Day-3 Experiments

13 Write the python program to implement Minimax algorithm for gaming.

Program:

```
def print_board(board):
    for row in board:
        print(" | ".join(row))
    print()
def check_winner(board, player):
    # Check rows, columns, and diagonals for a win
    for i in range(3):
        if all(board[i][j] == player for j in range(3)) or all(board[j][i] == player for j in range(3)):
            return True
    if all(board[i][i] == player for i in range(3)) or all(board[i][2 - i] == player for i in
range(3)):
        return True
    return False


# Check if the board is full
def is_full(board):
    return all(board[i][j] != ' ' for i in range(3) for j in range(3))
def minimax(board, is_maximizing):
    if check_winner(board, 'X'):
        return 1
    if check_winner(board, 'O'):
        return -1
    if is_full(board):
        return 0
    if is_maximizing:
        best_score = -float('inf')
        for i in range(3):
            for j in range(3):
```

```
            if board[i][j] == ' ':
                board[i][j] = 'X'
                score = minimax(board, False)
                board[i][j] = ' '
                best_score = max(best_score, score)
        return best_score
    else:
        best_score = float('inf')
        for i in range(3):
            for j in range(3):
                if board[i][j] == ' ':
                    board[i][j] = 'O'
                    score = minimax(board, True)
                    board[i][j] = ' '
                    best_score = min(best_score, score)
        return best_score
def find_best_move(board):
    best_score = -float('inf')
    move = (-1, -1)
    for i in range(3):
        for j in range(3):
            if board[i][j] == ' ':
                board[i][j] = 'X'
                score = minimax(board, False)
                board[i][j] = ' '
                if score > best_score:
                    best_score = score
                    move = (i, j)
    return move
def main():
```

```
board = [[' ' for _ in range(3)] for _ in range(3)]  # 3x3 empty board
print("Welcome to Tic-Tac-Toe!")
print("You are 'O' and the computer is 'X'.")
print_board(board)

while True:
    print("Your turn! Enter row and column (0, 1, or 2) separated by a space:")
    row, col = map(int, input().split())
    if board[row][col] != ' ':
        print("Invalid move! Try again.")
        continue
    board[row][col] = 'O'
    print_board(board)

    if check_winner(board, 'O'):
        print("You win!")
        break
    if is_full(board):
        print("It's a tie!")
        break
    print("Computer's turn:")
    row, col = find_best_move(board)
    board[row][col] = 'X'
    print_board(board)

    if check_winner(board, 'X'):
        print("Computer wins!")
        break
    if is_full(board):
        print("It's a tie!")
```

```
            break

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

Sample output:

```
============== RESTART: C:/Users/Admin/Desktop/AIProject/minmax.py ==============
Welcome to Tic-Tac-Toe!
You are 'O' and the computer is 'X'.
  |   |
  |   |
  |   |

Your turn! Enter row and column (0, 1, or 2) separated by a space:
0 0
O |   |
  |   |
  |   |

Computer's turn:
O |   |
  | X |
  |   |

Your turn! Enter row and column (0, 1, or 2) separated by a space:
1 0
O |   |
O | X |
  |   |

Computer's turn:
O |   |
O | X |
X |   |

Your turn! Enter row and column (0, 1, or 2) separated by a space:
2 0
Invalid move! Try again.
Your turn! Enter row and column (0, 1, or 2) separated by a space:
```

14 Write the python program to implement Apha & Beta pruning algorithm for gaming.

Program:

```
def alpha_beta(depth, index, is_max, values, alpha, beta):
    if depth == 3:
        return values[index]
    if is_max:
        max_eval = -float('inf')
        for i in range(2):
```

```
            eval = alpha_beta(depth + 1, index * 2 + i, False, values, alpha, beta)
            max_eval = max(max_eval, eval)
            alpha = max(alpha, eval)
            if beta <= alpha:
                break
        return max_eval
    else:
        min_eval = float('inf')
        for i in range(2):
            eval = alpha_beta(depth + 1, index * 2 + i, True, values, alpha, beta)
            min_eval = min(min_eval, eval)
            beta = min(beta, eval)
            if beta <= alpha:
                break
        return min_eval
values = [3, 5, 6, 9, 1, 2, 0, -1]
optimal_value = alpha_beta(0, 0, True, values, -float('inf'), float('inf'))
print("The optimal value is:", optimal_value)
```

15 Write the python program to implement Decision Tree.

Program:

```
from sklearn.tree import DecisionTreeClassifier
import pandas as pd
data = {
    'Age': ['<=30', '<=30', '31-40', '>40', '>40', '>40', '31-40', '<=30', '<=30', '>40', '<=30', '31-40', '31-40', '>40'],
    'Income': ['High', 'High', 'High', 'Medium', 'Low', 'Low', 'Low', 'Medium', 'Low', 'Medium', 'Medium', 'Medium', 'High', 'Medium'],
    'Student': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'Yes'],
    'Credit_Rating': ['Fair', 'Excellent', 'Fair', 'Fair', 'Fair', 'Excellent', 'Excellent', 'Fair', 'Fair', 'Fair', 'Excellent', 'Excellent', 'Fair', 'Excellent'],
    'Buys_Computer': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']
```

```
}
df = pd.DataFrame(data)
df = df.apply(lambda col: col.astype('category').cat.codes)
X = df[['Age', 'Income', 'Student', 'Credit_Rating']]
y = df['Buys_Computer']
model = DecisionTreeClassifier(criterion='entropy', random_state=0)
model.fit(X, y)
new_customer = pd.DataFrame([[0, 1, 1, 0]], columns=['Age', 'Income', 'Student',
'Credit_Rating'])
prediction = model.predict(new_customer)
print("Prediction for new customer (Buys_Computer):", 'Yes' if prediction[0] == 1 else 'No')
```

Sample output:

```
========== RESTART: C:/Users/Admin/Desktop/AIProject/decision tree.py ==
Prediction for new customer (Buys_Computer): Yes
```

16 Write the python program to implement Feed forward neural Network.

Program:

```
import numpy as np
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
def sigmoid_derivative(x):
    return x * (1 - x)
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [1], [1], [0]])
weights_input_hidden = np.random.rand(2, 2)
weights_hidden_output = np.random.rand(2, 1)
for epoch in range(10000):
    hidden_input = np.dot(X, weights_input_hidden)
    hidden_output = sigmoid(hidden_input)
    final_input = np.dot(hidden_output, weights_hidden_output)
    predicted_output = sigmoid(final_input)
    error = y - predicted_output
```

```
    d_predicted_output = error * sigmoid_derivative(predicted_output)
    error_hidden_layer = d_predicted_output.dot(weights_hidden_output.T)
    d_hidden_layer = error_hidden_layer * sigmoid_derivative(hidden_output)
    weights_hidden_output += hidden_output.T.dot(d_predicted_output) * 0.1
    weights_input_hidden += X.T.dot(d_hidden_layer) * 0.1
print("Final Prediction:")
print(predicted_output)
```

Sample output:

```
Type  help ,  copyright ,  credits  or  license()  for mo
>
============ RESTART: C:/Users/Admin/Desktop/AIProject/fw
Final Prediction:
[[0.22260328]
 [0.71998204]
 [0.71993327]
 [0.36369481]]
>
```

17 Write a Prolog Program to Sum the Integers from 1 to n.

Program:

```
sum(0,0).
sum(N,Sum):- N>0,N1 is N-1,sum(N1,S),Sum is N+S.
```

18 Write a Prolog Program for A DB WITH NAME, DOB.

Program:

```
% Facts representing the name and date of birth (DOB)
born(jan, 20, 3, 1977).
born(jeroen, 2, 2, 1992).
born(joris, 17, 3, 1995).
born(jelle, 1, 1, 2004).
born(jesus, 24, 12, 2000).
born(joop, 30, 4, 1989).
born(jannecke, 17, 3, 1993).
born(jaap, 16, 11, 1995).
```