ENPM690 ROBOT LEARNING HOMEWORK 2 (CMAC)

SUBMITTED BY:

MITHUN BHARADWAJ

114931115

Question 1: 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 the 30 test points.

The 1-D function I have taken is **sin(x).** The 100 evenly space function inputs are varied between (0,10). The function plot is shown below:

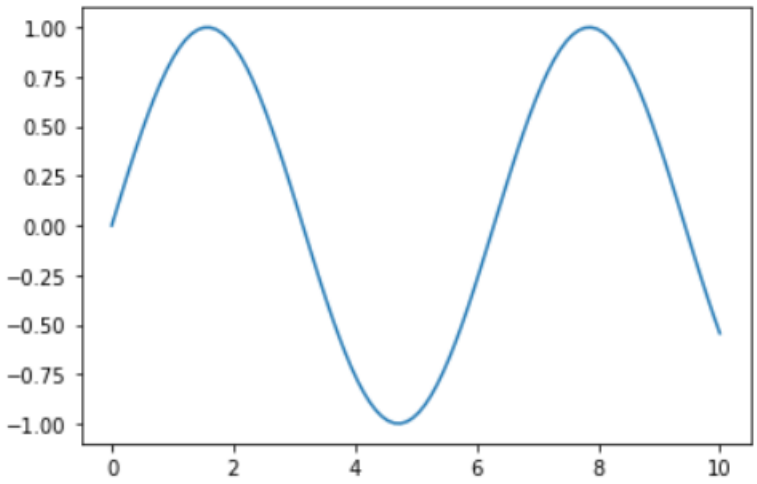


Fig 1. sin(x) between [0, 10]

There are 2 conditions checked for convergence:

- The magnitude of the updates is not varying too much with every iteration. The idea is that the updates will be large initially because the error is large, but as the algorithm begins to converge, the updates become very small and insufficient to make an impact. We've to keep in mind that this does not mean it has converged to the correct solution and hence the second condition.

- When the error between the actual value and predicted value is below some threshold (in this case 10^{-2}), i.e when the predicted value is close to the actual function, we can say that the algorithm has converged to the correct solution.

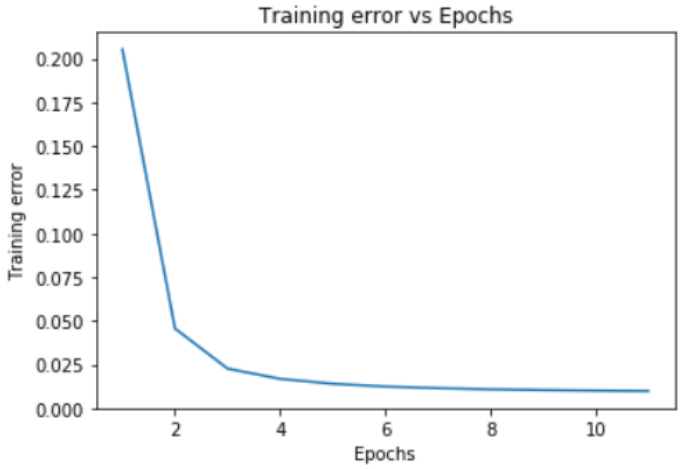
Effect of overlap area:

- As the overlap is increased, it increases the time for convergence and error.
- The accuracy for the same number of iterations decreases and the generalization of the algorithm decreases.
- As the overlap increases, a lot of common weights become responsible for multiple intervals and they've to produce a range of values. Since we just add the weights to get the value in an interval in the CMAC algorithm, the ability of the algorithm to generalize over a large interval decreases.
- CMAC is designed for generalization locally through discretization and calculated overlap in the continuous case.

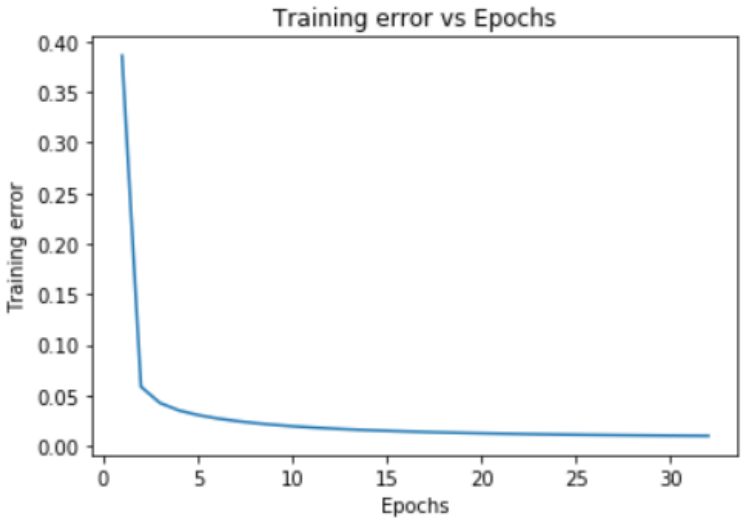
Approach:

- Generate the data for sin() function at 100 uniformly sampled points between 0 and 10. Split the data into training and test sets.
- Initialize a weight vector of length 35, with value 1.
- Discretized the input space, based on the number of overlapping weights we want and the number of intervals needed between 0 and 10.
- Created a binary (0s and 1s) association matrix such of size (number_of_intervals x number of weights).
- Each interval in the input sequence maps to a certain row in the association matrix. Each row in the matrix has a set of active weights depicted by 1.
- Based on the interval into which a particular input falls into, we call calculate the output by summing the active weights for that interval.
- Update the weight using the update rule specified below.
- Iterate till the algorithm converges.

Convergence:



2a. For 3 overlapping weights



2b. For 10 overlapping weights

Fig 2. Training error vs epochs for discrete CMAC

- The training error for the discrete case decreases rapidly initially and convergence slows down later.
- The convergence takes longer (condition of error being less than 10^{-2}) as the overlapping increases. (As can be seen in Fig 2a [10-15 iterations for 3 overlapping weights] and 2b [30-35 iterations for 10 overlapping weights])

- The learning rate was set to 0.25. Learning rate matters a lot in convergence. If it's too high, the updates will be too large, since the update equation is

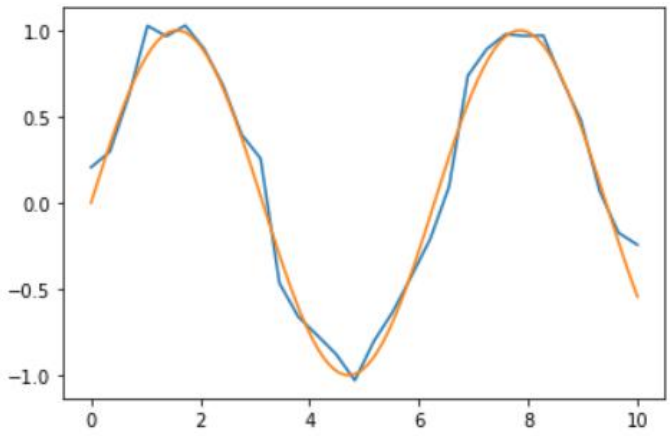
$$\boldsymbol{w(k+1) = w(k) + LR * error/(number\ of\ shared\ weights)}$$

Where,

- **LR – Learning Rate**
- **W(k+1) – weights at (k+1)th iteration, and**
- **W(k) – weights at kth iteration**

If the updates are too large, it's hard for the algorithm to fine tune the weights to converge. If the learning rate is too small, the updates are very small and it takes much longer to converge.

Result:



True curve: Orange

Predicted curve: Blue

Fig 3a. True vs predicted curve for 3 overlapping weights

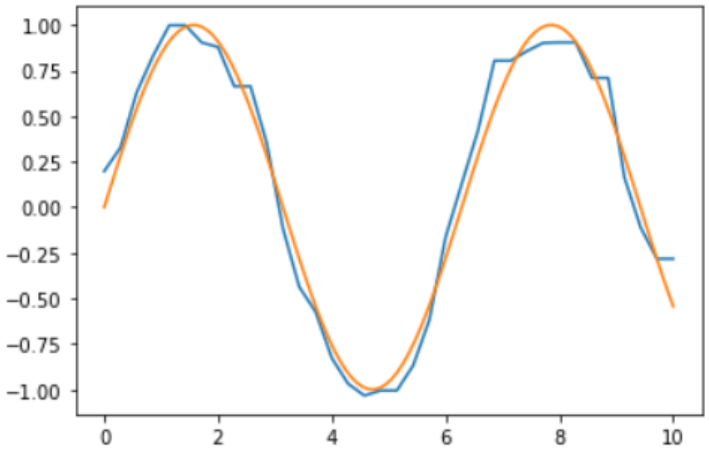


Fig 3b. True vs predicted curve for 10 overlapping weights

Error/ Accuracy:

In this case error is defined as:

$$Error = \frac{1}{N}||Y_{true} - Y_{predicted}||$$

Where **N is the number of test samples, Y is the output vector for all samples in the test set.**

- The training error was 0.0097598.
- The testing error was 0.0194277. As expected, the testing error is more than the training error. Since error and accuracy are equivalent metrics, I haven't detailed the accuracies here.

Question 2: Program a Continuous CMAC by allowing partial cell 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 the 30 test points. Compare the output of the Discrete CMAC with that of the Continuous CMAC.

- The 1D function is the same as the discrete case: **sin(x)**
- The convergence criteria are the same as the two conditions mentioned in the discrete case discrete case.
- Effect of overlap is similar here but lesser than the discrete case as we consider the amount of overlap (0 to 1) with the weights giving us a much higher capacity for representation of continuous curves. Even though it might affect the generalization ability because of the inherent nature of the CMAC algorithm, it is much less pronounced than the discrete case.

Approach:

- The only difference between the continuous and discrete CMAC is that the weight values which has partial overlap are also considered to contribute to the output.
- Just as before we find all the active weights for a particular interval. All weights which are fully considered are fully used to calculate the output. The weights which have a partial overlap will only be included after multiplying these weights with the amount of overlap (0 to 1).
- All the remaining steps are exactly same as the discrete case.
- The association matrix for the discrete CMAC was as shown in the figure:

$$x_1 = 0 \rightarrow \mathbf{a}_1 = 11110 \ldots 000000$$
$$x_2 = 1 \rightarrow \mathbf{a}_2 = 011110 \ldots 00000$$
$$x_3 = 2 \rightarrow \mathbf{a}_3 = 0011110 \ldots 0000$$
$$x_4 = 3 \rightarrow \mathbf{a}_4 = 00011110 \ldots 000$$
$$x_5 = 4 \rightarrow \mathbf{a}_5 = 000011110 \ldots 00$$

Fig 4: Association matrix

- For the continuous case, the same number of total overlapping weights was considered but including the partial overlap. This allows for slightly better generalization.

Convergence:

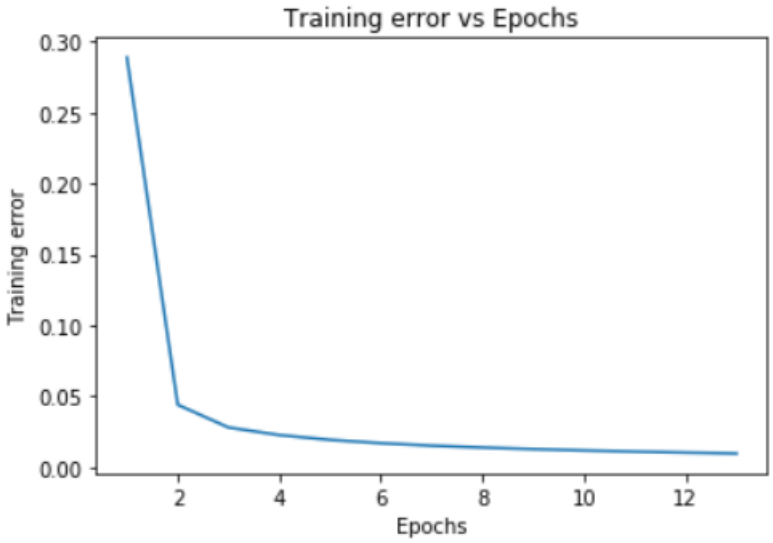


Fig 5a: Training error vs epoch for continuous CMAC for 5 shared weights

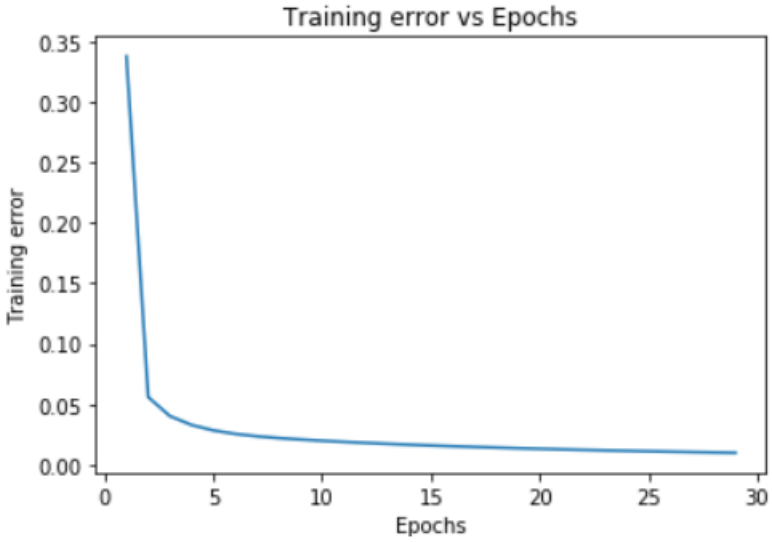


Fig 5b: Training error vs epoch for continuous CMAC for 10 shared weights

Result:

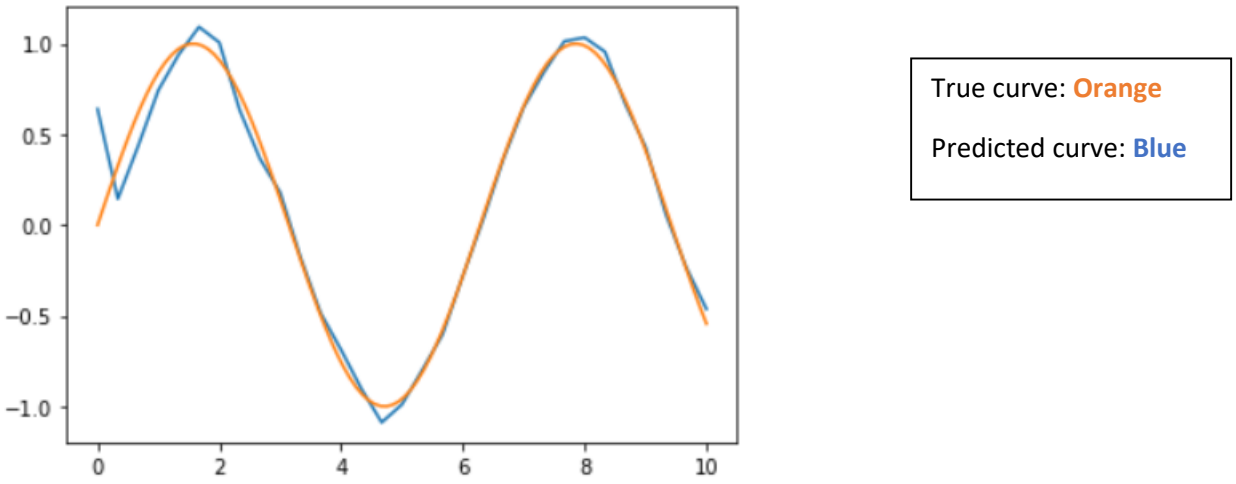


Fig 6a: True vs predicted curve for 5 overlapping weights

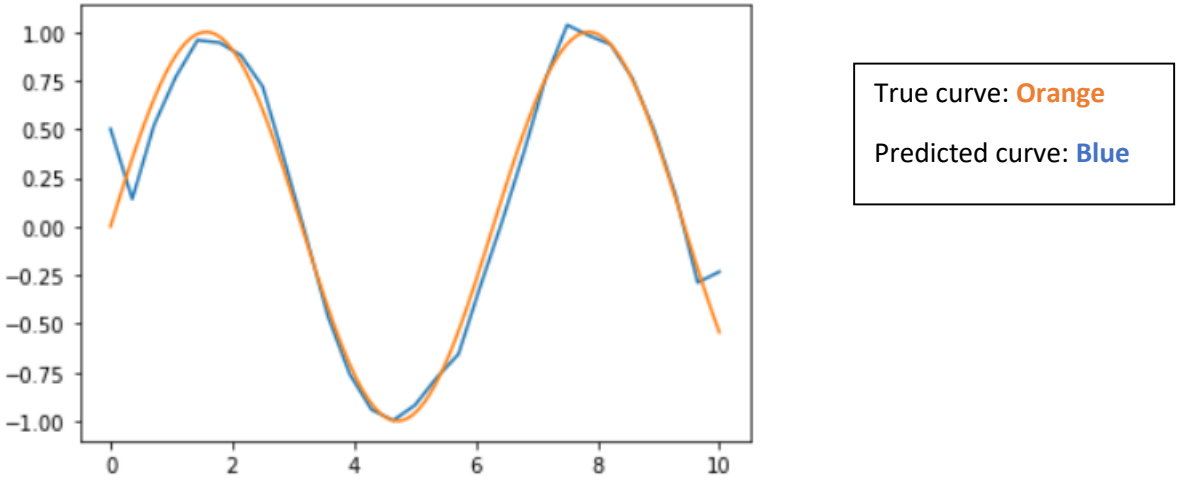


Fig 6b: True vs predicted curve for 10 overlapping weights

Observation/ Comparison between discrete and continuous CMAC:

- The convergence rate and accuracy is pretty similar for both discrete and continuous CMAC as can be seen from the training curves, i.e number of epochs required for the algorithm to converge is fairly similar.
- Continuous CMAC can predict the continuity of the curves better. From figures 3 and 6, it's clear that in the discrete case, the predictions are similar to chunks of step functions,

whereas in the continuous case, the predicted values form a more continuous curve. This is because of the partial overlap of the weights being considered in continuous CMAC, which is the major advantage.

- Since partial overlap is considered, it is necessary to be careful with the edge cases. Fig 6b gives an idea of the predictions near 0 and 10 (lower and higher limit of input range) if edges cases are not taken care of. This is because the index of the active weights can underflow or overflow because of partial overlap. It can be easily fixed by enforcing a few conditions. [Fig 6b is included to show the effect of partial overlap on edge cases].

Error/ Accuracy:

In this case error is defined as:

$$Error = \frac{1}{N}||Y_{true} - Y_{predicted}||$$

Where **N is the number of test samples, Y is the output vector for all samples in the test set.**

- The training error was 0.0095927.
- The testing error was 0.020567. As expected, the testing error is more than the training error. Since error and accuracy are equivalent metrics, I haven't detailed the accuracies here.

Question 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).

- Recurrent connections are based on the idea that it receives two inputs, the current and the previous. There is information in the sequence of the past output which is analogous of having memory in humans.
- This sequential information is preserved in the networks hidden states of the network passing through the recurrent units for each sequence.

- Some of the preliminary recurrent units are GRUs (Gated Recurrent Units) and LSTMs (Long Short Term Memory – Proposed to increase the memory/ reduce the vanishing and exploding gradient problem).
- In this method, the information is stored outside the normal flow of the recurrent network in gated cells. These analog gates receive, pass or block information based own their own set of weights. These are also adjusted via the recurrent networks learning process.
- To implement CMAC using recurrent connections, we would need to correct the output by error function and an additional function which depends on the previous output.
- This will make the function depend on itself or its previous outputs and create a path for the states to propagate through the sequence.

GITHUB:

https://github.com/mithun-bharadwaj/CMAC_Cerebellar_Model_Arithmetic_Computer

REFERENCE:

1. https://onlinelibrary.wiley.com/doi/full/10.1002/9780470050118.ecse057