ENPM 690: Robot Learning HW2

-Paras Savnani

Question 1:

Robot Learning Task: Autonomous UAVs are essentially Robotic systems that fly and navigate autonomously to achieve certain tasks like surveillance, recreational photography, material transport etc. Now, let's say we want to detect control surface (elevator, aileron, rudder) faults in a fixed winged UAV, i.e., during flight control surfaces can get stuck and can lead to the UAV crash scenario, potentially threatening ground population if the UAV is big enough. So, to avoid this we can try to detect faults as soon as it occurs and try to mitigate it to avoid crashing in below areas.

A model-based approach would be to estimate the states of the UAV using an Extended Kalman Filter and try to see if there is a spike above a defined threshold in the output signal to ascertain it as a fault.

On the contrary, a Robot learning (Machine Learning) based approach would be to collect the autonomous flight data with different types of faulty conditions and pose this problem as a supervised multi-class classification problem. Here, we would have labels for different fault categories, and we can train a classifier to classify the data. We can use Support Vector Machines (or any other ML algorithm) for this approach. For **feature engineering we can choose pilot command as input and the most prominent UAV states as output and concatenate them to build a high dimensional data point**, which we can classify. This method will be better and more generalizable than the Model-based approach where one might need to model the dynamics of the system and the environment very accurately.

Question 2:

We use C++ 14 to program a Discrete CMAC and we train it on a 1d function i.e., **x*sin(x)** and explore the effect of overlap area on generalization and time to convergence.

1) We take input space of 100 points between 0 to 2pi and we take the generalization factor as 2 and num of weights as 35.
2) Then, we randomly sample these points and divide them into 70:30 ratio for training and testing the algorithm.
3) Now we need to map the input 70 points to association space and multiply these elements with the weights and sum them to get the output:

$$U_{CMAC}(k) = \sum_{i=1}^{n} w_i(k)x_i(k)$$

4) To find the size of association vector that are mapped to the Association Matrix we use below formula:

   Association Vector Size = Num of weights – Generalization Factor + 1

5) We will be hashing contiguous regions (sliding window) to avoid storing random weight indices, therefore we will design the hash function proportionately and will store the start index of the weight vector in the Association Map. This will help us ascertain the active weights for each association element.
6) During training for each input, we will perform following steps:
   a. Get the active weights for that input.
   b. Sum the weights values (similar to multiply that input by 1 for the active weights and 0 for inactive weights and summing all values).
   c. Then we find the difference between the predicted value and actual value.
   d. Further, we use this difference to update the active weights using below formula:

   $$w_{t+1} = w_t + (e * \alpha)/gf$$

   where,

   w_{t+1} is weight at iteration t+1

   w_t is weight at iteration

   e is error

   α is learning rate

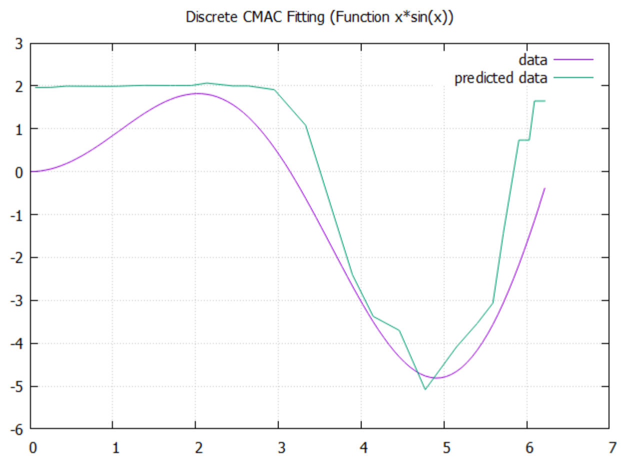
   gf = generalization factor

   e. This process is repeated until a predefined convergence threshold.

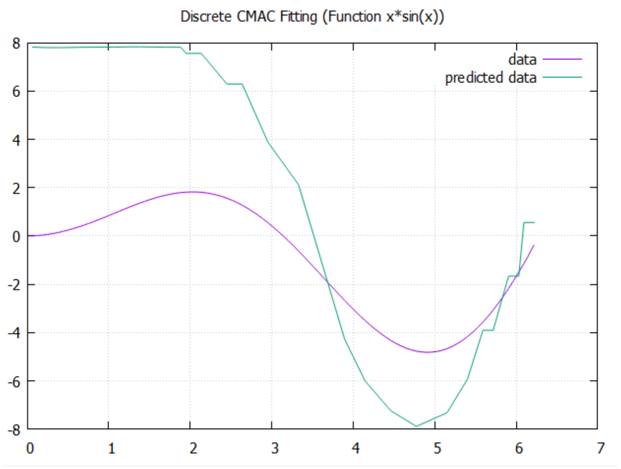
7) We use the trained network to predict the values of the test data (30 points) and find their accuracy. Finally, we plot them to visualize the results.

Following are the results for Discrete CMAC:

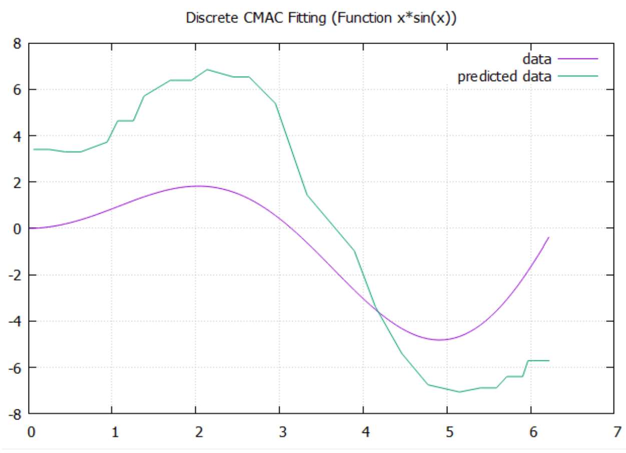
Generalization Factor	***Accuracy(%)***	***Convergence Time(ms)***
2	90.52	92.58
4	74.45	69.58
6	59.54	32.43
8	45.39	25.73
10	35.03	71.16
12	39.28	105.36
14	44.54	334.14
16	36.74	265.488
20	11.16	76.12
25	8.83	543.36
30	Network fails to converge	



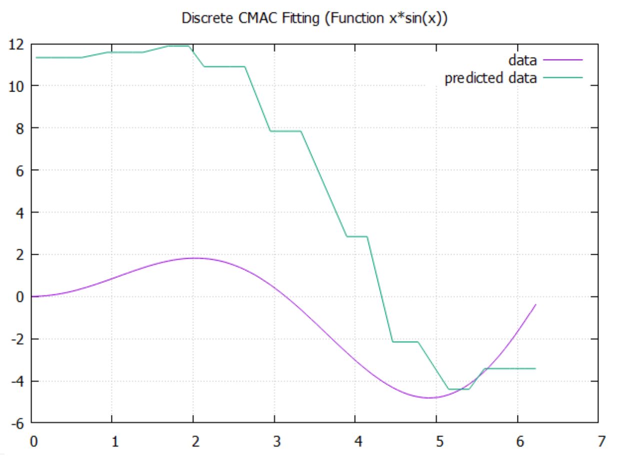
Gen_factor =2



Gen_factor = 8



Gen_factor = 14



Gen_factor = 25

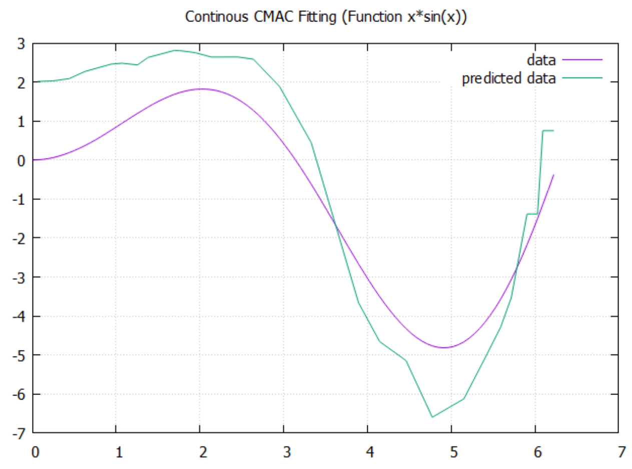
From the above results we can conclude that if we increase the generalization factor (i.e) overlap area for given number of weights, the accuracy decreases and the convergence time increases. We can see the predicted data curve diverging from the original data curve as we increase the generalization factor. Therefore, using a low generalization factor is recommeded for this configuration.

Question 3:

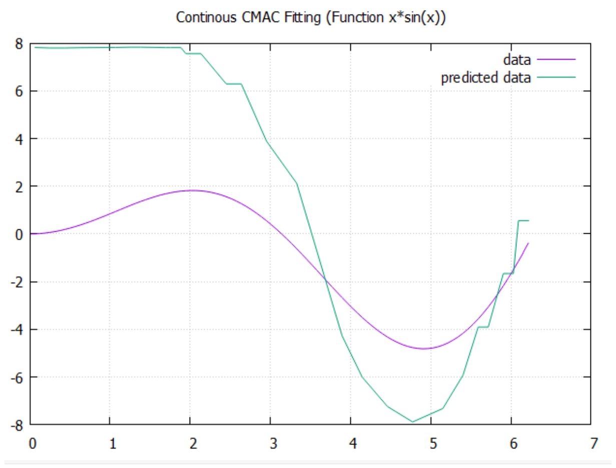
1) We follow similar process to Dicrete CMAC to train the Continous CMAC network. Similarly we train it on a 1d function i.e. **x*sin(x)**, to see the effect of overlap area on generalization.
2) As an add on, here we consider the proportion of weights based on the input values to calculate the sum of weights for the output.
3) We consider the partial overlap of the sliding window to calculate appropriate proporions to multiply with the active weight elements.
4) The training, prediction and convergence process are similar to the Discrete CMAC class.

Following are the results for Continuous CMAC:

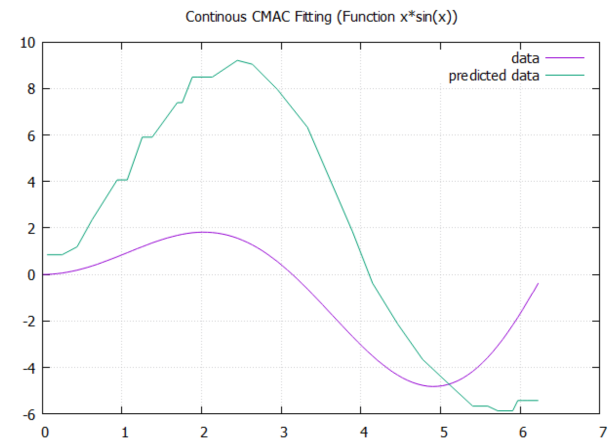
Generalization Factor	***Accuracy(%)***	***Convergence Time***
2	86.60	123.506
4	62.47	159.96
6	40.18	171.53
8	21.23	160.57
10	Network fails to converge	
12	19.35	201.97
14	6.11	137.57
16	Network fails to converge	
20	Network fails to converge	
25	Network fails to converge	



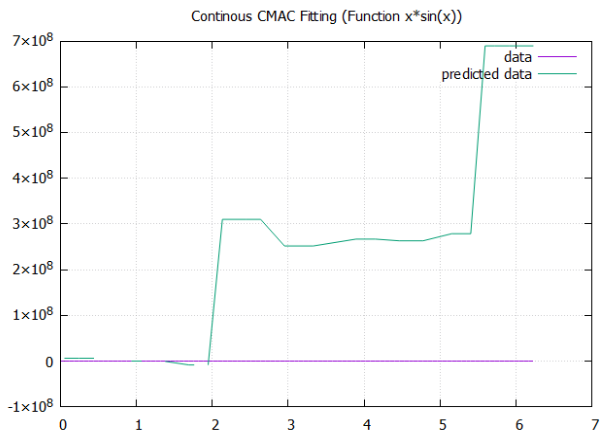
Gen_factor = 2



Gen_factor = 8



Gen_factor = 14



Gen_factor = 25

Similar to Discrete CMAC, from the above results we can conclude that if we increase the generalization factor (i.e) overlap area for given number of weights, the accuracy decreases and the convergence time remains almost same. We can see the predicted data curve diverging from the original data curve as we increase the generalization factor. Therefore, using a low generalization factor is recommeded for this configuration.

Question 4:

If we increase the number of weights, we are essentially allowing the network to learn more information about the input space. So, if we increase weights by some amount the network will learn better and will generalize better thus increasing the train and test accuracy.

But, if we increase the number of weights too much the network will nearly memorize the input mapping and will overfit on the input function. This won't allow it to generalize better where, it is strongly influenced by the input values. Thus, it will have a high train accuracy but low test accuracy.

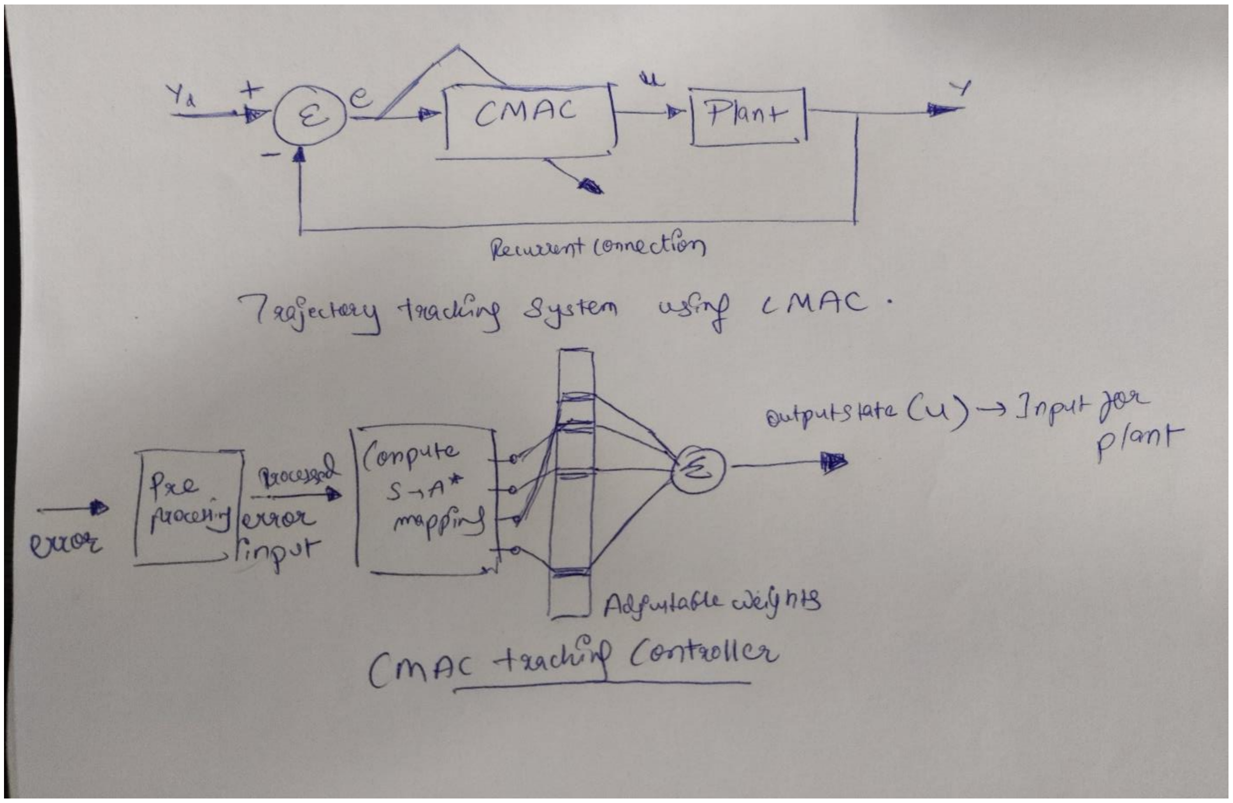
So, the number of weights should be optimized to get the highest possible accuracy.

Disadvantages of CMAC:

1) Large memory size requirement to implement the algorithm. The space complexity is O(n) proportional to the number of neurons.
2) Only a single layer is used to map input to output space. Thus, it cannot learn complex functions because it does not have a multilayer structure.
3) Hash function collisions happen with CMAC, and if we want to avoid collisions, we have to increase the hash table size.
4) The convergence of the CMAC is dependent upon hyperparameters like learning rate, generalization factor etc. and for some values it could lead to divergence. Thus, it is not always guaranteed to converge.

Question 5:

Reference Tracking/Trajectory Tracking is a problem that can be solved using traditional controllers like PID/LQR etc. We can also use a CMAC for this problem to output a desired trajectory based on input state, without taking time as input. Please refer the below flowchart for the system design:



We can train the CMAC based on the error generated from the output value and the next state value to generate the appropriate control signal to feed to the plant and thus track a certain desired trajectory. The CMAC controller contains 4 units:

1) Initially some preprocessing is required to scale error to usage scales and to feed in the CMAC network.
2) Then that scaled error is used to compute the Association mapping which will be multiplied with the active weights.
3) Next these weights are summed to get the desired control signal (output). It is fed in the plant and plant outputs the state of the system according to its internal dynamics.
4) This output is fed back to the Pre-processing unit via a recurrent connection (feedback) and again used to calculate the error.
5) Meanwhile the CMAC network learns using the backpropagation by calculating the difference between the output control value and the desired value and it updates the weights accordingly.