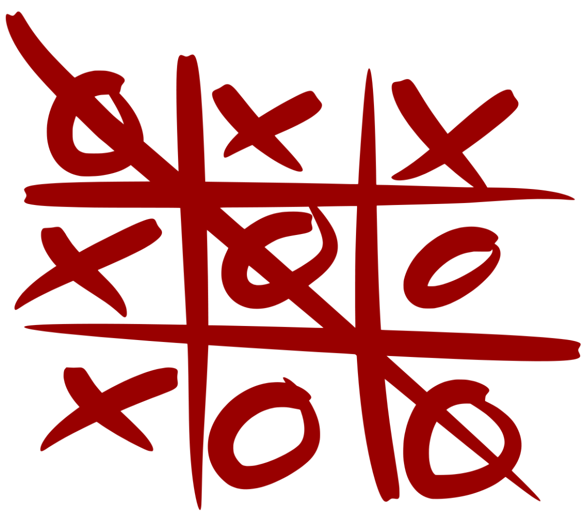
ML Lab Assignment 4

Implement a Machine Learning Program to Play 5x5 Tic Tac Toe Game Using the LMS Update Rule



REPORT BY : KAPIL RAVI RATHOD (177227)

LMS Weight Update Rule

- For each training example <b , Vtrain(b)>
  - Use the current weights to calculate $\hat{V}(b)$
  - For each weight wi, update it as

$$w_i \leftarrow w_i + \eta(V_{train(b)} - \hat{V}(b))x_i$$

The Implementation of the Tic Tac Toe Game Below is Based on the Above Rule.

REPORT BY : KAPIL RAVI RATHOD (177227)

177227 : ML Lab Assignment 4

TIC-TAC-TOE AI Learner Using LMS Update Rule

In [1]:
```
# Necessary Import Statements
import numpy as np
import pandas as pd
import copy
import math
import matplotlib.pyplot as plt
from tqdm import tqdm
```

Initialization of the Board

In [2]:
```
def initializeBoard(size = 4):
    '''
        Input Format:
            .> Size of Board
        Output Format:
            .> (size x size) matrix filled with '-' (empty cells)
    '''
    board = np.ones((size,size), dtype = str)
    board[:,:] = '-'
    return board
```

Determining the Winner of a Game

In [3]:
```
def determineWinner(board):
    '''
        A Function to Determine the Winner.
        Input Format:
            .> Board State
        Output Format:
            .> 100 if X wins
            .> (-100) if O wins
            .> 0 if draw
    '''
    # Determining the Number of Rows
    size = board.shape[0]


    ### For checking all Rows and Columns
    for i in range(size):
        if np.all(board[i,:] == 'X'): # 4 Xs in row i -> X
            return 100
        if np.all(board[i,:] == '0'): # 4 Os in row i -> O
            return -100
        if np.all(board[:,i] == 'X'): # 4 Xs in col i -> X
            return 100
        if np.all(board[:,i] == 'O'): # 4 Os in col i -> O
            return -100


    ### Checking the two diagonals - Set to false if other symbol encountered
    # leading => (top left to bottom right), trailing => [opposite]
    diagonalChecks = {
        'X_leading' : True, # leading diagonal has all Xs
        'O_leading' : True, # leading diagonal has all Os
        'X_trailing': True, # trailing has all Xs
        'O_trailing': True  # trailing has all Os
    }

    # In each iteration, if the value doesnt match, set to False
    for i in range(size):
        if board[i,i] != 'X':
            diagonalChecks['X_leading'] = False
        if board[i,i] != 'O':
            diagonalChecks['O_leading'] = False
        if board[i,(-i-1)] !=  'X':
            diagonalChecks['X_trailing'] = False
        if board[i,(-i-1)] !=  'O':
            diagonalChecks['O_trailing'] = False

    # check which one is remaining as true, if any
    if diagonalChecks['X_leading']:
        return 100
    if diagonalChecks['X_trailing']:
        return 100
    if diagonalChecks['O_leading']:
        return -100
    if diagonalChecks['O_trailing']:
        return -100
```

```
    ### Checking for a draw
    if np.sum(board == '-') == 0:
        return 0

    # Otherwise Game Not Over
    return -1
```

Initialization of Weights

In [4]:
```
weights = np.random.randn(33)/10
```

Feature Extraction

In [5]:
```
def extractFeatures(board):
    '''
        A Function to extract the (size*8 + 1) features from the Board
       Input Format:
           - Board
       Output:
           - Feature vector of length (size*8+1)
               * 1 is for the bias
               * for each n of {1 to size}, we have the following 8 features: ->
                   - number of rows with n Xs
                   - number of rows with n Os
                   - number of cols with n Xs
                   - number of cols with n Os
                   - if leading diagonal has n Xs
                   - if leading diagonal has n Os
                   - if trailing diagonal has n Xs
                   - if trailing diagonal has n Os
    '''

    # Get the size and create feature vector
    size = board.shape[0]
    feature_vector = np.zeros(size*8 + 1, dtype=np.float64)

    # Bias (First Element of the feature vector)
    feature_vector[0] = 1.0

    # calculate the number of Xs, Os, and -s in rows and columns
    num_x_row   = np.count_nonzero(board == 'X', axis = 1)
    num_x_col   = np.count_nonzero(board == 'X', axis = 0)
    num_o_row   = np.count_nonzero(board == 'O', axis = 1)
    num_o_col   = np.count_nonzero(board == 'O', axis = 0)
    num_emp_row = np.count_nonzero(board == '-', axis = 1)
    num_emp_col = np.count_nonzero(board == '-', axis = 0)

    # diagonal counters
    leading_X = 0       # Xs in leading
    leading_O = 0       # Os in leading
    leading_emp = 0     # -s in leading
    trailing_X = 0      # Xs in trailing
    trailing_O = 0      # Os in trailing
    trailing_emp = 0    # -s in trailing

    for i in range(size):
        if board[i,i] == 'X':
            leading_X += 1
        if board[i,i] == 'O':
            leading_O += 1
        if board[i,i] == '-':
            leading_emp += 1
        if board[i,-i-1] == 'X':
            trailing_X += 1
        if board[i,-i-1] == 'O':
            trailing_O += 1
        if board[i,-i-1] == '-':
            trailing_emp += 1
```

```
    # populate feature vector
    for i in range(1,size+1): # for each n of {1 to size}
        # for each row/col -> in first 4 of 8 positions
        for j in range(size):

            if num_x_row[j] == i and num_emp_row[j] == size-i:
                feature_vector[(i-1)*8 + 1] += 1
            if num_x_col[j] == i and num_emp_col[j] == size-i:
                feature_vector[(i-1)*8 + 2] += 1
            if num_o_row[j] == i and num_emp_row[j] == size-i:
                feature_vector[(i-1)*8 + 3] += 1
            if num_o_col[j] == i and num_emp_col[j] == size-i:
                feature_vector[(i-1)*8 + 4] += 1

        # diagonals -> in next 4 positions
        if leading_X == i and leading_emp == size-i :
            feature_vector[(i-1)*8 + 5]+=1
        if trailing_X == i and trailing_emp == size-i :
            feature_vector[(i-1)*8 + 6]+=1
        if leading_O == i and leading_emp == size-i :
            feature_vector[(i-1)*8 + 7]+=1
        if trailing_O == i and trailing_emp == size-i :
            feature_vector[(i-1)*8 + 8]+=1

    return feature_vector
```

Determining the Next Moves

In [6]:
```
def getPossibleStates(board, player):
    '''
        A Function to determine the next possible moves from the current.
        Input Format:
            - Board
            - Current Player
        Output Format:
            - Array of next possible board states
    '''
    boardStates = []

    size = board.shape[0]

    # go through each cell
    for i in range(size):
        for j in range(size):
            # if cell is empty, we can make a move here
            if board[i,j] == '-':
                # make a copy, set value, and add to the possible states
                temp_board = copy.deepcopy(board)
                temp_board[i,j] = player
                boardStates.append(temp_board)

    # return the next possible board_states
    return boardStates
```

Obtaining the Value of a Board

In [7]:
```
def calculateBoardValue(board):
    '''
        A Function to Calculate the Value of a given Board
        from the Learned Function
        Input Format:
            - Board
        Output Format:
            - Value of the Learned Function for the Board
    '''
    # Extract Features and Calculate Value = w.dot(featureVector.T)
    featureVector = extractFeatures(board)
    boardValue = np.dot(weights,featureVector.T)

    return boardValue
```

In [8]:
```
def calculateBoardValueMultiple(boards):
    '''
        A Function to Calculate the Value of a given board(s)
        from the Learned Function
        Input Format:
            - Boards -> Array of Boards
        Output Format:
            - Array of the Value of the Learned Lunction for the Boards.
    '''
    boardValues = []

    for board in boards:
        # Extract Features and Calculate Value = w.dot(featureVector.T)
        featureVector = extractFeatures(board)
        boardValue = np.dot(weights,featureVector.T)
        boardValues.append(boardValue)

    return boardValues
```

Printing the Board

In [9]:
```
def printBoard(board):
    '''
        A Function to Print the Board in Correct Format
        Input Format:
            - Board State
    '''
    size = board.shape[0]

    print("+",end='')
    for j in range(size):
        print('---', end = '+')
    for i in range(size):
        print("\n| ", end='')
        for j in range(size):
            print(board[i,j], end = ' | ')
        print("\n+",end='')
        for j in range(size):
            print('---', end = '+')
    print("")
```

A Function to Train the Computer

In [10]:
```
def train(size, alpha,epochs):
    '''
        A Function to Training the Computer.
        Inputs:
            - Size of Board
            - Alpha (Learning Rate)
            - epochs (Number of Iterations)
    '''

    for epoch in tqdm(range(epochs)):

        # Current Board History
        curr_board_history = []

        # Initialize Board and Current Token
        board = initializeBoard(size)
        current = 'X'

        # Loop till game isn't over
        while (determineWinner(board) == -1):

            # Obtain next possible states and calculate their values
            next_states = np.array(getPossibleStates(board, current))
            np.random.shuffle(next_states)
            next_values = calculateBoardValueMultiple(next_states)

            # Append board to history and set the next board state as the one wit
            curr_board_history.append(board)
            board = next_states[np.argmax(next_values)]

            # toggle move -> for training, we randomly toggle this
            # otherwise it always ends in a draw and weights never change
            rand_val = np.random.randn(1)
            if (rand_val > 0):
                current = 'X' if (current == 'O') else 'O'

        # Append final board state and determine winner
        curr_board_history.append(board)
        result = determineWinner(board)

        # Updating Weights
        global weights
        for idx, board_state in enumerate(curr_board_history):
            X = extractFeatures(board_state)
            if ((idx+2) < len(curr_board_history)):
                weights += alpha * (calculateBoardValue(curr_board_history[idx+2]
                              - calculateBoardValue(board_state))* X
            else:
                weights += alpha * (result - calculateBoardValue(board_state)) *
```

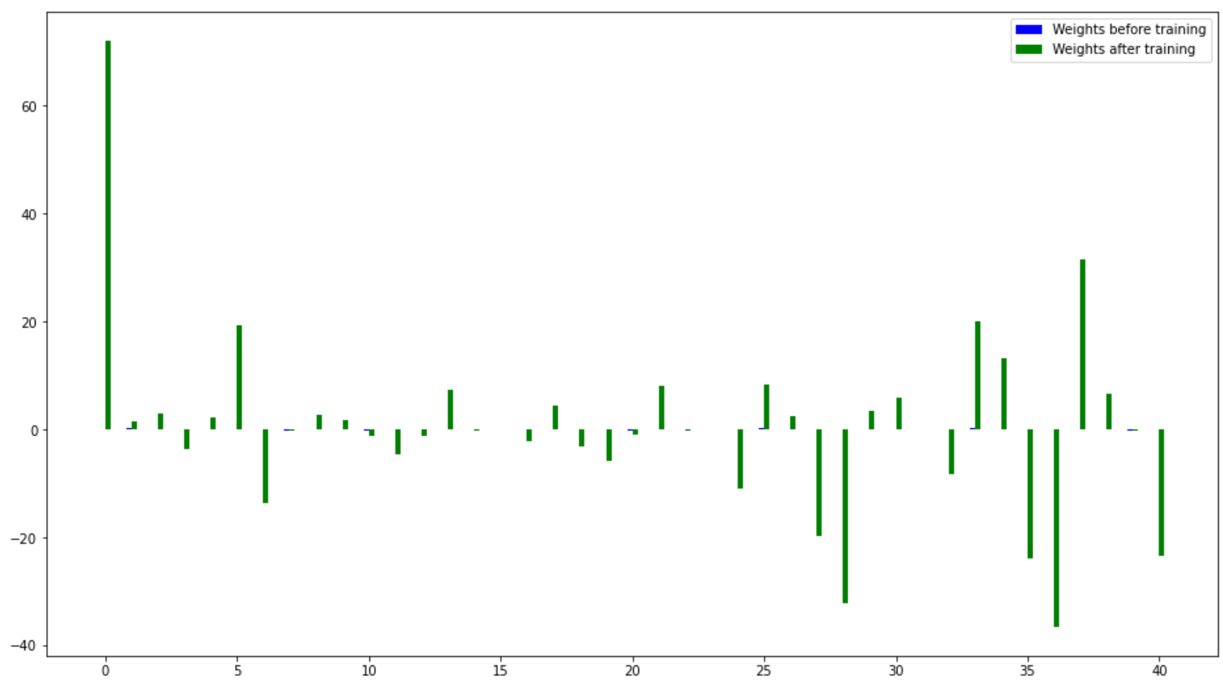
Hyperparameter Initialization

In [11]:
```
size = 5
alpha = 0.05
num_iters = 1000
num_features = size*8 + 1

# Randomly Initializing Weights
weights = np.random.randn(num_features) / 10
weights_copy = copy.deepcopy(weights)

# Invoke train function
train(size, alpha, num_iters)
```

```
100%|██████████| 1000/1000 [00:50<00:00, 19.94it/s]
```

Visualizing the Difference in Weights

In [12]:
```
plt.figure(figsize=(16,9))
x = np.arange(num_features)
ax = plt.subplot(111)
ax.bar(x-0.1, (weights_copy), width=0.2, color = 'blue', label='Weights before tr
ax.bar(x+0.1, (weights), width=0.2, color = 'green', label='Weights after trainin
ax.legend()
plt.show()
```



Test Function

In [13]:
```
def playGame(size):
    '''
        Allows human player to play against trained program
        After training, the program will play optimally, making it essentially in
    '''
    # initialise board
    board = initializeBoard(size)

    # signifies whose turn it currently is
    current = 'X'

    # loop while game is not over
    while (determineWinner(board) == -1):

        if current == 'X':
            print('Computer\'s move...')

            # Get next possible states, and calculate their values
            next_states = np.array(getPossibleStates(board, current))
            next_values = calculateBoardValueMultiple(next_states)

            # Set the next board state as the one with maximum value
            board = next_states[np.argmax(next_values)]

            # Printing the Board
            printBoard(board)

        else:
            print('Player\'s move')

            # get input move
            print('Enter x and y Coordinates (0 indexed, sperated by \'enter\'):
            while (True):
                try:
                    a = int(input())
                    b = int(input())

                    if (a == '-1' or b == '-1'):
                        return

                    if board[a,b] == '-':
                        break
                    else:
                        print("Please select an open position. Try again.")

                except:
                    print("Invalid input! Try again.")

            # mark an O and print the board
            board[a,b] = 'O'
            printBoard(board)

        # toggle move
        current = 'X' if (current == 'O') else 'O'

    # determine winner and print output
```

```
    result = determineWinner(board)
    print("\n-------Result:--------\n")
    if result == 100:
        print('Computer Wins!')
    elif result == -100:
        print('Player Wins!')
    else:
        print('Draw!')
```

In [14]:
```
# call the play function to test our program
playGame(size)
```

```
Computer's move...
+---+---+---+---+---+
| X | - | - | - | - |
+---+---+---+---+---+
| - | - | - | - | - |
+---+---+---+---+---+
| - | - | - | - | - |
+---+---+---+---+---+
| - | - | - | - | - |
+---+---+---+---+---+
| - | - | - | - | - |
+---+---+---+---+---+
Player's move
Enter x and y Coordinates (0 indexed, sperated by 'enter'): -1 to quit
2
2
+---+---+---+---+---+
| X | - | - | - | - |
+---+---+---+---+---+
| - | - | - | - | - |
+---+---+---+---+---+
| - | - | O | - | - |
+---+---+---+---+---+
| - | - | - | - | - |
+---+---+---+---+---+
| - | - | - | - | - |
+---+---+---+---+---+
Computer's move...
+---+---+---+---+---+
| X | - | - | - | - |
+---+---+---+---+---+
| - | - | - | - | - |
+---+---+---+---+---+
| - | X | O | - | - |
+---+---+---+---+---+
| - | - | - | - | - |
+---+---+---+---+---+
| - | - | - | - | - |
+---+---+---+---+---+
Player's move
Enter x and y Coordinates (0 indexed, sperated by 'enter'): -1 to quit
1
1
+---+---+---+---+---+
| X | - | - | - | - |
+---+---+---+---+---+
| - | O | - | - | - |
+---+---+---+---+---+
| - | X | O | - | - |
+---+---+---+---+---+
| - | - | - | - | - |
+---+---+---+---+---+
| - | - | - | - | - |
```

```
+---+---+---+---+---+
Computer's move...
+---+---+---+---+---+
| X | - | - | - | - |
+---+---+---+---+---+
| - | O | - | - | X |
+---+---+---+---+---+
| - | X | O | - | - |
+---+---+---+---+---+
| - | - | - | - | - |
+---+---+---+---+---+
| - | - | - | - | - |
+---+---+---+---+---+
Player's move
Enter x and y Coordinates (0 indexed, sperated by 'enter'): -1 to quit
1
0
+---+---+---+---+---+
| X | - | - | - | - |
+---+---+---+---+---+
| O | O | - | - | X |
+---+---+---+---+---+
| - | X | O | - | - |
+---+---+---+---+---+
| - | - | - | - | - |
+---+---+---+---+---+
| - | - | - | - | - |
+---+---+---+---+---+
Computer's move...
+---+---+---+---+---+
| X | - | - | - | - |
+---+---+---+---+---+
| O | O | - | - | X |
+---+---+---+---+---+
| - | X | O | - | - |
+---+---+---+---+---+
| - | - | - | X | - |
+---+---+---+---+---+
| - | - | - | - | - |
+---+---+---+---+---+
Player's move
Enter x and y Coordinates (0 indexed, sperated by 'enter'): -1 to quit
0
3
+---+---+---+---+---+
| X | - | - | O | - |
+---+---+---+---+---+
| O | O | - | - | X |
+---+---+---+---+---+
| - | X | O | - | - |
+---+---+---+---+---+
| - | - | - | X | - |
+---+---+---+---+---+
| - | - | - | - | - |
+---+---+---+---+---+
Computer's move...
+---+---+---+---+---+
```

```
| X | - | - | O | - |
+---+---+---+---+---+
| O | O | - | - | X |
+---+---+---+---+---+
| - | X | O | - | - |
+---+---+---+---+---+
| - | - | - | X | - |
+---+---+---+---+---+
| - | X | - | - | - |
+---+---+---+---+---+
Player's move
Enter x and y Coordinates (0 indexed, sperated by 'enter'): -1 to quit
0
4
+---+---+---+---+---+
| X | - | - | O | O |
+---+---+---+---+---+
| O | O | - | - | X |
+---+---+---+---+---+
| - | X | O | - | - |
+---+---+---+---+---+
| - | - | - | X | - |
+---+---+---+---+---+
| - | X | - | - | - |
+---+---+---+---+---+
Computer's move...
+---+---+---+---+---+
| X | - | - | O | O |
+---+---+---+---+---+
| O | O | - | - | X |
+---+---+---+---+---+
| - | X | O | - | - |
+---+---+---+---+---+
| - | X | - | X | - |
+---+---+---+---+---+
| - | X | - | - | - |
+---+---+---+---+---+
Player's move
Enter x and y Coordinates (0 indexed, sperated by 'enter'): -1 to quit
1
3
+---+---+---+---+---+
| X | - | - | O | O |
+---+---+---+---+---+
| O | O | - | O | X |
+---+---+---+---+---+
| - | X | O | - | - |
+---+---+---+---+---+
| - | X | - | X | - |
+---+---+---+---+---+
| - | X | - | - | - |
+---+---+---+---+---+
Computer's move...
+---+---+---+---+---+
| X | - | - | O | O |
+---+---+---+---+---+
| O | O | - | O | X |
```

```
+---+---+---+---+---+
| - | X | O | - | - |
+---+---+---+---+---+
| X | X | - | X | - |
+---+---+---+---+---+
| - | X | - | - | - |
+---+---+---+---+---+
Player's move
Enter x and y Coordinates (0 indexed, sperated by 'enter'): -1 to quit
0
1
+---+---+---+---+---+
| X | O | - | O | O |
+---+---+---+---+---+
| O | O | - | O | X |
+---+---+---+---+---+
| - | X | O | - | - |
+---+---+---+---+---+
| X | X | - | X | - |
+---+---+---+---+---+
| - | X | - | - | - |
+---+---+---+---+---+
Computer's move...
+---+---+---+---+---+
| X | O | - | O | O |
+---+---+---+---+---+
| O | O | - | O | X |
+---+---+---+---+---+
| - | X | O | - | - |
+---+---+---+---+---+
| X | X | - | X | X |
+---+---+---+---+---+
| - | X | - | - | - |
+---+---+---+---+---+
Player's move
Enter x and y Coordinates (0 indexed, sperated by 'enter'): -1 to quit
4
2
+---+---+---+---+---+
| X | O | - | O | O |
+---+---+---+---+---+
| O | O | - | O | X |
+---+---+---+---+---+
| - | X | O | - | - |
+---+---+---+---+---+
| X | X | - | X | X |
+---+---+---+---+---+
| - | X | O | - | - |
+---+---+---+---+---+
Computer's move...
+---+---+---+---+---+
| X | O | - | O | O |
+---+---+---+---+---+
| O | O | - | O | X |
+---+---+---+---+---+
| - | X | O | - | - |
+---+---+---+---+---+
```

```
| X | X | X | X | X |
+---+---+---+---+---+
| - | X | O | - | - |
+---+---+---+---+---+

-------Result:--------

Computer Wins!
```