Assignment 3:MLP

Dekel Viner(S2612925),Luis Knigge(S2560224)

September 3, 2019

1 Theory Questions

a) the credit assignment problem is the issue of determining how to credit or blame each hidden unit in its contribution to the output unit error.

b) The target output and the input.

c) The sigma function: $\sigma(a) = \frac{1}{1+exp(-(a-\theta)/\rho}$. The sigma function replaces the step function and returns a value between 1 and 0 to differentiate between outputs. This allows the derivative of the output or the sigma function to be taken. $\sigma\prime = \frac{1}{p}\sigma(1-\sigma)$

d) When the weights are all initialised with high values it would be difficult to answer the credit assignment problem using backpropagation because the activation would be high all across and all the hidden nodes will fire.

e) criteria 1: When epoch number reaches a certain pre specified value. The learning will may continue long after the answer to the network has been found and is therefor time wise uneffiecient.
criteria 2: When the error is lower then a certain pre specified value. Difficult to asses how low the error should be inorder to be determined sufficiently correct.
criteria 3: Cross-Validation

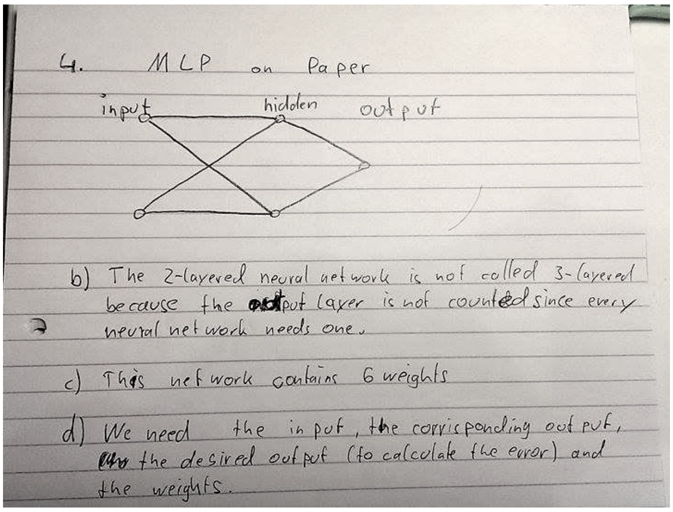
f) Reducing the value or rho in the sigma function would cause the sigma function curve to be steeper pushing the output closer to the extreme values of 1 and 0. This would then "radicalise" the error attributed to each node, meaning that nodes that contribute a lot to the error will now have a even higher error and therefor increase the overall change in weights per epoch.

g) By plotting the output of the network vs the training set and observing whether the network simply captures the overall feel of the training set curve and does not cover it entirely.

h) The incorporation of too much detail of a particular training data set in the identification of the weights, new data will not be accurately described.

i) Network prunning is the reduction of nodes inorder to achieve a more effiecient network. It can be accomplished by making each one of the hidden nodes irrelevant at a time by changing the incoming and outgoing weights to 0 and observing the effect it has on the error. The node that has the lowest effect can taken out.

2 A MLP on paper


4. MLP on Paper

input hidden output

b) The 2-layered neural network is not called 3-layered because the input layer is not counted since every neural network needs one.

c) This network contains 6 weights

d) We need the input, the corrisponding output, the desired output (to calculate the error) and the weights.

3 Testing the MLP

a) No, it is not guaranteed that the MLP will find the right solution. In some of the cases the decision plane will stay flat and the error will only decrease ever so slightly. A reason for this might be that the randomly chosen starting weights are to far away from the necessary weights that would make the MLP do its job.

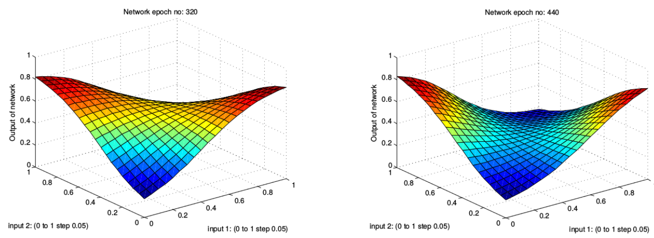
b) The MLP needs ca. 2100 to 2500 epochs to get an epoch_error below 0.1.

c) When the noise level is set to 0.5 the MLP will still find a solution but it takes about twice as long. Furthermore is the movement of the plane over the epochs very shaky and the error gets a lot more noisy during the end. The MLP finds a solution because the magnitude of the error becomes so big that the threshold to stop the mechanism is triggered. If you have a look at the final solution you can see that the shape of the plane is similar to a settle, a bridge or a valley, but is not so steep and the range of the outputs is about half as big in comparrison to the original network with a nois level of 0.05.

d) We can observe that the learning of the network is much faster. It finds a solution in around 300-500 epochs. The error is also a lot noisier. The noisy error can be explained by the noise level which influences the input data. The fast learning can be explained by the weight spread acting on the delta rule. The weight spread increases the range of the randomly chosen first weights and therefore their absolute value. These weights increase then the activation within the network and the errors. This will then increase the local gradients and therefore the delta-weights. This has then a direct influence on the speed of learning. What happens with the outputs is that they are further out from the center of the sigmoid functions which means that the the sigmoid function acts more like a step function.

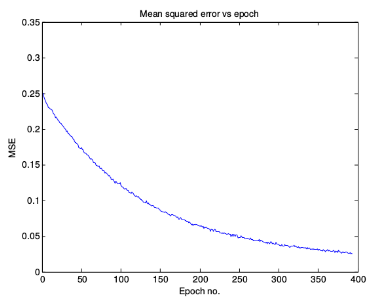
e) The two types of correct solutions are the bridge shape (left) and the valley shape (right). They emerge because they are both states with a very low error. Which of the shapes it is going to be depends on the initial values of the weights.

Figure 1: Possible shapes of the output plane



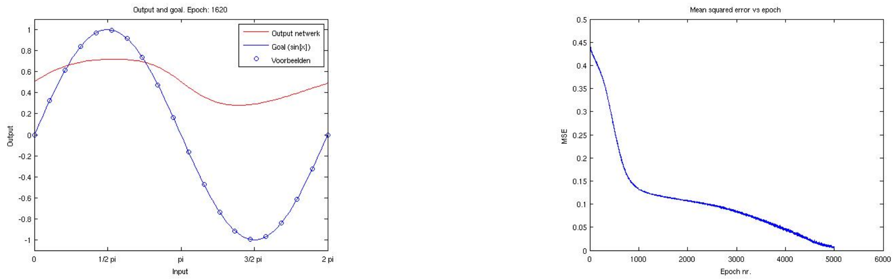
f) The shape of the error function is asymptotic towards the x-axis. This happens because the smaller the error the smaller is the delta-weights. The error influences the local gradient to become smaller which then decreases the change in weights.

Figure 2: Error vs. Time (epochs)



4 Another Function

a) The network seems not able to simulate the sinus function because the output graph is not aligned with the goal graph, but it still looks like a sign function. However the error function converges to 0.



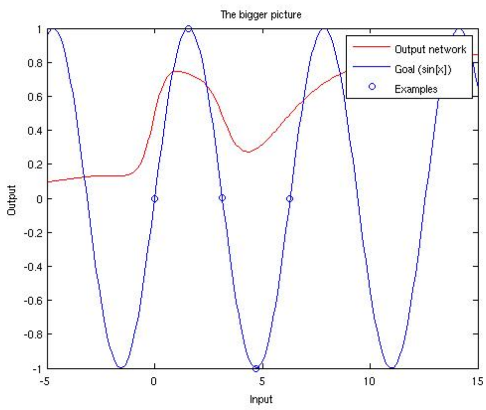
(a) Output (b) Error

Figure 3: Graphs for MLP Sinus

b) If only 5 example points are chosen the function becomes discrete for the computer because the point are lying either on the x-axis (output=0) or at the min-/maximum (output = 1 or -1).

c) If we look at the data points further out of the range for which we computed the network we can see that the output does not continue to swing between 0 and 1 as the sinus function does but rather stay at one of the values. Networks like this can only model these kind of functions in a certain range,

Figure 4: Big Domain for MLP output and comparison data($sin(x)$)



d) The XOR learning network still works, even when we plugged the changed code from the mlp_sinus into the mlp_2011. This is because the only difference is that the output is not transformed by the sigmoid function anymore so it is not so clear anymore to which output a certain input belongs but it is still enough. The network will not find so steep results as in the original network but the network will still give the right answers.

e) The network still works the reason for that being that by changing the output to the activation and not to a value between 0 and 1 that comes out of the sigma function retains the correlation in the outputs and can still be generalized. Using the sigma function would help to better differentiate between different outputs and would produce a more clearer graph but is not necessary in this senario.

5 Code

- MLP

mlp_2014.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

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 = 0.1;

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 = sigmoid(output_activation);

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

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

```
        % Compute the error on the hidden layer (backpropagate)
        hidden_error = local_gradient_output * w_output';

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

        % 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
    log_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 || epoch_error < min_error
        stop_criterium = 1;
    end


    % Implement a stop criterion here

    % 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,log_error)
title('Mean squared error vs epoch');
xlabel('Epoch no.');
ylabel('MSE');

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

- Sigmoid functions

sigmoid.m

```
%this functions calculates the sigmoid
function [output] = sigmoid(x)
row = 1.0;
   output = 1./(1+exp(1).^(-x/row));
end
```

d_sigmoid.m

```
%this functions calculates the differential of the sigmoid
function [output] = d_sigmoid(x)
    temp = sigmoid(x);
    output = temp .* (1 - temp);
end
```

- Output function

output_function.m

```

function [output] = output_function(x)
    output = sigmoid(x);
end
```

- MLP Sinus

mlp_sinus.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 = 0.05;                  % learning rate
max_epoch = 5000;                % maximum number of epochs


mean_weight = 0;
weight_spread = 1;

n_input = size(examples,2);
n_hidden = 20;
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 = 0.1

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_function2(output_activation);

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

        % Compute local gradient of output layer
        local_gradient_output = d_output_function2(output_activation)*output_error;      %use d_output_function

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

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

        % 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_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 || epoch_error < min_error
        stop_criterium = 1;
    end

    % Add your stop criterion here

    % 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,log_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
```

- Output functions for MLP Sinus

output_function2.m

```

function [output] = output_function2(x)
    output = x;
end
```

d_output_function2.m

```
function [output] = d_output_function2(x)
    output = 1;
end
```