ENPM690 HW2

Sahruday Patti

February 2022

1. **Formulate a robot learning task using machine learning terminology. Describe what are the inputs and outputs, and how and where the supervision takes part in.**

   An autonomous car moving along the road using output commands velocity and steering angle based on the input sensory information such as LIDAR can be formulated as a robot learning task using Machine learning.

   The control outputs to this Machine learning model is $u = [v, \omega]$,a quasi static system where $\omega = \dot{\alpha}$ is the rotational speed of the robot. These can be transformed to the command actions (v, ϕ), where v is the transnational speed and ϕ is the steering angle obtained by $\phi = \tan^{-1}(\frac{\omega b}{v})$.where b is the length of the car.The inputs maybe the LIDAR data around the car i.e 360 degree/270 degree etc. This LIDAR data contains information about the environment which may be the obstacle distance, the path that is in front that is to be followed etc.

   This robot learning model can be trained using data produced by a Path planning algorithm based on sensory information and corresponding command outputs. After completion of training the machine learning model can be used to take direct sensory information and produce corresponding output commands. This type of machine learning is called Supervised machine learning.

   Here, the supervision takes place at giving a corresponding output command based on sensory information at every moment of time the input is received.

   We can use Reinforcement learning for this robot learning task as well. we give the agent the sensory information and we give corresponding actions velocity and steering angle. For each destination reached we give a positive reward and for every collision we give a negative reward. The agent will by trial and error figure out an optimal policy to learn and drive around without collisions to reach destinations and there by increases the reward.

   In reinforcement learning, the supervision is done by the Agent. For every incoming data from the environment, the agent following an optimal policy takes the best action which gives the maximum reward.

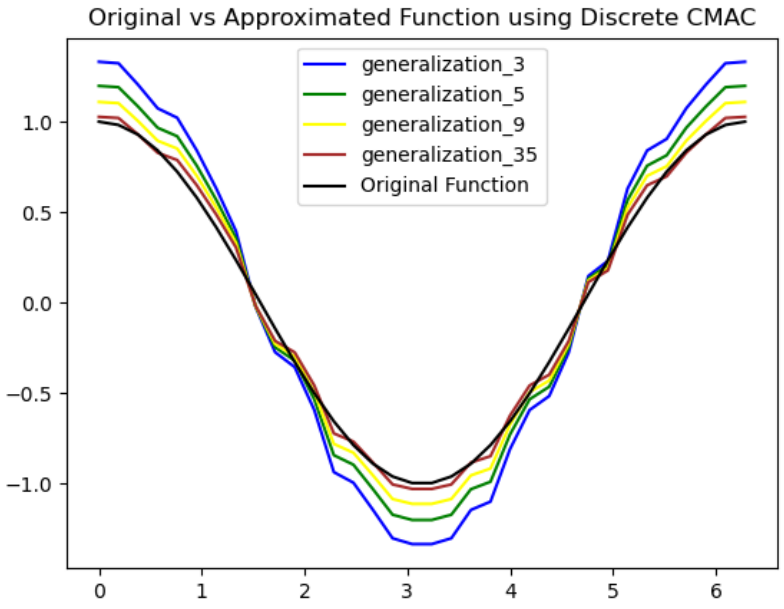
2. **Program a Discrete CMAC and train it on a 1-D function. Explore effect of overlap area on generalization and time to convergence. Use only 35 weights for your CMAC and sample your function at 100 evenly spaced points. Use 70 for training and 30 for testing. report the accuracy of your CMAC network using only 30 test points.**

   The 1-D function need to be approximation is:

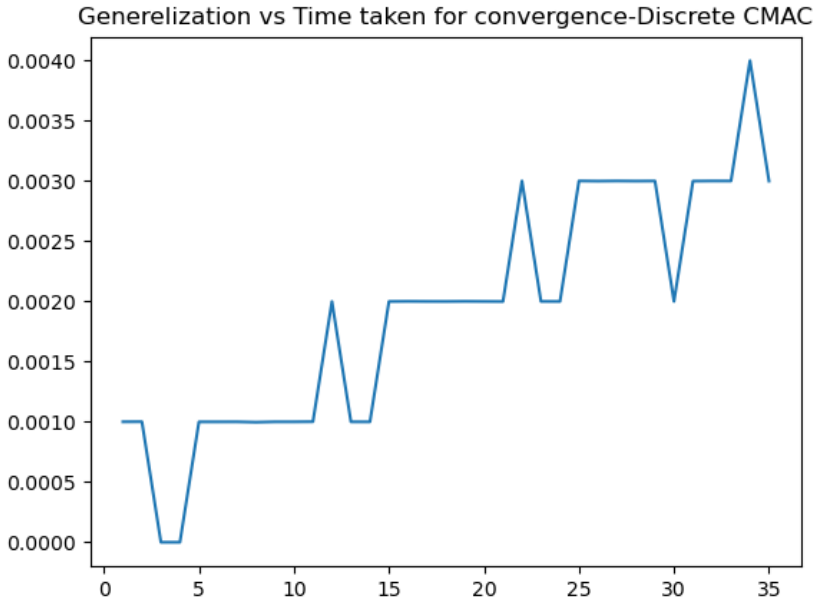
$$y = \cos(x) \tag{1}$$

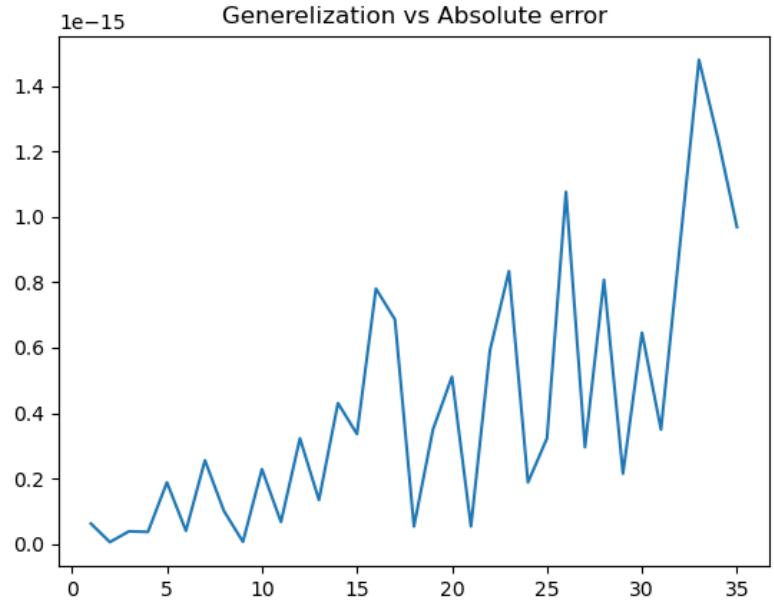
   Algorithm:
   1. 100 training points are generated between 0 to 2pi out of which 70 are used for training and 30 for testing.
   2. Generated a hash-map such that for each input there are K(generalizing value) keys and for each key there is a Value associated.
   3. While iterating through the inputs points, each input hash-map keys are updated with a value correction which is error/generalization.
   4. After training the CMAC, Predicted values for the corresponding test points are taken and we calculate the testing accuracy.

   The graph below shows the functions approximated by Discrete CMAC with different generalization values:

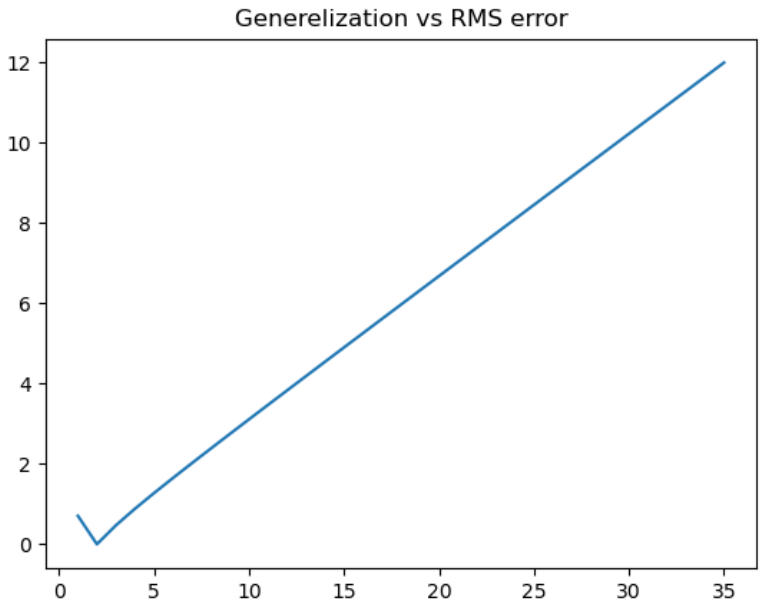


The generalization Vs time to convergence graph shows that, there is an increase in time for convergence with increase in generalization value.





The RMS error gives a clear picture:



Average Test error percentage for generalization value 3 is 22.359158
Average Test Accuracy percentage 77.640842
accuracy 0.0
Average Test error percentage for generalization value 5 is 13.857442
Average Test Accuracy percentage 86.142558
accuracy 0.0
Average Test error percentage for generalization value 9 is 8.377900
Average Test Accuracy percentage 91.622100
accuracy 0.0
Average Test error percentage for generalization value 35 is 4.330767
Average Test Accuracy percentage 95.669233

Although, the error percentage is less, the accuracy is 0. accuracy means the ratio of true predictions to total predictions.

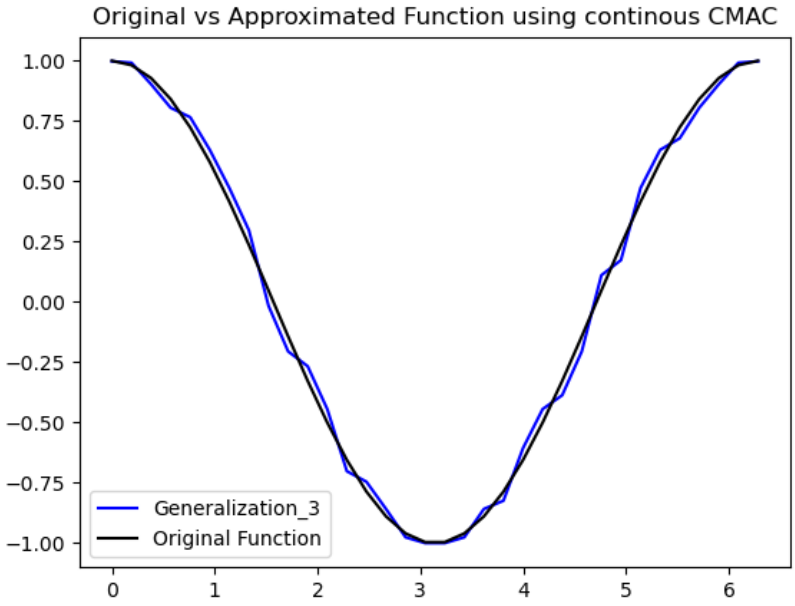
The CMAC algorithm forte is to generalize over a small neighbour hood of space and to be able to dichotomize for inputs that require different outputs. If we try to generalize for a 1D function which is constantly changing its value, the CMAC algorithm will not be able to generalize well.if we are generalizing over a neighbour hood then we will not be able to give accurate results for a constantly changing function such as cos. This can be seen from the accuracy given by the model.

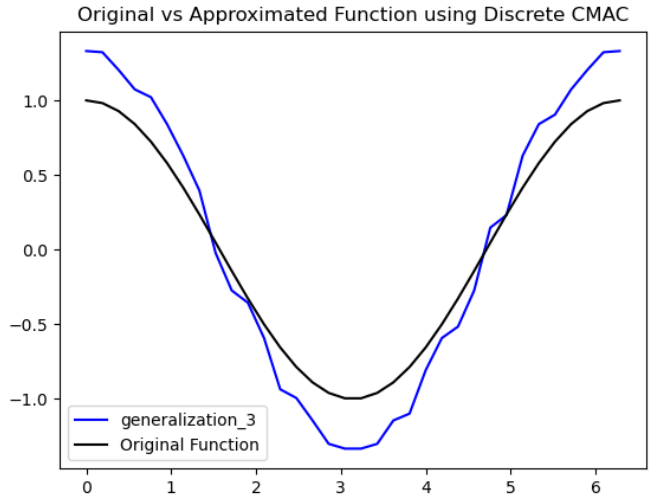
The CMAC doesn't give an accurate output i.e the absolute difference between the predicted output and original output differs always, as the cmac doesn't approximate the function accurately, so there is work to be done on CMAC to improve its accuracy for practical applications.

As we increase the generalization, it becomes easier and easier for the CMAC to approximate the exact function, because CMAC generalizes in a small neighbourhood space around each point so it can approximate the exact continuously changing curve. But there are infinite values between each neighbourhood so no matter how many weights we increase approximating the true function accurately is very hard but it comes closer and closer to the original function which is a continuous function and there will always be points in a neighbourhood where they require similar outputs.

3. **Program a Continuous CMAC by allowing a partial overlap, and modifying the weight update rule accordingly. Use only 35 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 30 test points. Compare the output of the Discrete CMAC with that of the continuous CMAC.**

   The algorithm is similar to the discrete CMAC, the only change is the partial cell overlap.when i update the weights in the continuous CMAC, I allowed a partial cell overlap which is 50 percent of weight update for each of top and bottom cell.



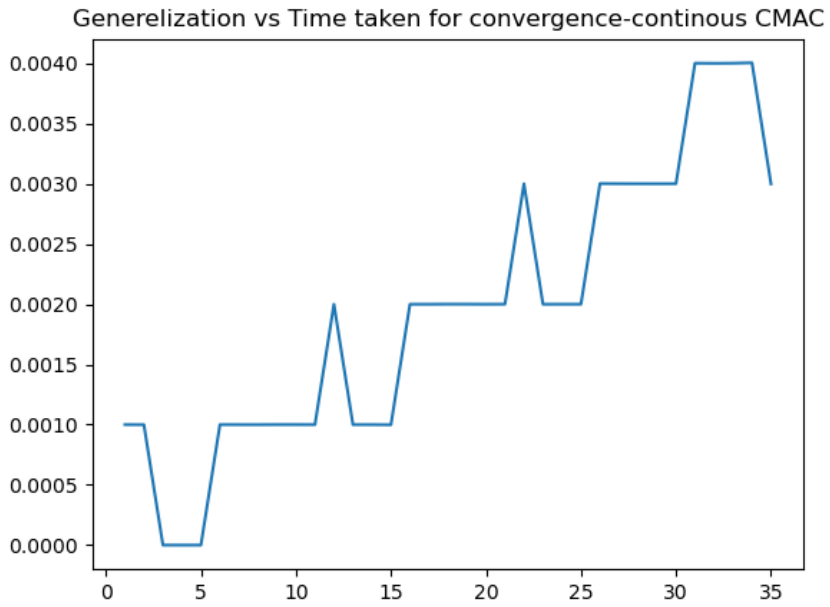


By interpreting the graph, we can surely say that the continuous CMAC approximates the Original function better as we allow the weights in the

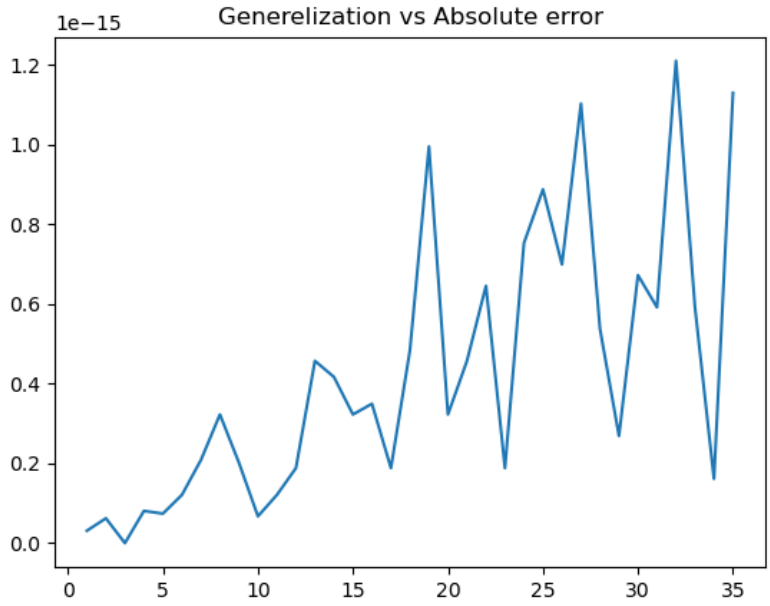
neighbourhood to be changed with the weight updates. This partial overlap will then be kind of a signal to the next association cell, that the next value is different in the neighbourhood and will require a different output.

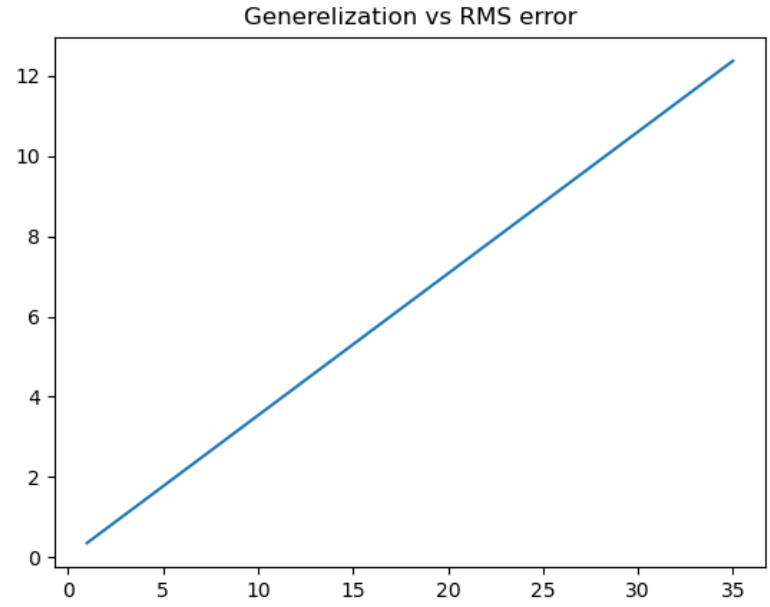
The one point to note is, how much ever the continuous CMAC approximates the function closely, it never is the true line i.e at any point of time the approximated function value is not equal to the true function value but it is very close, they don't differ by much because as we can see the overall error is less.

The Continuous function gives the same approximated function even if we increase the generalization value because the model has already dichotomized to a large extent already and it cant generalize any more due to lack of constrain in the weights.



Similar to the discrete cmac, The time for convergence increases as we increase the generalization value.





Average Test error percentage for generalization value 3 is 3.916968 Average Test Accuracy percentage 96.083032

The test data set accuracy for the Continuous CMAC is 0.

4. **Discuss how the results will change if the number of weights used in the model increases and what are the main disadvantages of CMAC.**

   As we increase the number of weights, the cmac ability to dichotomize increases, that is the neighborhood area around which it generalizes decreases. we can also say that, As we increase number of weights, the number of input combinations for which there is a mapping increases, so the neighbourhood area decreases.

   In simple terms we could say that as we increase the weights we are hard coding the function. So, if sufficiently large number of weights are provided such that we generalize to a lower and lower value say 0.00...1 then we could approximate the function 99.99...9 percent accurately.

   If we are approximating a function which changes in a large interval of space say a periodic square wave of high frequency and will also not have steep decrease/increase but rather a smooth curve. This kind of functions requires generalizing over a large neighbourhood of space but also be able to dichotomize where there is a smooth change in the value. The CMAC will be able to replicate this function more accurately. As CMAC has this ability to dichotomize over a small neighbourhood of space and also be able to dichotomize well.

   The disadvantage of CMAC algorithm is it dependent upon the Physical memory available to allocate the association cells. As we increase the memory size more and more weights can be allocated in turn increasing the dichotomizing ability of CMAC. As we increase the memory size, the computation time also increases.Also, CMAC can approximate the function closely but its accuracy is definitely a concern. say,we are simulating a control function for an autonomous car, we need to be more precise in our output motor commands in order to be practically useful.

5. **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)**

   Recurrent connections are different from the standard feed forward networks in a way that while training,it also learns the prior inputs while generating the output. The advantage of Recurrent networks is it remembers the intermediary information by generalizing over a neighbourhood due to iterative data training process. It possess Long short Term Memory or LSTM.

   To train the CMAC using Recurrent connections, a variable called previous error can be used to store the previous training error. A variable

called Learning rate can be used to tune how much it depends on the previous error. Now, when we update the weights we can update the weight bases on current error plus the learning rate times the previous error thus making it dependant on the previous error. So, as the number of iterations increase, the previous error decreases so it converges to a minimum error value iteratively. By following this idea, we can approximate the function using only state only and it will be independent of time. The learning rate can be varied depending on how much we want the network to be dependent on previous error.

In the cmac algorithm after the first epoch, the connections attain a local minima and it is not improved over time. But in Recurrent cmac, there is constant decrease in training error as the number of epochs increase and it approximates a 1D function which is close to the original function.
The below figure shows the approximated function using the recurrent cmac.The original function is a cos in the interval of 0 to pi/2

The function approximated using discrete cmac with Recurrent conections: