# HOMEWORK 2

# ENPM 690

# ROBOT LEARNING

Prof. Dr. Donald Sofge

UNIVERSITY OF
MARYLAND

**Kartik Venkat (116830751)**

This paper represents our own work in accordance with University regulations.

# Contents

# 1 Dataset Preparation

The following function was used to train the 1-D CMAC:

$$y = sin(x)$$

This dataset was created by taking 100 samples of the sin function from 0 to $2\pi$.

The Dataset was then split into test and train sets using the $train_test_split$ function form the sklearn library. The train set contains 70 datapoints and the test set contains 30 datapoints.

# 2 Discrete CMAC

## 2.1 Approach:

1. Generate Dataset

2. Train data till local convergence is found.

3. Test datapoints on the trained network.

4. Whole cells are used in weight updation.

5. Compute error

# 3 Continuous CMAC

## 3.1 Approach:

1. Generate Dataset

2. Train data till local convergence is found.

3. Test datapoints on the trained network.

4. sliding window is used in weight updation.

5. Compute error

## 4 Recurrent Networks

1. Recurrent Networks work on the principle of using two inputs, present and the past.

2. The idea is to store sequential memory like in humans.

3. LSTMs is one example of the use of Recurrent Networks that use gated inputs

# 5 Github Repository

This is the GitHub link: https://github.com/kartikv97/ENPM690/tree/master

## 5.1 README

ENPM 673 Perception for Autonomous Robots
@ Author Kartik Venkat,

**Instructions to run the code:**

1. Using Command Prompt:
   python ...PATH....py // use python3 if using Linux based OS
2. Using Spyder or any other IDE:
   Open the file and Run.

**Special Instructions:**

1. Install all package dependencies before running the code.
2. Update pip and all the packages to the latest versions.

## 5.2 CODE:

```python
# -*- coding: utf-8 -*-
"""
Created on Mon Feb 17 01:19:02 2020

@author: Kartik
"""
from sklearn.model_selection import train_test_split
import numpy as np
import math
import time
import random
import matplotlib.pyplot as plt

def Generate_Dataset(inputFunction):
    # 0- 360 degrees
    max_value=360
    min_value=0

    num_of_datapoints=100

    #Resolution = step size
    resolution_deg = float((max_value-min_value)/num_of_datapoints)
    #Resolution in Radians
    resolution_rad = float(resolution_deg*(np.pi/180))

    # convert degrees to radians
    input_dataset  = [resolution_rad * (i+1) for i in range(0,num_of_datapoints)]
    output_dataset = [inputFunction(input_dataset[i]) for i in range(0,num_of_datapoints)]
```

```

   #Split Dataset into training and testing sets. (70%training and 30%testing)
    input_train, input_test, output_train, output_test = train_test_split(input_dataset, output_dataset, test_size=0.3)
    train_global_indices= [input_dataset.index(i) for i in input_train]
    test_global_indices= [input_dataset.index(i) for i in input_test]

    return[input_dataset,output_dataset,input_train,input_test,output_train, output_test,train_global_indices,test_global_indices,resolution_rad]

    """
    print('res_deg',resolution_deg)
    print('res_rad',resolution_rad)

    print('x_train: ',input_train)
    print('x_test: ', input_test)
    print('y_train',output_train)
    print('y_test',output_test)


    print('input_dataset: ',input_dataset)
    print('train_global_indx:',train_global_indices)
    print('test_global_indx:',test_global_indices)

    """
# Initialize values
GeneralizationFactor= 5

neighbourhood_index = int(math.floor(GeneralizationFactor/2))

dataset = Generate_Dataset(np.sin)

input_dataset = dataset[0]
output_dataset = dataset[1]
input_dataset_size = len(input_dataset)

train_input_dataset = dataset[2]
train_output_dataset = dataset[4]
train_dataset_size = len(train_input_dataset)
train_global_indices = dataset[6]
training_CMAC_output = [0] #for i in range(0,train_dataset_size) ]

weights = [0 for i in range(0,input_dataset_size)]

test_input_dataset = dataset[3]
test_true_output_dataset = dataset[5]
test_dataset_size = len(test_input_dataset)
test_global_indices = dataset[7]
testing_CMAC_output = [0] #for i in range(0,test_dataset_size) ]

resolution_rad = dataset[8]

min_output_val = -1.0
max_output_val = 1.0
```

```

train_error = 1.0
test_error = 1.0

local_converge_threshold =0.01
learning_rate = 0.15
global_converge_threshold = 0.01
global_converge_iter = 20

convergence = False
convergence_time = 1000







def train():
    error = 1000

    for i in range (0, train_dataset_size):

        Local_Convergence = False
        # Locally store train data index values
        train_index = train_global_indices[i]
        error = 0
        iteration = 0
        # Generalization Factor offset
        offset_val = 0

        # Calculate offset for the top and bottom window cases
        if i - neighbourhood_index < 0:
            offset_val = i - neighbourhood_index
        if i + neighbourhood_index >= train_dataset_size:
            offset_val = train_dataset_size - (i + neighbourhood_index)

        # Run till Local convergence is achieved
        while Local_Convergence is False:
            cmac_output= 0
            for j in range (0, GeneralizationFactor):
                total_neighbourhood_index = train_index - (j - neighbourhood_index)

                if total_neighbourhood_index >=0 and total_neighbourhood_index < input_dataset_size :

                    weights[total_neighbourhood_index] = weights[total_neighbourhood_index] + (error/(GeneralizationFactor + offset_val))
                    cmac_output += input_dataset[total_neighbourhood_index]* weights[total_neighbourhood_index]

            error = train_output_dataset[i] - cmac_output
            iteration += 1

```

```python
            if iteration > 35:
                break
            if abs(MSE(train_output_dataset[i], cmac_output)) <= local_converge_threshold :
                Local_Convergence = True

def test(DataType, CmacType):

    cumulative_error = 0
    input_data = []
    if DataType is 'Train data' :
        input_data = test_input_dataset
        true_output = test_true_output_dataset
        test_indices = test_global_indices

    elif DataType is 'Test Data' :
        input_data = test_input_dataset
        true_output = test_true_output_dataset
        test_indices = test_global_indices

    cmac_output = [0 for i in range (0, len(input_data))]

    for i in range (0, len(input_data)):

        if DataType is 'Train Data' :
            index = test_indices[i]

        if DataType is 'Test Data' :
            index = find_nearest_key(input_dataset,input_data[i])

        error_index_val = float((input_dataset[index] - input_data[i])/resolution_rad)
        #If the actual value is lesser than nearest value, slide window to the left, partial overlap for first and last element
        if percentage_difference_in_value < 0 :
            max_offset = 0
            min_offset = -1
        #If the actual value is higher than the nearest value, slide window to the right, partial overlap for first and last element
        elif percentage_difference_in_value > 0 :
            max_offset = 1
            min_offset = 0

     #If its equal, then dont slide the window , all the elements must be completely overlapped
        else :
            max_offset = 0
            min_offset = 0

        for j in range(min_offset,GeneralizationFactor+max_offset):

            total_neighbourhood_index = train_index - (j - neighbourhood_index)

            if total_neighbourhood_index >=0 and total_neighbourhood_index < input_dataset_size :

                if j is min_offset :

                    if CmacType is 'Discrete':
                        weight = weights[total_neighbourhood_index]

                    if CmacType is 'Continuous' :
                        weight = weights[total_neighbourhood_index]*(1 - abs(error_index_val))

            elif j is GeneralizationFactor+max_offset :

                    if CmacType is 'Discrete':
                        weight = 0

                    if CmacType is 'Continuous' :
                        weight = weights[total_neighbourhood_index]* abs(error_index_val)
            else :
                weight = weights[total_neighbourhood_index]

            cmac_output[i] += input_dataset[total_neighbourhood_index]* weight

        error = true_output[i] - cmac_output[i]

        cumulative_error += abs(MSE(true_output[i],cmac_output[i]))

    return cmac_output, cumulative_error

def CMAC_Algorithm(CmacType):

    iterations = 0
    convergence_time = time.time()
    while iterations < global_converge_iter :

         train()

         training_CMAC_output,Training_Cumulative_Error = test('Train Data',CmacType)
         TrainError = Training_Cumulative_Error/train_dataset_size

         testing_CMAC_output,Testing_Cumulative_Error = test('Test Data', CmacType)
         TestError = Testing_Cumulative_Error/test_dataset_size

         iterations = iterations + 1

         if TestError <= global_converge_threshold :
             convergence = True
             break
    convergence_time = time.time() - convergence_time
    #plot()
    return TrainError, TestError

def MSE(a , b):
    mse = (a-b)* (a-b)
    return mse

```

```
def find_nearest_key(array,val):
    index_val = (np.abs(np.array(array)-val)).argmin()
    return index_val
def plot():

    sorted_train_input = [x for (y,x) in sorted(zip(train_global_indices,train_input_dataset))]
    sorted_train_output = [x for (y,x) in sorted(zip(train_global_indices,training_CMAC_output))]
    sorted_test_input =  [x for (y,x) in sorted(zip(test_global_indices,test_input_dataset))]
    sorted_test_output =  [x for (y,x) in sorted(zip(test_global_indices,testing_CMAC_output))]

    plt.subplot(221)
    plt.plot(train_input_dataset,train_output_dataset,’bo’,label=’True Output’)
    plt.plot(sorted_train_input,sorted_train_output,’ro’,label=’CMAC Output’)
    plt.title(’ Input Space Size = ’ + str(input_dataset_size) + ’\n Training data’ )
    plt.ylabel(’Output’)
    plt.xlabel(’Input ’)
    plt.legend(loc=’upper right’, shadow=True)
    plt.ylim((min_output_val,max_output_val))



################   MAIN  #################

TrainErrorContinuous,TestErrorContinuous= CMAC_Algorithm(’Continuous’)

TrainErrorDiscrete,TestErrorDiscrete= CMAC_Algorithm(’Discrete’)

print(’TrainErrorContinuous’,TrainErrorContinuous)
print(’TestErrorContinuous’,TestErrorContinuous)
print(’TrainErrorDiscrete’,TrainErrorDiscrete)
print(’TestErrorDiscrete’,TestErrorDiscrete)
```