Neural Networks for AI

Lab 3

Fernanda Marana (s3902064)
Floris Cornel (s2724685)

May 16, 2019

1 MLP

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

No, in most cases the network can find the solution, but sometimes - depending on the starting point - it can hit a plateau and never find a solution in which the error is nearly zero.

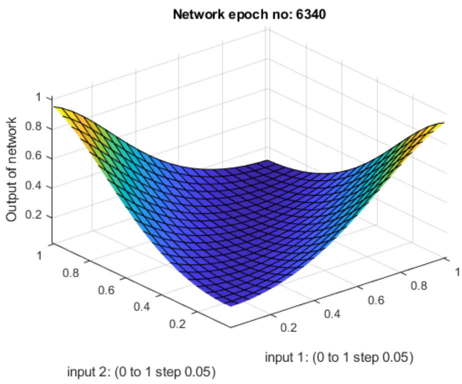

Figure 1: plot

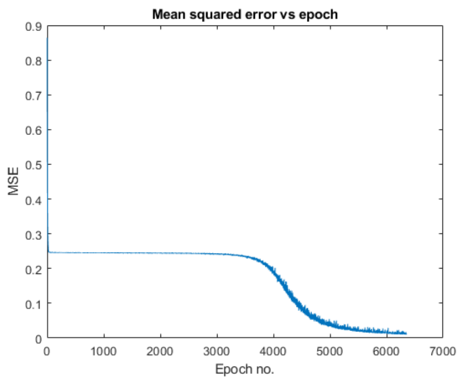

Figure 2: error

1.2 How many epochs are needed to find a solution?

There is no exact value of how many epochs. The network depending on the starting point can take more or less than 5000 epochs to find a solution in which the error is smaller than 0.01.

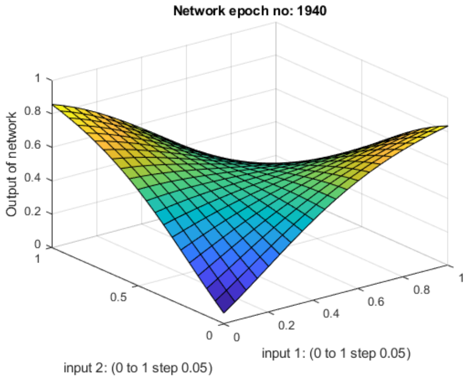

Figure 3: plot

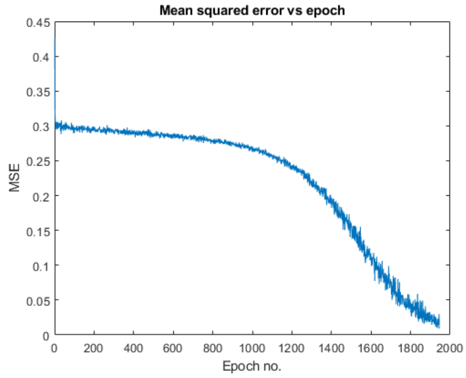

Figure 4: error

1.3 Set the noise level to 0.5? Explain what happens.

The neural networks - when the noise level is to 0.5 - does not find a solution. This happens because too much noise makes the mapping function too challenging to learn.

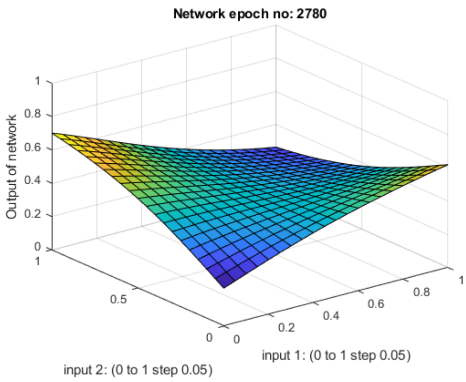

Figure 5: plot

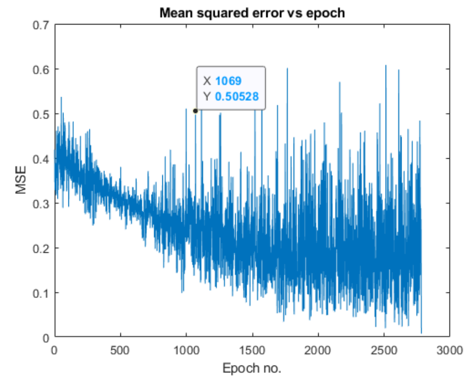

Figure 6: error

1.4 Set the noise level to 0.2. Change the weight spread to 5. What can you observe? Explain your results using the delta-rule..

With the noise still being big to confuse the network on the learning process and by having more distance between the weights, the network does not find the ideal goal although it finds values in which the total error becomes less than 0.01.

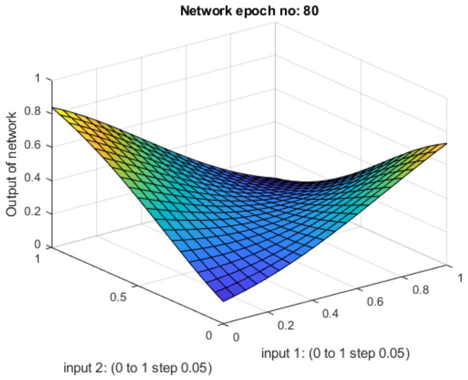

Figure 7: plot

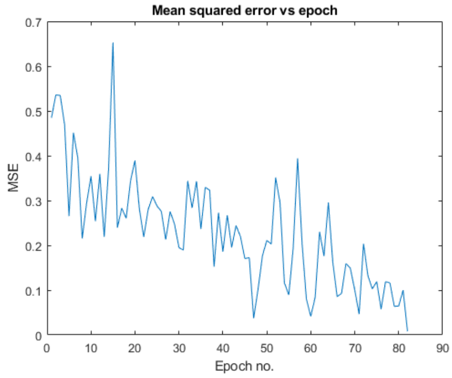

Figure 8: error

1.5 Set the noise level to 0.01. 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 two qualitatively different solutions to the XOR problems. One of them finds the goal result very quickly and the other other skips it and never finds this. This is because: 1) The noise is so small it barely has any effect on reducing the generalization error and 2) the weight spread is too bigger which is a problem for the initialization and may be responsible for causing the vanishing gradient situation on the neural network.

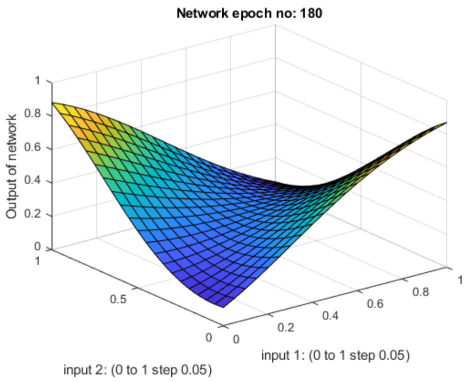

Figure 9: The first solution.

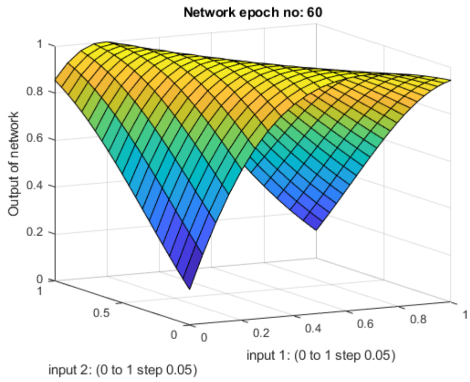

Figure 10: The second solution.

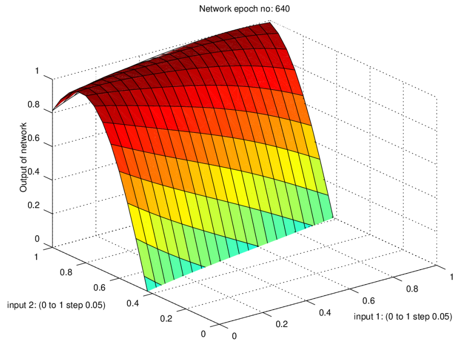

Figure 11: Network did not find the solution

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

The shapes that the graph usually have is of an hyperbolic paraboloid since it the network tries to approximate to the goal values - and thus minimize the error- by appling a gradient descendant learning rules in which the weights are updated .

2 Another function

2.1 Is the network capable of learning the sine function?

As you can see in the figures below, the network is capable of learning the sin function.

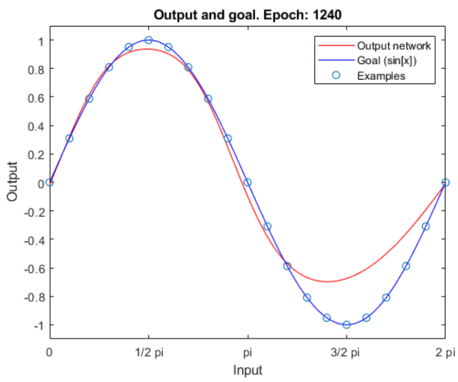

Figure 12: Plot of learned sine

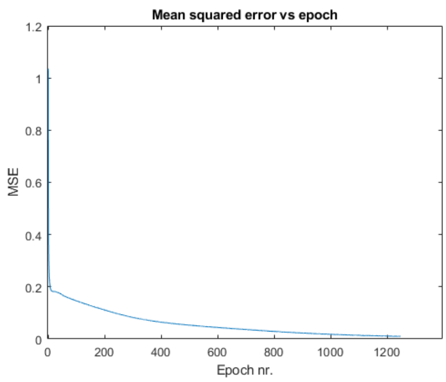

Figure 13: The corresponding error plot

2.2 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?

With less sample points of the sin, the representation of the sin is still very accurate.

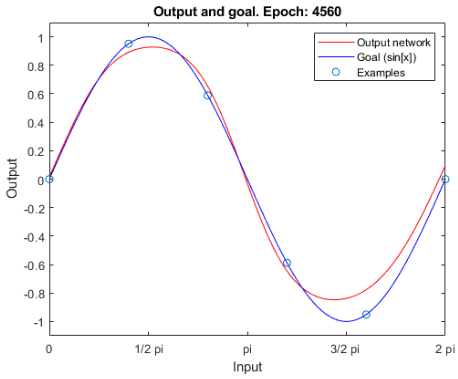

Figure 14: Learned sine with 5 example points

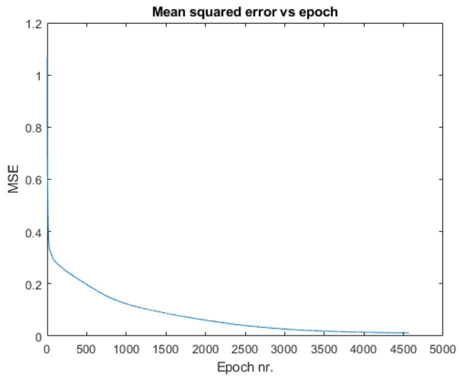

Figure 15: Error of learned sine with 5 example points

2.3 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 of the pi goes from 0 to 2π. Outside of the domain, we can see that the output quickly becomes less accurate.

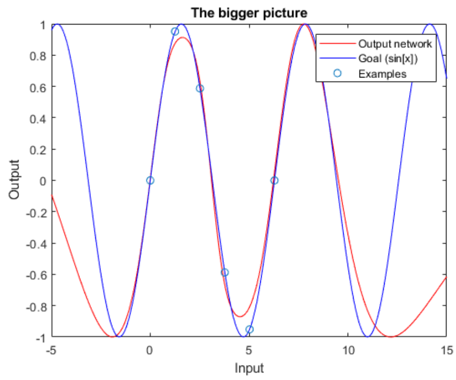

Figure 16: Extended plot of sine

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

After a lot of simulations, we were able to learn the sine with 3 hidden neurons. It was impossible for us to learn the sine with only 2 hidden neurons.

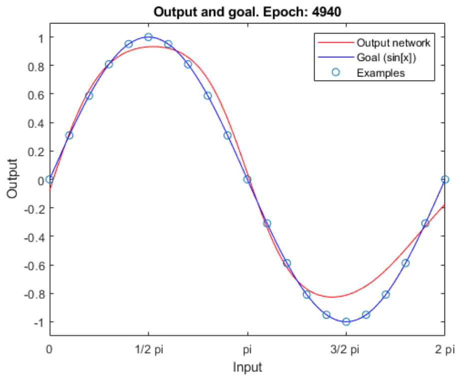

Figure 17: Plotting sine with 3 hidden neurons

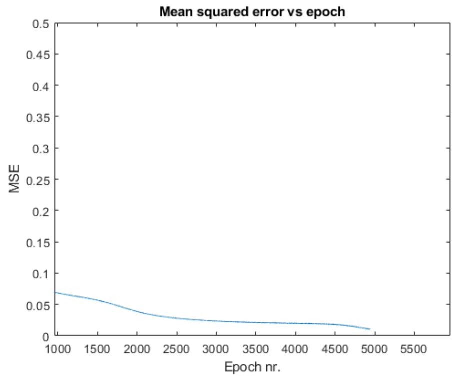

Figure 18: Error sine with 3 hidden neurons

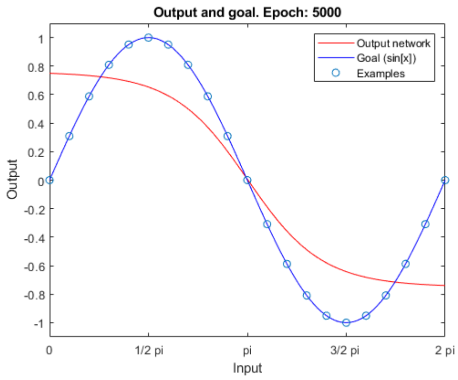

Figure 19: Plotting sine with 2 hidden neurons

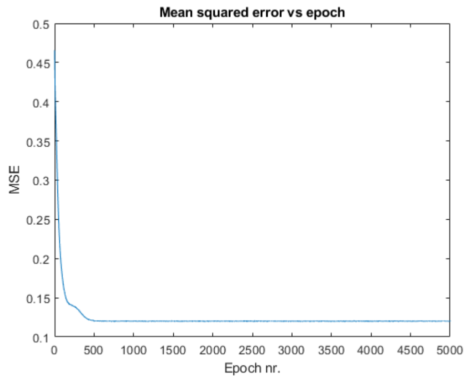

Figure 20: Error sine with 2 hidden neurons

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

We had to modify the output function for this assignment in order to correspond with the newly set goals which are both based on the sin function.

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

Listing 1: output_function.m

3 Code

```
% 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

mean_weight = 0;
weight_spread = 5;

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;
min_error = 0.01;

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
    %not sure
    output_activation = hidden_output * w_output ;

    % Compute the output
    output = output_function(output_activation);

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

    % Compute local gradient of output layer
    local_gradient_output = d_sigmoid(output_activation).*(goal(pattern) -output);


    % Compute the error on the hidden layer (backpropagate)
    hidden_error = 0 ;

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

    % Compute the delta rule for the output
    delta_output = learn_rate * transpose( hidden_output) * local_gradient_output  ;

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

    % Update the weight matrices
    w_hidden = upW(w_hidden, delta_hidden);
    w_output = upW(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) = sum(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

```

```
    % Implement a stop criterion here
    if h_error(epoch) < min_error || epoch == max_epoch

        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])
        pause(0.01)
    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 2: mlp.m

```
clear all
close all

% The number of examples taken from the function
n_examples = 5;

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

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

% Parameters for the network
learn_rate = 0.05    ;                % learning rate
```

```
max_epoch = 5000;              % maximum number of epochs


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;
min_error =0.01;

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
        %not sure
        output_activation = hidden_output * w_output;

        % Compute the output
        output = output_function(output_activation);

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

```
        % Compute local gradient of output layer
        local_gradient_output = d_sigmoid(output_activation).*(goal(pattern) -output);


        % Compute the error on the hidden layer (backpropagate)
        hidden_error = 0 ;

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

        % Compute the delta rule for the output
        delta_output = learn_rate * transpose( hidden_output) * local_gradient_output  ;

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

        % Update the weight matrices
        w_hidden = upW(w_hidden, delta_hidden);
        w_output = upW(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) = sum(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 h_error(epoch) < min_error || epoch == max_epoch

        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 network')
        hold on
        plot(input,sin(input),'b-','DisplayName','Goal (sin[x])')
        hold on
        scatter(examples, goal, 'DisplayName', 'Examples')
        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
        pause(0.01)
    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');
    hold off
    xlabel('Input')
    ylabel('Output')
    legend('location','NorthEast')
    title('The bigger picture')
end
```

Listing 3: mlp_sinus.m