ENPM808F HW2

Abhishek Kathpal
114852373

February 25, 2019

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.

   Solution:

   1D Discrete CMAC Algorithm is implemented by using the following steps:
   1. Divide the dataset into test and train datasets with ratio of 70:30.
   2. Initialize weight vector by assigning 35 random weights from 0 to 1.
   3. Run the loop for updating the weights. There are two stop conditions for this loop, one, if maxepochs are achieved then loop will stop and second, if all the points lie in error window of (-0.1,0.1).
   4. Now these updated weights are added and multiplied with the given input dataset.
   5. The updated weight vector is used as input for test function to find the error and accuracy.

   The CMAC architecture is shown in figure below:

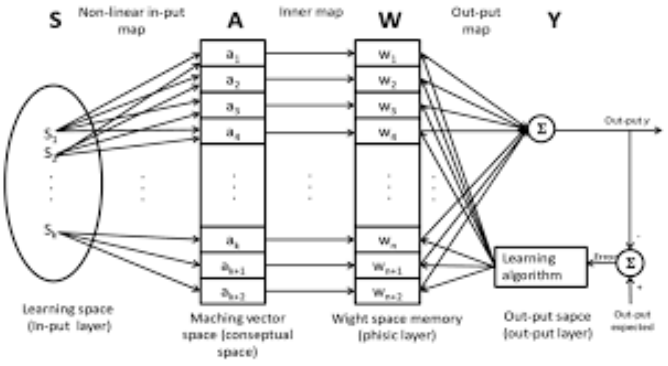

Figure 1: CMAC Architecture

Experiments: 1. I have varied the generalisation from 3 to 35 with step size of 8. to see how the output depends on generalisation fdactor. If generalisation factor is 5 then only 5 elements were associated from input space to weight vector. The output fits better on data as the generalisation factor is increased. You will get more tight fit with increase in generalisation factor. But there is

not much different with this small dataset after a particular generalisation factor i.e.19, although randomness of output decreases.

2. I have also plotted the accuracy and loss per epoch with varying generalisation factor.

Conclusion By increasing the Overlap area, there is less randomness in the output obtained and the rate of convergence is higher.

All the outputs from Discrete CMAC are shown below with g representing generalisation factor [3,11,19,27,35] in the outputs on next page.

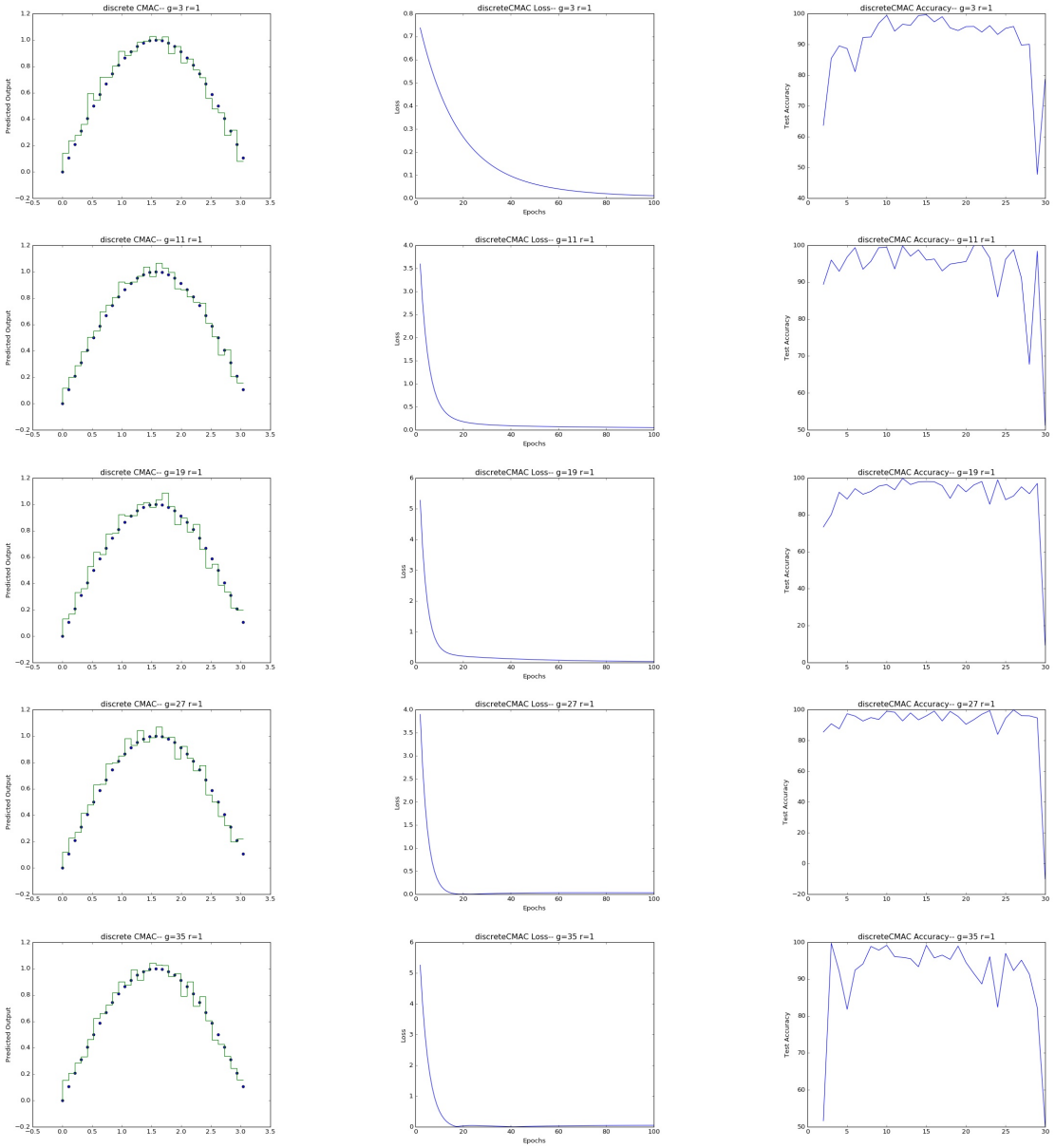

Figure 2: Discrete CMAC Outputs

To Run Code:
python cmac.py –Method discrete

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.

   The Continuous 1D CMAC is hown in figure below:

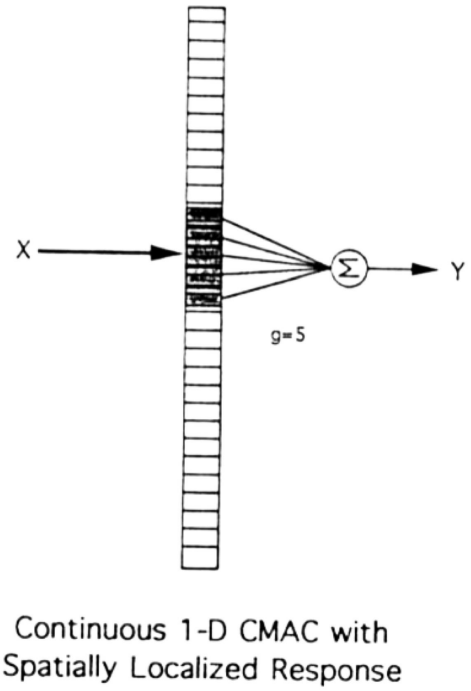

Figure 3: CMAC Architecture

   The code for the Continuous CMAC is very similar to dicrete CMAC. The main difference is that accociation cells are continuous. Continuous CMAC might use a particular ratio of first and last cell. If the first cell is being 75 percent used then last cell is used only 25 percent. Number of cells used is increased by 1.Similar experiments were done with Continuous CMAC as well.
   Ratio and Method can be passed as arguement to the cmac function.

   Conclusion By increasing the Overlap area, there is less randomness in the output obtained and the rate of convergence is higher. The COntinuous CMAC performs better than dicrete if generalisation factor is less. But for this dataset, if we increase the g value closer to 35 then both are producing similar outputs.

The output for this Continuous CMAC is shown below:

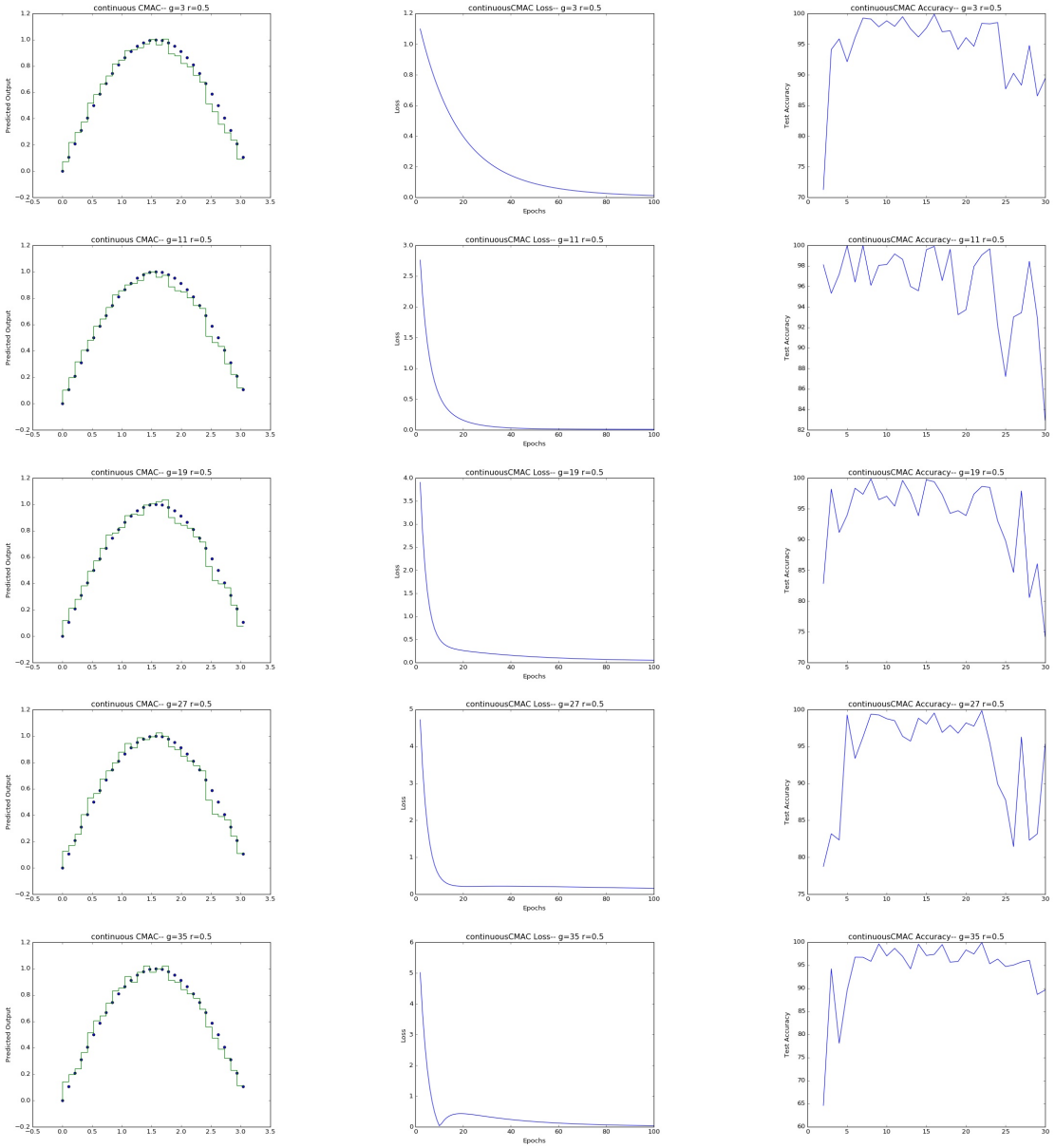

Figure 4: Continuous CMAC Outputs

To Run Code:
python cmac.py –Method continuous –Ratio (integer between (0,1))

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 networks using consists of nodes connected with feedback loops. Usually, these inputs at time t-1 will affect the output at time t. We can also use recurrent connections where we use feedback as states and this is usually helpful in trajectory following. For this implementation, LSTMs(Long short term memory units) can be used which store the information in gated cells and these gates recieve, forget and store information from the cells. This depends on weights which are being updated during recurrent neural network process. Another implementation would be using Recurrent Fuzzy CMAC, as you can see in the figure below that state feedback is given as input. This network can be divided into five layers Input Layer, Fuzzified Layer , Fuzzy Association Layer, Fuzzy Post-association Layer and Output layer. Association Layer consists of recurrent connections. the use of this network for trajectory control is explained in the paper attached in the folder. Learning Algorithm of RFCMAC is mentioned in Trajectory-Tracking pdf attached in Referencs folder.

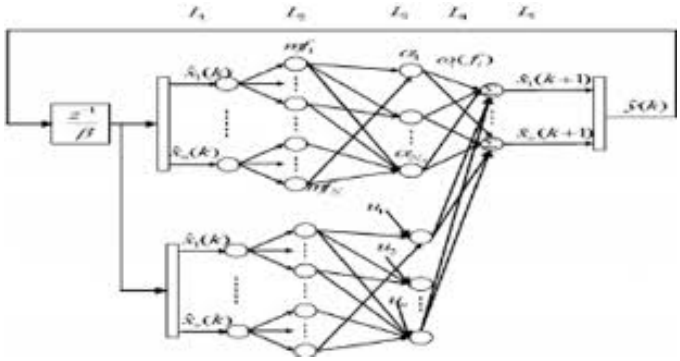

Figure 5: Recurrent Fuzzy CMAC

4. **Code:**
   Saving results in current directory
   Uncomment the plt.show() to visualize results

```
#!/usr/bin/evn python

"""
ENPM808F: Robot Learning
Spring 2019
CMAC Implementation - Discrete as well as Continuous

Author(s):
Abhishek Kathpal (akathpal@terpmail.umd.edu)
UID: 114852373
M.Eng. Robotics
University of Maryland, College Park
"""

import numpy as np
import math
import random
import time
import matplotlib.pyplot as plt
import argparse



def train_cmac(trainLabel,trainData,g=35,lr=0.01,maxEpochs=5000,r=1):
        # Selecting random 35 weights from 0 to 1
        weight = np.random.rand(35)
        pred=np.zeros((len(trainLabel),1))

        epochLoss = []
        numEpochs = []
        maxEpochs = 5000
        num = 0
        counter = 0
        while counter!=len(trainData) and num<maxEpochs:
                counter = 0
                num = num+1
                numEpochs.append(num)
                loss = []
                loss_abs = []
                for i in range(len(trainData)):
                        w = 0
                        start = int(math.floor((i)*35/70))
                        if r==1:
                                end = start+g # Discrete CMAC
```

```
                        else:
                                end = start+g+1 # Continuous CMAC
                        if end>34:
                                end=35

                        for j in range(start,end):
                                if j == start:
                                        w = w + weight[j]*r
                                elif j == end-1:
                                        w = w + weight[j]*(1-r)
                                else:
                                        w = w + weight[j]

                        pred[i] = w*trainData[i]
                        l = trainLabel[i] - pred[i]
                        loss.append(l)
                        for k in range(start,end):
                                if k== start:
                                        weight[k] = weight[k] + lr*loss[i]*r/g
                                elif k == end-1:
                                        weight[k] = weight[k] + lr*loss[i]*(1-r)/
                                else:
                                        weight[k] = weight[k] + lr*loss[i]/g

                epochLoss.append(abs(np.mean(loss)))
                # print("Epoch "+str(num))

                counter =0
                for i in range(len(trainData)):
                        if abs(loss[i])<0.1:
                                counter = counter +1
        return epochLoss,numEpochs,pred,weight


def test(testLabel,testData,weight,g,r=1):
        predTest = []
        acc = []
        for i in range(len(testLabel)):
                w = 0
                start = int(math.floor(i*35/30))
                if r==1:
                        end = start+g
                else:
                        end = start+g+1
                if end>34:
                        end = 35
                for j in range(start,end):
                        if j == start:
```

```
                        w = w + weight[j]*r
                    elif j == end-1:
                        w = w + weight[j]*(1-r)
                    else:
                        w = w + weight[j]

                predTest.append(w*testData[i])
                l = abs((testLabel[i] - predTest[i])*100/testLabel[i])
                acc.append(100-l)


        return predTest,acc


def main():

        Parser = argparse.ArgumentParser()
        Parser.add_argument('--Method', default="Continuous", help='Continuous/Di
        # Parser.add_argument('--Generalization',default=35,help='Any int- 3 to 3
        Parser.add_argument('--Ratio',default=0.5,help='Only required for Continu

        Args = Parser.parse_args()
        Method = Args.Method
        # g = Args.Generalization

        theta = math.pi
        trainData = []
        testData = []
        trainLabel = []
        testLabel = []

        # 70:30 split for train and test data
        for u in range(70):
                trainData.append(u*theta/70)

        for u in range(30):
                testData.append(u*theta/30)

        for i in trainData:
            trainLabel.append(math.sin(i))

        for i in testData:
            testLabel.append(math.sin(i))

        # print(trainData)
        # print(trainLabel)

        for g in range(3,36,8):
```

```
                lr = 0.01

                if Method.lower() == "continuous":
                        print("Continuous CMAC"+"\n")
                        r=Args.Ratio
                        print("Ratio ="+str(r)+"\n")
                else:
                        print("Discrete CMAC"+"\n")
                        r=1

                print("Generalization ="+str(g)+"\n")

                epochLoss,numEpochs,pred,weight = train_cmac(trainLabel,trainData

                # print(max(epochLoss))
                fig = plt.figure()
                plt.title(Method+"CMAC Loss"+"-- g="+str(g)+" r="+str(r))
                plt.plot(numEpochs[1:100],epochLoss[1:100])
                plt.xlabel("Epochs")
                plt.ylabel("Loss")
                # plt.show()
                plt.savefig(Method+" Loss-- g="+str(g)+" r="+str(r)+".jpg", dpi=f
                plt.close()

                predTest,acc = test(testLabel,testData,weight,g,r)
                fig = plt.figure()
                plt.title(Method+" CMAC"+"-- g="+str(g)+" r="+str(r))

                plt.ylabel("Predicted Output")
                plt.step(testData,predTest,'g--')
                # plt.scatter(testData,predTest)
                plt.scatter(testData,testLabel)
                # plt.show()
                plt.savefig(Method+"-- g="+str(g)+" r="+str(r)+".jpg",dpi = fig.d
                plt.close()

                fig = plt.figure()
                plt.title(Method+"CMAC Accuracy"+"-- g="+str(g)+" r="+str(r))
                n = np.linspace(1,30,30)
                plt.plot(n,acc)
                plt.ylabel("Test Accuracy")
                # plt.show()
                plt.savefig(Method+" Accuracy-- g="+str(g)+" r="+str(r)+".jpg", d
                plt.close()

if __name__ == '__main__':
    main()
```