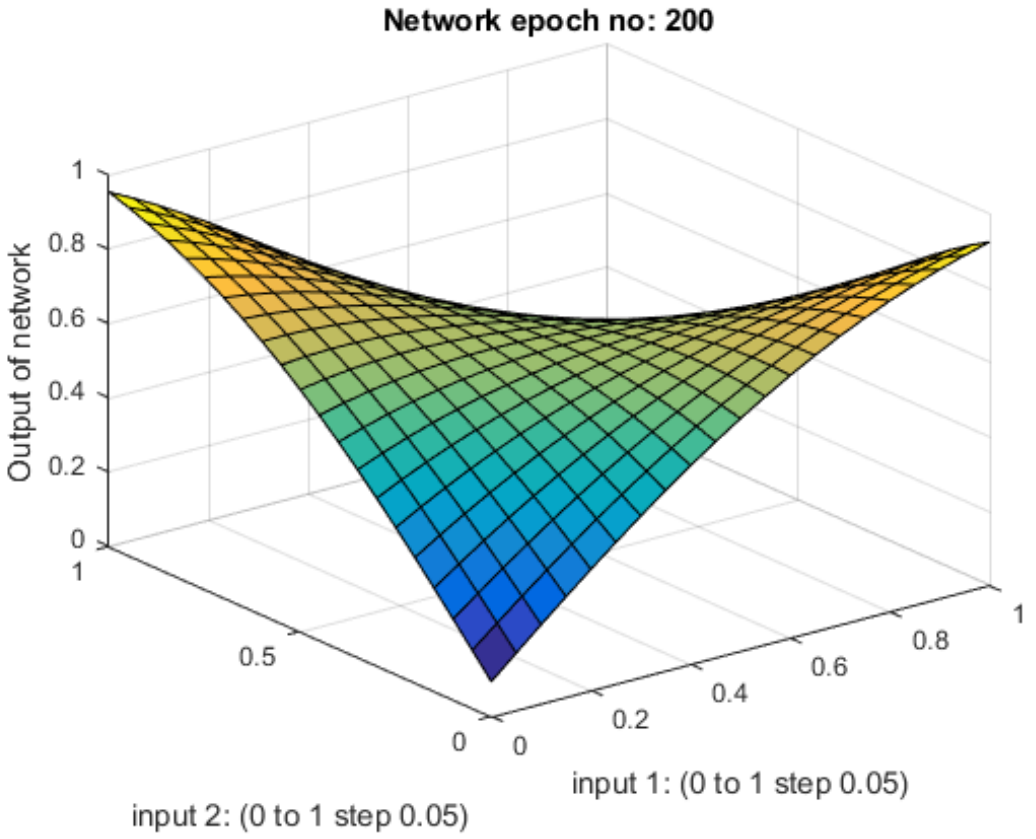

Figure 8: MLP output, second solution

5.3 Which shape does the graph of the error usually have? Explain the shape with the use of the delta-rule

Depending on the noise, it usually gives an output similar to $1/x$ which is decreasing quadratic. This is caused by a random initialised weights and a low learning rate. This causes the error to go down relatively steeply at the beginning, because the learning rate has a high effect on the weights. It becomes increasingly less steep. Because the weights become more correct, the effect of the learning rate becomes smaller and smaller.

6 Another Function

6.1 Is the network capable of learning the sine function?

Yes, it can approximate the sine function as shown in figure 10. It does not really learn it, but it is very similar.

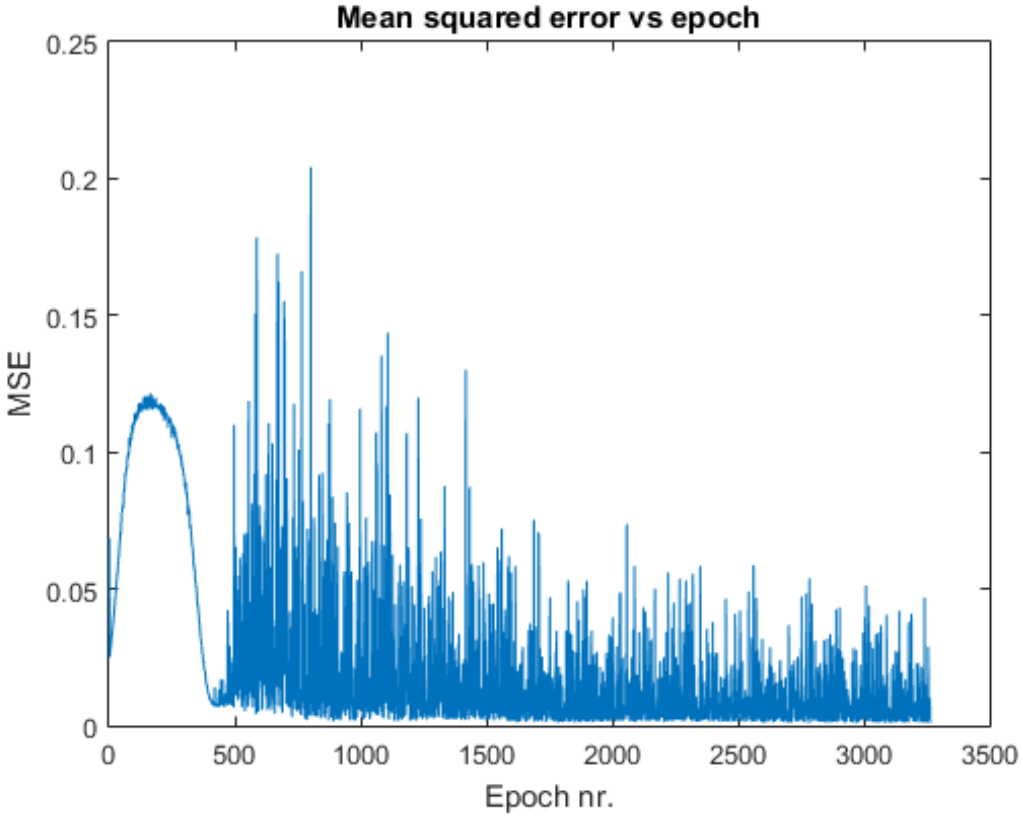

Figure 9: Error of the sine MLP

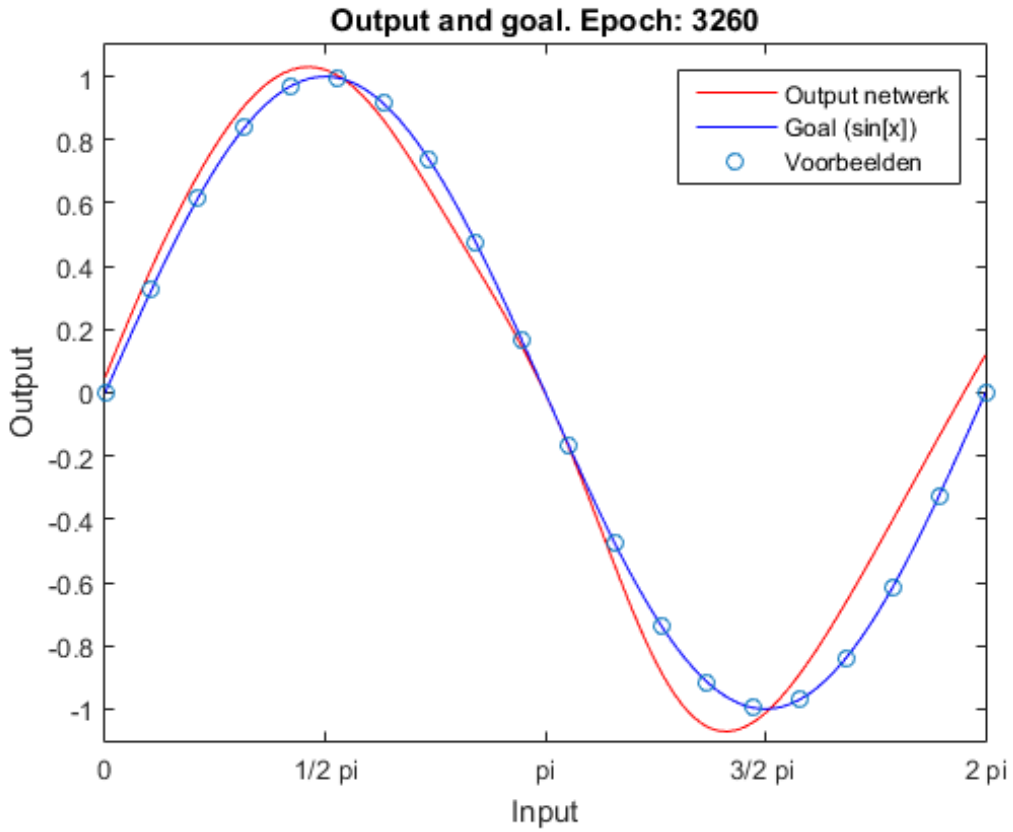

Figure 10: MLP output of the sine

6.2 Set n_examples in the top of the file to 5. Rerun the simulation. What can you observe? With which feature of neural networks does this phenomenon correspond?

Since there are only 5 examples given it becomes harder to learn the sine function, therefore it needs more epochs to learn it. Also it may cause overfitting, since there is no real requirement to form a sine function, because the small number of examples doesn't accurately describe a sine function.

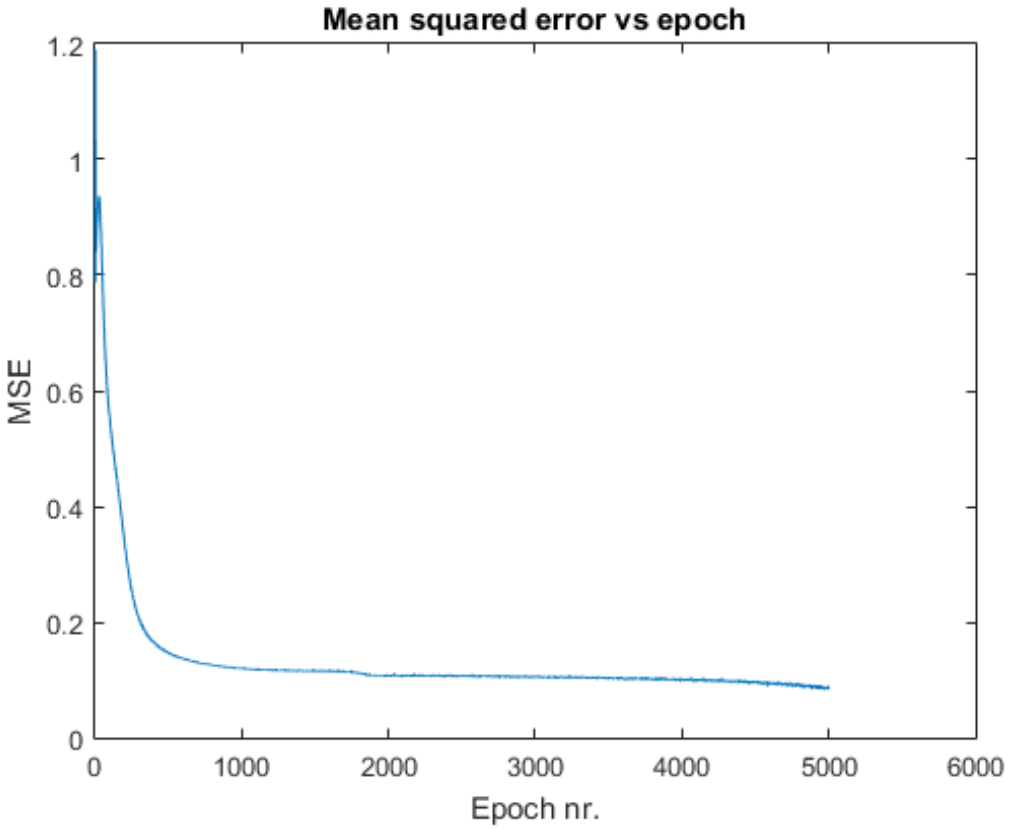

Figure 11: Error of the sine MLP

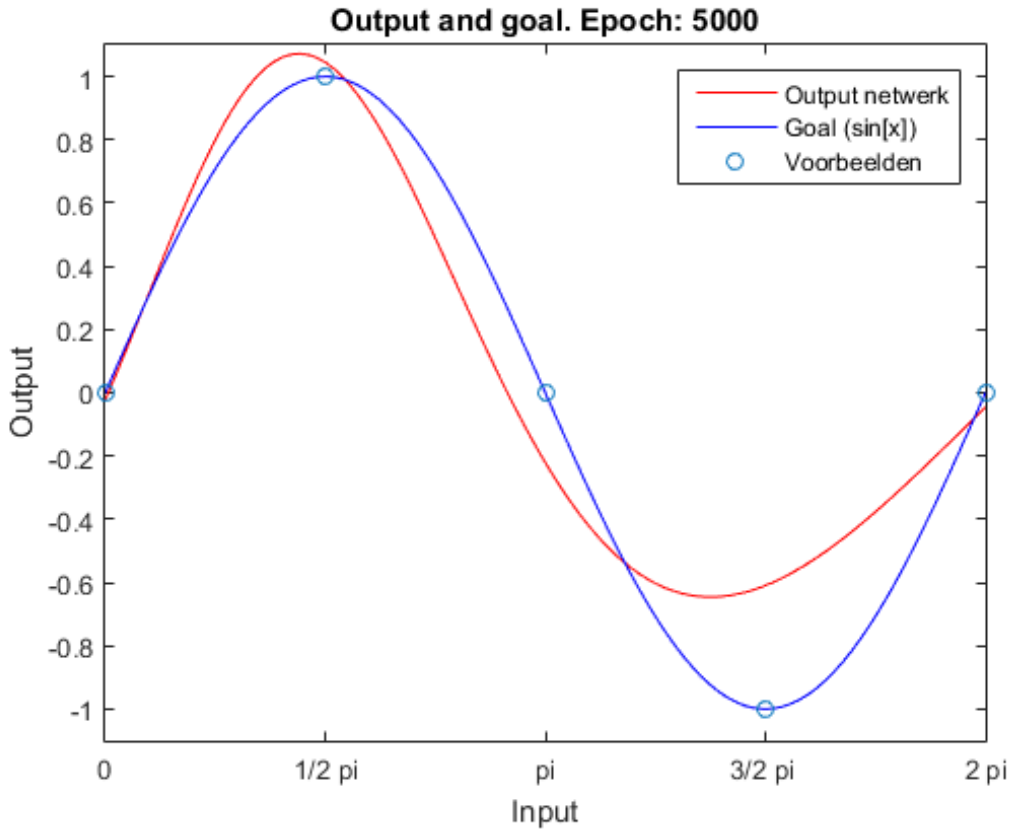

Figure 12: MLP output of the sine

6.3 Set plot_bigger_picture to true. How is the domain of the network determined? What happens if the input is outside of this domain?

The domain is set to 0 to 2π. If the input is outside the domain it does not take the form of the sine function. There are no samples at those position, and therefore it will not learn the sine. This is also shown in the figures below.

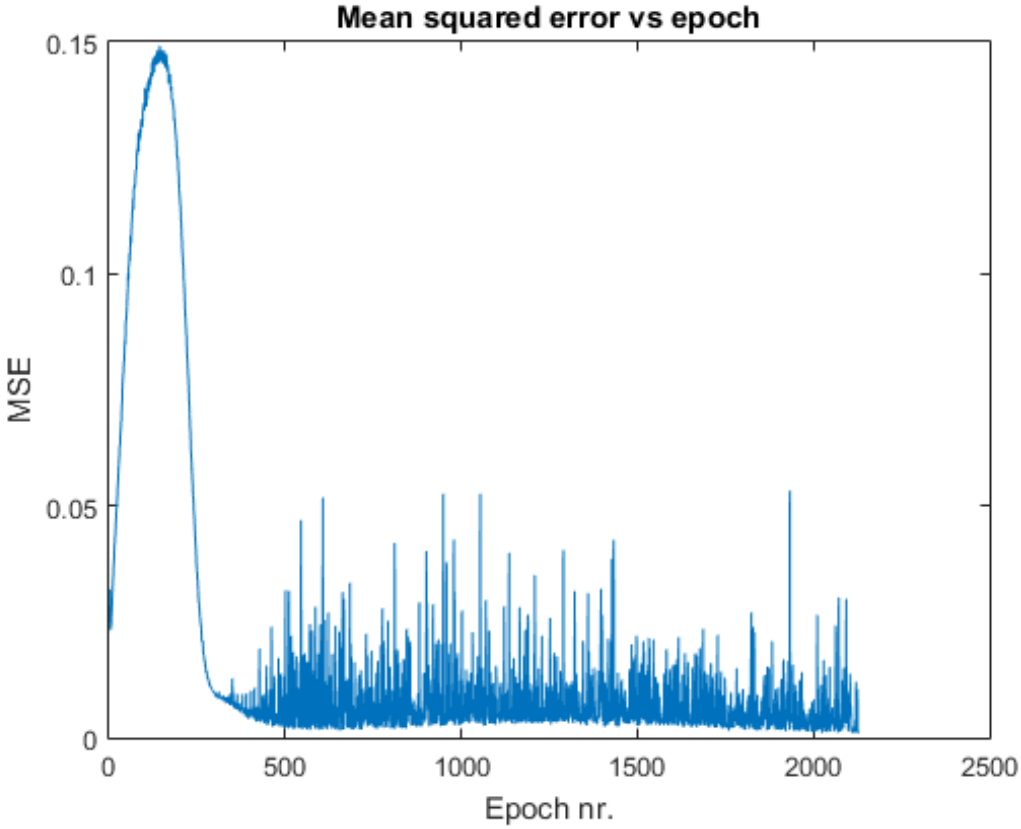

Figure 13: Error of the sine MLP

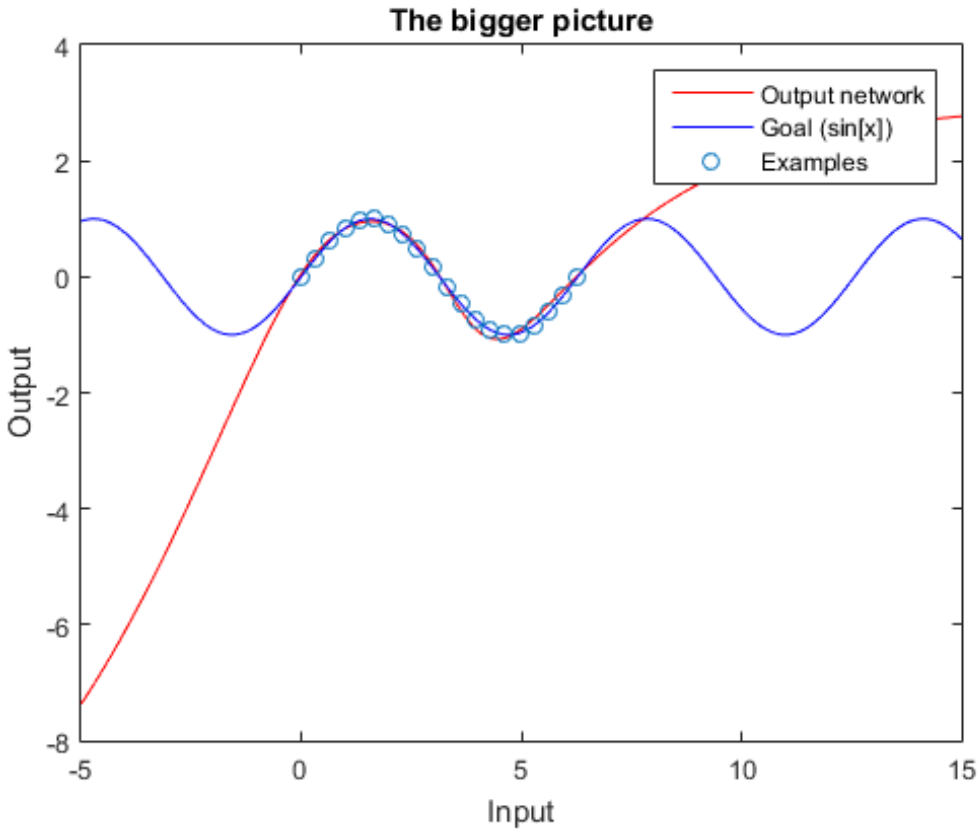

Figure 14: MLP output of the sine

6.4 At least how many neurons are required to learn a sine?

The number of neurons which is needed to learn a sine is at least three. We tested this and this was shown to be true. Because there are three linear functions to describe the general shape of a sine.

6.5 You have modified the output function for this part of the lab assignment. Does the XOR learning network still work? Why so?

Yes.

7 Code

```
%this functions calculates the sigmoid
function [output] = sigmoid(x)
    % Define the sigmoid function here
    output = 1./(1+exp(-x));
end
```

Listing 1: sigmoid.m

```
%this functions calculates the differential of the sigmoid
function [output] = d_sigmoid(x)
    temp = sigmoid(x);
```

```
    output = temp .* (1 - temp);
end
```

Listing 2: d_sigmoid.m

```
function [output] = output_function(x)
    % set output here
    output = x;
end
```

Listing 3: output_function.m

```
function [ output ] = d_output_function( x )
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
    output = 1;

end
```

Listing 4: d_output_function.m

```
% mlp.m Implementation of the Multi-Layer Perceptron

clear all
close all

examples = [0 0;1 0;0 1;1 1];
goal = [0.01 0.99 0.99 0.01]';

% Boolean for plotting the animation
plot_animation = true;

% Parameters for the network
learn_rate = 0.2;               % learning rate
max_epoch = 5000;              % maximum number of epochs
min_error = 0.01;

mean_weight = 0;
weight_spread = 2;

n_input = size(examples,2);
n_hidden = 20;
n_output = size(goal,2);

% Noise level at the input
noise_level = 0.01;

% Activation of the bias node
bias_value = -1;


% Initializing the weights
w_hidden = rand(n_input + 1, n_hidden) .* weight_spread - weight_spread/2 + mean_weight;
w_output = rand(n_hidden, n_output) .* weight_spread - weight_spread/2 + mean_weight;

```

```
% Start training
stop_criterium = 0;
epoch = 0;

while ~stop_criterium
    epoch = epoch + 1;

    % Add noise to the input data.
    noise = randn(size(examples)) .* noise_level;
    input_data = examples + noise;

    % Append bias to input data
    input_data(:,n_input+1) = ones(size(examples,1),1) .* bias_value;

    epoch_error = 0;
    epoch_delta_hidden = 0;
    epoch_delta_output = 0;

    % FROM HEREON YOU NEED TO MODIFY THE CODE!
    for pattern = 1:size(input_data,1)

        % Compute the activation in the hidden layer
        hidden_activation = input_data(pattern, :) * w_hidden;

        % Compute the output of the hidden layer (don't modify this)
        hidden_output = sigmoid(hidden_activation);

        % Compute the activation of the output neurons
        output_activation = hidden_output * w_output;

        % Compute the output
        output = output_function(output_activation);

        % Compute the error on the output
        output_error = goal(pattern) - output;

        % Compute local gradient of output layer
        local_gradient_output = d_output_function(output_activation) .* output_error;

        % Compute the error on the hidden layer (backpropagate)
        hidden_error = output_error * w_output;

        % Compute local gradient of hidden layer
        local_gradient_hidden = d_sigmoid(hidden_activation) .* (local_gradient_output *
     w_output)';

        % Compute the delta rule for the output
        delta_output = learn_rate * local_gradient_output' * hidden_output ;

        % Compute the delta rule for the hidden units;
        delta_hidden = learn_rate * local_gradient_hidden' * input_data(pattern, :);

        % Update the weight matrices
```

```
        w_hidden = w_hidden + delta_hidden';
        w_output = w_output + delta_output';

        % Store data
        epoch_error = epoch_error + (output_error).^2;
        epoch_delta_output = epoch_delta_output + sum(sum(abs(delta_output)));
        epoch_delta_hidden = epoch_delta_hidden + sum(sum(abs(delta_hidden)));
    end

    % Log data
    h_error(epoch) = epoch_error / size(input_data,1);
    log_delta_output(epoch) = epoch_delta_output;
    log_delta_hidden(epoch) = epoch_delta_hidden;

    % Check whether maximum number of epochs is reached
    if epoch > max_epoch
        stop_criterium = 1;
    end

    % Implement a stop criterion here
    if min_error >= epoch_error
        stop_criterium = 1;
    end


    % Plot the animation
    if and((mod(epoch,20)==0),(plot_animation))
        emp_output = zeros(21,21);
        figure(1)
        for x1 = 1:21
            for x2 =  1:21
                hidden_act = sigmoid([(x1/20 - 0.05) (x2/20 -0.05) bias_value] *
    w_hidden);
                emp_output(x1,x2) = output_function(hidden_act * w_output);
            end
        end
        surf(0:0.05:1,0:0.05:1,emp_output)
        title(['Network epoch no: ' num2str(epoch)]);
        xlabel('input 1: (0 to 1 step 0.05)')
        ylabel('input 2: (0 to 1 step 0.05)')
        zlabel('Output of network')
        zlim([0 1])
    end

end

% Plotting the error
figure(2)
plot(1:epoch,h_error)
title('Mean squared error vs epoch');
xlabel('Epoch no.');
ylabel('MSE');

```

```
% Add additional plot functions here (optional)
```

Listing 5: mlp.m

```
clear all
close all

% The number of examples taken from the function
n_examples = 20;

examples = (0:2*pi/(n_examples-1):2*pi)';
goal = sin(examples);

% Boolean for plotting animation
plot_animation = true;
plot_bigger_picture = true;

% Parameters for the network
learn_rate = 0.1;                % learning rate
max_epoch = 5000;               % maximum number of epochs
min_error = 0.02;

mean_weight = 0;
weight_spread = 1;

n_input = size(examples,2);
n_hidden = 50;
n_output = size(goal,2);

% Noise level at input
noise_level = 0.01;

bias_value = -1;

% Initializing the weights
w_hidden = rand(n_input + 1, n_hidden) .* weight_spread - weight_spread/2 + mean_weight;
w_output = rand(n_hidden, n_output) .* weight_spread - weight_spread/2 + mean_weight;

% Start training
stop_criterium = 0;
epoch = 0;

while ~stop_criterium
    epoch = epoch + 1;

    % Add noise to the input
    noise = randn(size(examples)) .* noise_level;
    input_data = examples + noise;

    % Append bias
    input_data(:,n_input+1) = ones(size(examples,1),1) .* bias_value;

    epoch_error = 0;
    epoch_delta_hidden = 0;
```

```
    epoch_delta_output = 0;
    for pattern = 1:size(input_data,1)
       % Compute the activation in the hidden layer
        hidden_activation = input_data(pattern, :) * w_hidden;

        % Compute the output of the hidden layer (don't modify this)
        hidden_output = sigmoid(hidden_activation);

        % Compute the activation of the output neurons
        output_activation = hidden_output * w_output;

        % Compute the output
        output = output_function(output_activation);

        % Compute the error on the output
        output_error = goal(pattern) - output;

        % Compute local gradient of output layer
        local_gradient_output = d_output_function(output_activation) .* output_error;

        % Compute the error on the hidden layer (backpropagate)
        hidden_error = output_error * w_output;

        % Compute local gradient of hidden layer
        local_gradient_hidden = d_sigmoid(hidden_activation) .* (local_gradient_output *
     w_output)';

        % Compute the delta rule for the output
        delta_output = learn_rate * local_gradient_output' * hidden_output ;

        % Compute the delta rule for the hidden units;
        delta_hidden = learn_rate * local_gradient_hidden' * input_data(pattern, :);

        % Update the weight matrices
        w_hidden = w_hidden + delta_hidden';
        w_output = w_output + delta_output';

        % Store data
        epoch_error = epoch_error + (output_error).^2;
        epoch_delta_output = epoch_delta_output + sum(sum(abs(delta_output)));
        epoch_delta_hidden = epoch_delta_hidden + sum(sum(abs(delta_hidden)));
    end

    h_error(epoch) = epoch_error / size(input_data,1);
    log_delta_output(epoch) = epoch_delta_output;
    log_delta_hidden(epoch) = epoch_delta_hidden;

    if epoch > max_epoch
        stop_criterium = 1;
    end

    % Add your stop criterion here
    if min_error >= epoch_error
```

```
        stop_criterium = 1;
    end

    % Plot the animation
    if and((mod(epoch,20)==0),(plot_animation))
        %out = zeros(21,1);
        nPoints = 100;
        input = linspace(0, 2 * pi, nPoints);
        for x=1:nPoints
            h_out = sigmoid([input(x) bias_value] * w_hidden);
            out(x) = output_function(h_out * w_output);
        end
        figure(1)
        plot(input,out,'r-','DisplayName','Output netwerk')
        hold on
        plot(input,sin(input),'b-','DisplayName','Goal (sin[x])')
        hold on
        scatter(examples, goal, 'DisplayName', 'Voorbeelden')
        hold on
        title(['Output and goal. Epoch: ' num2str(epoch)]);
        xlim([0 2*pi])
        ylim([-1.1 1.1])
        set(gca,'XTick',0:pi/2:2*pi)
        set(gca,'XTickLabel',{'0','1/2 pi','pi','3/2 pi ','2 pi'})
        xlabel('Input')
        ylabel('Output')
        legend('location','NorthEast')
        hold off
    end

end


% Plot error
figure(2)
plot(1:epoch,h_error)
title('Mean squared error vs epoch');
xlabel('Epoch nr.');
ylabel('MSE');

%Plot the bigger picture
if plot_bigger_picture
    figure(3)
    in_raw = (-5:0.1:15)';
    in_raw = horzcat(in_raw,(bias_value*ones(size(in_raw))));
    h_big = sigmoid(in_raw * w_hidden);
    o_big = output_function(h_big * w_output);

    plot(-5:0.1:15,o_big,'r-','DisplayName','Output network')
    hold on
    plot(-5:0.1:15,sin(-5:0.1:15),'b-','DisplayName','Goal (sin[x])')
    hold on
    scatter(examples, sin(examples), 'DisplayName', 'Examples');
```

```
156     hold off
157     xlabel('Input')
158     ylabel('Output')
159     legend('location','NorthEast')
160     title('The bigger picture')
161 end
```

Listing 6: mlp_sinus.m