Artificial Neural Networks and Deep Architectures, DD2437

Lab Assignment 1

Learning and Generalisation in feed-forward networks – from perceptron learning to backpropagation

Morgan Brolin, Abhishek Maji and Mukund Seethamraju

Aim/scope

In this lab we will constructed single and multi layer perceptron with batch and sequential learning ; delta and perceptron learning for the single perceptron and the generalized delta rule for the 2 layer perceptron. Constructing the single and double perceptron was done from scratch in python. The multi layer perceptron was constructed using mat-labs net library.

In part one of the assignment, we have developed a single layer perceptron and a double layer perceptron. For the single layer perceptron we assumed(by trial and error) that the best performing eta value was 0.01 for all the learning rules except batch delta learning which used a eta value of 0.003. For the double layer rule both sequential and batch generalized delta learning used a eta value of 0.01 because we assumed that gives the best result.

In the second part of the assignment, we have developed a multi-layer perceptron network for chaotic time-series prediction.
We have run a bench-mark test with Mackey-Glass time series, which is a solution to the following differential equation (with real parameters β and γ, and integer n and delay τ)

$$\frac{dx}{dt} = \beta \frac{x(t-\tau)}{1+x^n(t-\tau)} - \gamma x \quad \gamma, \beta, n > 0$$

We have discretized the above non-linear time delay differential equation by Euler method as follows:

$$x(t+1) = x(t) + \frac{\beta x(t-\tau)}{1+x^n(t-\tau)} - \gamma x(t)$$

In this assignment work, following parameter values are assumed:
$\beta = 0.2$, $\gamma = 0.1$, $n = 10$, $\tau = 25$, $x(0) = 1.5, and\ x(t) = 0\ for\ t < 0$

In this network, we have configured the network to predict $x(t+5)$ from five past values of the time series, that is: $x(t-20), x(t-15), x(t-10), x(t-5), and\ x(t)$. This is an embedded time-lagged representation, which is often used with feed-forward networks operating on time series.
We have picked 1200 points from $t = 301$ to 1500, and used for training, validation and testing.

Network Configuration

In this assignment, we have constructed a multilayer perceptron network with five inputs and one output to handle the time series prediction problem in MATLAB using the command 'feedforwardnet (hiddenSizes, trainFcn)' as shown in the Figure 1:

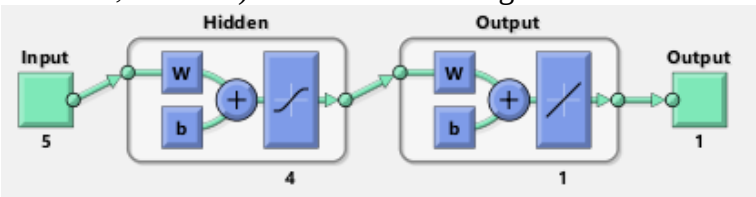
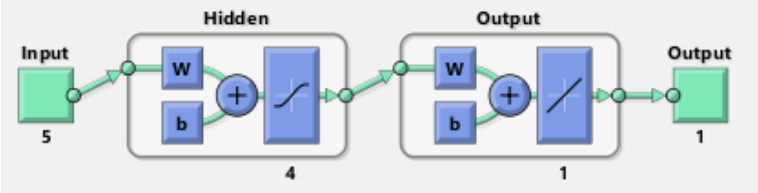


Figure 1: Simulated neural network architecture in MATLAB

We have already configured the dataset in an appropriate manner

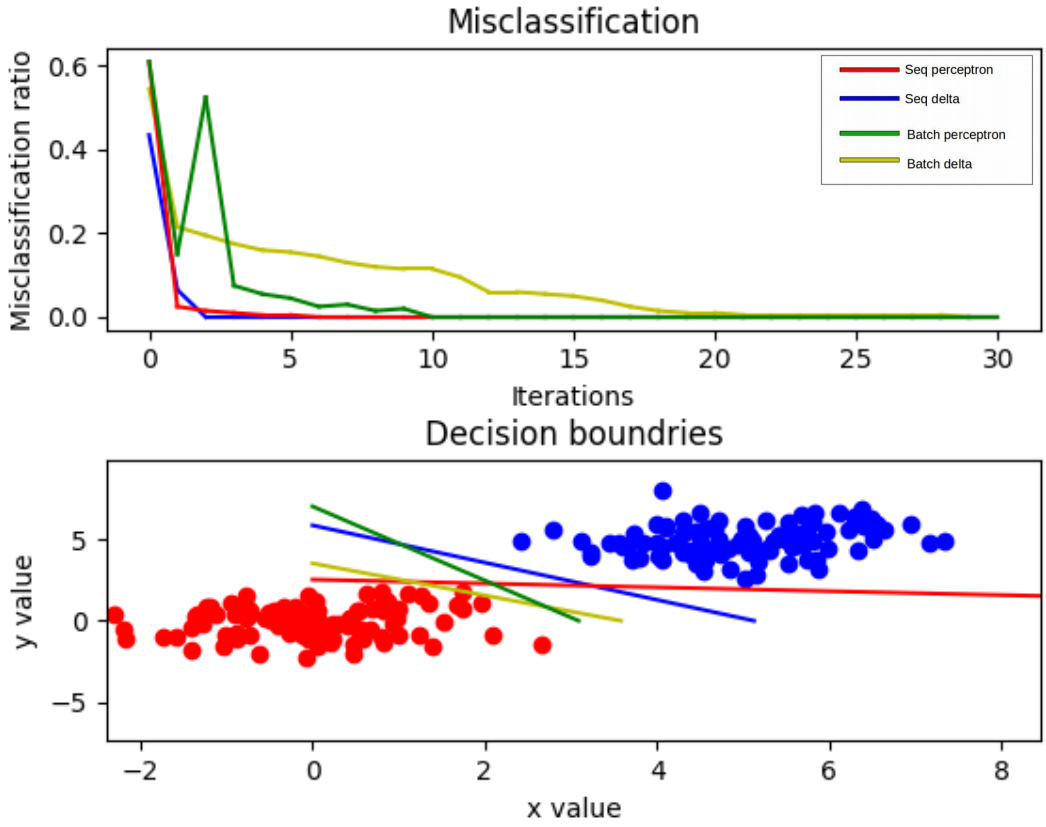


(as requested by the command ‘feedforwardnet (hiddenSizes, trainFcn)’) as described in the previous section.assumptions
We have setup the training process with a batch backprop algorithm and early stopping to control the duration of learning (number of epochs), thereby preventing from overfitting, based on error estimate on the hold-out validation data subset.

Result

3.1.1 generate linear separable data

3.1.2 apply both perceptron and delta rule , both batch and sequential



The upper figure has the y axis representing the percentage of hit ratio of the boundary and the x axis is the number of iterations.

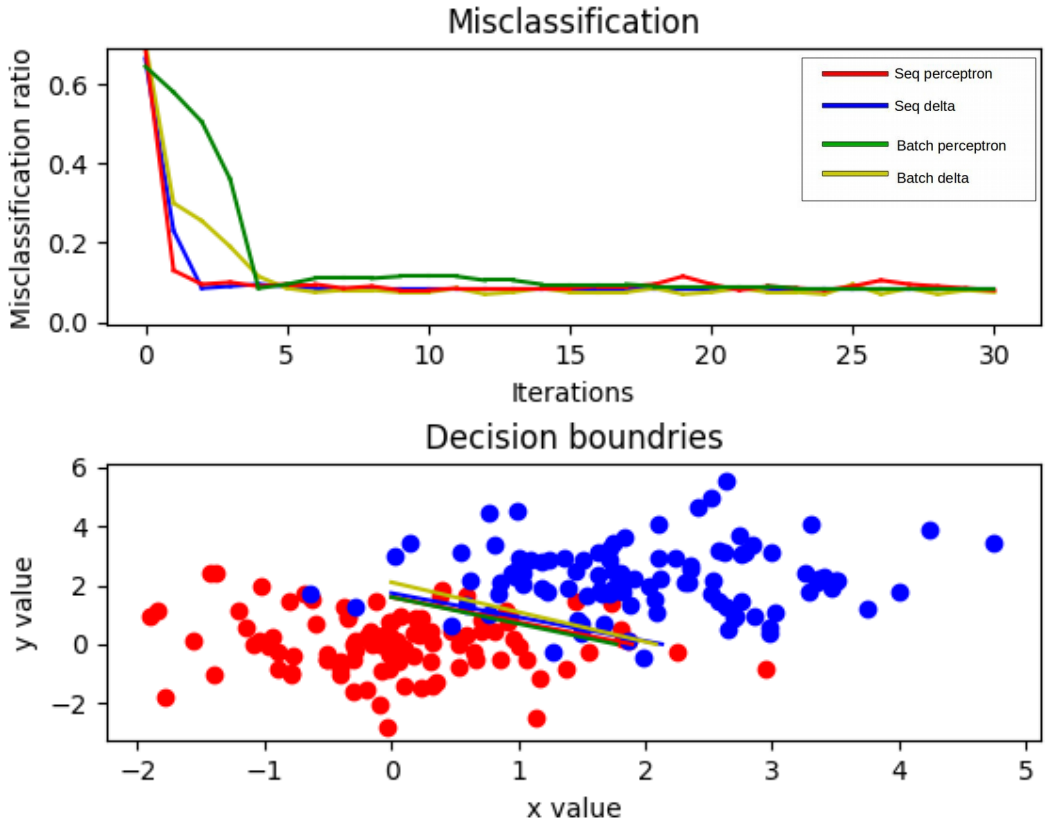
The lower figure has the points and decision boundaries plotted with x and y axis representing the inputs 2 dimensions.

Blue is sequential delta learning , green is batch delta learning , yellow is batch perceptron learning and red is seq perceptron learning.

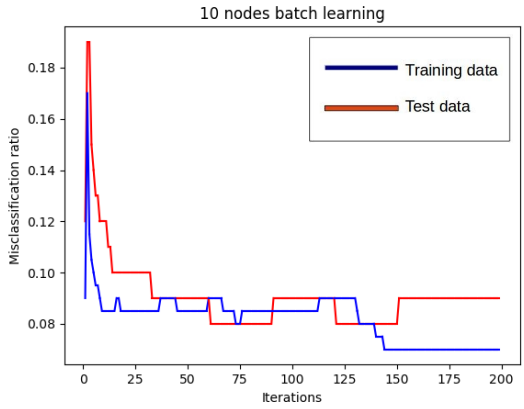
ccsm

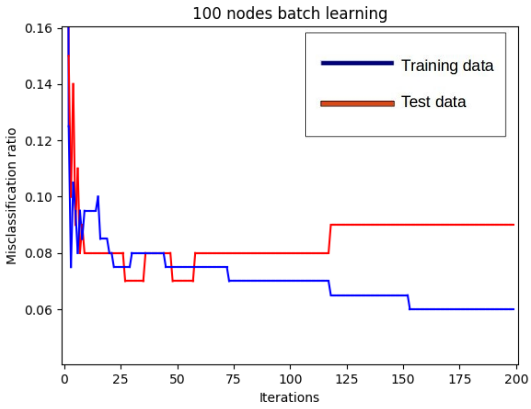
3.13 apply on non separable data

Figures represent the same as 3.1.2 but with non linearly separable data points.

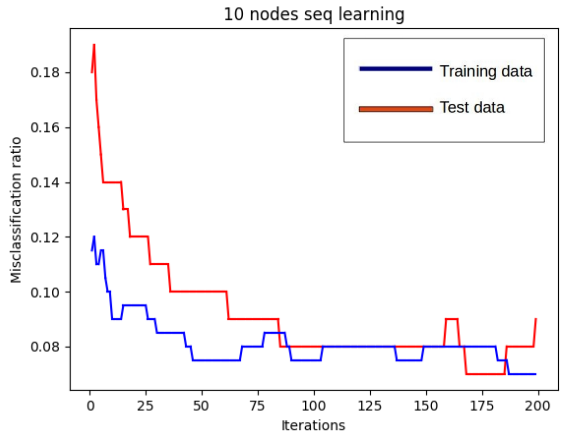


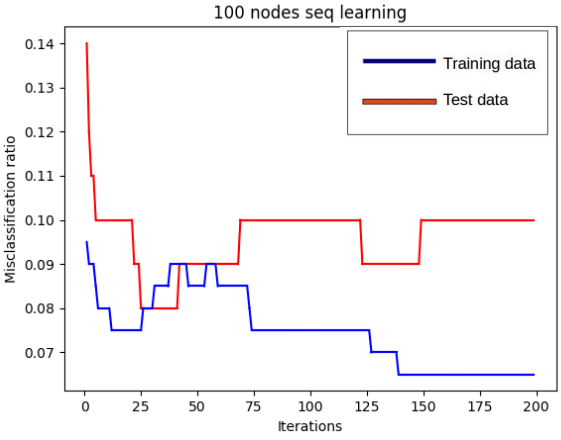
3.2.1 apply a 2 layer perception on the non linearly separable data





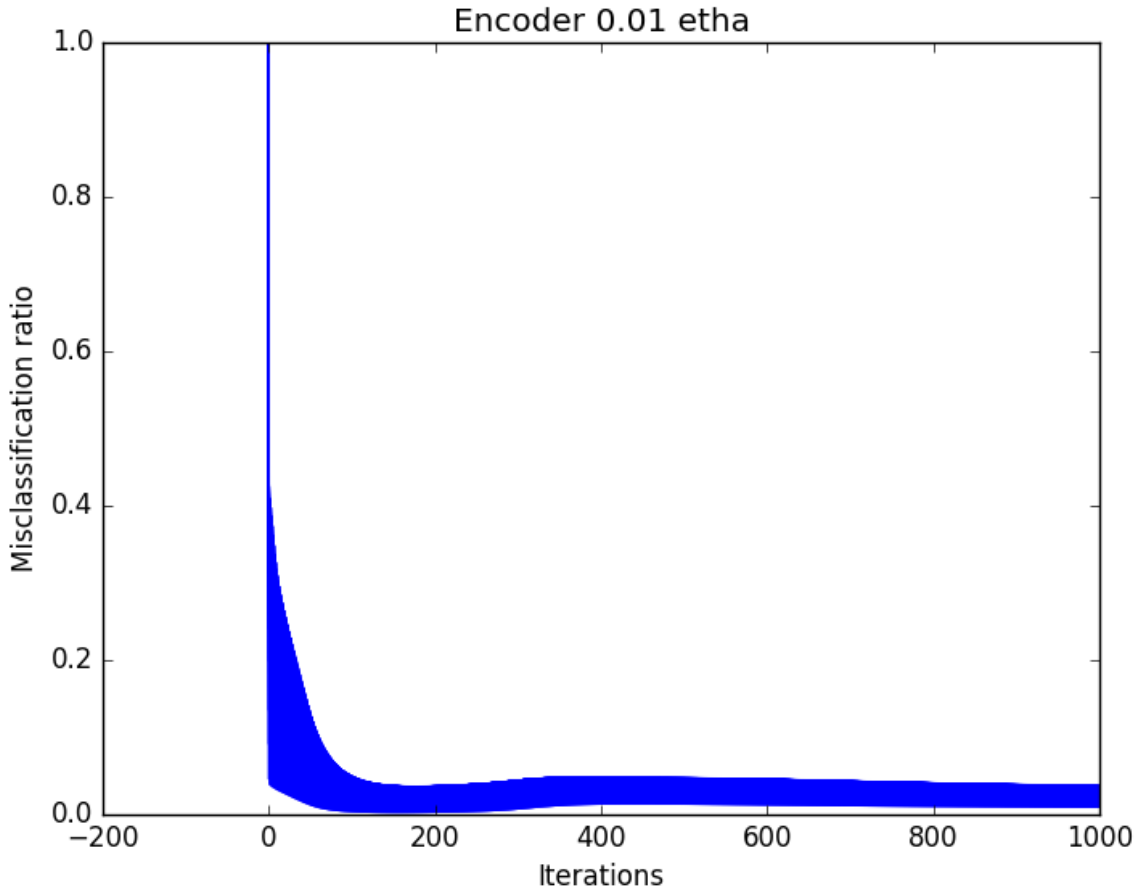
Batch generalized delta learning on the non separable data points, left has 10 nodes right figure has 100 nodes. Eta was 0.01 and iterations was 200





Sequential generalized delta learning on the non separable data points, left has 10 nodes right figure has 100 nodes. Eta was 0.01 and iterations was 200

3.2.2 Encode data



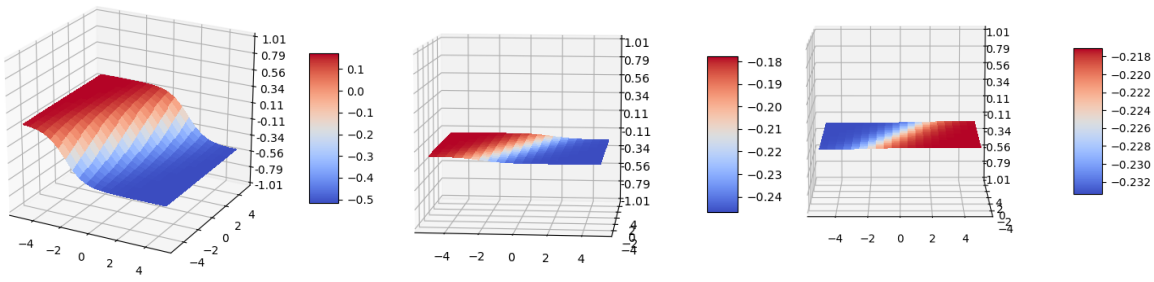
Misclassification ratio: 0.0336240606093%
Input:[-1, -1, -1, -1, -1, -1, -1, 1](Since we didn't shuffle the data points and since we had bias this is the last datapoint in the last iteration)
Output:-0.70632634,-0.63676901,-0.94725753,-0.84560314,-0.64226212,-0.99405245,-0.67868344, 0.70330544

W1 [[1.92959131 -0.57362607 -1.56239632 -0.14613004 1.06388968 -0.33957202
  -0.75138177 1.7690247 -0.65992172]
 [0.20510326 0.33795054 1.73735169 0.08315803 1.29611419 0.83917615
   0.07760361 -0.18794393 -0.77492575]
 [-1.21160533 1.40826485 -0.18575151 -0.07929553 -0.34556404 -2.06460193
   1.3049647   1.00834156 0.18770335]]

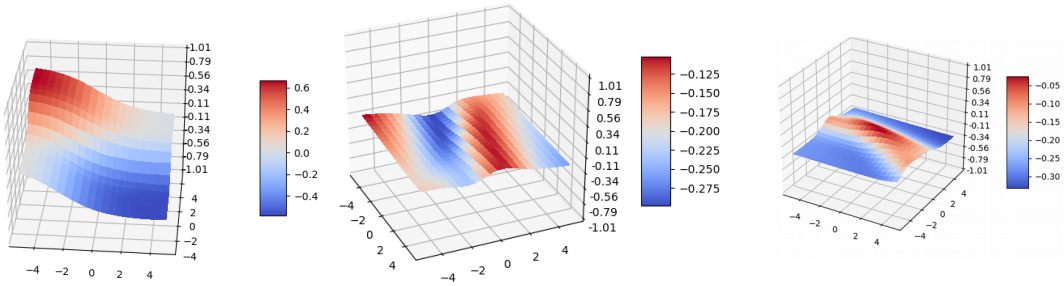
W2[[2.68798077 -0.58618022 -1.34563565 -0.38720626 0.78123571 -1.1950983
  -0.69006346 2.73368198]
 [2.2477886   2.47781358 2.99577053 2.54765771 2.30928078 1.99439504
   2.32715276 1.96850108]
 [-1.80841736 1.81844547 -0.14120811 0.09681383 -0.34149733 -2.70287211
   1.71565217 2.34218319]
 [0.58667943 -0.40441714 0.71416446 0.45518583 1.20028778 -1.82015101
  -0.72056921 0.07592255]]

[-1, -1, -1, -1, -1, -1, -1, 1]
W1 [[1.92959131 -0.57362607 -1.56239632 -0.14613004 1.06388968 -0.33957202
  -0.75138177 1.7690247 -0.65992172]
 [0.20510326 0.33795054 1.73735169 0.08315803 1.29611419 0.83917615
   0.07760361 -0.18794393 -0.77492575]
 [-1.21160533 1.40826485 -0.18575151 -0.07929553 -0.34556404 -2.06460193
   1.3049647   1.00834156 0.18770335]] W1
W2 [[2.68798077 -0.58618022 -1.34563565 -0.38720626 0.78123571 -1.1950983
  -0.69006346 2.73368198]
 [2.2477886   2.47781358 2.99577053 2.54765771 2.30928078 1.99439504
   2.32715276 1.96850108]
 [-1.80841736 1.81844547 -0.14120811 0.09681383 -0.34149733 -2.70287211
   1.71565217 2.34218319]
 [0.58667943 -0.40441714 0.71416446 0.45518583 1.20028778 -1.82015101
  -0.72056921 0.07592255]] W2

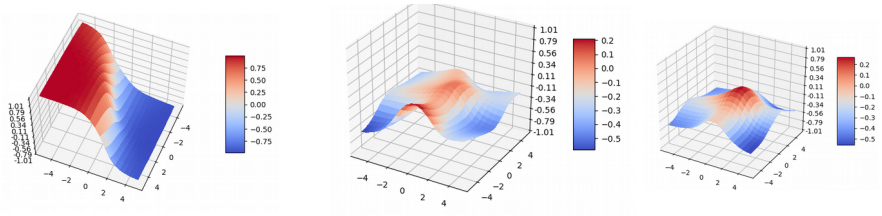
3.3.1 generate data



these are for 1 node after 0,30 and 300 iterations



these are for 5 nodes after 0,30,300 iterations



these are for 10 nodes after 0,30,300 iterations

3.3.2 train the network and visualize result1nodes
3.3.3 evaluate generalized performance
Training with a subset of points(N) and number of nodes(Nodes) to approximate the function in 3.3.1.

Assignment - Part II
In the second part of the assignment, we have developed a multi-layer perceptron network for chaotic time-series prediction.
We have run a bench-mark test with Mackey-Glass time series, which is a solution to the following differential equation (with real parameters β and γ, and integer n and delay τ)

$$\frac{dx}{dt} = \beta \frac{x(t-\tau)}{1+x^n(t-\tau)} - \gamma x \quad \gamma, \beta, n > 0$$

We have discretized the above non-linear time delay differential equation by Euler method as follows:

$$x(t+1) = x(t) + \frac{\beta x(t-\tau)}{1+x^n(t-\tau)} - \gamma x(t)$$

In this assignment work, following parameter values are assumed:

$\beta = 0.2$, $\gamma = 0.1$, $n = 10$, $\tau = 25$, $x(0) = 1.5, and\ x(t) = 0\ for\ t < 0$

Depending on the values of the parameters, this equation displays a range of periodic and chaotic dynamics as shown in the for the aforementioned parameter values.

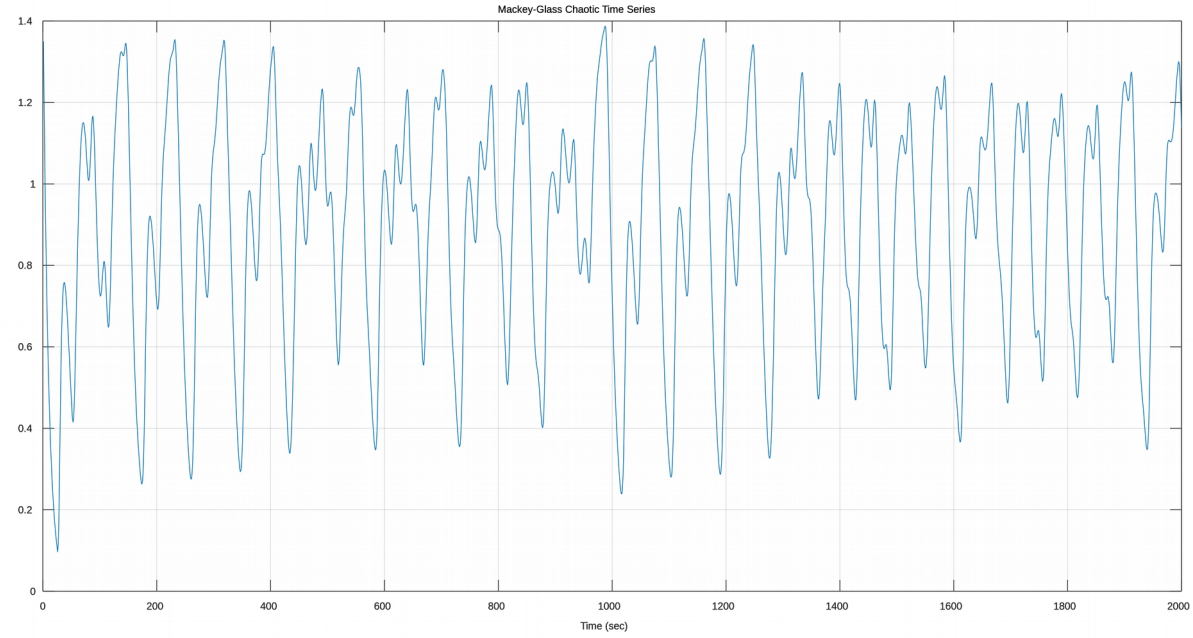


Figure 2: Dynamics in the Mackey-Glass equation

Data

In this network, we have configured the network to predict $x(t+5)$ from five past values of the time series, that is: $x(t-20), x(t-15), x(t-10), x(t-5), and\ x(t)$. This is an embedded time-lagged representation, which is often used with feed-forward networks operating on time series.

We have picked 1200 points from $t = 301 \text{to} 1500$, and used for training, validation and testing.

Network Configuration for task 1

In this assignment, we have constructed a multilayer perceptron network with five inputs and one output to handle the time series prediction problem in MATLAB using the command 'feedforwardnet (hiddenSizes, trainFcn)' as shown in the Figure 1:

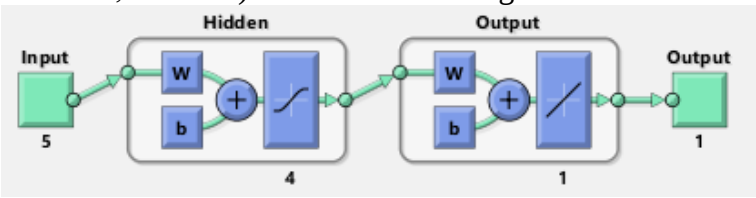


Figure 3: Simulated neural network architecture in MATLAB

We have already configured the dataset in an appropriate manner (as requested by the command 'feedforwardnet (hiddenSizes, trainFcn)') as described in the previous section.

We have setup the training process with a batch backprop algorithm and early stopping to control the duration of learning (number of epochs), thereby preventing from overfitting, based on error estimate on the hold-out validation data subset.

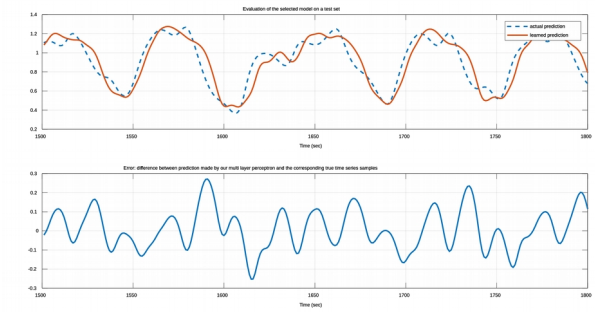
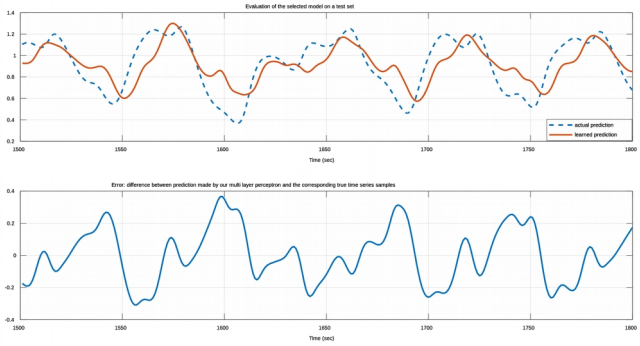
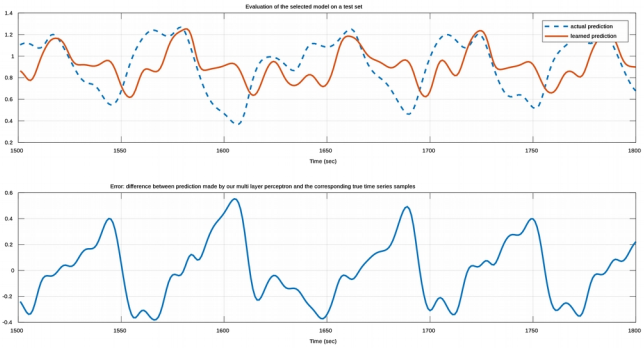
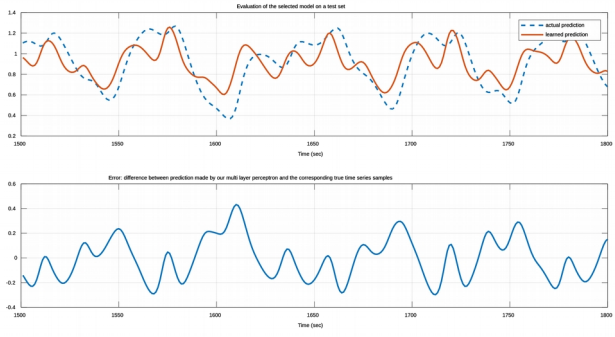
PART-1

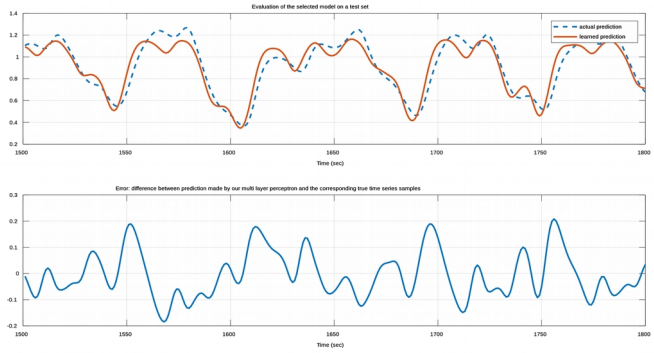
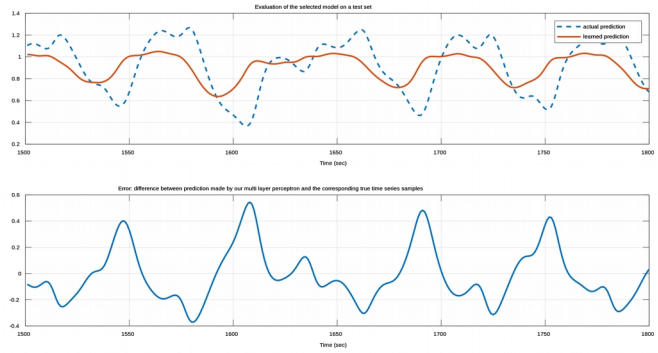
Training a neural network for different configurations with the use of early stopping and a regularisation technique. In this section we checked the performance of the neural network with different no of nodes in the hidden layer and different values of regularisation parameters. In the following table, results for different network configurations considered are presented. We have considered 8 and 4 nodes in the hidden layer and 0, 0.2 and 0.8 regularisation parameter values. We have compared the predicted output of the trained network to the target output.

For the data points and chosen network configuration, we have never observed the problem the over fitting. However, we have observed that with more the number of nodes in the hidden layer the error

between the predicted output and the target output is minimum. Regularisation parameter has a strong influence on the error between the predicted output and the target output. Regularisation parameter, with R = 0.5 we obtained the best result.

Table 1: Evaluation of the selected models on a test set - the conclusive estimate of the generalisation error on the unseen data set

Different network configurations	Comparison of target output and the predicted output of a two layer NN and error in generalisation
Case- 1: N=8 & R = 0	
Case – 2: N=8 & R = 0.2	
Case – 3: N=8 & R = 0.8	
Case – 4: N=4 and R = 0	

Case – 5: N=4 and R = 0.2	
Case – 6: N=4 and R = 0.8	

From the different network configurations as shown in the Table 1, the best network configuration we observed is when number of nodes in hidden layer, N = 8 and regularisation parameter, R = 0.5. For the same configuration, the generalization error is shown in the

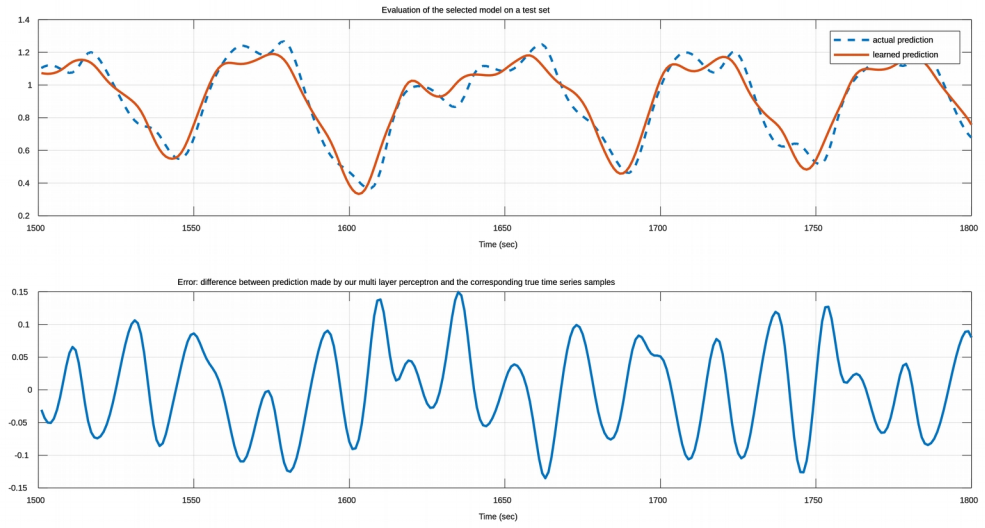


Figure 4: Generalisation error for number of nodes in the hidden layer, N = 8 and regularisation parameter, R = 0.5

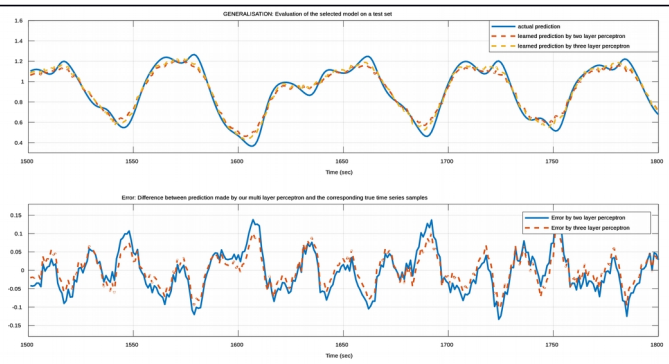
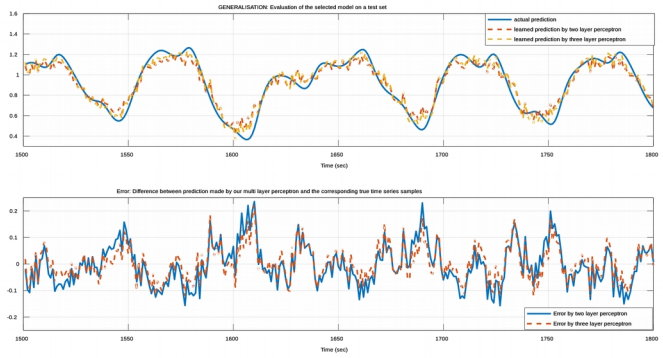
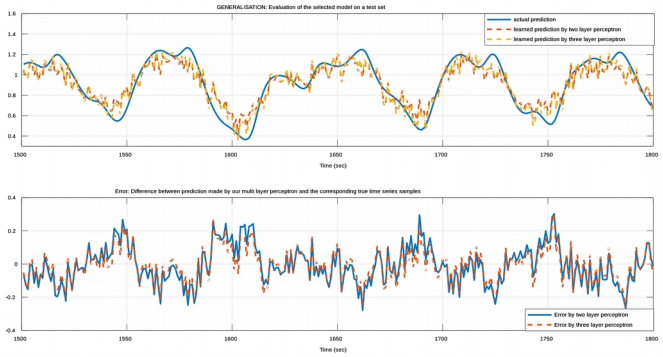
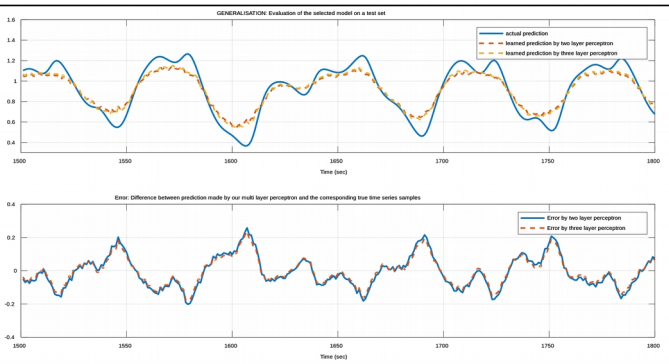
Part-2

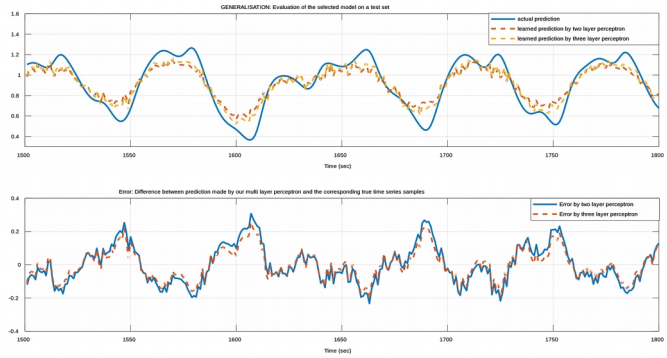
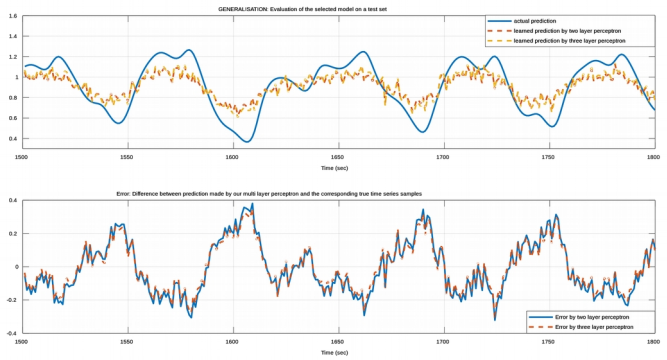
In part 2 of the assignment, we have chosen a three-layer network having two hidden layers and one output layer. We have restricted the number of nodes per layer to be 8. In this section, we have added zero mean white gaussian noise with different values of standard deviation to the already existing data.

Case-1: Influence of number of nodes in the second hidden layer to different noise level when number of nodes in the first hidden layer is 8. We have observed that with more number of nodes in the second hidden layer the generalization error is minimum at the cost of computation time. With

the injected noise in the data pattern, chance of over fitting is minimized. Although at higher values of standard deviation of the injected noise, the generalisation error is higher.

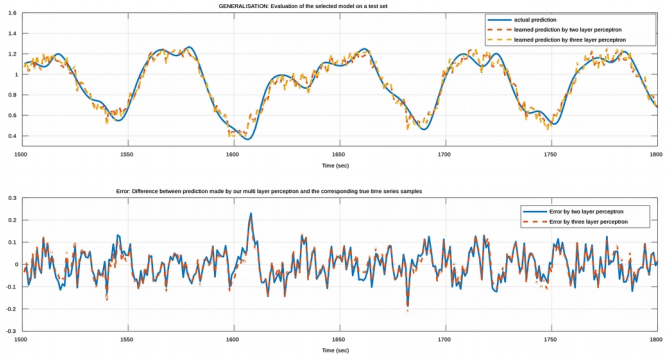
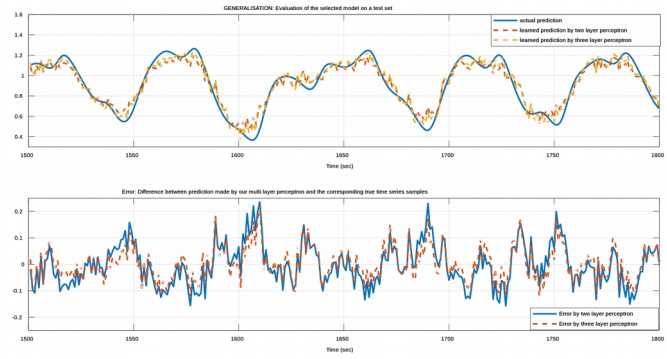
Table 2: Generalisation error with different nodes in the second hidden layer and different values of standard deviation of injected noise

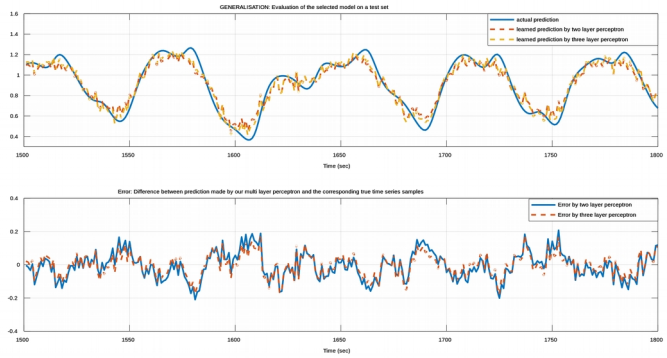
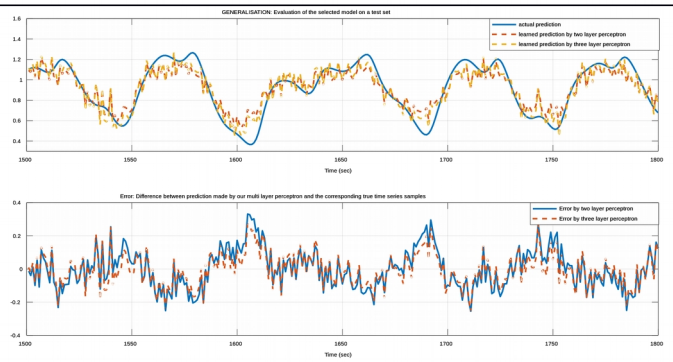
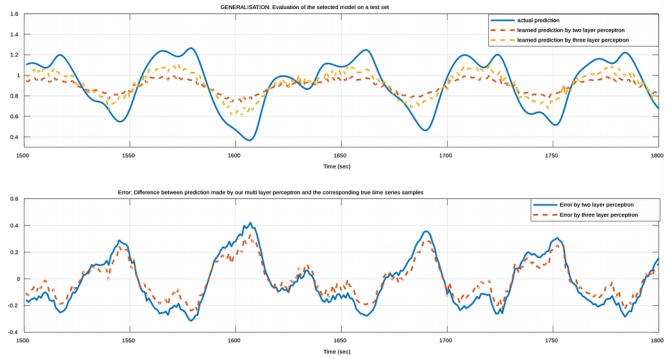
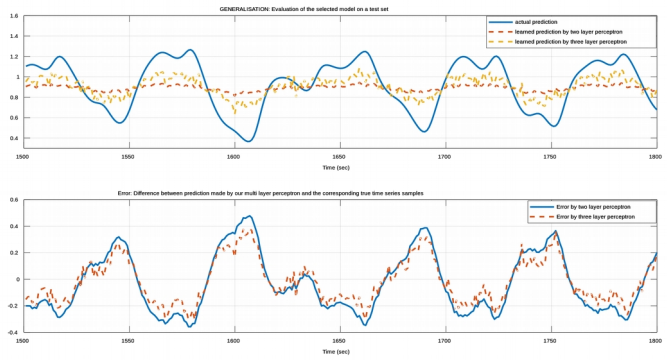
No of nodes in the hidden layers	Standard deviation of the additive Gaussian noise	Comparison of target output and the predicted output of a two layer NN and a three layer NN and error in generalisation
$N = 8$	$\sigma = 0.03$	
$N = 8$	$\sigma = 0.09$	
$N = 8$	$\sigma = 0.18$	
$N = 4$	$\sigma = 0.03$	

$N = 4$	$\sigma = 0.09$	
$N = 4$	$\sigma = 0.18$	

Case-2: The influence of regularisation with the amount of injected noise is illustrated in this section. Generalisation error is more with higher values of regularisation parameter as depicted in the following table:

Table 3: Influence of regularisation parameter with injected noise level

Regularisation parameter	Standard deviation of the additive Gaussian noise	Comparison of target output and the predicted output of a two layer NN and a three layer NN and error in generalisation
$R = 0.2$	$\sigma = 0.09$	
$R = 0.2$	$\sigma = 0.18$	

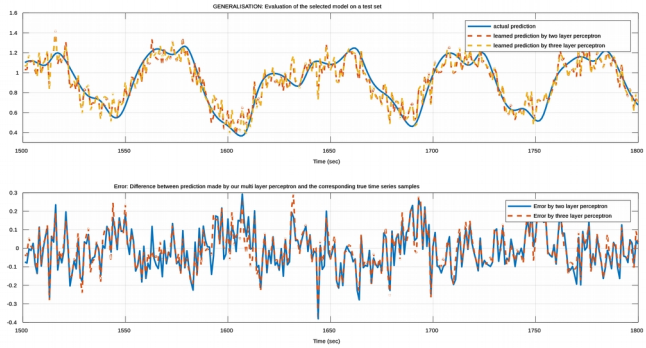
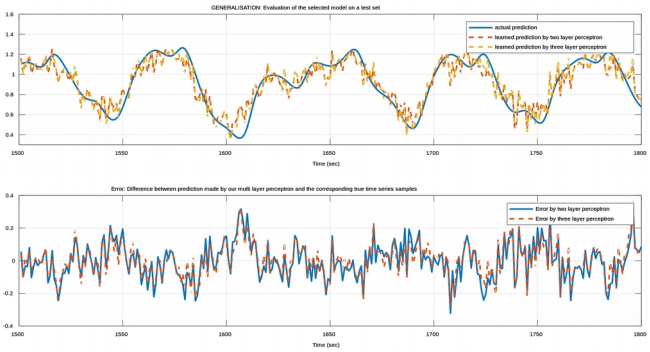
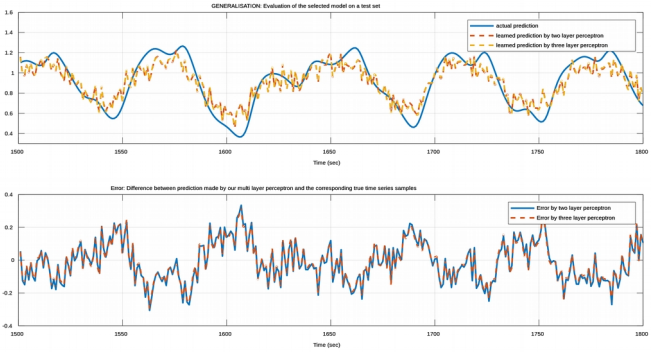
$R = 0.5$	$\sigma = 0.09$	
$R = 0.5$	$\sigma = 0.18$	
$R = 0.8$	$\sigma = 0.09$	
$R = 0.8$	$\sigma = 0.18$	

Case-3: Computational time

With same value of etha and same number of iteration, higher number of nodes in the second hidden layer result in higher computational time as observed in the following simulation results:

Table 4: Computational time for different network configurations

No of hidden nodes	Computational time (s)		Comparison of target output and the predicted output of a two layer NN and a three layer NN and error in generalisation
	3-layer perceptron	2-layer perceptron	

8	0.3930	0.3210	
4	0.4960	0.4	
2	0.7370	0,7170	

From this section we have the following conclusions:

1. With more number of nodes at the second hidden layer, the generalisation error is minimum.
2. With higher values of standard deviation of the injected noise, regularisation parameter has minimum influence.
3. With same values of etha and number of iteration, higher number of nodes in the second hidden layer result in higher computational time.

Conclussion

3.1.2

As expected the sequential have better performance per epoch than the batch learning and the corresponding perceptron learning has similar performance but for non linearly separable data you can see that the perceptron learning performs slightly worse because it over-fits to the points while the delta draws a decision boundary more in the middle.

3.13

Both perceptron learning algorithms does worse than the delta learning because of the non-convergent learning because it can never fit all the data points it continuously tries to adjust to the data points causing the decision boundary never to converge.

3.2.1

With higher complexity of the model (more nodes and lower theta value+iterations) the model over-fits to the data causing a miss in the testing. If we had used early stopping we would stop both algorithm around 100 iterations to get the best result because after 75 it starts to over fit, as expected batch converges slower than the sequential learning and should be stopped later to avoid over fitting.

3.2.2

Since the hidden layer has 3 nodes I imagined it could encode 2^3=8 bytes of data by each hidden node having a output if that data has that value. For example 1,-,1,-1… would have the value since the active state is on index 8 it should have the value of 1,1,1 from the hidden node. This seems to be the case if you look at the weights to first hidden layer node input with a hidden input at index 1,5,8 both have a positive to the hidden node 1 but you need to have one more input data having a positive hidden node 1 to get a representation in the hidden node in terms of a three letter binary representation. So it seems that the neural network is doing something with the weights to the hidden layer.

3.3.3

When you have a low amount of data points for training you need a high amount of nodes to represent the data

If you have a low n you need a high complexity = high amount of nodes to represent the data.