Neural Networks

Assignment 2 - The Multi-Layer Perceptron

George Petridis (s3001393)
Jussi Boersma (s2779153)

March 9, 2017

3 Theory Questions

3.a What is the credit assignment problem?

The credit assignment problem constitutes the challenge of assigning the right weights to the hidden nodes that produced the error in the output. Since the output layer is connected to many hidden nodes, it is difficult to know with certainty how much each of the hidden nodes has contributed to the output error.

3.b What are the two factors that determine the error of a neuron in a hidden layer after the forward pass has finished?

The error of a neuron in a hidden layer after the forward pass has finished is determined by the error of the hidden nodes and by the error in the output that these hidden nodes have contributed to.

3.c Give the mathematical definition of the sigmoid function. Clearly state which symbols are constants and what the variable is. How do the constants of the function affect the shape? Also write down the derivative.

The mathematical definition of the sigmoid function is:

$$\sigma(\alpha) = \frac{1}{1+e^{-(\alpha-\theta)/\rho}}$$

Where:

- α is the node's activation
- e = 2.7183 (mathematical constant)
- ρ determines the shape of the function (large values make the curve more flat, while small values make the curve to rise more steeply (here we implicitly assign to it the value 1)
- θ is the neuron's threshold

The sigmoid's derivative is given by the function:

$$\sigma'(\alpha) = \frac{1}{\rho} \cdot \sigma(\alpha) \cdot (1\text{-}\sigma(\alpha))$$

3.d Why is it impractical to initialize the weights with very high values?

The weights should not be initialized with very high values, because this could cause the nodes to reach outputs close to their extreme values, which is called premature saturation. In such cases the nodes may be thought to be pre-trained to an arbitrary value for that pattern and in order to produce useful outputs, they will first have to undergo a process of unlearning.

3.e Describe three criteria that you can use to determine when to stop learning. Provide one disadvantage per criterion.

In order to determine when to stop learning we can use the following three criteria:

- Accept any error for which all output nodes have responses to all patterns that are closest to the correct Boolean value as defined by the target, however this can be defined only for Boolean training.
- Prescribe some very low value ϵ for the mean pattern error e_p, which we should decrease accordingly if we are interested in the number of bit errors in the network. However we do not know what the right value for ϵ is. This approach assumes the net can decrease its mean error below the given value of ϵ which is not necessarily the case if the net does not have enough nodes and connections to approximate the required function to the desired degree.
- Stop when the rate of change of the error is sufficiently small. In this case as an error minimum is approached, the error surface becomes shallower and so the change in error at each epoch becomes ever smaller. However it may be the case that a local minimum has been approached instead of a global minimum and this would correspond to a partial solution for the network.

3.f How can you speed up the learning of a network besides increasing the learning rate? Explain.

Besides increasing the learning rate, we can speed up the learning of a network by introducing a momentum term ($\lambda\Delta$w(n-1)) in the learning rule. The momentum constant λ is greater than zero and is also less than 1 to ensure convergence. In this case the learning rule takes the form:

$$\Delta w(n)=\alpha\delta^j(n)x(n)+\lambda\Delta w(n-1)$$

Using this relation recursively, w(n) can be expressed as a sum of gradient estimates which are evaluated at the current and previous training steps. Therefore, if over a number of epochs these estimates are consistently of the same sign, the weight change will increase as previous gradient estimations contribute cumulatively to the current update. This is when we say that the network gathers momentum.

3.g How can you verify that a network is generalizing for some training data?

In order to verify that a network is generalizing for some training data we should present to it a pattern that it has not seen before. If the network classifies the unseen pattern correctly, then this means that it is generalizing well.

3.h Describe the problem of overfitting.

Overfitting can occur when the network has too many hidden units and too much freedom to choose its decision surface. This way it may be incorrectly classifying patterns in order to take care of all the noise and complexities in the data without regarding their underlying relationships.

3.i What is network pruning? Describe at least two ways of accomplishing network pruning.

Network pruning is the process by which non-important weights are removed from the network, leaving this way only those that are important to model the underlying data trends.

One way of pruning is by weight decay, where at each iteration step a contribution is delivered that decreases every weight by a specific uniform amount. In this case both large and small weights decay at the same rate and the important weights for the network's correct functioning are allowed to grow, while the non-important ones decay to zero and are removed.

Another way of pruning is the reduced error pruning, where each node is replaced with its most popular class and if the output accuracy is unaffected then the change is kept and we proceed to the next one.

4 An MLP on paper

4.a Draw a two-layer neural network with 2 input neurons, 3 hidden neurons and 2 output neurons. Enumerate the neurons from top to bottom.

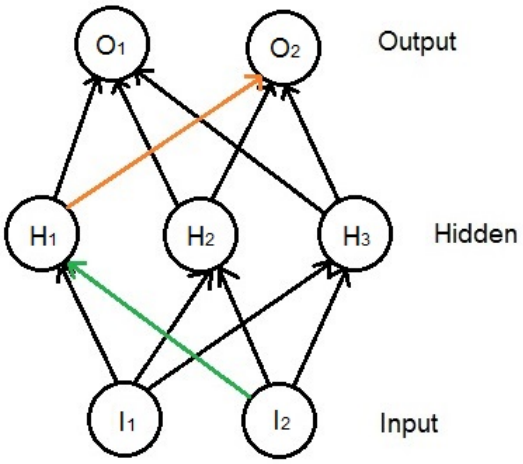


Figure 1: A two-layer neural network

4.b Why is this considered a two-layer network instead of a 3-layered network?

This is considered a two-layer network because only the hidden and the output layers have active nodes. The input layer is not counted, as it does not perform any computations.

4.c What are the dimensions (number of rows and columns) of W^h and W^o?

The dimensions of W^h are 3x2 and the dimensions of W^o are 2x3.

4.d How do the weights in a single column of a weight matrix relate to each other?

For the W^h matrix the weights of a single column correspond to all the connections from the same input node to one hidden neuron, while for the W^o matrix the weights of a single column correspond to all connections from the same hidden node to one output neuron.

4.e Mark $w^h{}_{12}$ and $w^o{}_{21}$ in your drawing by giving the connections a color.

We have depicted $w^h{}_{12}$ with a green arrow and $w^o{}_{21}$ with an orange arrow in Figure 1.

4.f Extend the drawing of your network by creating an augmented input layer. What are the dimensions of W^h and W^o now?

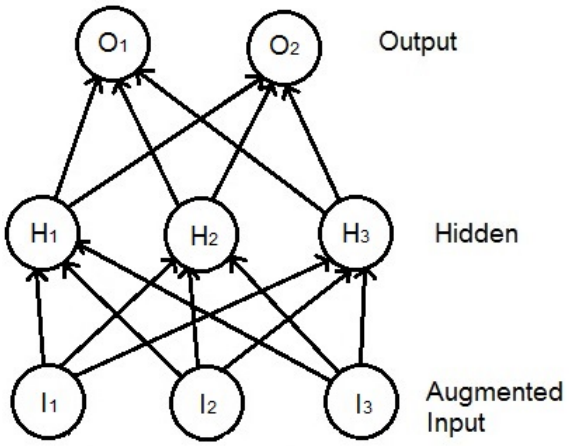


Figure 2: Extended neural network with an augmented input layer.

The new dimensions of W^h are now 3x3 and the dimensions of W^o are 2x3.

4.j How do we compute the activation of the output layer α^o?

The activation of the output layer can be computed as follows:

$$\alpha^o = \sum w_{ji} x^p{}_i$$

5 Implementing an MLP in Matlab

5.1 Computing the activation (forward pass)

5.1.1 The hidden layer

We computed the hidden_activation using matrix vector multiplication:

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

5.1.2 Determining the output at the hidden layer

The hidden layer uses a logistic function to compute the output $\sigma(\mathrm{x})= \frac{1}{1+e^{-x}}$. This function is implemented in sigmoid.m as follows:

```
%This functions calculates the sigmoid
function [output] = sigmoid(x)
    % Define the sigmoid function here
     [output] = 1.0 ./ ( 1.0 + exp(-x));
end
```

5.1.3 The activation of the output

The activation of the output layer is computed by:

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

5.1.4 The output function

Here we have implemented the output function such that it uses the sigmoid function for now:

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

5.2 The backward pass: backpropagation

5.2.1 The slope: $\frac{d}{dx}sigmoid(x)$

Here we have implemented the derivative of the output function:

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

5.2.2 The local gradient at the output layer

By using the goal matrix and the output of the output layer we determined the local gradient at the output layer:

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

5.2.3 The local gradient at the hidden layer

We used the local gradient at the output layer, the matrix W^o and the activation a^h to compute the local gradient at the hidden layer:

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

5.2.4 Computing ΔW

We computed ΔW^o and ΔW^h as follows:

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

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

5.2.5 Updating the weight matrices

Finally we updated the weight matrices:

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

5.3 Implementing the stop criterion

So that the while loop terminates when the error drops below a certain value, we have created a parameter named min_error and set it to 0.01.

```
% Implement a stop criterion here
min_error = 0.01;  % minimum error for the while loop to stop

if h_error(epoch) < min_error
    stop_criterium = 1;
end
```

6 Testing the MLP

Here we set the learn_rate to 0.2, the number of hidden neurons to 2 and the noise_level to 5% = 0.05.

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

With these settings we see that it is not guaranteed that the network finds a solution, as it is not able to minimize the error close to the desired value of 0.01. With only 2 neurons the network reaches the maximum number of epochs (5000) and terminates, before finding a solution.

6.b (Here we have set the number of hidden neurons to 20 How many epochs are needed to find a solution?

Now the network finds a solution after around 2700-3000 epochs.

6.c Set the noise_level to 0.5. Explain what happens.

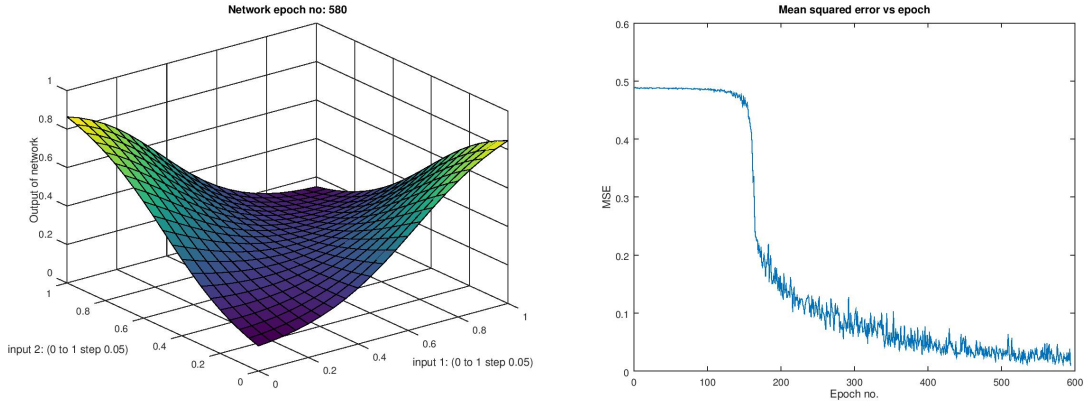
By setting the noise_level to 0.5 we see that the network becomes unstable, having large spikes during its calculations. Eventually, after the maximum number of 5000 epochs has passed the network is not able to reach a solution.

6.d 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 network now reaches a solution much more quickly (less than 600 epochs).

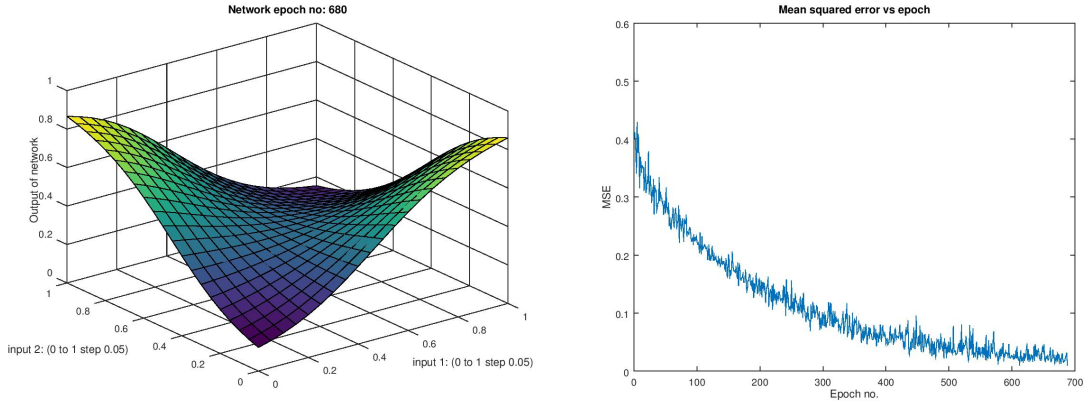
6.e Set the noise_level to 1%. Leave the weight_spread at 5. There are two qualitatively different solutions to the XOR problem. What are these two? Include a figure of both solutions.

In Figure 3 and Figure 4 we can see the two qualitatively different solutions to the XOR problem that we get by using these settings:



(a) (b)

Figure 3: First solution to the XOR problem



(a) (b)

Figure 4: Second solution to the XOR problem

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

The graph usually has the shape of the error presented in Figure 4.

7 Another function

For this section we implemented the for-loop and the stop-criterion from the file mlp.m into the file mlp_sinus.m. We changed the output function accordingly, so that the network could also put out negative values:

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

In this case the derivative of the output function was also changed accordingly:

```
%this functions calculates the differential of the output function
function [output] = d_output_function(x)
    output = 1;
end
```

And subsequently the derivative of the output function was used in the delta rule:

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

The learning rate was set to 0.05 and the maximum number of epochs to 5000. In addition we used 20 neurons in the hidden layer.

7.a Is the network capable of learning the sine function?

As seen from Figures 5 and 6, the network was capable to learn the sine function with these settings.

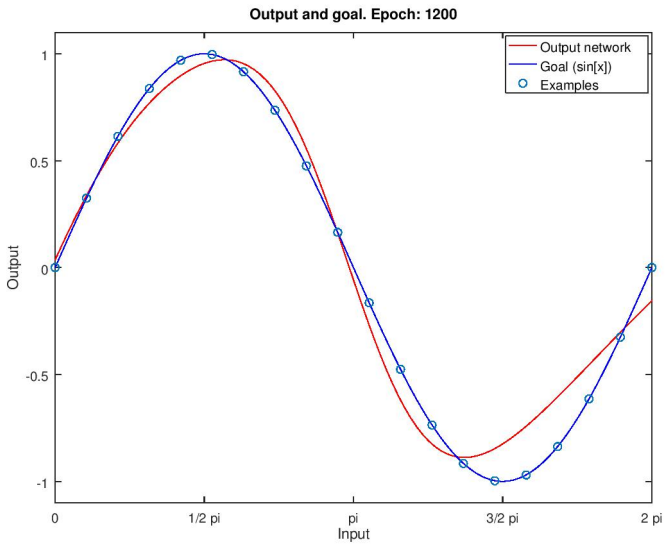


Figure 5

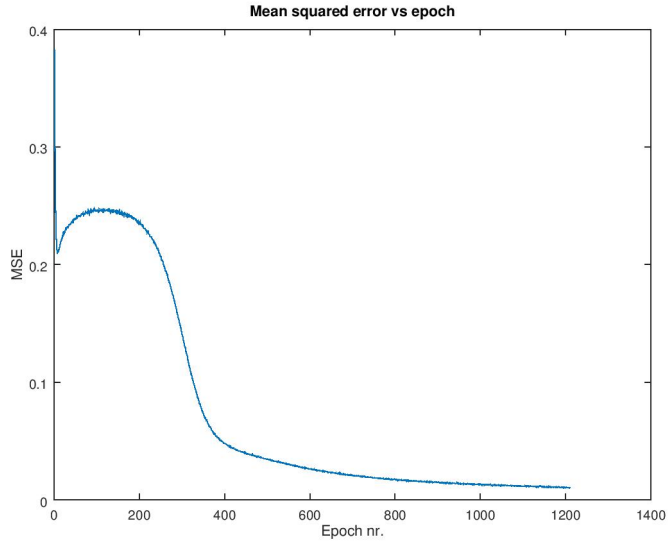


Figure 6

7.b 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?

As we see from Figures 7 and 8, with only 5 examples the network is still able to learn the sine function, although this time it does this in a slower manner (it reaches the maximum number of 5000 epochs). This phenomenon corresponds to the feature of generalization, which allows neural networks to extract statistical regularities from the training set and to respond to inputs not seen during training by classifying them appropriately with one of the previously seen patterns.

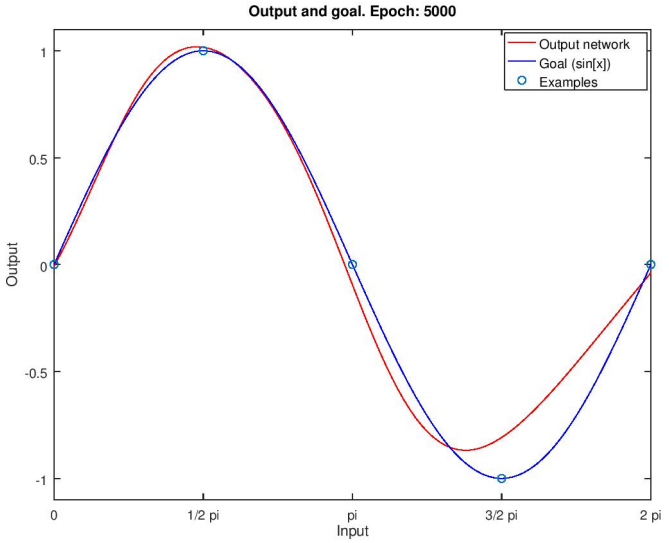


Figure 7

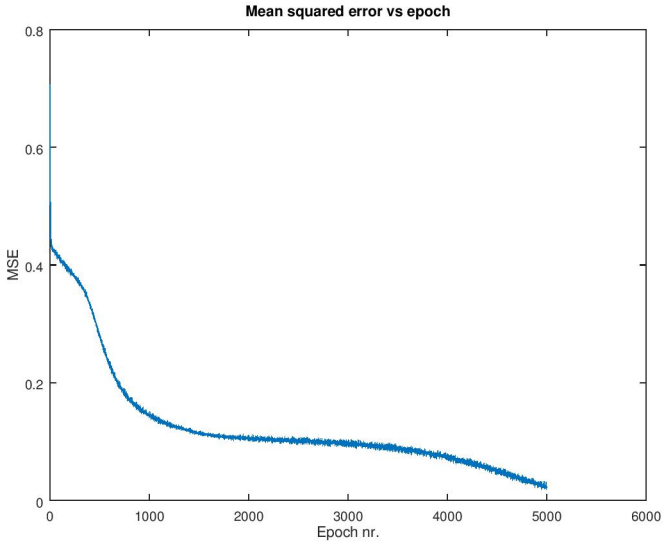


Figure 8

7.c 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 network can be determined by observing the "Bigger picture" plot in Figure 9, where we can see that it is approximately [-1, 6]. As we can see when the input is outside of this domain the output gets values that do not correspond to the target output.

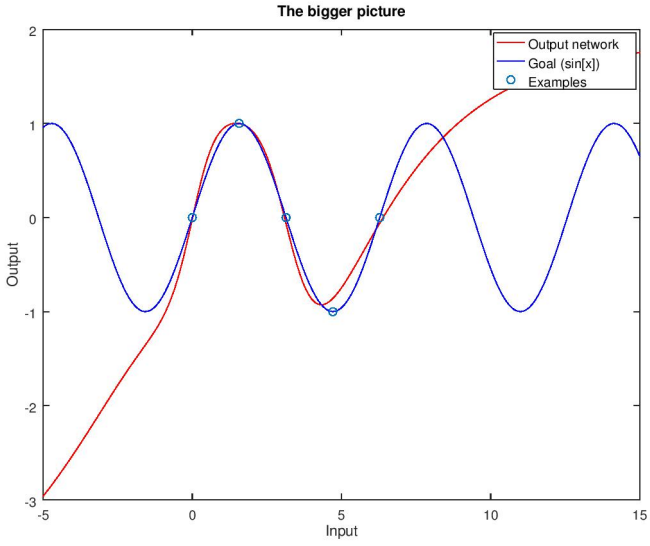


Figure 9

7.d At least how many neurons are required to learn a sine?

After various tries, we found out that in order for the network to learn a sine, at least 4 neurons are needed (3 neurons also seemed to be approximating the learning as well, but not as good as 4 neurons). The results of this number of neurons can be seen in Figures 10, 11 and 12:

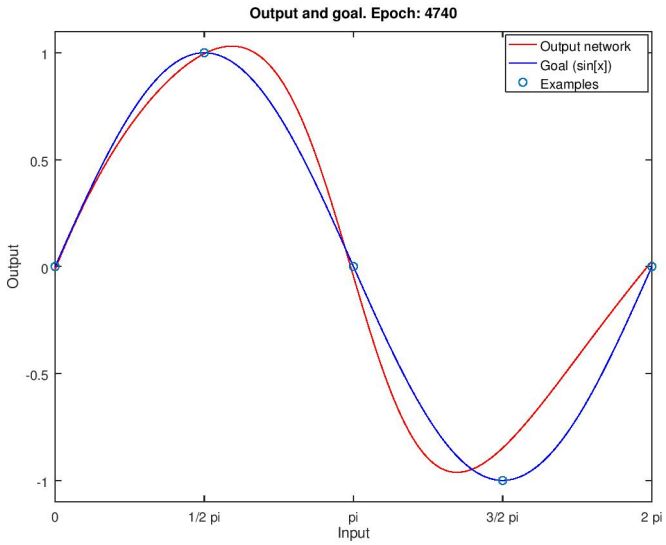


Figure 10

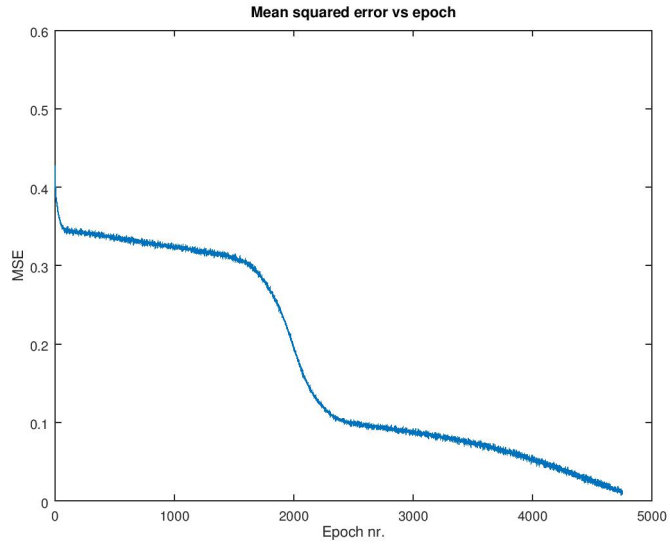


Figure 11

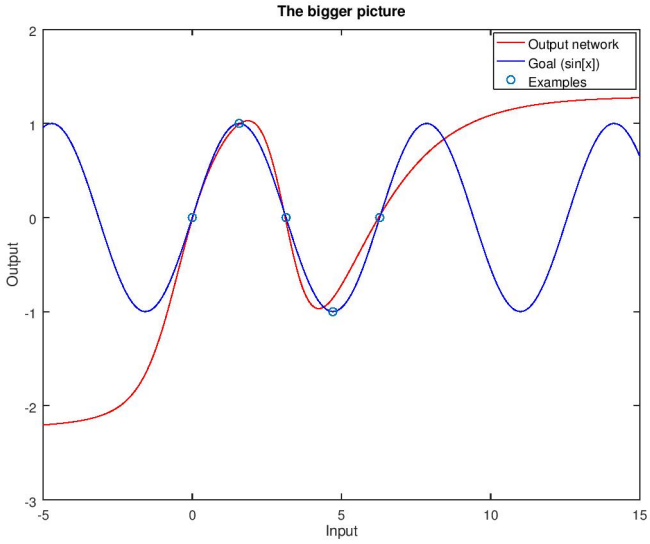
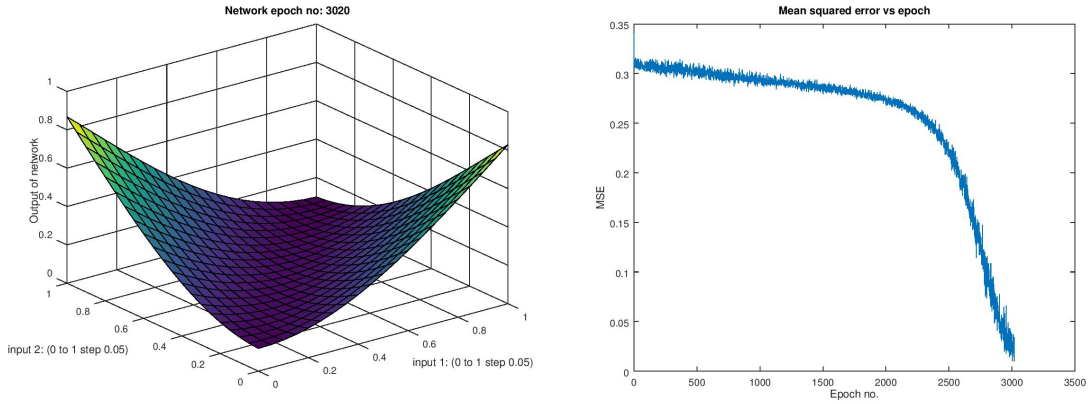


Figure 12

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

The XOR learning network still works if we use as output function f(x)=x as seen from Figure 13. This happens because even though we have changed the output function we are still using the sigmoid to calculate the gradient descent of the hidden layers, which results in the proper training of the network for the XOR function.



(a) (b)

Figure 13