# Tic-Tac-Toe Game with Hebbian Learning: Documentation

**1. Overview**

This Tic-Tac-Toe game uses Hebbian learning to predict the player's turn and suggest the best possible move based on the current board state. The game also displays the winner and resets the board after a win.

**2. Dependencies**

* **NumPy**: Used for numerical operations like matrix manipulation and dot products.
* **Tkinter**: Used for creating the graphical user interface (GUI) and handling user interactions.

**3. Code Description**

**3.1. Hebbian Learning**

**hebbian\_train(inputs, targets)**

This function implements the Hebbian learning rule, which adjusts the weights between input and target vectors in a neural network. The learning rule states that the connection between two neurons strengthens when they are activated together.

* **Inputs**:
  + inputs: A list of input vectors, where each vector represents the current state of the Tic-Tac-Toe board (9 elements).
  + targets: A list of target vectors, where each vector represents the expected output (e.g., the predicted player's turn or best move).
* **Output**:
  + A weight matrix updated using the Hebbian learning rule.

**Hebbian Rule**:

W=W+X⋅Y

Where:

* W is the weight matrix.
* X is the input vector (reshaped as a column vector).
* Y is the target vector (reshaped as a row vector).

**Code**:

python

CopyEdit

def hebbian\_train(inputs, targets):

weights = np.zeros((len(inputs[0]), len(targets[0]))) # Initialize weight matrix

for x, y in zip(inputs, targets): # Loop through inputs and targets

x = np.array(x).reshape(-1, 1) # Reshape input into column vector

y = np.array(y).reshape(1, -1) # Reshape target into row vector

weights += x @ y # Update weights using the outer product

return weights # Return the learned weight matrix

**predict(input\_vector, weights)**

This function predicts the output (turn or move) based on the current board state and the weight matrix. It calculates the dot product between the input vector and the weight matrix and returns the index of the highest value.

* **Inputs**:
  + input\_vector: The current state of the Tic-Tac-Toe board, represented as a list of 9 elements.
  + weights: The weight matrix learned through Hebbian training.
* **Output**:
  + The index of the predicted move.

**Code**:

python

CopyEdit

def predict(input\_vector, weights):

result = np.dot(input\_vector, weights) # Calculate the dot product

return np.argmax(result) # Return the index of the maximum value in the result

**3.2. Game Logic**

**check\_winner(board)**

This function checks if a player has won by evaluating all possible winning combinations (rows, columns, and diagonals) on the Tic-Tac-Toe board.

* **Input**:
  + board: A list of 9 elements representing the Tic-Tac-Toe board. Each element can be 1 (X), -1 (O), or 0 (empty).
* **Output**:
  + 1 if X wins, -1 if O wins, or 0 if no winner is found.

**Code**:

python

CopyEdit

def check\_winner(board):

winning\_combinations = [

[0, 1, 2], [3, 4, 5], [6, 7, 8], # Rows

[0, 3, 6], [1, 4, 7], [2, 5, 8], # Columns

[0, 4, 8], [2, 4, 6] # Diagonals

]

for combination in winning\_combinations:

if board[combination[0]] == board[combination[1]] == board[combination[2]] and board[combination[0]] != 0:

return board[combination[0]] # Return the winner (1 for X, -1 for O)

return 0 # No winner yet

**3.3. GUI Design**

* **Tkinter** is used to create the graphical user interface (GUI) for the Tic-Tac-Toe game. It consists of 9 buttons representing the 9 cells of the Tic-Tac-Toe board.
* The buttons are placed in a 3x3 grid, and each button's text is updated based on the current state of the board (X, O, or empty).
* The GUI also includes:
  + A label to display the predicted player's turn (Predicted Turn: X or O).
  + A reset button to restart the game.

**Code** for GUI:

python

CopyEdit

root = tk.Tk()

root.title("Hebbian Tic-Tac-Toe Predictor")

buttons = []

for i in range(9):

btn = tk.Button(root, text="", width=6, height=3, font=("Arial", 20), command=lambda i=i: click\_cell(i))

btn.grid(row=i//3, column=i%3)

buttons.append(btn)

turn\_label = tk.Label(root, text="Predicted Turn: ?", font=("Arial", 14))

turn\_label.grid(row=3, column=0, columnspan=3)

reset\_btn = tk.Button(root, text="Reset", command=reset\_board)

reset\_btn.grid(row=4, column=0, columnspan=3)

update\_gui() # Update the GUI for the first time

root.mainloop()

**4. Game Flow**

1. **Start Game**: The game board is initialized with empty cells (0).
2. **Player's Turn**: Each player's turn is displayed on the GUI. The system predicts the next turn using Hebbian learning.
3. **Predicted Move**: The system suggests the best possible move based on the current board state and previous training.
4. **Check for Winner**: After each move, the game checks if there is a winner using the check\_winner() function.
5. **Display Winner**: If a player wins, a popup displays the winner, and the game resets after clicking "OK."
6. **Reset Game**: The "Reset" button allows players to start a new game.

**5. How Hebbian Learning Works in the Game**

1. **Training**:
   * During training, the system learns from different board states and the corresponding player turns or best moves.
   * The weight matrix is updated using the Hebbian learning rule, strengthening connections between board configurations and the corresponding outcomes.
2. **Prediction**:
   * After training, when the game is played, the system uses the learned weights to predict the next turn and suggest the best possible move.
3. **Implementation**:
   * The Hebbian learning rule adjusts the weight matrix such that the output (turn or move) aligns with the training examples.

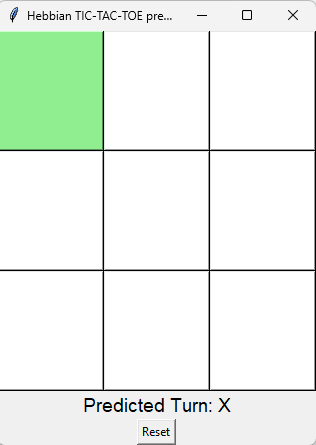
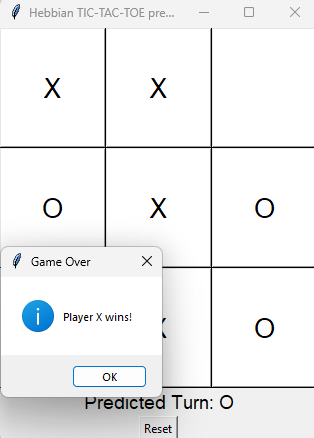
**6. How to Use the Code**

1. **Run the Python Script**:
   * Ensure that you have Python and the required libraries (numpy, tkinter) installed.
   * Run the script, and the Tic-Tac-Toe game window will open.
2. **Play the Game**:
   * Players take turns clicking on the cells to place their marks (X or O).
   * The system predicts the player's turn and suggests the best possible move.
   * The game checks for a winner after every move.
3. **Reset the Game**:
   * Click the "Reset" button to start a new game.

**7. Conclusion**

This Tic-Tac-Toe game demonstrates the application of Hebbian learning in predicting the next move and player's turn. The combination of machine learning and interactive GUI provides an enjoyable game experience. The game also offers insights into how Hebbian learning can be used to model decision-making in a simple board game.

**Index**

****

**References**

* **Hebbian Learning**: <https://en.wikipedia.org/wiki/Hebbian_learning>
* **NumPy Documentation**: https://numpy.org/doc/stable/
* **Tkinter Documentation**: <https://docs.python.org/3/library/tkinter.html>