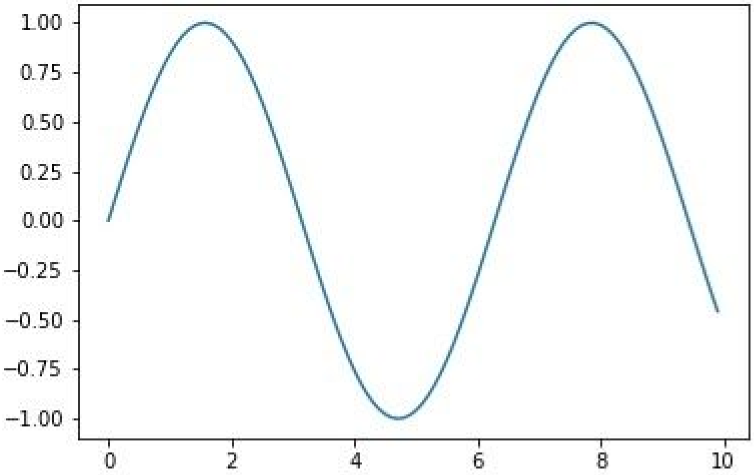
HOMEWORK 2

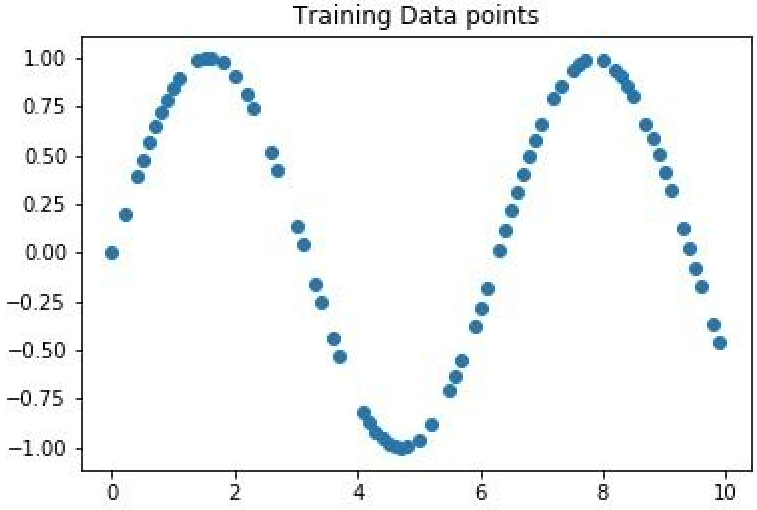
Charan Karthikeyan Parthasarathy Vasanthi, 115522428

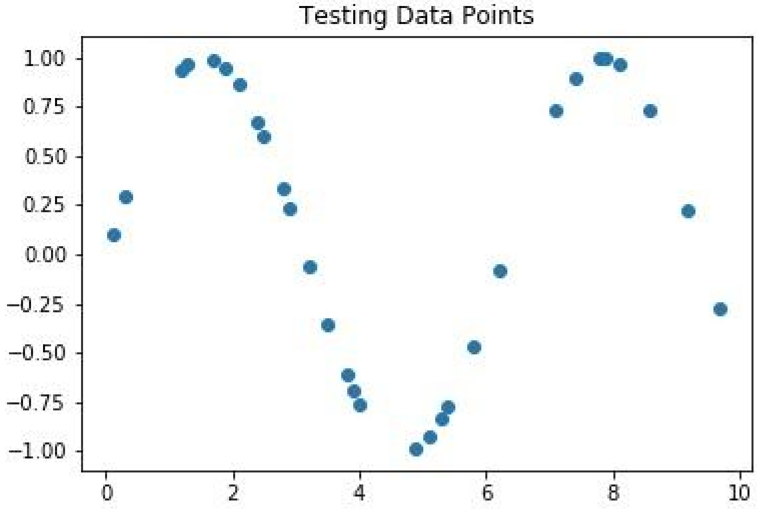
1.) Program a Discrete CMAC and train it on a 1-D function (ref: Albus 1975, Fig. 5) Explore effect of overlap area on generalization and time to convergence.
Use only 35 weights weights for your CMAC, and sample your function at 100 evenly spaced points. Use 70 for training and 30 for testing. Report the accuracy of your CMAC network using only the 30 test points.

Ans: The 1-D discrete CMAC function is trained using the sinusoidal function, where the value of x are values from0 to 100 with 0.1 point increase from the previous value and the y value is the sin of x values. The graph of the function is shown below.

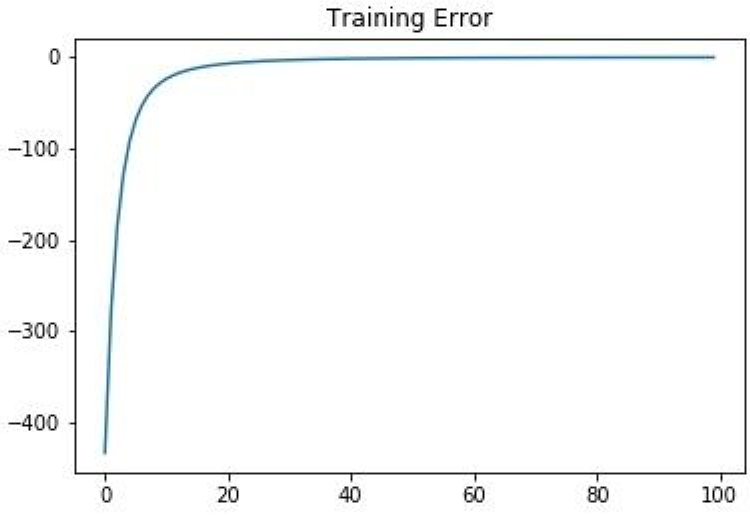


The train and test data are split into 70 and 30 from the 100 uniformly spaced points. The separated set of points are shown in the graph below.

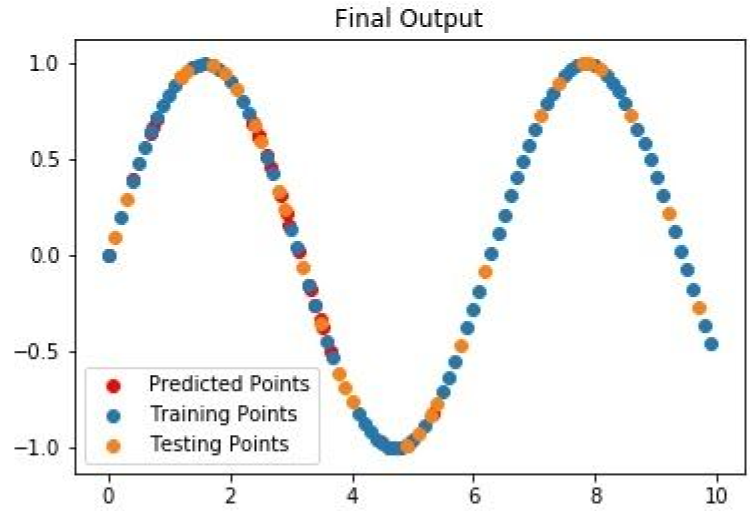




Training the Discrete CMAC. The function to sort the files and the function to split the training points from the training set is written at the beginning of the program. The program is done with 100 data points and 35 weights. The program then maps the associative indexes of their corresponding weights based on the generalization factor which in our case has been set to 3. The weights are updated at the end of each epoch and this is made to run for 100 epochs. The convergence graph of the training data is shown below.

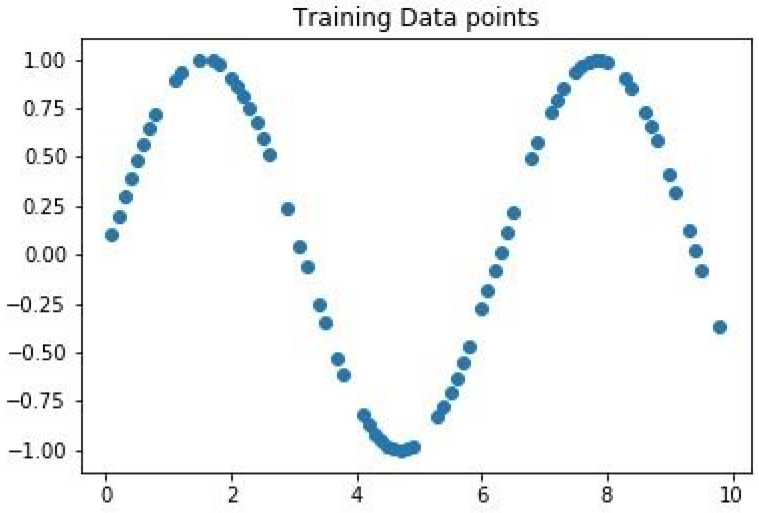


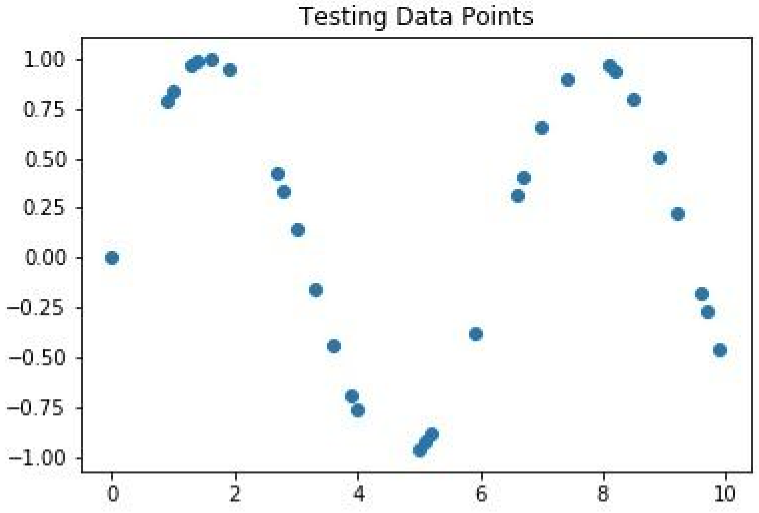
The testing samples are then applied to the updated weights and the final output with the training points, the testing points and the predicted points are shown below

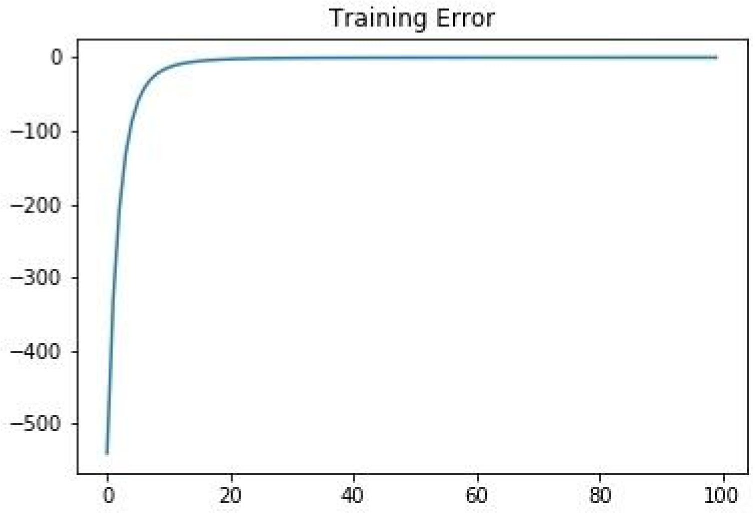


2.) Program a Continuous CMAC by allowing partial cell overlap, and modifying the weight update rule accordingly. Use only 35 weights weights for your CMAC, and sample your function at 100 evenly spaced points. Use 70 for training and 30 for testing. Report the accuracy of your CMAC network using only the 30 test points. Compare the output of the Discrete CMAC with that of the Continuous CMAC.

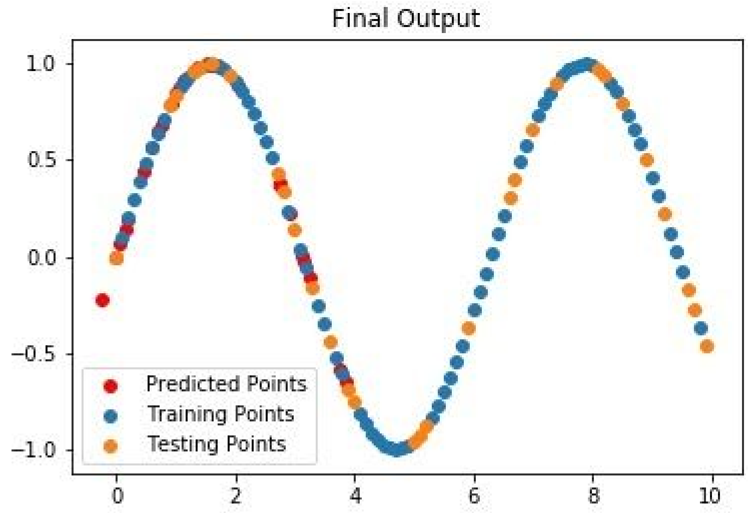
Ans: The training and testing points are generated in the same way as done in the Discrete CMAC. The functions are also similar to that of the Discrete CMAC with small changes in the method of weights updation. The part of the generalization factor is changes where the weights are assigned to differing parts of the generalization factor depending on the index number and the position of the values in the current cell or frame. The generalization factor is kept the same at 3 for the CMAC implementation.The weights are updated after each epoch for 100 epochs. The convergence of the curve is shown below.







The test data is then applied to the updated weights and the final output will all the points are shown below.



From the graphs from the continuous CMAC and discrete CMAC

3.) Discuss how you might use recurrent connections to train a CMAC to output a desired trajectory without using time as an input (e.g., state only)

Ans: RNN is a type of NN where the output form the previous steps are used as inputs to the current step and so on. The RNN contains memory about all the previous steps that it has taken and helps in the prediction later on. The recurrent connections in the RNN are connected to the CMAC at weights update point and the increases the generalization factor of the CMAC allowing it to look at all the previous steps and prediction to make further prediction to the given data. The RNN can be used in this way using only the states of the function or the data to predict the outputs, without using the the time function.

Github Link :
https://github.com/Charan-Karthikeyan/CMAC.git